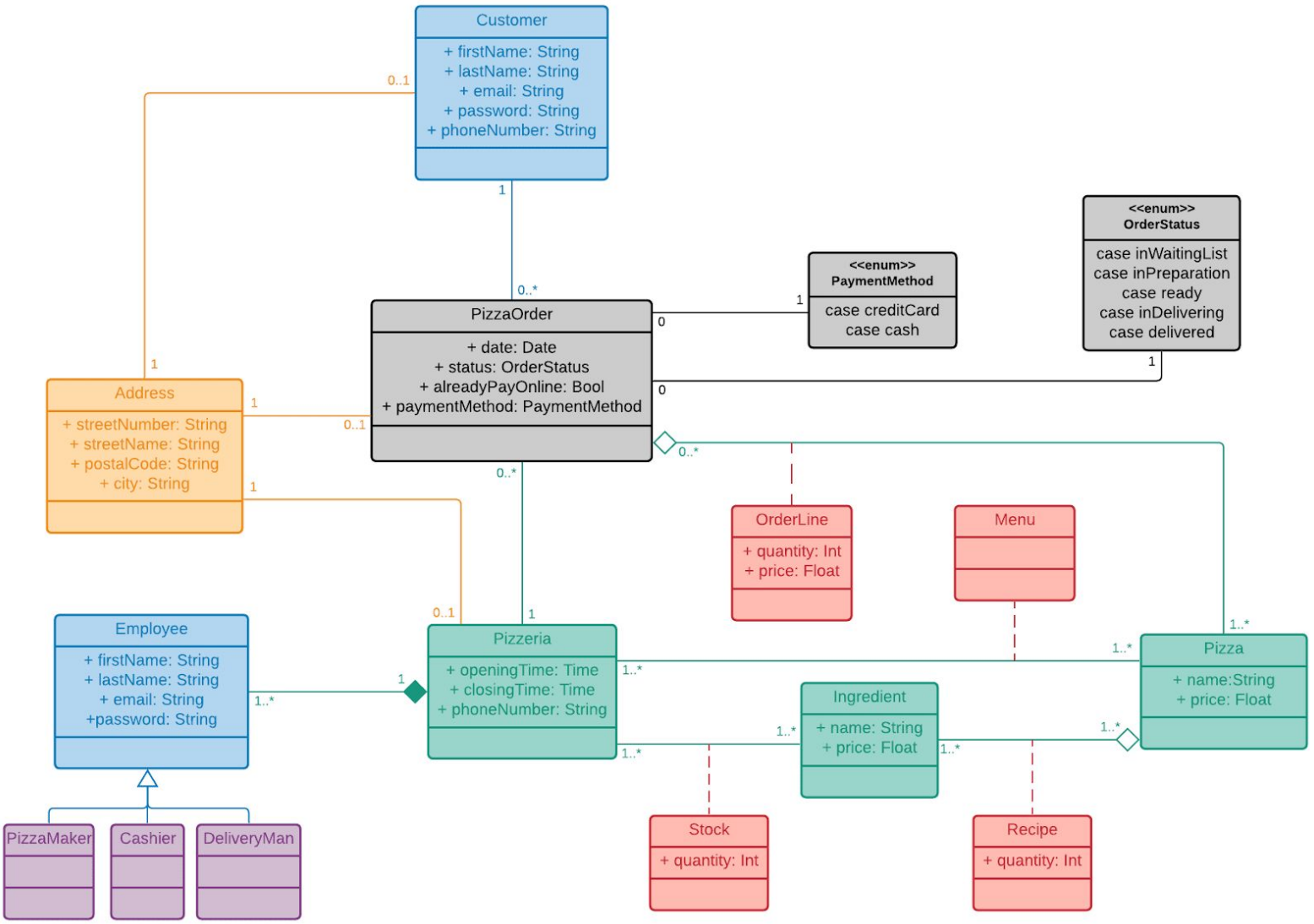
Poudja CANESSANE 2020

Spécifications techniques de OC Pizza

Projet 7 OpenClassrooms: Concevez la solution technique d'un système de gestion de pizzeria

Table des matières

1) Diagramme de classes



Dans l'optique de modéliser le domaine fonctionnel, nous avons élaboré ce diagramme de classes qui permet de mettre en lumière les relations des différentes entités constituant le site web.

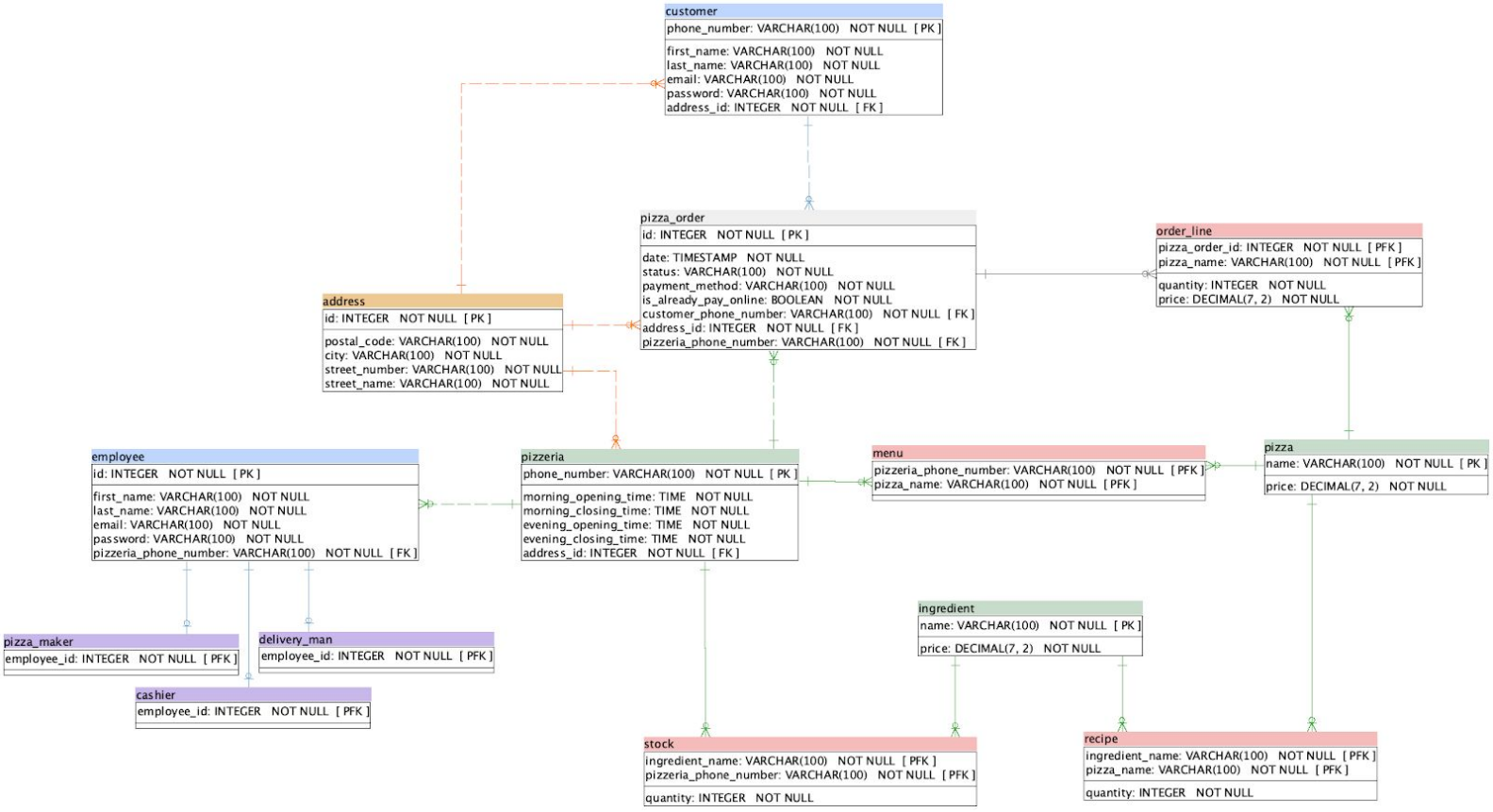
Nous pouvons y voir les 7 classes "principales" qui sont *Customer*, *Employee*, *Address*, *PizzaOrder*, *Pizzeria*, *Ingredient* et *Pizza*.

Puis celles que l'on pourrait qualifier de "secondaires" sont à proprement parler des classes filles de *Employee* à savoir *PizzaMaker*, *Cashier* et *DeliveryMan*. Ainsi que les classes d'association que sont *OrderLine*, *Menu*, *Stock* et *Recipe*.

Nous pouvons ensuite nous intéresser aux types d'associations qui, d'une part, lient *Ingredient* à *Pizza* et *Pizza* à *PizzaOrder* avec une flèche d'**agrégation**. Puis d'autre part, lient *Employee* à *Pizzeria* avec une flèche de **composition** puisqu'un employé ne peut travailler que dans une pizzeria à un instant T.

Enfin penchons nous sur la multiplicité des associations. Les 3 associations qui lient *Address* à *Customer*, *PizzaOrder* et *Pizzeria* sont de la catégorie ***one-to-one***. Celles de la catégorie ***one-to-many*** lient *PizzaOrder* à *Customer* et *Pizzeria*. Toutes les autres sont des ***many-to-many*** (exceptées les 3 flèches d'héritages et les 2 autres liants *PizzaOrder* aux énumérations *PaymentMethod* et *OrderStatus*).

2) Modèle physique de données



Le modèle physique de données est une étape intermédiaire entre la modélisation du domaine fonctionnel et la création de la base de données. C'est à partir de ce dernier que nous avons pu générer automatiquement le script SQL de création du schéma de la base de données grâce au logiciel de modélisation *SQL Power Architect*.

La différence avec le diagramme de classes est que nous devons spécifier pour chaque table une **clé primaire**.

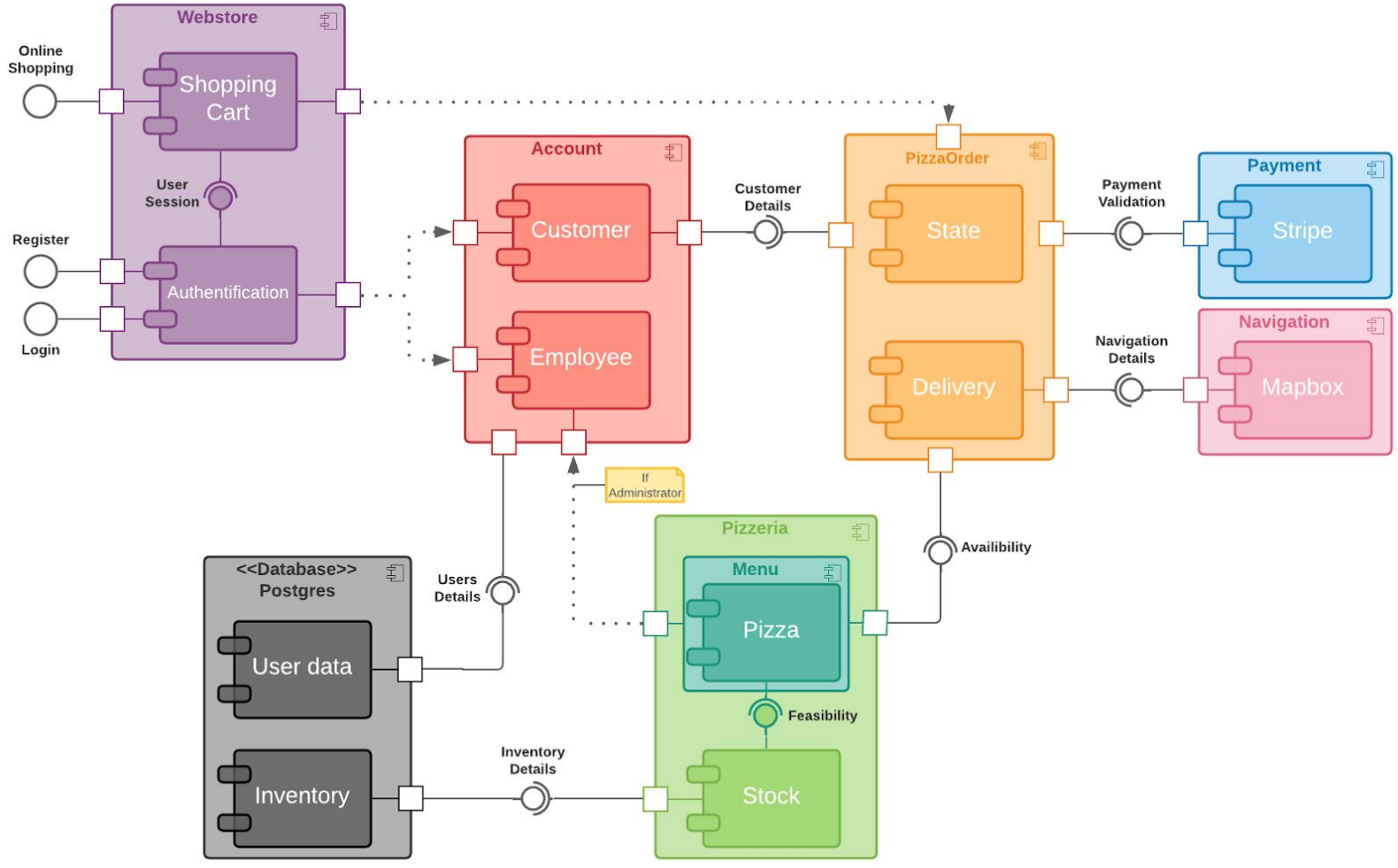
Soit il s'agit d'une colonne qui a vocation d'être unique est constante comme le nom pour les tables *ingredient* et *pizza* ou le numéro de téléphone pour les tables *customer* et *pizzeria*.

Soit nous créons une nouvelle colonne auto-incrémentée généralement nommée *id* comme pour les tables *pizza_order*, *address* et *employee*.

Les associations sont traduites par les **clés étrangères** (relation non-identifiante).
La clé primaire des tables d'associations est composée des clés étrangères des 2 autres tables qu'elles associent (relation identifiante) comme pour la clé primaire des tables filles de *employee*.

Il faut également indiquer la précision des colonnes de types *VARCHAR* (ici 100) et *DECIMAL* (ici 7,2) ainsi que si une colonne supporte les valeurs nulles (ici elles sont toutes *NOT NULL*).

3) Diagramme de composants



Nous avons ensuite préparé ce diagramme de composants qui permet de détailler les interactions des différents composants en représentant les interfaces fournies et requises ainsi que les dépendances de ces derniers.

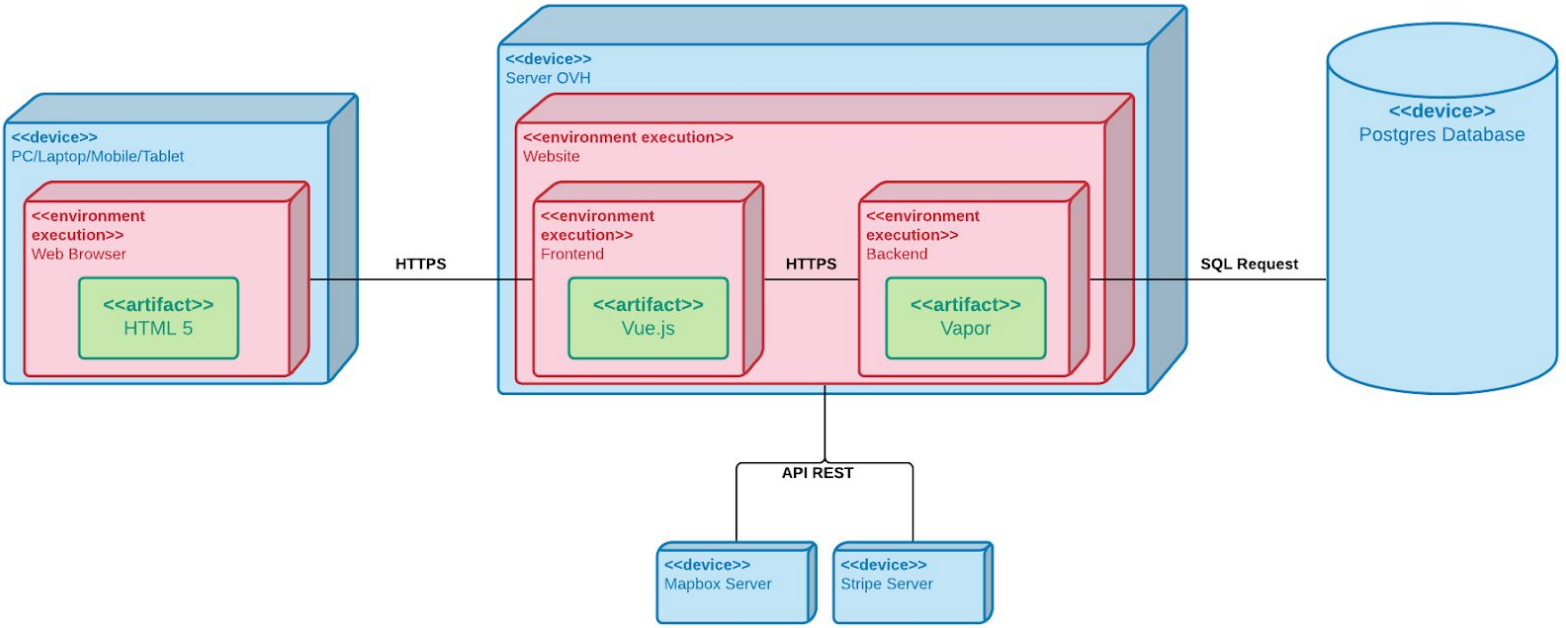
Nous y voyons les composants internes que sont *Webstore*, *Account*, *PizzaOrder*, *Payment*, *Navigation* et *Pizzeria*. Ainsi que le composant externe *Postgres*, la base de données.

Nous comprenons, en observant ce diagramme, qu'il faut impérativement s'inscrire ou se connecter pour commander sur le site. Et qu'il y a 2 types de comptes, un pour les clients, l'autre pour les employés (qui ont la capacité de gérer le menu d'une pizzeria s'il s'agit d'un administrateur, c'est-à-dire soit un responsable d'un point de vente soit le directeur). Les comptes requièrent des données des utilisateurs stockées dans la base de données.

Le panier d'un client dépend de sa commande qui elle-même nécessite les interfaces fournies par les composants *Customer*, *Stripe*, *Mapbox* et *Pizza* pour accéder aux informations du client, valider le paiement, obtenir les informations de navigation ainsi que la disponibilité des pizzas.

Sachant qu'une pizza est disponible si elle est réalisable selon les stocks et cette information est fournie par le composant *Inventory* de la base de données.

4) Diagramme de déploiement



Analysons finalement ce diagramme de déploiement qui explique comment les composants du système sont répartis sur les infrastructures physiques.

L'utilisateur accède à la partie *Frontend* du site web via le protocole **HTTPS** depuis son appareil grâce à son navigateur web qui utilise le langage **HTML 5**.

Le site web est composé du *Frontend* qui utilise le framework **Vue.js** et du *Backend* qui utilise le web framework **Vapor** (open-source écrit en Swift. Il permet d'écrire du code asynchrone, de créer des applications Web, des sites et des API modernes avec HTTP. Presque 100 fois plus rapide que les frameworks Web populaires utilisant Ruby et PHP. Les applications Vapor sont très concises et puissantes. Le temps de débogage est fortement réduit avec l'autocompletion, les warnings et les errors ainsi que les breakpoints).

Ces 2 parties communiquent via le protocole **HTTPS**. Le site est hébergé dans un serveur OVH du fait de son rapport espace de stockage et tarif.

Notre système est dépendant de APIs *Stripe* et *Mapbox* ainsi que de la base de données *Postgres* avec laquelle il communique via le Backend avec des requêtes SQL.