



Application web de prise de commande à distance pour fluidifier le service d'un restaurant interne.

Résumé

Dans ce dossier sont présentées les différentes étapes de conception et de réalisation d'une application web pour le groupe Société Générale. Cette application a pour but de permettre aux collaborateurs (utilisant les services de restauration interne) de commander leur repas de leur smartphone. Ainsi, elle implémente les processus de suivi de commande classique, la gestion et la génération des bons de commande mais aussi la gestion des états pour que les agents des points de vente puissent connaître rapidement le statut d'une commande. Enfin, l'application génère des graphiques statistiques sur l'état des ventes à une date donnée. Cette application sera disponible dans un premier temps à un ensemble de 500 bêta testeurs. Pour conclure, l'objectif est de développer les commandes à distance afin de gagner en productivité, de gérer une plus grande quantité de commandes et ce, de manière plus fluide par l'utilisation d'un outil moderne, personnalisé et évolutif.

Abstract

In this file we introduced the different stages in the creation and designing of a web application for the Société Générale company. This app aims at enabling our collaborators (who use our on-site catering services) to order their meal with their smartphone. This app will therefore implement the follow-up basic order process, the supervision and issuing of the purchase orders as well as the management of the order status so that our point of sales agents may quickly be informed of the order status. Finally, this app may generate statistical graphs showing the sales status at any given time. This app will be available to five hundred beta testers to start with. As a conclusion, our goal is to expand the concept of distance orders so as to gain in productivity, generate a larger amount of orders following a smoother process based on this customized, modern and upgradeable tool.

Remerciement

Je souhaite avant tout remercier mon formateur Frank Marschall pour le temps qu'il a consacré à m'apporter les outils méthodologiques indispensables à la conduite de ce projet. Je remercie également Michael Guedj de m'avoir intégrée dans ses équipes au sein de FRM/CCR chez Societe Generale, de m'avoir fait confiance pour mener à bien de nombreux projets. Un grand merci également à toute l'équipe de l'ItSchool de Simplon et plus particulièrement à Priscilla Galbas de m'avoir contacté pour participer à cette aventure merveilleuse. Enfin, un grand merci à Yolande qui m'a apporté une aide précieuse en acceptant de relire mon mémoire.

CHAPITRE 1: Expression des besoins

1 Problématique	7
2 Objectifs	7
3 Contraintes	7
4 Le contexte	8
5 Les besoins	8
5.1 La méthode des personas	9
5.2 Les personas du projet	9
5.2.1 Les clients	9
5.2.2 Les agents du points de vente	10
5.2.3 Les manageurs du point de vente	11

CHAPITRE 2: Spécifications fonctionnelles

1 Les acteurs du projet	14
2 Les Users stories	14
2.1 Authentification	15
2.2 Prise de commande	15
2.3 Gestion des commandes	15
2.4 Tableau de bord	15
3 Réalisation des maquettes	15
3.1 L'écran du menu et le panier	16
3.2 L'écran de validation de la commande	17
3.3 L'écran de gestion des commandes	18
3.4 L'écran d'affichage du tableau de bord	19
4 Les issues	20

CHAPITRE 3: Spécifications techniques

1 Diagrammes de cas d'utilisations	23
1.1 Définition	23
1.2 Le diagramme de cas d'utilisation de la prise de commande	24
1.3 Le diagramme des cas d'utilisation de la gestion des commandes	24
1.4 Le diagramme de cas d'utilisation de la gestion du compte personnel	25
2 Diagrammes de séquences	26
2.1 Définition	26
2.2 Diagramme de séquence : enregistrement du panier d'achat	27
2.3 Diagramme de séquence : confirmation du panier d'achat	28
2.4 Diagramme de séquence : gestion des commandes	29
3 Diagramme de classes	29
4 Notions de base des protocoles de communication sur le Web	31
5 L'architecture	33
5.1 Généralités	33
5.2 L'architecture client-serveur	34
5.3 L'architecture 3-tier	35
5.4 L'architecture multi-tier	36
5.5 Le diagramme d'architecture du projet	37
6 Le langage de programmation	38

6.1 Les différents langages de programmation	38
6.2 Comparaison	40
6.2.1 Les spécifications du langage PHP	40
6.2.2 Les spécifications du langage JavaScript	40
6.2.3 Les spécifications du langage Python	41
6.2.4 Les spécifications du langage Java	42
6.2.5 Le langage retenu	43
7 Les frameworks	43
7.1 Les frameworks backend	43
7.1.1 Spring Boot	44
7.1.2 Hibernate	44
7.1.3 Spring Data Jpa	45
7.1.4 Spring Security	45
7.2 Les frameworks front-end	45
7.2.1 React	46
7.2.2 Vue	46
7.2.3 Angular	47
7.2.4 Framework front-end utilisé	48
8 Les systèmes de gestion de bases de données (SGBD)	49
8.1 Les types de base de données	49
8.1.1 Le modèle relationnel	49
8.1.2 Le modèle NoSQL	50
8.2 Les SGBD les plus populaires	51
8.2.1 Oracle Database	51
8.2.2 MySQL	52
8.2.3 Microsoft SQL Server	52
8.2.4 MongoDB	52
8.2.5 PostgreSQL	53
8.3 Le SGBD retenu	53
9 Modélisation de la base de données	53
10 Sécurité	58
11 WebSocket	59

Chapitre 4 : Réalisation

1 Arborescence du projet	53
2 Prise de commande	66
3 Gestion des commandes	71
4 Notifications et WebSocket	73
5 Tableau de bord	75
6 Tests	
Conclusion	83

CHAPITRE 1:

Expression des besoins

1 Problématique

Les restaurants internes du siège situés à la Défense accueillent chaque jour environ 20000 collaborateurs sur une plage horaire très réduite, ce qui génère une forte affluence et un temps d'attente important. (Cf. 4 pour plus de précisions) De plus, avec la crise sanitaire actuelle, les cantines internes doivent maintenant respecter le protocole sanitaire, notamment la distanciation sociale. C'est complexe logistiquement étant donné le nombre de personnes qui se présentent sur une plage horaire restreinte.

2 Objectifs

L'objectif de notre projet est de proposer un système de gestion des commandes à destination d'un futur restaurant d'entreprise permettant d'éliminer ces phénomènes de forte affluence et de temps d'attente important. Ce projet pourra dans un second temps s'étendre à tous les restaurants internes de la Société Générale. L'application *Ceraon* propose une solution concrète et efficace à la problématique exposée ci-dessus. (cf. chap.1.1) Elle permet d'une part aux utilisateurs "clients" de consulter le menu depuis leur smartphone, de passer commande à distance et de recevoir un message spécifiant que sa commande est prête. Elle permet d'autre part aux utilisateurs "managers" / "agents" une plus grande efficacité par la possibilité de visualiser les commandes en cours, (pour le manager) de visualiser les statistiques de vente du jour ainsi que l'historique des ventes quelque soit la date.

3 Contraintes

Notre première contrainte est le temps imparti. En effet, ce projet passionnant est réalisé pendant notre temps libre. Nous avons veillé à faire des choix pertinents pour respecter les délais de livraison prévus. Nous avons donc pris en considération le temps de formation conséquent de certains outils dans les choix de développement.

La seconde contrainte est celle du coût. En effet, l'entreprise ne dispose d'aucun budget pour la réalisation de cette application. Nous avons donc favorisé les outils gratuits pour les choix techniques.

La troisième contrainte que nous avons respecté est la réalisation d'une application web. Nous avons fait ce choix afin de faciliter la conception de l'application et d'éviter l'installation de l'outil sur tous les postes. De plus, l'application web, du fait qu'elle soit accessible à partir d'un navigateur web, offre une grande liberté d'action: Une fois installée sur un serveur, celle-ci peut être maintenue facilement et ne nécessite pas d'être redéployée à chaque mise à jour.

Ce choix s'inscrit dans une démarche écologique axée sur le respect de l'environnement.

Enfin la dernière contrainte est la volumétrie. En effet, les collaborateurs du site étant très nombreux, l'application doit pouvoir supporter de nombreuses connexions en même temps.

4 Le contexte

Les agents des points de vente (de restauration interne) utilisent uniquement des caisses enregistreuses. Ils doivent y entrer la commande du client. Les tarifs sont différents selon le statut du collaborateur, ainsi les internes ont une réduction de 2,40 euros sur leur repas et les externes, une majoration de 4,90 euros. Ils prennent compte du menu du jour à leur arrivée dans le point de vente. Ils font ensuite leur choix et passent aux caisses pour régler la commande à l'aide de leur badge personnel. Par ailleurs, le badge peut-être rechargé via l'application CollabPay.

Le système actuel fonctionne bien mais peut être perfectionné. On peut observer qu'à la pause méridienne, les collaborateurs en nombre très important s'agglutinent devant les menus à l'entrée, puis attendent longtemps aux stands pour récupérer le repas choisi et enfin passent aux caisses. Afin d'enrayer le double problème d'affluence et de temps d'attente, nous souhaitons favoriser la commande à distance via un smartphone ou un ordinateur. Ainsi, l'application permettra la consultation des plats du jour à distance, la commande à distance et l'envoi d'un message au client lui spécifiant que sa commande est prête. Ces trois solutions concrètes permettront de résoudre ces phénomènes de forte affluence et de temps d'attente important.

5 Les besoins

L'application *Ceraon* répond à des besoins multiples. Tout d'abord, elle offre des services à une clientèle diversifiée (personnel de la Société générale). En effet, le site de La Défense compte de nombreux collaborateurs de nationalités diverses, le groupe étant implémenté dans 57 pays. C'est pourquoi l'application est disponible en anglais pour faciliter son utilisation par tous. La possibilité de proposer plusieurs langues pourra être envisagée dans un second temps .

Si *Ceraon* est utilisée par les clients, elle l'est également par ceux qui travaillent dans ce point de vente (managers / agents). Elle a donc pour vocation de répondre à leurs besoins fonctionnels pour les assister.

Pour définir les utilisateurs de l'application *Ceraon* et leurs besoins, nous avons utilisé la méthode des personas. Les besoins des "clients et des "managers / agents" seront donc détaillés dans la sous-partie "les personas du projet" (Cf. chap.2.2.b)

5.1 La méthode des personas

Le terme "persona" vient du latin et signifie "masque de théâtre". Les Personas sont "un archétype d'utilisateur, à qui l'on a donné un nom et un visage, et qui est décrit avec attention, en termes de besoins, de buts et de tâches" (Brangier et al., 2012). Ils permettent de construire une vision partagée de l'utilisateur.

Les personas sont utilisés pour faciliter la manipulation d'entités complexes : les utilisateurs et leurs besoins. Ainsi, cette méthode me permettra de bien cerner chaque entité-utilisateur. En effet, les personas sont une représentation tangible des enjeux propres à chaque groupe d'utilisateurs.

Si l'équipe de création du projet vient à s'élargir, cette méthode nous permettra de partager une représentation des enjeux utilisateurs auprès des différents membres de l'équipe et ainsi faciliter la communication. Les personas sont utiles en phase amont et tout au long du projet. Ils sont utilisés comme outils d'évaluation, et comme moyen d'argumentation des choix opérés durant la conception de l'application.

Enfin, cette méthode permet de valider la compréhension des utilisateurs du système. C'est une étape primordiale car elle permet de définir les caractéristiques, les forces, les faiblesses et les habitudes de chacun et ainsi définir les fonctionnalités qui répondent à leurs attentes.

5.2 Les personas du projet

5.2.1 Les clients

Les clients du point de vente sont les collaborateurs de l'entreprise. Ils déjeunent en majorité dans les restaurants internes sur la même plage horaire, à savoir entre 12H00 et 14H00. Chaque collaborateur prend en moyenne une heure pour manger chaque jour.

Par ailleurs, le temps du repas est considérablement augmenté si ceux-ci choisissent de se restaurer à l'extérieur. Cela est dû à la distance entre l'entreprise et le restaurant extérieur (marche) et au temps d'attente qui peut excéder les trente minutes.

Les clients souhaitent, compte tenu du temps limité dédié à la restauration, perdre le moins de temps possible et être servi le plus rapidement possible. Ils aimeraient donc avoir la possibilité de commander très rapidement et d'être prévenus lorsque la commande est prête sans faire de queue afin de gagner un temps précieux. Ils souhaiteraient enfin se sentir protégés au restaurant interne que ce soit à l'entrée, aux stands, aux caisses ou à table malgré l'épidémie.

Samantha

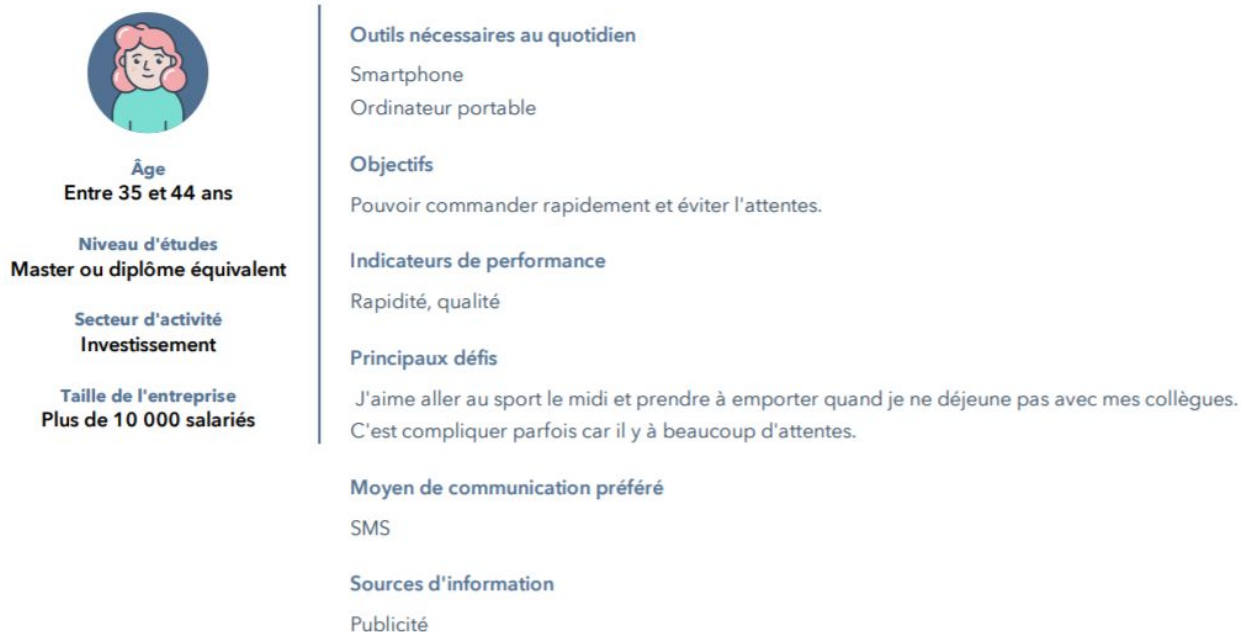


Figure 1

5.2.2 Les agents du points de vente

Les agents s'occupent de la préparation des bons de commande et de l'accueil des clients. Il doivent également veiller à l'hygiène et au bon respect des règles sanitaires et de distanciation physique.

Les agents souhaitent que les commandes soient plus fluides mais aussi aimeraient avoir une vision globale des commandes en cours afin de livrer plus rapidement les produits demandés.

Ils souhaitent un outil qui permettrait une plus grande efficacité et une diminution de leur charge de travail. En effet, les agents aimeraient préparer la commande reçue et remettre la commande prête au client de façon simplifiée.

Les bénéfices d'une commande en ligne seraient de limiter les réclamations des clients, de diminuer le niveau de stress des agents et d'obtenir un gain de temps et d'efficacité (diminution des erreurs de commande).

Cette commande à distance souhaitée permettrait enfin de limiter fortement le contact direct avec le client, ce qui favoriserait le respect du protocole sanitaire en vigueur actuellement.

Roberto



Âge
Entre 25 et 34 ans

Niveau d'études
Licence ou diplôme équivalent

Secteur d'activité
Vente

Taille de l'entreprise
11 à 50 salariés

Outils nécessaires au quotidien

- Logiciel de caisse
- Smartphone

Objectifs

Je suis étudiant en commerce et je travaille ici à mi-temps. Mon objectif est de pouvoir partir à l'heure et rejoindre mes amis. J'aime quand le service est fluide et rapide.

Responsabilités

Je dois assembler et remettre les commandes aux clients. Je dois également veiller aux mesures de distanciations physiques dans le point de vente. Et nettoyer les surfaces de contact régulièrement.

Principaux défis

- Gestion du stress pendant le pic du service
- Communications et relations avec les clients

Indicateurs de performance

Satisfaction de la clientèle, efficacité

Supérieur hiérarchique

Simone, Responsable de point de vente

Figure 2

5.2.3 Les managers du point de vente

Les managers du point de vente veillent au bon déroulement du service. Ils ont des objectifs de vente précis et remontent leur chiffre d'affaires au responsable du secteur.

Pour faciliter la création d'outil de reporting, ils souhaitent un tableau de bord pour pouvoir visualiser directement les données.

Le tableau de bord souhaité leur permettrait de gagner un temps précieux qui pourrait être réinvesti dans la gestion et l'aide des équipes, ce qui favoriserait leur efficacité. (Par exemple: être en renfort au point "accueil des clients").

Ce tableau de bord leur offrirait également la possibilité de visualiser les catégories de produits les plus vendues et les moins vendues (statistiques de la journée / historique par date), ce qui leur faciliterait le travail de gestion des stocks en fonction des demandes. (Commandes plus importantes des produits "phares" et élimination des produits "vaches à lait").

Enfin, le manager aimerait un outil favorisant une plus grande autonomie et une meilleure efficacité de son équipe, ce qui améliorerait ses objectifs.

Michelle



Âge

Entre 35 et 44 ans

Niveau d'études

Master ou diplôme équivalent

Secteur d'activité

Vente

Taille de l'entreprise

5 001 à 10 000 salariés

Outils nécessaires au quotidien

Tableur excel

Logiciel de caisse

Responsabilités

Je dois veiller au bon déroulement du service et faire en sorte que mon équipe réalise les objectifs du jours.

Indicateurs de performance

Motivation de l'équipe.

Satisfaction des clients.

Chiffre d'affaire.

Objectifs

Que les équipes soient efficace et performantes pour réaliser les objectifs tout en assurant une satisfaction optimable de notre clientèle.

Supérieur hiérarchique

Responsable de secteur

Principaux défis

Les nouvelles normes sanitaires peuvent faire baisser la confiance des clients si elles ne sont pas bien respectées.

Figure 3

CHAPITRE 2:

Spécifications fonctionnelles

1 Les acteurs du projet

L'expression des besoins des personas permet d'identifier les principaux acteurs du système. Le projet comprend trois types d'acteurs :

- Les clients (USER) ont accès au menu après s'être inscrits et authentifiés. L'authentification permettant notamment de consulter les tarifs à appliquer selon le statut du collaborateur. Ils peuvent créer une commande directement sur leur(s) smartphone(s). Ils peuvent consulter leur historique de commandes.
- Le manager (MANAGER) est celui qui a accès à toutes les fonctionnalités de l'application. Il peut annuler une commande ou supprimer un produit d'un bon de commande. Il a également accès aux statistiques de vente par jour et à l'historique des ventes.
- Les agents du point de vente (STAFF) peuvent voir les commandes réalisées par les clients et les bons de commandes correspondants pour pouvoir les traiter. Ils peuvent spécifier un bon de commande terminé. Enfin, ils peuvent signaler une erreur si un produit n'est plus disponible. Ils reçoivent une notification lorsqu'une nouvelle commande est envoyée et lorsqu'une commande est prête à être délivrée au client.

2 Les Users stories

Dans la méthode agile Scrum, la User Story illustre un besoin fonctionnel exprimé par les types d'utilisateurs. C'est en quelque sorte une phrase simple qui permet de décrire avec précision le contenu d'une fonctionnalité à développer. Elle vise ainsi à assurer que l'on délivre bien de la valeur pour nos usagers. Elle permet de coller au maximum au besoin et à la vision de l'utilisateur. La User Story facilite ainsi la tâche des équipes de développement en facilitant leur compréhension de la fonctionnalité à développer. L'emploi de la User Story engendre un gain de temps considérable. D'autant plus qu'elle permet d'aligner la vision à la fois du Product Owner, des développeurs, du Scrum Master, des testeurs et de tout autre collaborateur en les rassemblant autour d'un langage commun.

Nous allons ici indiquer les users stories qui permettront de développer l'application *Ceraon*.

2.1 Authentification

L'utilisateur souhaite pouvoir s'authentifier pour accéder aux données personnelles de manière sécurisée.

Le manager veut pouvoir s'identifier pour accéder au contenu personnalisé et sécurisé.

2.2 Prise de commande

L'utilisateur attend pouvoir voir les produits disponibles afin de choisir son repas du jour. Il désire pouvoir ajouter, modifier ou supprimer des produits de son panier avant de commander. Ainsi, la commande sera conforme au choix réalisé.

2.3 Gestion des commandes

L'agent aimerait pouvoir voir les commandes envoyées par les utilisateurs pour les traiter mais aussi signaler lorsqu'une commande est terminée, ainsi l'utilisateur pourra venir chercher son repas au bon moment sans attendre et l'agent pourra traiter les commandes suivantes directement.

L'utilisateur a besoin d'être averti s'il y a une erreur sur la commande pour la modifier tout de suite afin de ne pas perdre de temps sur la pause méridienne.

2.4 Tableau de bord

Le manager aimerait pouvoir voir les ventes du jour sous forme de graphique téléchargeable, ainsi il pourra savoir d'un seul coup d'oeil si les objectifs sont atteints et en informer sa direction.

Les US nous ont permis de créer sur notre repository Github, les issues à réaliser par priorité. Chaque issue contient les spéc à développer et les critères d'acceptation. Les templates des maquettes y seront attachés. Le client pourra ainsi valider par commentaire directement sur Github.

3 Réalisation des maquettes

Les maquettes présentent de nombreux avantages pour le développement d'une application. Le premier point positif est qu'elles offrent une représentation schématique de la version finale de l'application. Ainsi, les maquettes fournissent un premier "retour critique" de l'application, non seulement au niveau de l'aspect globale, mais également sur le contenu et la visibilité des fonctionnalités. Elles permettent donc d'avoir une vision un peu plus concrète de ce qui va

certainement être réalisé. Elles permettent aussi de s'assurer que chaque partie comprenne bien le déroulement et soit d'accord sur la direction globale que prend le projet. Elles permettent par ailleurs, d'avoir un plan d'organisation précis des interfaces et de gagner du temps lors de l'élaboration de celles-ci. Enfin, au cas où certaines interfaces ne peuvent pas être réalisées, elles permettent à d'autres développeurs de réaliser les vues manquantes.

Les maquettes ont été réalisées uniquement pour certaines vues, le but ici n'étant pas de connaître chaque section de l'application dans le détail, mais de donner une vision globale de l'interface, qui sera reprise par la suite dans les autres vues.

3.1 L'écran du menu et le panier

L'écran du menu est très important, c'est la page vers laquelle l'utilisateur est redirigé après son authentification. Le template est décliné dans deux versions: Une version pour les écrans d'ordinateur et une seconde version pour les téléphones mobiles.

Dans la version "desktop", la liste des produits se trouve sur la gauche. Les produits sont affichés par catégorie. Des onglets permettent de changer de catégorie de produits très facilement. Un clic sur un produit ouvre une fenêtre sur laquelle l'utilisateur peut choisir la quantité voulue, il est à même de valider ou d'annuler afin de fermer la fenêtre. Si validation il y a, le produit est ajouté dans le panier.

Le panier, accessible sur la droite de l'écran, permet à l'utilisateur de voir les produits qu'il a ajoutés ainsi que le montant total de son panier. Ce composant permet au client de supprimer ou de modifier une quantité d'un produit.

Un bouton "Go to checkout" visible sur le panier permet à l'utilisateur de passer à l'étape suivante de la commande. Le panier est automatiquement sauvegardé s'il est modifié.

Dans la version mobile, le panier est accessible via une icône, un badge signale à l'utilisateur si des éléments y sont présents. Un clic sur cette icon permet d'ouvrir une barre latérale qui contient le composant panier avec les mêmes fonctionnalités détaillées que celles de l'écran desktop.

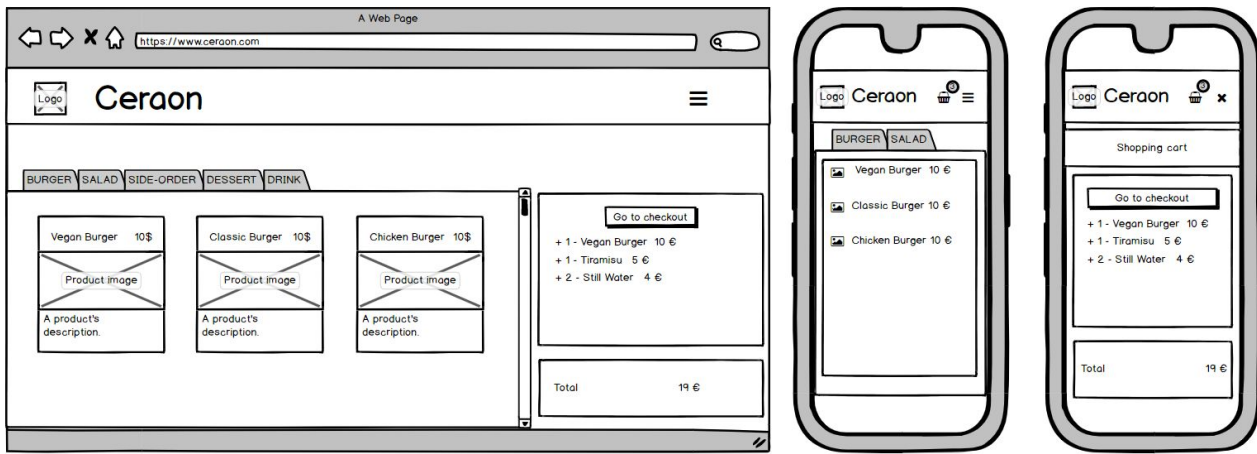


Figure 4

3.2 L'écran de validation de la commande

Cet écran est celui de la deuxième étape du processus de prise de commande. Lorsque l'utilisateur clique sur le bouton "Valider mon panier" il est alors redirigé vers une nouvelle page pour vérifier et valider sa commande. Il peut ici indiquer des informations supplémentaires, notamment s'il souhaite se restaurer sur place ou à emporter. Il a la possibilité de revenir en arrière s'il veut modifier son panier. Un composant lui indique le solde de sa carte et un lien lui permet de recharger son compte s'il le souhaite. L'utilisateur n'a pas la possibilité de valider sa commande avec un compte négatif ou présentant un solde insuffisant. La validation de la commande entraîne automatique un débit sur son compte collaborateur.

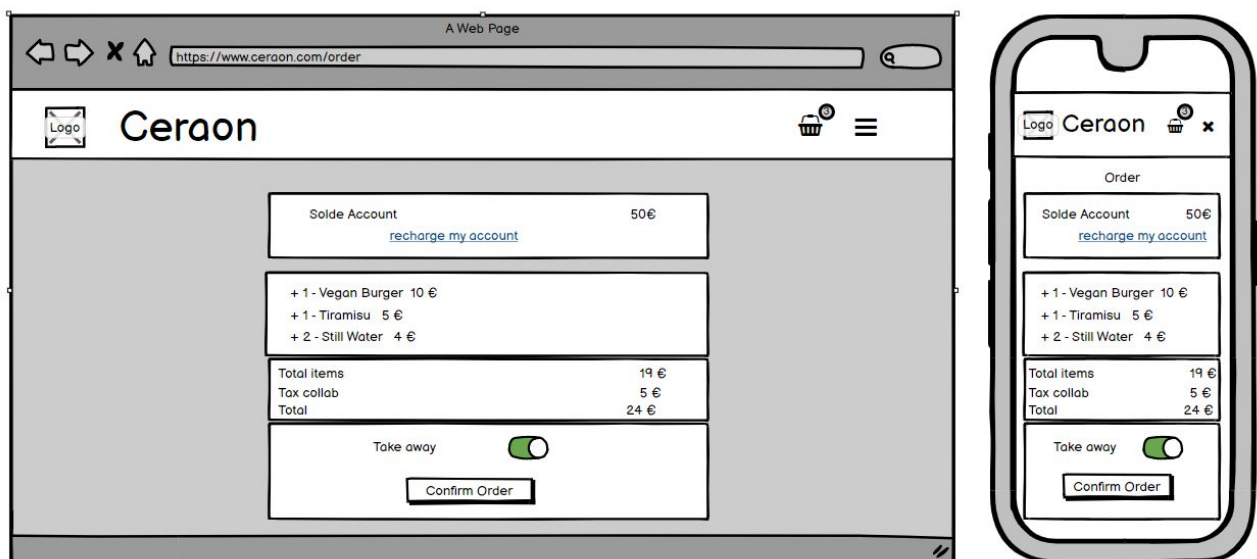


Figure 5

3.3 L'écran de gestion des commandes

Cet écran permet aux agents du point de vente ainsi qu'au manager de traiter les commandes envoyées par les utilisateurs. Ainsi, dans la maquette suivante (figure 6), nous pouvons voir l'interface de la liste des commandes. Au vu du nombre important d'informations qu'il est possible d'indiquer, nous avons choisi les informations les plus utiles afin d'être le plus clair possible pour les agents.

Ainsi, ils peuvent voir les commandes ouvertes, le statut est directement visible par un code couleur (rouge pour une commande payée, orange pour une commande terminée et vert une commande en cours).

La partie « order details » contient des informations sur une commande particulière et permet aux agents d'avoir plus d'informations sur une commande spécifique. Ce composant est ouvert en cliquant sur un bouton d'action « détails » de la commande choisie. Un bouton en haut à droite permet de fermer ce composant, la liste de commandes prend alors tout l'écran afin de gagner en visibilité.

Les boutons d'action apparaissent sur les commandes en fonction de leur statut. Ainsi, les agents savent exactement les actions qui doivent être accomplies sur chaque commande.

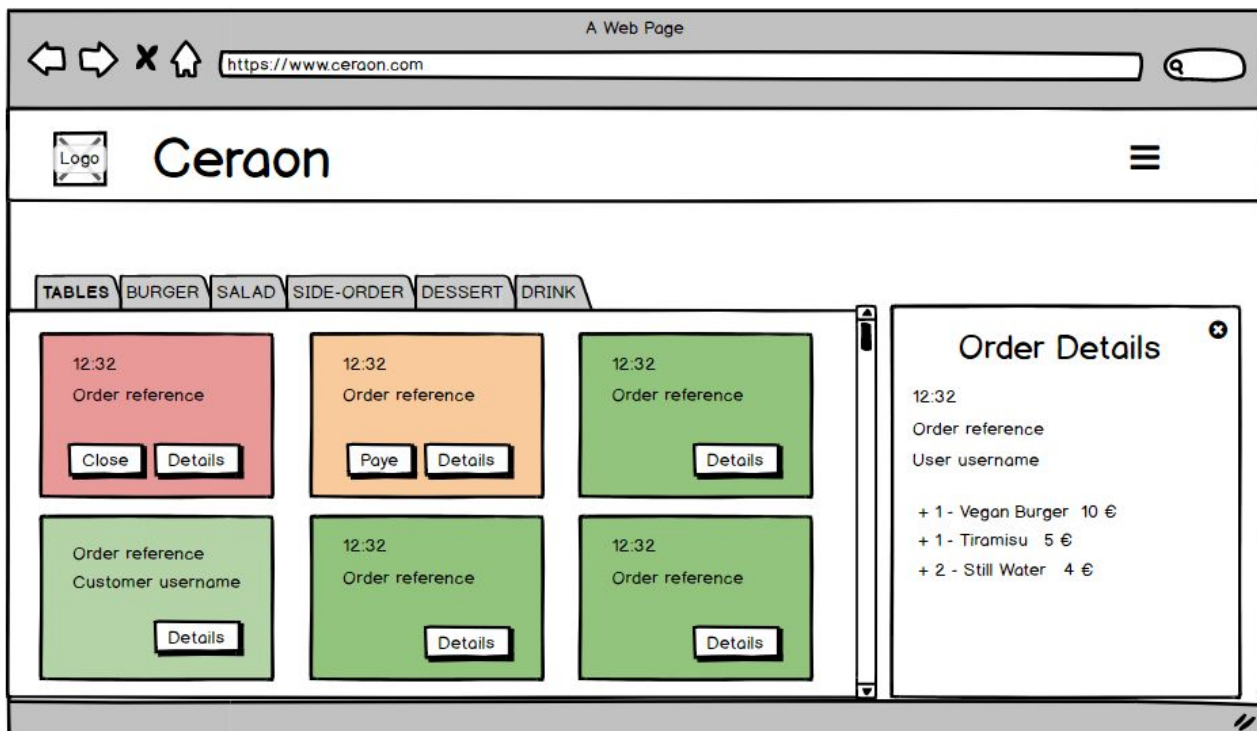


Figure 6

A l'aide du menu, une vue différente est choisie par l'agent. Ainsi, dans le template suivant (Cf. Figure 7), où le menu "BURGER" est sélectionné, nous

pouvons voir la liste des bons de commande pour la catégorie "Burger". Le ou les agents responsable(s) de la confection de ces produits peuvent alors les faire.

L'interface leur permet également de signaler lorsque les bons de commande sont terminés et prêts à être livrés aux clients mais également de signaler une erreur. Par exemple, le produit n'est plus disponible, il est immédiatement retiré du menu et n'est plus disponible à la commande.

L'utilisateur est immédiatement prévenu via une notification si cette action est réalisée. Cette fonctionnalité apporte une réelle plus-value à l'application car elle fait gagner du temps aux agents qui peuvent alors traiter d'autres commandes en attendant la modification de l'utilisateur et n'ont pas à signaler les produits manquants. De leur côté, les utilisateurs sont avertis tout de suite et ont la possibilité de choisir un nouveau produit disponible. Cette fonctionnalité est implémentée à l'aide du protocole WebSocket, nous la détaillons dans le chapitre 3.

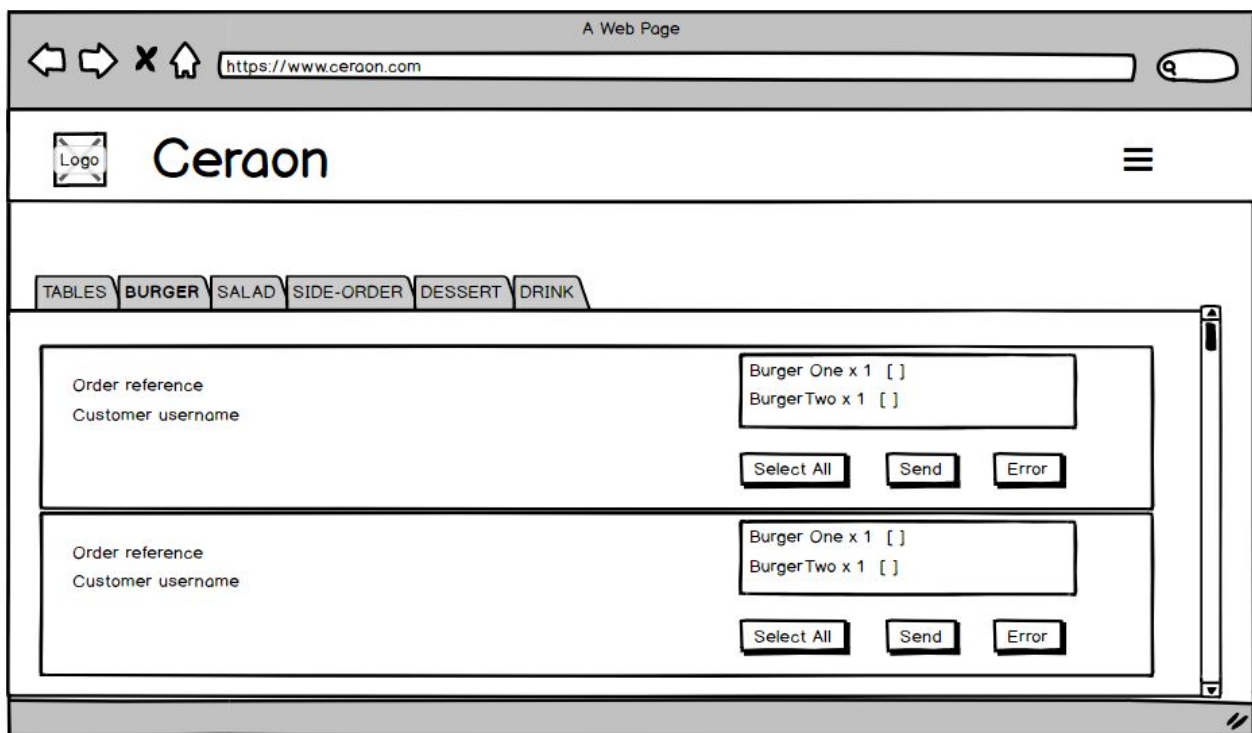


Figure 7

3.4 L'écran d'affichage du tableau de bord

Le tableau de bord est une solution qui donne au manager la capacité d'être plus productif et de gagner du temps pour réaliser des analyses des ventes. Il est majoritairement utilisé sur une version desktop. L'image ci-dessous (Cf. Figure 8) représente l'interface du tableau de bord de l'application. Celui-ci présente diverses statistiques, sous forme de texte ou de graphs (histogrammes, camemberts etc...). Il comprend une barre de filtre pour indiquer la date de la

requête, la date par défaut étant celle du jour. Les informations sont présentées à titre indicatif. Les informations à afficher sur cette page sont encore à déterminer avec le responsable du point de vente. Cette maquette a pour fonction de donner une tendance générale du tableau de bord.

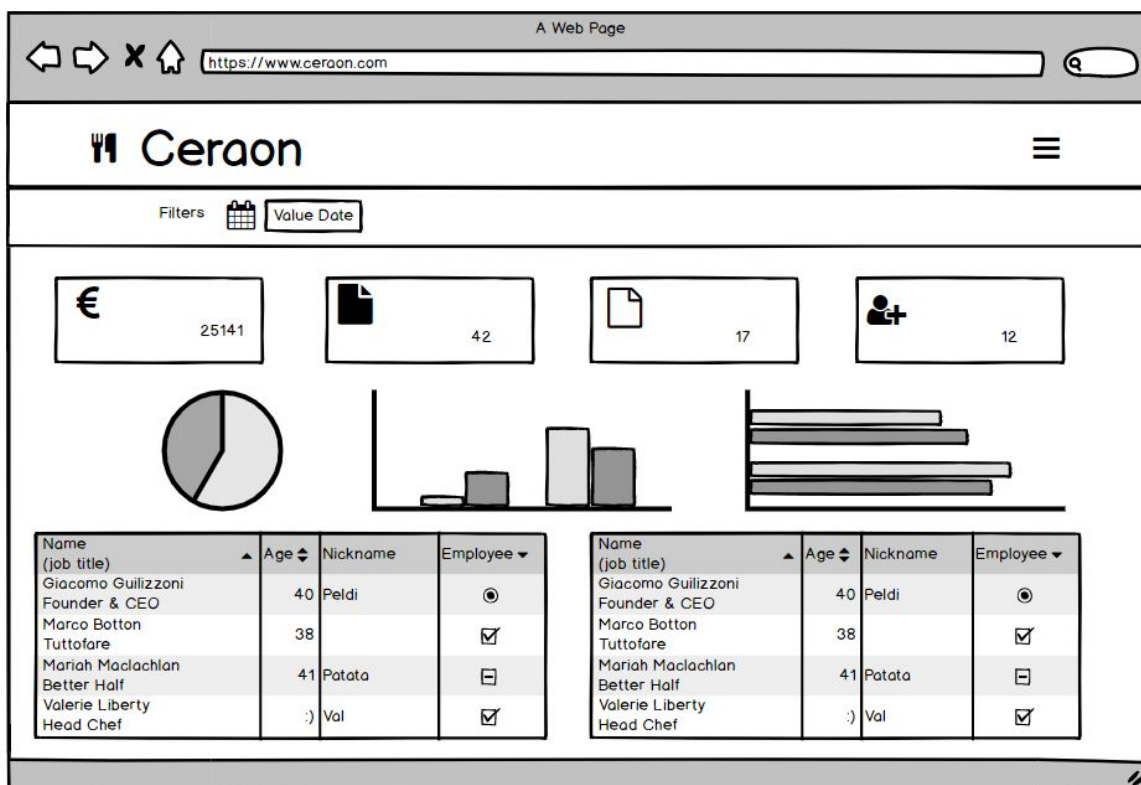


Figure 8

4 Les issues

La figure 9, ci-dessous, présente le processus de développement sur Github. Les users stories nous ont permis de créer des issues. Celles-ci contiennent toutes les spécifications à développer mais aussi les critères d'acceptations et les templates correspondants.

As a member of staff, I want to be able to see the new orders so that I can make it. #2

Open patituka opened this issue on 5 Jul · 0 comments

The screenshot shows a GitHub issue page for the repository 'Ceraon'. The issue title is 'As a member of staff, I want to be able to see the new orders so that I can make it. #2'. The issue was opened by 'patituka' on July 5th. The issue content includes 'Spécifications fonctionnelles détaillées' (detailed functional specifications) and 'Critères d'acceptations' (acceptance criteria). The specifications list three points: displaying orders by product type, signaling errors, and notifying the client. The acceptance criteria are 'En cours de développement' (in development) and 'En phase d'acceptation' (in acceptance phase). A mockup of the 'Ceraon' web interface is shown, featuring a navigation bar with 'TABLES', 'BURGER', 'SALAD', 'SIDE-ORDER', 'DESSERT', and 'DRINK'. The main content area displays two order forms, each with 'Order reference' and 'Customer username' fields, and a list of items: 'Burger One x 1' and 'Burger Two x 1'. The right sidebar shows the issue's metadata, including assignees, labels, projects, milestones, linked pull requests, notifications, and participants.

Figure 9

Ci-dessous, la figure 10 représente les fiches du board du projet tel qu'il était il y a quelques mois. Ces fiches une fois créées sont priorisées.

The screenshot shows a project board for the repository 'Ceraon', updated on June 17th. The board is divided into three columns: 'To do' (3 results), 'In progress' (1 result), and 'Done' (0 results). The 'To do' column contains three user stories: 'As a Manager, I want to be able to see the sale data in one click so that I can analyse it.' (opened by patituka), 'As a member of staff, I want to be able to see the new orders so that I can make it.' (opened by patituka), and 'As a user, I want to be able to authenticate so that a can access my personal data.' (opened by patituka). The 'In progress' column contains one user story: 'As an user I want to be able to browse through the menu so that I can choose before make an order.' (opened by patituka). The 'Done' column is empty.

Figure 10

CHAPITRE 3:

Spécifications techniques

La phase de conception de l'application est la plus importante du projet. Elle consiste à modéliser l'application et à préparer son développement, de manière à ce qu'elle s'intègre correctement dans le point de vente. A l'issue de cette phase de conception, nous sommes censés avoir un cahier des charges fonctionnel. Nous devons également avoir des documents comparatifs concernant les langages et les frameworks disponibles qui expliquent les critères déterminant le choix d'une technologie par rapport aux autres. Nous sommes capables également de prévoir plusieurs autres documents qui expliquent le fonctionnement de l'application, les technologies utilisées ainsi qu'un document résumant la phase de modélisation de la base de données.

1 Diagrammes de cas d'utilisations

La connaissance des fonctionnalités à implémenter est essentielle pour établir le diagramme des cas d'utilisation de l'application. L'étude réalisée préalablement permet d'avoir des éléments solides pour lister les fonctionnalités à implémenter et faciliter la réalisation de ce diagramme des cas d'utilisation. L'objectif étant de décrire les exigences du fonctionnement de l'application, chaque cas d'utilisation correspondant à une fonction métier. Dans cette section, nous allons détailler les différents diagrammes des cas d'utilisation réalisés.

1.1 Définition

Le diagramme des cas d'utilisation permet de modéliser les besoins des utilisateurs en identifiant les grandes fonctionnalités du système et en représentant les interactions fonctionnelles entre les acteurs et les fonctionnalités. Il est l'un des diagrammes les plus structurants dans l'analyse d'un système. Les éléments de base de cas d'utilisation sont :

- L'acteur : Entité externe qui agit sur le système et qui peut consulter ou modifier l'état du système. En réponse à l'action d'un acteur, le system fournit un service qui correspond à son besoin.
- Les cas d'utilisations : Ensemble d'actions réalisées par le système, en réponse à une action d'un acteur. L'ensemble des cas d'utilisations décrit les objectifs du système.

1.2 Le diagramme de cas d'utilisation de la prise de commande

La figure ci-dessous (Cf. Figure 11) représente le diagramme des cas d'utilisation de la prise de commande par le client. L'agent principal de ce diagramme est l'utilisateur. L'application doit pouvoir lui permettre de prendre une commande, et pour cela implémenter des fonctionnalités comme la consultation de la liste des produits et la création du panier. Pour pouvoir accéder à ces fonctionnalités, l'utilisateur est obligé de se connecter.

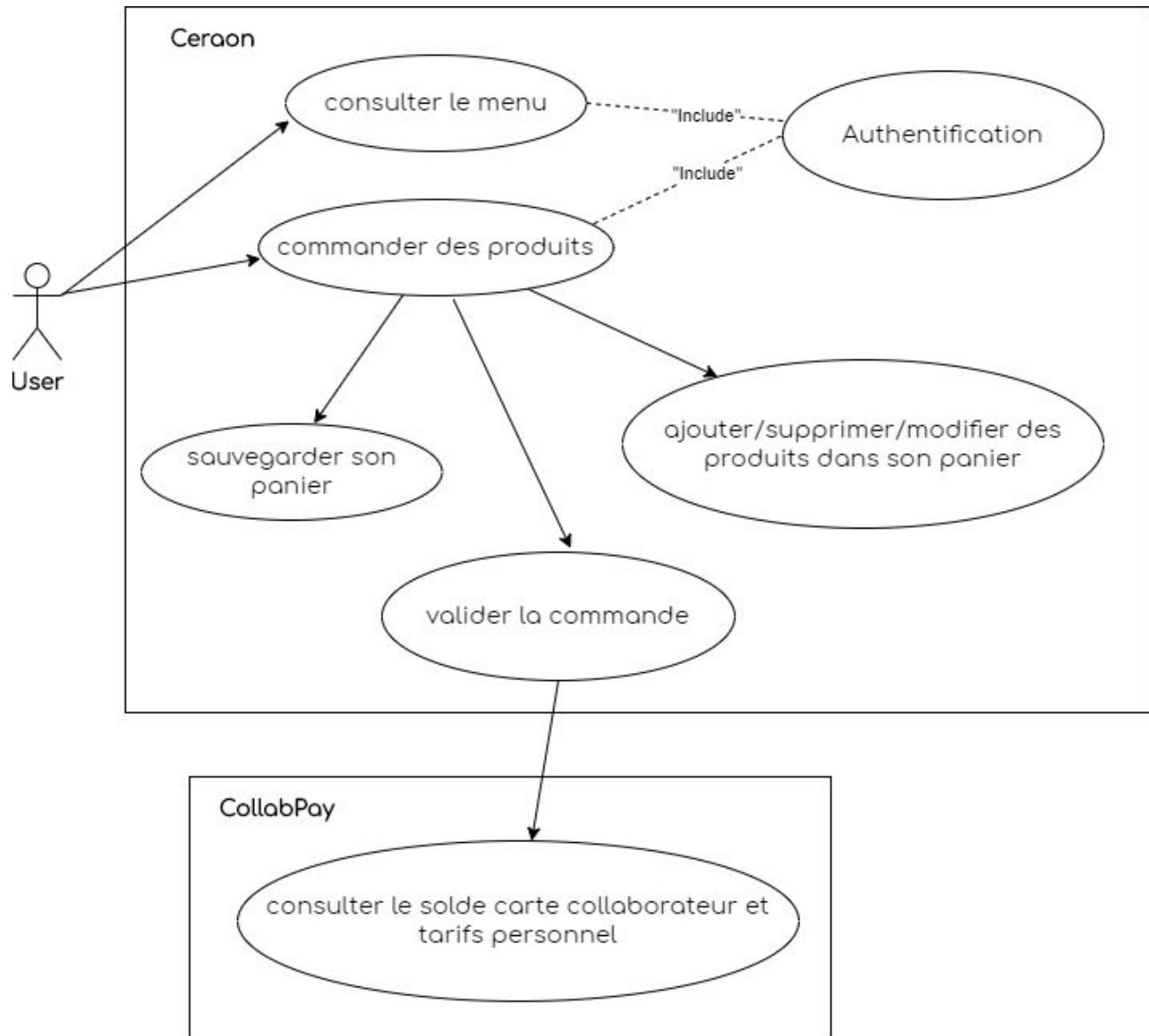


Figure 11

1.3 Le diagramme des cas d'utilisation de la gestion des commandes

La figure ci-dessous (Cf. Figure 12) représente le diagramme des cas d'utilisation de la gestion des commandes. Comme nous pouvons le constater, les deux acteurs principaux de ce diagramme sont le manager et l'agent du

point de vente. L'application doit donc pouvoir leur permettre de gérer les commandes, et pour cela implémenter des fonctionnalités comme la consultation de la liste des commandes, en fonction de leur état.

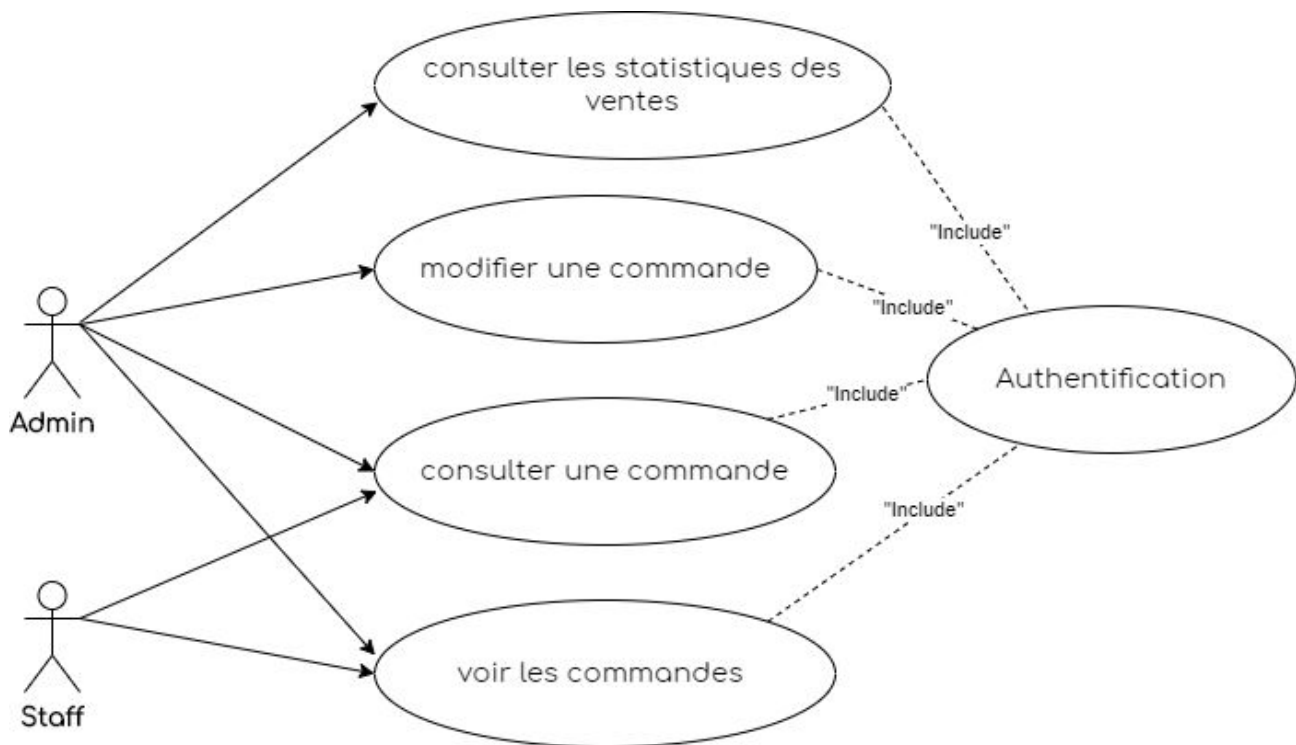


Figure 12

1.4 Le diagramme de cas d'utilisation de la gestion du compte personnel

L'application doit également permettre de gérer son compte utilisateur comme le montre le diagramme des cas d'utilisation suivant (Cf. Figure 13). Notre application propose de s'authentifier afin de sauvegarder de manière sécurisée les données personnelles et de permettre aux utilisateurs de les consulter.

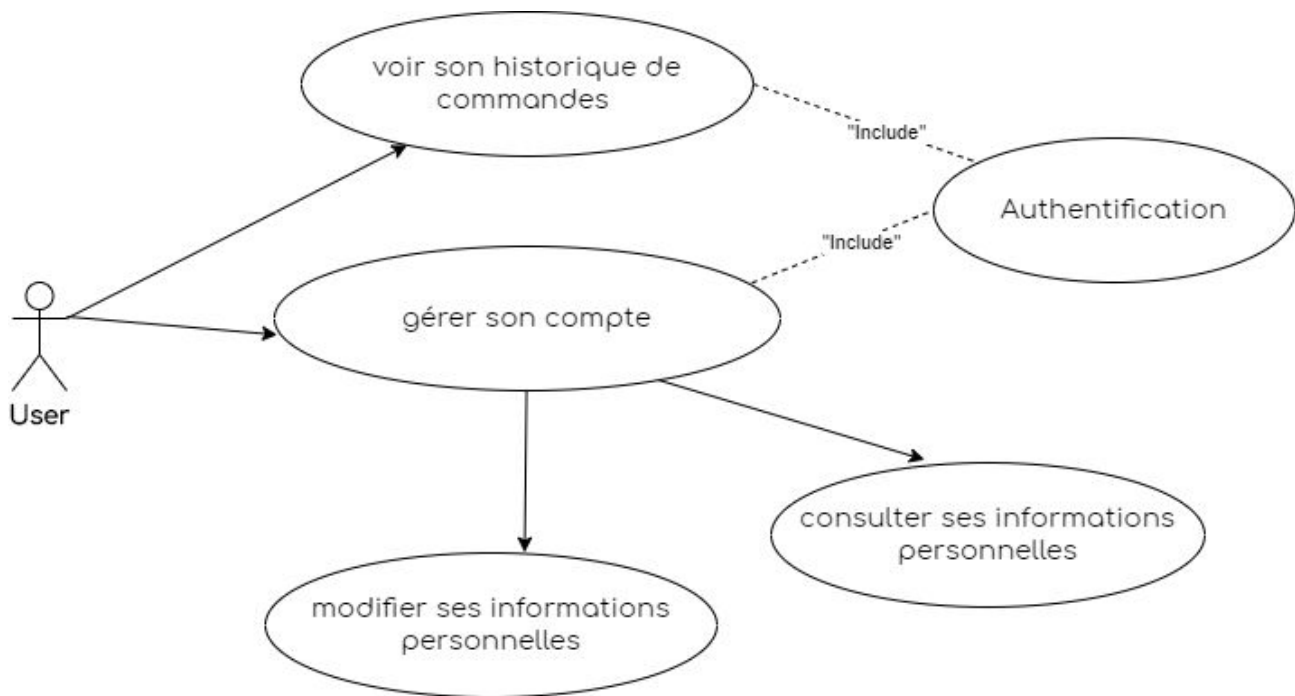


Figure 13

2 Diagrammes de séquences

Afin de décrire plus en détail les scénarios des cas d'utilisations du système, nous choisissons de réaliser des diagrammes de séquences. Un diagramme de séquences est un diagramme d'interaction qui expose en détail la façon dont les opérations sont effectuées : quels messages sont envoyés et quand ils le sont. Ainsi, nous allons voir les échanges de messages et les appels d'opérations entre les objets qui permettent au système de répondre à une sollicitation de son environnement.

2.1 Définition

Les diagrammes de séquences montrent d'une part les opérations entre les acteurs du système, à savoir ses utilisateurs et son environnement, et d'autre part, le système lui même représenté en un ensemble d'objets. Les messages qui sont échangés peuvent être des opérations liées aux cas d'utilisations ou des opérations internes qui appartiennent aux classes du système. Les échanges de messages sont représentés de manière chronologique, c'est-à-dire de haut en bas.

Les diagrammes de séquences sont composés des éléments suivants :

- Les lignes de vie : Une ligne verticale qui représente la séquence des événements, produite par un participant, pendant une interaction, alors que le temps progresse en bas de ligne. Ce participant peut être une instance d'une classe, un composant ou un acteur.

- Les messages : Il y a deux types de messages dans le diagramme de séquences, le premier est dit "message synchrone" utilisé pour représenter des appels de fonction ordinaires dans un programme, le deuxième est appelé "message asynchrone", utilisé pour représenter la communication entre des threads distincts ou la création d'un nouveau thread.
- Les occurrences d'exécution : Elles représentent la période d'exécution d'une opération.
- Les commentaires : Ils peuvent être joints à tout point sur une ligne de vie.
- Les itérations : Ils représentent un message de réponse suite à une question de vérification.

2.2 Diagramme de séquence : enregistrement du panier d'achat

Après avoir choisi ses produits, le client clique sur le bouton "Valider mon panier". Si un panier est déjà présent et que celui-ci n'a pas été modifié par l'utilisateur alors ce bouton redirige ledit client vers la page de confirmation de la commande. Sinon, cette action entraîne un appel HTTP pour persister le panier d'achat en base de donnée. C'est cet événement qui est détaillé dans le diagramme de la figure 14 ci-dessous. Depuis la couche service, on va d'abord vérifier si un panier a déjà été enregistré précédemment. Si un panier existe déjà, il est modifié avec l'ajout des nouveaux produits. Chaque changement entraîne une modification de la quantité de l'instance du produit supprimé ou ajouté au panier. Si aucun panier n'existe, un nouveau panier est initialisé avant l'ajout des produits.

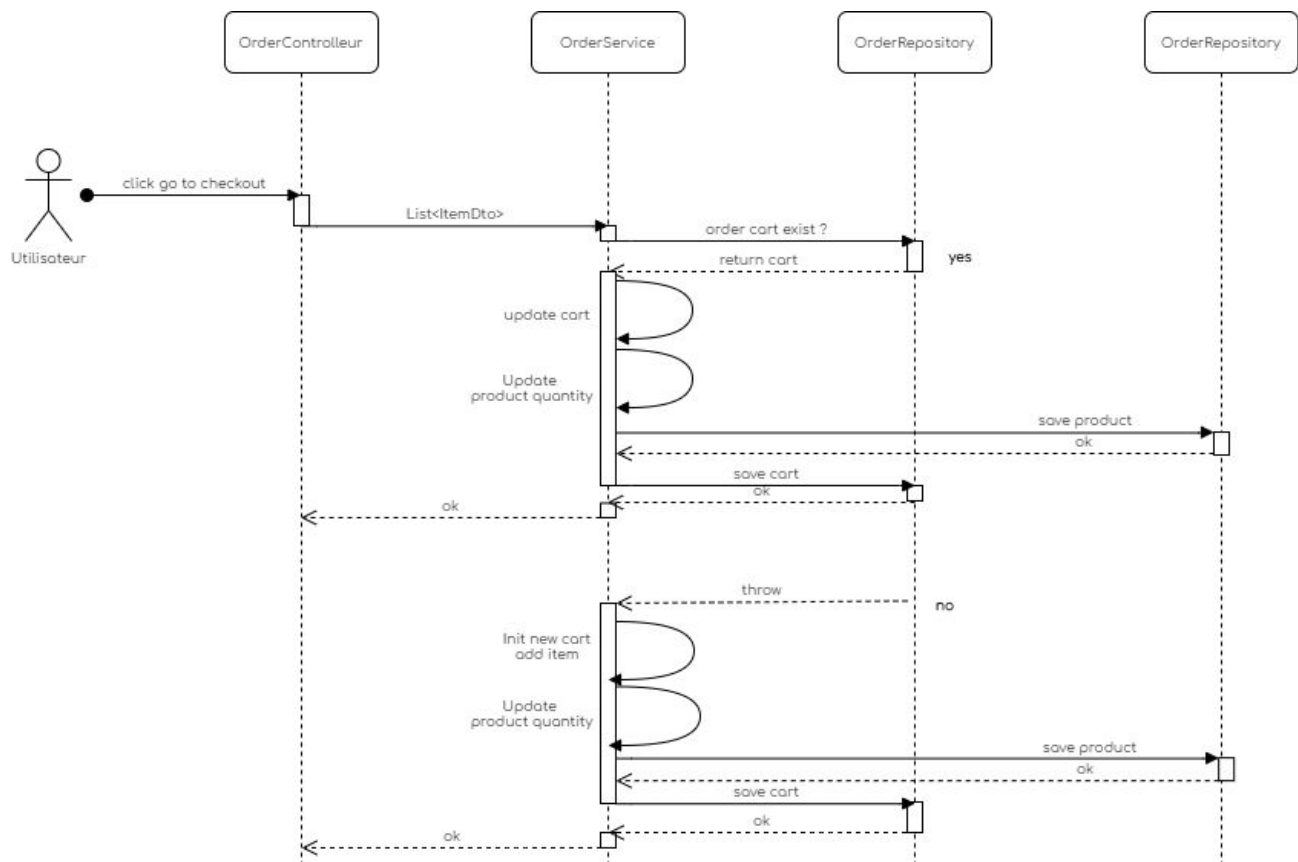


Figure 14

2.3 Diagramme de séquence : confirmation du panier d'achat

Nous passons maintenant à l'étape suivante, c'est-à-dire que le client se trouve sur la page de confirmation de la commande. Il doit cliquer sur le bouton "Confirmer ma commande". Cette action entraîne un appel HTTP qui est détaillé dans le diagramme ci-dessous (Cf. Figure 15).

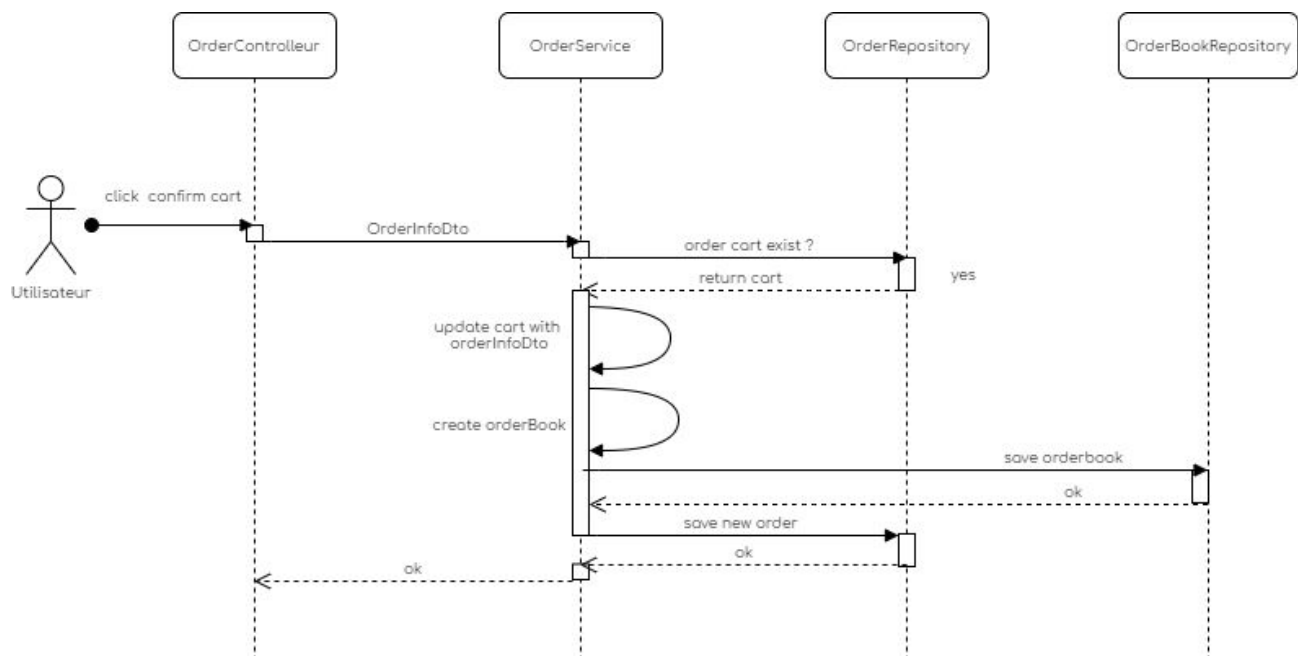


Figure 15

2.4 Diagramme de séquence : gestion des commandes

Ce diagramme représente la séquence qui suit l'action de l'agent lorsqu'il va cliquer sur le bouton pour signaler que le ou les produits sélectionnés sont terminés.

Lorsque l'agent signale une erreur, si le produit n'est plus disponible, la commande de l'utilisateur client doit être modifiée et le produit sera indiqué non visible. Il n'apparaîtra plus dans le menu et ne pourra plus être ajouté au panier d'achat.

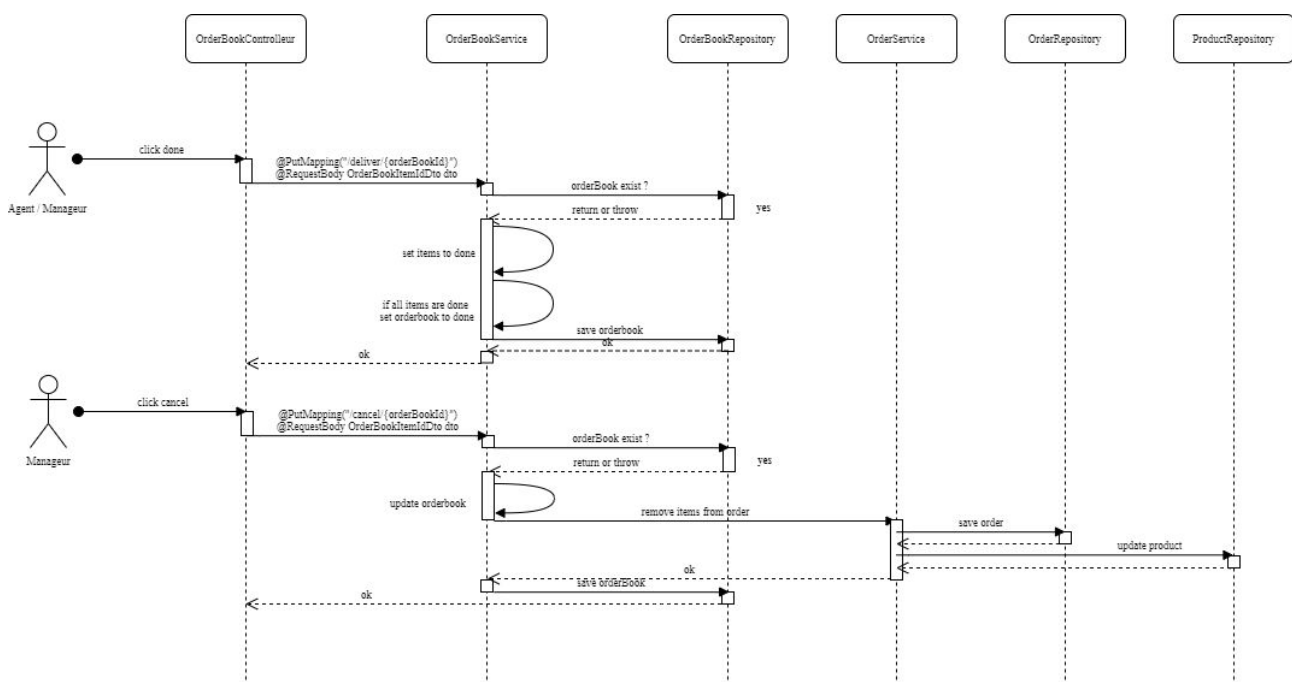


Figure 16

3 Diagramme de classes

La figure 17, ci-dessous, représente le diagramme de classes actuel de la modélisation. (Dernière version)

On peut remarquer qu'une commande (Order) ne peut exister sans un utilisateur (Account) et que celui-ci peut faire plusieurs commandes. La classe Order est au centre de l'application. Le diagramme de classes permet également de spécifier les types de relation. Aussi, une classe peut-être prépondérante sur une autre

dans le sens où une classe peut-être de type "composant" et une autre de type "composite". Nous pouvons voir dans ce diagramme (Cf. Figure 17) qu'une commande contient un ensemble de bons de commande (Order Books) et qu'un bon de commande est un ensemble d'articles (Order Book Items). Par ailleurs, on parle d'agrégation faible pour la relation entre Order et Order Books et d'agrégation forte (ou de composition) pour la relation entre Order Books et Order Book Items. L'agrégation faible est visible par un losange vide alors que l'agrégation forte est signalée par un losange noir. Ainsi, une commande peut exister sans bons de commande, c'est le cas lorsque la commande a le statut "CART" signifiant "panier de commande". En revanche, un bon de commande contient obligatoirement un ensemble d'articles. La suppression d'un bon de commande entraîne forcément la suppression des articles qui le compose. En d'autres termes, la suppression d'une commande n'est possible que si aucun bon de commande ne lui est associé. En revanche, la suppression d'un compte client n'entraîne pas la suppression des commandes qu'il a effectuées, ces commandes étant terminées et comptabilisées dans les ventes réalisées du point de vente.

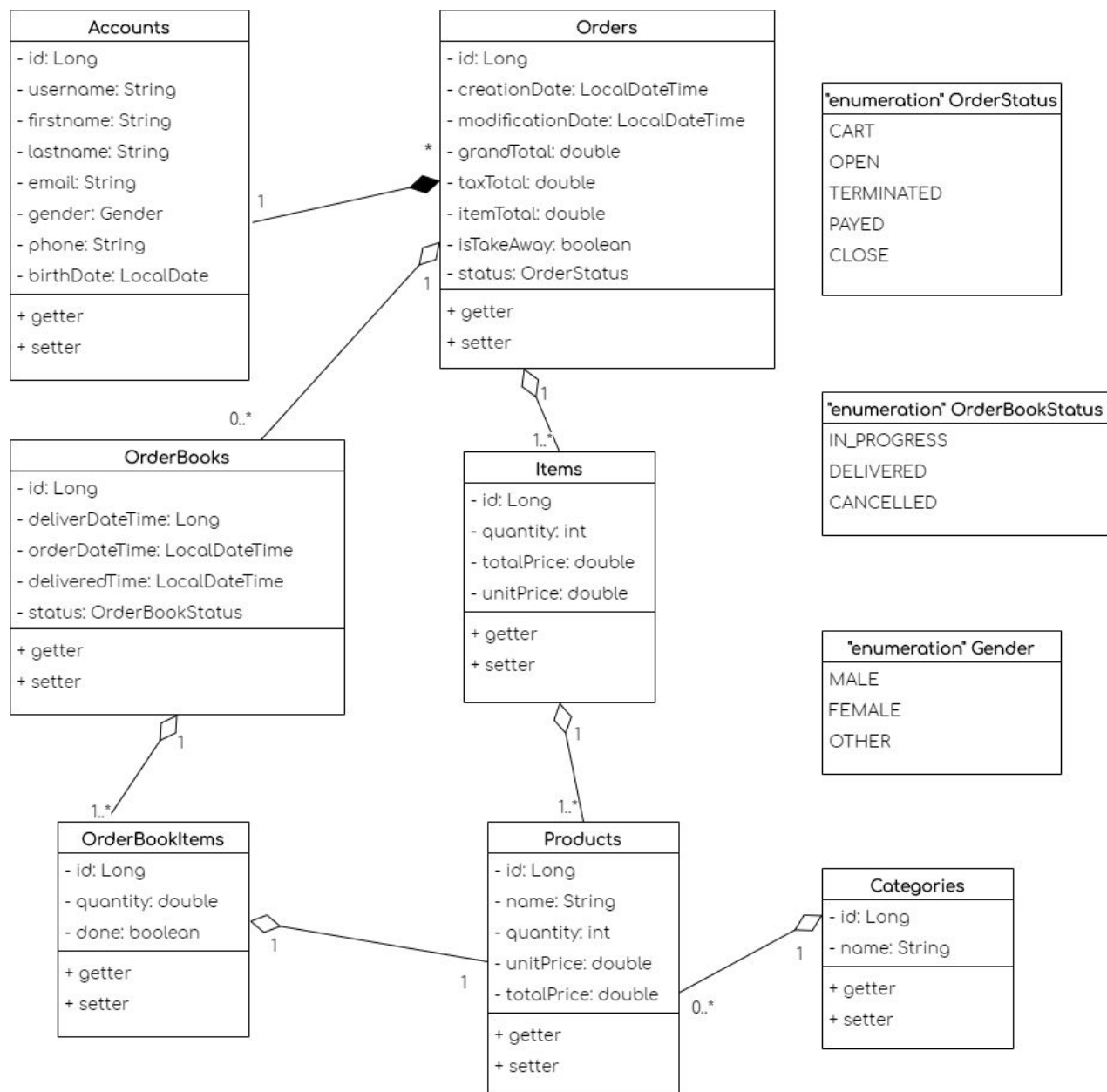


Figure 17

4 Notions de base des protocoles de communication sur le Web

Deux ordinateurs peuvent échanger des informations par connexion filaire ou par Wifi. Lorsque plus de deux ordinateurs sont reliés et s'échangent des informations, on appelle ce phénomène "un réseau informatique". Internet est donc un réseau de réseaux. Pour gérer la transmission de données d'internet, deux protocoles sont utilisés et constituent le fondement d'Internet.

- Le protocole IP pour Internet Protocol qui permet d'attribuer une adresse unique à chaque ordinateur (nommée « adresse IP ») et fournit les mécanismes pour acheminer les données à bon port.
- Le protocole TCP, pour Transfert Control Protocol qui compose des échanges de paquets IP pour proposer des services plus adaptés.

Le web, ou plus précisément le World Wide Web, est un système documentaire construit sur Internet dans lequel les documents, nommés hypertextes ou pages web, sont reliés les uns aux autres par des hyperliens. Ces documents sont affichés dans des navigateurs qui permettent, grâce aux hyperliens, de naviguer d'une page à une autre. Les données qui sont échangées sur le web le sont avec le protocole HTTP, HyperText Transfer Protocol.

Lorsque deux machines s'échangent des données sur le web avec le protocole HTTP, leur rôle est asymétrique. Le serveur fournit des ressources ou des services, tandis que le client, utilise les ressources et les services du serveur. Le protocole HTTP fonctionne selon un principe de requête/réponse : le client formule une requête auprès du serveur, lequel renvoie en retour une réponse au client. Le terme "client" désigne à la fois l'ordinateur qui émet la requête et le navigateur web. Le serveur désigne l'ordinateur destinataire de la requête et le programme chargé de traiter la requête et de formuler une réponse.

La première fois qu'un client envoie une requête HTTP à un serveur, il ouvre une connexion TCP avec le serveur. Il envoie alors sa requête à travers cette connexion, et le serveur envoie en retour sa réponse par la même connexion. Cette connexion TCP peut être maintenue ouverte pour des requêtes ultérieures. HTTP spécifie la structure des messages que le client et le serveur échangent à travers la connexion TCP.

HTTP est un protocole d'échange d'informations sur le Web basé sur le TCP/IP. La cible d'une requête HTTP est une ressource identifiée par une URL. Une URL est composée de quatre éléments qui permettent d'identifier les ressources de manière unique sur Internet : le protocole, l'hôte, le port, et le chemin (qui mène à la ressource sur le serveur). Le protocole HTTP catégorise aussi les codes "retour" d'une requête. Cela permet de connaître l'état d'une requête et de trouver l'erreur plus rapidement en cas de problème.

- 1xx pour les codes d'informations
- 2xx pour les codes de succès
- 3xx pour les codes de redirection
- 4xx pour les erreurs du client
- 5xx pour les erreurs du serveur.

Pour notre projet, nous utilisons nous aussi la méthodologie REST. Ce n'est pas un protocole mais un style d'architecture utilisé pour la conception de services Web. Il a été introduit en 2000 par Roy Fielding. Cette méthodologie est une alternative au protocole SOAP. L'architecture REST fournit un ensemble de contraintes qui permettent d'accéder aux données et de les manipuler.

L'adjectif anglais « RESTful » qualifie les systèmes qui respectent les contraintes architecturales de REST. Ces systèmes sont utilisés par les colosses de l'informatique (Google, Amazon...), REST se base uniquement sur le protocole HTTP pour la communication client-serveur.

S'il est possible de faire une lecture avec POST ou faire une suppression avec GET, l'application ne respecte plus les exigences REST et n'est plus « RESTful ». En effet, la méthode GET par exemple, est censée être utilisée uniquement pour la lecture d'information.

Il est important de noter que la réponse à une requête n'est pas une ressource, c'est la représentation de la ressource que le client reçoit. En effet, une ressource peut avoir plusieurs représentations (XML, JSON, HTML et TEXT). Quand le client demande une ressource, il doit définir le format de réponse qu'il souhaite.

Cependant, un des principes intéressants de REST est qu'il fournit en plus du résultat de la demande, des liens hypermédia permettant d'accéder à d'autres fonctions ou d'autres ressources disponibles sur le serveur (HATEOAS: Hypermedia As The Engine Of Application State). Par exemple, si un appel vers le serveur demande la liste des cinq premières commandes alors ce dernier retourne non seulement la liste, mais envoie également un lien permettant d'aller consulter les cinq suivantes ou renvoie vers d'autres fonctions notamment la suppression ou la modification.

En conclusion, REST propose des contraintes architecturales qui permettent à un client et à un serveur d'échanger des données par le biais du protocole HTTP.

5 L'architecture

Dans cette partie, nous allons, dans un premier temps, examiner les mécanismes et les protocoles nécessaires au fonctionnement d'une application Web. Dans un second temps, nous présenterons à la fois l'architecture choisie pour l'application *Ceraon* et la réalisation du diagramme d'architecture.

1.5.1 Généralités

L'objectif d'un système d'information est de permettre à plusieurs utilisateurs d'accéder aux mêmes informations. Pour cela, il faut centraliser des données au sein d'une base de données. L'évolution des systèmes d'information est basée sur une meilleure subdivision entre les tâches à réaliser pour permettre l'exploitation de ces données par les utilisateurs finaux. Ceci permet de structurer plus efficacement les informations, ce qui permet une meilleure organisation et une meilleure efficacité technique.

Tout système d'information nécessite la réalisation de trois groupes de fonctions : le stockage des données, la logique applicative et la présentation. Ces trois

parties sont indépendantes les unes des autres. Ainsi, nous pouvons modifier la présentation sans modifier la logique applicative. La conception de chaque partie doit également être indépendante, toutefois la conception de la couche la plus basse est utilisée dans la couche d'au-dessus. Ainsi, la conception de la logique applicative se base sur le modèle de données alors que la conception de la présentation dépend de la logique applicative. Regardons plus en détail ces trois parties:

Le stockage et l'accès aux données: Le système de stockage des données a pour but de conserver une quantité plus ou moins importante de données de façon structurée. On peut utiliser pour cette partie des systèmes très variés qui peuvent être des systèmes de fichiers, des mainframes, des systèmes de base de données relationnelles, etc... Le point commun entre tous ces systèmes est qu'ils permettent tous le partage des données qu'ils contiennent via un réseau.

La logique applicative: La logique applicative est la réalisation informatique du mode de fonctionnement du système. Cette logique constitue les traitements nécessaires sur l'information afin de la rendre exploitable par chaque utilisateur. Les utilisateurs peuvent avoir des besoins très variés et évolutifs. Il devient alors nécessaire de permettre l'évolution du système sans pour autant devoir tout reconstruire. Cette partie utilise les données pour les présenter de façon exploitable par l'utilisateur. Il convient donc de bien identifier les besoins des utilisateurs afin de réaliser une logique applicative utile tout en structurant les données utilisées.

La présentation: c'est la partie visible pour l'utilisateur. On parle d'interface homme-machine, soit une interface permettant à l'homme d'interagir avec la machine. Elle a donc une importance primordiale pour rendre attrayante l'utilisation de l'application. Elle est rendue possible grâce aux langages de rendus, en l'occurrence pour une application Web: Le HTML5, le CSS3 et le Javascript (qui ajoute une partie fonctionnelle à ce rendu).

5.2 L'architecture client-serveur

Ce type d'architecture est constitué uniquement de deux parties: Un client qui gère la présentation et la logique applicative d'une part, un serveur qui stocke les données de façon cohérente et gère éventuellement une partie de la logique applicative d'autre part. Dans ce type d'architecture, on constate une certaine indépendance du client par rapport au serveur. On peut également constater deux inconvénients à cette technologie: Elle possède une sécurité limitée et surtout son déploiement peut-être long et laborieux.

5.3 L'architecture 3-tier

L'architecture 3-tier consiste à séparer la réalisation et l'implémentation des trois parties vues précédemment (stockage des données, logique applicative, présentation). Tout comme dans le client-serveur, cette séparation signifie qu'il est possible de déployer chaque partie sur un serveur indépendant. La mise en place de ce type d'architecture permet une plus grande évolutivité du système. Il est ainsi possible de commencer par déployer les deux serveurs sur la même machine, puis de déplacer le serveur applicatif sur une autre machine lorsque la charge devient excessive. Les éléments permettant la réalisation classique d'un système en architecture 3-tier sont les suivants:

- Un système de base de données relationnel pour le stockage des données
- Un serveur applicatif pour la logique applicative
- Un navigateur web pour la présentation

5.4 L'architecture multi-tier

Egalement appelée architecture distribuée, une architecture multi-tiers va plus loin dans le découpage de l'application. Les deux parties développées côté serveur sont déployées sur plusieurs serveurs. L'objectif de ce type d'architecture est de permettre l'évolutivité du système sur plusieurs aspects: la quantité des données stockées, la disponibilité du serveur, le nombre d'utilisateurs.

Un système conçu en architecture multi-tiers doit être indépendant du serveur sur lequel il s'exécute. Il existe deux types de répartition possibles dans cette architecture: Il est possible de répartir les données et de répartir la logique applicative. Ces répartitions permettent de résoudre des problèmes de natures différentes. Elles peuvent donc être mises en place soit séparément, soit en parallèle, sur le même système. Regardons plus en détail les deux types de répartition existantes:

Répartition des données sur différents serveurs: Il s'agit d'utiliser plusieurs sources de données dans une même application. Chaque source possède sa propre structure de données. Chaque serveur de données peut être géré de façon très différente. Toutefois, une interface de programmation commune à toutes les sources doit pouvoir exister. Il peut exister des relations entre les données des différents serveurs, il est alors nécessaire de gérer des transactions distribuées entre les différents serveurs de données. Cette répartition des données correspond à la notion de base de données distribuées. Les bases de données distribuées permettent de résoudre deux types de problèmes. La première est la performance du système: la répartition des données permet d'augmenter la disponibilité des données. Toutefois, il est nécessaire de bien penser cette répartition afin de ne pas démultiplier le nombre de transactions distribuées nécessaires. Ceci aurait pour effet de diminuer la performance plus que de l'augmenter. Le deuxième type de problèmes est la réutilisation des systèmes existants. En effet, de nombreux systèmes informatiques stockent

actuellement une grande quantité de données. Toutefois, ces systèmes ne sont pas toujours bien coordonnés entre eux. Ceci risque d'entraîner des redondances dans les données et des incohérences entre plusieurs sources de données. L'objectif est donc de réutiliser ces systèmes dans une même application afin de restructurer le système d'information sans pour autant repartir de zéro (Ce qui est de toute façon impossible). Les systèmes réutilisés sont bien souvent hétéroclites (mainframe, SGBDR,...) et nécessitent d'être réunis en utilisant diverses technologies (moniteurs transactionnels, serveurs d'applications,...).

Répartition de la logique applicative: La répartition de la logique applicative permet de distribuer les traitements sur différentes machines. Cette distribution est facilitée par l'utilisation de la programmation orientée objet et plus particulièrement de ce qu'on appelle les composants. Un composant possède entre autre la caractéristique d'être accessible à travers le réseau. Un composant peut ainsi être instancié puis utilisé au travers du réseau. Il est également possible de trouver un serveur permettant l'utilisation d'un composant, ce qui permet une forte évolutivité ainsi qu'une résistance aux pannes importantes (le service sera toujours disponible sur un serveur même si une machine tombe en panne). La réalisation de ce type de répartition nécessite l'utilisation de technologies spécifiques. D'une part, il faut permettre la communication entre les différents éléments de l'application (RMI), d'autre part le déploiement et l'évolution du système doivent pouvoir être assurés. Certaines technologies proposent une architecture cohérente pour la mise en place des différentes technologies intervenant sur un système distribué (EJB). Les buts recherchés lors de la mise en place de ce type d'architecture sont la performance, l'évolutivité, la maintenabilité...

5.5 Le diagramme d'architecture du projet

Ceraon est programmée sur un mode d'architecture multi-tier. Cette architecture est fiable, disponible, et d'une excellente évolutivité. L'application est déployée uniquement sur la partie "serveur". Il suffit d'installer un navigateur web pour que le client puisse accéder à l'application. Cette facilité de déploiement a pour conséquence non seulement de réduire son coût mais aussi de permettre une évolution régulière du système. Cette évolution ne nécessite que la mise à jour de l'application sur le serveur applicatif. Ceci est très important car l'évolutivité est un des problèmes majeurs de l'informatique.

En outre, cette architecture est plus sécurisée. En effet, dans un système client-serveur, tous les clients accèdent à la base de données, ce qui la rend vulnérable. Avec une architecture multi-tiers, l'accès à la base est effectué uniquement par le serveur applicatif. Ce serveur est le seul à connaître la façon de se connecter à cette base. Il ne partage aucune des informations permettant

l'accès aux données avec la base de données, en particulier le login et le password de la base. Il est alors possible de gérer la sécurité au niveau de ce serveur applicatif, par exemple en maintenant la liste des utilisateurs avec leurs mots de passe ainsi que leurs droits d'accès aux fonctions du système.

De plus, cette architecture nous permet d'ajuster sur chaque couche la RAM, le CPU et les disques selon les besoins. Nous avons besoin de RAM et de CPU sur le tier web et le tier stockage a surtout besoin de disque.

La figure 18, ci-dessous présente l'architecture de *Ceraon*. Celle-ci est composée d'un serveur qui permet au client de récupérer l'application front-end. Il y a un lien entre le client (navigateur, app mobile) et ce serveur qui permet de télécharger l'application. Ensuite, le client communique avec les serveurs backend (auth et resource). Le serveur resource communique également avec Collab Pay, une application externe qui permet de consulter le compte de la carte du collaborateur et de débiter ce même compte lorsqu'un achat est effectué. Les deux serveurs backend et l'application Collab Pay communiquent avec leur base de données respective.

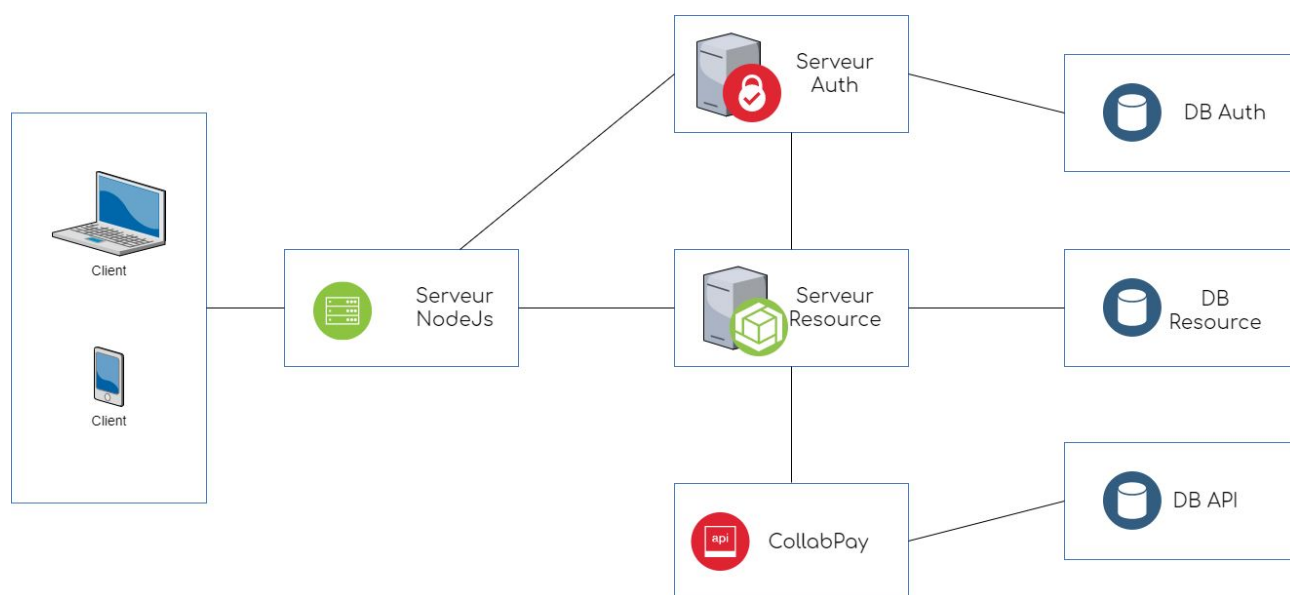


Figure 18

6 Le langage de programmation

Le langage de programmation utilisé influe beaucoup sur notre projet, c'est-à-dire sur la manière dont le projet s'est développé, en fonction des avantages et des inconvénients du langage. Il convient donc de choisir le langage en considérant les besoins de l'entreprise et de notre projet, pour éviter de devoir changer de langage en cours, ce qui constituerait une perte de temps

considérable. De plus, un langage optimisé et facile à apprendre permet d'avoir une meilleure optimisation de la charge du CPU, ce qui a pour conséquence de préserver le matériel et de faciliter la maintenance. Ces avantages sont en cohérence avec l'approche Green IT du projet. Pour choisir un langage de programmation adéquat, il nous paraît donc pertinent de comparer les langages disponibles entre eux en fonction des besoins de l'entreprise et de notre projet.

6.1 Les différents langages de programmation

Nous présentons dans cette partie les critères, la réflexion qui a mené à sélectionner notre langage. Un des critères pour choisir le langage est le suivant: L'application doit être accessible dans un navigateur, ce que nous avons expliqué dans la partie " contraintes particulières de développement" (Pour plus de détails, Cf. Chap.1.3) Comme il existe beaucoup de technologies web, nous devons les comparer. Nous avons alors choisi de limiter notre comparaison aux technologies web les plus connues et les plus utilisées en s'appuyant sur les statistiques de RedMonk.

L'étude RedMonk présente en effet, les vingt meilleurs langages de programmation en se basant sur deux grandes communautés : le nombre de questions posées sur le site Stack Overflow pour chaque langage et le nombre de projets référencés sur GitHub. Ce classement trimestriel permet de repérer les grandes tendances au sein de la profession de développeurs. Nous avons donc analysé l'histogramme présentée ci-dessous (Cf. Figure 19) pour retenir uniquement les langages les plus répandus et les plus connus sur le web en général.

Choisir des langages très utilisés propose de nombreux avantages: Cela permet tout d'abord de bénéficier d'un meilleur support au moment du codage de l'application, et donc de développer une application plus rapidement. Cela permet ensuite d'obtenir un outil plus robuste en suivant les conseils de développeurs plus expérimentés et enfin, cela permet de faciliter la maintenance ou l'évolution du produit par des personnes extérieures au projet. Plus ces personnes auront accès à des ressources variées, plus il leur sera facile de trouver des réponses à leurs problèmes.

RedMonk Language Rankings

September 2012 - January 2020

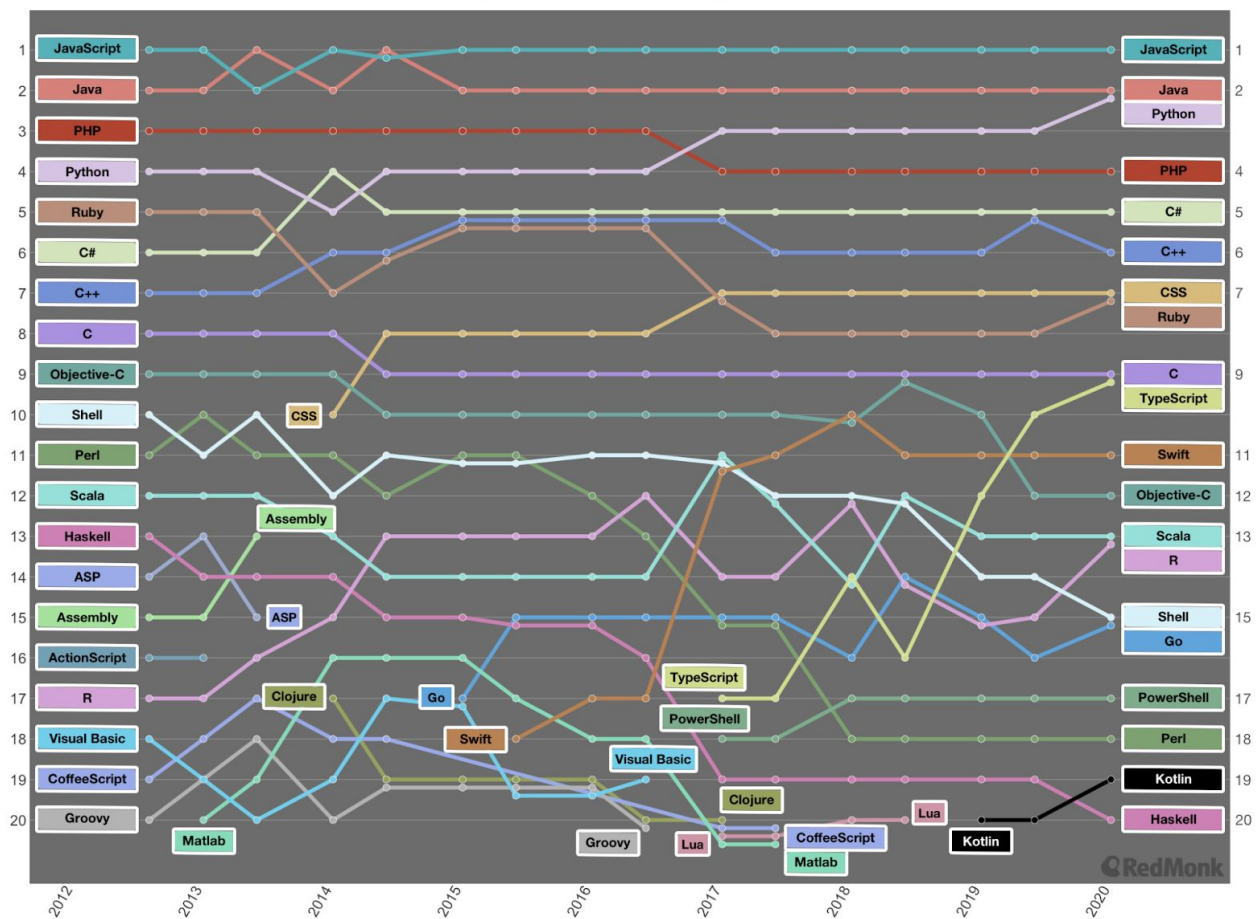


Figure 19

6.2 Comparaison

Nous venons de voir la manière dont nous avons sélectionné les langages les plus utilisés, première étape pour choisir le langage le plus adapté au projet. Nous allons maintenant, présenter et comparer les spécifications des langages les plus utilisés (avantages et inconvénients) afin de sélectionner le langage de notre projet.

6.2.1 Les spécifications du langage PHP

Présentation et avantages:

PHP est un acronyme récursif qui signifie « PHP : Hypertext Processor ». Il s'agit d'un langage de script Open Source très utilisé et spécialement conçu pour le développement web. Il est plus souvent utilisé côté serveur. Le serveur interprète le code PHP et génère la page HTML en conséquence. Sa syntaxe s'inspire du C, du Java et de Perl. Il est peu typé. PHP permet, entre autres, de collecter des données de formulaires, de créer des pages web de manière dynamique ou

d'envoyer / recevoir des cookies. Il peut être utilisé sur les systèmes d'exploitation les plus répandus tel que Linux, Solaris, OpenBSD, Microsoft Windows ou Mac OS X. Il est également présent sur de nombreux serveurs web comme Apache ou IIS. Il permet de programmer de deux manières différentes: En procédural ou en orienté objet. De plus, il est possible d'utiliser des objets Java comme des objets PHP de manière transparente dans une application PHP. Il supporte un grand nombre de bases de données dont MySQL.

Inconvénients:

Le langage PHP a un défaut à nos yeux puisqu'il utilise une exécution multi-thread pour l'exécution des séquences d'instructions multiples. En effet, l'instruction multi-thread parallèle manque de rapidité en ce qui concerne le partage de mémoire en comparaison avec JAVA qui utilise l'exécution thread à thread. C'est pourquoi nous n'avons pas sélectionné ce langage.

6.2.2 Les spécifications du langage JavaScript

Présentation:

JavaScript est un langage de programmation web orienté prototype, contrairement aux autres langages de programmation. Ce paradigme permet, entre autres, de moduler les prototypes à volonté en leur ajoutant des attributs et des méthodes. Il s'agit d'un langage interprété. JavaScript a longtemps été un langage destiné à être téléchargé et exécuté chez le client.

Avantages:

Ces dernières années ont vu la montée de nouvelles API et plateformes telles que Node JS développée par Google, qui permettent d'utiliser la puissance de JavaScript à la fois chez le client et sur le serveur. Ainsi, les développeurs peuvent coder la totalité de leur application dans un seul langage pour permettre une meilleure coordination du client et du serveur. Les frameworks implémentant JavaScript chez le client et sur le serveur exploitent, entre autres, les sockets qui permettent au serveur d'envoyer des informations vers le client sans que celui-ci ait besoin d'envoyer de requête au préalable. Cela permet d'introduire une dimension « temps réel » dans les applications. Cette dimension "temps réel" n'est pas disponible chez les autres langages de programmation. Les interpréteurs JavaScript sont disponibles sous plusieurs systèmes d'exploitation comme Linux, Mac OS X ou Windows.

Inconvénients:

Bien que la technologie JavaScript côté serveur soit très intéressante en terme de possibilités, elle reste néanmoins une technologie assez jeune dans ce domaine et encore peu utilisée au sein des entreprises, on ne connaît donc pas sa fiabilité (stabilité des versions). C'est pourquoi nous n'avons pas sélectionné ce langage.

6.2.3 Les spécifications du langage Python

Présentation et avantages:

Python a été créé par Guido van Rossum et a été rendu public pour la première fois en 1991. Il s'agit d'un langage de programmation interprété, de haut niveau, orienté objet. Tous ces éléments sont les principales raisons pour lesquelles il est devenu l'un des langages de codage les plus populaires au monde.

Les langages de programmation de haut niveau mettent l'accent sur la simplicité avant tout. Toute la philosophie de conception de Python est basée sur la lisibilité. Le langage utilise des espaces et une syntaxe très simple pour atteindre cet objectif. Cela lui permet d'être parmi les langages de programmation les plus faciles à apprendre.

La partie orientée objet est également très importante. La POO, ou "Programmation Orientée Objet", est une philosophie de programmation qui se concentre sur les objets et les données plutôt que sur la logique nécessaire à leur manipulation. La "Programmation Orientée Objet" facilite la maintenance du code et permet de réutiliser le code pour d'autres projets distincts.

Inconvénients:

Python ne propose pas de version compatible avec ses anciennes versions. On pense notamment aux défauts générés lors de la migration de la version 2.7 vers la version 3 où de nombreux bugs sont apparus, où des applications ne fonctionnaient plus, ce qui est coûteux en termes de temps, d'efficacité et d'argent. Si nous nous projetons, cela signifie que le choix de ce langage pour notre application pourrait générer des coûts (temps, argent) en cas de changement de version (Si le changement de version est nécessaire).

6.2.4 Les spécifications du langage Java

Présentation et avantages:

Java est un langage de programmation sécurisé de haut niveau et orienté objet. Il a été créé et développé en 1991 par James Gosling, sur la base de C et C++. Il est conçu pour représenter le slogan WORA qui signifie «Write Once, Run Anywhere» (écrire une fois, exécuter n'importe où). C'est ce slogan qui a fait la renommée de ce langage tout en révélant sa caractéristique dominante. Java est universel, il est conçu pour pouvoir être exécuté sur n'importe quelle plate-forme et avec n'importe quelle application.

De plus l'API (Application Programming Interface) de Java est formée de plusieurs milliers de classes regroupées en paquetages qui portent sur des domaines variés de l'informatique. Ceci est un énorme avantage puisqu'il augmente la

productivité de développement. Il existe aussi énormément d'API tierces de très bonnes qualités, pour des fonctionnalités qui viendraient à manquer au JDK. Java permet de développer des applications d'une façon orientée objet et permet d'avoir une application bien structurée, modulable, maintenable beaucoup plus facilement et efficace. Cela augmente une fois de plus la productivité.

Argumentation en faveur du choix de JAVA:

Java conserve une compatibilité assez bonne avec les anciens codes, compilés avec une version antérieure. Par exemple, si on prend un code Java compilé en Java Byte Code qui date d'il y a huit ans, on peut encore le lancer aujourd'hui. Non seulement, il peut se lancer mais en plus, le programme aura très certainement moins de bugs et sera beaucoup plus performant. Ceci est la garantie de la pérennité d'une application développée en Java. En plus de cet avantage considérable, on remarque que ce langage est en constante amélioration: Chaque nouvelle version apporte des nouveautés au niveau de l'API et des améliorations au niveau des performances et de la stabilité.

Ces deux observations importantes nous engagent à sélectionner ce langage qui est manifestement le plus fiable et le plus performant.

6.2.5 Le langage retenu

Le choix du langage s'est finalement porté sur JAVA pour plusieurs raisons. Il s'agit tout d'abord, d'un langage robuste, portable et très populaire sur le web. Cela permet un meilleur support et une meilleure maintenance. La deuxième raison qui valide notre choix est qu'il s'agit d'un langage éprouvé depuis plusieurs années. JAVA a fait ses preuves, il a démontré qu'il était assez robuste pour répondre aux besoins de l'entreprise et de notre projet. Celle-ci veut effectivement s'appuyer sur des technologies matures et fiables pour fonctionner de manière optimale. La troisième raison est importante pour notre sélection: C'est la compatibilité de la dernière version avec toutes les anciennes versions, ce qui garantit la pérennité du système, contrairement à Python qui a fait défaut lors de la migration de la version 2.7 vers la version 3. Cela signifie effectivement que l'application ne générera pas de coût supplémentaire en cas de changement de version (Si le changement de version est nécessaire). De plus, cela permettra à de futurs développeurs de maintenir ou de faire évoluer rapidement l'application. Enfin, la dernière raison de ce choix est importante également pour notre sélection: C'est le fait que ce langage nous est familier car c'est celui appris durant la formation. Du coup, ce langage permet d'éviter un temps de formation supplémentaire et permet donc de gérer la contrainte de temps imposée. (Pour plus de détails, cf. Chapitre 1.3)

7 Les frameworks

Un Framework n'est pas indispensable pour la création de notre application web. Ceci dit, pour que notre application web soit robuste, facile à faire évoluer et réalisable selon les contraintes de temps imposées, un framework est nécessaire et représente un outil idéal à nos yeux. Nous allons présenter ci-dessous les spécificités et les avantages offerts par les différents frameworks backend choisis pour leur complémentarité, en fonction de nos objectifs et de nos contraintes. Nous allons ensuite présenter les différents frameworks front-end en présentant leurs avantages et leurs inconvénients pour sélectionner le framework front-end le plus pertinent pour notre projet dans un dernier temps.

7.1 Les frameworks backend

Nous présentons ci-dessous les spécificités et les avantages offerts par les différents frameworks backend que nous avons choisi. Ils sont complémentaires.

7.1.1 Spring Boot

Spring Boot est un framework puissant et léger utilisé pour le développement d'applications. Spring est un outil qui prend en charge plusieurs applications web utilisant Java comme langage de programmation. Ces derniers temps, il est devenu le framework le plus populaire dans la communauté Java. Les fonctionnalités de base de Spring peuvent être utilisées par n'importe quelle application Java. De plus, il offre un support à de nombreuses applications comme Spring Security, JPA, Spring Data, Social Integration, Web-services, etc. Spring supporte la POA ou "Programmation Orientée Aspect", un paradigme de programmation permettant d'aller au-delà de l'approche objet en terme de modularisation des composants.

Spring est un conteneur dit «léger», autrement dit, une infrastructure similaire à un serveur d'application JEE. Il prend en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendance entre ces objets.

Le patron de conception "factory" (fabrique) de Spring prend en charge la création d'objet et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendance avec ses objets.

Spring propose également tout un ensemble d'abstractions permettant de gérer entre autres : le mode transactionnel (en spécifiant l'annotation `@Transactional`), la persistance d'objets, la création d'une interface Web, l'appel et la création de WebServices.

7.2.2 Hibernate

Hibernate est une solution open source de type ORM (Object Relational Mapping) qui permet de faciliter le développement de la couche persistance d'une application. Hibernate permet donc de représenter une base de données en objets Java et vice versa.

Hibernate facilite la persistance et la recherche de données dans une base de données en réalisant lui-même la création des objets et les traitements de remplissage de ceux-ci en accédant à la base de données. La quantité de code ainsi épargnée est très importante d'autant que ce code est généralement fastidieux et redondant.

Hibernate est très populaire notamment à cause de ses bonnes performances et de son ouverture à de nombreuses bases de données. Hibernate supporte les principales bases de données du marché et entre autres MySQL et PostgreSQL.

Ce framework facilite beaucoup la conversion des bases de données. De plus, sa capacité à gérer plusieurs bases de données facilite son évolution. Il est performant, facile à mettre à l'échelle, à modifier et à configurer.

7.3.3 Spring Data Jpa

Le projet Spring-Data-JPA est l'un des projets de Spring reposant sur Spring-Data. Spring-Data réduit considérablement le coût économique de la couche d'accès aux données relationnelles ou NoSQL. Il simplifie l'accès aux bases de données.

Avec Spring-Data-JPA, nous pourrions bénéficier d'un certain nombre de méthodes d'accès à la base sans écrire une ligne de code d'implémentation. Cela va améliorer grandement la mise en œuvre de la couche d'accès aux données en réduisant considérablement l'effort d'écriture du code d'implémentation, en particulier pour les méthodes CRUD et de recherche.

La notion centrale dans Spring-Data-JPA est la notion "Repository". Le repository est une interface dans laquelle seront déclarées les méthodes utiles en ce qui concerne l'accès aux données.

7.4.4 Spring Security

Spring Security apporte une solution aux problématiques d'authentification et d'autorisation. Un des nombreux avantages de Spring Security, c'est sa facilité de personnalisation. Spring Security permet également la sécurisation d'une application Spring et permet d'ajouter facilement des autorisations sur des requêtes web, des méthodes ou même des objets.

7.2 Les frameworks front-end

Nous présentons ci-dessous les différents frameworks front-end en présentant leurs avantages et leurs inconvénients afin de sélectionner le framework front-end le plus pertinent pour notre projet.

7.2.1 React

Présentation et avantages:

Créé en 2013 par Facebook, React a été développé pour répondre à des problématiques de performance que rencontrait le site à l'époque. C'est un framework qui est donc léger et optimisé pour gérer un trafic très important. La dernière version en date est la 16.13.1 sortie en mars 2020.

React est un framework minimaliste (qui n'est pas seulement une librairie mais qu'on utilise pour son CLI create-react-app). En effet, React n'a pas vocation à gérer tous les aspects de notre application. Il n'y a pas de gestion des formulaires, ni de gestion de l'état de l'application, ni de gestion du routing, ni de gestion de la validation etc. Il faut à chaque fois, utiliser des librairies indépendantes qui n'ont aucun soutien de la part de Facebook : par exemple Redux pour la gestion d'état, React-Router pour le routing etc.

React est donc particulièrement accessible : le nombre de choses à connaître pour débiter est forcément bien moindre qu'un framework comme Angular qui inclut toutes ces fonctionnalités directement.

React peut être utilisé très facilement sur un grand nombre d'applications, notamment ce qu'on appelle des applications serveurs.

Inconvénients:

Si le framework React propose une grande liberté d'action, en contrepartie, il permet aussi de faire de nombreuses erreurs. Pour un junior, cela signifie qu'il est possible de faire du "spaghetti code". Cela complexifie grandement la maintenance, ce qu'il faut éviter au maximum. C'est pourquoi ce framework est écarté de nos choix.

7.2.2 Vue

Présentation et avantages:

Sorti en 2014, ce framework a été créé par un ancien Googler ayant travaillé plusieurs années avec Angularjs du nom de Evan You. Vue.js se présente comme étant la synthèse entre Angular.js et React. La popularité du framework ne cesse de grandir et il s'est fait une place de choix malgré la compétition féroce. La dernière version est la 2.6.11 sortie le 13 décembre 2019.

Si on compare ce framework avec React, Vue.js permet d'utiliser des composants réactifs et réutilisables avec une syntaxe simple qui est populaire et très simple à

comprendre. Leur point commun est qu'ils utilisent tous les deux de nombreuses optimisations comme le DOM virtuel.

La principale différence entre les deux est que Vue est composé de plusieurs librairies que l'on peut ajouter à notre application au fur et à mesure des besoins. Vue.js est en ce sens un framework progressif. De plus, contrairement à React, c'est la même équipe qui développe tous ces outils, ils sont donc totalement compatibles et bien maintenus.

Inconvénients:

Ce framework est très récent et il est peu utilisé. Cela signifie qu'on ne connaît pas sa fiabilité (stabilité). De plus, il nécessite d'être formé(e), ce qui requiert un coût en termes de temps voire d'argent si on acquiert cette compétence par une formation payante. C'est pourquoi ce framework n'a pas été sélectionné.

7.2.3 Angular

Présentation et avantages:

Angular est officiellement sorti en 2016, c'est donc le framework le plus récent comparé à Vue et React. Ce framework a plusieurs versions. Angular.js la version 1 est totalement abandonnée et n'est plus maintenue par Google. Angular recommande l'utilisation du langage TypeScript. Il sort une nouvelle version majeure tous les 6 mois environ, la version 10 est sortie en juin 2020.

Angular, est un framework dit orienté, autrement dit, Google pense qu'il y a une bonne manière de développer une application Web et organise tous le framework de cette manière. On peut même dire que c'est le framework le plus orienté dans la mesure où il y a toujours une façon recommandée de travailler. Cela apporte un cadre bénéfique pour des juniors ou des apprenants.

Toutes les problématiques du Web sont gérées par le framework lui-même et il n'y a pas besoin de librairies pour les fonctionnalités communes des applications Web (requêtes HTTP, routing, formulaires...), ce qui est plus pratique car tous les outils sont déjà disponibles et ne nécessitent pas de recherches extérieures, ce qui est un gain de temps intéressant.

Les inconvénients:

La contrainte majeure est qu'avec Angular, nous avons peu de libertés sur la façon dont nous allons concevoir l'application, contrairement à Vue ou à React qui laissent une grande liberté dans la conception de l'architecture. L'autre point important, c'est qu'il y a énormément d'éléments à apprendre pour maîtriser ce framework, donc ce framework nécessite un investissement beaucoup plus important.

Nous devons utiliser Typescript, qui est un langage qui compile en JavaScript avec des fonctionnalités supplémentaires et un typage fort - qui permet une meilleure maintenabilité. Nous allons nous familiariser avec RxJS qui est une librairie de programmation réactive difficile à aborder lorsque l'on débute car c'est un paradigme moderne complexe. S'il est plus long et plus difficile de développer en Typescript, le code produit est en revanche de qualité supérieure car plus lisible et plus maintenable, ce qui vaut le coup. De plus, ce langage est le plus soutenu par la communauté professionnelle : Il est développé par de nombreux ingénieurs de Microsoft et bénéficie du soutien de Google qui l'utilise largement. Si des questions apparaissent, des réponses sont donc facilement accessibles.

Pour résumer, le rapport gain- perte est donc en faveur d'un temps consacré à cet apprentissage.

Pour conclure sur les points positifs de ce framework:

Les applications Angular sont globalement toutes architecturées de la même façon. Le framework met en oeuvre et recommande les bonnes pratiques. En quelque sorte, Angular nous impose la meilleure façon d'organiser le code. Ce qui est à notre sens très positif et bénéfique. En comparaison avec React, si une application a été codée par un débutant, les chances sont extrêmement élevées que l'on y trouve du spaghetti code, autrement dit, du code très difficile à maintenir puisque mal organisé. Ce risque est largement amoindri pour Angular.

7.2.4 Framework front-end utilisé

Les frameworks que nous venons de présenter sont ni bons, ni mauvais mais proposent des approches différentes. Ainsi, React est volontairement minimaliste, simple à utiliser et laisse une complète liberté alors que Angular est plus complet, complexe et la "pratique" est orientée. Vue offre une approche intéressante également même s'il est moins utilisé par les développeurs. Malgré cette considération, nous avons fait notre choix en comparant les inconvénients explicités ci-dessus (3).

Pour la conception de l'application *Ceraon*, nous avons donc fait le choix d'utiliser Angular afin de privilégier la qualité du code et sa maintenabilité. Ce choix est soutenu par le fait qu'il y ait une documentation très bien réalisée et facile à comprendre y compris pour un débutant. Enfin, il y a une communauté très importante, ce qui permet d'obtenir facilement des réponses aux problèmes rencontrés.

8 Les systèmes de gestion de bases de données (SGBD)

Une fois le langage et le framework choisis, nous devons nous intéresser à la question de la base de données qui elle aussi est très importante. Dans l'optique d'une optimisation de l'outil, on va choisir le système de gestion de bases de données le plus efficace possible puisque son adéquation avec les besoins du programme impacte directement le temps de développement et la stabilité du système.

8.1 Les types de base de données

8.1.1 Le modèle relationnel

Introduit par E. Codd en 1970, le modèle relationnel s'appuie sur des considérations théoriques formulées sous forme de douze règles formant le cahier des charges initial. Ce modèle est basé sur la notion de relation.

Une relation est un ensemble de n-uplet (n est fixe) qui correspond chacun à une propriété de l'objet à décrire. Ce modèle permet la gestion précise de contraintes d'intégrité qui garantissent la cohérence des données.

Les systèmes de gestion de base de données relationnel (SGBDR) sont interfacés à l'aide d'un langage unique: le SQL (Structured Query Language). Ce langage permet d'effectuer l'ensemble des opérations nécessaires sur la base de données. Ce langage permet également la gestion de transaction. Une transaction est définie par quatre propriétés essentielles: Atomicité, Cohérence, Isolation et Durabilité (ACID). Ces propriétés garantissent l'intégrité des données dans un environnement multi-utilisateurs.

L'Atomicité permet à la transaction d'avoir un comportement indivisible; autrement dit, toutes les modifications sur les données effectuées dans la transaction sont effectives, ou bien, aucune n'est réalisée. On comprend l'intérêt de ce concept dans l'exemple simple d'une transaction débitant un compte A et créditant un autre compte B : Il est clair que la transaction n'a de sens que si les deux opérations sont menées à leur terme. L'atomicité de la transaction garantit que la base de donnée passera d'un état cohérent à un autre état cohérent. La cohérence des données de la base est donc permanente.

L'Isolation des transactions signifie que les modifications effectuées au cours d'une transaction ne sont visibles que par l'utilisateur qui effectue cette transaction. Au cours de la transaction, l'utilisateur pourra voir des modifications en cours qui rendent la base apparemment incohérente, mais ces modifications ne sont pas visibles par les autres et ne le seront qu'à la fin de la transaction si celle-ci est correcte. (Elle ne rend pas la base incohérente).

La Durabilité garantit la stabilité de l'effet d'une transaction dans le temps, même en cas de problèmes graves tel que la perte d'un disque.

8.1.2 Le modèle NoSQL

Les bases de données NoSQL sont souvent présentées comme une solution idéale. Cependant, un certain nombre de points sont à considérer avant de faire le choix d'une solution de stockage non relationnelle. L'une des raisons les plus fréquemment invoquée concerne le gain de performances dû au passage d'une base de données relationnelle à une base de données NoSQL. Dans la pratique, un gain de performance est observé lorsque la charge ou la quantité de données devient trop importante pour une utilisation classique d'un système de stockage relationnel. En général, le gain de performances est alors dû au fait qu'il est beaucoup plus aisé de répartir la charge et les données sur plusieurs serveurs. Il est important de noter qu'avec une application et un design relationnel bien construit, les limites des systèmes relationnels ne sont pas facilement atteintes. Il faut compter plusieurs centaines de requêtes par seconde, avec des ensembles de données contenant soit beaucoup de petites entrées, soit des entrées de taille importante avant d'arriver aux limites d'une base de données relationnelle.

Parmi les raisons principales qui ont mené à la création de ces systèmes de stockage NoSQL, on retrouve notamment :

- La possibilité d'utiliser autre chose qu'un schéma fixe sous forme de tableaux dont toutes les propriétés sont fixées à l'avance ;
- La possibilité d'avoir un système facilement distribué sur plusieurs serveurs et avec lequel un besoin supplémentaire en stockage ou en montée en charge se traduit simplement par l'ajout de nouveaux serveurs.

Néanmoins, aucun système n'est parfait. Ainsi, les principaux inconvénients apportés par les choix de design des NoSQL sont les suivants :

- Le schéma flexible apporte une plus grande liberté au développeur et lui permet de stocker de façon optimale des ensembles de données dont les entrées peuvent être très disparates. Cependant, le langage permettant d'effectuer des requêtes vers le système NoSQL est beaucoup moins riche et la complexité intrinsèque de la requête est déplacée du SQL vers la logique de l'application elle-même ;
- Les systèmes NoSQL les plus populaires ne respectent pas l'ensemble des propriétés A.C.I.D. comme le fait un système relationnel classique. Cela se traduit en pratique par un effort supplémentaire dans certains cas pour s'assurer de la cohérence des données.

8.2 Les SGBD les plus populaires

Il convient de choisir le système de gestion de base de données en fonction du besoin, des contraintes de maintenabilité et des critères de performance. Pour cela, une première étape consiste à étudier la popularité des solutions disponibles grâce au classement proposé sur certains sites de statistiques. Ce qui est proposé ci-dessous, Figure 20, issu du site DB-engines.com

Rank			DBMS	Database Model	Score		
Jul 2020	Jun 2020	Jul 2019			Jul 2020	Jun 2020	Jul 2019
1.	1.	1.	Oracle +	Relational, Multi-model	1340.26	-3.33	+19.00
2.	2.	2.	MySQL +	Relational, Multi-model	1268.51	-9.38	+38.99
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model	1059.72	-7.59	-31.11
4.	4.	4.	PostgreSQL +	Relational, Multi-model	527.00	+4.02	+43.73
5.	5.	5.	MongoDB +	Document, Multi-model	443.48	+6.40	+33.55
6.	6.	6.	IBM Db2 +	Relational, Multi-model	163.17	+1.36	-10.97
7.	7.	7.	Elasticsearch +	Search engine, Multi-model	151.59	+1.90	+2.77
8.	8.	8.	Redis +	Key-value, Multi-model	150.05	+4.40	+5.78
9.	9.	↑ 11.	SQLite +	Relational	127.45	+2.64	+2.82
10.	10.	10.	Cassandra +	Wide column	121.09	+2.08	-5.91

Figure 20, source: <https://db-engines.com/en/ranking>

La figure 20, ci-dessus, représente le classement des dix systèmes de gestion de base de données les plus utilisés. On retrouve ainsi Oracle en tête de liste, suivi de MySQL, Microsoft SQL Server, MongoDB et PostgreSQL. Parmi eux, les trois premiers sont des bases relationnelles, tandis que MongoDB est une base orientée document (NoSQL) et PostgreSQL est une base relationnelle objet.

Maintenant que l'étude de la popularité des solutions proposées est réalisée, dans cette partie, nous allons présenter et comparer ces bases de données pour choisir celle qui correspond aux besoins et aux contraintes de l'application *Ceraon*.

8.2.1 Oracle Database

Oracle Database est un système de gestion de base de données relationnelle objet. Il est développé par Oracle Corporation et a été distribué pour la première fois en 1980. Il est implémenté en C et C++, il est disponible sur la plupart des systèmes d'exploitation (Linux, Solaris, OSX, Windows). Il est également compatible avec une grande variété de langage de programmation, dont Java. Il respecte le principe des transactions ACID.

Nous n'avons pas retenu Oracle Database car cette solution est payante, ce qui est en contradiction avec notre contrainte de coût (Cf. Chap.3.3 pour plus de précisions). Nous souhaitons utiliser des solutions gratuites.

8.2.2 MySQL

MySQL est le deuxième système de gestion le plus populaire d'après le classement réalisé par le site DB engines. Il s'agit d'une base de données relationnelle open-source sous licence GPLv2. Elle est également développée par Oracle Corporation, anciennement par MySQL AB et Sun Microsystems. La première version a été distribuée en 1995. MySQL est implémenté en C et C++. Ce système est disponible sous FreeBSD, Linux, Solaris, OSX et Windows. Il supporte également une grande variété de langages, dont Java et respecte également le principe des transactions ACID.

Nous avons opté pour cette solution car elle est gratuite mais aussi parce qu'elle est open-source, ce qui est très intéressant puisque le code est libre de droit. Il est alors téléchargeable et modulable à souhait.

8.2.3 Microsoft SQL Server

Microsoft SQL Server est un système de gestion de base de données développé par Microsoft et sorti en 1989. Disponible sous licence commerciale, une licence gratuite est également disponible mais est limitée en terme de fonctionnalités. Actuellement dans sa version 2014, Microsoft SQL Server est développé en C++ et supporte moins de langage que Oracle et MySQL. Il respecte lui aussi le principe des transactions ACID.

Nous n'avons pas retenu ce système car la version gratuite est très limitée au niveau des fonctionnalités. Nous préférons nous tourner vers un système qui propose plus de fonctionnalités tout en étant gratuit. Ce qui nous permettrait de respecter notre contrainte de coût en maximisant les potentialités.

8.2.4 MongoDB

MongoDB est un système à part, comparé à ceux présentés ci-dessus. En effet, il s'agit d'une base de données orientée document, faisant partie de la mouvance NoSQL. Cette tendance vise à gérer les bases de données de manière transparente, sans avoir recours, comme son nom l'indique, au SQL classique, dans un souci de simplicité. Ce sont des bases de données prévues pour des applications dites Big Data. Dans MongoDB, ce principe s'applique par la manipulation d'objets au format BSON (JSON Binaire).

Conçu par MongoDB Inc, la première version de MongoDB a été publiée en 2009, ce qui en fait le plus récent des systèmes étudiés. Il est open-source et disponible sous licence AGPL v3. Implémenté en C++, il supporte une grande variété de langage modernes et anciens, dont entre autre Java, et est disponible sous Linux, OSX, Windows et Solaris.

Nous n'avons pas sélectionné ce système puisqu'il n'est pas adapté à notre projet. Notre projet n'a pas des centaines de requêtes par seconde à traiter. De plus, notre schéma de données est fixe. Enfin, les systèmes NoSQL ne respectent pas l'ensemble des propriétés A.C.I.D, nous préférons donc privilégier une cohérence des données la plus optimale possible et ce, sans effort.

8.2.5 PostgreSQL

PostgreSQL est un SGBD de plus en plus populaire depuis quelques années. Il est souvent vu comme l'alternative à MySQL. PostgreSQL est une base de données relationnelle assez particulière car orientée objet. Elle possède des extensions objet permettant de définir des fonctions en base de données et d'appliquer les principes d'héritage. Ce type de base de données est très utile

lorsque les données stockées sont très complexes et que le stockage des relations objets tels que l'héritage présentent un intérêt certain. Sa première version est sortie en 1989, sous licence BSD. Il est implémenté en C, supporte moins de langages que les autres. Pour chacun des langages usuels, ou avancés, PostgreSQL dispose d'une interface la plus souvent proposée sous la forme d'un pilote du moteur de base de données. C'est le cas notamment, pour PHP, Ruby, Java, Perl, Python, ou C et C++, mais aussi pour de très nombreux autres langages.

Nous ne sélectionnons pas ce système car notre schéma de données est relativement simple, sans héritage entre les entités. Les avantages de ce système ne sont pas pertinentes pour notre projet.

8.3 Le SGBD retenu

Nous venons de comparer plusieurs solutions. Nos critères de sélection sont les suivantes: Tout d'abord, opter pour une base de données relationnelle, ce qui est beaucoup plus adapté pour notre projet (Schéma fixe, aucunes données complexes, pour plus d'informations, cf. iii). Puis, choisir une solution dont la popularité est grande grâce au classement de juillet 2020 issu de DB-engines.com, le dernier critère concerne nos contraintes. La première contrainte est le coût, la solution doit être gratuite et la deuxième contrainte est la volumétrie, la solution doit être performante pour qu'elle puisse supporter plusieurs connexions en même temps. C'est donc naturellement MySQL qui a retenu notre attention, il répond à tous nos critères. (Pour plus de précisions, voir B- MySQL)

9 Modélisation de la base de données

On peut visualiser à la page suivante,(Cf. figure 21) le script de la base de données généré automatiquement par Workbench et corrigé par nos soins.

Quelques explications concernant le choix d'encodage sont nécessaires:

- ENGINE = InnoDB

InnoDB est le moteur de stockage par défaut de MySQL depuis la version 5.5.52. Son principal avantage est qu'il permet des transactions ACID (atomiques, cohérentes, isolées et durables), ainsi que la gestion des clés étrangères (avec vérification de la cohérence).

- DEFAULT CHARACTER SET = utf8mb4

UTF-8 est un codage à longueur variable, cela signifie que l'enregistrement d'un point de code nécessite un à quatre octets. Cependant, le codage de MySQL "utf8" ne stocke qu'un maximum de trois octets par point de code. Ainsi, le jeu de

caractères "utf8"/"utf8mb3" ne peut pas stocker tous les points de code Unicode: il ne prend en charge que la plage 0x000 à 0xFFFF, appelée " plan multilingue de base ".

- COLLATE = utf8mb4_0900_ai_ci

Une collation est un attribut d'un littéral et permet notamment de spécifier l'ordre de classement des lettres conforme au dictionnaire spécifique à une langue, si l'on doit tenir compte de la casse (et donc faire la distinction entre majuscules et minuscules) ou non, si l'on doit tenir compte des éléments diacritique d'une lettre (et donc faire la distinction entre les lettres avec et sans accents) ou non, etc.. Les suffixes ci et ai signifient respectivement la sensibilité ou non à la casse (case insensitive/sensitive) et la sensibilité aux accents (accent insensitive/sensitive).

Nous avons utilisé le charset utf8mp4 car l'application *Ceraon* est certes, conçue en anglais mais nous souhaitons permettre le stockage de caractères de langue, des emoji et des symboles puisque nous envisageons qu'elle soit disponible en français dans une prochaine version. (Une version avec d'autres langues est aussi envisagée pour un plus grand confort des collaborateurs étrangers présents sur le site).

```

1
2
3  -- Schema ceraon
4
5  CREATE SCHEMA IF NOT EXISTS `ceraon` DEFAULT CHARACTER SET utf8mb4 COLLATE
6  utf8mb4_0900_ai_ci ;
7  USE `ceraon` ;
8
9  -- Table `ceraon`.`accounts`
10
11  CREATE TABLE IF NOT EXISTS `ceraon`.`accounts` (
12    `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
13    `avatar_url` VARCHAR(255) NOT NULL,
14    `birth_date` DATE NULL DEFAULT NULL,
15    `creation_date` DATETIME NULL DEFAULT NULL,
16    `email` VARCHAR(200) NOT NULL,
17    `firstname` VARCHAR(50) NULL DEFAULT NULL,
18    `gender` ENUM('M', 'F') NULL DEFAULT NULL,
19    `lastname` VARCHAR(50) NULL DEFAULT NULL,
20    `phone` VARCHAR(20) NULL DEFAULT NULL,
21    `username` VARCHAR(30) NOT NULL,
22    PRIMARY KEY (`id`),
23    UNIQUE INDEX `accounts_username_UNIQUE` (`username` ASC),
24    UNIQUE INDEX `accounts_email_UNIQUE` (`email` ASC))
25  ENGINE = InnoDB
26  DEFAULT CHARACTER SET = utf8mb4
27  COLLATE = utf8mb4_0900_ai_ci;
28
29  -- Table `ceraon`.`categories`
30
31  CREATE TABLE IF NOT EXISTS `ceraon`.`categories` (
32    `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
33    `picture` VARCHAR(255) NULL DEFAULT NULL,
34    `name` VARCHAR(50) NOT NULL,
35    PRIMARY KEY (`id`),
36    UNIQUE INDEX `categories_name_UNIQUE` (`name` ASC))
37  ENGINE = InnoDB
38  DEFAULT CHARACTER SET = utf8mb4
39  COLLATE = utf8mb4_0900_ai_ci;
40
41  -- Table `ceraon`.`orders`
42
43  CREATE TABLE IF NOT EXISTS `ceraon`.`orders` (
44    `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
45    `creation_date` DATETIME NOT NULL,
46    `grand_total` DECIMAL(10,2) UNSIGNED NOT NULL,
47    `is_take_away` VARCHAR(1) NOT NULL,
48    `modification_date` DATETIME NULL DEFAULT NULL,
49    `reference` BINARY(16) NOT NULL,
50    `status` ENUM('CART', 'OPEN', 'TERMINATED', 'PAYED', 'CLOSE') NOT NULL DEFAULT
51    'CART',
52    `tax_total` DECIMAL(10,2) UNSIGNED NOT NULL,
53    `total_items` DECIMAL(10,2) UNSIGNED NOT NULL,
54    `account_id` BIGINT(20) UNSIGNED NOT NULL,
55    PRIMARY KEY (`id`),
56    UNIQUE INDEX `orders_reference_UNIQUE` (`reference` ASC),
57    INDEX `orders_account_id_IDX` (`account_id` ASC),
58    CONSTRAINT `orders_account_id_FK`
59      FOREIGN KEY (`account_id`)
60      REFERENCES `ceraon`.`accounts` (`id`))
61  ENGINE = InnoDB
62  DEFAULT CHARACTER SET = utf8mb4
63  COLLATE = utf8mb4_0900_ai_ci;
64
65  -- Table `ceraon`.`products`
66
67  CREATE TABLE IF NOT EXISTS `ceraon`.`products` (
68    `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
69
70
71

```



```

72     `creation_date` DATETIME NOT NULL,
73     `description` VARCHAR(50) NOT NULL,
74     `hidden` ENUM('T', 'F') NOT NULL DEFAULT 'F',
75     `image_url` VARCHAR(255) NULL DEFAULT NULL,
76     `modification_date` DATETIME NULL DEFAULT NULL,
77     `name` VARCHAR(255) NOT NULL,
78     `price` DECIMAL(3,1) UNSIGNED NOT NULL,
79     `quantity` SMALLINT(3) UNSIGNED NOT NULL,
80     `reference` INT(5) UNSIGNED NOT NULL,
81     `tax_value` DECIMAL(3,1) UNSIGNED NOT NULL,
82     `category_id` BIGINT(20) UNSIGNED NOT NULL,
83     PRIMARY KEY (`id`),
84     UNIQUE INDEX `products_reference_UNIQUE` (`reference` ASC),
85     INDEX `products_category_id_IDX` (`category_id` ASC),
86     CONSTRAINT `products_category_id_FK`
87     FOREIGN KEY (`category_id`)
88     REFERENCES `ceraon`.`categories` (`id`))
89 ENGINE = InnoDB
90 DEFAULT CHARACTER SET = utf8mb4
91 COLLATE = utf8mb4_0900_ai_ci;
92
93
94 -----
95 -- Table `ceraon`.`items`
96 -----
97 CREATE TABLE IF NOT EXISTS `ceraon`.`items` (
98     `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
99     `quantity` SMALLINT(5) UNSIGNED NOT NULL,
100    `total_price` DECIMAL(10,2) UNSIGNED NOT NULL,
101    `unit_price` DECIMAL(10,2) UNSIGNED NOT NULL,
102    `order_id` BIGINT(20) UNSIGNED NOT NULL,
103    `product_id` BIGINT(20) UNSIGNED NOT NULL,
104    PRIMARY KEY (`id`),
105    UNIQUE INDEX `order_id_product_id_unit_price_UNIQUE` (`order_id` ASC, `product_id`
106    ASC, `unit_price` ASC),
107    INDEX `items_product_id_FK` (`product_id` ASC),
108    CONSTRAINT `items_order_id_FK`
109    FOREIGN KEY (`order_id`)
110    REFERENCES `ceraon`.`orders` (`id`),
111    CONSTRAINT `items_product_id_FK`
112    FOREIGN KEY (`product_id`)
113    REFERENCES `ceraon`.`products` (`id`))
114 ENGINE = InnoDB
115 DEFAULT CHARACTER SET = utf8mb4
116 COLLATE = utf8mb4_0900_ai_ci;
117
118 -----
119 -- Table `ceraon`.`orders_books`
120 -----
121 CREATE TABLE IF NOT EXISTS `ceraon`.`orders_books` (
122     `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,
123     `deliver_date_time` DATETIME NULL DEFAULT NULL,
124     `delivered_time` BIGINT(20) UNSIGNED NULL DEFAULT NULL,
125     `order_date_time` DATETIME NOT NULL,
126     `status` ENUM('IN_PROGRESS', 'DELIVERED', 'CANCELLED') NOT NULL DEFAULT
127     'IN_PROGRESS',
128     `order_id` BIGINT(20) UNSIGNED NOT NULL,
129     PRIMARY KEY (`id`),
130     INDEX `orders_books_order_id_FK` (`order_id` ASC),
131     CONSTRAINT `orders_books_order_id_FK`
132     FOREIGN KEY (`order_id`)
133     REFERENCES `ceraon`.`orders` (`id`))
134 ENGINE = InnoDB
135 DEFAULT CHARACTER SET = utf8mb4
136 COLLATE = utf8mb4_0900_ai_ci;
137
138 -----
139 -- Table `ceraon`.`order_book_items`
140 -----
141 CREATE TABLE IF NOT EXISTS `ceraon`.`order_book_items` (
142     `id` BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT,

```



```

143     `done` VARCHAR(1) NOT NULL,
144     `quantity` SMALLINT(5) UNSIGNED NOT NULL,
145     `product_id` BIGINT(20) UNSIGNED NOT NULL,
146     `order_book_id` BIGINT(20) UNSIGNED NOT NULL,
147     PRIMARY KEY (`id`),
148     INDEX `order_book_items_product_id_FK` (`product_id` ASC),
149     INDEX `order_book_items_order_book_id_FK` (`order_book_id` ASC),
150     CONSTRAINT `order_book_items_order_book_id_FK`
151         FOREIGN KEY (`order_book_id`)
152         REFERENCES `ceraon`.`orders_books` (`id`),
153     CONSTRAINT `order_book_items_product_id_FK`
154         FOREIGN KEY (`product_id`)
155         REFERENCES `ceraon`.`products` (`id`))
156 ENGINE = InnoDB
157 DEFAULT CHARACTER SET = utf8mb4
158 COLLATE = utf8mb4_0900_ai_ci;
159

```

Figure 21

La figure 22, ci-dessous, présente le schéma de la base de données générée par Workbench à partir du script (figure 21).

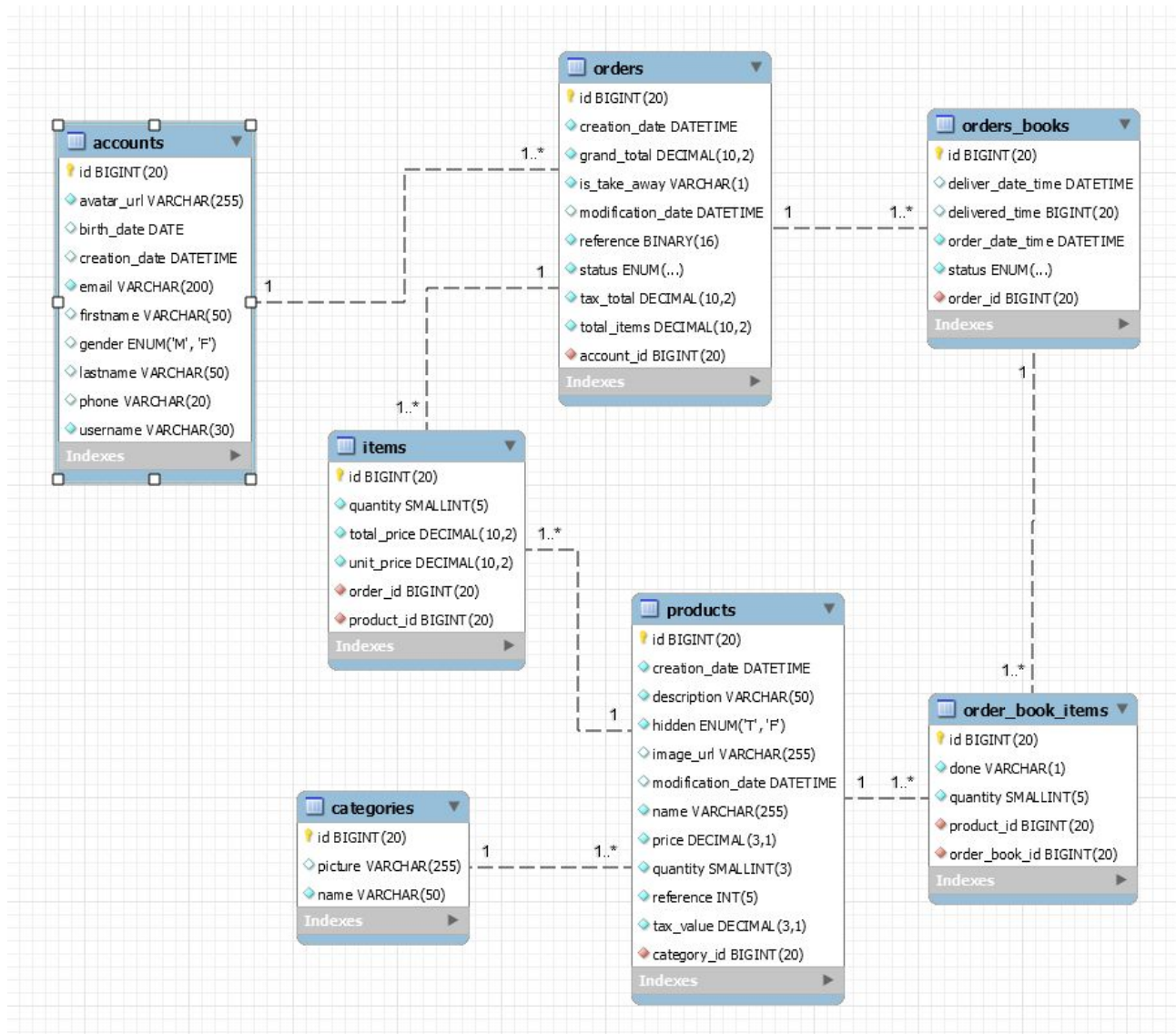


Figure 22

10 Sécurité

Le framework d'autorisation OAuth 2.0 permet aux systèmes d'interagir dans un contexte sécuritaire optimal. Il est incontournable dans le domaine du web et utilisé notamment par Facebook et Google. Nous allons dans cette partie, présenter les détails technique liés à OAuth2 pour comprendre cet outil qui sert à implémenter un serveur d'autorisation pour sécuriser les API du projet et un serveur de ressources protégées.

OAuth 2.0 a été conçu pour être utilisé avec une connexion sécurisée : le protocole HTTPS. OAuth 2.0 se différencie en partie de son prédécesseur OAuth1 par le fait que l'aspect sécuritaire lié à l'intégrité et la confidentialité des données transmises pendant la demande d'autorisation est délégué au protocole TLS.

Le schéma classique d'un processus d'authentification et d'autorisation fait intervenir deux parties : un serveur en mesure d'authentifier et d'autoriser l'accès à une ressource et un utilisateur qui fournit ses identifiants pour accéder à la ressource. La ressource désignant toutes les informations exposées par le serveur et appartenant à l'utilisateur.

Pour OAuth, le processus d'autorisation est un peu plus complexe et peut faire intervenir jusqu'à quatre acteurs :

- Le propriétaire de la ressource qui est une entité (par exemple un utilisateur) en mesure de donner l'accès à une ressource protégée.
- Le client qui demande l'accès à la ressource au nom de son propriétaire. Le terme client peut désigner aussi bien une application mobile qu'un serveur web. Autrement dit, il permet d'identifier l'entité qui souhaite accéder à une ressource.
- Le serveur de la ressource qui héberge les ressources protégées.
- Le serveur d'autorisation qui délivre le droit d'accès à la ressource protégée au client après avoir authentifié le propriétaire de la ressource.

L'enregistrement nécessite au moins trois informations : l'identifiant du client, le mot de passe ou paire de clés (publique/privée) pour les clients confidentiels et une ou plusieurs URL de redirection.

La demande d'accès à une ressource protégée se traduit par la délivrance d'un token au client. Le token représente une chaîne de caractère unique permettant d'identifier le client et les différentes informations utiles durant le processus d'autorisation. Ce token va donc transiter dans chaque requête entre le client et le serveur. De plus, nous pouvons générer un token différent pour chaque requête afin de se protéger contre les attaques de type CSRF (Cross-Site Request Forgery) à savoir l'usurpation d'identité.

Nous pouvons distinguer deux types de token :

- Le token d'accès ou Access Token qui permet au client d'accéder à la ressource protégée. Ce token a une durée de validité limitée et peut avoir une portée limitée. Cette notion de portée permet d'accorder un accès limité au client. Ainsi, un utilisateur peut autoriser un client à accéder à ses ressources qu'en lecture seule.
- Le token de rafraîchissement ou Refresh token permet au client d'obtenir un nouveau token d'accès une fois que celui-ci a expiré. Sa durée de validité est beaucoup plus élevée que celle du token d'accès même si elle est limitée. Son utilisation permet au client d'obtenir un nouveau token d'accès sans l'intervention du propriétaire de la ressource protégée.

En résumé, OAuth 2.0 formalise un ensemble de mécanismes permettant à une application cliente d'accéder à une ressource protégée au nom de son propriétaire (resource owner) ou en son propre nom. Cette autorisation se traduit par la délivrance d'un token d'accès (et éventuellement d'un token de rafraîchissement) qui permet au client de dialoguer avec le serveur hébergeant les ressources protégées (serveur de ressource).

11 WebSocket

Une des fonctionnalités de l'application est de prévenir l'utilisateur lorsque sa commande est prête et de prévenir les agents lorsqu'un utilisateur envoie une nouvelle commande. Les messages sont envoyés automatiquement depuis le serveur ressource lorsque ces actions sont réalisées.

HTTP est le protocole standard utilisé pour le Web mais il n'est pas adapté à une utilisation de type interactive car il repose sur le modèle requête/réponse. Le client envoie une requête au serveur qui répond en lui renvoyant une réponse. Le client doit attendre la réponse. La transmission de données ne peut se faire que dans une direction en même temps. Pour contourner cette limitation, nous avons songé à utiliser le polling. Le client effectuerait alors des requêtes synchrones au serveur périodiquement pour obtenir des notifications. Cette technique est simple mais peu efficace car elle nécessite beaucoup de connexions selon la fréquence utilisée. Cette technique pourrait être intéressante si les données étaient périodiquement modifiées côté serveur, il aurait suffi alors de synchroniser les requêtes sur les notifications. Dans notre cas, cela générerait beaucoup d'appels inutiles.

Le protocole WebSocket est beaucoup plus adapté à notre cas de figure. Celui-ci permet d'amorcer un canal de communication bidirectionnel entre une page web et un serveur via un socket TCP. L'avantage d'une telle connexion est de permettre au serveur d'émettre des notifications vers le client Web sans recevoir au préalable une requête de la part du client.

La figure 23, ci-dessous, présente la structure de l'implémentation souhaitée pour ce projet. Le Message Broker est un programme intermédiaire. Il reçoit les messages envoyés avant la distribution aux adresses nécessaires, deux canaux seront ouverts à la communication dans l'application.

Un premier canal <<Action>> s'ouvre à chaque connexion d'un membre de l'équipe du point de vente, les agents ou les managers. Ce canal leur permet de recevoir tous les messages venant des utilisateurs. Les messages sont envoyés automatiquement lorsqu'une action est déclenché depuis le serveur. Par exemple, lorsqu'un client confirme une commande un message est envoyé via ce canal pour avertir les agents qu'une nouvelle commande a été créé. Tous les membres de l'équipe pourront voir ce message.

Le deuxième canal <queue/username> est un canal qui s'ouvre à chaque connexion d'un utilisateur. Celui-ci est le seul à pouvoir voir les message qui sont envoyés. Pour permettre cela, un chemin d'accès contenant le username de l'utilisateur est créé. Lorsqu'une commande de cet utilisateur est terminé, un message est envoyé automatiquement depuis le serveur.

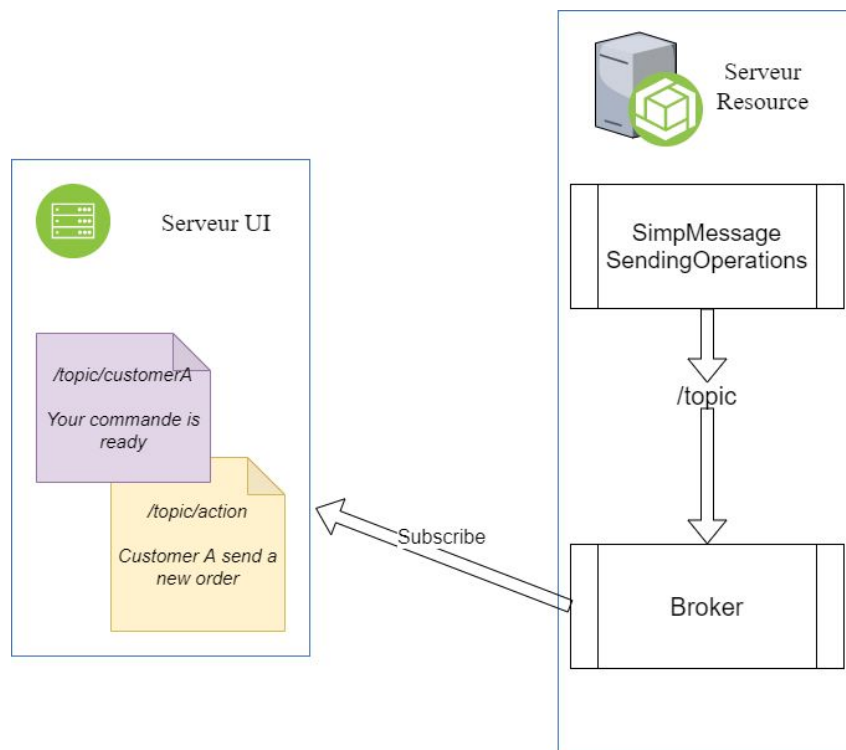


Figure 23

Le protocole est utilisé uniquement (pour le moment) dans le sens du serveur vers le client. Dans une future version, les 2 sens pourront être utilisés pour permettre aux clients d'envoyer des messages aux agents directement depuis leurs smartphones.

CHAPITRE 4: Réalisation

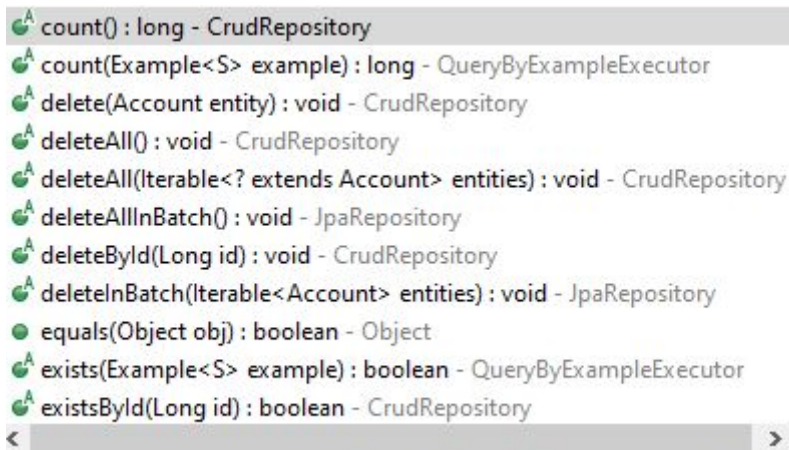
La phase de réalisation est essentiellement centrée sur l'implémentation de la première version de l'application, nous allons présenter les difficultés rencontrées et les solutions trouvées aux problèmes posés.

1 Arborescence du projet

La figure 25, ci-dessous, représente l'ensemble des dossiers de classes du serveur de ressource de l'application *Ceraon*. L'application est conçue en couches.

Quelques explications de l'arborescence du projet illustrée par la figure 25 sont nécessaires:

- Le package "configuration" comprend toutes les classes de configuration, notamment pour la sécurité, les websockets, le swagger ...
- Le package "controllers" contient les classes annotées `@RestController`. Cette annotation marque la classe en tant que contrôleur où chaque méthode renvoie à un objet de domaine au lieu d'une vue. Cette annotation est la combinaison de `@Controller` et de `@ResponseBody`. Les contrôleurs contiennent les endpoints des API et vont exposer un service.
- Le package "services" contient les interfaces service et leurs implémentations respectives. Les classes d'implémentation sont annotées `@Service`. Les beans `@Service` contiennent la logique métier.
- Le package "repositories" contient les interfaces annotées `@Repository`. Ces interfaces étendent toutes `JpaRepository`. Avec cet héritage de `JpaRepository`, nous disposons ainsi de plein de méthodes CRUD et de recherche standard comme illustré ci-dessous (figure 24) :



```
count() : long - CrudRepository
count(Example<S> example) : long - QueryByExampleExecutor
delete(Account entity) : void - CrudRepository
deleteAll() : void - CrudRepository
deleteAll(Iterable<? extends Account> entities) : void - CrudRepository
deleteAllInBatch() : void - JpaRepository
deleteById(Long id) : void - CrudRepository
deleteInBatch(Iterable<Account> entities) : void - JpaRepository
equals(Object obj) : boolean - Object
exists(Example<S> example) : boolean - QueryByExampleExecutor
existsById(Long id) : boolean - CrudRepository
```

Figure 24

- Le package "entities" contient les classes annotées `@Entity`. Cette annotation nous indique que cette classe est une classe persistante. Ces classes sont également annotées `@Table`, ce qui permet de fixer le nom de la table dans laquelle les instances de cette classe vont être écrites.

- Le package "dtos" contient les classes de DTO (Data Transfert Object). Les dtos vont permettront de réduire la quantité de données qui doit être transmise.

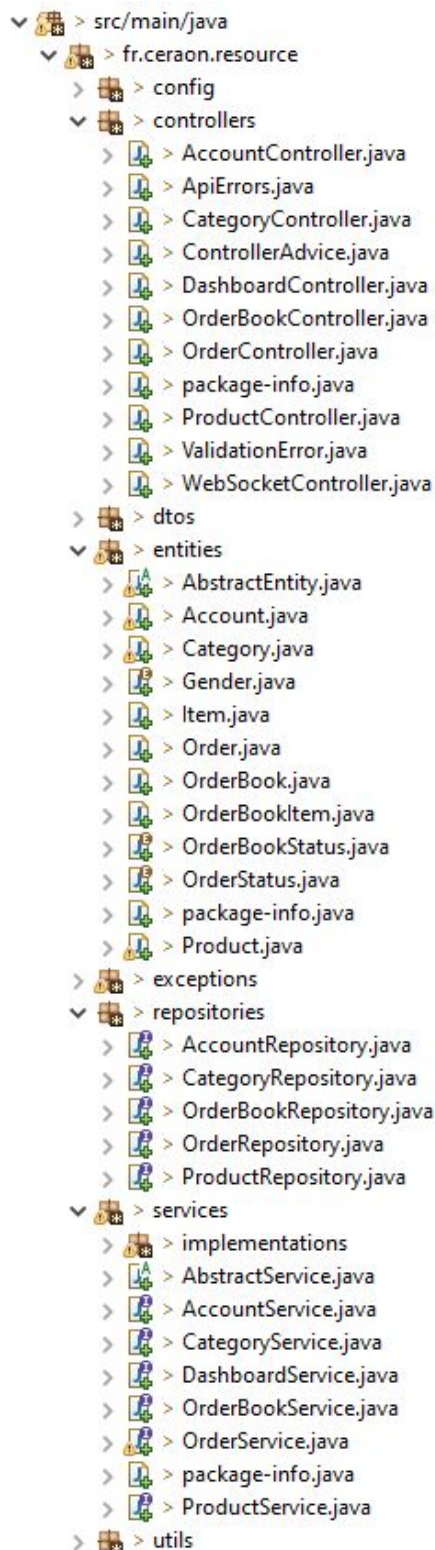


Figure 25

L'arborescence de la partie front-end est présenté dans la figure 26, ci-dessous :

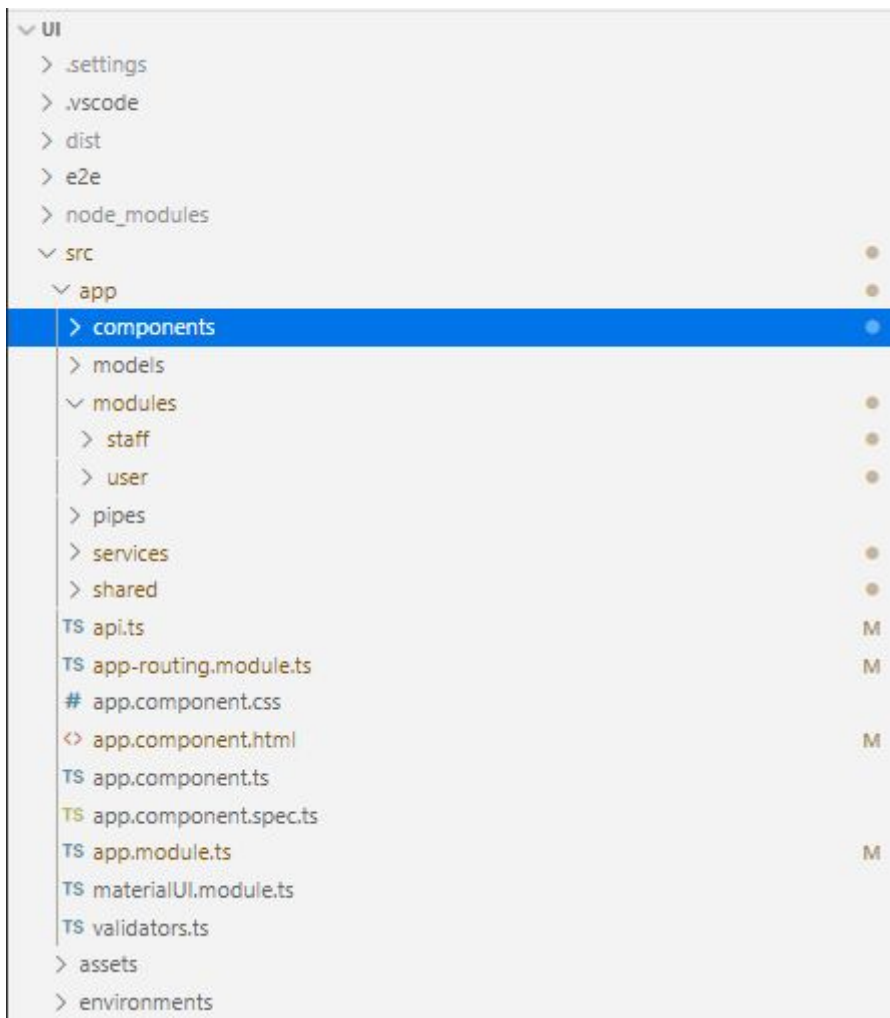


Figure 26

Quelques explications de l'arborescence de la partie front-end sont nécessaires:

- Le dossier "composants" comprend tous les composants utilisés dans l'ensemble de l'application, c'est à dire les composants login, header, footer ...
- Le dossier "shared" comprend les composants qui doivent être importés dans tous les modules, comme par exemple le composant pour afficher une icône. Le module shared va également importer des modules comme celui qui contient tous les composants Material Design pour Angular (MaterialUIModule), mais aussi le module qui contient toutes les Pipes (PipeModule).
- Le dossier services contient tous les services permettant d'appeler les apis. Chaque service injecte le service HttpClient qui contient les méthodes nécessaires pour effectuer des requêtes HTTP.
- Le dossier "models" comprend les modèles des objets utilisés pour pouvoir typer les requêtes, les réponses .. Le typage est très important pour détecter les erreurs pendant la compilation de l'application.
- Le dossier "modules" comprend les modules. Nous avons créé un module pour la partie "staff" et un module pour la partie "user". A chaque

authentification, le module correspondant au profil de l'utilisateur est loadé, ainsi que les éléments, comme le tableau de bord par exemple, qui ne sont accessible qu'à un type d'utilisateur. Ils ne sont pas loadés au lancement de l'application. Cela améliore la performance globale.

2 Prise de commande

La figure 27, ci-dessous, présente les écrans d'affichages du menu et du panier de la version mobile. Dans la version desktop, le panier est toujours visible à droite des produits pour que l'utilisateur puisse voir tous les ajouts et les changements réalisés. Dans la partie mobile, le panier est caché dans le menu. Il est accessible via un bouton icône. Cette icône affiche un badge contenant le nombre d'articles présents. Il est désactivé si le panier est vide pour ne pas surcharger la vue sur mobile.

La sidebar utilisée pour l'affichage du panier est la même que celle utilisée pour le menu de navigation. La croix permet de fermer la sidebar pour revenir sur la page. Le bouton "go to checkout" permet d'accéder à la page de validation de la commande. Si le panier d'achat a été modifié, il sera alors persister en base.

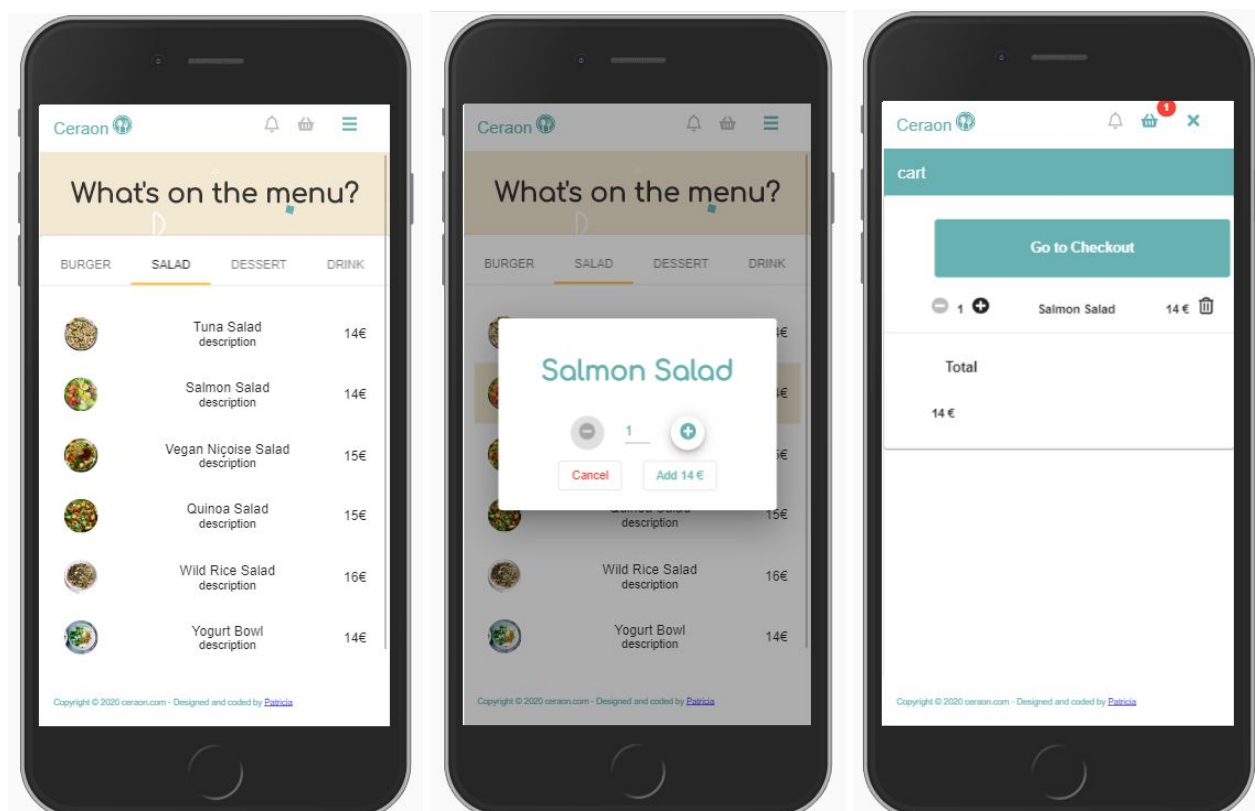


Figure 27

Pour réaliser ces différents écrans, la difficulté fut de déterminer comment afficher le panier sous la forme d'une icône dans le header, et sous la forme

d'une liste dans la sidebar. Dans le composant CartComponent, un Input permet de spécifier la vue à afficher.

Les Inputs et les Outputs sont une manière pour un composant de communiquer avec son parent : Les Inputs permettent à un composant de recevoir des données de son parent. Les Outputs permettent à un composant de transmettre un événement ou des données à son parent. Les Inputs/Outputs sont donc une manière de rendre un composant plus générique et plus réutilisable, puisqu'il peut recevoir des données de l'extérieur (Inputs) et renvoyer des données (Outputs).

Ainsi le même composant permet l'affichage dans le header et dans la sidebar mais également pour la partie desktop dans l'écran du menu. La figure 28, ci-dessous, présente le fichier TypeScript du composant angular.

```
6
7  @Component({
8    selector: 'app-cart',
9    templateUrl: './cart.component.html',
10   styleUrls: ['./cart.component.css'],
11 })
12 export class CartComponent implements OnInit {
13
14   @Input() isMobile: boolean;
15   @Input() view: string;
16
17   @Output() closeSidenav = new EventEmitter();
18
19   cart$: Observable<Cart>;
20   cartItemCount$: Observable<number>;
21
22   constructor(private cartService: CartService) {}
23
```

Figure 28

La figure 29, ci-dessous, représente le code html de ce même composant. L'affichage de la vue est gérée par une condition en fonction d'un Input.

L'avantage d'utiliser le même composant est la gestion des données et des Observables.

Un Observable est un objet qui émet des informations auxquelles on souhaite souscrire. Ces informations peuvent venir d'un champ de texte dans lequel l'utilisateur rentre des données, ou de la progression d'un chargement de fichier, par exemple. Elles peuvent également venir de la communication avec un serveur : le client HTTP emploie les Observables.

Pour cet exemple, nous devons afficher la liste des produits ajoutés au panier mais également le nombre de produits pour le badge de l'icône. Le nombre de produits est obtenu par somme des produits de la quantité et du prix. Cette valeur pouvant être nécessaire dans d'autres composants de l'application, nous avons créé un Pipe pour transformer la liste des produits en leur valeur totale.

Un Pipe est un élément du framework Angular permettant d'effectuer des transformations directement dans le template. Ils sont notamment utilisés pour mettre en forme une date ou un nombre mais aussi pour effectuer des filtres.

```
@Pipe({
  name: 'itemsAmount',
  pure: false
})
export class ItemsAmountPipe implements PipeTransform {

  transform(value: Item[]): number {

    const amount: number = value
      ? value.reduce((total, item) => total += (item.product.price * item.quantity), 0)
      : 0;

    return amount;
  }
}
```

Figure 29

La figure 29, ci-dessus, présente le code du Pipe customisé "itemsAmount", qui prend une liste de produits en entrée et qui retourne (return) à l'aide de sa fonction "transform" le montant total de cette liste.

La figure 30 présente le code du composant CartComponant. Tous les Observables sont récupérées depuis CartService une seule fois pour tous les affichages. Tous les affichages reçoivent les modifications de la souscription aux Observables en même temps.

Le Pipe async affiche les Observables en appelant implicitement la méthode subscribe afin de "bind" les valeurs contenus dans l'Observable. Ce Pipe unsubscribe automatiquement à la destruction de la vue. Ainsi, son utilisation permet de ne pas avoir à gérer la subscription.

De plus la condition `*ngIf="{cart: cart$|async, count: cartItemAmount$|async} as obs"` retourne toujours true. La vue est toujours affichée même si une des Observables n'est pas encore disponible. Elle permet de les charger une fois en appelant la méthode subscribe. Cela évite de requêter le serveur plusieurs fois.

```

src > app > modules > user > cart > src > cart.component.ts > @ isicon
1 import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
2 import { Observable } from 'rxjs';
3 import 'rxjs/add/operator/takeUntil';
4 import { Cart, Item } from '../models/order';
5 import { CartService } from '../../services/cart.service';
6
7 @Component({
8   selector: 'app-cart',
9   templateUrl: './cart.component.html',
10   styleUrls: ['./cart.component.css'],
11 })
12 export class CartComponent implements OnInit {
13
14   @Input() isMobile: boolean;
15   @Input() view: string;
16
17   @Output() closeSidenav = new EventEmitter();
18
19   carts: Observable<Cart>;
20   cartItemCounts: Observable<number>;
21
22   constructor(private cartService: CartService) {}
23
24   ngOnInit() {
25     this.cartService.getCartOpen().take(1).subscribe(() => {});
26     this.carts = this.cartService.currentCarts;
27     this.cartItemCounts = this.cartService.currentCartItemCounts;
28   }
29
30   delete(item: Item) {
31     this.cartService.remove(item);
32   }
33
34   update(item: Item, n: number) {
35     this.cartService.updateQuantity(item, n);
36   }
37
38   goToCheckout() {
39     this.cartService.goToCheckout();
40   }
41
42   isIcon() {
43     return this.view === 'icon';
44   }
45 }
46
47
src > app > modules > user > cart > src > cart.component.html > @ ng-container
1 <ng-container *ngIf="(cart | async, count: cartItemCounts | async) as obs">
2   <span *ngIf="isIcon()">
3     <div class="container basket">
4       <mat-card>
5         <mat-card-header class="header">
6           <button mat-flat-button color="primary" class="header-btn" (click)="goToCheckout(); this.closeSidenav.emit();"
7             [disabled]="obs.count <= 0">
8             <h3> Go to Checkout </h3>
9           </button>
10         </mat-card-header>
11         <mat-card-content>
12           <div *ngIf="obs.count <= 0; else userCart" fxLayout="row" fxLayoutAlign="center center">
13             <h3 mat-subheader>Your basket is empty.</h3>
14           </div>
15           <ng-template #userCart>
16             <ul class="basket-content">
17               <li *ngFor="let item of obs.cart?.items" fxLayout="row" fxLayoutAlign="center center"
18                 class="basket-item">
19                 <span class="quantity" fxFlex="20%" fxLayout="row" fxLayoutAlign="center center">
20                   <button mat-icon-button (click)="update(item, -1)" [disabled]="item.quantity<1">
21                     <app-icon [name]="minus_circle"/></app-icon>
22                   </button>
23                   <h4 mat-line>{{item.quantity}}</h4>
24                   <button mat-icon-button (click)="update(item, 1)">
25                     <app-icon [name]="plus_circle"/></app-icon>
26                   </button>
27                 </span>
28                 <span class="name" fxFlex="60%" fxLayoutAlign="center center">
29                   <h4 mat-line>{{item.product.name}}</h4>
30                 </span>
31                 <span class="amount" fxFlex="10%" fxLayoutAlign="center center">
32                   <h4 mat-line>{{item.product.price*item.quantity}} </h4>
33                 </span>
34                 <span class="trash" fxFlex="10%" fxLayoutAlign="center center">
35                   <button mat-icon-button (click)="delete(item)">
36                     <app-icon [name]="trash"/></app-icon>
37                   </button>
38                 </span>
39               </li>
40             <mat-divider></mat-divider>
41             <h3 mat-subheader>Total</h3>
42             <h4 mat-line>{{ obs.cart?.items | itemsAmount }} </h4>
43           </ul>
44           </ng-template>
45         </mat-card-content>
46       </mat-card>
47     </div>
48   </span>
49   <span *ngIf="!isIcon()">
50     <button mat-icon-button
51       matBadgeColor="warn"
52       [color]="!green"
53       [disabled]="obs.count <= 0"
54       matBadge="{{ obs.cart?.items | itemsQuantity }}"
55       [matBadgeSidenav]="obs.count <= 0">
56     <app-icon [name]="basket"/></app-icon>
57   </span>
58 </ng-container>
59

```

Figure 30

La figure 31 est en deux parties, ci-dessous. Elle présente l'écran de commande. Ce même écran est utilisé pour la validation mais aussi pour la consultation de la commande en cours.

La première partie est l'affichage qui permet de valider la commande, ainsi l'utilisateur client peut spécifier s'il souhaite manger sur place ou à emporter. Il peut également consulter directement le solde de sa carte et cliquer sur le lien permettant de la recharger si son solde n'est pas suffisant.

La deuxième partie présente l'écran après que l'utilisateur ait cliqué pour confirmer la commande. Ici, la commande est présentée sous forme de facture avec les taxes.

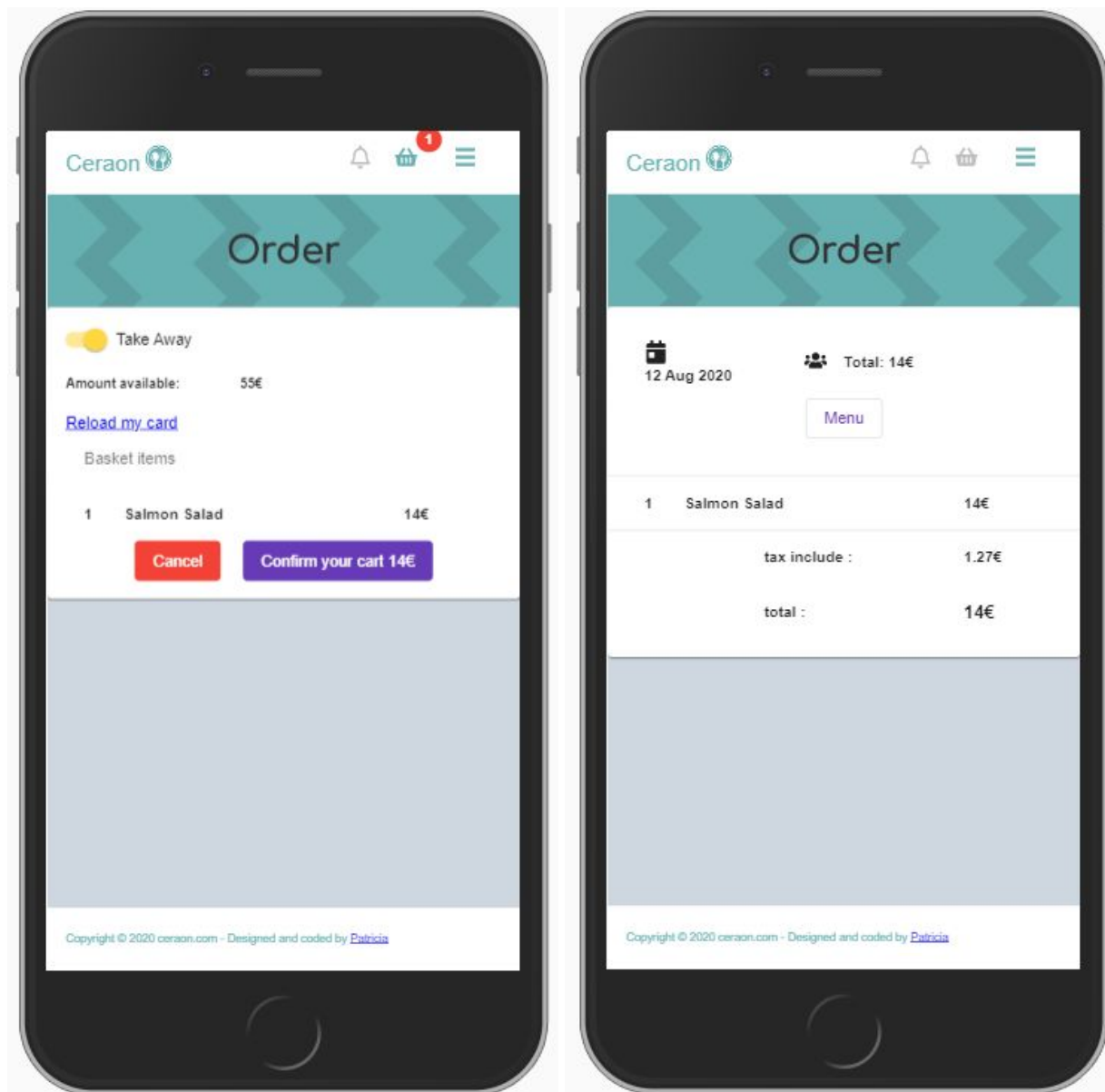


Figure 31

Nous avons vu le lancement d'une commande, regardons maintenant la réalisation de l'interface pour gérer les commandes.

3 Gestion des commandes

L'interface de gestion des commandes étant majoritairement utilisée sur desktop ou tablette, nous présentons donc les visuelles desktop (Figures 32 et 33) mais une version mobile est également disponible.

Quelques explications de l'interface de gestion des commandes sont nécessaires, elles correspondent aux figures 32 et 33:

Un menu permet de changer de vue, ainsi dans la partie "ORDERS", nous pouvons voir la commande du client. L'icône de notification est d'ailleurs activée et un badge signale qu'une notification a été envoyée.

Un bouton permet d'afficher le détail de la commande sur la droite. L'affichage visualisable est le même que celui de la partie "Order" de l'utilisateur. On a utilisé les mêmes composants dans le but de faciliter la maintenance et éviter la réécriture du code. Ainsi, si des changements doivent être effectués sur une commande, un seul composant doit être modifié.

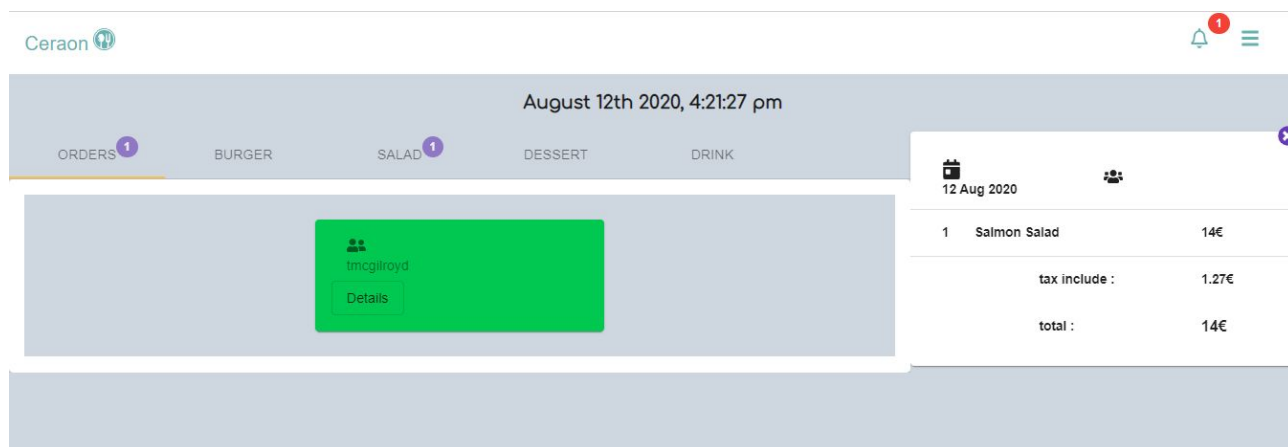


Figure 32

Les badges qui indiquent le nombre d'éléments dans chaque menu permettent de voir visuellement si des bons de commande sont à traiter dans ces catégories. La figure 33, ci-dessous, présente le bon de commande de la partie "SALAD".

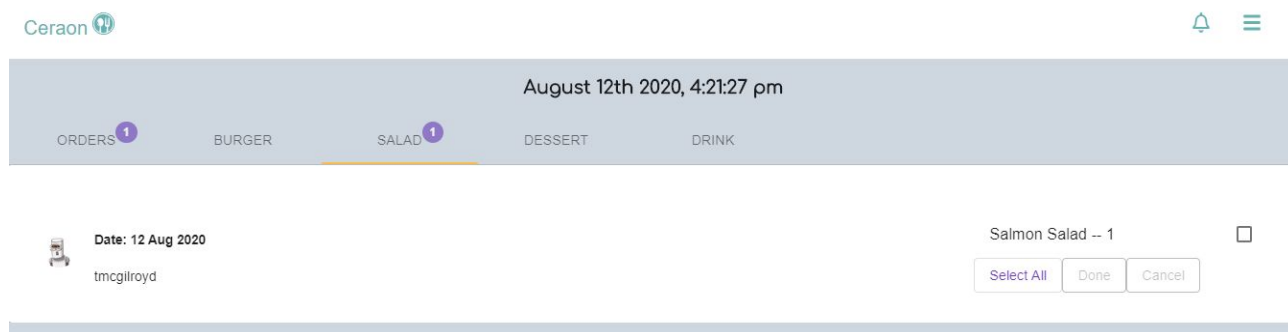


Figure 33

Un fois ce bon de commande traité, l'agent peut sélectionner le produit et cliquer sur "Done". Si le produit n'est plus disponible, l'agent clique sur "cancel". Cela supprime automatiquement le produit de la commande. Cette action va également modifier la visibilité. Le produit ne sera plus visible dans le menu. L'utilisateur reçoit alors une notification.

Regardons maintenant dans la partie 4, ci-dessous, comment sont gérées les notifications.

4 Notifications et WebSocket

Les figures ci-dessous présentent les fonctions appelé signalé le traitement d'une commande.

```
private void deliverOrderBook(Long orderBookId) {  
    OrderBook orderBook = repo.findById(orderBookId).orElseThrow(ResourceNotFoundException::new);  
    orderBook.setDeliverDateTime(LocalDateTime.now());  
    orderBook.setDeliveredTime(Duration.between(orderBook.getOrderDateTime(), LocalDateTime.now()).getSeconds());  
    orderBook.setStatus(OrderBookStatus.DELIVERED);  
    orderService.setOrderTerminated(orderBook.getOrder());  
    simpMessagingTemplate.convertAndSendToUser(orderBook.getOrder().getAccount().getUsername(), "/queue", "your order is ready");  
}
```

Figure 34

Dans la figure 34, ci-dessus, nous pouvons voir le traitement d'une commande qui est prête à être livrée au client. Les données de date sont modifiées pour les statistiques, le statut de la commande passe à "TERMINATED" et une notification est envoyé au client par le biais d'un websocket. Pour cela, nous avons créé un service générique dédié à l'utilisation de WebSocket. Celui-ci prends en paramètre 3 éléments:

- Une injection du service Rx Stomp Service permettant l'initialisation d'accès au Broker de messagerie.
- Une éventuelle configuration du service RxStompService via l'interface Injectable RxStompConfig.
- Une classe contenant d'éventuelles options supplémentaires, dont notamment le endpoint du broker.

Dans cet exemple, le endpoint du broker comprend le username de l'utilisateur, lui seul peut accéder à ce message. Dans la figure suivante nous pouvons voir comment la connexion au broker est effectuée et comment les messages sont ensuite transmis. Le username du client est récupéré après son authentification avec la constante USER_CONFIG. Cet username permet au niveau du constructeur de définir le endpoint de notre serveur sur lequel notre client va souscrire « /username/queue » pour écouter les notifications émises en temps réel.

A chaque notification envoyée, le message est ajouté à la liste des notifications à l'aide de la méthode next() du sujet "subjectMessages\$". Ainsi chaque composant qui souscrit à l'Observable de ce sujet va recevoir les messages envoyés par le broker. Le Behavior Subject est un type qui permet de nous abonner à des messages comme pour un observable. Nous pouvons également lui assigner des valeurs.

```

8  export const progressStompConfig: InjectableRxStompConfig = {
9      websocketFactory: () => {
10         return new WebSocket(Api.WEBSOCKET_USER);
11     }
12 };
13
14 export const USER_CONFIG = new InjectionToken('User Configuration', {
15     factory: () => {
16         return {
17             username: inject(AuthService).username
18         };
19     }
20 });
21
22 @Injectable({
23     providedIn: 'root'
24 })
25 export class UserWebSocketService extends WebSocketService implements OnDestroy {
26
27     private subjectMessages$ = new BehaviorSubject<string[]>([]);
28     private subjectNewMessageCount$ = new BehaviorSubject<number>(0);
29
30     private destroy$ = new Subject();
31
32     messages$ = this.subjectMessages$.asObservable();
33     count$ = this.subjectNewMessageCount$.asObservable();
34
35     constructor(stompService: RxStompService, @Inject(USER_CONFIG) config) {
36         super(
37             stompService,
38             progressStompConfig,
39             new WebSocketOptions('/user/' + config.username + '/queue')
40         );
41     }
42 }

```

Figure 35

```

TS user-message.component.ts X
src > app > components > user-message > TS user-message.component.ts > UserMessageComponent > resetCount
1  import { Component, OnInit, Input } from '@angular/core';
2  import { Observable } from 'rxjs';
3  import { UserWebSocketService } from 'src/app/services/user-websocket.service';
4  import * as _ from 'lodash';
5
6  @Component({
7      selector: 'app-user-message',
8      templateUrl: './user-message.component.html'
9  })
10 export class UserMessageComponent implements OnInit {
11
12     @Input() view: string;
13
14     messages$: Observable<string[]>;
15     count$: Observable<number>;
16
17     constructor(private userWebSocketService: UserWebSocketService) {}
18
19     ngOnInit(): void {
20         // Init Progress WebSocket.
21         this.userWebSocketService.initProgressWebSocket();
22         this.messages$ = this.userWebSocketService.messages$;
23         this.count$ = this.userWebSocketService.count$;
24     }
25
26     resetCount() {
27         this.userWebSocketService.resetSubjectCount();
28     }
29
30 }
31

```

```

user-message.component.html X
src > app > components > user-message > user-message.component.html > ng-container > span > button
1  <ng-container *ngIf="{ count: count$ | async, messages: messages$ | async } as obs;">
2
3      <span *ngIf="view === 'icon'">
4          <button mat-icon-button
5              [color]="'green'"
6              [disabled]="obs.messages?.length < 1"
7              matBadgeColor="warn"
8              [matBadge]="obs.count"
9              [matBadgeHidden]="obs.count > 1"
10             (click)="resetCount()">
11              <app-icon [name]="'bell'" /></app-icon>
12          </button>
13      </span>
14
15      <span *ngIf="view !== 'icon' && obs.count > 0">
16          <mat-card>
17              <mat-list *ngFor="let message of obs.messages">
18                  <mat-list-item>
19                      <h5 fxFlex="70%">
20                          {{ message }}
21                      </h5>
22                      <button mat-icon-button>
23                          <app-icon [name]="'times'" /></app-icon>
24                      </button>
25                  </mat-list-item>
26              </mat-list>
27          </mat-card>
28      </span>
29
30 </ng-container>

```

Figure 36

Dans la figure suivante nous pouvons voir le composant qui permet d'afficher les notifications. J'ai ici utilisé la même méthode que pour afficher le panier, à savoir une vue icône et une vue liste.

A chaque clic sur l'icône le badge est remis à 0 avec la fonction reset Count() comme nous pouvons le voir sur la figure suivante ou le badge sur l'icône de notification n'est plus visible.



Figure 37

Nous avons évité au maximum la duplication de code pour suivre le principe DRY (Do Not Repeat Yourself). Ce principe a été rendu populaire par le livre *The Pragmatic Programmer* de Andrew Hunt et David Thomas. Ainsi comme nous allons le voir en détail dans le paragraphe suivant avec les widgets du tableau de bord mais aussi toutes les icônes, nous avons créé des composants d'affichages génériques.

5 Tableau de bord

La figure 38 représente le tableau de bord développé pour le manager du point de vente. Les widgets de ce dashboard se génèrent à chaque changement de date.

Une seule requête est envoyée pour créer l'ensemble des widgets afin de gagner en performance, les requêtes vers la base pouvant être coûteuses.

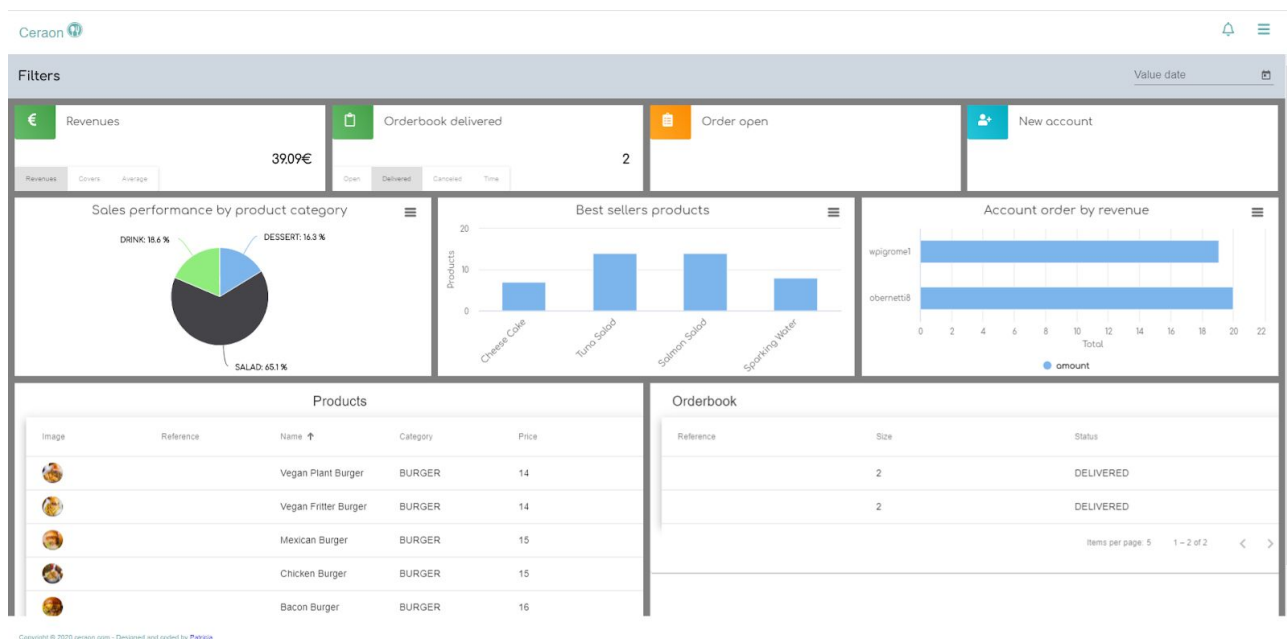


Figure 38

La figure 39 présente le contrôleur REST et la requête Get qui permet de récupérer les données pour générer le tableau de bord. Cette api n'est

disponible qu'aux utilisateurs ayant un rôle ADMIN, cela est géré par Spring Security grâce à l'annotation `@PreAuthorize("hasRole('ROLE_ADMIN')")`.

```
14- /**
15-  * a {@code RestController} to create the dashboard.
16-  *
17-  */
18- @RestController
19- @RequestMapping("/dashboard")
20- @PreAuthorize("hasRole('ROLE_ADMIN')")
21- public class DashboardController {
22-
23-     private final DashboardService service;
24-
25-     /**
26-      * Protected constructor to autowire needed bean.
27-      * <p>
28-      * injects {@code DashboardService} interface
29-      *
30-      * @param service the injected {@code DashboardService}.
31-      */
32-     public DashboardController(DashboardService service) {
33-         this.service = service;
34-     }
35-
36-     /**
37-      * Returns a {@code DashboardDto} by given date.
38-      * <p>
39-      * @param date the filtering date to get data
40-      * @return a {@code DashboardDto}
41-      */
42-     @GetMapping("/{date}")
43-     @Secured("ROLE_ADMIN")
44-     protected DashboardDto getDashboardByDate(@PathVariable("date") @DateTimeFormat(pattern = "ddMMyyyy") String date) {
45-         return service.getDashboardByDate(date);
46-     }
47- }
48-
49-
```

Figure 39

Les figures suivantes représentent des exemples de requêtes créées pour afficher le tableau de bord. Par exemple, la figure 40, permet de récupérer les données concernant le nombre de bons de commande en fonction de leur statut. Le dto `OrderBookStatsDto` est instancié directement dans la requête avec son constructeur.

```
@Query("SELECT new fr.ceraon.resource.dtos.OrderBookStatsDto("
    + "COUNT(CASE When ob.status = 'IN_PROGRESS' then 1 ELSE 0 END), "
    + "COUNT(CASE When ob.status = 'DELIVERED' then 1 ELSE 0 END), "
    + "COUNT(CASE When ob.status = 'CANCELLED' then 1 ELSE 0 END), "
    + "AVG(CASE When ob.status = 'DELIVERED' then ob.deliveredTime ELSE 0 END)) "
    + "FROM OrderBook ob WHERE ob.orderDateTime BETWEEN :start AND :end")
OrderBookStatsDto getOrderBookStats(@Param("start") LocalDateTime start, @Param("end") LocalDateTime end);
```

Figure 40

La figure 41 représente la requête pour récupérer le chiffre d'affaire par client selon une date donnée.

```
@Query("SELECT new fr.ceraon.resource.dtos.RevenueByNameDto(o.account.username, SUM(o.totalItems)) "
    + "FROM Order o WHERE o.creationDate BETWEEN :dateFrom AND :dateTo GROUP BY o.account")
Set<RevenueByNameDto> totalByCustomer(@Param("dateFrom") LocalDateTime dateFrom,
    @Param("dateTo") LocalDateTime dateTo);
```

Figure 41

La figure 41 représente la requête pour récupérer le chiffre d'affaire par catégorie selon une date donnée.

```

@Query("SELECT new fr.ceraon.resource.dtos.RevenueByNameDto(c.name, SUM(i.product.price*i.quantity)) "
+ "FROM Order o " + "INNER JOIN o.items oi JOIN Item i on oi.id = i.id "
+ "INNER JOIN Product p on i.product.id = p.id " + "INNER JOIN Category c on c.name = p.category.name "
+ "WHERE o.creationDate BETWEEN :dateFrom AND :dateTo " + "GROUP BY c ")
Set<RevenueByNameDto> totalByCategory(@Param("dateFrom") LocalDateTime dateFrom,
@Param("dateTo") LocalDateTime dateTo);

```

Figure 42

Ainsi, comme nous pouvons le voir dans la figure 43, ci-dessous, qui représente l'implémentation de l'interface `Dashboard Service`, une seule méthode "getDashboardByDate" nous permet de récupérer tous les éléments nécessaires pour créer le tableau de bord. Ainsi, le client a besoin d'une seule requête pour récupérer toutes les informations. Cela permet de gagner en performance.

```

@Service
public class DashboardServiceImpl implements DashboardService {

    private final OrderRepository orderRepo;
    private final OrderBookRepository orderBookRepo;

    /**
     * @param orderRepo
     * @param orderBookRepo
     */
    public DashboardServiceImpl(
        OrderRepository orderRepo,
        OrderBookRepository orderBookRepo) {
        this.orderRepo = orderRepo;
        this.orderBookRepo = orderBookRepo;
    }

    private Set<RevenueByNameDto> totalByCategory(DayTimeGap dayTimeGap) {
        return orderRepo.totalByCategory(dayTimeGap.getStart(), dayTimeGap.getEnd());
    }

    private Set<RevenueByNameDto> totalByProduct(DayTimeGap dayTimeGap) {
        return orderRepo.totalByProduct(dayTimeGap.getStart(), dayTimeGap.getEnd());
    }

    private SalesDto getAverageSales(DayTimeGap dayTimeGap) {
        return orderRepo.getAverageSales(dayTimeGap.getStart(), dayTimeGap.getEnd());
    }

    private Set<RevenueByNameDto> totalByAccount(DayTimeGap dayTimeGap) {
        return orderRepo.totalByCustomer(dayTimeGap.getStart(), dayTimeGap.getEnd());
    }

    private OrderBookStatsDto getOrderbookStats(DayTimeGap dayTimeGap) {
        return orderBookRepo.getOrderbookStats(dayTimeGap.getStart(), dayTimeGap.getEnd());
    }

    @Override
    public DashboardDto getDashboardByDate(String date) {
        DayTimeGap dayTimeGap = new DayTimeGap(date);
        return new DashboardDto(
            getAverageSales(dayTimeGap),
            totalByCategory(dayTimeGap),
            totalByAccount(dayTimeGap),
            totalByProduct(dayTimeGap),
            getOrderbookStats(dayTimeGap));
    }
}

```

Figure 43

Dans la partie front end, nous avons utilisé la librairie Gridster2 pour créer la grille responsive qui contient les widgets. Trois types de widgets ont été réalisés :

- Widgets génériques pour l'affichage simple des chiffres statistiques tels que le chiffre d'affaire, le nombre de couverts, le panier moyen ...
- Widgets graphiques réalisés à l'aide de la librairie HighChart.
- Widgets de tableau de données.

Ainsi, dans la figure 44, ci-dessous, nous pouvons voir le composant dashboard. Gridster nous permet de générer une grille à laquelle nous pouvons ajouter des configurations. Le type de widget est donc ajouté directement dans l'objet permettant de créer les cases de la grille.

```
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css'],
})
export class DashboardComponent implements OnInit, OnDestroy {

  gridsterOptions: GridsterConfig;
  filters: Filters;
  unitHeight: number;
  path = 'dashboard';

  private destroy$ = new Subject();

  dashboard: GridsterItem[] = [
    { cols: 6, rows: 1, y: 0, x: 0, type: 'widget', category: 'sales' },
    { cols: 6, rows: 1, y: 0, x: 6, type: 'widget', category: 'orderbook' },
    { cols: 6, rows: 1, y: 0, x: 12, type: 'widget', category: 'order' },
    { cols: 6, rows: 1, y: 0, x: 18, type: 'widget', category: 'account' },
    { cols: 8, rows: 2, y: 1, x: 0, type: 'piechart' },
    { cols: 8, rows: 2, y: 1, x: 8, type: 'column' },
    { cols: 8, rows: 2, y: 1, x: 16, type: 'barchart' },
    { cols: 12, rows: 3, y: 3, x: 0, type: 'products' },
    { cols: 12, rows: 3, y: 3, x: 12, type: 'orderbooks' }
  ];
}
```

Figure 44

Dans un composant appelé parent (figure 45), nous allons sélectionner le composant du widget à afficher en fonction du type spécifié.

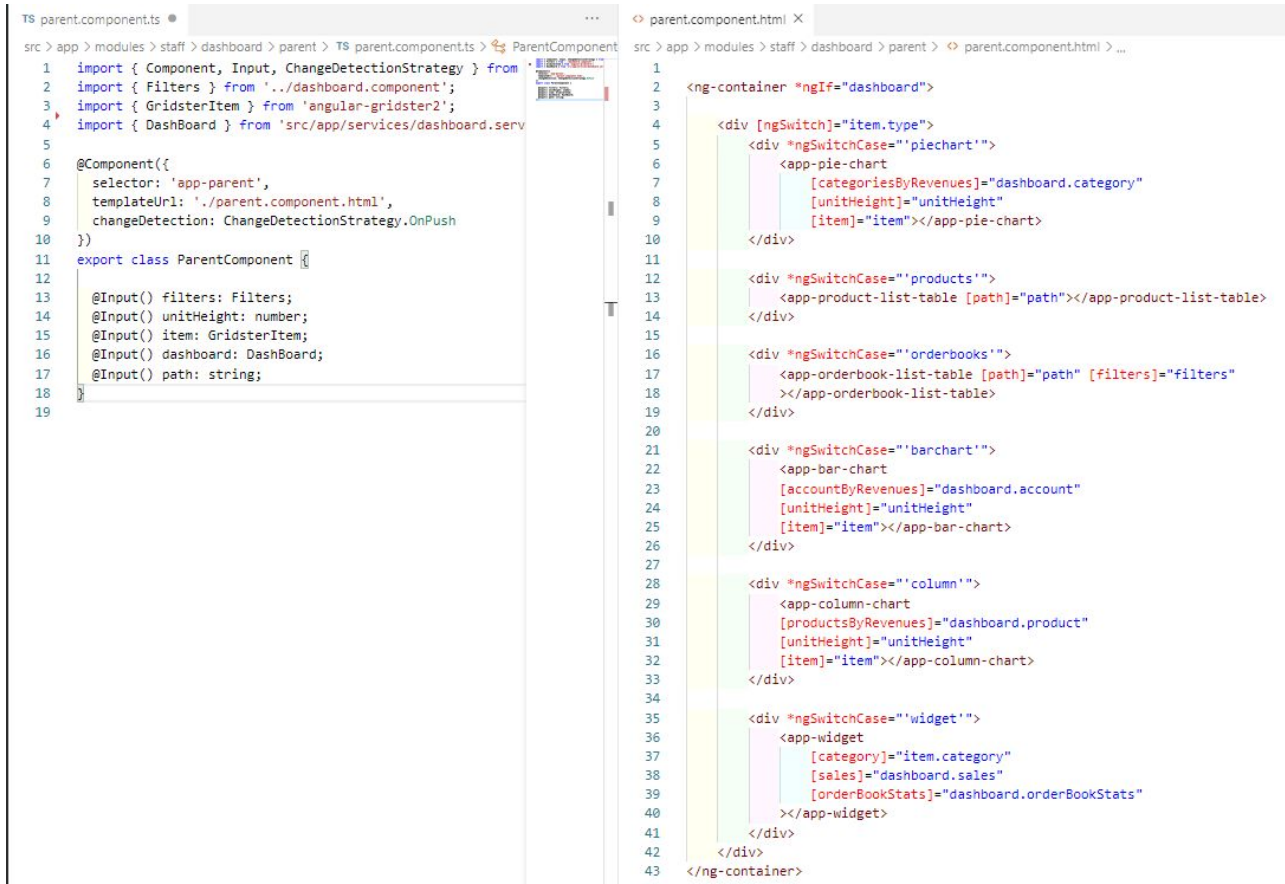


Figure 45

Les composants widgets (figure 46) sont affichés en fonction de la catégorie indiquée.

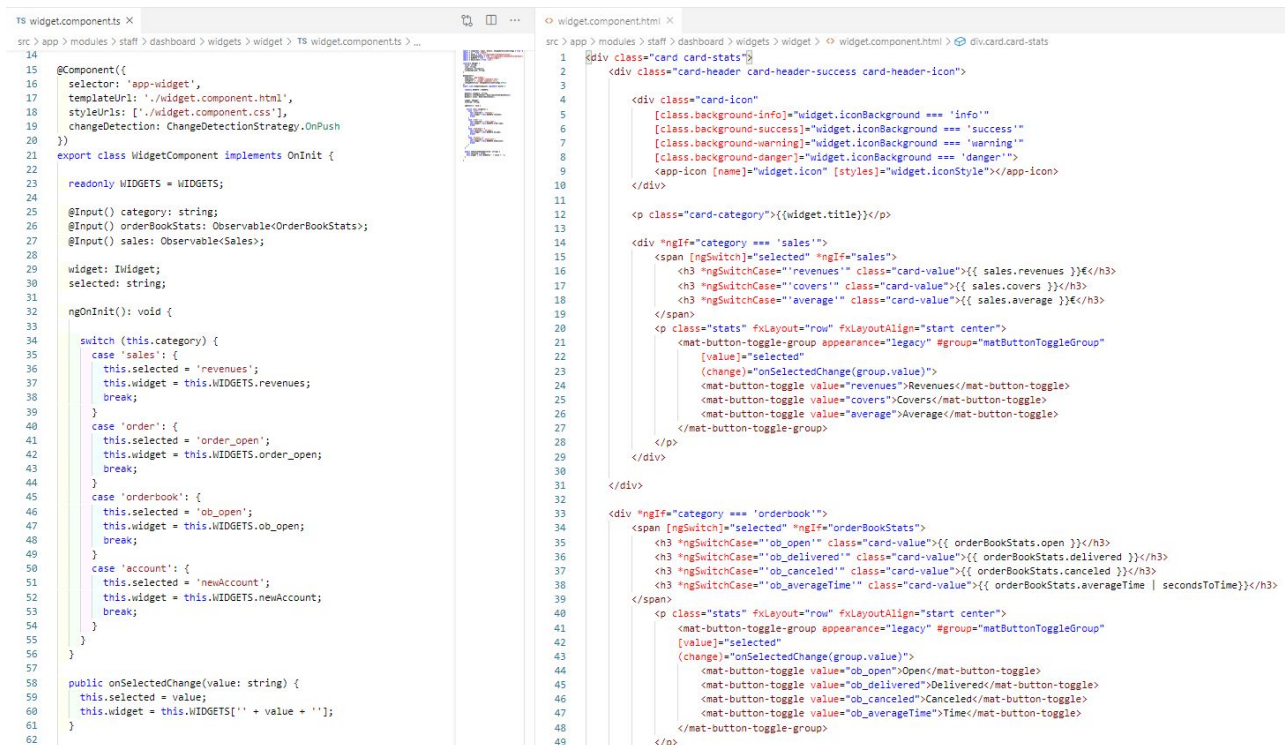


Figure 46

Une configuration est importée dans le composant, elle contient l'ensemble des éléments nécessaires pour personnaliser le widget tel que l'icône à utiliser mais aussi son titre. Par exemple, pour l'affichage du widget "revenue" nous utilisons la configuration ci-dessous (figure 47) :

```
export const WIDGETS = {
  revenues: {
    title: 'Revenues',
    icon: 'euro_sign',
    iconStyle: [['color', 'white'], ['height', '30px'], ['width', '30px']],
    iconBackground: 'success',
  },
},
```

Figure 47

Concernant les icônes, nous avons fait le choix de créer une librairie de csv. Ainsi, un composant spécial pour les icônes peut-être utilisé en spécifiant simplement le nom de l'icône voulue. Des éléments tel que le style peuvent également être précisés dans les Inputs de ce composant.

6 Tests

Pour effectuer des tests, nous utilisons une base de données spécifique qui est générée à chaque test avec hibernate. Un fichier import.sql contient tous les inserts à persister. La figure 48, ci-dessous, présente le fichier de propriété pour les tests et un extrait du fichier pour importer les données.

```
1 spring.profiles.active=test
2
3 spring.datasource.username: usertest
4 spring.datasource.password: ceraontest
5
6 spring.datasource.url=jdbc:mysql://localhost:3306/ceraon_test?allowMultiQueries=true&c
7
8 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
9
10
11 # =====
12 # = JPA / HIBERNATE
13 # =====
14
15 spring.jpa.show-sql = false
16 spring.jpa.hibernate.ddl-auto = create-drop
17 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
18 spring.jpa.properties.javax.persistence.validation.mode=none
19
```

```
10 INSERT INTO categories(name,picture) VALUES ("BURGER","assets\\img\\burger.jpg");
11 INSERT INTO categories(name,picture) VALUES ("SALAD","assets\\img\\salad.jpg");
12 INSERT INTO categories(name,picture) VALUES ("DRINK","assets\\img\\cocktail.jpg");
```

Figure 48

La figure 49, présente une partie des tests d'intégrations effectués. Le profil "test" doit être spécifié au niveau de la classe de test avec l'annotation `@ActiveProfiles("test")` pour que le fichier properties correspondant soit utilisé.



Figure 49

Le premier test lancé permet de vérifier que toutes les insertions ont bien été importées dans la base de données de test. Le deuxième test appelle la méthode du controller pour créer un nouvel account. Pour ce faire, nous utilisons un fichier csv (figure 50) qui contient un json avec les éléments permettant la création d'un account.

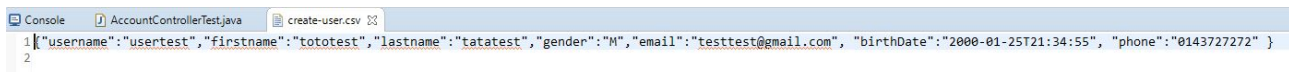


Figure 50

Conclusion

Notre projet est de proposer une application web qui gère des commandes à distance pour des restaurants internes d'entreprise. Les objectifs de cette application sont d'éliminer les phénomènes de forte affluence et de temps d'attente important remarqués à mon travail. Les étapes de conception et de réalisation de notre application exposés dans ce présent mémoire ont pris en considération les contraintes de temps, de coût et de volumétrie. Nous avons réfléchi aux besoins des utilisateurs en nous aidant de la méthodologie des personas. Ce qui nous a permis d'établir les fonctionnalités les plus pertinentes à destination de notre cible. A partir de nos contraintes et des besoins établis, nous avons choisi les outils les plus adaptés pour créer cette application mais aussi par une analyse précise des avantages et des inconvénients des technologies à disposition.

Ceci dit, certains éléments doivent être implémentés pour améliorer l'application créée comme par exemple, l'option multi langage.

J'ai pris beaucoup de plaisir à créer cette application. Ma soif de connaissances a été nourrie par la construction de ce chef d'oeuvre que je souhaite continuer à perfectionner.