Fichier technique

Contexte
- Objectif
- Cahier des charges

Description du domaine fonctionnel
- Diagramme de classe
- Tables
  - User
  - Group
  - Permission
  - Restaurant
  - Address
  - Command
  - Meal
  - Ingredient
  - Bill
  - Contact
- Modèle physique de données
- Schéma de déploiement

Contexte

« OC Pizza » est un jeune groupe de pizzeria en plein essor et spécialisé dans les pizzas livrées ou à emporter. Il compte déjà 5 points de vente et prévoit d'en ouvrir au moins 3 de plus d'ici la fin de l'année.

1. Objectif

Définir le domaine fonctionnel et à concevoir l'architecture technique de la solution répondant aux besoins du client, c'est-à-dire :

- modéliser les objets du domaine fonctionnel
- identifier les différents éléments composant le système à mettre en place et leurs interactions
- décrire le déploiement des différents composants que vous envisagez
- élaborer le schéma de la ou des bases de données que vous comptez créer

2. Cahier des charges

Un des responsables du groupe a demandé de mettre en place un système informatique, déployé dans toutes ses pizzerias et qui lui permettrait notamment :

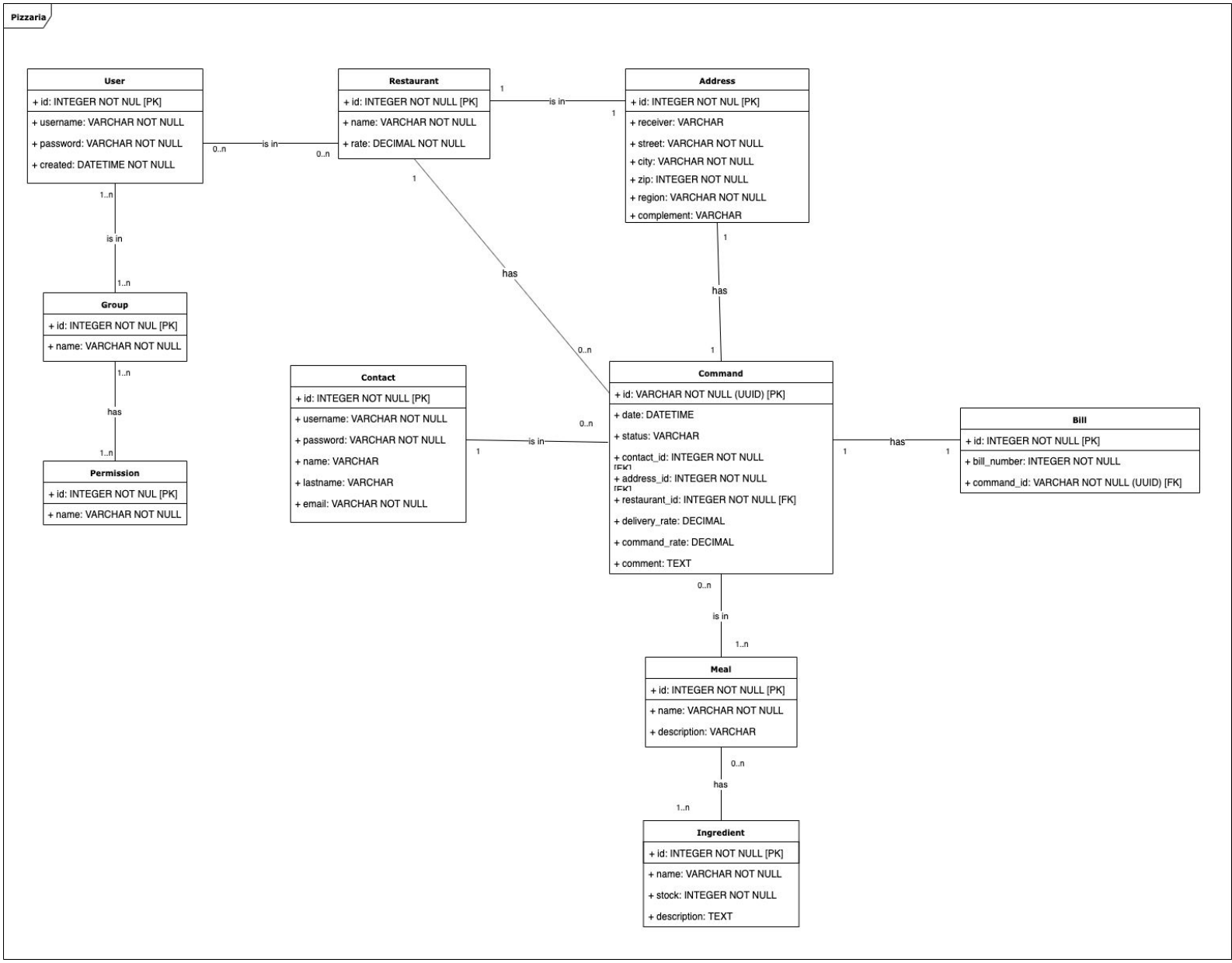
- d'être plus efficace dans la gestion des commandes, de leur réception à leur livraison en passant par leur préparation ;
- de suivre en temps réel les commandes passées et en préparation ;
- de suivre en temps réel le stock d'ingrédients restants pour savoir quelles pizzas sont encore réalisables ;
- de proposer un site Internet pour que les clients puissent :
    - passer leurs commandes, en plus de la prise de commande par téléphone ou sur place
    - payer en ligne leur commande s'ils le souhaitent – sinon, ils paieront directement à la livraison
    - modifier ou annuler leur commande tant que celle-ci n'a pas été préparée

- de proposer un aide mémoire aux pizzaiolos indiquant la recette de chaque pizza
- d'informer ou notifier les clients sur l'état de leur commande

Description du domaine fonctionnel

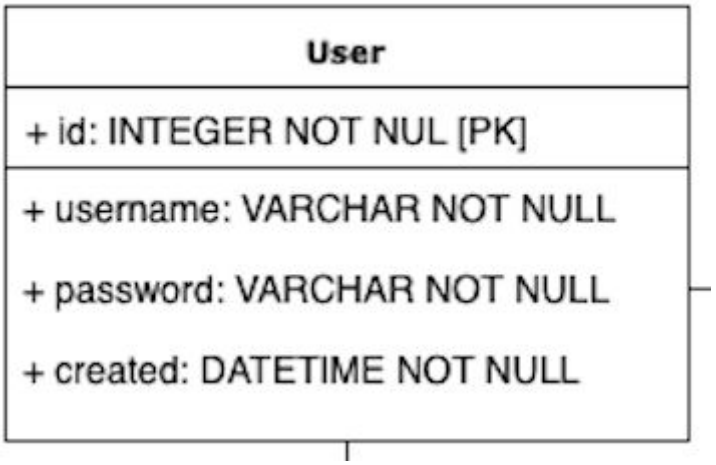
Nous allons ici décrire le domaine fonctionnel grâce au diagramme de classe en décrivant chaque table (*le diagramme de classe est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci).*

1. Diagramme de classe



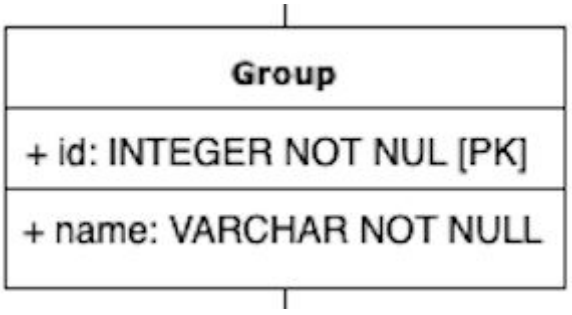
2. Tables

User



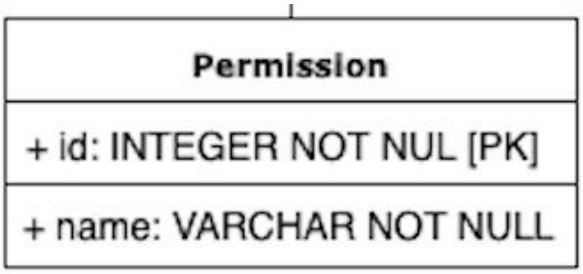
La table **User** contient tout le personnel administratif *(directeur, administrateur, pizzaiolo etc...)*.
Un **User** est lié à au moins un **restaurant** et est dans un **groupe**, ce qui permet de cloisonner les **User** et de gérer leurs **permissions**.

Group



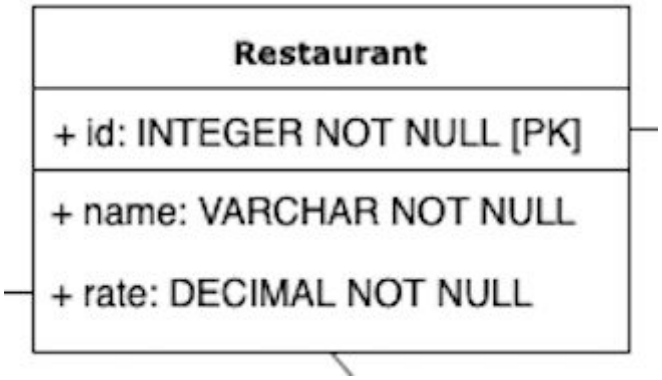
La table **Group** contient tous les groupes de **permission** *(administrateur pour autoriser les **User** qui y sont liés à ajouter des recettes et surveiller les commandes, livreur pour autoriser les **User** qui y sont liés à livrer une pizza etc...).*
Un **Group** contient des **permission**.

Permission



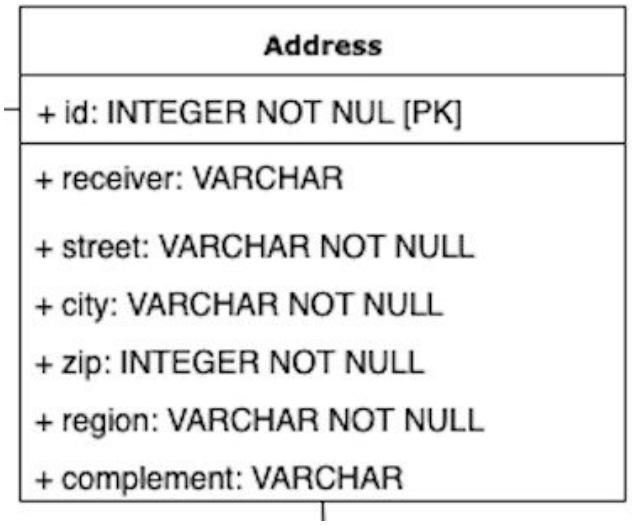
La table **Permission** regroupe toutes les permissions de sécurité.
Elles sont utilisées pour limiter l'accès à des composants ou fonctionnalités spécifiques de l'application *(permission de supprimer une recette, permission d'effectuer une livraison, permission de consulter les commandes en cours etc...).*
Une **Permission** est lié à un **Group**.

Restaurant



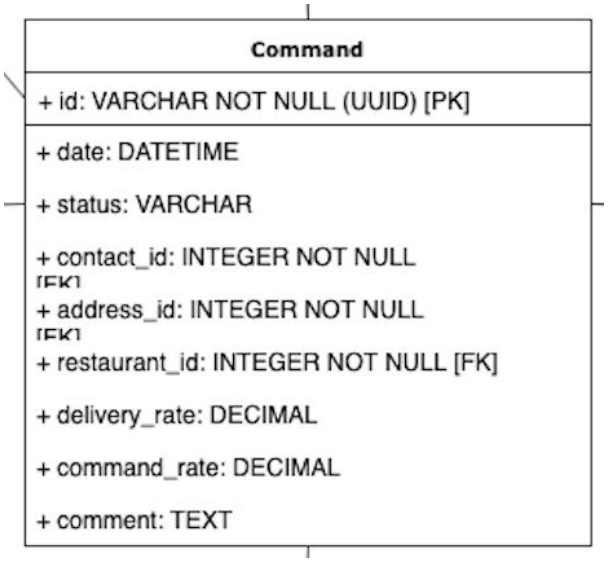
La table **Restaurant** contient de toutes les pizzeria.
Chaque pizzeria est noté grâce au champ **rate**, ce champ est modifié à la fin de la livraison si le client note sa livraison après l'avoir reçu.
Le **Restaurant** est lié aux tables : **User**, **Command** et **Address**.

Address



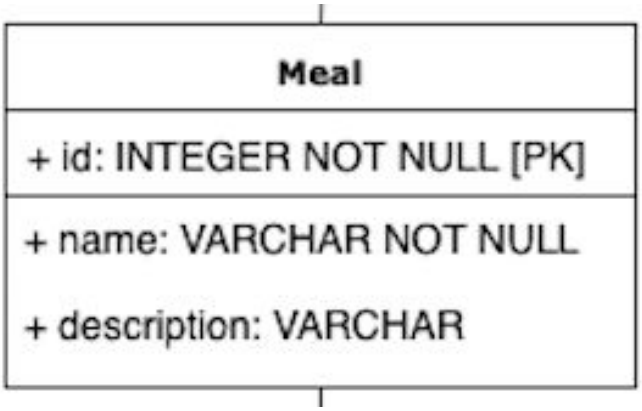
La table **Address** contient les adresses de chaque **Restaurant** ainsi que les adresses de chaque **Command**.

Command



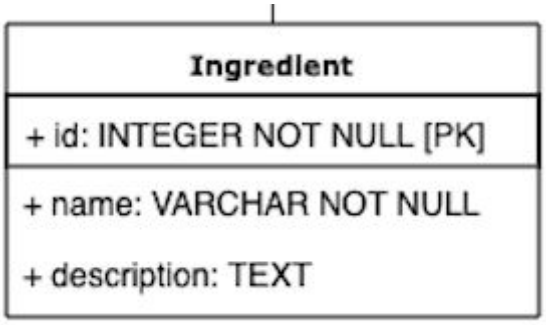
La table **Command** contient les commandes effectuées par les **Contact**.
Chaque instance de **Command** est liée à un **Contact** et à une **Address** *(pour la livraison)* ainsi qu'à un **Restaurant**.
Dans une **Command** se trouve un ou plusieurs **Meal** qui eux même sont composées d'**Ingredient**.
La **Command** est lié à une **Bill** *(*Facture*)* qui est généré au paiement de celle-ci.

Meal



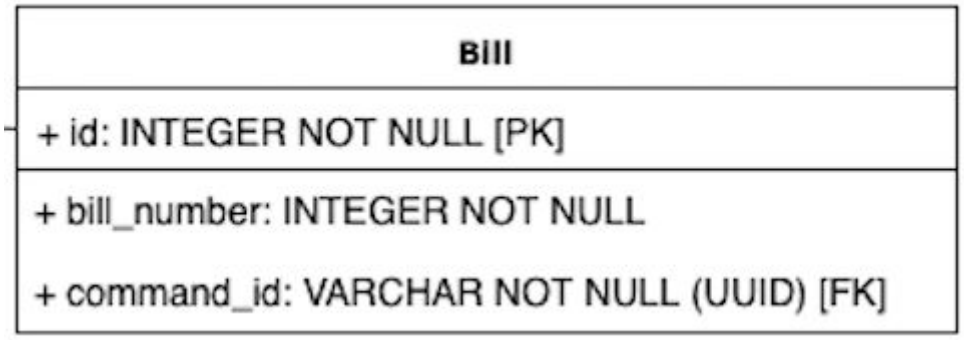
La table **Meal** contient tous les repas disponible dans le site chaque **Meal** est composé d'**Ingredient**.
Elle est accessible à un **Contact** pour qu'il effectue une **Command** mais aussi aux **User** qui sont dans le **Group** nommé cooker *(ce sont les cuisinier qui prépare les repas)*.

Ingredient



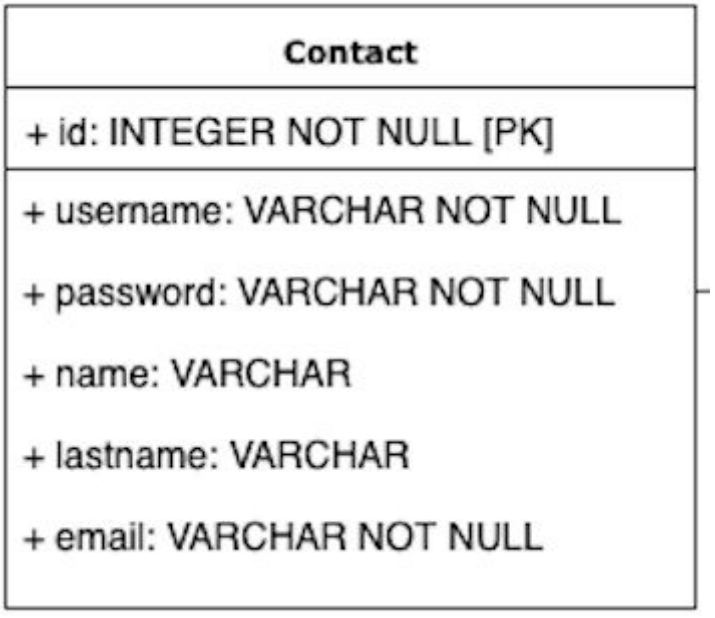
La table **Ingredient** contient tous les ingrédients nécessaires à la conception d'un **Meal**.
Elle permet aussi au **Contact** de voir quels sont les ingrédients qui compose son repas et peut demander à ne pas mettre certains ingrédients.

Bill



La table **Bill** permet de générer une facture qui sera envoyée à un **Contact** après le paiement d'une **Command**.

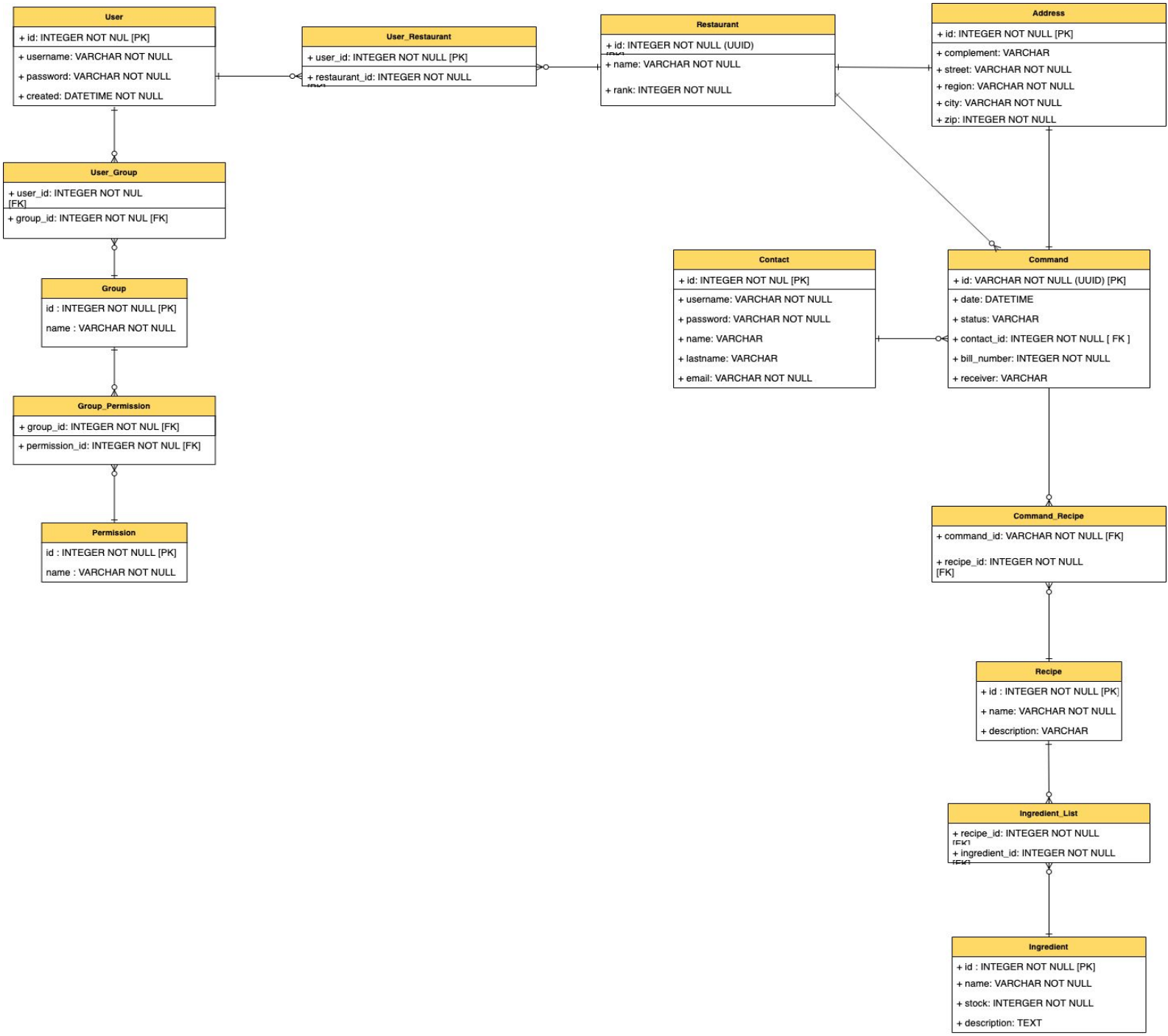
Contact



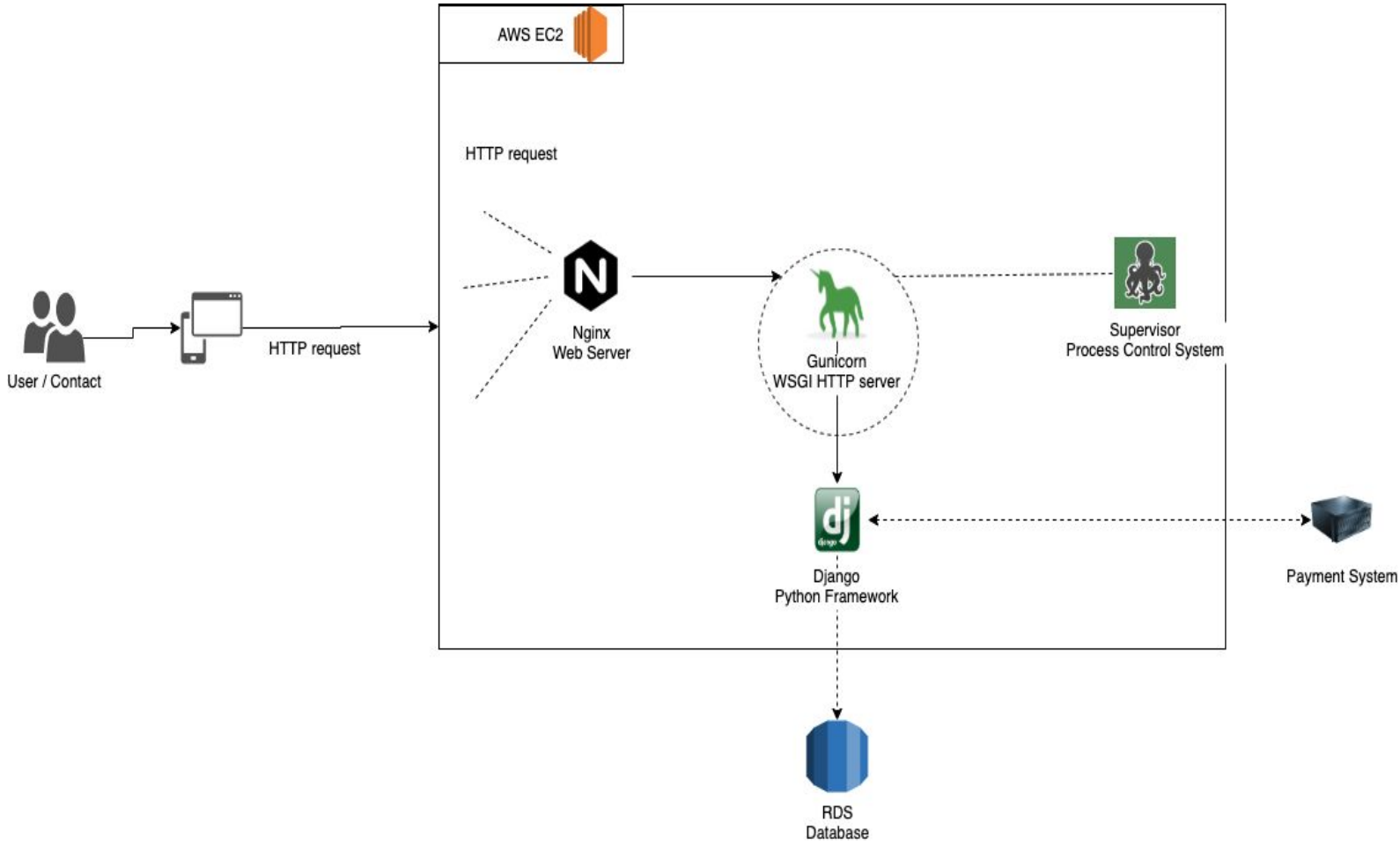
La table **Contact** contient tous les contacts du système.
Il peut effectuer une **Command** en choisissant un ou plusieurs **Meal**.
Il peut noter une **Command** après l'avoir reçu, ce qui affectera la note du restaurant et celle du **User** étant dans le **Group** livreur.

3. Modèle physique de données

Après avoir conçu le diagramme de classe présent ci-dessus j'ai pu créer ce modèle physique de données.



4. Schéma de déploiement



Le fonctionnement de l'application se déduit comme ca:

1. Le **User / Contact** envoie une requête pour payer sa commande.
2. Le serveur **Nginx** reçoit la requête et la transmet au serveur **WSGI Gunicorn**.
3. Le serveur **Gunicorn** transforme la requête **HTTP** en objet **python** et le transmet au framework **Django**.
4. **Django** fait appel au *service de paiement externe*.
5. *(dans le cas ou le paiement est acté)* **Django** lance la requête **SQL** d'enregistrement de la commande vers le service **AWS RDS**.
6. Le service **RDS** renvoie le résultat de la requête à **Django**.
7. **Django** renvoie la réponse à **Gunicorn.**
8. **Gunicorn** transforme l'objet **Python** en requête **HTTP** et la renvoie à **Nginx**.
9. La requete **HTTP** est renvoyé au **User / Contact**.

Pourquoi utiliser Amazon Web Service ?

Modèle tarifaire à la carte

Chaque service est "à la carte", c'est-à-dire que vous payez pour ce que vous utilisez.
C'est tout à fait logique pour l'infrastructure de serveur, car le trafic a tendance à être très dense, en particulier plus le site est grand.

Vitesse de déploiement

Les fournisseurs traditionnels prennent de 48 à 96 heures pour approvisionner un serveur. Ensuite, il faut passer quelques heures à le peaufiner et tout tester.
Amazon Web Service réduit ce temps de déploiement à quelques minutes.
Si vous utilisez leur **Amazon Machine Images** *(image d'un système d'exploitation)*, vous pouvez avoir une machine déployée et prête à accepter des connexions en un court laps de temps.
Ceci est important lorsque, par exemple, vous lancez une promotion qui génère des tonnes de trafic à des intervalles spécifiques, ou lorsque vous avez simplement besoin de flexibilité pour gérer la demande lorsqu'un nouveau produit est lancé.

Sécurité

AWS offre le même niveau de sécurité de classe mondiale aux startup et au très grosse entreprise. Les centres de données de l'entreprise maintiennent les normes les plus élevées, ce qui vous évite d'avoir à sécuriser vos propres installations.
AWS gère également "des dizaines de programmes de conformité dans son infrastructure" et un vaste réseau de soutien à la sécurité qui peut offrir une vue en temps réel sur les activités suspectes et les vulnérabilités potentielles.

Automatisation simple

Une autre des multiples raisons de choisir AWS est la possibilité de démarrer et d'arrêter diverses instances à des moments prédéterminés.
*(Exemple: la possibilité de programmer des services tels que **Elastic Compute Cloud** (**EC2**) et **Relational Database Service** (**RDS**) signifie qu'ils n'auront pas à fonctionner pendant les heures creuses ou les week-ends)*.