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

Rapport d'analyse

Élèves :

GHOUMARI Kiyan
GUEYE Tidiane Pape
MOULIN Valentine
RANDRIAMANANA Nathan
ZOULKARNAYNI Nikiema

Encadrant :

ENEMAN Donatien

Responsable :

PEPIN Rémi

Année 2020-2021

Sommaire

Table des figures

Avant-propos

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 rapport d'analyse, 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 renouveller 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. C'est la raison pour laquelle, nous décidons dans un premier temps de ne pas traiter les avis ainsi que l'authentification de la clientèle.

Les fonctionnalités de base à implémenter pour notre application sont les suivantes (se référer aux diagrammes de cas d'utilisation) :

- Elle permet à l'utilisateur de pouvoir chercher des restaurants (possibilité de filtrer)
- Elle permet à l'utilisateur de pouvoir consulter les détails d'un restaurant
- Elle permet à l'utilisateur de pouvoir passer une commande
- Elle permet à l'utilisateur de pouvoir ajouter,modifier,supprimer des plats ou menus d'un restaurant
- La possibilité d'effectuer des tests unitaires
- Une interface console simple à destination des clients qui utilisent l'API

Nous pourrons éventuellement implémenter les fonctionnalités suivantes :

- Une interface console simple à destination des restaurateurs qui utilisent l'API
- Une gestion de l'authentification

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

Le temps accordé à chaque tâche de la phase de développement est à titre indicatif car nous n'avons pas une idée claire du temps que chacune nous prendra.

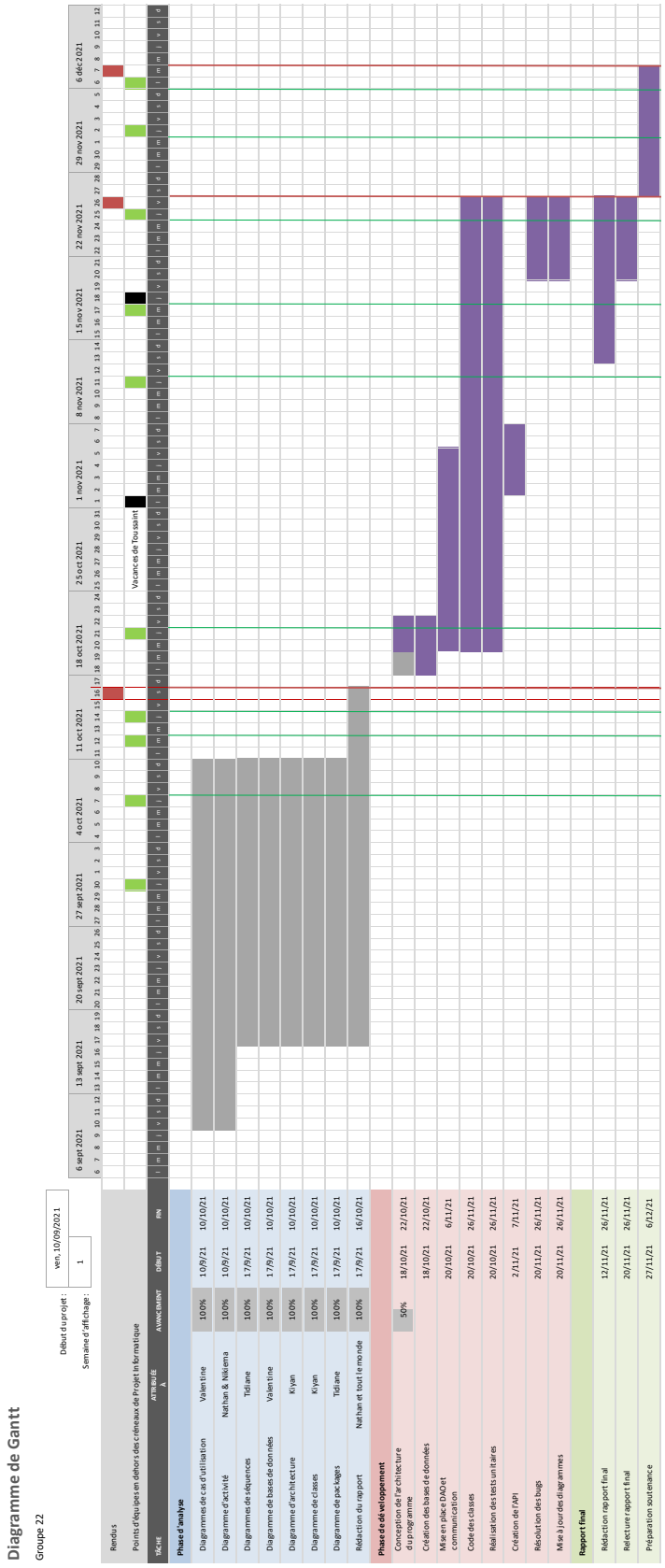

FIGURE 1 – Diagramme de GANTT

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.

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

Dans l'optique d'entretenir une discipline et un rythme de travail régulier, nous avons convenu de réserver un créneau d'une heure 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 considérons qu'un travail hebdomadaire de minimum 3 heures est nécessaire.

1.2.2 Modélisation

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 future.

1.2.3 Répartition des tâches

À ce jour, nous ne connaissons 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 sont 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

Ce projet nous incite à élire un chef de projet ou de planning qui aura un rôle d'encadrant et devra s'assurer du bon déroulement du projet. Nous devons aussi désigner un chef technique qui s'occupera 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. Comme il est difficile à ce stade d'estimer le niveau de chacun, nous avons choisi un chef technique temporaire.

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

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 permet aussi de convenir sur une manière efficace de partager des fichiers lorsque nous serons amenés à coder ultérieurement. 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 "push" et "pull" de Gitlab.

1.2.5 Rendu du dossier d'analyse

Le rendu du dossier d'analyse est principalement composé des diagrammes UML que nous avons élaboré. Ce rapport permet aussi de faire un état des lieux quant à notre compréhension du sujet et de ce que nous devons concrètement réaliser.

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 votre 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

La 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 pour le client et une pour l'API. L'application cliente permettra de faire l'affichage pour l'utilisateur alors que l'API fera réellement les traitements.

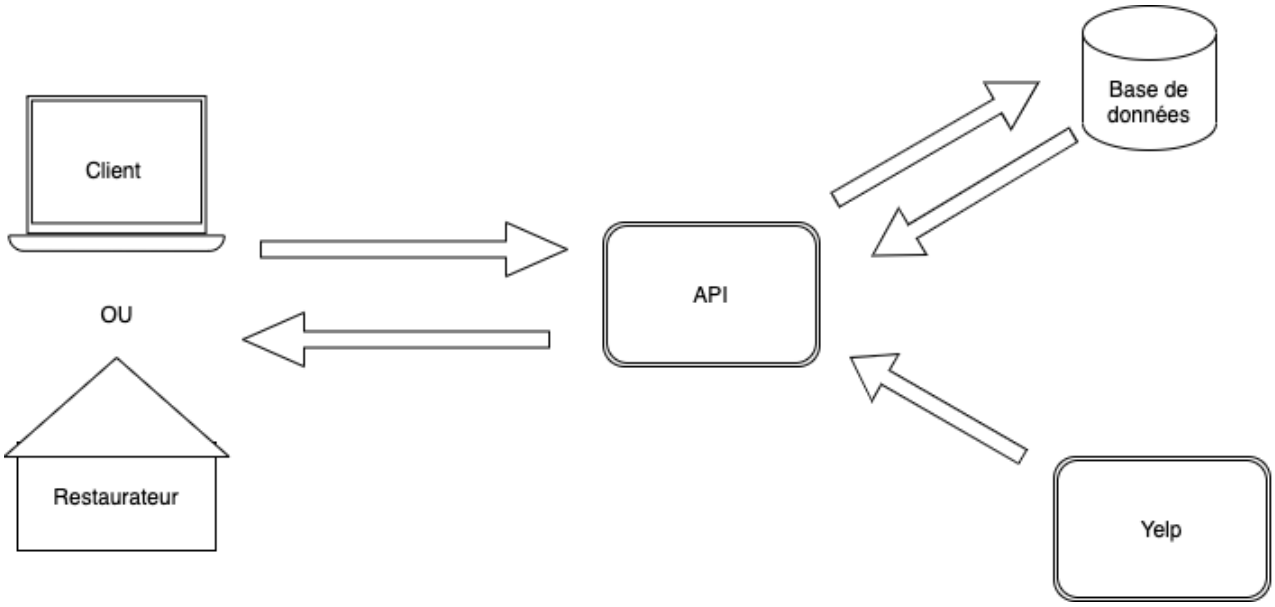

Figure 2 – Diagramme d'architecture

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. Puis 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.. Il peut aussi consulter la liste de ses commandes. L'utilisateur aura la possibilité de voir l'état de sa commande : 'en préparation', 'en livraison', 'livrée'.

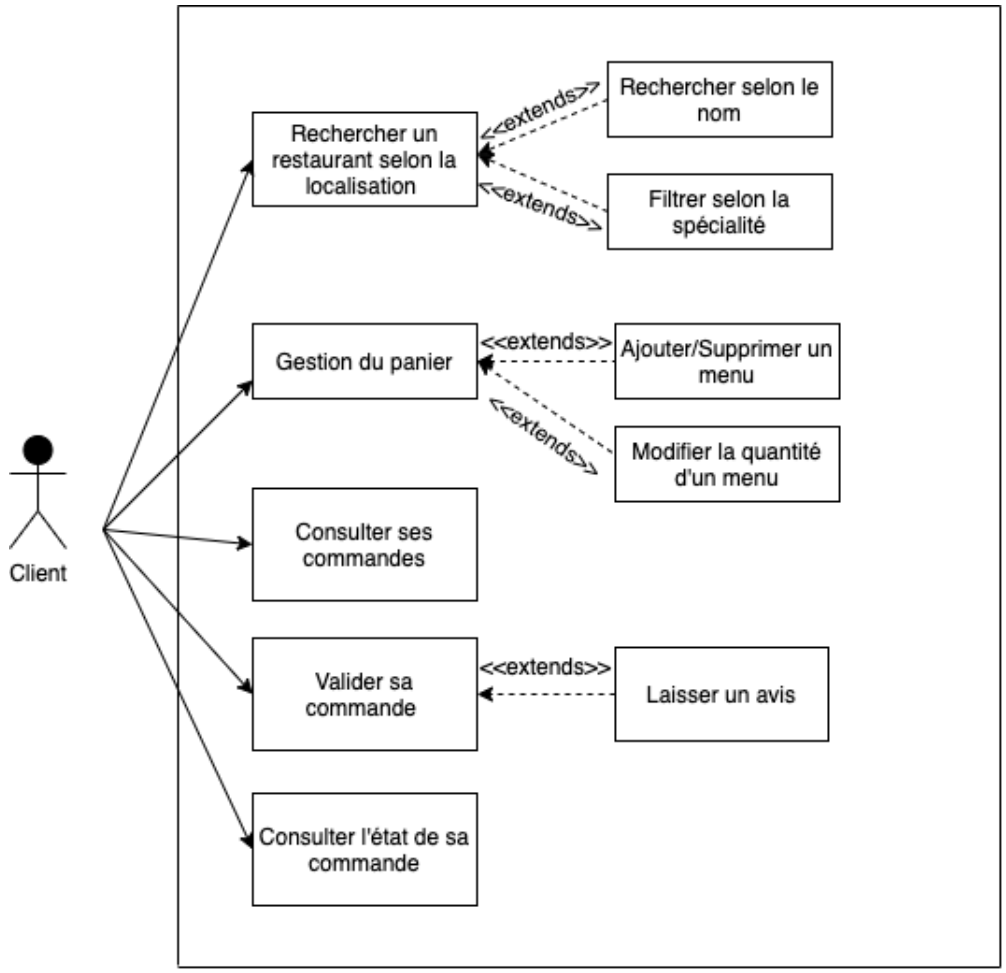

Figure 3 – Diagramme de cas d'utilisation de l'application cliente

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 selon une spécialité, la localisation du client, ou selon le nom du 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. L'API permettra aussi au client de consulter son historique de commande, par exemple s'il souhaite retrouver le nom du restaurant dans lequel il avait commandé, ou s'il souhaite voir l'avancée de sa commande (en préparation, en livraison, livrée).

En tant que restaurateur, l'API permettra à celui-ci 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 prix, etc.. Il pourra aussi consulter la liste de ses commandes classées par ordre chronologique pour plus de facilité pour le restaurateur. S'il le souhaite, il pourra uniquement consulter la liste de ses commandes sur une période de temps donnée (par exemple : le jour-même à partir de 18h représentant les commandes du soir). Le restaurateur, contrairement au client, agit directement sur l'API.

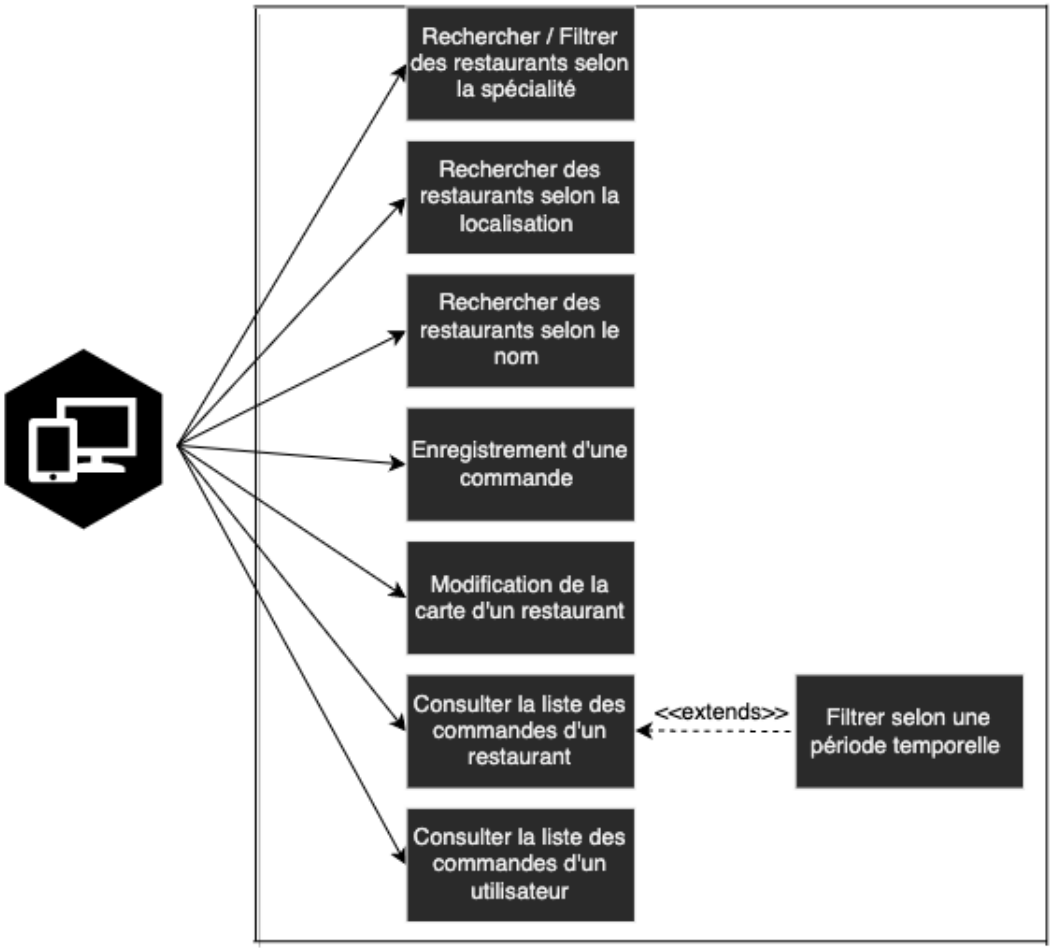

Figure 4 – Diagramme de cas d'utilisation de l'API

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 entrer son adresse afin de lui proposer des restaurants à proximité de chez lui. S'il le souhaite, il peut soit consulter l'entièreté des restaurants, ou filtrer par spécialité (italien, burger, etc..) ou entrer le nom du restaurant. 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 fera une commande.

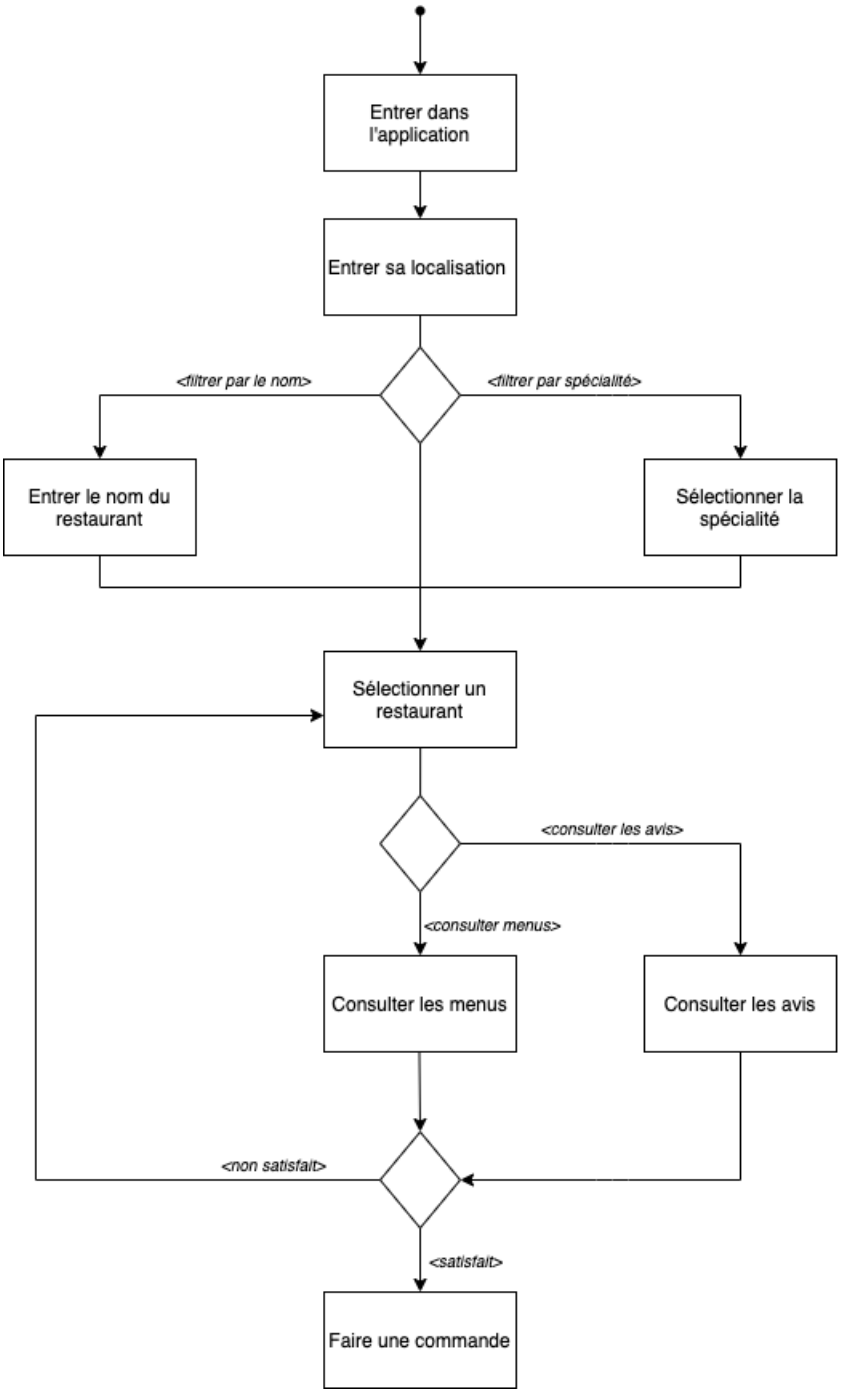

FIGURE 5 – Diagramme d'activité de l'utilisateur

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 clareté. 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.

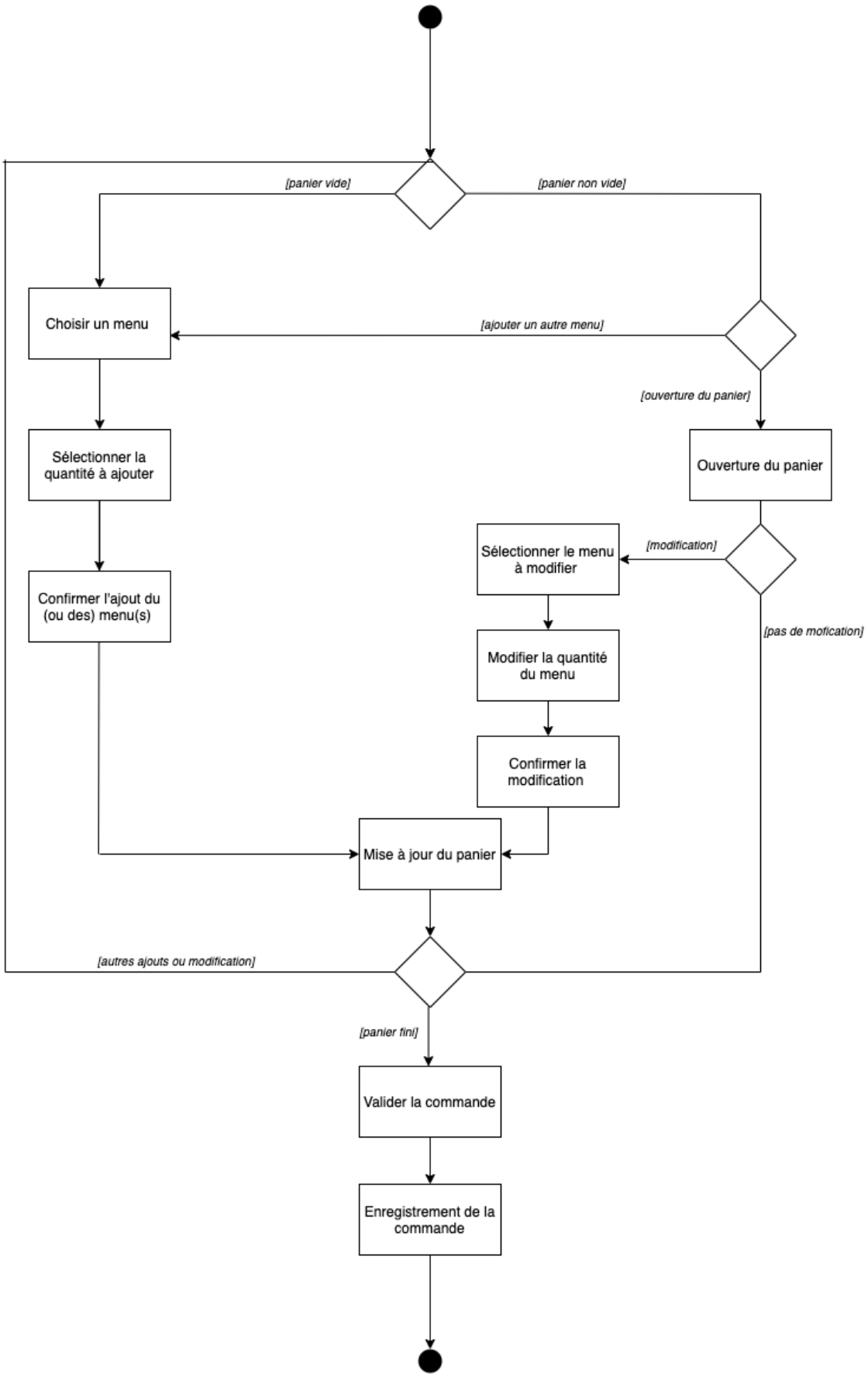

FIGURE 6 – Diagramme d'activité de l'utilisateur : création de commande

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*

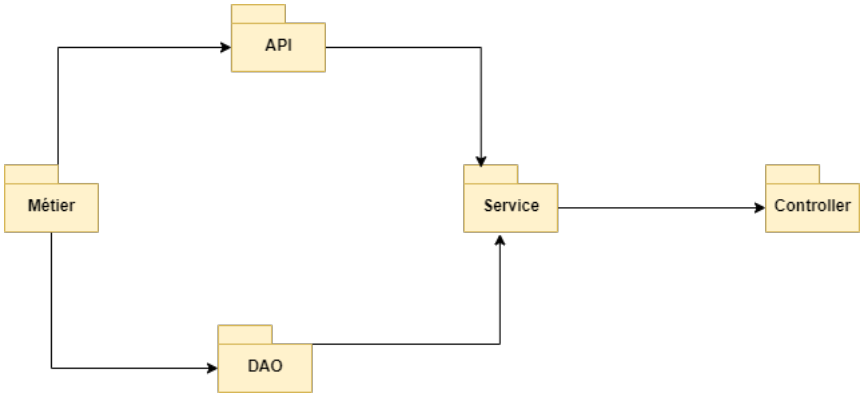

FIGURE 7 – Diagramme de packages de l'API

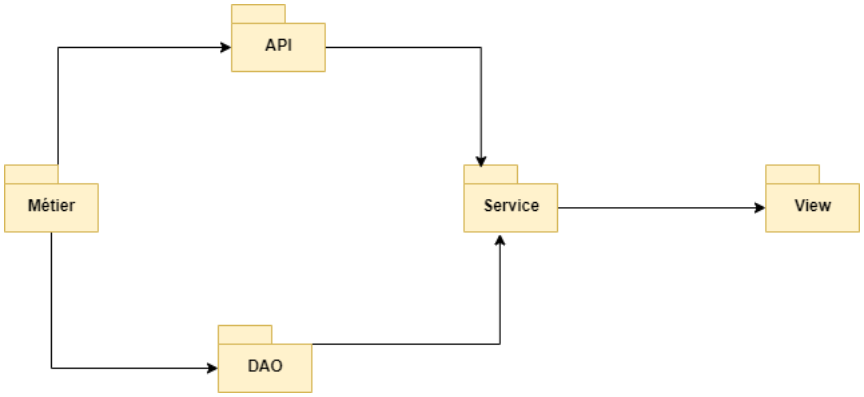

FIGURE 8 – Diagramme de packages du client

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 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.

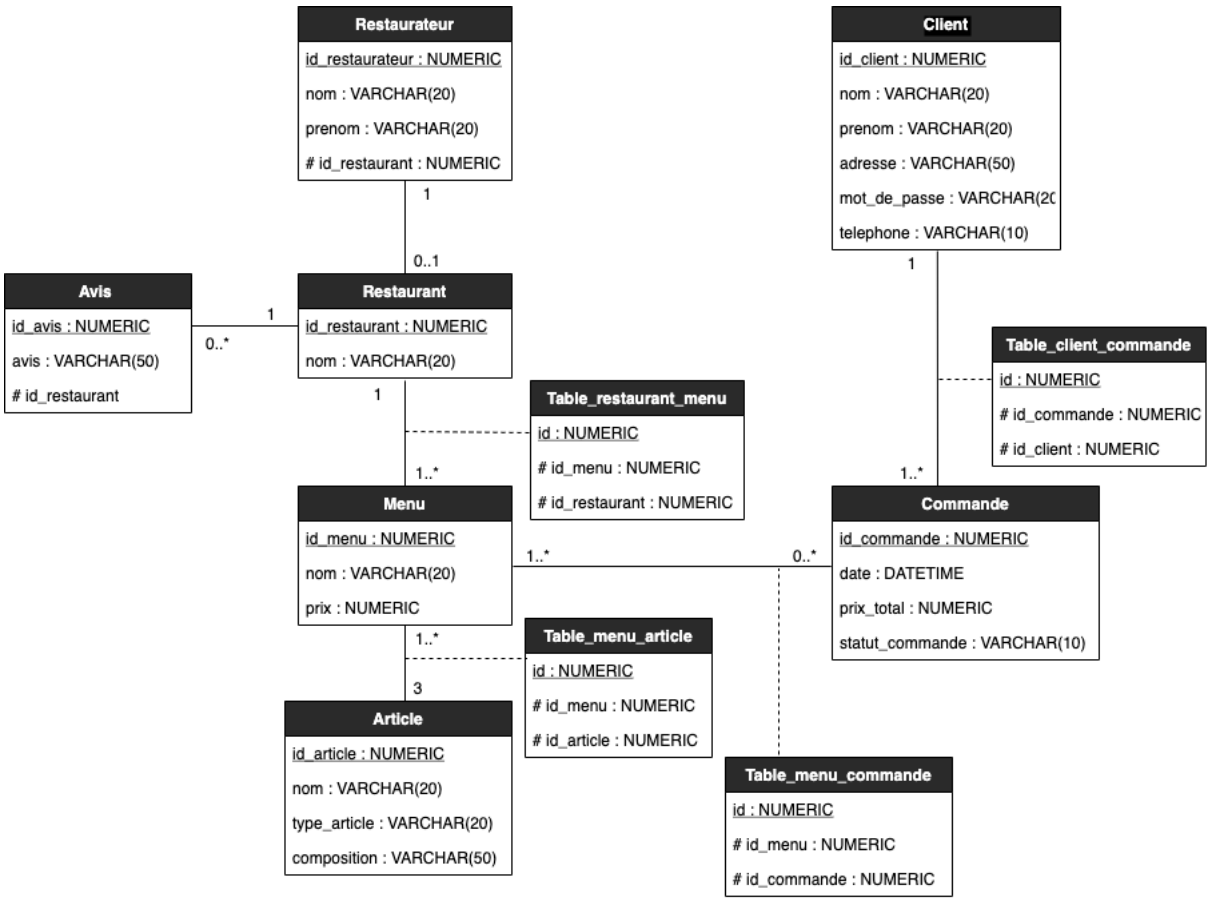

FIGURE 9 – Diagramme de base de données

Table	Variable	Type	Explication
Restaurateur	id_restaurateur	NUMERIC	Identifiant du restaurateur
	nom	VARCHAR	Nom de famille du restaurateur
	prénom	VARCHAR	Nom de famille du restaurateur
Restaurant	id_restaurant	NUMERIC	Identifiant du restaurant
	nom	VARCHAR	Nom du restaurant
Avis	id_avis	NUMERIC	Identifiant de l'avis
	avis	VARCHAR	Avis laissé par un client
Menu	id_menu	NUMERIC	Identifiant d'un menu
	nom	VARCHAR	Nom du menu
	prix	NUMERIC	Prix unitaire du menu
Article	id_article	NUMERIC	Identifiant d'un article
	nom	VARCHAR	Nom d'un article
	type_article	VARCHAR	Type d'article (plat, dessert ou boisson)
	composition	VARCHAR	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	VARCHAR	Statut de la commande (enregistrée, en préparation, en livraison, livrée)
Client	id_client	NUMERIC	Identifiant du client
	nom	VARCHAR	Nom du client
	prenom	VARCHAR	Prénom du client
	adresse	VARCHAR	Adresse du client (numéro, rue, code, postal, ville)
	mot_de_passe	VARCHAR	Mot de passe du client crypté
	telephone	VARCHAR	Numéro de téléphone du client

FIGURE 10 – 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 utilisateur 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'API client : la couche communiquant directement avec l'utilisateur (récupération des données d'entrée et affichage des résultats) ;
— l'API métier : envoie des requêtes soit vers notre base de données, soit vers l'API YELP et récupère les résultats. Ces derniers sont ensuite transmis au client pour affichage.

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.

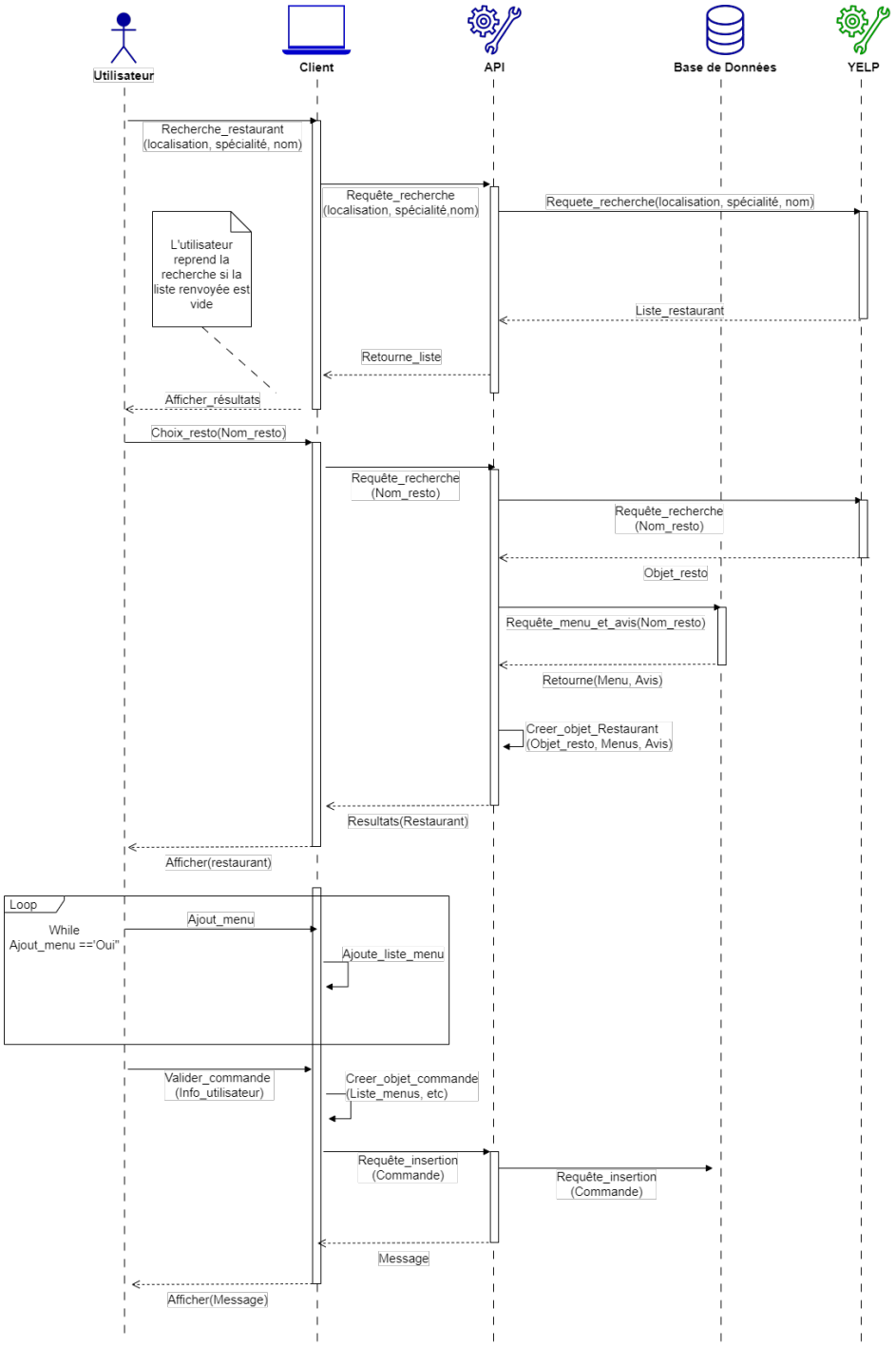

FIGURE 11 – Diagramme de séquences de l'exécution d'une commande

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.

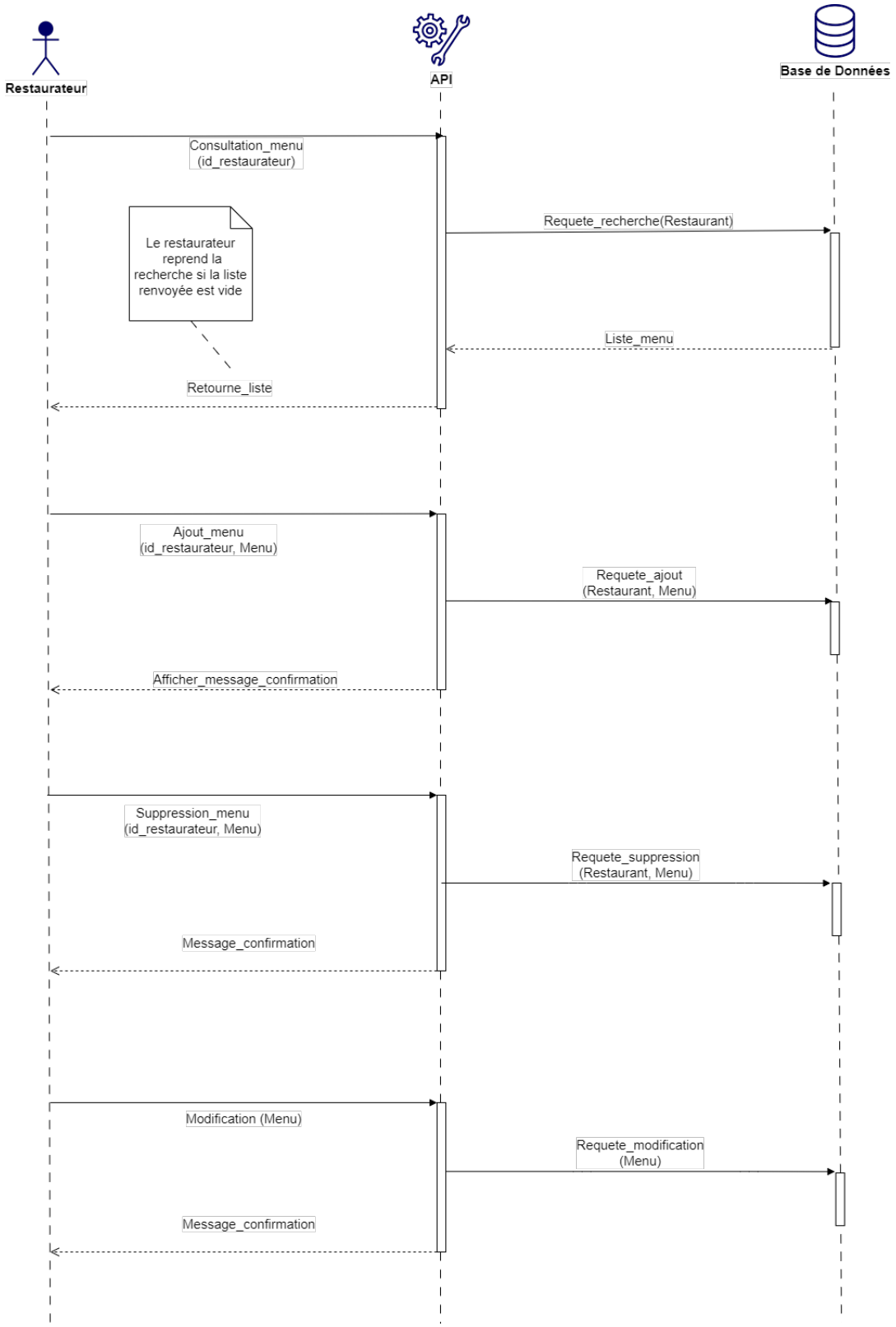

FIGURE 12 – Diagramme de séquences du restaurateur

2.9 Diagramme de classes

Nous avons décidé de scinder notre application en 7 classes pour commencer :

— La classe **Utilisateur** : Un utilisateur de l'application, que ce soit un client ou un restaurateur, possède un identifiant et s'identifie par son nom, son prénom ainsi que son numéro de téléphone. Son rôle dans l'application se résume à un booléen (0 pour un client et 1 pour un restaurateur).
— La classe **Restaurant** : Un restaurant possède un identifiant et un nom. On peut également voir si le restaurant est ouvert ou non grâce aux données de YELP. Cette classe comporte des méthodes permettant aux restaurateurs d'ajouter ou enlever des menus et leurs prix et aux clients d'ajouter des avis sur le restaurant en question. Pour se faire nous avons créer 2 autres classes : **Avis** et **Menus**.
— La classe **Avis** : Les avis écrits par les clients possèdent un identifiant.
— 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** : Un utilisateur peut avoir plusieurs adresses tandis qu'un restaurant n'en possède qu'une seule. Cette classe est composé des attributs numéro (num), adresse, ville et code postale permettant ainsi de bien localiser les consommateurs et de leur proposer des restaurants proche de chez eux.
— La classe **Commande** : Une commande possède un identifiant. 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.

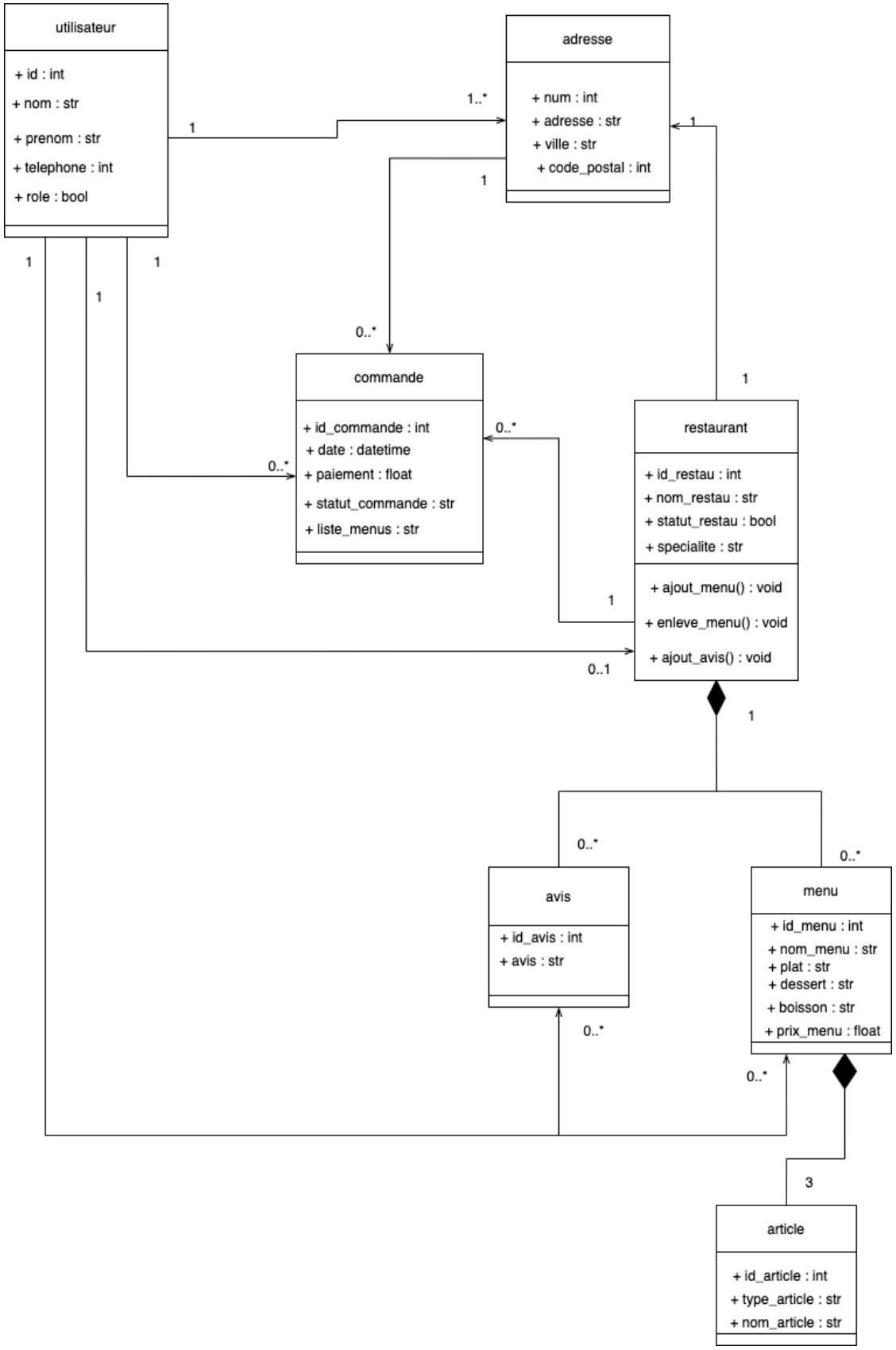

FIGURE 13 – Diagramme de classes

Conclusion

Notre projet a 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.

Concernant l'organisation du projet, nous avons désigné comme chef de projet Valentine Moulin, et nous nous sommes répartis les tâches équitablement. Nous avons convenu d'un rendez-vous hebdomadaire le jeudi matin d'une durée d'une heure et demie environ sous réserve que toute l'équipe soit disponible.
Pour se faire, nous avons établi différents diagrammes représentant la structure de notre projet. Nous avons commencé par les diagrammes de cas d'utilisation pour l'application cliente et un pour l'API afin de distinguer les différences.

* Le diagramme d'architecture permet de montrer l'intérêt de distinguer les deux cas d'utilisation et de faire deux applications : l'une gérant tout le traitement : l'API et l'application cliente permettant l'affichage.
* Le diagramme d'activité du client permet de voir l'enchaînement des opérations : de l'entrée dans l'application à l'enregistrement de la commande.
* Le diagramme de base de données permet de visualiser les différentes tables que nous aurons. En particulier, nous distinguons une table restaurateur de client. Ce diagramme permet de voir aussi les différentes liaisons entre les tables, par exemple le fait qu'un menu est composé de 3 articles. Nous considérons que la base de données Restaurant ne contient pour l'instant que l'identifiant et son nom puisque les informations complémentaires seront actualisées via l'API de Yelp.
* Le diagramme de classes permet, lui, de simplifier le problème en le divisant en petites "classes" comme son nom l'indique et de mieux comprendre les éléments qui constituent notre système et les relations qu'il y a entre eux.
* Dans les diagrammes suscités, la temporalité des opérations n'est pas soulignée. Afin de la mettre en relief, nous avons utilisé deux diagrammes de séquences, pour l'utilisateur et le restaurateur, mettant en exergue la succession des tâches qui se déclenchent lorsqu'un cas d'utilisation spécifique est lancé. La particularité du diagramme de séquence de l'utilisateur réside, entre autre, dans la présence de l'API de YELP qui fournit les dernières informations sur les restaurants (position, horaires, etc.).