***IT Consulting & Development***

Résumé

Définition du domaine fonctionnel et conception de l’architecture technique  
 de la solution



**Spécifications techniques**

**OC-PIZZA**

**Table des matières**

[2 Contexte 2](#_Toc50817895)

[3 Diagramme de classe 4](#_Toc50817896)

[3.1 Gestion des utilisateurs : 4](#_Toc50817897)

[3.2 Gestion des commandes : 4](#_Toc50817898)

[3.3 Gestion des produits : 5](#_Toc50817899)

[4 Modèle physique de données 7](#_Toc50817900)

[4.1 Gestion des utilisateurs : 7](#_Toc50817901)

[4.2 Gestion des commandes : 7](#_Toc50817902)

[4.3 Gestion des produits : 8](#_Toc50817903)

[5 Diagramme de composants 11](#_Toc50817904)

[5.1 SearchEngine : 11](#_Toc50817905)

[5.2 Authentication : 11](#_Toc50817906)

[5.3 ShoppingCart : 11](#_Toc50817907)

[6 Diagramme de déploiement 13](#_Toc50817908)

[6.1 User : 13](#_Toc50817909)

[6.2 Web Server : 13](#_Toc50817910)

[6.3 Application Server : 13](#_Toc50817911)

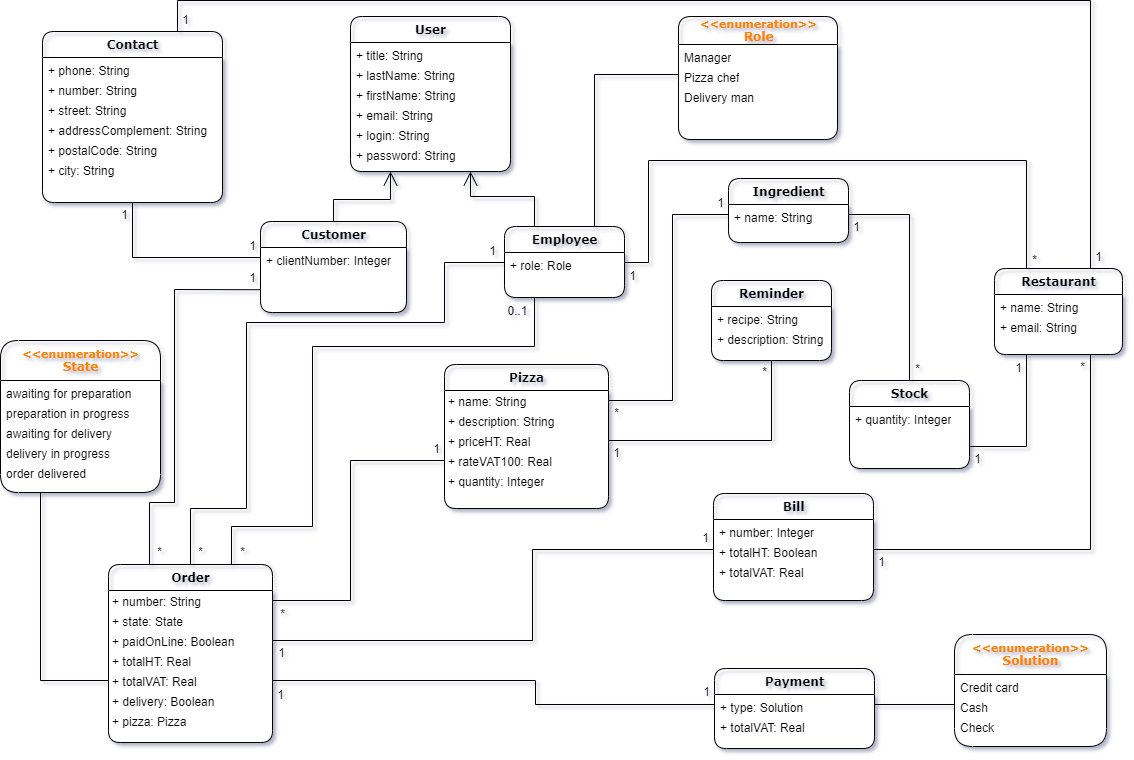
[6.4 Database Server : 14](#_Toc50817912)

# Contexte

« **OC Pizza** », jeune groupe de pizzeria en plein essor, est spécialisé dans la vente en livraison ou à emporter de pizza. Le groupe compte actuellement 5 points de vente et prévoit d’en ouvrir au moins 3 de plus d’ici la fin de l’année.

Dans le cas présent, il est demandé de mettre en place un nouveau système informatique qui sera déployé dans toutes les pizzerias du groupe. Ce système doit permettre notamment :

* D’améliorer l’efficacité de la gestion des commandes. Depuis leur réception à leur livraison, en passant bien entendu par leur préparation
* De suivre en temps réel les commandes passées, et celles en préparation
* De suivre en temps réel le stock d’ingrédients restants afin de connaitre les pizzas encore réalisables.
* De proposer un site internet afin qu’un client puisse :
* Passer une commande en plus de la prise de celle-ci par téléphone. Ou directement dans un point de vente
* Payer en ligne s’il le souhaite. Le cas échéant, la commande sera réglée lors de la livraison à domicile ou dans un point de vente
* Modifier ou annuler la commande tant que celle-ci n’a pas été préparée
* De proposer un aide-mémoire aux pizzaïolos indiquant la recette des produits proposés
* D’informer ou notifier les clients sur l’état de leur commande



# Diagramme de classe

Le diagramme ci-dessus représente les classes qui seront utilisées dans le cadre de notre solution technique.

Il peut être divisé en 3 parties distinctes, une première qui concernera la « **Gestion des utilisateurs** », une seconde pour la « **Gestion des commandes** » et une dernière pour la « **Gestion des produits** ».

Voici les classes que ces différentes parties contiennent :

## Gestion des utilisateurs :

* **User** : Classe mère utilisée pour identifier tous les utilisateurs, que ce soit les clients (classe héritée « **Customer** ») ou les employés (classe héritée « **Employee** »). La classe « **Employee** » possède un attribut « **role** » permettant d’identifier le rôle d’un employé au sein de la société. Cet attribut est une instance de l’énumération « **Role** ».
* **Contact** : Classe qui est utilisée pour stocker les coordonnées des clients, mais également des différents points de vente (cinq actuellement, huit avant la fin de l’année).

## Gestion des commandes :

* **Order** : Cette classe est instanciée lorsque la commande est validée par le client. Elle contient un numéro unique et les différentes informations de la commande passée. En plus de l’attribut « **delivery** » qui concerne la livraison de la commande, un autre booléen apparait ici, « **paidOnLine** ». Cet attribut varie selon que la commande soit déjà payée (en ligne) ou pas, notamment dans le cadre d’une livraison ou de la récupération d’une commande dans un point de vente. Un autre attribut nommé « **state** » varie selon l’état d’avancement de la commande. Les différents états sont contenus dans l’énumération « **State** ».

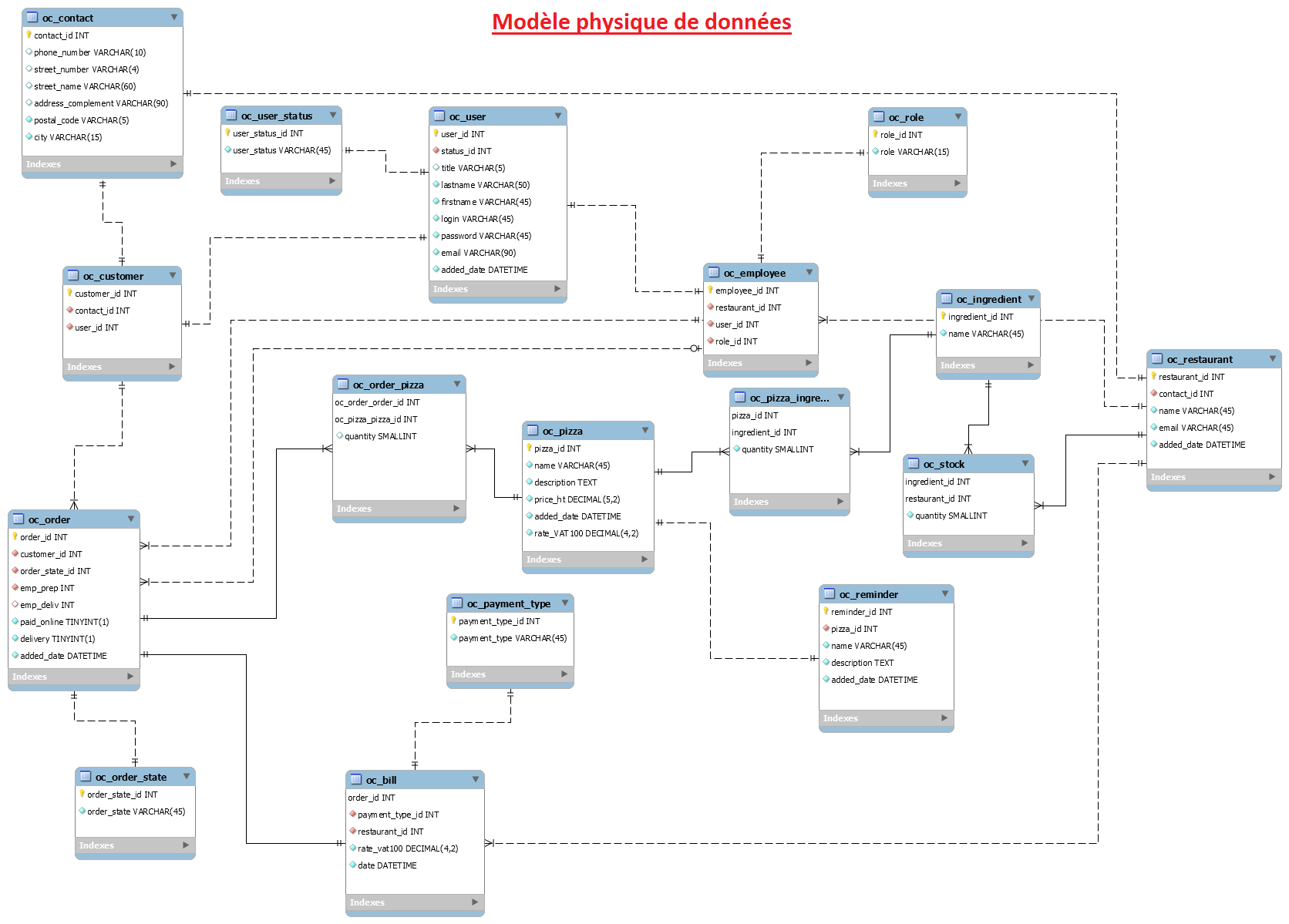
Une liaison est présente avec la classe « **Customer** », afin d’identifier le client à l’origine de la commande. Et également une double liaison avec la classe « **Employee** », dont une première, correspondante à l’employé qui a préparé la commande, et une autre, au livreur.

* **Bill / Payment** : Ces deux classes concernent la facturation de la commande au client. La classe « **Payment** » est utilisée lors du paiement de la commande et contient comme attribut « **type** » qui est la solution de paiement choisie par le client (*carte de crédit, espèce, chèque*). Les différentes solutions sont contenues dans une énumération nommée « **Solution** ».

Une autre classe nommée « **Bill** » concerne la facture de la commande. Elle permet de garder un historique des commandes traitée, elle est également liée au point de vente qui a traité la commande.

## Gestion des produits :

* **Pizza** : Cette classe est commune aux différentes parties énoncées. Elle est instanciée notamment lors de l’ajout d’un produit dans la commande et contient les informations de celui-ci.
* **Ingredient / Stock** : Ces deux classes concernent l’inventaire des ingrédients nécessaires à la confection des pizzas. La classe « Stock » est, quant à elle, liée à un point de vente en particulier.
* **Reminder** : La classe « **Reminder** » est utilisée comme aide-mémoire et contient les recettes des différentes pizzas proposées par le groupe. Elle est commune aux différents points de vente.
* **Restaurant** : Cette dernière est utilisée pour identifier les différents points de vente. Elle est également liée à la classe « **Contact** » pour obtenir ici les coordonnées des différents points de vente.



# Modèle physique de données

Le diagramme ci-dessus (**modèle physique de données**) représente l’architecture de la base de données et également les relations qu’il existe entre les différentes tables.

De la même manière que le **diagramme de classe**, nous pouvons ici le diviser en trois parties distinctes, dont voici les tables principales :

## Gestion des utilisateurs :

* **OC\_User** : La table « **OC\_User** » est utilisée pour stocker les informations d’identification des différents utilisateurs présents dans la base de données. Cette table contient notamment les clients, mais également les employés du groupe. Afin de différencier ces deux types d’utilisateur, il existe un champ « **status\_id** » lié à la table « **OC\_User\_Status** ». Ce champ représente les différents types d’utilisateur énumérés dans la table « **OC\_User\_Status** ».
* **OC\_Customer**: Les champs de la table « **OC\_Customer** » sont liés aux utilisateurs « **clients** » de la table « **OC\_User** ». C’est-à-dire, ceux qui ont un « **status\_id** » équivalent à « **Customer** » (voir les explications de la table « **OC\_User** »). La table « **OC\_Customer** » est également liée à la table « **OC\_Contact** » qui contient les coordonnées des clients.
* **OC\_Employee**: De la même façon que la table « **OC\_Customer** », la table « **OC\_Employee** » est liée aux utilisateurs « **employés** » de la table « **OC\_User** », et donc, ceux qui ont un « **status\_id** » équivalent à la valeur « **OC\_Employee** ». Un champ « **role\_id** » est également présent dans cette table, et lié à la table « **OC\_Role** » qui énumère les différentes fonctions au sein du groupe. Un champ « **restaurant\_id** » est également présent pour identifier l’établissement d’accueil de l’employé.

## Gestion des commandes :

* **OC\_Order** : La table « **OC\_Order** » contient les informations des commandes passées par les clients. Elles sont identifiées et liées à un client par le champ « **customer\_id** ». Un autre champ « **order\_state\_id** » représente l’état actuel de la commande. Il est lié à la table « **OC\_Order\_State** » qui énumère les différents états d’une commande (en cours de préparation, livrée etc..).

Deux liaisons à la table « **OC\_Employee** » sont également visibles. Un premier champ nommé « **emp\_prep** » et lié à « **employee\_id** » permet d’identifier l’employé chargé de préparer la commande. Ce champ ne peut pas être « **null** » dans la mesure ou un préparateur est obligatoire pour chaque commande. Contrairement au livreur identifié ici par le champ nommé « **emp\_deliv** » et lié à « **employee\_id** », qui lui peut avoir une valeur « **null** ». La livraison n’étant pas obligatoire lors du passage d’une commande.

* **OC\_Bill** : Cette table contient les informations de facturation des différentes commandes. Un champ « **order\_number** » contient le numéro de commande lié à la facture. Le champ « **payment\_id** » lié à la table « **OC\_Payment\_Type** » permet d’identifier le type de paiement utilisé (CB, espèce, chèque) énuméré dans cette même table. Le champ « **restaurant\_id** » permet d’identifier l’établissement d’origine de la facture.

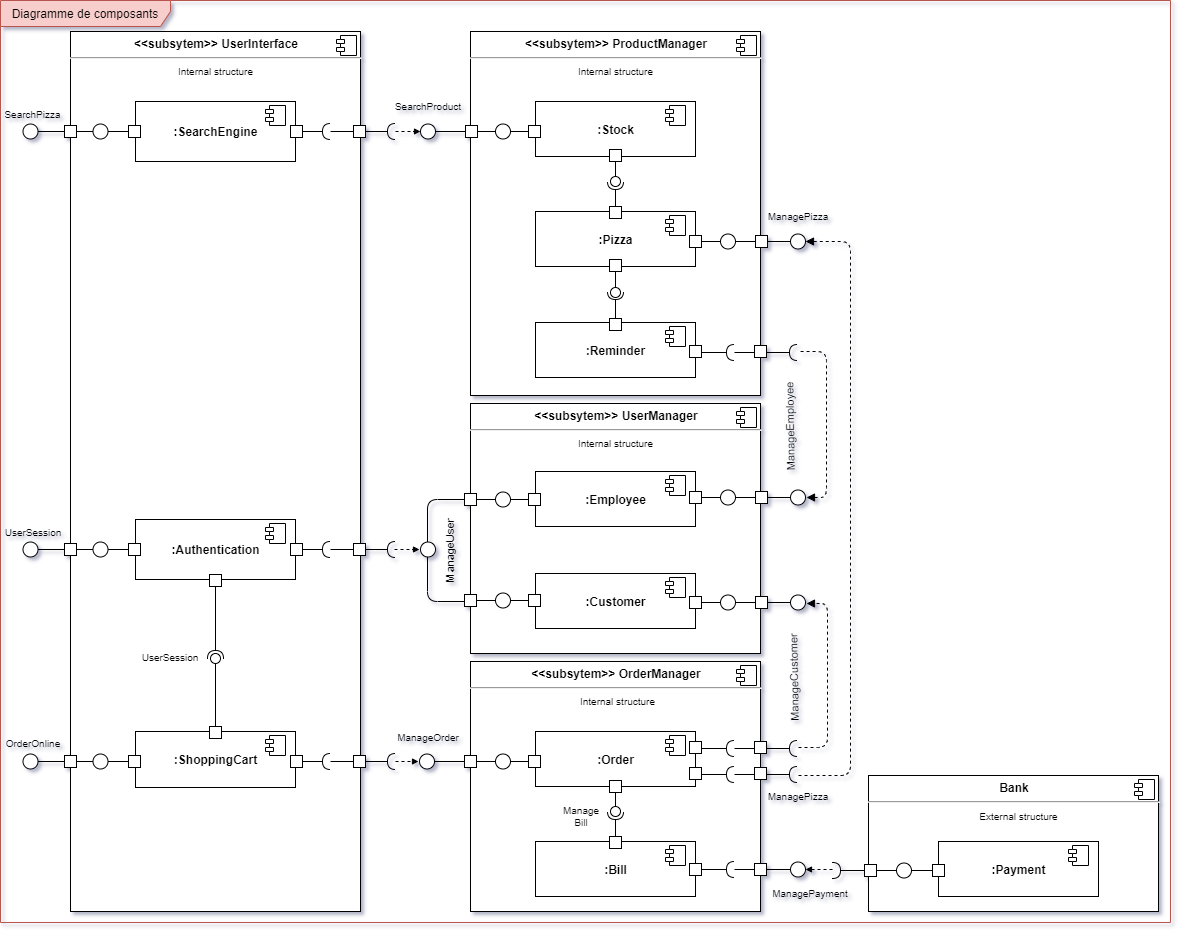
## Gestion des produits :

* **OC\_Pizza** : Cette table contient les différents produits proposés par le groupe. Outre le nom et la description du produit, on y retrouve également son prix hors-taxe à l’unité et la TVA à appliquer. Elle est également liée à la table « **OC\_Order** » par l’intermédiaire d’une relation many-to-many avec la table intermédiaire nommée « **OC\_Order\_Pizza** ». Cette dernière permettant de lier un ou plusieurs produits à une commande, et en quantité définie par le champ « **quantity** » dans cette même table.
* **OC\_Ingredient**: La table « **OC\_Ingredient** » contient tous les ingrédients utilisés pour confectionner les différentes pizzas proposées. On y trouve un « **id** » et le nom de l’ingrédient. Elle est liée à la table « **Pizza** » par l’intermédiaire d’une relation many-to-many avec autre table nommée « **OC\_Pizza\_Ingredient** », permettant notamment d’identifier les ingrédients utilisés pour chaque pizza et en quantité définie.
* **OC\_Stock**: Cette table liée à « **OC\_Ingredient** » permet de connaitre la disponibilité des différents ingrédients ainsi que la quantité restante. De manière à identifier les pizzas encore réalisables à n’importe quel moment du service. En plus de « **ingredient\_id** », la clé primaire composée contient également « **restaurant\_id** » qui permet d’associer un ingrédient à un point de vente en particulier. Nous pourrons donc avoir, par exemple :

|  |  |  |
| --- | --- | --- |
| **Table Stock** | | |
| **Ingredient\_id** | **Restaurant\_id** | **quantité** |
| 1 | 1 | 187 |
| 1 | 2 | 329 |
| 1 | 3 | 67 |

Dans cet exemple, un même ingrédient apparait dans trois points de vente différents.

* **OC\_Restaurant**: Cette table permet d’identifier les différents points de vente. Elle est également liée à la table « **OC\_Contact** » par l’intermédiaire du champ « **contact\_id** » pour retrouver les coordonnées des différents points de vente.
* **OC\_Reminder**: « **OC\_Reminder** » est la table utilisée pour l’aide-mémoire, contenant les recettes des produits proposés. Elle est destinée à être utilisée (si nécessaire) par les pizzaïolos lors de la réalisation des pizzas.



# Diagramme de composants

Le diagramme de composants ci-dessus décrit l’organisation du système du point de vue des modules de classes. Il permet notamment d’illustrer les dépendances et relations existantes.

La partie gauche nommée « **UserInterface** » peut-être considéré comme le point de départ lors de l’utilisation de l’application. Il s’agit là de la partie interface utilisateur.

## SearchEngine :

Dans cette partie, nous voyons que lors de la recherche d’une pizza, par un client par exemple, le système fait appel au « **ProductManager** » et notamment au composant « **:Stock** » lui-même lié au composant « **:Pizza** ».

## Authentication :

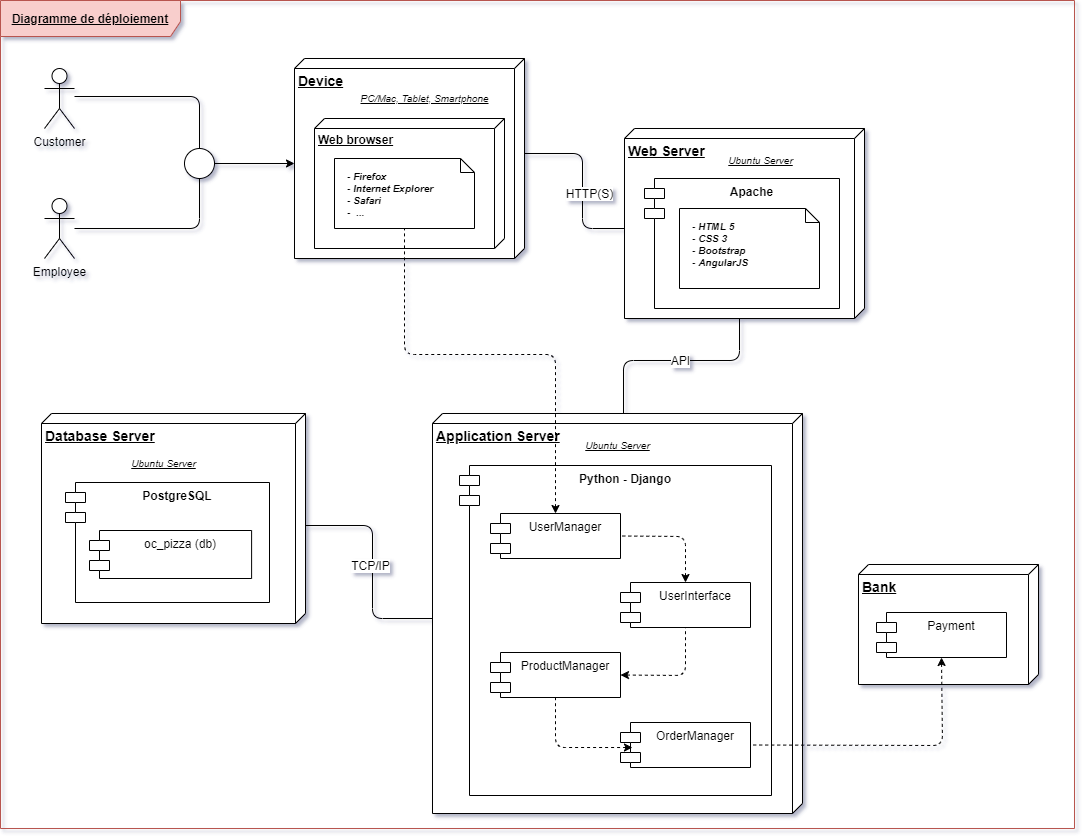
Cette partie illustre l’authentification d’un utilisateur. On peut y voir notamment le moment où l’utilisateur est défini comme étant un employé ou un client « **ManageUser** ».

* **Employee** : Nous voyons ici la relation entre « **:Employee** » et « **:Reminder** » qui lui se trouve dans « **ProductManager** »
* **Customer**: Nous mettons, ici, en évidence la relation qu’il existe entre « **:Customer** » et « **:Order** » (situé dans « **OrderManager** »).

## ShoppingCart :

Cette dernière partie met en évidence les composants qui entrent en jeu lors du processus de commande. Nous pouvons voir que pour accéder au panier « **:ShoppingCart** », il est nécessaire que l’utilisateur soit authentifié « **:Authentication** ».

Il existe également une relation avec « **:Order** » se trouvant dans « **OrderManager** ». Celui-ci fournit ensuite les informations au composant « **:Bill** » chargé de la partie facturation et qui fournit, lui-même, les informations de paiement au système bancaire. Illustré ici par le composant « **:Payment** » dans structure externe « **Bank** ».



# Diagramme de déploiement

Ce diagramme illustre la façon dont les éléments (nécessaires au bon fonctionnement de l’application) sont répartis et communiquent au sein de l’infrastructure.

A noter que le système d’exploitation installé sur les différents serveurs sera **Ubuntu Server**, basé sur **Debian**.

## User :

Nous pouvons voir les deux types d’utilisateurs (**Customer** et **Employee**), qui, pour interagir avec l’interface web, utilisent un périphérique (PC/Mac, tablette ou smartphone, l’application étant responsive). Par le biais d’un navigateur web (Firefox, Internet Explorer, Safari, Chrome …), ils communiquent avec le serveur web nommé ici « **Web Server** » en utilisant le protocole **HTTP(S)**.

## Web Server :

Le serveur Web est défini comme étant le « **Frontal** ». Il est celui qui héberge « **Apache http Server** » utilisé pour la partie site web du projet. Notamment grâce à **HTML 5**, **AngularJS (Framework JavaScript)**, **CSS** et **Bootstrap**. Il communique également par le protocole **TCP/IP** avec le serveur d’application.

Une API est également utilisée pour récupérer les données fournies par le serveur d’application « Application Server ». Ou, dans l’autre sens, fournir des données au serveur d’application.

## Application Server :

Le serveur d’application (**Application Server**) héberge lui la partie application. **Python**, ainsi que le Framework **Django** (nécessaires à l’application) sont installés sur ce serveur.

Concernant les relations, nous pouvons voir que l’utilisateur interagit avec l’élément « **UserManager**» notamment lors de l’authentification. Lui-même est lié à l’élément « **UserInterface** » qui est chargé d’afficher l’interface de l’utilisateur authentifié.

On peut également noter la liaison entre « **OrderManager** » et « **Payment** » nécessaire lors du paiement d’une commande à la banque qui est un élément externe à l’infrastructure.

Ce serveur communique également avec le serveur de base de données « **Database Server**» en utilisant le protocole **TCP/IP**

## Database Server :

Nous terminons avec le serveur hébergeant la base de données. Ce serveur communique très régulièrement avec le serveur d’application.

**MySQL** est installé sur ce serveur ainsi que **PhpMyAdmin / MySQL Workbench** pour administrer la base hébergée et nommée « **oc\_pizza** », qui contient elle-même toutes les tables nécessaires au fonctionnement de l’application.

A savoir que nous aurions pu choisir **PostgreSQL** comme **SGBDR** en lieu et place de **MySQL**. Nous utiliserions alors **PgAdmin 4** pour administrer la base.