Ecole Nationale de la Statistique et de l'Analyse de l'Information

Rapport Final

Élèves :

GHOUMARI Kiyan
GUEYE Tidiane Pape
MOULIN Valentine
RANDRIAMANANA Nathan
ZOULKARNAYNI Nikiema

Encadrant :

ENEMAN Donatien

Responsable :

PEPIN Rémi

Année 2020-2021

Table des matières

Table des figures

Introduction

Cuisiner peut parfois devenir une tâche difficile lorsque l'on se trouve à court d'idées ou lorsque nous sommes pris à plein temps sur un projet passionnant. C'est la raison pour laquelle les élèves de l'ENSAI se ruent pour réserver les mets concoctés par l'association la plus populaire de l'École : l'EJR. Une demande importante face à une offre limitée. Telle est la recette à succès de cette incroyable association qui a su innover sur la composition de ces plats pour plaire à une large clientèle.

Cependant, une association avec tant de potentiel a besoin de moyens adaptés pour s'agrandir et répondre à la demande. La situation sanitaire a notamment mis à mal les caisses de l'EJR. Elle a donc besoin d'une solution technologique innovante qui lui permettra de renflouer ses caisses. Et ce, même pendant les heures de fermeture de l'association. Qui n'a jamais eu recours à Uber Eats face à la faible offre de restaurants sur le campus de Ker Lann. Il est temps pour l'association de prendre le monopole de la restauration des ENSAIENS.

Dans ce projet, nous allons vous présenter "EnsaEats", notre application qui met en relation les restaurateurs et leurs clients. Elle se base sur les données des restaurants fournis par l'API de *YELP*, un site web où les utilisateurs peuvent soumettre un avis portant sur les produits ou services d'établissements locaux, tels que des restaurants ou des écoles. Outre la possibilité d'expérimenter la création d'une plateforme pour renouveler l'association EJR, ce projet informatique a aussi pour but de nous donner les compétences nécessaires pour déployer et gérer une API : des compétences pouvant être appréciées aussi bien dans le secteur public que privé.

Chapitre 1

Organisation

Nous sommes une équipe de cinq personnes provenant de filières diverses et n'ayant jamais travaillé ensemble. Pour assurer une bonne synergie de groupe, le respect des échéances de rendu et une répartition du travail satisfaisante pour tous, nous nous sommes efforcés de nous organiser le plus clairement et efficacement possible.

1.1 Cahier des charges

Le cahier des charges est explicité clairement dans la présentation du sujet "EnsaEats". Il est composé d'une partie obligatoire qui consiste à implémenter les fonctionnalités de base de l'application. Selon le temps qu'il nous reste, nous pourrons également traiter des fonctionnalités avancées. Compte tenu du temps dont nous disposons, un système de recommandation nous semble très ambitieux pour ce projet.

Les fonctionnalités que nous avons implantés pour notre application sont les suivantes :

— Elle permet à un client de pouvoir chercher des restaurants (possibilité de filtrer selon la spécialité, selon la localité ou selon le nom du restaurant) ;
— Elle permet à un client de pouvoir consulter les détails d'un restaurant ;
— Elle permet à un client de pouvoir passer une commande ;
— Elle permet à un client de pouvoir donner un avis sur un restaurant ;
— Elle permet à un client de pouvoir ajouter,modifier,supprimer des plats ou menus d'un restaurant ;
— Effectuer des tests unitaires ;
— Une interface console simple à destination des clients qui utilisent l'API client ;

1.2 Planning

Pour mieux s'organiser, nous avons réalisé un diagramme de GANTT (figure 1) afin de s'assurer de la continuité des tâches à réaliser jusqu'au rendu du rapport d'analyse. Nous avons découpé le diagramme de Gantt en trois phases :

— Une phase d'analyse jusqu'au rendu du rapport intermédiaire
— Une phase de développement jusqu'au rendu du rapport final
— Une phase de préparation pour la soutenance

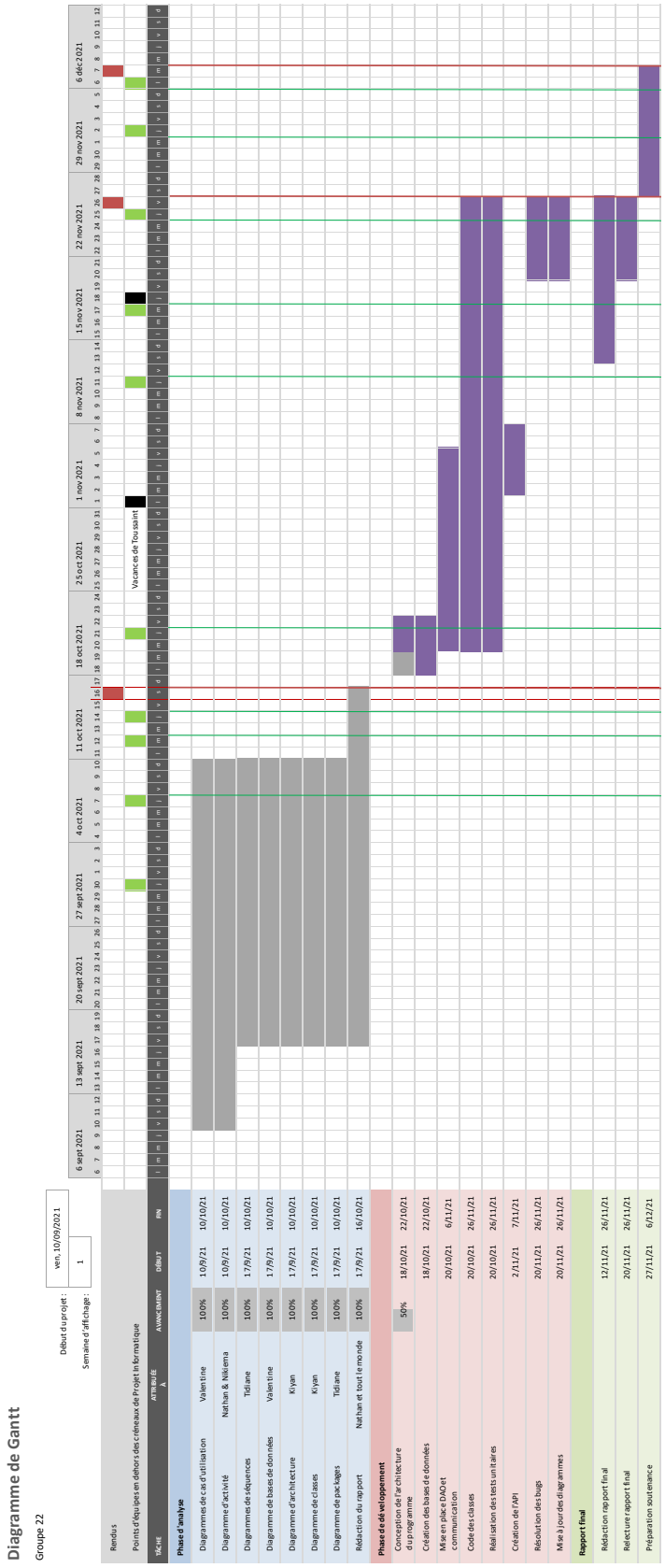

FIGURE 1 – Diagramme de GANTT

1.2.1 Organisation générale

D'une part, des séances de suivi encadrées par notre tuteur ont lieu de manière hebdomadaire. Cela nous permet de mieux comprendre les attendus du projet ainsi que certains points que nous n'avons pas compris des cours ou travaux pratiques complémentaires d'informatique.

D'autre part, dans l'optique d'entretenir une discipline et un rythme de travail régulier, nous avons convenu de réserver un créneau d'une à deux heures par semaine pour faire le point tous ensemble. Les importantes différences d'emploi du temps au sein du groupe nous poussent à mettre en place un vote hebdomadaire. Pour le travail individuel, nous avons considéré qu'un travail hebdomadaire de minimum 3 heures était nécessaire. Pour communiquer entre nous nous avons utilisé Teams que ce soit par appel ou par message.

1.2.2 Modélisation

Dès l'annonce de la composition des groupes, nous avions commencé à échanger sur les sujets qui nous intéressaient avant de conclure sur un vote cumulatif. À la suite du premier suivi, nous avions travaillé sur une première version du diagramme de GANTT, de cas d'utilisation et d'activité. En parallèle, nous avions effectué des recherches personnelles pour essayer de comprendre davantage ce qu'était une API.

La modélisation est une étape fondamentale en tant que point de départ de notre projet. Elle s'appuie sur la construction de diagrammes UML qui permettent de définir la structure globale de l'application ainsi que les besoins de l'utilisateur. Tout d'abord, nous disposons du diagramme de cas d'utilisation qui résume l'ensemble des tâches assurées par l'application. Ensuite, le diagramme d'activité permet de résumer séquentiellement les principales fonctionnalités de celle-ci. Ayant une portée plus pratique, le diagrammes de classes et de bases de données définissent l'architecture en vue de notre implémentation.

1.2.3 Répartition des tâches

Ce projet nous a incité à élire un chef de projet ou de planning qui a un rôle d'encadrant et doit s'assurer du bon déroulement du projet. Nous avons aussi désigné un chef technique qui s'occupe d'aider les membres du groupes sur des questions d'ordre technique (déboguage, efficacité du code, gestion des classes complexes, astuces de code). Le chef de projet réputé pour sa capacité d'organisation a été rapidement désigné. Tandis que le chef technique s'est naturellement imposé au fil du projet.

Les chefs désignés :
* Chef de projet : Valentine MOULIN
* Chef technique : Nathan RANDRIAMANANA

Au début, nous ne connaissions pas les compétences ou expériences de chacun en programmation orienté objet. Dans un soucis d'efficacité, il nous a donc fallu répartir les tâches de la manière la plus équitable possible en fonction des préférences de chacun. Les tâches de modélisation ont été réparties de la manière suivante :

* **Valentine** : Diagramme de Gantt, Diagrammes de cas d'utilisation, diagramme de base de données
* **Kiyan** : Diagramme d'architecture, diagramme de classes
* **Nikiema** et **Nathan** : Diagramme d'activité
* **Nathan** : Rédaction du rapport d'analyse
* **Tidiane** : Diagramme de séquences, diagramme de packages

Concernant la programmation, courant de la première semaine de codage il fut compliqué de se répartir les tâches car nous n'avions pas vraiment d'idées de par quoi commencer, et la suite logique. Ainsi, pendant les deux premières semaines beaucoup de tâches ont été faites par plusieurs d'entre nous. Ce qui fut une perte énorme de temps et d'efficacité.

1.2.4 Gestion du partage des fichiers

Pour une gestion efficace du projet et de notre avancée, nous avons ouvert un compte Gitlab afin de partager les différents diagrammes. Cette initiative nous a aussi permis de convenir sur

une manière efficace de partager des fichiers lorsque nous étions amenés à coder. Pour la plupart des diagrammes, nous avons utilisé l'extension "drawio" de l'environnement VScode pour concevoir efficacement nos diagrammes. Pour mettre à jour les changements des autres diagrammes en partage, nous avons utilisé les fonctionnalités classiques de Gitlab "push" et "pull".

1.3 Outils mis à disposition

Afin de construire notre API, nous disposons de plusieurs outils accessibles grâce au langage de programmation *Python*.

1.3.1 YELP

Fondé en 2004 par d'ex-employés de Paypal, YELP est une multinationale publiant des avis participatifs sur des entreprises. Elle dispose d'une API donnant un accès gratuit à des données spécifiques relatives à plusieurs restaurants localisés dans 32 pays. L'API dispose de plusieurs « End Point » (paramètre de recherches) permettant de trouver des informations selon des critères très diversifiées (Ex : Recherche par nom ; Recherche par numéro de téléphone ; Recherche par transaction [1] ; etc.).

Les données commerciales sur les restaurants concernent, entre autre, le nom, l'adresse, le numéro de téléphone, les photos, l'évaluation Yelp, les niveaux des prix et les heures d'ouverture.

1.3.2 FastAPI

Le FastAPI est un framework python permettant de concevoir des API robustes, performantes et faciles à maintenir. Comme ce framework est basé sur OpenAPI, de nombreuses options sont disponibles. Il propose une documentation interactive. On peut tester directement l'API depuis notre navigateur. Ce framework permet de mettre en avance plusieurs points tels que :

— la rapidité
— la facilité d'apprentissage et d'adaptation
— la réduction de code dupliqué et génération automatique de documentation

Le framework guide le développeur sur les entrants/sortants des différentes étapes du code (de la base de données au rendu HTTP). Il nous permet de structurer et valider les données dans le code. Le framework permet aussi d'élaborer des tests unitaires simplifiés et compréhensibles. Nous pouvons utiliser une base de données aux tests unitaires afin de ne pas affecter la base principale. Ce framework nous servira à construire notre API.

1. Rechercher les restaurants qui acceptent la livraison de nourriture

Chapitre 2

Modélisation UML

La modélisation UML nous a conduit à réaliser les diagrammes suivants : d'architecture, de cas d'utilisation, d'activité, de classes, de packages, de base de données et de séquences.

2.1 Diagramme d'architecture

Le diagramme d'architecture permet d'avoir un point de vue plus concret de ce qui se passe aux niveaux des données. Tout d'abord, un utilisateur, que ce soit un client ou un restaurateur, transmet ses données personnelles à l'API lorsqu'il se connecte. L'API permet à des applications de communiquer entre elles et de s'échanger mutuellement des données et services. Ces données sont ensuite envoyées à notre base de données. L'API reçoit également des données, de YELP, concernant les restaurants (restaurant ouvert ou fermé par exemple) ainsi que des données de notre base de données qui peuvent ensuite être transmises aux utilisateurs. **Il est donc nécessaire de faire deux applications : une application client et notre API**. L'application cliente permettra de faire l'affichage pour le client alors que l'API fera réellement les traitements et communique avec notre base de données et l'API de Yelp.

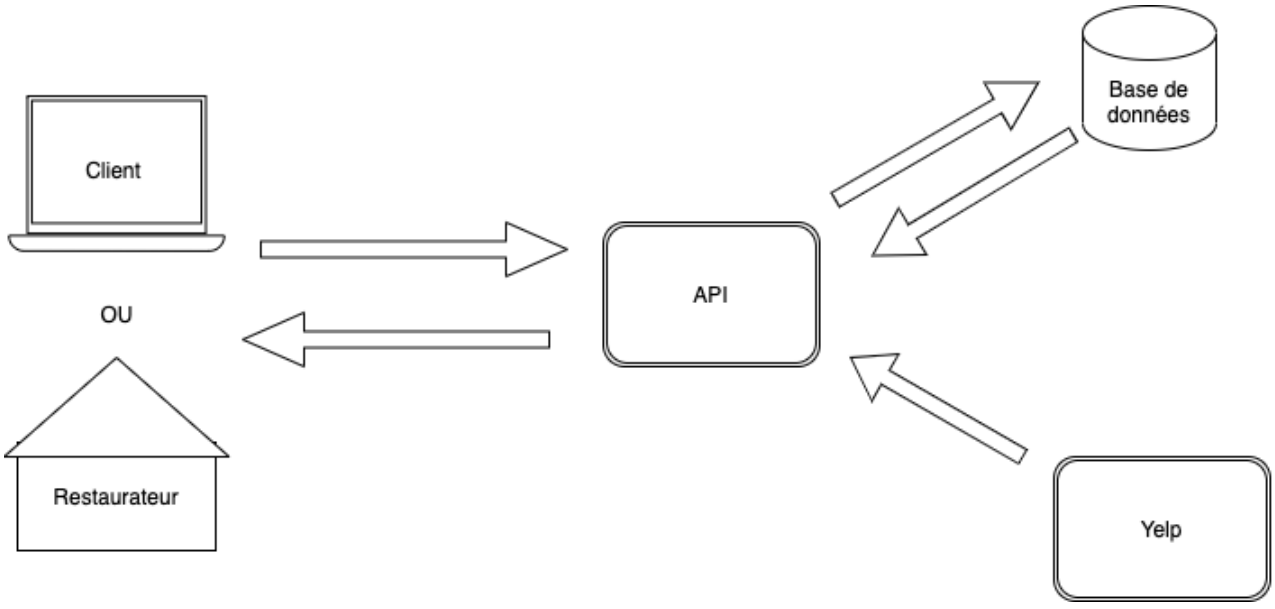

Figure 2 – Diagramme d'architecture

2.2 Diagramme de cas d'utilisation de l'application cliente

Le client est l'unique utilisateur de l'application cliente. Ainsi, en tant que client, celui-ci peut donc rechercher des restaurants autour de chez lui grâce à son adresse qu'il rentrera préalablement. S'il le souhaite, l'utilisateur peut filtrer selon une spécialité (italien, burger, japonais, etc..) ou par nom (Mcdonalds, Sushishop, etc..) si celui-ci a déjà choisi le restaurant dans lequel il souhaite commander. Le client peut aussi gérer son panier : ajouter un menu, supprimer un menu, modifier la quantité etc..

Nous n'avions finalement pas eu le temps d'implémenter le fait que le client puisse voir l'état de sa commande : 'en préparation', 'en livraison', 'livrée' et le fait que le client puisse consulter ses commandes.

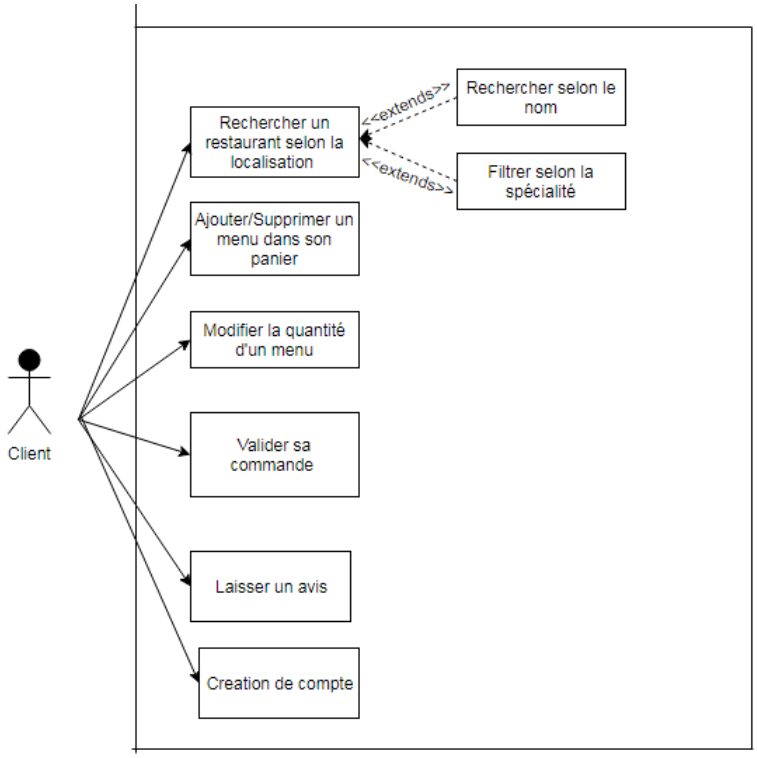

FIGURE 3 – Diagramme de cas d'utilisation de l'application cliente

2.3 Diagramme de cas d'utilisation de l'API

En tant que client, l'API va permettre à l'utilisateur de rechercher des restaurants la localisation du client, et de manière facultative une spécialité ou un nom de restaurant. Si celui-ci crée un panier et le valide (ref. diagramme d'activité du client), l'API enregistrera une commande uniquement lorsque le client aura complètement validé son panier. L'API ne prend donc pas en compte les ajouts/suppressions de menus dans un panier qui sont fait dans l'interface.

L'API permettra aussi au client de consulter toutes ses commandes. Cela peut-être utile s'il souhaite retrouver le nom du restaurant dans lequel il avait commandé, ou quel menu il avait commandé, etc.

Le restaurateur, contrairement au client, agit uniquement sur l'API grâce aux routers.

Dans un premier temps, si l'utilisateur n'a pas de compte restaurateur celui-ci doit s'en créer un pour avoir accès aux différentes fonctionnalités d'un restaurateur. Après quoi, il pourra modifier ses informations, récupérer ses informations ou tout bonnement supprimer son compte.

En tant que restaurateur, l'API lui permettra de modifier la carte de son restaurant (ref. diagramme de séquence du restaurateur) que ce soit par l'ajout ou suppression d'un menu, l'ajout ou suppression d'un article, modification d'un menu ou d'un article, etc.. Il pourra aussi consulter la liste de ses commandes. Il pourra de plus consulter l'avis de son restaurant mais aussi des autres restaurants.

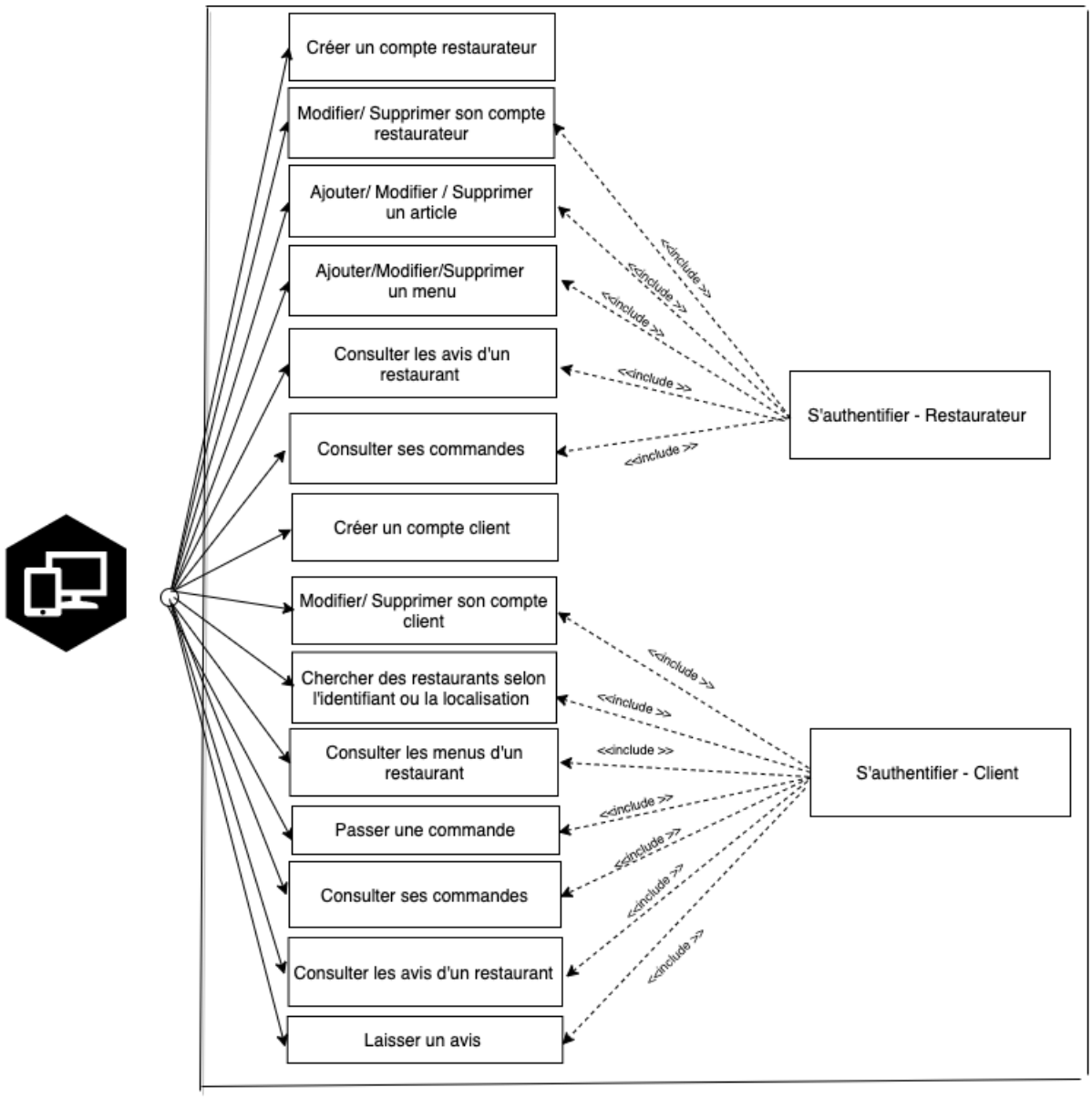

FIGURE 4 – Diagramme de cas d'utilisation de l'API

2.4 Diagramme d'activité de l'application cliente

Par souci de clarté, nous avons preféré couper en deux notre diagramme d'activité de l'application cliente. Ainsi, vous retrouverez un diagramme de l'utilisateur de l'entrée dans l'application au début de la commande, et un diagramme d'activité dédié à la création d'une commande.

Le diagramme d'activité de l'application cliente montre les différentes tâches que peut faire un client sur l'application. Lorsqu'il entre sur l'application, le client doit se connecter ou se créer un compte. À la suite de quoi, il aura le choix entre saisir une adresse ou prendre l'adresse indiquée lors de la création de son compte. S'il le souhaite, il peut entrer un rayon, ou un "term" permettant de filtrer par nom de restaurant ou spécialité.

Lorsqu'il sélectionne un restaurant, le client a le choix entre consulter les menus ou les avis. Si le client est satisfait par les menus (et par les avis) celui-ci peut faire une commande.

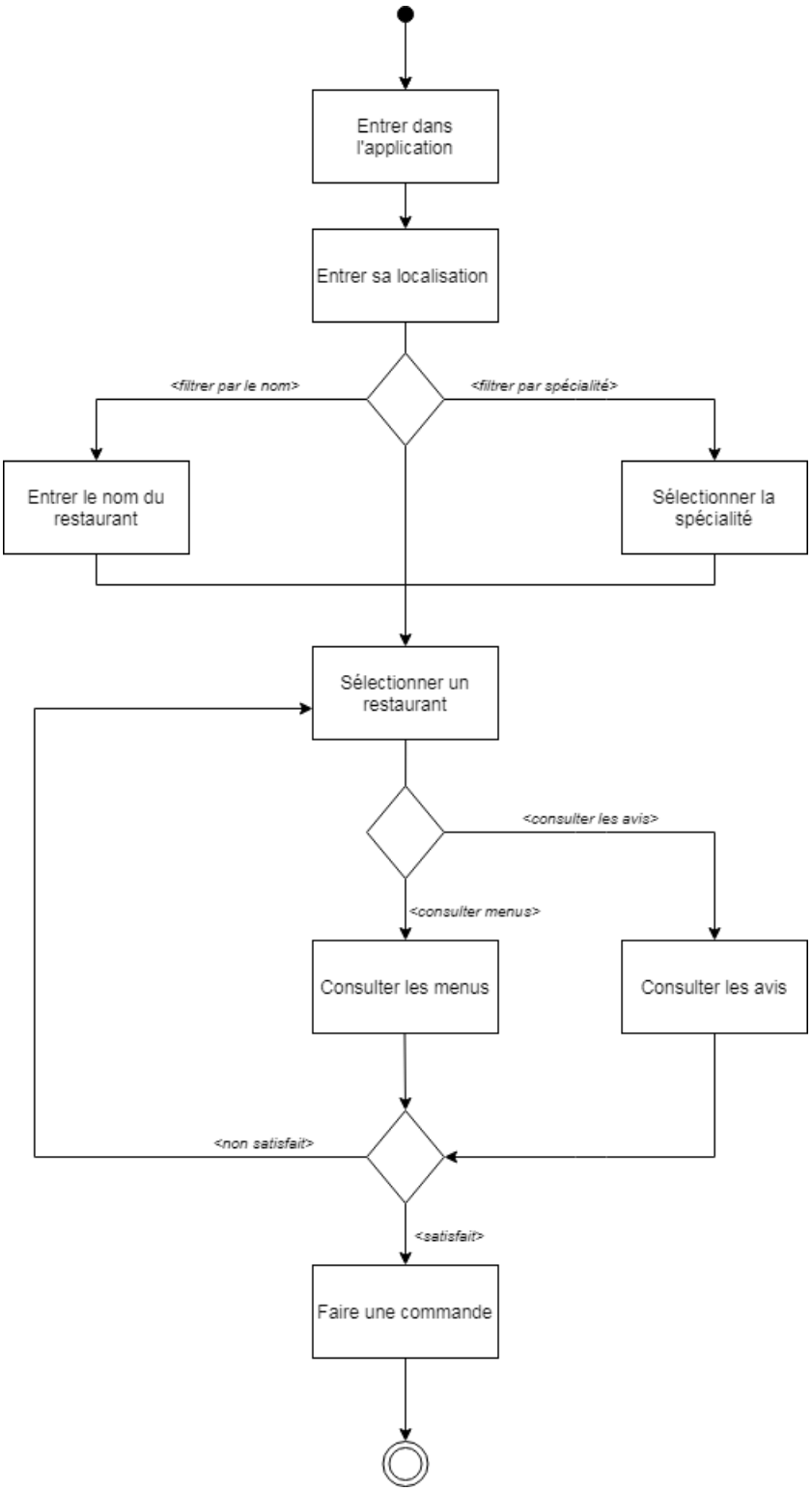

FIGURE 5 – Diagramme d'activité du client

Lorsque le client a selectionné un restaurant dans lequel il souhaite commander :

— Si son panier est vide le client sélectionne un menu : il aura accès à la composition du menu et au prix. Si le client est satisfait et souhaite l'ajouter au panier il n'aura qu'à indiquer la quantité qu'il souhaite et confirmer l'ajout du(ou des) menu(s) au panier. Le panier est ainsi mis à jour, et le prix total de la commande est actualisé ;
— Si son panier est non vide, le client peut consulter son panier et le modifier s'il le souhaite. Le panier est mis à jour ;

Lorsque le client considère avoir fini son panier, il valide sa commande. Une commande sera alors créée et enregistrée dans la base de données des commandes.

À savoir que le client a la possibilité à chaque étape de revenir en arrière, nous avons preféré ne pas faire figurer le retour en arrière à chaque étape par soucis de clarté. Ainsi, si un client a indiqué un nom de restaurant mais que finalement celui-ci ne lui plaît pas il peut tout à fait revenir en arrière et consulter la liste complète des restaurants, ou filtrer par spécialité ou indiquer un autre nom de restaurant. Ou encore s'il souhaite dans un premier temps consulter les avis d'un restaurant, il peut tout à fait revenir en arrière pour consulter les menus. Ou pour finir, si celui-ci a selectionné un restaurant, et à commencer un panier il peut tout de même changer de restaurant et son panier sera vidé instantanément.

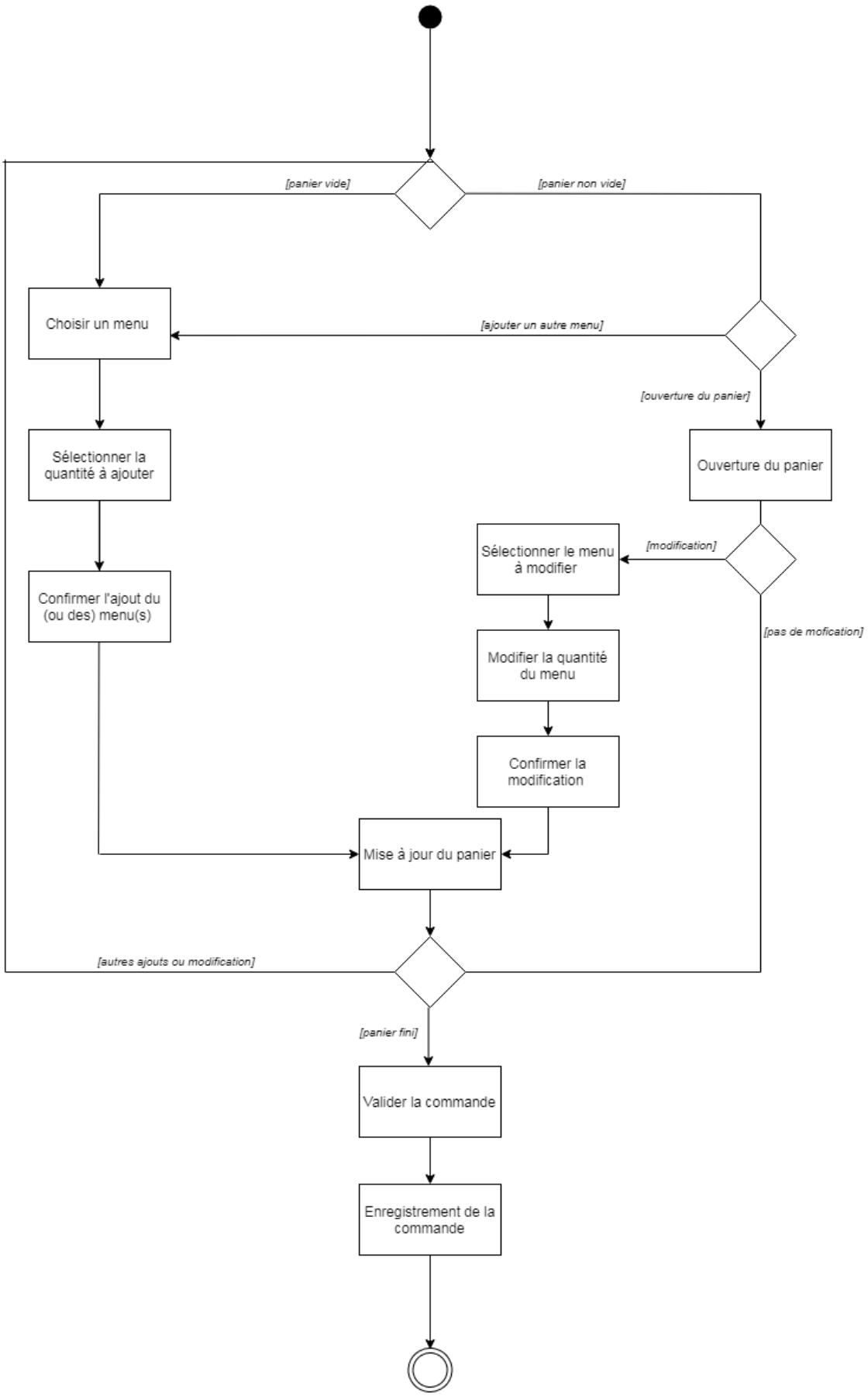

FIGURE 6 – Diagramme d'activité du client : création de commande

2.5 Diagramme de packages

Le diagramme de package ci-dessous montre les interactions entre les différents packages de l'application. D'abord, le package **controller** contiendra l'ensemble des modules permettant le fonctionnement de l'API client, laquelle communique directement avec l'utilisateur de l'application. Ce dossier fera appel au package **service** qui contient l'ensemble des méthodes permettant l'exécution des cas d'utilisation de l'API. De plus, nous avons le package **DAO** qui contient les éléments permettant de se connecter aux sources de données (la base de données SQL). Ensuite, l'application renferme un package **métier** qui contient l'ensemble des objets métier de l'application. Enfin, nous avons le package **API** qui renferme les méthodes permettant de communiquer avec **YELP**. Vis-à-vis de l'interface de nos programmes, nous distinguons le package **Controller** pour l'API et **View** qui permettra de faire l'affichage pour le client : par exemple, une view pour l'accueil ou pour le panier côté client et au niveau du package **Controller**, la gestion des requêtes côté API. C'est dans ce dernier que nous nous servirons du framework *FastAPI*

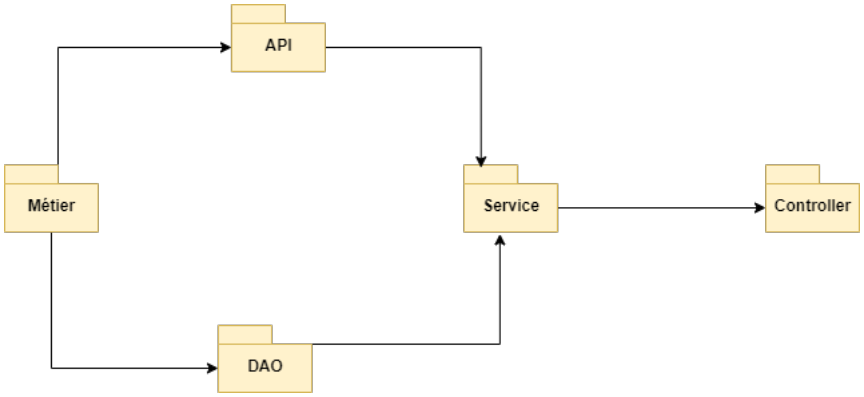

FIGURE 7 – Diagramme de packages de l'API

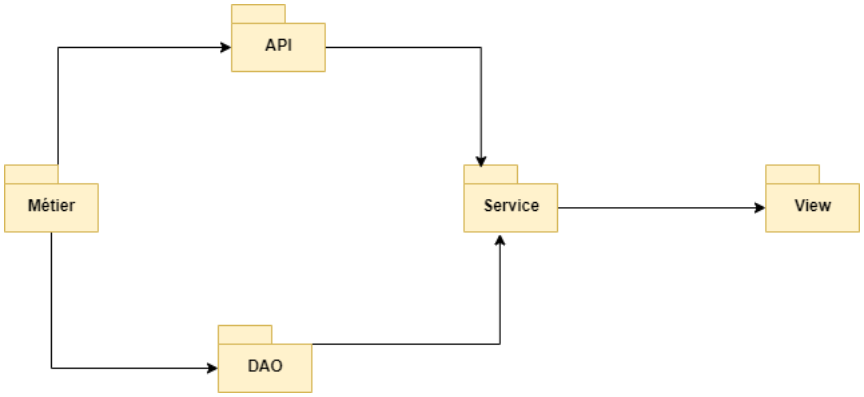

FIGURE 8 – Diagramme de packages du client

2.6 Diagramme de base de données

Pour le diagramme de base de données nous avons décidé de faire 11 tables dont 4 tables d'association. Pour commencer, nous distinguons un restaurateur d'un client.

Un restaurateur s'il possède un restaurant, il en possède un unique, inversement un restaurant pourra avoir un unique restaurateur. Les informations sur le restaurant seront actualisées grâce à l'API de YELP et ne sont pas stockées dans la base de données. Le restaurant peut posséder des avis qui seront anonymisés. Le restaurant aura ou non des commandes dans la table Commandes et proposera au minimum 1 menu à sa carte. Nous considérons le cas ici où les restaurants ne proposent uniquement que des menus : les menus étant composés de 3 éléments (plat, dessert, boisson). Le type de l'article sera référencé dans la base de données des articles.

Pour le client, la table réfère toutes les informations nécéssaires : son nom, prénom, adresse, son mot de passe utile pour l'identification, qui sera crypté, et son numéro de teléphone. Le client peut avoir des commandes.

Pour les commandes celles-ci détiennent un identifiant, une date de commande qui permettra au restaurateur de filtrer ses commandes, et un prix total. Une commande appartient à un unique client, et contient un ou des menus d'un unique restaurant.

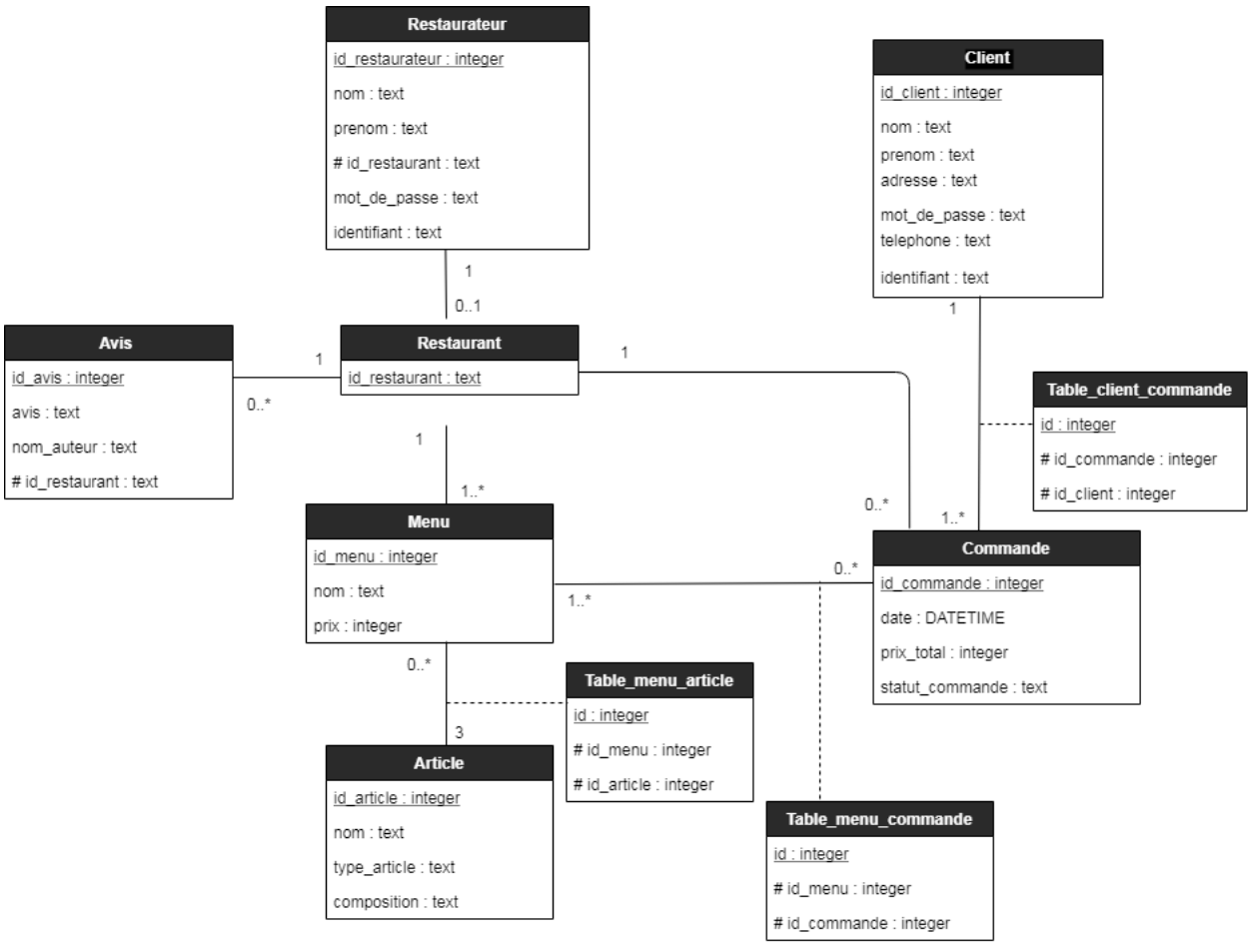

Figure 9 – Diagramme de base de données

Table	Variable	Type	Explication
Restaurateur	id_restaurateur	NUMERIC	Identifiant du restaurateur
	nom	TEXT	Nom de famille du restaurateur
	prénom	TEXT	Prenom du restaurateur
	identifiant	TEXT	identifiant du restaurateur
	mot_de_passe	TEXT	Mot de passe du restaurateur
	id_restaurant	TEXT	identifiant du restaurant du restaurateur
Restaurant	id_restaurant	TEXT	Identifiant du restaurant
Avis	avis	TEXT	Avis laissé par un client
	date	TEXT	date à laquelle l'avis est laissé par le client
	id_avis	NUMERIC	Identifiant de l'avis
	nom_auteur	TEXT	Nom du client qui laisse l'avis
	id_restaurant	TEXT	Identifiant du restaurant
Menu	id_menu	NUMERIC	Identifiant d'un menu
	nom	TEXT	Nom du menu
	prix	NUMERIC	Prix unitaire du menu
Article	id_article	NUMERIC	Identifiant d'un article
	nom	TEXT	Nom d'un article
	type_article	TEXT	Type d'article (plat, dessert ou boisson)
	composition	TEXT	Composition de l'article (ingrédients, allergènes,etc...)
Commande	id_commande	NUMERIC	Identifiant d'une commande
	date	DATETIME	Date et heure de la commande
	prix_total	NUMERIC	Prix total de la commande avec frais de livraison
	statut_commande	TEXT	Statut de la commande (enregistrée, en préparation, en livraison, livrée)
Client	id_client	NUMERIC	Identifiant du client
	nom	TEXT	Nom du client
	prenom	TEXT	Prénom du client
	adresse	ADRESSE	Adresse du client (numéro, rue, code, postal, ville)
	mot_de_passe	TEXT	Mot de passe du client crypté
	identifiant	TEXT	création d'un identifiant pour le client
	telephone	TEXT	Numéro de téléphone du client

Table 1 – Descriptif des variables de la base de données

2.7 Diagramme de séquences d'exécution d'une commande

Le diagramme de séquence ci-dessous présente, dans l'ordre vertical, la suite des opérations qui s'effectue lorsqu'un client recherche un restaurant et passe sa commande. En plus de la base de données des commandes et de l'API de YELP, ces deux cas d'utilisation font intervenir les deux grandes composantes suivantes :

— l'interface Client : la couche communiquant directement avec l'API. Elle affiche les resultats de la requête de l'utilisateur ;

— l'API : envoie des requêtes soit vers notre base de données, soit vers l'API de YELP et récupère les résultats. Ces derniers sont ensuite transmis au Client pour affichage grâce aux Views.

Dans ce diagramme, d'abord, il convient de noter que les requêtes de recherche de restaurant déclenchent une interrogation de l'API de YELP afin de s'assurer d'avoir les dernières informations (Position, Horaires) des restaurants. Ensuite, notre base de données enregistre les commandes effectuées.

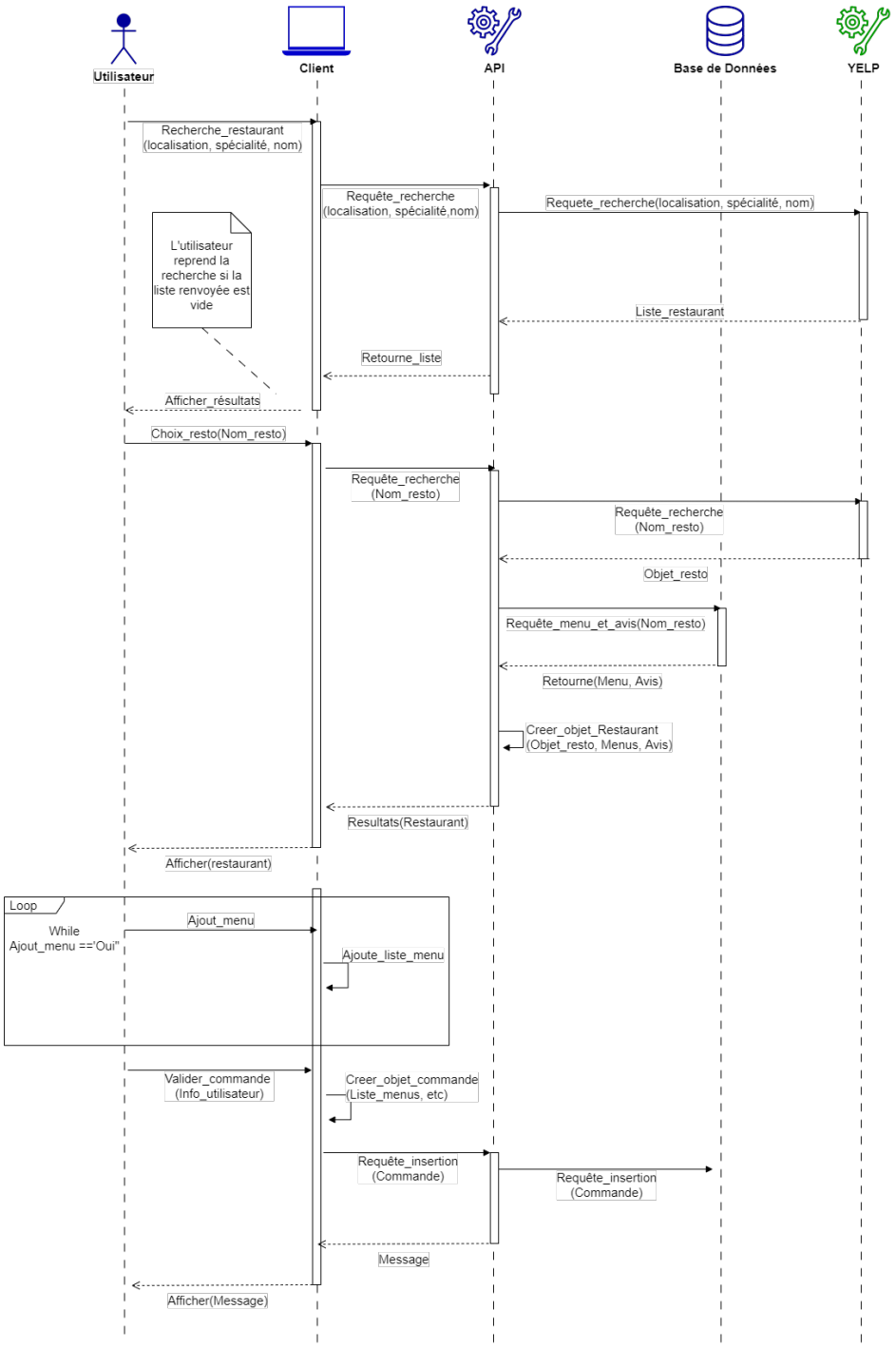

FIGURE 10 – Diagramme de séquences de l'exécution d'une commande

2.8 Diagramme de séquences du restaurateur

Plutôt que de présenter un diagramme d'activité pour le restaurateur, nous avons privilégié son diagramme de séquences. Cette figure précise les interactions entre le restaurant, l'application et la base de données lors d'une requête du restaurateur. Le restaurateur agit directement sur notre API (API métier comme expliqué dans la partie précédente) qui s'occupe des requêtes SQL sur notre base de données : la consultation, l'ajout, la suppression ainsi que la modification des menus de son restaurant. Les menus étant stockés dans notre base de données.

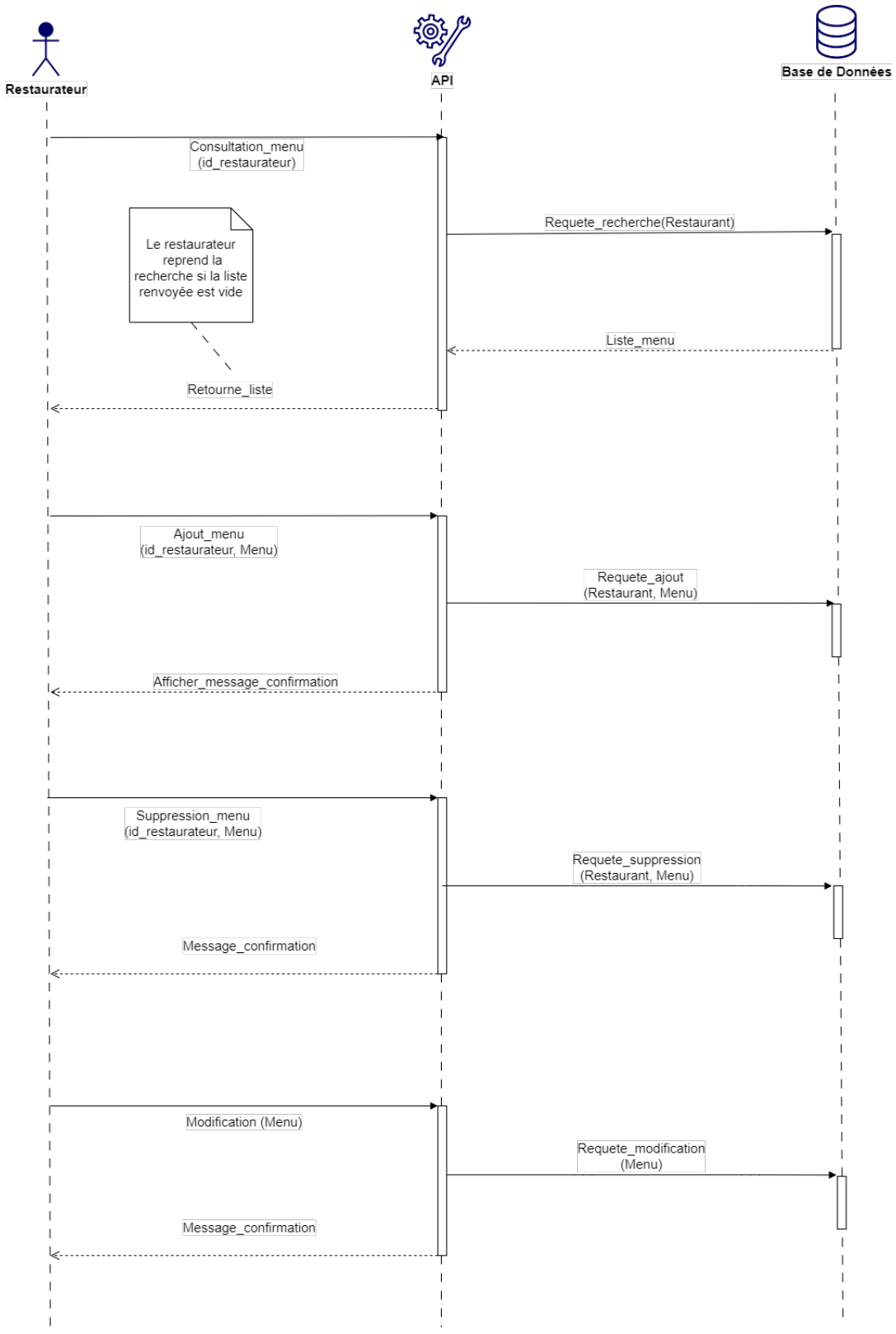

FIGURE 11 – Diagramme de séquences du restaurateur

2.9 Diagramme de classes

Nous avons décidé de scinder notre application en 8 classes :

— La classe **Client** : Un client possède un identifiant, un mot de passe et s'identifie par son nom, son prénom ainsi que son numéro de téléphone et son adress ;

— La classe **Restaurateur** : Un restaurateur possède un identifiant, un mot de passe et s'identifie par son nom, son prénom ainsi que son numéro de téléphone et son adresse. A cela s'ajoute, l'identifiant de son restaurant ;.
— La classe **Restaurant** : Un restaurant possède un identifiant, un nom, une adresse et un statut. On peut également voir si le restaurant est ouvert ou non grâce aux données de YELP à travers la variable statut ;
— La classe **Avis** : Les avis écrits par les clients possèdent un identifiant, une date et un identifiant permettant de connaêtre le client ;
— La classe **Menus** : Un menu est constitué d'un identifiant, d'un nom, d'un prix et de 3 articles. En effet, chaque menu se compose d'un plat, d'un dessert et d'une boisson obligatoirement ;
— La classe **Article** : Afin de reconnaître les articles des menus, à chacun est attribué un identifiant, un nom et son type (plat, dessert ou boisson). On ne lui donne pas de prix car on ne peut pas acheter un article seul mais les menus en ont un ;
— La classe **Adresse** : Une adresse dans notre application est composée du numero de la rue plus le nom de la rue, le code postale et le pays. Un client et un restaurant possèdent chacun une adresse. Le restaturateur quand à lui peut avoir plusieurs adresse. ;
— La classe **Commande** : Une commande possède un identifiant, la date à laquelle la commande a été réalisée, la liste des menus. Cette classe permet de voir le statut de la commande (livrée, en cours de livraison. . .), ainsi que le prix total (ou prix du menu plus celui de la livraison) de la commande. Elle comporte également une méthods permettant aux restaurateurs d'ajouter ou enlever des menus.

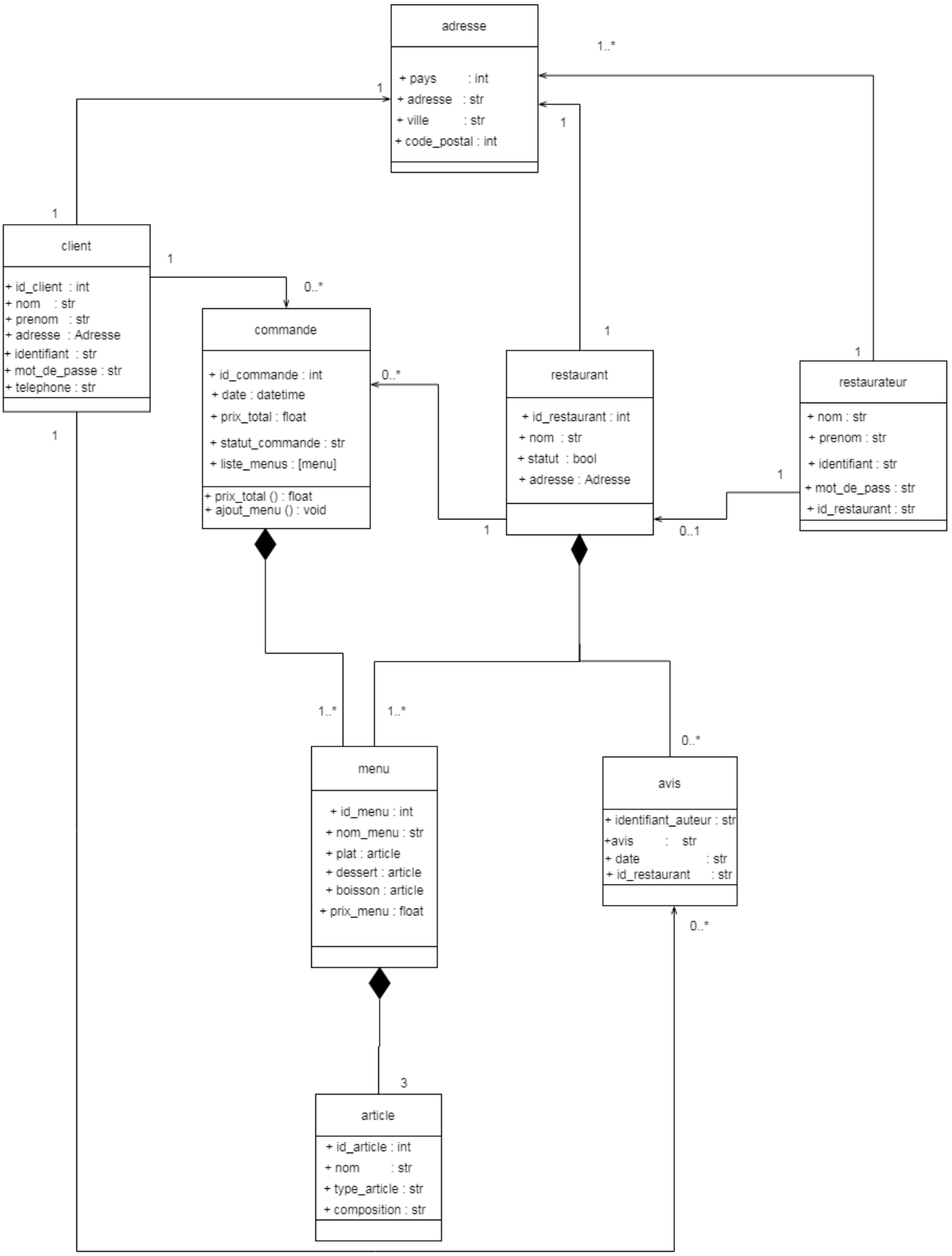

FIGURE 12 – Diagramme de classes

.

Chapitre 3

Modélisation de données

Le modèle conceptuel des données nous a permis d'exprimer une vue d'ensemble de la base de données et de mettre en avant les relations qui unissent les différentes entités. Une fois ce modèle conceptuel réalisé, il faut trouver un moyen de convertir les associations en des entités. En effet, pour pouvoir coder notre base de données sous formes de requêtes SQL, nous devons représenter notre base de données à l'aide d'un autre modèle plus propice : il nous faut un modèle logique des données.

3.1 Modèle logique des données

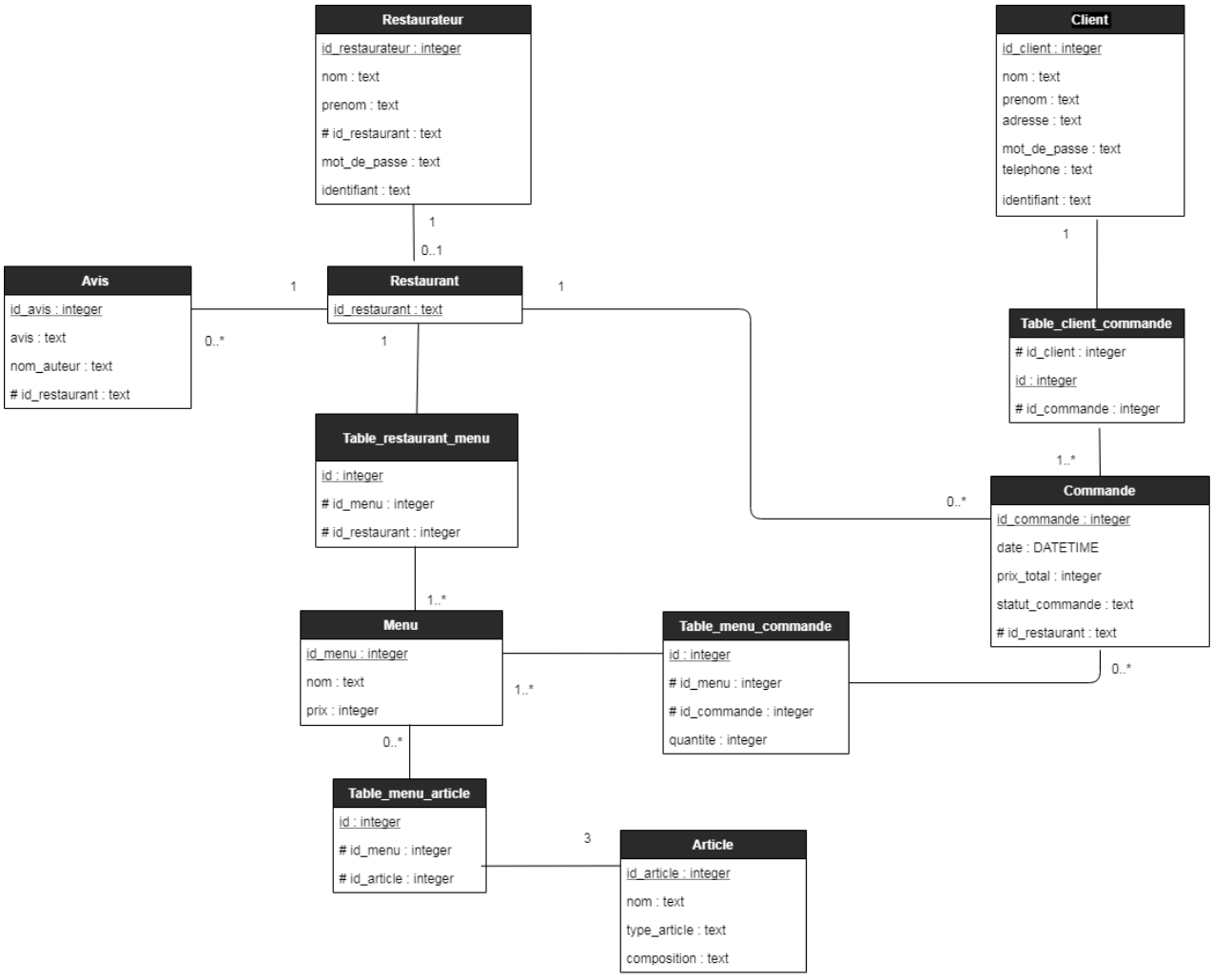

FIGURE 13 – Diagramme logique des données

Comme on peut le voir nous avons intégré 4 tables dans les associations : Table_restaurant_menu, Table_menu_article, Table_menu_commande et Table_client_commande.

La table Table_menu_article représente la relation entre Menu et Article. Ainsi, elle stocke, pour chaque Menu, les 3 articles la composant. C'est pour cela qu'elle possèdent deux clés étrangères. Un même article n'est présent qu'une fois dans la table menu mais peut être présent plusieurs fois dans la table Table_menu_article, pour différents menus.

La table Table_restaurant_menu représente la relation "one to many" entre Restaurant et Menu. Ainsi, elle stocke, pour chaque restaurant tous les menus servis. Elle possède donc deux clés étrangères.

La table Table_menu_commande représente, elle, la relation "many to many" entre restaurant et menu. Tandis que la table Table_client_commande représente une relation "one to many" entre client et commande.

3.2 Modèle physique des données

Maintenant que nous avons réussi à regrouper notre base de données dans un modèle logique, nous devons créer un modèle aidant à visualiser la structure de notre base de données avant qu'elle ne soit construite. Après les modèles conceptuel et logique, le modèle physique des données et la troisième et dernière étape de modèlisation.

C'est une étape assez simple : nous abandonnons la forme graphique en gardant la notion de tables et d'entités dans le but de nous rapprocher le plus possible d'un langage de programmation SQL. On garde les informations qui sont très utiles pour faire les liens tels que les identifiants, les clés primaires (soulignées) ainsi que les clés étrangères (précédées du symbole ♯).*Cequidonne* :

Restaurant (id_restaurant)

Avis (id_avis, avis, nom_auteur, ♯*id_restaurant*)

Restaurateur (id_restaurateur, nom, prenom, mot_de_passe, identifiant, ♯*id_restaurant*)

Menu (id_menu, nom, prix)

Article (id_article, nom, type_article, composition)

Table_menu_article (id, ♯*id_menu*, ♯*id_article*)

Table_restaurant_menu (id, ♯*id_restaurant*, ♯*id_menu*)

Commande (id_commande, date, prix_total, statut_commande, ♯*id_restaurant*)

Table_menu_commande (id, quantite, ♯*id_menu*, ♯*id_commande*)

Client (id_client, nom, prenom, identifiant, mot_de_passe, adresse, telephone)

Table_client_commande (id, ♯*id_commande*, ♯*id_client*)

Nous avons maintenant toutes les informations nécessaires pour la suite, c'est-à-dire la création de nos tables en SQL.

Chapitre 4

Fonctionnalités de l'application

4.1 Choix d'implémentation

Au cours de la conception de notre application, certains choix ont du être pris et d'autres se sont imposés à nous. En effet, toutes ces décisions ont été prises dans un souci de simplification du problème et de réalisme.

4.1.1 Clients et Restaurateurs

Au début du projet, nous avions regroupé tous les différents utilisateurs dans une seule et même case Utilisateurs ce qui nous permettait de grandement et grossièrement simplifier le problème. Cependant, ce premier choix manquait malheureusement de réalisme. En effet, notre application est destinée à deux types d'utilisateurs : d'un côté, les Clients, et de l'autre, les Restaurateurs. Ce premier choix fut donc abandonné car il ne permettait aucune distinction entre ces deux types d'utilisateurs qui possèdent des champs d'actions très différents.

Les Clients ont un champ d'action assez limité. Ils ne peuvent que passer des commandes ainsi que lire et écrire des avis sur les restaurants. Tandis que les Restaurateurs ont la possibilité de créer de nouveaux articles et d'ajouter de nouveaux menus à leur carte.

Par conséquent, une fois définis les différents profils d'utilisateurs et leurs champs d'action, nous avons vite pris le parti de les séparer en deux.

4.1.2 Sélection du jeu de données

Lors de la construction de la base de données de notre projet, proposée dans le chapitre précédent, nous avons créé, à l'aide de requêtes SQL, 10 tables dont 3 tables d'association : Table_menu_article, Table_menu_commande et Table_client_commande.

Afin de ne pas faire d'erreurs pour les chaînes de caractères, nous avons préféré mettre le type TEXT plutôt que VARCHAR(n). En effet, le type TEXT nous facilite la tâche car il peut contenir un très grand nombre de caractères (une longueur maximale de 65 535 caractères), ce qui peut aider pour des données ayant des longueurs assez variables telles que Adresse ou Avis.

De plus, nous avons construit un jeu de données grâce à l'API de YELP. Notre jeu de données contient alors 50 restaurants et 5 menus, ce qui paraît suffisant pour réaliser des tests. En ce qui concerne les avis, ils sont en anglais et nous avons rajouter un unique auteur à tous ces avis, un certain Patrick. Ce choix manque certes de réalisme mais on peut considérer qu'au commencement de l'application nous nous basons seulement sur les avis d'un seul critique culinaire, Patrick (l'un des meilleurs de sa génération). Ensuite, les avis sur les restaurants se créeront petit à petit avec d'autres profils d'utilisateurs.

4.2 Explication détaillée de l'API

4.2.1 Récupération des données sur l'API de Yelp

Avant de créer notre propre API, la première étape était de récupérer sur l'API de Yelp les informations disponibles sur les restaurants. Pour se faire nous avons implémenter une classe **YelpApiService** permettant avec la méthode get_business_by_id de récupérer les informations sur un restaurant en rentrant l'identifiant de celui-ci ou de récupérer les informations de plusieurs restaurants autour d'une adresse et facultativement en fonction d'un terme et d'un rayon par défaut égal à 3000 mètres grâce à la méthode get_businesses. Ces deux méthodes renvoient un json.

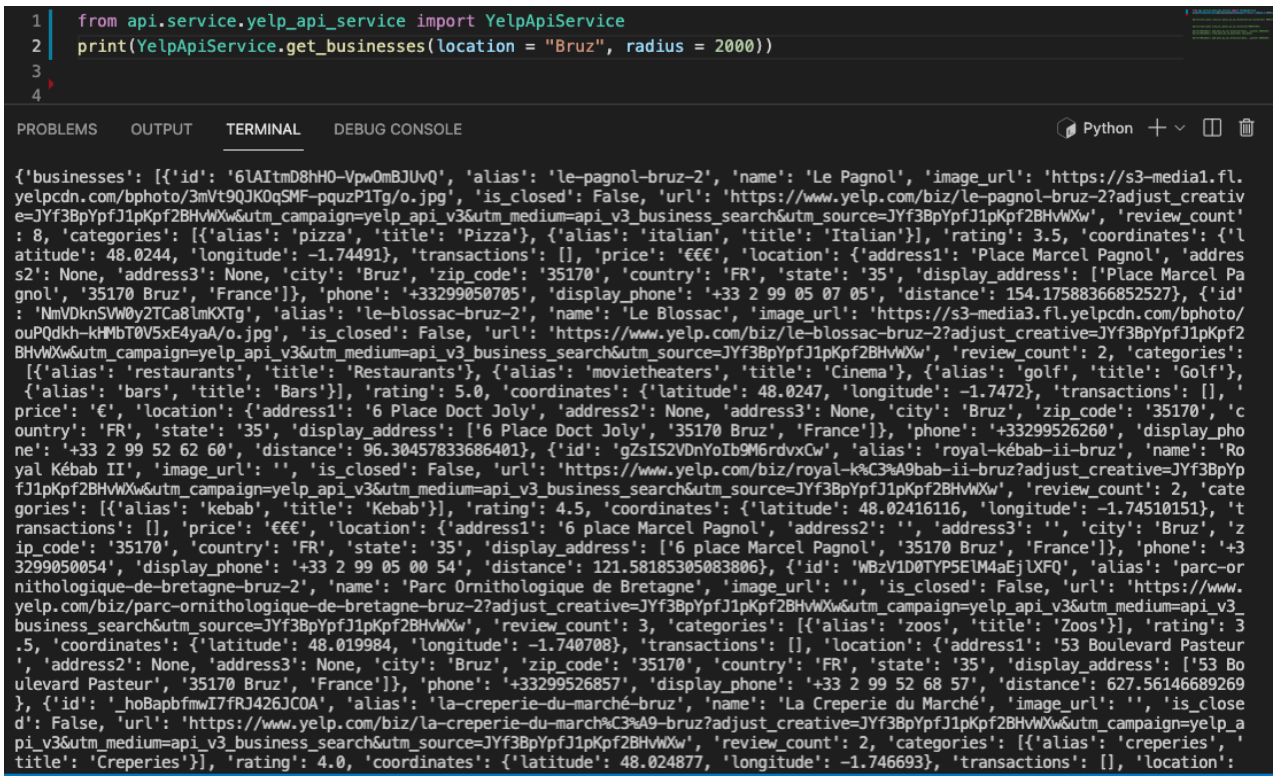

FIGURE 14 – JSON des restaurants

Pour pouvoir traiter les restaurants et leurs informations il était primordial de les transformer en type Restaurant et les adresses en type Adresse. Les méthodes permettant d'une part de passer d'un format json à une liste de restaurant (ou à un restaurant le cas échéant) se trouvent dans le package YelpMapper. Celles-ci permettent aussi de pouvoir trier les informations que nous considérons pertinentes à faire figurer dans notre API (i.e identifiant, nom, adresse, ouvert/fermé). Comme vous pouvez le voir, l'adresse du restaurant est au format dict dans le json. Ainsi, nous avons implémenter la méthode yelp_location_to_location afin qu'elle en ressorte un objet Adresse.

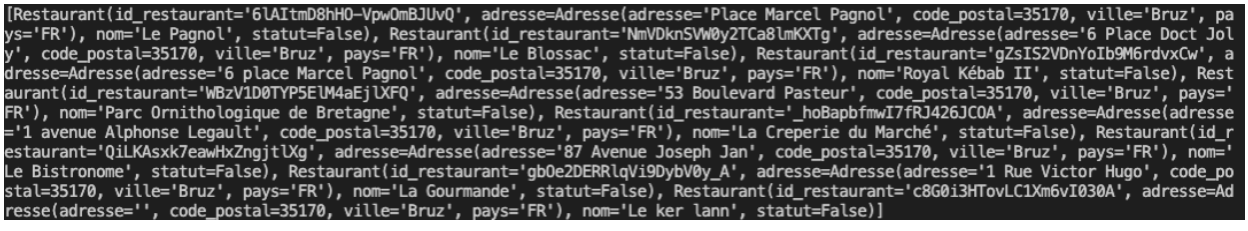

FIGURE 15 – Sortie après YelpMapper

4.2.2 Les différents routers disponibles sur notre API

Pour la création de l'API, nous avons utilisé le package FastApi.

1. **L'authentification :**
   L'utilisateur qu'il soit client ou restaurateur doit avoir un compte pour accéder à quelconques fonctionnalités. Ainsi, nous avons distingué restaurateur et client. Lorsqu'un restaurateur se crée un compte sur l'API, plusieurs vérifications sont faites. D'une part, si l'identifiant et/ou le mot de passe indiqué(s) ne sont pas vides, si l'identifiant (correspondant à un "pseudo") n'est pas déjà pris, si le restaurant n'a pas déjà un propriétaire, et si l'identifiant du restaurant est bien référencé dans l'API de Yelp.
   Lorsqu'il modifie ses informations : le restaurateur doit au préalable indiquer son identifiant et son mot de passe. Si ces informations de connexion ne sont pas correctes aucune modification ne pourra être effectuée. Dans le cas où ses informations de connexion sont bonnes, les mêmes informations que lors de la création d'un compte restaurateur sont vérifiées. Pour supprimer son compte restaurateur, il n'a qu'à indiquer son identifiant et son mot de passe son compte sera automatiquement supprimé de la base de données.
   Pour le client, lors de la création d'un compte client il est vérifié que l'identifiant indiqué n'est pas déjà pris ou ne soit pas vide, également que le mot de passe ne soit pas vide par soucis de sécurité. Pour la modification de ses informations, le client doit insérer son identifiant et son mot de passe sans quoi les modifications ne seront pas prises en compte. L'identifiant peut être changé à condition que celui-ci ne soit pas déjà pris par un autre client.

2. **Restaurants :**
   Les informations des restaurants sont directement tirées de l'API de Yelp. Un utilisateur peut chercher un restaurant par son identifiant, ou récupérer les restaurants autour d'une localisation. Si l'identifiant indiqué est faux l'API reçoit une erreur "Le restaurant n'est pas dans l'API de Yelp".

3. **Articles :**
   Dans notre cas, les articles composent les menus. Un restaurateur peut ajouter, supprimer ou modifier un article. Si un restaurateur souhaite supprimer un article, non seulement l'article mais les menus composés de cet article sera aussi supprimé. L'ajout d'article serait utile en particulier si nous avions développé le fait qu'un client puisse commander des articles à l'unité et pas seulement en menu.

4. **Menus :**
   Un restaurateur peut ajouter un menu à un restaurant : pour cela il faudra que celui-ci renseigne son pseudo et son mot de passe. Si ceux-ci correspondent aux informations du restaurateur du restaurant renseigné par l'identifiant alors son menu sera ajouté. Pour ajouter un menu, le restaurateur doit renseigner les informations du menu : le nom, le prix mais aussi 3 articles qui seront automatiquement ajoutés à la base de données. Pour modifier un menu, il doit encore une fois entrer ses identifiants de connexion et l'identifiant du menu à modifier puis inscrire les informations du menu. Pour le prix et le nom il doit les renseigner même s'il ne souhaite pas les modifier. S'il souhaite changer un article il devra renseigner les informations nécessaires et pour les articles qu'il souhaite garder le restaurateur n'a qu'à rentrer l'identifiant.
   Le restaurateur ne pourra en aucun cas changer l'identifiant de son menu.
   Un restaurateur ou un client peut avoir accès aux menus d'un restaurant en renseignant l'identifiant du restaurant en question et ses informations personnelles : identifiant et mot de passe. Si la connexion échoue, l'utilisateur n'a pas accès aux menus.
   Enfin, un menu peut être supprimé par le restaurateur.
5. **Commandes :**
   Un client peut passer une commande mais pas un restaurateur. Une commande est composée d'au moins un menu. Si le client souhaite une quantité n d'un menu, il n'a qu'à renseigner la quantité souhaitée dans la liste des quantités.

Il est aussi possible pour un client de récupérer l'entiereté de ses commandes et respectivement pour un restaurateur.

6. **Avis :**
   Les clients peuvent laissés un avis à un restaurant en indiquant l'identifiant du restaurant, leur avis, un pseudo (= identifiant) qui ne doit pas forcément être égal à celui de leur compte afin de pouvoir garder un anonymat. Les restaurateurs et clients peuvent récupérer les avis d'un restaurant en indiquant l'identifiant du restaurant.
   Pour toutes les fonctionnalités liées à la modification ou la suppression d'un menu, voir les commandes d'un restaurant etc.. si le restaurateur n'est pas le propriétaire du restaurant une Exception dont le status est 403 lui indique qu'il ne dispose pas des droits d'accès.

4.3 Explication détaillée du client

Cette partie présente en détail les différentes étapes que le client exécutent afin de passer une commande. Ces étapes concernent l'authentification ou l'inscription, la recherche de restaurant, le choix de menus, la composition du panier et, enfin, la validation de la commande.

4.3.1 Authentification

Lorsque l'utilisateur démarre l'application Client, il a le choix entre s'authentifier directement, s'il a déjà un compte, ou en créer un.

FIGURE 16 – View authentification

La création de compte se fait grâce au view suivant. Une fois le compte crée le client peut maintenant faire la recherche de restaurant selon différents critères.

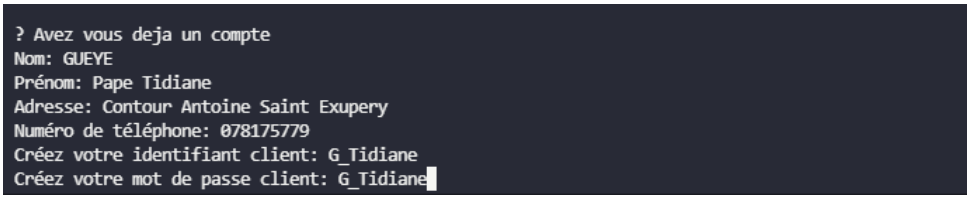

FIGURE 17 – View authentification (suite)

4.3.2 Recherche de restaurant

Les paramètres de recherche inclus la **localité**, **le nom du restaurant** et **le rayon**. L'utilisateur peut aussi voir les restaurants disponibles aux alentours de son adresse.

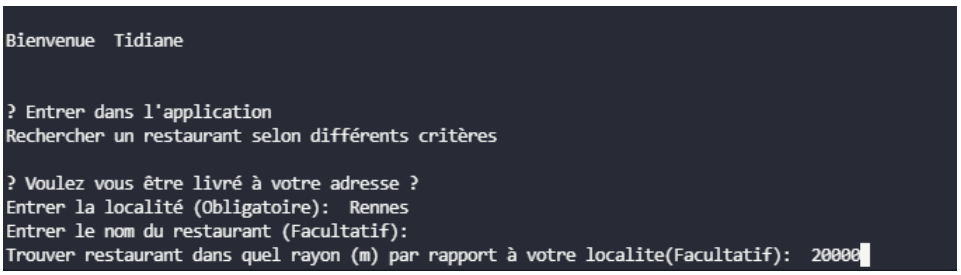

FIGURE 18 – Recherche restaurant - paramètres

A cette étape, nous effectuons une requête GET vers notre API en précisant les différents paramètres donnés par l'utilisateur.

FIGURE 19 – Liste restaurant

4.3.3 Consultations des avis et menus du restaurant

Après avoir choisi le restaurant qui lui plait, l'utilisateur est redirigé vers une view lui permettant, soit de voir les menus du restaurant, soit de voir les avis du restaurant.

FIGURE 20 – Vitrine du restaurant

Dans consulter les avis, on retrouve tous les avis concernant le restaurant et on a la possibilité de donner notre avis sur le restaurant. Ensuite dans consulter les menus, on a les différents menus disponibles dans le restaurant.

4.3.4 Composition du panier

Une fois la liste des menus affichés, l'utilisateur effectue son choix de menu et la quantité, et l'ajoute dans son panier.

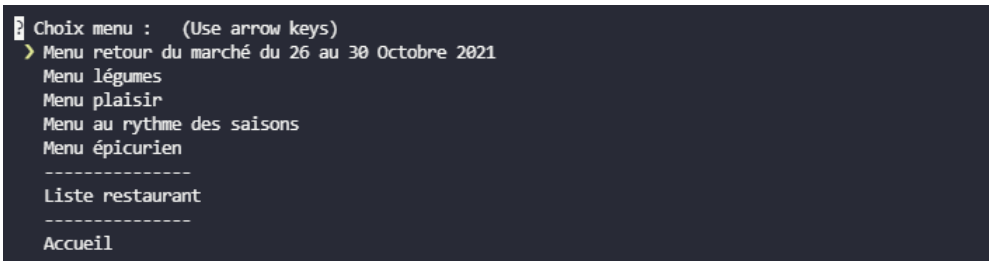

FIGURE 21 – Liste des menus

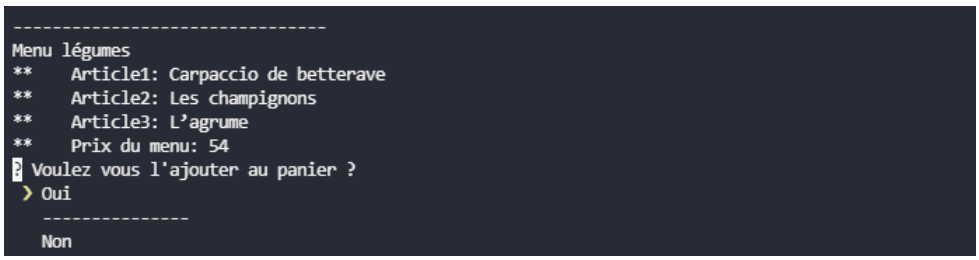

FIGURE 22 – Ajout du menu

Nous offrons aussi la possibilité au client de modifier son panier, d'avoir un récapitulatif du panier avant de valider et de pouvoir annuler son panier quand il le souhaite. Enfin, en validant sa commande, celle-ci est enregistrée dans la base de données et l'utilisateur revient à l'accueil.

FIGURE 23 – Validation de la commande

Chapitre 5

Bonnes pratiques de codage

La modélisation des datas apporte, évidemment, d'innombrables avantages. Cependant, utiliser ou créer des modèles biaisés génèrent des dangers inhérents importants. Afin d'éviter tout risque de mauvaise conception, nous avons adopté quelques bonnes pratiques de data modeling que nous décrirons par la suite.

5.1 Tests unitaires

Nous avons réalisé des tests unitaires fonctionnels afin de vérifier que notre application fonctionne mais aussi pour montrer quelques cas d'applications concrets de nos services. Nous les faisons pour anticiper certaines erreurs spécifiques, respecter le cahier des charges et gagner en efficacité. Il était préférable pour nous de ne montrer que quelques tests qui illustrent les spécificités de notre implémentation.

Pour cela, nous avons choisi d'effectuer ces tests sur la classe restaurant_service de notre package service qui est une classe centrale de notre application. Les méthodes testées sont celles permettant d'ajouter, supprimer ou modifier des articles et des menus.
Pour ces classes, notre DAO renvoie des booléens à chaque insertion, suppression ou modification des articles et menus. Nos tests vérifient que le tuple de booléens retourné correspond bien à des True.

Pour le test des méthodes permettant de récupérer les restaurants de Yelp par des requêtes API, nous étudions le cas où un mauvais id d'un restaurant est renseigné. Dans ce cas, nous sommes censé recevoir une exception particulière que nous avons implémenté pour ce cas précis où on ne retrouve pas l'id d'un restaurant. De même, lorsqu'on renseigne une valeur de périmètre (radius) autour de la localisation trop élevé, nous devons récupérer une liste vide.

5.2 Tests utilisateurs

Selon la norme ISO 9241-11, le test utilisateur est défini comme le « degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction dans un contexte d'utilisation spécifié ». En d'autres termes, c'est une méthode d'évaluation d'un produit ou d'un service par des utilisateurs. Les tests utilisateurs sont indispensables tant ils constituent une étape clé pour garantir l'efficacité d'un dispositif interactif. Ils sont également une aide à la décision de réalisation de projet de conception, c'est pourquoi, les tests utilisateurs constituent un réel avantage compétitif pour les entreprises. Dans notre cas, les tests utilisateurs ont été réalisés par nos soins.

Tout au long du projet, nous nous sommes efforcés de tester notre code avec la plus grande rigueur possible à notre portée.

5.3 Codage à plusieurs

Gitlab est une plateforme opensource et collaborative de développement basé sur Git. Gitlab permet d'héberger des projets web, du code, de la documentation. La gestion des différentes versions et conflits est prise en compte dans Gitlab et permet le travail de nombreux collaborateurs simultanément, efficacement et de manière interactive. Dans le cas de notre projet, cet outil a été très important et apprécié par tous les membres de l'équipe. Il a permis de travailler efficacement et simultanément sur plusieurs packages. Lors de la conception de l'API métier, nous avons eu à travailler à plusieurs sur le même package. Nous avons également pu conserver chacun les différentes versions de notre projet. Git a été un outil essentiellement dans la réalisation de ce projet.

Chapitre 6

Bilans personnels

6.1 Ghoumari Kiyan

Tout d'abord, le sujet Ensaeats faisait réellement parti de mes préférés. Je trouvais cela vraiment intéressant de pouvoir coder une application que l'on connait tous et qui a su révolutionner le monde de la livraison et de la restauration, notamment grâce à la crise sanitaire. J'aime, tout particulièrement, mettre à exécution mes connaissances lorsqu'il s'agit de traiter des sujets qui me touchent dans mon quotidien d'étudiant. Je ne m'attendais pas à un sujet simple, à prime abord, mais j'ai tout de même été surpris , au cours du projet, par la complexité et la coordination qu'il demandait. En effet, nos emplois du temps étaient assez différents et c'était assez complexe de trouver des créneaux similaires avec les autres membres de mon groupe pour les réunions. Heureusement, notre groupe a su rester concentré et trouver des solutions aux petits problèmes qui ont pu survenir.

En ce qui concerne l'aspect technique, j'ai notamment réalisé le diagramme de classes qui changea plusieurs fois tout au long du projet puisque nos idées ont, eux aussi, changé et des soucis de faisabilité se sont imposés. Je me suis également occupé de la création du modèle conceptuel ainsi que des modèles logique et physique des données qui sont les 3 grandes étapes aidant à la construction d'une base de données. De plus, j'ai pu appliquer mes connaissances en codage dans la conception de la DAO. Nathan m'a notamment été d'une grande aide lorsqu'il a fallu commencer à coder. Il a, pour moi, rempli son rôle de chef technique à la perfection, m'aiguillant sur certains aspects de la DAO lorsque j'en avais besoin. Puis vers la fin du projet, il m'a été donnée comme mission de construire le plan du rapport final ainsi que sa rédaction et la modification des incohérences.

Pour conclure, je dirai que je suis heureux d'avoir eu la chance de travailler sur ce sujet complexe mais passionnant et fière du travail d'équipe fournit tout au long du projet.

6.2 GUEYE Pape Tidiane

Ce projet a été d'un grand apport aussi bien technique que personnel. Tout d'abord, j'ai commencé le projet avec des lacunes en informatique, notamment en modélisation UML et programmation python. Toutefois, grâce aux échanges effectuées avec le groupe et mes recherches sur le sujet, j'ai pu combler ce gap et participer pleinement au projet.

Au début du projet, nous somms partie sur une approche assez simpliste en ne faisant aucune distinction en l'utilisateur lamda et le restaurateur. Ceci a permis de faciliter la compréhension et de ne pas être intimidé par la complexité du sujet. Ensuite, nous avons introduit les spécificités de chacun ainsi que les différences entre l'API et l'application Client. Sur le plan technique, le projet m'a permis de comprendre les différentes phases de conception d'un projet informatique ainsi que le paradigme de programmation orienté objet. En terme de participation, j'ai beaucoup plus travaillé sur le côté Client, avec le diagramme de séquence du Client et la création des Views permettant faisant office d'interface de communication entre l'utilisateur et les couches services de l'API. Le travail sur les views n'était pas simple car il y avait beaucoup de liaisons entre les views. Cela demandait toujours de se mettre à la place d'un utilisateur qui veut faire une commande et mettre les fonctionnalités nécessaires pour faciliter la navigation. Ayant fait un peu de développement web, j'étais motivé de travailler sur les views vu que j'avais bien appréhender le role des sessions et la manière dont les views communiquent.

Ma grande découverte a été l'outil "git" qui nous a permis de gérer le versionage du projet, géré les conflits sur les fichiers et avoir un projet qui est toujours à jour. En effet, avant cette découverte j'avais toujours des dossiers du type : projet-bon, projet-bon-final, projet-final-final, etc. L'usage de cet outil assez simple qu'on se croirait faire un jeu d'enfant, nous a permis de suivre l'avancement de chacun dans le projet.

Enfin, j'ai beaucoup apprécié l'ouverture des membres du groupe. Nous avons eu des échanges assez intéressantes et abordées différents problèmes inhérents au projet. Tout au long du projet nous nous sommes entraider afin que personne ne soit bloqué. A cela s'ajoute les remarques que chacun pouvait avoir sur le travail de l'autre afin d'apporter des améliorations. En travaillant ainsi, nous avons pu répondre aux exigences avancées du projet.

En résumé, durant ce projet j'ai beaucoup appris et j'ai bien aimé travaillé avec mon groupe. Je pense avoir bien progressé par rapport à mon niveau en début d'année en informatique.

6.3 Moulin Valentine

Aux premiers abords, le sujet me paraissait intéressant et c'était celui que je préfèrais. Cependant, je n'avais pas anticipé la complexité derrière.

Le rôle de chef de projet m'a été rapidement donné. Ce projet informatique intense soit-il reste une expérience enrichissante. Malgré des bonnes notes en programmation et en informatique en général en première année à l'ENSAI ou en licence, ce projet a tout de même été une réelle épreuve. Une épreuve ; de par sa complexité au niveau de la compréhension et de la programmation et tout ça dans le peu de temps qui nous été attribué.

L'utilisation de GitLab fut compliqué pour moi, n'ayant pas utilisé GitLab l'année dernière pour le projet de Traitement de donneés, mais j'ai su rapidement m'y faire. Côté programmation, je me suis occupée de la création de l'API, des différents routers, des objets métiers et du package DAO qui avait déjà été commencés par Tidiane et Nathan. Ce partage de tâche était efficace, cependant je n'ai quasiment pas touché au côté client et particulièrement aux views.

Au delà des compétences techniques que ce projet a pu m'apporter, j'ai dû apprendre à travailler avec mes coéquipiers. Le travail à 5 fut de temps en temps complexe : pour se mettre d'accord, pour se partager les tâches, ne pas créer de conflit sur Git etc.

Tout au long du projet, je suis passée par différentes phases. La première étant lors des débuts du fonctionnement de l'API : j'étais très motivée pour commencer à programmer et à l'idée de voir des améliorations concrètes sur l'API. Puis, est venu le moment où beaucoup d'erreurs apparaissaient, le moment où on stagnait, le projet n'avançait plus. En est suivi une période où ma motivation pour ce projet était au plus bas, et je n'arrivais pas à voir la fin. Je voyais beaucoup de groupes ayant un projet fonctionnel et nous toujours pas. Pour autant, j'ai réussi à retrouver la motivation de finir ce projet grâce à notre tuteur Eneman Donatien.

Pour conclure, malgré la complexité de ce projet, les baisses de motivation , je suis globalement fière de ce que nous avons rendu.

6.4 Randriamanana Nathan

Mon rôle dans ce projet était celui du chef technique.

J'ai trouvé le sujet "EnsaEats" très ambitieux et c'est pour cela que je l'ai placé dans mes premiers voeux. De manière générale, j'ai trouvé que le projet pouvait être simple mais qu'il pouvait devenir rapidement difficile si on ne modélisait pas correctement notre application, avant de coder. Avoir été interdit de coder les premières semaines m'a aidé à aller au plus simple pour être plus efficace. Il m'a fallu deux ou trois séances avec notre encadrant pour comprendre ce qu'on devait vraiment faire.

Le fait d'avoir un projet ambitieux était à la fois moteur et pénalisant dans mon cas. Je me suis rendu assez vite compte que notre application allait forcément manquer de réalisme compte tenu du délai que nous avions pour produire ce travail de groupe. En effet, pendant les vacances qui suivaient le rendu du rapport d'analyse, j'ai effectué du webscraping pour récupérer les menus des restaurants se trouvant sur YELP parce que l'API de YELP ne fournit pas directement les menus des restaurants. Cela devenait vite fastidieux dans la mesure où les restaurants listés sur YELP ne possède pas forcément de site internet et même, le cas échéant, l'affichage des menus diffèrent selon les sites : sur le site directement, en pdf, en image ou ne figure pas du tout sur le site officiel... Bien que cela n'ait pas été dans les attentes du projet, j'ai pu me familiariser avec requests et l'appel de requête API.

La première semaine où nous avons codé (lors des journées d'immersion dédiées), nous nous sommes répartis les métiers et services en deux : je me suis occupé de la DAO et de la partie des services qui s'occupe des requêtes SQL tandis que deux membres du groupe s'occupait des requêtes YELP. Au départ, comme on ne connaissait pas notre niveau, je n'étais pas sûr de pouvoir être chef

technique. Finalement, j'ai pu être plus à l'aise sur la DAO et j'ai commencé à aider mes collègues sur les problèmes de requête SQL et de gestion des packages Python ainsi que des paramètres d'environnement avec le serveur distant PostgreSQL de l'école. J'ai aussi découvert comment gérer les ports sur uvicorn pour déployer l'API en local.

Ensuite, il a fallu beaucoup communiquer avec ceux qui s'occupaient des services dont Valentine qui a aussi entamé le développement des routeurs pour l'interface de notre API. J'ai aussi commencé à gérer les routeurs comme la DAO, les services et les routeurs sont très liés. J'ai donc pu contribuer en partie la gestion des endpoints. J'ai priorisé l'API mais j'ai aussi assisté Tidiane pour créer les views en l'aidant à lui fournir les bonnes requêtes sur notre API et en faisant l'intermédiaire entre ce que gérait Valentine dans la DAO et ce que devait afficher les views.

Pour plus de confort, j'ai tenté de déployer en continu notre API sur un serveur distant en utilisant "Deta" qui était proposé dans la documentation de FastApi qu'on peut obtenir via le lien suivant : https ://oyooou.deta.dev/docs. Malheureseument, cela ne permet pas de se connecter à PostgreSQL. Outre l'incapacité de "Deta" à se connecter à PostgreSQL, il me faudrait des droits d'accès au réseau. J'ai aussi pu effectuer quelques tests unitaires.

J'ai trouvé l'organisation du travail de notre groupe assez difficile. Planifier des réunions et créer des tableaux ne suffit pas pour avoir une cohésion de groupe, il a été nécessaire et plus efficace de voir directement la personne. J'ai trouvé que coder côte-à-côte pouvait parfois être très productif et stimulant : demander à l'autre ce qu'on veut lui envoyer et savoir qu'on pense à la même chose est rassurant.

En tant que chef technique, j'essayais d'aider les membres de mon groupe du mieux que possible mais le développement de l'application s'est présenté particulièrement prenant. Les problèmes techniques surviennent à tout moment et notre travail ne peut pas être indépendant des autres.

En conclusion, j'ai beaucoup appris de ce projet et je suis content de savoir que je pourrai déployer une API à l'avenir. Je ne pensais pas que ça pouvait aussi bien se faire sur Python.

6.5 Zoulkarnayni Nikiema

Ce projet informatique m'a permis d'apprendre plusieurs notions en informatique. J'ai été membre d'une équipe composée de 5 personnes. Lors de l'attribution des groupes, je vous avoue que je ne connaissais pas très bien les camarades de mon groupe.

Pour ce rapport sur le diagramme UML, je me suis occupé des diagrammes d'activité du client et celui qui permet au client de passer une commande. Cette tâche a été réalisée avec l'aide de Nathan et Valentin. De plus, je me suis occupée de présenter FastAPI et Yelp. Ensuite, dans la couche métier de notre api métier, je me suis occupé du code des diagrammes de classes. Après cette tâche, par la demande de notre chef de projet, j'ai dû me mettre sur le rapport pour faire une correction sur tous les diagrammes du projet. J'ai également effectué une modification sur les parties incohérentes du rapport. De plus, j'ai documenté toutes les classes et fonction de notre package Api métier.

Pour conclure, j'ai trouvé ce projet très formateur, mais néanmoins très frustrant, parce que je considère que je n'avais pas un très bon niveau en développement d'applications. Néanmoins, j'ai vraiment apprécié ce projet informatique et je pense que l'informatique sera certainement essentielle lors de mon futur travail. Je félicite aussi notre tuteur Eneman Donatien qui fut d'une grande aide pendant tout le long du projet.

Conclusion

Notre projet avait pour but de fournir une application permettant d'une part aux clients de commander des menus auprès de restaurants et aux restaurateurs de proposer des menus, et s'ils le souhaitent, de modifier leur carte.

Notre API est fonctionnelle et réponds aux besoins. Elle permet aux restaurateurs de gérer leur compte (création, modification, suppression), de gérer leur restaurant par l'ajout de menu, la modification, ou la suppression, de pouvoir récupérer toutes ses commandes et de voir l'avis de son restaurant mais aussi des autres restaurants.

L'authentification sur l'API nous permet de contrôler qu'un client ne puisse pas modifier la carte d'un restaurant. De plus, un restaurateur ne peut pas modifier la carte d'un restaurant dont il n'est pas le propriétaire.

Notre API est fonctionnelle aussi pour les clients mais les clients utilisent uniquement l'application client qui envoie des requêtes à notre API. Ainsi, sur la console, le client peut se créer un compte. Pour accéder à toutes les fonctionnalités, le client doit être connecté. Il peut chercher des restaurants autour de chez lui ou d'une autre adresse, spécifié ou non le rayon autour de l'adresse, et/ou une spécialité. Il peut consulter les avis d'un restaurant et, comme le nom de notre projet l'indique, il peut voir les menus d'un restaurant et commander s'il le souhaite.

Limites et améliorations

Plusieurs améliorations auraient pu être faite. D'une part, les restaurateurs n'agissent que sur l'API. Ainsi, une interface console simple à destination du restaurateur aurait pu être faite.

Dans notre fonctionnement, les restaurants ne proposent que des menus, ceci est très restrictif et pas vraiment réaliste. Une amélioration possible aurait été que les clients puissent aussi commander des articles à l'unité.

Une troisième amélioration est le suivi de la commande. En effet, notre classe Commande a un attribut statut_commande. Or celui-ci n'est jamais modifié et reste toujours égal à "en cours". On pourrait imaginer un suivi de la livraison comme le propose UberEats.

Annexes

Code Base de données

```
CREATE TABLE ensaeats.Restaurant ( id_restaurant text PRIMARY KEY ) ;

CREATE TABLE ensaeats.Avis (
id_avis SERIAL PRIMARY KEY,
avis text,
date date,
nom_auteur text,
id_restaurant text,
FOREIGN KEY(id_restaurant) REFERENCES
ensaeats.Restaurant(id_restaurant)
) ;

CREATE TABLE ensaeats.Restaurateur (
id_restaurateur SERIAL PRIMARY KEY,
nom text,
prenom text,
identifiant text,
mot_de_passe text,
id_restaurant text,
FOREIGN KEY (id_restaurant) REFERENCES
ensaeats.Restaurant(id_restaurant)
) ;

CREATE TABLE ensaeats.Menu(
id_menu SERIAL PRIMARY KEY,
nom text,
prix INT
) ;

CREATE TABLE ensaeats.Article(
id_article SERIAL PRIMARY KEY,
nom text,
type_article text,
composition text
) ;

CREATE TABLE ensaeats.Table_menu_article(
id SERIAL PRIMARY KEY,
id_menu INT,
id_article INT,
FOREIGN KEY (id_menu) REFERENCES
ensaeats.Menu(id_menu),
FOREIGN KEY (id_article) REFERENCES
ensaeats.Article(id_article)
) ;

CREATE TABLE ensaeats.Table_restaurant_menu(
id SERIAL PRIMARY KEY,
id_menu INT,
id_restaurant text,
FOREIGN KEY (id_menu) REFERENCES
ensaeats.Menu(id_menu),
FOREIGN KEY (id_restaurant) REFERENCES
ensaeats.Restaurant(id_restaurant)
) ;
```

```
CREATE TABLE ensaeats.Commande(
id_commande SERIAL PRIMARY KEY,
date date ,
prix_total INT,
statut_commande text,
id_restaurant text,
FOREIGN KEY (id_restaurant) REFERENCES
ensaeats.Restaurant(id_restaurant)
);
CREATE TABLE ensaeats.Table_menu_commande(
id SERIAL PRIMARY KEY,
id_menu INT,
id_commande INT,
quantite INT,
FOREIGN KEY (id_menu) REFERENCES
ensaeats.Menu(id_menu),
FOREIGN KEY (id_commande) REFERENCES
ensaeats.Commande(id_commande) );

CREATE TABLE ensaeats.Client(
id_client SERIAL PRIMARY KEY,
nom text,
prenom text,
identifiant text,
mot_de_passe text,
adresse text,
telephone text
);

CREATE TABLE ensaeats.Table_client_commande(
id SERIAL PRIMARY KEY,
id_commande INT,
id_client INT,
FOREIGN KEY (id_commande) REFERENCES
ensaeats.Commande(id_commande),
FOREIGN KEY (id_client) REFERENCES
ensaeats.Client(id_client)
);
```