CFPT Informatique

GrinEat

Documentation Technique TPI 2022

BRIAN.GRN
18/05/2022

Table des matières

1 Table des versions

N° de version	Date	Auteur	Changements apportés
1.0	18.05.2022	Brian Grin	Version finale

2 Introduction

2.1 Généralités

Ce document a pour but de décrire le procédé et le raisonnement qui a servi à réaliser le projet « GrinEat » dans le cadre de mon travail de pratique individuel (TPI). Ce travail sert à vérifier ma légitimité à l'obtention du CFC Informaticien à l'issu de ce travail.

GrinEat est une application web permettant de consulter les restaurants autour d'un périmètre choisi autour d'une adresse fournie et de consulter les menus du restaurant sélectionné.

L'affichage se fait sous 2 formats :

- Une liste de restaurant dans l'ordre croissant de la distance par rapport à l'adresse renseignée
- Une carte interactive affichant des points sur la carte là où se situe les restaurants

L'application communique directement avec un serveur qui sera développé en même temps que le client et qui s'occupera de toute la partie de traitement l'application aura juste a affiché ce qu'on lui fournit.

J'ai choisi ce projet car je souhaitais faire un projet qui serait différent des autres projets afin de proposer de l'originalité parmi les autres travaux de diplôme et je souhaitais proposer un travail dont je suis satisfait et qui comporte un challenge, ce projet comporte des fonctionnalités et logiques que je souhaitais aborder comme l'utilisation d'une carte interactive et la communication client-serveur. Je suis un consommateur occasionnel de service de livraison mais je trouvais la réalisation d'une application similaire intéressante. Malheureusement que 88h sont accordés pour réaliser ce travail j'aurais souhaité pousser plus loin et faire un affichage en temps réel du livreur sur la carte à l'aide d'une simulation mais cela demanderait plus de temps.

Néanmoins cela reste un bon projet et un très bon défi regroupant tout ce dont on attend d'un apprenti finalisant son apprentissage.

3 Analyse concurrentielle

3.1 JustEat ou Eat.ch

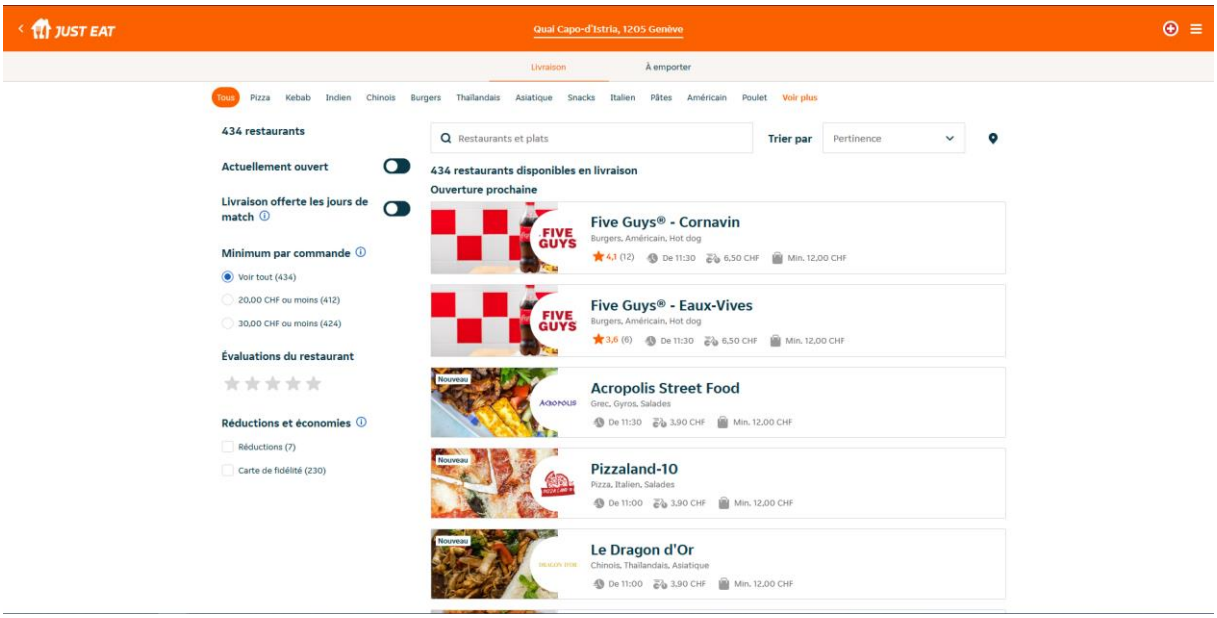

Figure 1 : Page principale de eat.ch

Il est le premier apparu en suisse. Il a un design plutôt basique mais les informations sont bien regroupées on prend le site en main très rapidement. Il permet aussi d'effectuer des commandes à distance à l'emporter pas que de la livraison.

Il reste un bon concurrent mais selon moi pas le plus grand, le design reste très basique et il n'y a pas de carte interactive qui sera surement un plus pour GrinEat.

3.2 Smood

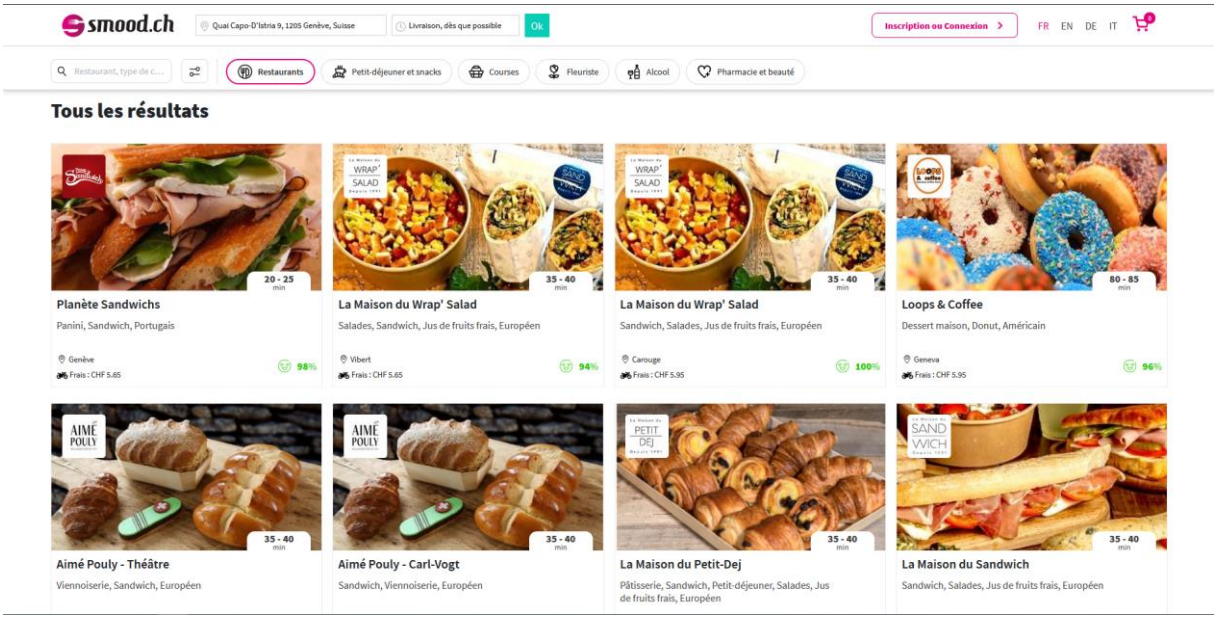

Figure 2 : Page principale de smood.ch

Il est le dernier arrivé à Genève (mais créé avant Ubereats). Son design parait plus moderne qu'eat.ch avec des animations mais je pense même si ce n'est qu'un détail il est important selon moi il n'y a pas de carte interactive de nouveau ici mais les restaurants sont proposés

en ligne, sur un écran 16:9 on a 4 restaurants par ligne comparé à eat.ch qui comportait juste une colonne de restaurant ce qui forçait à scroller plus pour consulter la liste des restaurants.

Il permet de planifier des commandes c'est un plus comparé au concurrent précédent et il permet aussi de faire des commandes à emporter depuis le site.

Surtout, il est celui avec le meilleur système de filtre séparant les filtres par cuisines et spécialités par exemple pour la cuisine thaï ou pour la spécialité Pad Thai.

En conclusion, il se place au-dessus d'eat.ch selon moi avec son design moderne et les petits détails qui font la différence.

3.3 UberEats

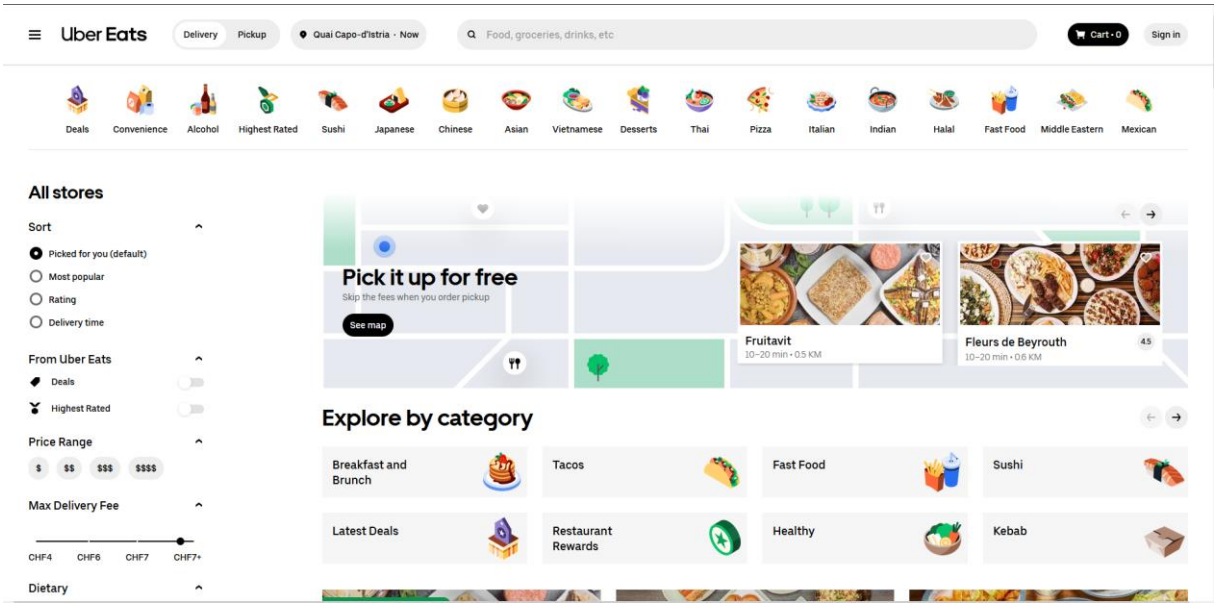

Figure 3 : Page principale UberEats

Finalement j'en parle en dernier car juste par son nom il est surement déjà considéré pour le plus grand concurrent, qui ne connait pas UberEats quand l'on parle de service de livraison surement celui le plus présent dans le monde filial de l'entreprise américaine Uber.

La renommée c'est bien beau mais que donne le site ? Il est surement celui avec le design le plus attirant beaucoup d'images présentes, pour les restaurants comme les autres sites mais aussi pour les catégories. Il propose la liste des restaurants en ligne aussi.

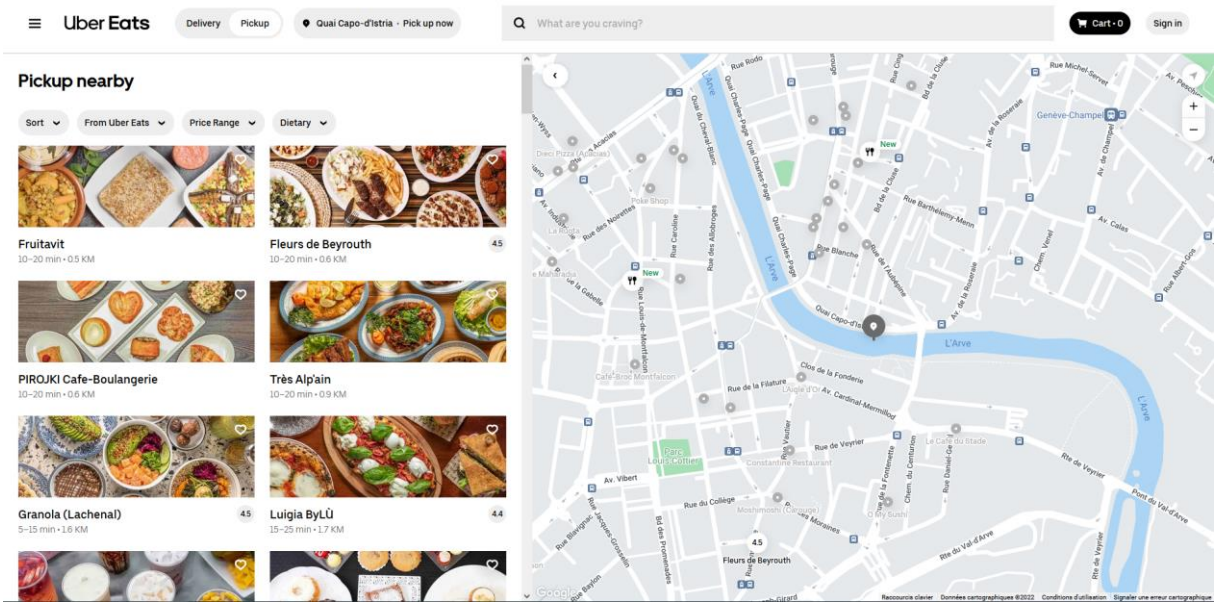

Figure 4 : Carte interactive UberEats

Surtout il est le seul des trois à proposer une carte interactive qui était jusqu'à maintenant notre fonctionnalité clé mais par chance cette carte interactive ne présente que les lieux pour faire des commandes à emporter et pas les restaurants qui livrent.

Finalement, il est surement le plus grand concurrent ici pour le projet GrinEat dans un contexte lucratif. Bien que la carte interactive soit pas présentée directement mais une fonctionnalité en plus laissant le choix à l'utilisateur de l'utiliser ou non et ne soit pas utiliser dans le même but que GrinEat. Néanmoins j'aurais un reproche quand même à ce site qui est le fait que la navigation comportant la barre de recherche et les catégories ne soit pas statique quand l'on scroll ce qui force l'utilisateur à remonter si finalement il souhaite effectuer une recherche ou filtrer par une catégorie.

3.4 Positionnement par rapport à la concurrence

Ce projet se réalisant dans un cadre scolaire pour l'obtention d'un diplôme, n'a donc pas de but lucratif réel. Cependant observer la concurrence et chercher à faire mieux reste une motivation il nous donne un bon sens de l'analyse pour juger au détail ce qu'on peut faire de mieux.

Des meilleurs projets similaires sont donc déjà présents et le temps accordé ne permet pas de réaliser un site aussi complet c'est pourquoi tout en restant conforme au cahier des charges. GrinEat regroupera dans la mesure du possible le meilleur de ses concurrents. Le point fort de GrinEat sera la carte interactive qui facilitera l'estimation du temps d'une livraison et l'usage pour l'utilisateur comparé à une liste qui donne juste l'adresse du restaurant on ne connait pas forcément toutes les adresses.

4 Rappel de l'énoncé

4.1 Organisation

Rôle	Prénom/Nom	Courriel
Elève	Brian Grin	brian.grn@eduge.ch
Maitre d'apprentissage	Antoine Schmid	antoine.schmid@edu.ge.ch
Experts	Pascal Court	pascal.court@hotmail.com
	Francesco Foti	francesco.foti@devinfo.net

4.2 Livrables

Au maitre d'apprentissage :

- Accès au repository GIT avec droit de clone
- Un dump de la base de données contenant des données de test

Aux experts :

- Un exemplaire PDF du rapport TPI, contenant :
  - Documentation technique
  - Manuel utilisateur
  - Code source
  - Planning

4.3 Matériel et logiciels à disposition

Matériel :

- Ordinateur fixe standard
- Ordinateur portable personnel sous l'OS : Manjaro
- Disque dur SSD Windows
- Disque dur SSD Manjaro
- Clavier et souris standard

Logiciels :

- **Visual Studio Code** (IDE)
- **PHPMyAdmin** pour la database
- **DBDiagram** pour les diagrammes
- **Postman** pour tester l'API

4.4 Description du site

GrinEat est un site web permettant de trouver des restaurants et de consulter leurs menus. La recherche des restaurants est basée selon une adresse saisie manuellement, le rayon (distance à vol d'oiseau entre le restaurant et l'adresse), le type de cuisine ou le nom du restaurant. Les restaurants trouvés sont affichés sur une carte et sous forme de liste. La sélection d'un restaurant permet d'afficher le menu.

Voici les fonctionnalités qui doivent être implémentée :

- Pour le serveur :
  - Retourner toutes les catégories de la base de données
  - Retourner les restaurants
  - Retourner le menu des restaurants

- Filtrer les restaurants par un rayon (distance à vol d'oiseau entre le restaurant et l'adresse)
- Filtrer par une ou des catégorie(s) de restaurants
- Recherche par nom de restaurant

- Pour le client :
  - Rechercher un restaurant selon :
    - Une adresse saisie manuellement
    - Le rayon (distance à vol d'oiseau entre le restaurant et l'adresse)
    - La catégorie
    - Le nom du restaurant
  - Consulter les restaurants trouvés sur une carte interactive
  - Consulter les restaurants trouvés selon une liste
  - Sélectionner un restaurant pour afficher sa page de menu

5 Méthodologie

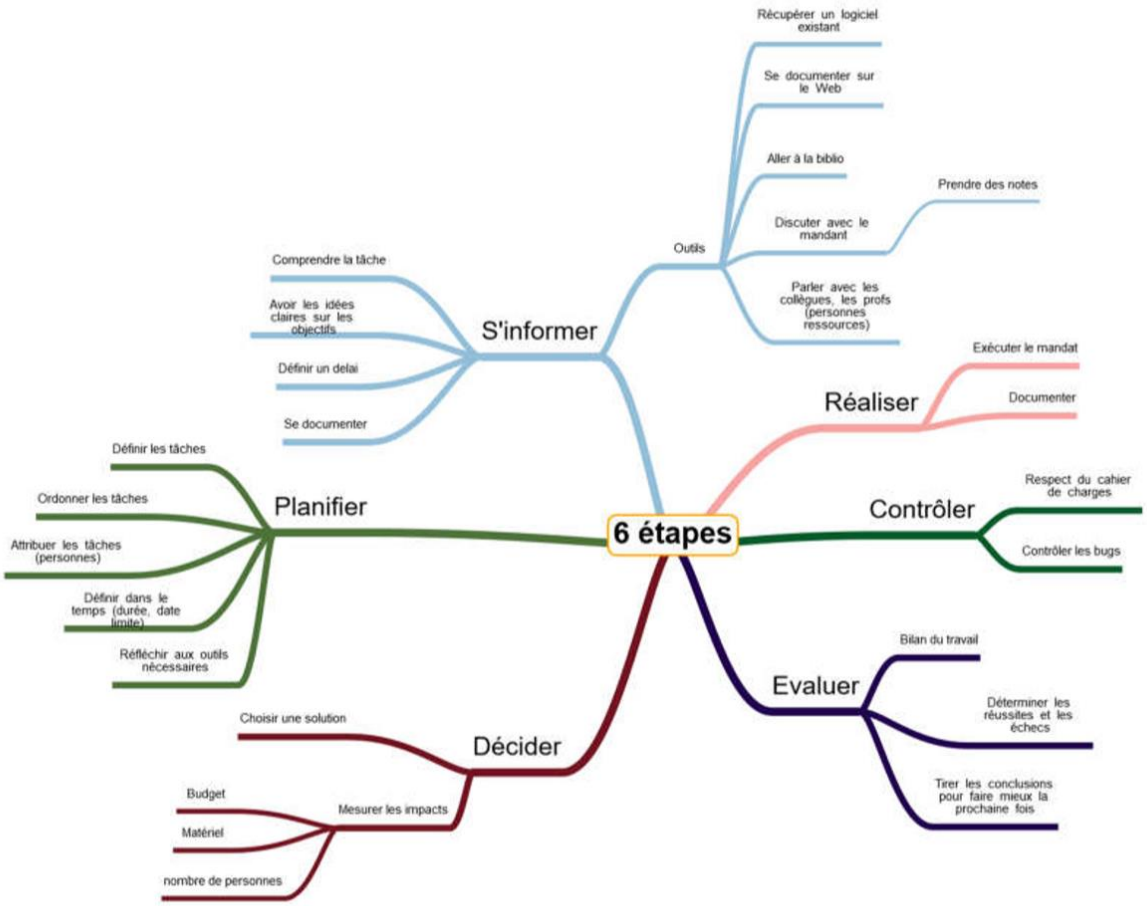

Figure 5 : La méthode en six étapes

5.1 S'informer

Au départ j'ai d'abord pris connaissance de ce qui m'était donné à faire dans l'énoncé. J'ai réfléchi un peu déjà à la manière dont je pourrais m'y prendre ce qui m'a mené à des incompréhensions dont j'ai pu demander l'explication par mail à mon maître d'apprentissage. J'ai aussi consulté les sites déjà présents du même type afin de les analyser et pouvoir m'en inspirer.

5.2 Planifier

J'ai ensuite commencé la réalisation de mon planning prévisionnel afin d'estimer le temps nécessaire pour la réalisation des tâches et avoir un fil conducteur auquel je puisse me suivre et me rendre compte de mon avance ou retard. La réalisation du planning m'aura mené à une première réflexion des étapes et de la manière dont j'allais procéder pour réaliser ce travail j'ai pu éclaircir certains points dont je n'étais pas sûr. Pour la séparation des tâches j'ai regroupé l'administratif ensemble et pour ce qui est du projet je l'ai séparé en deux (Front et Back end) le serveur et le client avec en priorité la réalisation du serveur car sans lui aucunes données ne sera affichée sur le client. J'ai réalisé tout cela sur une feuille Excel mis à disposition pour les élèves passant leur TPI où l'on a deux feuilles (le planning prévisionnel et le planning effectif).

5.3 Décider

Après discussion avec mon formateur et visualisation de solutions possibles. J'ai décidé d'ajouter une page d'accueil au site où l'on récupèrera l'adresse de l'utilisateur pour y baser tous les affichages.

J'ai décidé de faire une API pour récupérer les données et donc de séparer le client et le serveur.

5.4 Réaliser

J'ai commencé par créer la base de données et importer les données. Ensuite j'ai commencé par le développement du serveur finalement contrairement à l'idée de départ. J'ai ensuite commencé le client dès que le serveur était opérationnel en grande partie.

5.5 Contrôler

Pour chaque entrée du serveur API j'effectue des tests à l'aide d'un outil externe nommé Postman pour faire des requêtes http et tester que le serveur fonctionne bien avant de commencer l'implémentation du côté client. Après implémentation je test directement sur l'application que les fonctionnalités fonctionnent bien. Si un problème apparait je peux rapidement vérifier s'il provient du client ou du serveur et le fixer.

5.6 Evaluer

Pour terminer j'ai relu la grille d'évaluation point par point et je me suis assuré que tout était en ordre. J'ai reconsulté mon journal de bord (papier) afin de ravoir une vue d'ensemble de la réalisation de ce projet pour rédiger ma conclusion avec une rétrospective de mon travail.

6 Planification

6.1 Planning prévisionnel

Tâches à réaliser	Temps	02.05.2022	03.05.2022	04.05.2022	05.05.2022	09.05.2022	10.05.2022	11.05.2022	12.05.2022	16.05.2022	17.05.2022	18.05.2022	Total
Lecture enoncé	01:30	01:30											01:30
Immersion (Création git, vérifications du poste, temps pour adapt.)	02:00	00:30	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:10	00:05	00:05	02:00
Planification & Journal de bord (se documenter, réflexions, croquis, remplir planning)	05:00	02:00	00:30	00:20	00:20	00:20	00:20	00:20	00:20	00:10	00:10	00:10	05:00
													00:00
DEVELOPPEMENT BACK END SERVEUR API (NODE JS)													00:00
Création DB, importation fichier CSV et vérification/ajout de données	01:30		01:30										01:30
Créer un endpoint GET sur le serveur pour récupérer les restaurants	01:30		00:30	01:00									01:30
Créer un endpoint GET sur le serveur qui retourne toutes les catégories	01:30		00:30	01:00									01:30
Convertir adresse en coordonnées	02:00				01:00	01:00							02:00
Récupérer que les restaurants dans le rayon demandé	02:00					01:00	01:00						02:00
Trier par nom si un nom de restaurant est renseigné (SOUNDEX MYSQL)	02:00							01:00	01:00				02:00
Trier par catégorie si des catégories sont renseignés	02:00									02:00			02:00
													00:00
DEVELOPPEMENT FRONT END (HTML/CSS/JS)													00:00
Page d'accueil avec un champ demandant une adresse (API pour l'autocomplete ?)	01:00				01:00								01:00
Page principale avec une carte interactive centrée sur l'adresse renseignée	02:00				01:00	01:00							02:00
Liste à côté de la map affichant les restaurants dans l'ordre croissant de la distance	02:00						02:00						02:00
Champ pour la recherche par nom	01:00							01:00					01:00
Select option pour trier par catégorie	01:00							01:00					01:00
Select option editable pour trier par rayon	01:30								01:30				01:30
Mettre à jour l'affichage avec les nouvelles fonctionnalités implémentées	04:00				00:30					01:10	02:20		04:00
													00:00
ADMINISTRATION													00:00
Conception du document technique	27:00	04:00	03:00	03:00	01:00	02:00	02:00	02:00	02:00	02:00	03:00	03:00	27:00
Conception du manuel utilisateur	02:00		00:20		00:30				00:30		00:40		02:00
Débugging, tests, résolutions	17:30		01:00	02:00	02:00	02:00	02:00	02:00	02:00	02:00	01:15	01:15	17:30
Finitions, optimisations, révisions	08:00		00:30	00:30	00:30	00:30	00:30	00:30	00:30	00:30	00:30	03:30	08:00
	88:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	
												88:00	88:00

Figure 6 : Planning prévisionnel

6.2 Planning effectif

Tâches à réaliser	Temps	02.05.2022	03.05.2022	04.05.2022	05.05.2022	09.05.2022	10.05.2022	11.05.2022	12.05.2022	16.05.2022	17.05.2022	18.05.2022	Total
Lecture enoncé	01:30	00:30											00:30
Immersion (Création git, vérifications du poste, temps pour adapt.)	02:00	00:45		00:05	00:10	00:10	00:10	00:10	00:10				01:40
Planification & Journal de bord (se documenter, réflexions, croquis, remplir planning)	05:00	04:45	00:30	00:30	00:20	00:20	00:20	00:15	00:10	00:15	00:15		07:40
													00:00
DEVELOPPEMENT BACK END SERVEUR API (NODE JS)													00:00
Création DB, importation fichier CSV et vérification/ajout de données	01:30		01:20				00:20	00:30					02:10
Créer un endpoint GET sur le serveur pour récupérer les restaurants	01:30			01:30									01:30
Créer un endpoint GET sur le serveur qui retourne toutes les catégories	01:30		01:00										01:00
Convertir adresse en coordonnées	02:00			00:40									00:40
Récupérer que les restaurants dans le rayon demandé	02:00			01:15									01:15
Trier par nom si un nom de restaurant est renseigné (SOUNDEX MYSQL)	02:00							01:30					01:30
Trier par catégorie si des catégories sont renseignés	02:00			00:30	01:00								01:30
Remplir la table countries avec un script qui récupère tout les pays de l'API restcountries	00:25		00:25										00:25
Créer un endpoint GET sur le serveur pour récupérer les menus des restaurants	00:45				00:45								00:45
Refaire le serveur en PHP	03:30				03:30								03:30
Modifier le serveur pour accepté soit une adresse soit des coordonnées (latitude/longitude)	02:00								02:00				02:00
DEVELOPPEMENT FRONT END (HTML/CSS/JS)													00:00
Page d'accueil avec un champ demandant une adresse (API pour l'autocomplete ?)	01:00					04:00							04:00
Page principale avec une carte interactive centrée sur l'adresse renseignée	02:00					01:00	00:30	00:45					02:15
Liste à côté de la map affichant les restaurants dans l'ordre croissant de la distance	02:00					01:00	01:30						02:30
Champ pour la recherche par nom	01:00						00:30	00:45					01:15
Select option pour trier par catégorie	01:00						00:30	00:20					00:50
Select option editable pour trier par rayon	01:30						00:30	00:30					01:00
Page des menus des restaurants	02:00										02:30		02:30
Recherche avec ou sans autocomplete (coordonnées ou adresse si sans autocomplete)										00:45			00:45
ADMINISTRATION													00:00
Conception du document technique	27:00	02:00	04:45	01:00	00:45	00:30	03:00	03:15	03:00	03:00	02:15	05:00	28:30
Conception du manuel utilisateur	02:00											01:00	01:00
Débugging, tests, résolutions	17:30			02:00	01:00	00:30			02:10				05:40
Finitions, optimisations, révisions	08:00			00:30	00:30	00:30	00:40		02:30	02:00	03:00	02:00	11:40
	92:40	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	
												88:00	88:00

Figure 7 - Planning effectif

6.3 Bilan planning

J'étais parti au départ pour le développement de d'avancer petit à petit le serveur et le client dont j'ai finalement changé la méthode j'ai décidé finalement d'achever en premier le serveur puis de faire le client quand le serveur était presque terminé. Des tâches que je n'avais pas prévu à la base se sont montrés et j'ai perdu pas mal de temps lorsque j'ai refait le serveur dans une nouvelle technologie. J'effectuais les tests après chaque ajout de fonctionnalité sur le serveur et avant chaque implémentation sur le client pour être sûr que la fonctionnalité fonctionnait bien du côté du serveur au cas où une erreur survenait lors de l'implémentation puis j'ai essayé d'avancer la documentation au fur et à mesure malgré qu'on peut constater que peut-être plus de temps aurait été nécessaire.

7 Analyse organique

L'architecture du projet est séparée en deux parties distinctes client et serveur. Le projet dispose d'un client web et deux versions de serveur. C'est un modèle client-serveur qui est une structure d'application distribuée. Les serveurs sont utilisés comme des API et respectent les normes des API REST.

7.1 Technologies utilisées

- Côté client :
  - HTML
  - CSS
  - Javascript
  - Leaflet
  - MapBox

- Côté serveur :
  - NodeJS
  - PHP 7.4.28
  - MariaDB

7.2 Environnement

Pour la rédaction de la documentation technique, du manuel utilisateur et la conception du planning prévisionnel et effectif je travaille sur le disque dur Windows 10 fourni par l'école dans la cadre de l'apprentissage, Windows donnant accès à la suite Office pour Excel et Word qui sont les logiciels les plus pratiques.

Pour ce qui est du développement je code directement depuis mon ordinateur personnel portable sous Manjaro qui est une distribution Linux basée sur Arch Linux ou un autre disque dur SSD sous Manjaro sur l'ordinateur fixe. J'y ai installé LAMP en installant les paquets requis (style : apache, php, php-apache, mysql).

J'ai utilisé Visual Studio Code comme IDE pour le développement et phpMyAdmin pour les manipulations avec la base de données.

7.3 Description de la base de données

L'encodage par défaut de la base de données est utf8mb4 et le moteur est InnoDB.

7.3.1 Modèle logique de données

Un modèle de base de données était livré avec le cahier des charges je n'y ai apporté aucune grande modification à part le nommage des tables et champs et l'ajout d'un champ « nameEnglish » et « nameFrench » dans les tables « countries » et « categories », car lors de l'ajout j'avais ajouté les noms anglais en premier sans penser que l'affichage de l'application serait en français.

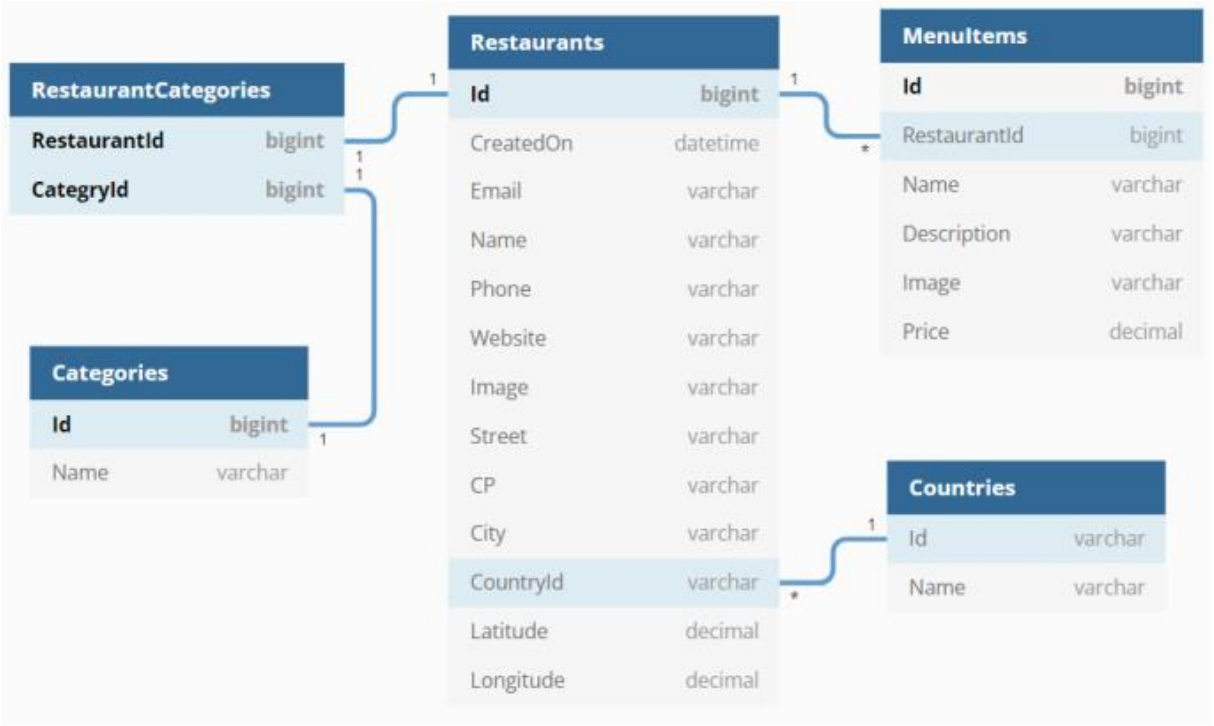

Figure 8 - Modèle livré avec le cahier des charges

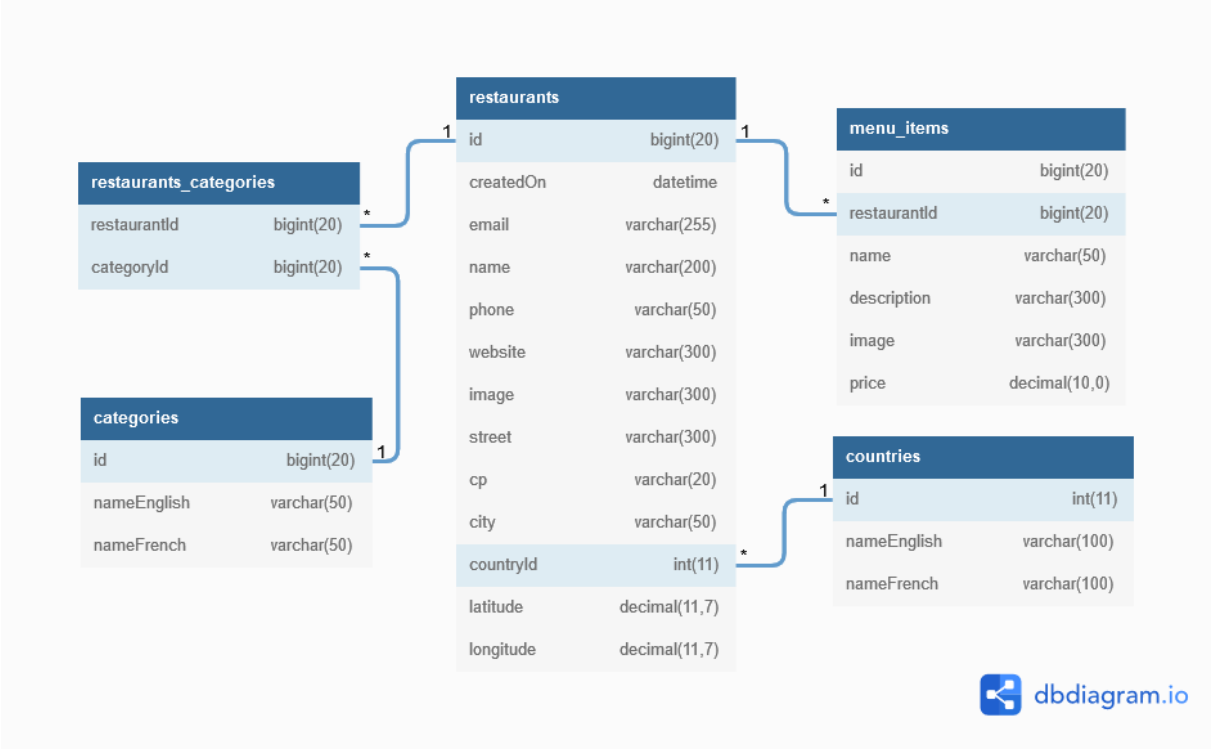

Figure 9 - Modèle utilisé avec l'application

7.3.1.1 Détails « restaurants »

Colonne	Type	Null	Valeur par défaut	Commentaire
id	bigint (20)	Non		Primary Key / Auto Increment
createdOn	datetime	Non	current_timestamp()	

email	varchar(255)	Oui	NULL	
name	varchar(200)	Non		
phone	varchar(50)	Oui	NULL	
website	varchar(300)	Oui	NULL	
image	varchar(300)	Non		
street	varchar(300)	Non		
cp	varchar(20)	Non		
city	varchar(50)	Non		
countryId	int(11)	Non		Index / Foreign Key
latitude	decimal(11,7)	Non		
longitude	decimal(11,7)	Non		

7.3.1.2 Détails « menu_items »

Colonne	Type	Null	Valeur par défaut	Commentaire
id	bigint (20)	Non		Primary Key / Auto Increment
restaurantId	bigint(20)	Non		Index / Foreign Key
name	varchar(50)	Non		
description	varchar(300)	Non		
image	varchar(300)	Non		
price	decimal(10,0)	Non		

7.3.1.3 Détails « countries »

Colonne	Type	Null	Valeur par défaut	Commentaire
id	bigint (20)	Non		Primary Key / Auto Increment
nameEnglish	varchar (100)	Non		
nameFrench	varchar(100)	Non		

La table countries a été rempli à l'aide d'un script fait par moi-même avec lequel j'ai appelé l'API « restcountries » pour récupérer tous les pays et je les ai insérés dans la base de données.

7.3.1.4 Détails « restaurants_categories»

Colonne	Type	Null	Valeur par défaut	Commentaire
restaurantId	bigint (20)	Non		Primary Key / Foreign Key
categoryId	bigint (20)	Non		Primary Key / Foreign Key

7.3.1.5 Détails « categories »

Colonne	Type	Null	Valeur par défaut	Commentaire
id	bigint (20)	Non		Primary Key / Auto Increment
nameEnglish	varchar (50)	Non		
nameFrench	varchar(50)	Non		

7.4 Arborescence du projet

7.4.1 Arborescence du serveur NodeJS

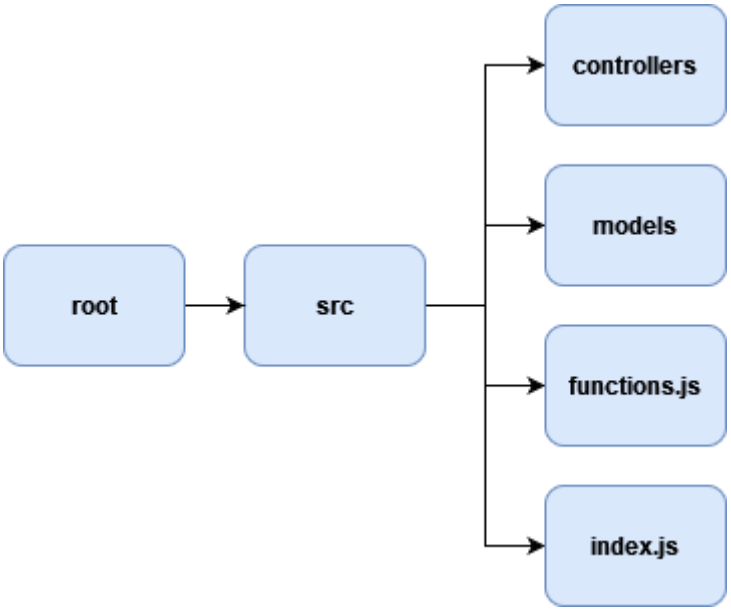

Figure 10 - Schéma représentant l'arborescence du serveur NodeJS

7.4.2 Arboresence du serveur PHP

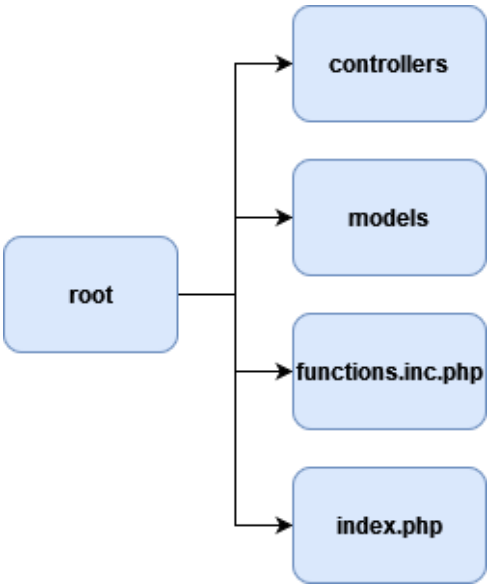

Figure 11 - Schéma représentant l'arborescence du serveur PHP

7.4.3 Arborences des serveurs

Les deux serveurs sont très similaires voir même casis identiques que ce soit au niveau de l'arboresence ou bien même du code car les deux sont faits en architecture MVC (Modèle-vue-contrôleur) la vue dans ce projet étant donné l'affichage des données au format JSON.

Le choix au départ de partir sur un serveur se justifie par le fait que le cahier des charges ne stipulait aucune technologie obligatoire et confirmer par mon formateur je me sentais plus à l'aise en JavaScript pour faire des appels à des APIs externe (pour convertir une adresse reçue en géocodage)

Finalement j'ai eu la visite d'un expert et le choix n'étant pas attendu j'ai remis en question mon choix je me suis documenté pour partir en PHP car j'avais aussi un squelette disponible fait par mes soins et la logique étant déjà codé côté NodeJS il suffisait de reproduire le code en PHP. Le PHP nécessite aucune librairie pour faire un serveur tout le code est facilement explicable ce qui facilitera la présentation à l'oral car la compréhension du code est plus simple.

7.4.3.1 Dossier « root »

Ceci est la seule différence entre les deux serveurs ce dossier côté NodeJS contient les fichiers du type : « package.json », pour la gestion des paquets grâce à NPM. Alors que côté PHP c'est bien la racine du projet où se trouve l'index et les scripts.

7.4.3.2 Dossier « src »

Il est présent que du côté NodeJS et il est l'équivalent du dossier « root » du serveur PHP.

7.4.3.3 Dossier « controllers »

Il contient tous les contrôleurs de chaque route.

7.4.3.4 Dossier « models »

Il contient les classes PHP dont le nom et les propriétés font référence aux tables de la base de données et la classe MyPdo.php côté PHP et Db.js côté NodeJS pour la connexion à la base de données.

7.4.3.5 Fichier « functions.inc.php ou function.js »

Il contient les fonctions utiles comme la fonction pour convertir une adresse en coordonnées latitude longitude ou bien la formule d'Haversine pour calculer une distance à vol d'oiseau.

7.4.3.6 Fichier « index.php ou index.js »

Il est le fichier principal qui gère les routes et appelle le bon contrôleur.

7.4.4 Arborescence du client

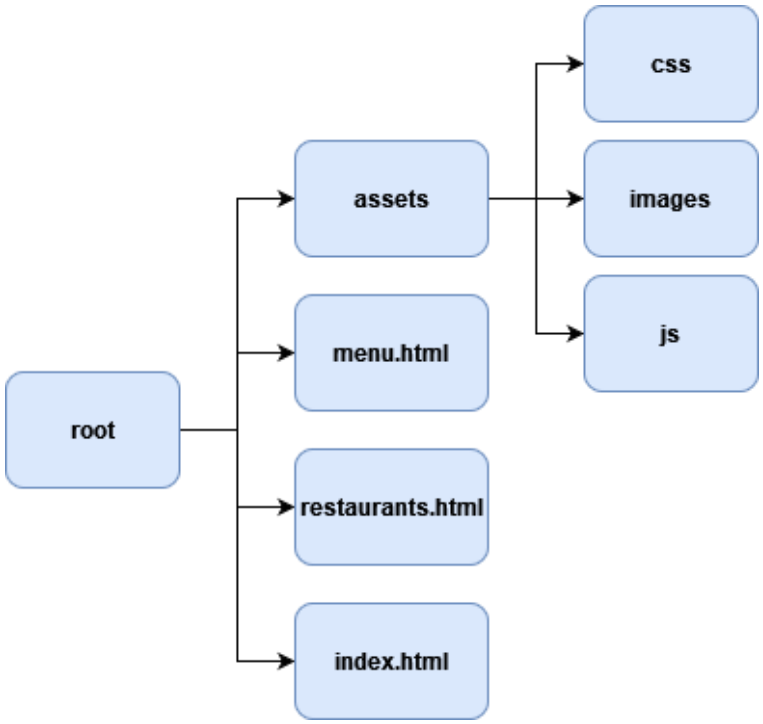

Figure 12- Schéma représentant l'arborescence du client

7.4.4.1 Dossier « root »
C'est la racine du client dans ce cas le dossier s'appelle « grineat-client » où se trouve tous les éléments du projet.

7.4.4.2 Dossier « assets »
Il contient les éléments que l'on « importe » dans le code.

7.4.4.3 Dossier « css »
Il contient toutes les feuilles de style du site dans notre cas il n'y a que style.css

7.4.4.4 Dossier « images »
Il contient les images statiques comme les icones des marqueurs sur la carte interactive ou la loupe se trouvant dans la barre de recherche par nom.

7.4.4.5 Dossier « js »
Contient les scripts JavaScript de chaque page.

7.4.4.6 Fichier menu.html
C'est la page pour consulter les menus l'affichage est dynamique à partir de l'idRestaurant référencé dans l'url.

7.4.4.7 Fichier « restaurants.html »
C'est la page où se trouve les restaurants que l'on peut consulter.

7.4.4.8 Fichier « index.html »
C'est la page principale sur laquelle on arrive quand on accède au site.

7.5 Documentation de l'API

Titre	**Récupérer toutes les catégories**
URL	**/categories**
Méthode	**GET**
Paramètre d'URL	Aucun
Paramètre des données	Aucun

Titre	**Récupérer toutes les restaurants**
URL	**/restaurants**
Méthode	**POST**
Paramètre d'URL	Aucun
Paramètre des données	**Un des deux champs entre address et coordinates est obligatoire voici 2 exemples différents de la structure avec chacun des champs :**

```
{
    "coordinates": { // obligatoire
        "longitude": [decimal],
        "latitude": [decimal]
    },
    "radius": [int], // recherche par rayon (optionel)
    "categories": [ // recherche par catégories (optionel)
        [int]
    ],
    "name": [string], // recherche par nom (optionel)
}
```

Figure 13 - Exemple de paramètres de données utilisant les coordonnées

```
{
    "address": [string], // nom de rue obligatoire
    "radius": [int], // recherche par rayon (optionel)
    "categories": [ // recherche par catégories (optionel)
        [int]
    ],
    "name": [string], // recherche par nom (optionel)
}
```

Figure 14 - Exemple de paramètre de données utilisant une adresse

Les autres champs eux sont optionnels voici un exemple avec les champs remplis :

```
{
    // Les 2 champs peuvent être remplis les coordonnées seront utilisés en priorité
    "address": "Quai Capo-d'Istria 9",
    "coordinates": {
        "latitude": 46,
        "longitude": 6
    },
    "radius": 6, // "6 km" est aussi correct
    "categories": [ // Il faut renseigner l'id de la catégorie récupéré au préalable
        6,
        1
    ],
    "name": "McDo"
```

Figure 15 - Exemple de paramètre de données remplis

Titre	**Récupérer tous les menus d'un restaurant**
URL	**/restaurants/{id}/menus**
Méthode	**GET**
Paramètre d'URL	**Required :** idRestaurant=[integer] Exemple : idRestaurant=2
Paramètre des données	Aucun

7.6 Fonctions remarquables

7.6.1 Index.php du serveur

```php
// Include CORS headers
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE');
header('Access-Control-Allow-Headers: X-Requested-With');
header('Content-Type: application/json');

// Include Models & functions
include "functions.inc.php";
include "models/MyPdo.php";
include "models/Category.php";
include "models/Restaurant.php";
include "models/MenuItems.php";

// create a api variable to get HTTP method dynamically
$api = $_SERVER['REQUEST_METHOD'];
// get headers
$headers = getallheaders();
// get endpoint from url
$endpoint = $_GET['endpoint'] ?? '';
// get id from url
$id = intval($_GET['id'] ?? '');
// get context
$context = $_GET['context'] ?? '';

switch ($endpoint) {
    case '':
        echo json_encode("Welcome to the API");
        break;
    case 'categories':
        include "controllers/categoriesController.php";
        break;
    case 'restaurants':
        switch($context) {
            case '':
                include "controllers/restaurantsController.php";
                break;
            case 'menus':
                include "controllers/menu_itemsController.php";
                break;
            default:
            http_response_code(400);
            break;
        }
        break;
    default:
        http_response_code(404);
        break;
}
```

Voici un exemple du fichier index.php du serveur PHP il est presque identique côté NodeJS on peut constater qu'on y inclut les modèles on récupère la méthode de l'appel que les clients effectuent sur le serveur à l'aide d'un fichier .htaccess on réécrit l'URL pour assurer les normes REST par exemple :

« grineat-api/restaurants » on récupère le texte après le « / » qu'on garde dans la variable endpoint puis on le passe dans le switch afin d'appeler le bon contrôleur si l'endpoint existe.

8 Analyse fonctionnelle

Le projet ne comporte pas énormément de fonctionnalités visuelles et le manuel utilisateur les décrits déjà très bien je vais donc juste survoler l'interface.

8.1 Interface

8.1.1 Page principale

Figure 16 - Page principale

Un champ se trouve sur cette page on y attend une adresse une fonctionnalité d'autocomplétion est présente pour faciliter la recherche.

8.1.2 Page des restaurants

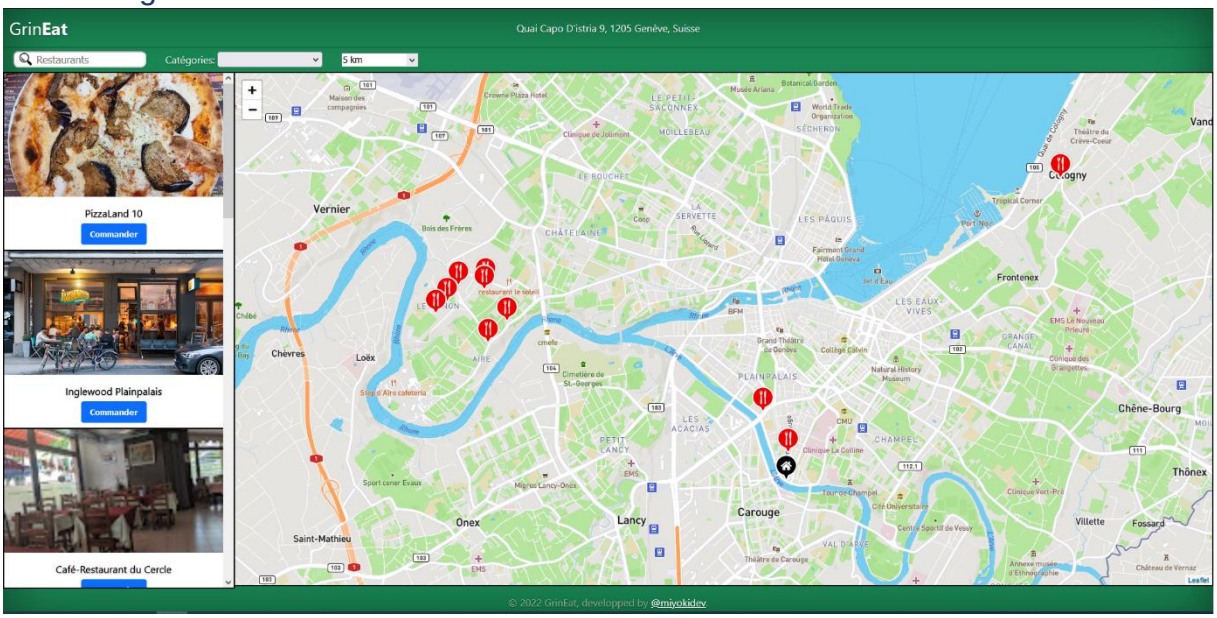

Figure 17 - Page des restaurants

Description des éléments de la page de haut en bas, gauche à droite :

- Le logo qui permet de retourner sur la page principale et effectuer une nouvelle recherche à partir d'une autre adresse (Texte **GrinEat)**
- L'adresse situé au milieu de l'header permettant de recentrer la carte sur l'adresse renseignée
- Un champ de recherche par nom de restaurant
- Un select qui permet de filtrer par une catégorie
- Un select éditable qui permet de filtrer par un rayon en kilomètre
- Une liste dynamique déroulante avec des cartes ayant une image, un nom d'un restaurant et un bouton commander
- Une carte interactive affichant les restaurants qui sont aussi présents dans la liste dynamique

8.1.3 Page des menus

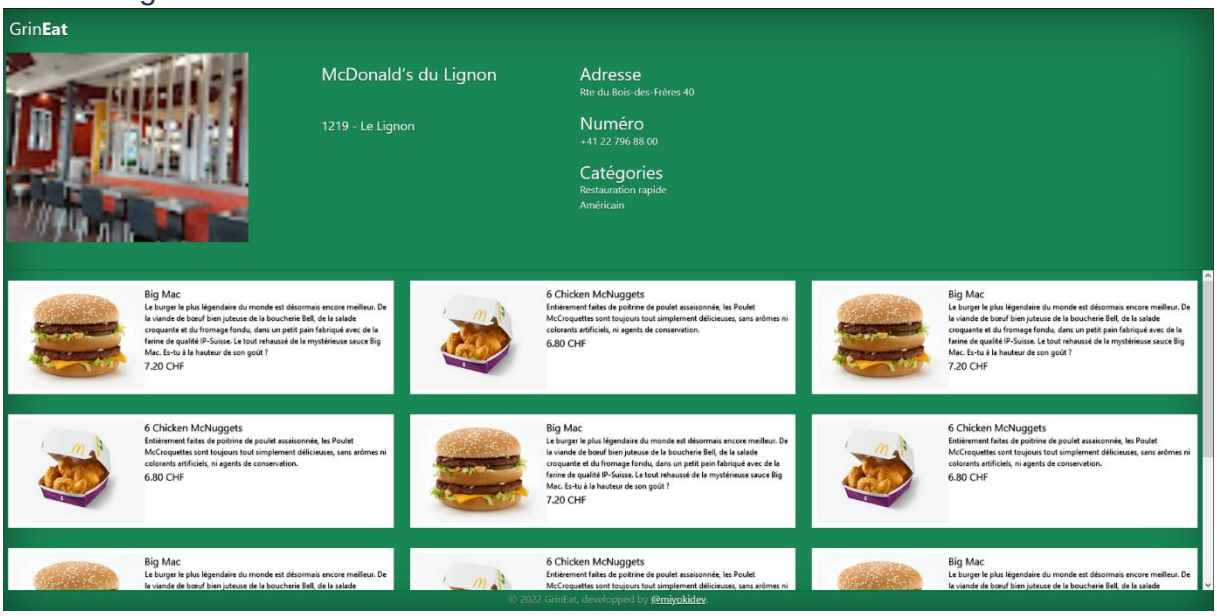

Figure 18 - Page des menus

Cette page est affichée dynamiquement à grâce à l'idRestaurant mis en paramètre dans l'URL de la page. On y affiche le nom du restaurant l'adresse, la ville, le code postal, le numéro du restaurant et les 2 premières catégories.

En dessous de la description du restaurant on affiche dynamiquement les menus du restaurant.

9 Plan de test

Dans le cadre du développement de mon application, le cahier des charges exige que je mette en place un protocole de test.

9.1 Périmètre

Pour ce qui est du périmètre de mon protocole de test, le plus important sera la communication entre le client et le serveur puis l'affichage ensuite je vais donc séparer mes tests en deux points :

1. Les tests uniquement côtés serveurs (que l'API reçoive bien les données envoyées par http, le traitement et qu'elle renvoie les bonnes informations Ex : je demande la liste des catégories le serveur me renvoie bien un JSON avec toutes les catégories) Cette partie sera testé avec Postman.
2. Les tests côtés client (On affiche les bonnes informations reçus par le serveur on envoie bien les bonnes informations Ex : Quand je cherche McDonald's on m'affiche des McDonald's sur la carte).

9.2 Description des tests

Nom du test	1.1 | Récupérer les catégories
Description	On souhaite récupérer un JSON contenant les catégories existant dans la base de données.
Comment le tester	Effectuer une requête HTTP GET sur l'endpoint /categories du serveur

Nom du test	1.2 | Récupérer les restaurants sans filtre
Description	On souhaite récupérer un JSON contenant les restaurants en envoyant juste notre adresse sans filtre il retourne les restaurants dans un périmètre de 5 km par défaut.
Comment le tester	Effectuer une requête HTTP POST sur l'endpoint /restaurants du serveur avec dans le body en JSON une propriété address contenant une addresse.

Nom du test	1.3 | Erreur 500 pas d'adresse
Description	On teste un cas où l'on ne renseigne pas d'adresse et on s'attend à une erreur 500 renvoyé par le serveur.
Comment le tester	Effectuer une requête HTTP POST sur l'endpoint /restaurants du serveur avec rien dans le body

Nom du test	1.4 | Récupérer les restaurants en filtrant par catégories
Description	On souhaite récupérer un JSON contenant les restaurants filtrés par les catégories qu'on envoie et notre adresse.
Comment le tester	Effectuer une requête POST sur l'endpoint /restaurants du serveur avec dans le body en JSON une propriété address contenant l'adresse et une propriété categories contenant un tableau de 1 ou plusieurs catégories.

Nom du test	1.5 | Récupérer les restaurants en filtrant par nom
Description	On souhaite récupérer un JSON contenant les restaurants dont le nom ressemble au nom qu'on envoie et notre adresse.

Comment le tester	Effectuer une requête POST sur l'endpoint /restaurants du serveur avec dans le body en JSON une propriété address contenant l'adresse et une propriété name contenant le nom qu'on cherche.

Nom du test	1.6 | Récupérer les restaurants en filtrant par rayon
Description	On souhaite récupérer un JSON contenant les restaurants dans un rayon autour de l'adresse qu'on envoie.
Comment le tester	Effectuer une requête POST sur l'endpoint /restaurants du serveur avec dans le body en JSON une propriété address contenant l'adresse et une propriété radius avec le nombre en kilomètre exemple : 5 (pour 5km).

Nom du test	1.7 | Erreur 500 adresse chaine vide
Description	On teste un cas où l'on ne renseigne une adresse vide et on s'attend à une erreur 500 renvoyé par le serveur.
Comment le tester	Effectuer une requête HTTP POST sur l'endpoint /restaurants du serveur avec un JSON contenant la propriété adresse mais une chaine vide

Nom du test	1.8 | Récupérer les éléments d'un menu d'un restaurant
Description	On souhaite récupérer les éléments du menu d'un restaurant, recevoir un JSON contenant les éléments quand on renseigne l'id d'un restaurant
Comment le tester	Effectuer une requête HTTP GET sur l'endpoint /restaurants/{id}/menu

Nom du test	2.1 | Rentrer une adresse valide sur la page d'accueil
Description	On souhaite rentrer notre adresse et être rediriger sur la page des restaurants.
Comment le tester	Rentrer une adresse à l'aide de l'autocomplete séléctionnez l'adresse que vous souhaitez.

Nom du test	2.2 | Entrer une adresse invalide sur la page d'accueil
Description	On souhaite rentrer une adresse invalide et recevoir une alerte indiquant que l'adresse est invalide.
Comment le tester	Rentrer une adresse aléatoire sans l'aide de l'autocomplétion.

Nom du test	2.3 | Tri par nom
Description	On souhaite filtrer notre recherche de restaurants par nom de restaurant
Comment le tester	Effectuer le test 1.2 en y ajout dans le body du JSON une propriété name avec le nom qu'on cherche

Nom du test	2.4 | Tri par rayon
Description	On souhaite filtrer notre recherche de restaurants par rayon entre l'adresse et les restaurants
Comment le tester	Effectuer le test 1.2 en y ajout dans le body du JSON une propriété radius avec le rayon soit en nombre entier soit en string contenant un nombre

Nom du test	2.5 | Tri par catégories
Description	On souhaite filtrer notre recherche de restaurants par catégories
Comment le tester	Effectuer le test 1.2 en y ajout dans le body du JSON une propriété categories avec un tableau de nombre entier étant l'id des catégories dont on souhaite filtrer

9.3 Scénarios de tests

Tableau récapitulatif de l'état des tests au fur et à mesure des 11 jours.

✗ = signifie que le test est raté.

✓ = signifie que le test est passé

= signifie que le test est passé partiellement

Test	J 1	J 2	J 3	J 4	J 5	J 6	J 7	J 8	J 9	J 10	J 11
1.1	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.2	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.3	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.4	✗	✗	#	✓	✓	✓	✓	✓	✓	✓	✓
1.5	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
1.6	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.7	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
1.8	✗	✗	#	✓	✓	✓	✓	✓	✓	✓	✓
2.1	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
2.2	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
2.3	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
2.4	✗	✗	✗	✗	✗	#	✓	✓	✓	✓	✓
2.5	✗	✗	✗	✗	✗	#	✓	✓	✓	✓	✓

10 Conclusion

10.1 Difficultés rencontrées

Je n'ai pas réellement rencontré de grande difficulté dans le développement puisque je maitrisais mon sujet et savait plus ou moins à quoi m'attendre.

Mes plus gros obstacles je dirais ont été le départ, je suis habitué à me lancer rapidement dans le développement et devoir tout planifier à l'avance réfléchir aux tâches en avance fut difficile pour moi j'avais du mal à directement visualiser les tâches, la concentration rester concentré 8h par jours sans perdre de temps car chaque minute compte et le faux départ en réalisant le serveur en nodejs puis en changeant après pour le PHP qui m'aura quand même coûté presque une journée.

10.2 Améliorations possibles

Une retouche complète du design au niveau de la CSS ne ferait pas de mal au projet qui n'est pas totalement responsive et dont l'affichage peut être douteux selon la taille de l'écran.

L'ajout d'une table organisation dans la base de données pour les entreprises ayant plusieurs restaurants dans une même ville comme McDonald's.

La distinction dans les catégories pour différencier type de cuisine et spécialités.

L'usage d'un framework d'interface pour faciliter l'affichage dynamique quand on récupère les données côté client.

Pour finir les fonctionnalités basiques qui aurait dû être ajouté si le temps le permettait :

- Page de connexion/inscription
- Système d'ajout au panier
- Système de commande avec suivi

10.3 Bilan personnel

Arrivé à la fin de ce TPI, je peux me permettre de dire que je suis satisfait de mon travail.

C'est la première fois que je réalise un projet de A à Z et que je le termine j'ai beaucoup de mal à rester accroché à un projet j'aime beaucoup explorer tout ce que je peux et être contraint à suivre un cahier des charges aura été un défi pour moi de pas perdre mon temps sur des points pas important et me focaliser sur les éléments demandés par le cahier des charges.

11 Glossaire

Mot technique	Signification
API	Application Programming Interface ou Interface de programmation d'applications en français, désigne un logiciel qui sert d'intermédiaire dans la communication entre deux applications et qui leur permet de partager des données. Ils permettent la connexion de différents logiciels, une connexion permise peu importe le langage de programmation et facilite l'intégration de logiciels dans une architecture existante.
API REST	Une API REST (également appelée API RESTful) est une API qui respecte les contraintes du style d'architecture REST et permet d'interagir avec les services web RESTful. L'architecture REST (Represented State Transfer) a été créée par l'informaticien Roy Fielding.
IDE	Integrated development environment ou environnement de développement en français est un ensemble d'outils qui permet d'augmenter la productivité des programmeurs (exemple : Visual Studio Code).
JSON	JavaScript Object Notation est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de rerpésenter de l'information structurée.
Framework	Un framework (appelé aussi infrastructure logicielle, infrastructure de développement, etc) c'est lui qui établit les fondations d'un logiciel ou son squelette applicatif. Il permet de simplifier et d'uniformiser le travail des développeurs.

12 Table des illustrations

13 Bibliographie

https://expressjs.com/fr/guide/routing.html

https://axios-http.com/docs/intro

http://php.net/

https://docs.microsoft.com/fr-fr/azure/architecture/best-practices/api-design

https://stackoverflow.com/questions/14750275/haversine-formula-with-php

https://leafletjs.com/reference.html

https://docs.mapbox.com/mapbox-gl-js/guides/