Introduction :

Dans le cadre du projet REST réalisé en module INFO2 - nous avons réalisé une API REST d'un service de livraison de repas. Nous avons pris l'exemple du service de livraison Uber Eats, nous sommes partie sur ce choix car nous nous sommes rendus compte que de nos jours les plateformes de ce genre sont de plus en plus utilisées dans la vie quotidienne, nous le voyons par notre propre utilisation mais également par notre entourage.
Ce genre de service ont pris énormément d'envergure ces dernières années, par exemple la sponsor de la Ligue 1 qui est la ligue de football française est sponsorisé par Uber Eats nous l'appelons "Ligue 1 Uber Eats".

Pour en revenir à notre API REST, elle a pour but de rechercher, regrouper et créer des clients, des livreurs, des plats, des restaurants ainsi que de gérer des commandes. Pour réaliser ce projet, nous avons fait le choix technologique d'utiliser javascript qui est un langage que nous maîtrisons relativement bien, couplé à la spécification OpenAPI.
Dans un premier temps, ce rapport présentera l'API à l'aide de son diagramme de classe UML. Ensuite, il présentera les spécifications OpenAPI que nous avons écrites. Par la suite, il expliquera nos différents choix de développements, avant de finalement présenter un manuel orienté utilisateur.

Concernant la gestion de projet, nous avons démarré le projet en spécifiant le diagramme de classe de notre API. Après l'avoir finalisé nous avons commencé par créer la spécification Yaml de notre serveur REST sur Swagger et à partir de cette étape, Ethan ainsi que Loris se sont chargés de la partie programmation tandis que Rémy ainsi que moi (Mikâil) nous avons rédigé le rapport.

Présentation de l'API :

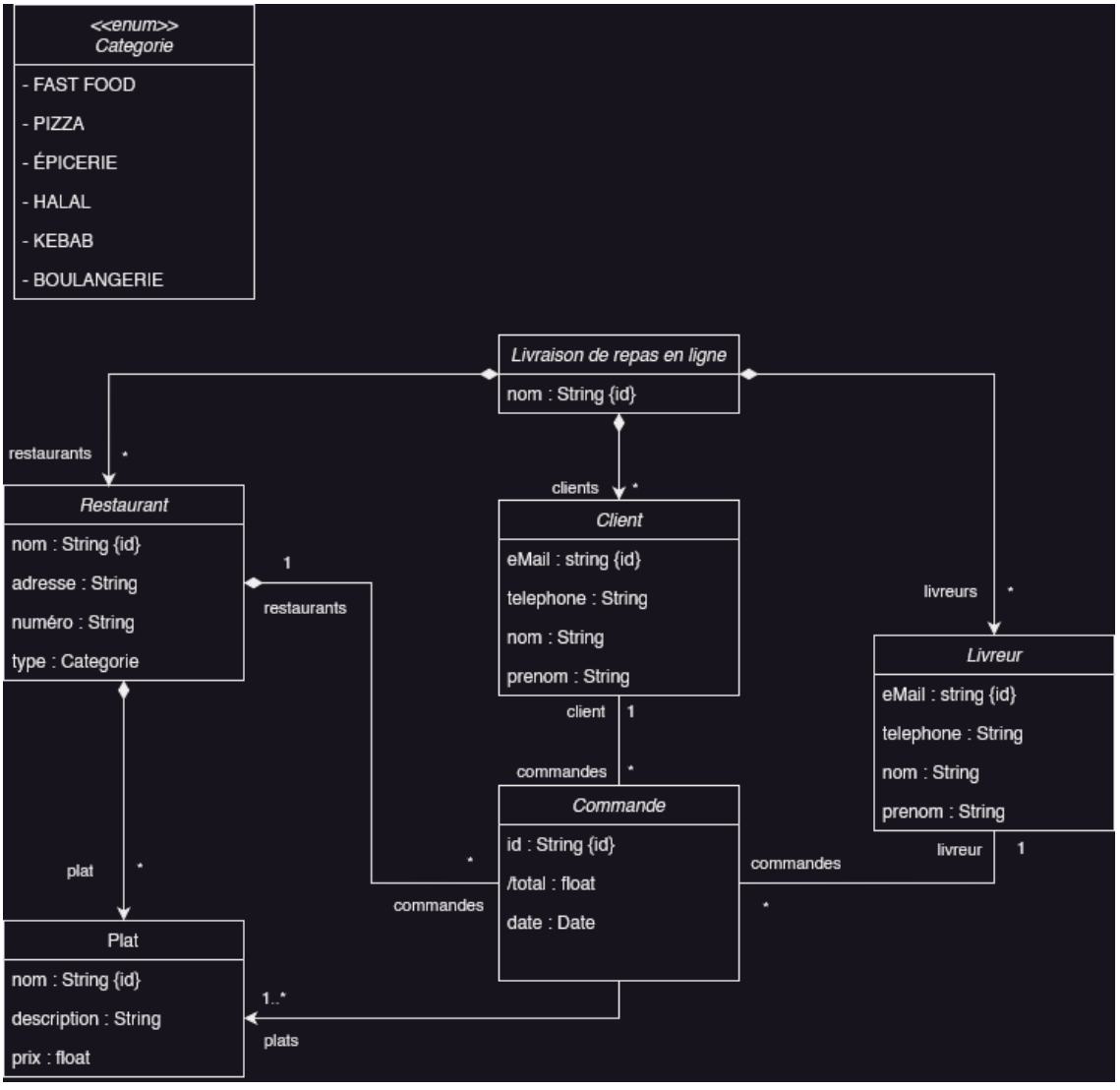
Voici notre diagramme de classe global, comme expliqué précédemment nous sommes partie sur une API de service de livraison de repas dans la même lignée que Uber Eats ou Deliveroo. La table au sommet nommée "Livraison de repas en ligne" correspond au nom de la plateforme, dans cette plateforme nous retrouvons différents restaurants d'où la liaison avec la table "Restaurant", cette cardinalité nous montre qu'une plateforme peut avoir plusieurs restaurants ou bien ne pas en voir. Chaque restaurant est identifié par le nom de l'établissement qui est unique, nous retrouvons également d'autres informations le concernant comme, une adresse, un numéro de téléphone et un type. Sur les plateformes de ce genre nous retrouvons différents types d'établissement d'où la spécification du type via une table énumérative "Categorie". Ces restaurants proposent différents plats caractérisés par un nom avec une rapide description et enfin un prix. De plus les établissements peuvent avoir plusieurs commandes qui sont identifiées par un id unique, la date à laquelle la commande a été passée ainsi que prix total de la commande, c'est commande concerne donc des plats est sont affiliés chacun à un seul livreur ainsi qu'à un seul client. Qui sont chacun identifiés par une adresse mail unique, et caractérisés par un numéro de téléphone, un nom et un prénom. C'est deux acteurs sont quant à eux

directement reliés à la plateforme donc nous retrouvons les tables “Restaurant”, “Client” et “Livreur” en haut du diagramme directement relié au premier niveau.

Nous retrouvons différentes composition sur le diagramme, si nous supprimons un restaurants de la plateforme alors tous les plats associés à ce restaurant seront également effacer de même pour les commandes associées à ce restaurant. Si nous prenons le niveau au-dessus, si nous supprimons l’intégralité de la plateforme alors les restaurants, clients et livreurs seront supprimés.

Concernant la table énumérative nous nous sommes limités qu’au différentes catégorie lister, si nous partions sur le même exemple que la plateforme numéro une Uber Eats nous aurions eu bien trop de catégorie différentes et parfois trop ressemblantes. D’où notre choix pour ces catégories.

Les liaisons entre la table de de commande et les tables de livreur, client et restaurant sont bidirectionnel car un restaurant peut avoir plusieurs commandes et chaque commande n’est rattaché qu’à un seul restaurant, un client peut passer plusieurs commande mais chaque commande ne concerne qu’un seul client et enfin un livreur peut livrer plusieurs commande mais chaque commande n’en possède qu’un seul . Or la liaison avec la table “Plat” est unidirectionnel car une commande se compose d’un ou plusieurs plat(s), mais dans un plan il n’y a pas de commande.



Spécification OpenAPI :

Concernant notre spécification OpenAPI, nous avons une première spécification qui fonctionne, nous pouvons naviguer dans notre API or cette version ne respecte pas notre diagramme de classe. Nous pouvons avoir accès aux plats directement depuis la plateforme sans passer par le restaurant. C'est pour cela que nous avons tenté tant bien que mal de corriger cela en faisant une seconde spécification plus correcte. Or par manque de temps nous n'avons pas pû finir le projet, chaque membre du groupe à eu des contretemps personnel d'où notre rendu tardif du projet, malgré la date de rendu dépassée.

Pour visualiser nos spécifications nous vous renvoyons vers les fichiers "Nouvelle scpécification.png" qui correspond à la dernière réalisé ainsi qu'au fichier "ancienne spécification.png" qui sont dans le répertoire Git.

Choix de développement :

Nous avons choisi de développer notre serveur REST en JavaScript (NodeJs) car nous avons déjà pu expérimenter cette solution dans les précédents TPs, certaines personnes du groupe ont plus de connaissance de d'expérience. A ce jour, nous n'avons pas encore développé de fonctionnalités avancées tel que du filtrage sur des restaurants ou bien des plats en particulier. Nous avons choisi de générer un serveur grâce à la spécification que nous avons réalisée en amont. Ce choix nous semblait le plus logique car il nous fournissait déjà des bases fiables pour notre serveur REST.

Manuel orienté utilisateur :

Le serveur peut être lancé via le fichier `readme.md` (avec certains IDE, c'est possible), ou en exécutant simplement la commande `npm start`, qui démarrera alors le serveur en local sur le port 8080 : http://localhost:8080/.
Pour ce qui est des données, elles ne nous semblaient pas être le cœur du projet, et c'est pour celà que nous avons choisi de générer une très petite quantité de données à la main. Ces données sont stockées dans différents fichiers Json, un par classe. Actuellement, il n'y a aucune persistance, et les modifications apportées aux données seront perdues au prochain redémarrage de l'API. La sauvegarde des données était une possibilité, mais cela signifiait que des données de mauvaise qualité, liées à des erreurs, pouvaient aussi être persistées.