# RAPPORT SAE S4.A02.1
# Web - Backend (API Rest)


Thomas DUJARDIN

Simon BARBEAU

FÉVRIER 2023 - BUT2 INFORMATIQUE

# SOMMAIRE

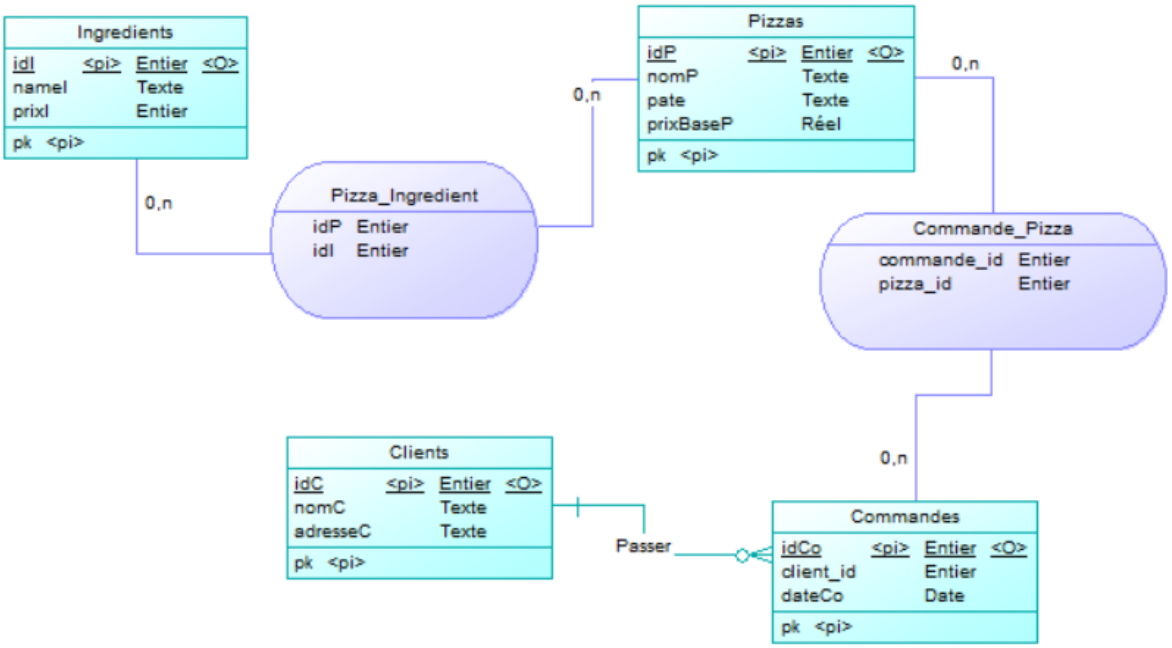
1) Modèle Conceptuel de Donnée (MCD)

2) Script SQL des créations de tables

3) DIAGRAMME DE CLASSE

4) Notre raisonnement

1) Voici notre MCD :



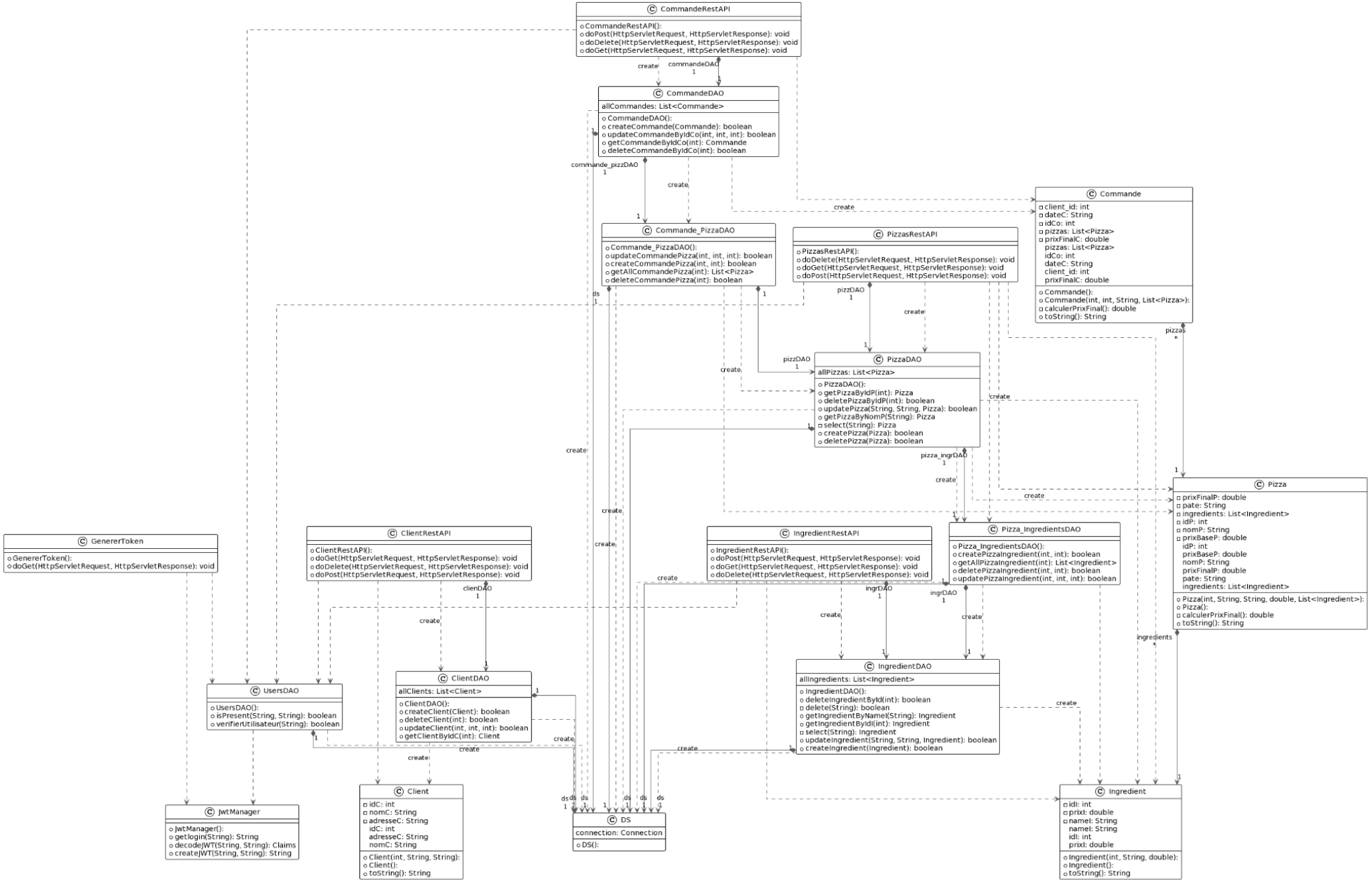

2) Voici notre script SQL :

```sql
DROP TABLE commande_pizza;
DROP TABLE commandes;
DROP TABLE clients;
DROP TABLE pizza_ingredients;
DROP TABLE pizzas;
DROP TABLE ingredients;
DROP TABLE users;

CREATE TABLE ingredients(
    idI int PRIMARY KEY,
    namei varchar(50),
    prixI decimal(10,2)
);

INSERT INTO ingredients VALUES (1,'pomme de terre', 1.50);
INSERT INTO ingredients VALUES (2,'poivrons', 1);
INSERT INTO ingredients VALUES (3,'poulet', 1);
INSERT INTO ingredients VALUES (4,'lardons', 1);
INSERT INTO ingredients VALUES (5,'oignon', 1);
INSERT INTO ingredients VALUES (6,'champignons', 1);
INSERT INTO ingredients VALUES (7,'mozzarella', 1);
```

```sql
INSERT INTO ingredients VALUES (8,'compté', 1);
INSERT INTO ingredients VALUES (9,'cheddar', 1);
INSERT INTO ingredients VALUES (10,'gorgonzola', 1);
INSERT INTO ingredients VALUES (11,'reblochon', 1);


CREATE TABLE pizzas(
    idP int PRIMARY KEY ,
    nomP varchar(50),
    pate varchar(100),
    prixBaseP decimal(10,2)
);

INSERT INTO pizzas VALUES (1,'4 fromages', 'new-yorkaise', 8.80);
INSERT INTO pizzas VALUES (2,'tartiflette', 'new-yorkaise', 8.80);
INSERT INTO pizzas VALUES (3,'barbecue', 'new-yorkaise', 8.80);

CREATE TABLE pizza_ingredients (
    pizza_id INT,
    ingredient_id INT,
    -- complexifie la chose quantite varchar(10),
    PRIMARY KEY (pizza_id, ingredient_id),
    FOREIGN KEY (pizza_id) REFERENCES pizzas(idP) on delete cascade,
    FOREIGN KEY (ingredient_id) REFERENCES ingredients(idI) on delete cascade
);

-- POUR LA PIZZA 4 FROMAGES ON AOUTE TOUT LES INGRE UN PAR UN
INSERT INTO pizza_ingredients VALUES (1, 7);
INSERT INTO pizza_ingredients VALUES (1, 8);
INSERT INTO pizza_ingredients VALUES (1, 9);
INSERT INTO pizza_ingredients VALUES (1, 10);

-- POUR LA PIZZA TARTIFLETTE ON AOUTE TOUT LES INGRE UN PAR UN
INSERT INTO pizza_ingredients VALUES (2, 1);
INSERT INTO pizza_ingredients VALUES (2, 4);
INSERT INTO pizza_ingredients VALUES (2, 6);
INSERT INTO pizza_ingredients VALUES (2, 11);

-- POUR LA PIZZA BBQ ON AOUTE TOUT LES INGRE UN PAR UN
INSERT INTO pizza_ingredients VALUES (3, 3);
INSERT INTO pizza_ingredients VALUES (3, 5);
INSERT INTO pizza_ingredients VALUES (3, 8);


create table clients (
    idC int PRIMARY KEY,
    nomC varchar(50),
    adresseC varchar(50)
);

INSERT INTO clients values (1, 'Thomas', 'Nieppe');
INSERT INTO clients values (2, 'Simon', 'Lille');
```

```sql
create table commandes (
    idCo int PRIMARY KEY,
    client_id int,
    dateCo date,
    FOREIGN KEY (client_id) REFERENCES clients(idC) on delete cascade
);

INSERT INTO commandes values (1, 1, '2023-02-18');
INSERT INTO commandes values (2, 2, '2023-02-18');

create table commande_pizza (
    commande_id INT,
    pizza_id INT,
    PRIMARY KEY (commande_id, pizza_id),
    FOREIGN KEY (commande_id) REFERENCES commandes(idCo) on delete cascade,
    FOREIGN KEY (pizza_id) REFERENCES pizzas(idP) on delete cascade
);

 -- POUR LA COMMANDE DE THOMAS
INSERT INTO commande_pizza VALUES (1, 1);
INSERT INTO commande_pizza VALUES (1, 3);

 -- POUR LA COMMANDE DE SIMON
INSERT INTO commande_pizza VALUES (2, 2);

create table users (
    user_id INT,
    login varchar(20),
    pwd varchar(20)
);

INSERT INTO users values (1,'simon','simon');
INSERT INTO users values (2,'thomas','thomas');
```

3) Voici notre UML : Accessible sur le lien ci : [Diagramme de classe]



## 4) Notre raisonnement

a) Nous avons créé 4 DTO qui créent un pojo de chaque table sql et qui permettent de passer du réel à un objet java pour le manipuler.

   Pour chaque DTO, nous avons retranscrit le contenu de sa table sql correspondante (créer des attributs, un constructeur et ses getters/setters)

b) Nous avons ensuite créés les DAO qui ont pour rôle de manager les pojos en se connectant à la base de donnée (chaque DAO respecte le CRUD : create, read, update and delete en implémentant chacune de ses méthodes)

c) Enfin, nous avons créé autant de contrôleurs que de DAO. Ils permettent de faire fonctionner l'API (de retourner les données).

   Nous pouvons faire des gets, des posts, des deletes.

Toutes les requetes ont été testé et ont été exporté dans un fichier texte disponible sur notre gitlab : (https://gitlab.univ-lille.fr/thomas.dujardin2.etu/pizzaland/-/tree/main/REST)

# Détails des contrôleurs API :

- Le contrôleur ingredientRestAPI accessible par ***/ingredients/***
  - GET :
    - ***/ingredients*** = retourne liste de toute la collection
    - ***/ingredients/{id}*** = retourne ingrédient identifié
    - ***/ingredients/{id}/name*** = retourne le nom de l'ingrédient identifié
  - POST :
    - ***/ingredients/*** = ajouter un ingrédient
  - DELETE :
    - ***/ingredients/{id}*** = supprime ingredient

- Le contrôleur pizzaRestAPI accessible par ***/pizzas/***
    - GET :
        - ***/pizzas/*** = retourne liste de toute la collection
        - ***/pizzas/{id}*** = retourne pizza identifié avec ses ingrédients
        - ***/pizzas/{id}/prixfinal*** = retourne le prix final de la pizza identifié
    - POST :
        - ***/pizzas*** = supprime une pizza
        - ***/pizzas/{id}*** = ajout d'un ingrédient à la pizza identifié
    - DELETE :
        - ***/pizzas/id*** = supprime une pizza
        - ***/pizzas/{id}/{idIngredient}*** = supprime l'ingrédient identifié dans la pizza identifié
    - PATCH :
        - ***/pizzas/id*** = modifie l'attribut de la pizza identifié

- Le contrôleur commandeRestAPI accessible par ***/commandes/***
    - GET :
        - ***/commandes*** = retourne liste de toute la collection
        - ***/commandes/{id}*** = retourne le détail de la commande identifié
        - ***/commandes/{id}/prixfinal*** = retourne le prix final de la commande identifié
    - POST :
        - ***/commandes*** = ajoute une nouvelle commande

Bonus :

- Le contrôleur clientRestAPI accessible par ***/client/***
    - GET :
        - ***/clients*** = retourne liste de toute la collection
    - POST :
        - ***/clients/*** = ajouter un client
    - DELETE :
        - ***/clients/{id}*** = supprime le client identifié