

# MeetUs



Thèse de Bachelor présentée par

**Marc Vachon**

pour l'obtention du titre Bachelor of Science HES-SO en

**Ingénierie des technologies de l'information avec orientation en**

**Logiciels et Systèmes complexes**

Professeur-e HES responsable

**Orestis Malaspinas**

**Septembre 2019**

# SOMMAIRE DÉTAILLÉ

|  |    |
|--|----|
| Sommaire détaillé.....                 | 2  |
| Remerciements .....                    | 5  |
| Enoncé .....                           | 6  |
| Résumé .....                           | 7  |
| Table des illustrations.....           | 8  |
| Liste des acronymes .....              | 10 |
| Introduction .....                     | 11 |
| Chapitre 1: Analyse.....               | 13 |
| 1.1. Objectif.....                     | 13 |
| 1.2. Fonctionnalités .....             | 13 |
| 1.2.1. Rendez-vous .....               | 13 |
| 1.2.2. Utilisateurs .....              | 14 |
| 1.3. Données .....                     | 15 |
| 1.3.1. Données externes.....           | 15 |
| 1.3.2. Données de l'application.....   | 15 |
| 1.4. Format JSON.....                  | 16 |
| 1.4.1. Templates .....                 | 16 |
| 1.5. Langage .....                     | 18 |
| Chapitre 2: Architecture .....         | 19 |
| 2.1. Backend.....                      | 19 |
| 2.1.1. Serveur .....                   | 20 |
| 2.1.2. Routes.....                     | 20 |
| 2.1.3. Algorithme de rendez-vous ..... | 21 |
| 2.2. Base de données .....             | 22 |

|             |                                     |    |
|-------------|-------------------------------------|----|
| 2.2.1.      | Structure .....                     | 22 |
| 2.2.2.      | Requêtes SQL.....                   | 23 |
| 2.3.        | Frontend .....                      | 24 |
| 2.3.1.      | Templates .....                     | 24 |
| 2.3.2.      | Design.....                         | 26 |
| 2.3.3.      | Dark mode .....                     | 27 |
| 2.4.        | Communications client/serveur.....  | 28 |
| 2.4.1.      | Que sont les sockets ?.....         | 29 |
| 2.4.2.      | Schéma .....                        | 29 |
| 2.5.        | Fichier de configuration .....      | 30 |
| 2.5.1.      | Données .....                       | 30 |
| 2.6.        | Fonctionnement.....                 | 32 |
| 2.6.1.      | Utilisateurs .....                  | 32 |
| 2.6.2.      | Choix de la date/heure.....         | 32 |
| 2.6.3.      | Calcul du lieu de rendez-vous ..... | 32 |
| 2.6.4.      | Récupéré les trajets.....           | 33 |
| Chapitre 3: | Technologies .....                  | 34 |
| 3.1.        | Asynchronicité.....                 | 34 |
| 3.1.1.      | Qu'est-ce que l'asynchronicité..... | 34 |
| 3.1.2.      | Promesse/ Callback .....            | 34 |
| 3.2.        | Limitations des API.....            | 36 |
| 3.2.1.      | API utilisées .....                 | 37 |
| 3.2.2.      | Normalisations .....                | 38 |
| Chapitre 4: | Résultat.....                       | 39 |
| 4.1.        | Scalabilité.....                    | 39 |
| 4.1.1.      | Problématique.....                  | 39 |
| 4.1.2.      | Idées de solutions .....            | 39 |

|  |    |
|--|----|
| 4.2. Performances .....                | 40 |
| Conclusion.....                        | 42 |
| État actuel .....                      | 42 |
| Réflexion .....                        | 42 |
| Améliorations .....                    | 43 |
| Annexes .....                          | 45 |
| Annexe 1 : Manuel d'installation ..... | 45 |
| Préparation/ configuration .....       | 45 |
| Lancement.....                         | 48 |
| Annexe 2 : Configuration .....         | 49 |
| Références documentaires .....         | 50 |

## **REMERCIEMENTS**

Je tiens à remercier mon professeur et encadrant M. Malaspinas pour son suivi et son soutien tout au long de la durée de ce projet.

Merci à l'Hepia et ses professeurs pour m'avoir formé et fourni le matériel nécessaire pendant ces trois années passées ensemble.

Je remercie aussi mes camarades de classe qui m'ont aidé, soutenu et avec qui nous avons échangé durant toutes les formations jusqu'à la concrétisation de ce projet.

Et enfin merci à ma famille de m'avoir soutenu et permis de me consacrer entièrement à mes études.

# ENONCÉ

## MEETUS : APPLICATION D'ORGANISATION DE RÉUNION

### ORIENTATION : LOGICIELS ET SYSTÈMES COMPLEXES

#### Descriptif :

Le but de ce projet est d'écrire une application web facilitant l'organisation de réunions de plusieurs personnes se trouvant à des lieux différents. L'application demandera aux utilisateurs leur point de départ et des créneaux horaires possibles, et retournera un lieu et une heure de rendez-vous convenant à tout le monde et minimisant le temps de parcours en transports publics.

#### Travail demandé :

- Étude de la technologie web pour l'implémentation de l'application.
- Développement d'un prototype de frontend de l'application.
- Développement de l'algorithme de recherche de lieu et heure optimaux.
- Développement du backend de l'application.
- Étude de scalabilité à beaucoup d'utilisateurs.

Candidat-e :

**VACHONE MARC**

Filière d'études : ITI

Professeur-e(s) responsable(s) :

**ORESTIS MALASPINAS**

**En collaboration avec :**

Travail de bachelor soumis à une convention  
de stage en entreprise : non

Travail de bachelor soumis à un contrat de  
confidentialité : non

# RÉSUMÉ

Il est souvent compliqué de convenir d'un rendez-vous avec un grand nombre de personnes habitant toutes dans des lieux différents. Ça peut être le cas pour les employés d'une entreprise lors d'un rendez-vous de projet comme pour un cadre familial, dans les deux cas il n'est pas facile de trouver une date permettant à tout le monde d'être là et encore moins de choisir le lieu idéal pour se réunir. C'est ce à quoi le projet d'application proposée par l'école permet de répondre. Son but est de choisir une date, une heure et un lieu de rendez-vous avec les données des utilisateurs et de choisir à leur place, le rendez-vous idéal. L'application va choisir une date permettant à un maximum de personne d'être présente et le lieu sera choisi afin que le temps de trajet soit équitable pour chaque utilisateur. Un trajet en transport public allant jusqu'à la ville de destination sera affiché pour chacun d'entre eux. Il comportera les étapes entre le point de départ de l'utilisateur et la ville de destination défini. L'heure et le jour d'arrivée seront aussi pris en compte. Le but ici est la réalisation d'un prototype d'application utilisable par des utilisateurs. La partie la plus importante du développement est l'algorithme choisissant le rendez-vous et son fonctionnement avec des données externes. Pour ce qui est de l'interface, le choix s'est porté sur une application web. Ce choix est le plus judicieux actuellement car cela permet à un maximum de personnes de l'utiliser. Il n'y aura besoin que d'une connexion internet pour y accéder et ne nécessite le développement que d'une seule interface.



Candidat-e :

**VACHON MARC**

Filière d'études : ITI

Professeur-e(s) responsable(s) :

**MALASPINAS ORESTIS**

Travail de bachelor soumis à une convention de stage en entreprise : non

Travail soumis à un contrat de confidentialité : <oui/non>

# TABLE DES ILLUSTRATIONS

|   |    |
|---|----|
| Figure 1: Structure JSON de rendez-vous .....                                 | 16 |
| Figure 2: Structure JSON d'un utilisateur .....                               | 17 |
| Figure 3: Structure JSON du trajet d'un utilisateur .....                     | 17 |
| Figure 4: Schéma d'architecture .....   | 19 |
| Figure 5: Routes du site.....   | 20 |
| Figure 6: Exemple de paramétrages d'une fonction .....                        | 21 |
| Figure 7: Fonction ajout d'utilisateur du fichier meetus.js .....             | 22 |
| Figure 8: Structure de la base de données .....                               | 22 |
| Figure 9: Objet JavaScript d'un utilisateur .....                             | 23 |
| Figure 10: Requête d'ajout d'un utilisateur.....                              | 23 |
| Figure 11: Template de l'accueil du site.....                                 | 25 |
| Figure 12: Template de la page de création d'un meeting.....                  | 25 |
| Figure 13: Template d'un meeting.....   | 26 |
| Figure 14: SASS vs CSS .....  | 26 |
| Figure 15: Format mobile du site .....  | 27 |
| Figure 16: Thème clair .....  | 28 |
| Figure 17: Thème foncé .....  | 28 |
| Figure 18: Socket du côté client .....  | 29 |
| Figure 19: Socket du côté serveur .....                                       | 29 |
| Figure 20: Exemple de code d'un socket.....                                   | 30 |
| Figure 21: Paramètres des fonctions du fichier de configuration.....          | 31 |
| Figure 22: Paramètres de la base de données du fichier de configuration ..... | 31 |



|  |    |
|--|----|
| Figure 23: Tableau des villes et leur différence .....   | 33 |
| Figure 24: Boucle avec des promesses.....                | 35 |
| Figure 25: Attente que les promesses est finies .....    | 36 |
| Figure 26: Accéder à phpMyAdmin.....                     | 45 |
| Figure 27: Insérer une base de données .....             | 46 |
| Figure 28: Navigation de phpMyAdmin .....                | 46 |
| Figure 29: Création d'un utilisateur sur phpMyAdmin..... | 47 |
| Figure 30: Fichier de configuration.....                 | 47 |
|  |    |
| Tableau 1: Résultats du test Geodb/fichier local.....    | 40 |
| Tableau 2: Données du fichier de configuration .....     | 49 |

# LISTE DES ACRONYMES

|             |                                   |  |
|-------------|-----------------------------------|--|
| <b>API</b>  | Application programming interface | Un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade à travers laquelle un logiciel offre des services à d'autres logiciels. |
| <b>CSS</b>  | Cascading Style Sheets            | Un langage décrivant la présentation de documents HTML.  |
| <b>HTTP</b> | Hypertext Transfer Protocol       | Un protocole de communication client/serveur développé pour le World Wide Web.   |
| <b>JSON</b> | JavaScript Object Notation        | Un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée.            |
| <b>PWA</b>  | Progressive Web App               | Une application web apparaissant et ayant un fonctionnement proche de celui d'une application native.  |
| <b>SASS</b> | Syntactically Awesome Stylesheets | Un préprocesseur CSS. C'est un langage dynamique de génération de feuille de style.  |
| <b>SQL</b>  | Structured Query Language         | Un langage structuré permettant d'exploiter des bases de données relationnelles.   |
| <b>URL</b>  | Uniform Resource Locator          | Adresse web. Désigne une chaîne de caractères identifiant une ressource du World Wide Web.   |

# INTRODUCTION

Il est souvent difficile de regrouper un grand nombre de personnes à une date et un lieu qui conviennent à tout le monde. C'est souvent lors de réunions de projet et que les personnes travaillent à différents endroits qu'il est le plus difficile de satisfaire tout le monde. De plus, il faut qu'une personne prenne le temps d'organiser le rendez-vous, fasse les démarches vers toutes les personnes devant être présentes et définisse le rendez-vous. Le projet réalisé ici propose d'éviter de perdre du temps à devoir planifier ce rendez-vous.

Ce projet a été proposé par l'école. L'idée est d'éviter à une personne d'effectuer toutes les démarches de planification d'un rendez-vous. Ce projet d'application a pour but de calculer lui-même le rendez-vous optimal entre les différents membres et de leur proposer un trajet personnalisé en transport public. Une personne n'aura qu'à créer ce rendez-vous en spécifiant simplement la période à laquelle il doit se dérouler et envoyer l'identifiant de celui-ci aux personnes concernées. Chacun pourra renseigner lui-même ses disponibilités ainsi que son lieu de départ, l'application se chargera elle-même de trouver un lieu d'arrivée, la date et l'heure permettant à un maximum de personnes d'être présentes. Quand le rendez-vous est fixé, chaque utilisateur reçoit un trajet entre son point de départ et le lieu de destination défini.

L'objectif premier est d'obtenir un algorithme efficace de calcul de rendez-vous comprenant une date, une heure et un lieu. Celui-ci dépendra des données de ses utilisateurs et de l'utilisation de données externes. C'est grâce à ces données externes que le choix de la ville de destination sera effectué et que l'on recevra les trajets jusqu'à elle. L'application sera découpée en une partie serveur contenant l'algorithme et d'une interface pour les utilisateurs. Dans un second temps, si l'algorithme correspond aux attentes, j'ajouterai des fonctionnalités à l'interface.

Pour permettre la réalisation de ce projet, une analyse des technologies web est nécessaire. Elle permettra de faire les choix des technologies à utiliser ainsi que la structure à employer.

## STRUCTURE DU DOCUMENT

---

Le document se découpe en 4 chapitres. Il commence avec l'analyse du travail à faire avec les fonctionnalités et les notions nécessaires à la réalisation de cette application. Ensuite il y a le chapitre concernant l'architecture du travail qui traite la structure de l'application et du fonctionnement de celle-ci. Le troisième chapitre parle des technologies employées et des conséquences que cela a sur le fonctionnement du site. Pour finir, le quatrième chapitre évoque les résultats, les tests effectués et donne des pistes afin d'améliorer le site. Pour terminer, nous retrouvons une conclusion qui revient sur l'état actuel de l'application, du parcours de création et du futur de celle-ci.

En fin de document se trouve les annexes ainsi que les références bibliographiques.

# Chapitre 1: ANALYSE

Avant de commencer un projet conséquent, il est important d'en connaître son but, ses fonctionnalités principales et de bien les définir, il sera toujours possible d'en ajouter une fois la base terminée. Dans le cas de cette application, il faut aussi connaître quelles données utiliser pour que notre application soit la plus optimale possible. Il est nécessaire de savoir si des données doivent être stockées dans une base de données et si oui lesquelles et comment.

## 1.1. OBJECTIF

---

L'objectif principal de cette application est de rendre, pour plusieurs utilisateurs, un rendez-vous contentant : une date, une heure et un lieu. Chacune de ces données sera calculé à partir des données utilisateurs et de données externes à l'application. L'objectif étant d'obtenir un rendez-vous le plus optimal pour une majorité des utilisateurs. Pour ce qui est de la date et l'heure il ne sera pas possible de satisfaire tout le monde mais d'en proposer une qui satisfasse le plus de personne. Quant au lieu, il dépendra principalement des données à disposition.

Le principe de l'application est le traitement de données pour afin de donner un résultat le plus satisfaisant possible. La source des données externes n'est pas le point central à cette application. Il est ici question de créer un algorithme le plus polyvalent possible et lui offrir une interface afin d'interagir avec. La source de donnée doit pouvoir changer sans que l'algorithme ne change.

## 1.2. FONCTIONNALITÉS

---

Avant de commencer, j'ai décidé de me limiter aux fonctionnalités qui me semblaient les plus importantes. En dehors de celles-ci j'en ai imaginé d'autres pouvant être ajoutées ultérieurement.

### 1.2.1. Rendez-vous

Un rendez-vous correspond à une date, une heure et un lieu. Ces trois paramètres sont le résultat de calculs et recherches en fonction des données entrées par les utilisateurs. La date et l'heure seront choisies dans l'optique de permettre au plus grand nombre de personnes d'être présentes.

Le choix du lieu devra offrir un temps de trajet le plus court possible (entre le lieu de départ et celui d'arrivée) et faire que chaque utilisateur obtienne une durée de déplacements la plus proche possible. Ainsi ça éviterait qu'un usager mette quatre heures de trajet pendant qu'un autre ne mette que deux heures pour arriver à la même destination.

N'importe qui arrivant sur la page d'accueil du projet, peut créer un rendez-vous. Il suffit d'entrer les informations demandées telles que le nom de rendez-vous, celui du créateur et la période dans laquelle le rendez-vous doit avoir lieu. Une fois le rendez-vous créé un code lui sera attribué et celui-ci pourra être donné aux personnes concernées. Avec ce code nous allons pouvoir charger le rendez-vous en question et avoir accès aux données de celui-ci.

Lorsque nous ajoutons, nous modifions ou nous supprimons un utilisateur, le rendez-vous va vraisemblablement changer. Pour chaque action sur un utilisateur nous allons vérifier que le rendez-vous, la date et le lieu soient toujours optimisés. Si un minimum d'utilisateurs ne peut être présent à la même date, un mail sera envoyé afin que chacun en soit informé et une date arrangeant un maximum de monde sera choisie.

### **1.2.2. Utilisateurs**

Un utilisateur est une personne dont la participation sera prise en compte dans le résultat de celui-ci. En effet ses nouvelles données seront ajoutées au rendez-vous qui les prendra en compte et se mettra automatiquement à jour en changeant potentiellement ses résultats.

Les fonctionnalités utilisateurs sont assez classiques. Il est possible d'en ajouter, d'en modifier et d'en supprimer. Lorsque nous ajoutons un utilisateur, nous pouvons renseigner son prénom, son nom, son email, son adresse et les dates auxquelles il est disponible. Les dates renseignées doivent être prévues dans la période définie par le rendez-vous. Chaque utilisateur ajouté appartient uniquement à un rendez-vous en particulier, il n'y a pas de compte. Il est donc nécessaire d'entrer ses informations à chaque nouveau rendez-vous.

## **1.3. DONNÉES**

---

Dans ce projet nous utilisons beaucoup de données externes pour faire des calculs. Il nous faut des données concernant des villes et les transports publics en Suisse. Toute la logique de l'application tourne autour du traitement de ces données.

### **1.3.1. Données externes**

Pour cette application il est nécessaire d'utiliser des données externes et le meilleur moyen de le faire est d'utiliser des API pour récupérer les données d'une source choisie. Une API est une interface permettant de communiquer avec une application dans le but d'accéder à ses services. Dans le cadre du web, une API permet généralement aux développeurs d'utiliser les services offerts par une application - le plus souvent externe - sans se soucier de la logique interne de cette dernière. Pour ce projet nous utilisons une API qui nous permet par exemple de récupérer les trajets en transports publics par exemple, données compliquées à recueillir autrement et de ce fait nous assure un gain de temps certain.

Afin d'interagir avec les API depuis le serveur, on va installer une librairie, « request ». La librairie request est conçue pour simplifier les appels HTTP et va nous permettre de récupérer facilement les données d'une API.

### **1.3.2. Données de l'application**

Outre l'utilisation de données externes, notre application a besoin de sauvegarder des données propres aux rendez-vous et aux utilisateurs. Nous allons donc utiliser une base de données pour structurer et sauvegarder les données.

## 1.4. FORMAT JSON

---

JSON est le format principal pour envoyer et recevoir des données quand nous passons par une API. L'avantage d'utiliser JSON avec JavaScript est que sa notation est dérivée de celle des objets JavaScript et il est donc très simple de passer de l'un à l'autre. C'est ce que je fais avec les données reçues.

### 1.4.1. Templates

#### Rendez-vous

Ceci est la forme d'un objet rendez-vous à sa création. Les champs comme « location », correspondant au lieu de destination, seront remplis une fois que des utilisateurs auront été ajoutés et qu'une destination aura été choisie. On peut donc voir que le tableau contenant les utilisateurs, « user\_table » est, lui aussi, vide pour le moment.

Le champ « period » contient les dates auxquelles commence et termine les dates possibles pour le rendez-vous.

Le champ « name » correspond au nom du rendez-vous et « creator » au nom du créateur du rendez-vous.

```
{
  "name": "Bachelor 2019",
  "creator": "Marc Vachon",
  "percentage": 0,
  "period": {
    "date_from": {
      "day": 27,
      "month": 5,
      "year": 2019
    },
    "date_to": {
      "day": 12,
      "month": 6,
      "year": 2019
    }
  },
  "date": {
  },
  "location": "",
  "user_table": []
}
```

1

Figure 1: Structure JSON de rendez-vous

---

<sup>1</sup> Source : Code de l'application



## Utilisateurs

Les utilisateurs sont contenus dans le tableau « user\_table » d'un rendez-vous. Ils ont chacun leurs propres données. « calendar » contient chaque date entrée par l'utilisateur. La date possède le jour, le mois, l'année et de quelle heure à quelle heure on est disponible.

Le tableau « journey » est vide lors de la création d'un utilisateur mais sera rempli une fois le rendez-vous mis à jour et le trajet calculé. Il contient les données du trajet entre notre adresse et le lieu de rendez-vous. Il y a le temps de trajets et chaque étapes de celui-ci.

```
{
  "name": "Marc Vachon",
  "email": "vachon.marc@outlook.com",
  "address": "Genève",
  "coordinate": {
    "lat": 46.2017559,
    "lon": 6.1466014
  },
  "calendar": [
    {
      "day": 27,
      "month": 5,
      "year": 2019,
      "from": {
        "hour": 12,
        "minutes": 30
      },
      "to": {
        "hour": 22,
        "minutes": 15
      }
    }
  ],
  "journey": {
    "time": {
      "hour": 2,
      "minutes": 36
    },
    "steps": []
  }
},
```

2

Figure 2: Structure JSON d'un utilisateur

## Étapes du trajet

Le trajet comporte des étapes contenues dans le tableau « steps ». Chacune de ces étapes contient le moyen de transport attribué et les données du lieu de départ et d'arrivée. Dans les deux cas, on retrouve le lieu, l'heure et la plateforme.

```
"steps": [
  {
    "to": {
      "hour": "11",
      "minutes": "25",
      "station": "Zürich HB",
      "platform": "5"
    },
    "from": {
      "hour": "10",
      "minutes": "35",
      "station": "Luzern",
      "platform": "5"
    },
    "transport": "IR 2638"
  }
]
```

3

Figure 3: Structure JSON du trajet d'un utilisateur

---

<sup>2</sup> Source : Code de l'application

<sup>3</sup> Source : Code de l'application

## 1.5. LANGAGE

---

L'application se divisant en 2 parties, un serveur avec l'algorithme de calcul de rendez-vous et une interface utilisateur j'ai choisi de faire une application web. Pour ce qui est du serveur, l'utilisation de Node JS est apparue comme une évidence car c'est une plateforme logicielle récente qui est très performante. Node JS permet aussi d'utiliser JavaScript du côté serveur et l'utilisation de plusieurs bibliothèques externes. Un autre avantage est que ce soit le frontend ou le backend les deux auront le même langage de programmation ce qui est un gain de temps d'adaptation, certain. Il y a aussi le fait que la structure des objets JavaScript est similaire à celle de JSON.

## Chapitre 2: ARCHITECTURE

Après avoir analysé ce que nous allons implémenter en premier et ensuite défini nos besoins, il va falloir s'intéresser à la conception de l'application. Nous allons regarder la structure utilisée pour chaque partie nécessaire au bon fonctionnement de l'application ainsi que la manière de le faire.

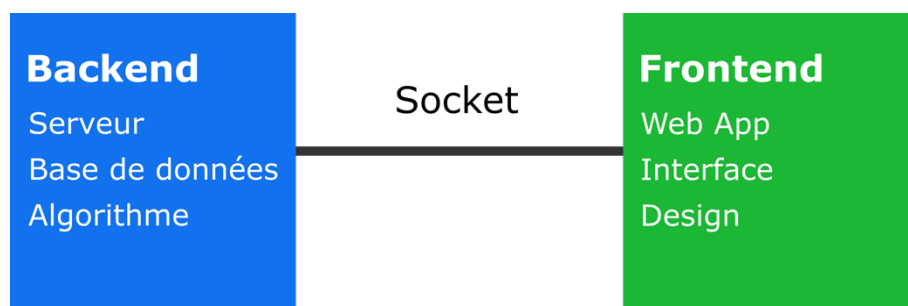


Figure 4: Schéma d'architecture

### 2.1. BACKEND

---

Le backend est la partie principale de l'application. Le backend est la partie qui contient la logique de l'application. C'est ici que nous créons réellement l'application avec son fonctionnement et ses fonctionnalités.

Dans notre cas la structure du backend comporte plusieurs parties. Il y a le serveur avec ses routes, la connexion avec la base de données ainsi que la communication avec celle-ci, les différents fichiers des fonctionnalités de l'application et la gestion des demandes des utilisateurs.

L'architecture backend a été conçue afin de pouvoir évoluer facilement sans avoir à modifier tous son fonctionnement. Les fonctions concernant les données externes à l'application (récupérée depuis une API par exemple) sont paramétrables dans le fichier de configuration, ce qui nous permet d'ajouter de nouvelles sources de données sans que le code général ne soit modifié.

---

<sup>4</sup> Source : Schéma créé pour le document

### 2.1.1. Serveur

Le serveur est la partie principale du backend. C'est lui que nous exécutons pour lancer le site web. C'est aussi dans celui-ci que nous ajoutons les middlewares principaux pour le bon fonctionnement de notre application et que nous lui spécifions certains paramètres comme l'emplacement des sources du projet, le fichier contenant les routes et celui contenant les sockets.

### 2.1.2. Routes

Les routes sont les chemins d'accès aux différentes fonctionnalités du site. Dans notre cas les routes servent à atteindre une page du site en question. Elles sont contenues dans leur propre fichier et importées dans celui du serveur. Elles sont créées grâce à Express JS un middleware de routing pour Node JS. Express nous permet d'organiser avec plus de clarté les routes et ajoute des fonctionnalités à celles-ci. Il nous permet aussi de rendre une template pour une route en question ou d'utiliser des paramètres dans les liens comme un id par exemple.

Ici nous retrouvons les trois routes correspondant aux trois pages du site. La première qui est l'accueil nous permet soit de charger un rendez-vous existant soit de nous rediriger vers la page de création d'un rendez-vous. La seconde est la page de création d'un rendez-vous, et enfin la dernière est la page du rendez-vous lui-même qui contient l'id du rendez-vous à charger.

```
app.get('/', function (req, res) {
  res.render("index.ejs");
});

app.get('/meeting-:id', function (req, res) {

  db_request.check_id(req.params.id, function(resp){
    if(resp){
      res.render("meeting.ejs", {id_meeting: req.params.id});
    }else{
      res.render("index.ejs");
    }
  })
});

app.get('/create-meeting', function (req, res) {
  res.render("create-meeting.ejs");
});
```

5

*Figure 5: Routes du site*

---

<sup>5</sup> Source : Code de l'application

### 2.1.3. Algorithme de rendez-vous

L'algorithme de rendez-vous c'est le code qui permet de calculer un rendez-vous par rapport aux données contenues dans la base de données et des données externes. C'est le code propre à notre application. Le code a été séparé en différentes classes, qui correspondent chacune aux données traitées, la classe « user » contient les fonctions pour gérer un user par exemple. Les classes sont toutes reliées les unes aux autres. Dans chaque classe on retrouve en premier les fonctions publiques qui sont exportées pour être utilisables à l'extérieur du fichier et ensuite les fonctions privées propres à la classe.

L'intérêt d'avoir séparé le code de cette manière est que nous pouvons facilement intervenir en ajoutant, en modifiant ou en supprimant des parties du code ou des fonctions. Le plus important dans la construction de ces fonctions est leurs arguments et ce qu'elles nous retournent. Si par exemple on décidait de changer la source des données qu'on utilise pour récupérer la liste des villes, il faudrait que le contenu retourné corresponde à ce qui est demandé. Un entier, un tableau, etc.

```
switch (method) {  
  case "geodb":  
    return geodb(point, limit, population, range)  
    break;  
  
  case "local":  
    return search.load_cities(point.lat, point.lon, range, population, limit)  
    break;  
}
```

6

*Figure 6: Exemple de paramétrages d'une fonction*

Pour faciliter la lecture et le fonctionnement du code, une classe principale, appelée Meetus, a été ajoutée. Elle résume les fonctionnalités principales. La fonction d'ajout d'utilisateur va donc elle-même créer l'utilisateur, l'ajouter dans la base de données et mettre à jour le rendez-vous avec les nouvelles informations. Ce qui est intéressant avec cette démarche c'est que lorsque nous voulons ajouter un utilisateur nous pouvons utiliser cette fonction sans nous soucier de ce que cela implique.

---

<sup>6</sup> Source : Code de l'application

```

add_user: (idMeeting, data, callback) => {
  user.add_user(idMeeting, data, function(result){

    appointment.choose_appointment(idMeeting, function(r){

      journey.update_center(idMeeting, function(res){

        journey.update_journeys(idMeeting, function(c){

          callback(res)
        })
      })
    })
  })
},

```




Figure 7: Fonction ajout d'utilisateur du fichier meetus.js

## 2.2. BASE DE DONNÉES

Dans le cadre de cette application, la base de données va nous servir pour stocker les données relatives à un rendez-vous. Le système de gestion de la base de données utilisé ici est MySQL.

### 2.2.1. Structure

La structure de notre base de données se veut la plus simple possible, elle ne comporte qu'une table où seul un id, la date de création et le fichier JSON du rendez-vous sont stockés. Grâce à cette structure les requêtes sont très simples et nous pourrions très facilement envisager de faire évoluer sa structure ou bien le gestionnaire de base de données en ne changeant que les requêtes.

|   | meetus t_meeting    |
|---|---------------------|
|  | idMeeting : int(11) |
|  | date_added : date   |
|  | data : json         |

8

Figure 8: Structure de la base de données

<sup>7</sup> Source : Code de l'application

<sup>8</sup> Source : Interface de Mamp

## 2.2.2. Requêtes SQL

Avec MySQL il nous est possible de faire des requêtes dans un champ JSON afin de modifier directement le contenu de celui-ci. Dans notre cas cela nous permet d'ajouter très facilement des objets JavaScript, convertis en JSON, à l'endroit souhaité dans le fichier. Cela nous permet de traiter les données et créer l'objet souhaité sur le serveur avant de l'envoyer dans la base de données.

### Objet JavaScript d'un utilisateur

```
var user = {  
  name: $('#name').val(),  
  email: $('#email').val(),  
  address: $('#city').val(),  
  calendar: create_calendar(),  
  journey: {}  
}
```

Figure 9: Objet JavaScript d'un utilisateur

La commande "JSON\_ARRAY\_APPEND" dans la requête permet l'ajout des données dans le tableau défini. Ici ce sera les données concernant un utilisateur. "CAST AS JSON" nous permet de dire à MySQL d'ajouter les données en tant que contenu JSON (le « ? » sera remplacé par les données du second paramètre de la commande query), si nous ne lui précisons pas il va les ajouter comme si c'était un string et ne respectera plus la structure JSON.

### Requêtes pour l'ajout d'un utilisateur

```
add_user: (idMeeting, data, callback) => {  
  let query = "UPDATE t_meeting SET data = JSON_ARRAY_APPEND(data,  
    '$.user_table', CAST(? AS JSON)) WHERE idMeeting = ?"  
  
  db.query(query, [data, idMeeting], (error, results, fields) => {  
    if (error) throw error;  
    callback(results)  
  })  
},
```

10

Figure 10: Requête d'ajout d'un utilisateur

---

<sup>9</sup> Source : Code de l'application

<sup>10</sup> Source : Code de l'application

## 2.3. FRONTEND

---

Le frontend est la partie avec laquelle un utilisateur va interagir. Cela comprend les boutons, l'interface, l'envoi des données renseignées au serveur et le design qui va mettre le tout en forme. Ici l'interface choisie est celle d'un site internet ce qui permet un développement unique et est accessible depuis n'importe quel appareil connecté.

Se rapprochant plus d'une application web que d'un site, la structure est un peu différente. On appelle cette structure une « PWA », Progressive Web App, qui consiste en des pages ou des sites web apparaissant à l'utilisateur de la même manière que les applications natives. Ce type d'applications tente de combiner les fonctionnalités offertes par la plupart des navigateurs modernes avec les avantages de l'expérience offerte par les appareils mobiles. Elle se consulte à la manière d'un site internet, depuis une URL, mais sans les contraintes d'une application native, utilisation de la mémoire, téléchargement depuis un store (apple store, etc.), etc.

### 2.3.1. Templates

Dans notre cas une template est une page web avec sa structure et ses dépendances. Dans le site on retrouve trois pages distinctes. L'accueil, le rendez-vous et la page de création de rendez-vous. Chacune de ces pages est construite en plusieurs parties, comme la barre de navigation, qu'on retrouve dans chacune d'elles ou bien les fonctions JavaScript propre à chacune d'elles.

Sur la page d'accueil on a accès au chargement d'un rendez-vous et à sa création. Pour charger un rendez-vous il faut entrer le code lui correspondant. Choisir de créer un rendez-vous va nous rediriger sur la page correspondante.



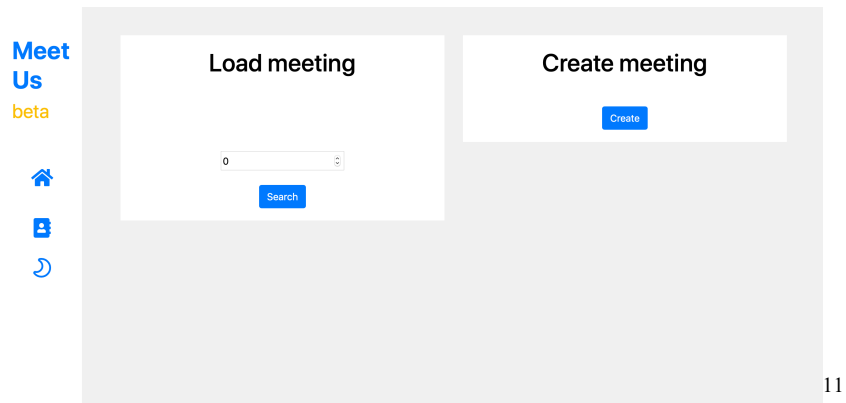


Figure 11: Template de l'accueil du site

Pour créer un rendez-vous il suffit de remplir les champs du formulaire. Insérer un nom pour le rendez-vous, notre nom en tant que créateur de celui-ci et la période pendant laquelle il doit se dérouler. La période consiste en deux dates, les utilisateurs ne pourront entrer leur disponibilité que pendant cette période.

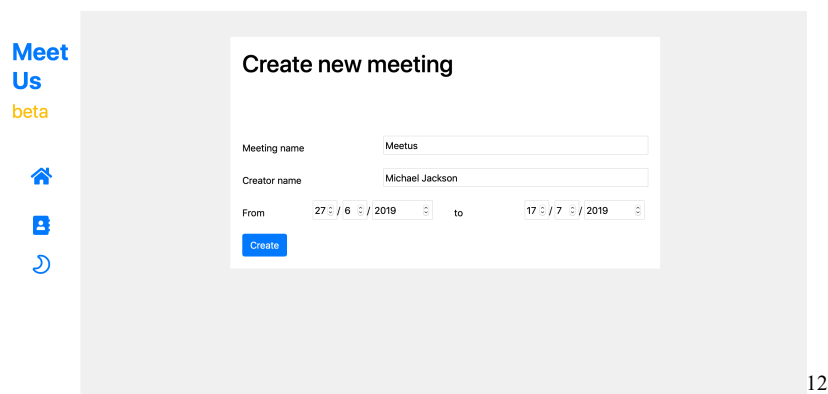


Figure 12: Template de la page de création d'un meeting

Pour la page de meeting le template est construit un peu différemment des autres pages. Le template est dans un premier temps chargé comme précédemment mais sans les données du rendez-vous, celles-ci seront chargées dans un second temps. C'est lorsque la page sera prête que nous allons demander, à l'aide des sockets, les données à afficher. Nous allons lui envoyer l'id du rendez-vous correspondant et en retour recevoir les données contenues dans la base de données. C'est dans un fichier JavaScript qu'on va créer un template html et l'ajouter à notre page.

<sup>11</sup> Source : Interface de l'application

<sup>12</sup> Source : Interface de l'application

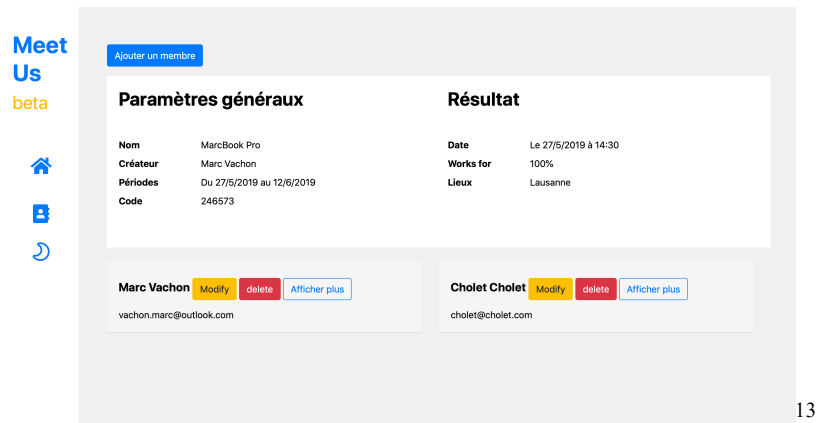


Figure 13: Template d'un meeting

## 2.3.2. Design

Mettant en forme le contenu du frontend, le design définit la manière avec laquelle les utilisateurs vont interagir. De ce fait il est basé sur le modèle du flat design afin d'être le plus simple et pratique possible. Ayant pour but de me rapprocher plus d'une application que d'un site, j'ai utilisé le framework Bootstrap afin que le site soit « responsive » et donc utilisable aussi bien sur un ordinateur, une tablette que sur mobile.

La feuille de style a été réalisée avec le préprocesseur de feuille de style « SASS ». Cela m'a permis de faire des feuilles de style avec une structure plus pratique que du CSS. Cette structure nous permet d'imbriquer les paramètres de style les uns dans les autres. Cette hiérarchie est plus agréable à lire. Un fichier SASS utilise l'extension « .scss ».



Figure 14: SASS vs CSS

<sup>13</sup> Source : Interface de l'application

<sup>14</sup> Source : <https://zurb.com/word/sass>

Le fichier SASS sera ensuite compilé en fichier CSS lequel sera ensuite utilisé pour notre site. JavaScript et JQuery ont aussi été utilisés pour certaines fonctions comme l'ajout d'un utilisateur ou celui des dates pour un utilisateur.

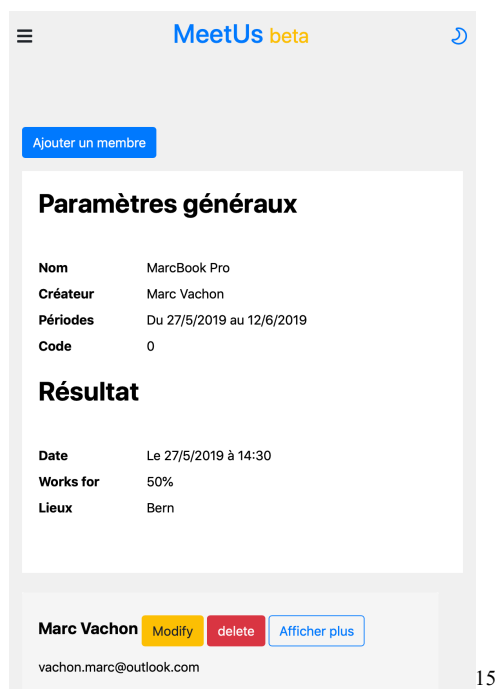


Figure 15: Format mobile du site

### 2.3.3. Dark mode

Le dark mode est apparu récemment dans nos systèmes. Ce mode est le fait de pouvoir passer notre application d'un mode avec des couleurs claires à un mode avec des couleurs sombres, en général pour l'utilisation de nuit. Son intérêt, autre qu'esthétique est surtout faite pour ne pas agresser nos yeux.

Ces derniers mois, plusieurs grandes entreprises ont commencé à intégrer un mode sombre dans leur système. Ce n'est pas une fonction vitale pour ce programme, mais c'est un petit plus au goût du jour qui sera en parfait accord avec les futures mises à jour des navigateurs web, qui ont pour la plupart déjà intégré cette fonctionnalité.

---

<sup>15</sup> Source : Interface mobile de l'application

Pour activer ce mode sombre, il suffit d'appuyer sur la lune dans la barre de navigation. L'application va immédiatement changer de thème et sauvegarder votre choix dans un cookie pour ne pas devoir changer dès que nous arrivons sur le site.

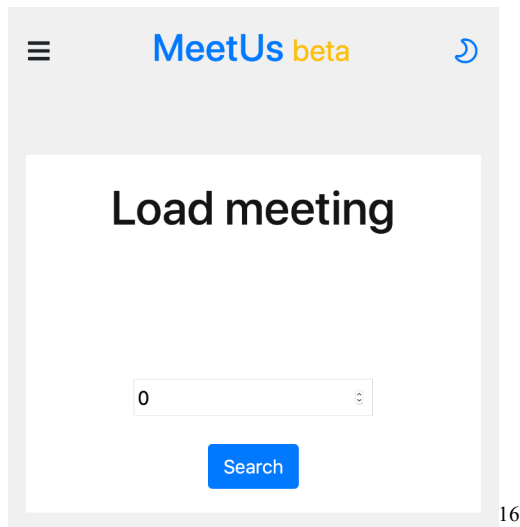


Figure 16: Thème clair

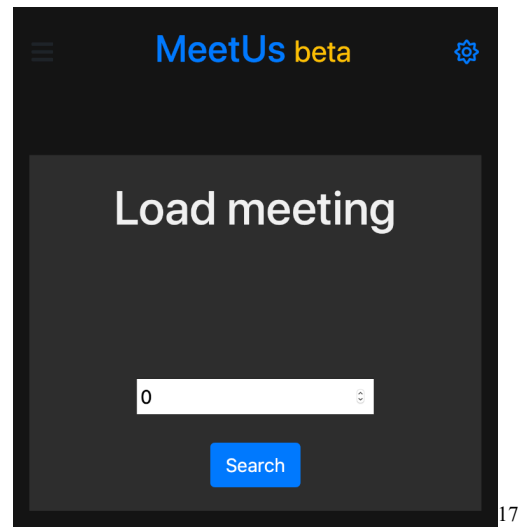


Figure 17: Thème foncé

## 2.4. COMMUNICATIONS CLIENT/SERVEUR

---

La communication client/serveur c'est le moyen de faire communiquer une interaction d'un utilisateur avec l'interface vers le serveur. Dans notre cas, lorsqu'on ajoute un utilisateur on va indiquer au serveur pour qu'il puisse exécuter cette demande.

Afin de faire communiquer le plus simplement les utilisateurs avec le serveur, les sockets, ont paru être une évidence. C'est un moyen simple de pouvoir exécuter des commandes depuis le client vers le serveur et envoyer une réponse en retour depuis le serveur. C'est aussi un bon moyen d'interagir entre les deux sans avoir à recharger la page web, ce qui est toujours plus confortable surtout si la connexion internet n'est pas très puissante.

---

<sup>16</sup> Source : interface de l'application

<sup>17</sup> Source : Interface de l'application

### 2.4.1. Que sont les sockets ?

Les sockets sont une interface de connexion bidirectionnelle entre client et serveur. Cela comprend deux parties : une bibliothèque du côté client qui s'exécute dans le navigateur et une du côté serveur pour Node JS. Son fonctionnement est très simple, le serveur va attendre la connexion d'un client sur le même réseau. Une fois connectées les deux parties, client et serveur attendent de recevoir une communication de l'un ou l'autre. Après en avoir reçu une, ils exécuteront le code correspondant.

### 2.4.2. Schéma

Pour utiliser les sockets avec Node JS on doit installer le middleware « socket.io ». C'est une bibliothèque JavaScript pour les applications Web en temps réel.

Afin d'avoir un fichier de socket le plus simple possible du côté serveur, nos sockets ne font qu'exécuter une commande de la classe meetus (créer un rendez-vous, un utilisateur, etc.) et renvoyer un socket au client avec les nouvelles données à afficher, cela me permet d'avoir une structure très simple.

#### Client

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('http://localhost:8080')
</script>
```

18

Figure 18: Socket du côté client

#### Serveur

```
var io = socket(server)

io.on('connection', function(socket){
  console.log('SOCKET: A connection has been made')
  console.log('SOCKET: The socket is', socket.id)
```

19

Figure 19: Socket du côté serveur

---

<sup>18</sup> Source : Code de l'application

<sup>19</sup> Source : Code de l'application

## Schéma d'un socket

```
socket.on('add_user', function(idMeeting, data){
  meetus.add_user(idMeeting, data, function(){
    get_meeting(idMeeting)
  })
})
```

20

Figure 20: Exemple de code d'un socket

## 2.5. FICHER DE CONFIGURATION

---

L'application a pour but de traiter avec des données externes et celles de ces utilisateurs pour leur rendre un résultat (données du rendez-vous) le plus optimal possible. Il y a plusieurs moyens de fournir ces données externes mais dans notre cas j'utilise des API. Aux vues des possibilités en matière d'API et donc de données disponibles pour les requêtes, j'ai décidé de faire un fichier de configuration nous permettant de choisir les données à utiliser. Cela permet aussi d'avoir une application que l'on peut faire évoluer en ajoutant d'autres ressources sans avoir à modifier tout le code.

### 2.5.1. Données

Étant donné que l'optimisation des résultats de l'application dépend des données utilisées, il est possible de paramétrer la recherche des données avec plusieurs paramètres.

Pour choisir une ville de destination j'ai donc testé deux manières différentes, la première avec l'API GeoGb et la seconde avec un fichier JSON contenant une liste de ville suisses et leurs informations. Nous avons aussi le choix pour le rayon dans lequel chercher les villes autour du point géographique, de la population de la ville et du nombre maximum de résultats à vérifier.

---

<sup>20</sup> Source : Code de l'application

```

"city_search":{
  "method": "geodb",
  "population": 10000,
  "range": 30,
  "limit": 10
},
"transport_search":{
  "method": "transport_api"
}

```

21

Figure 21: Paramètres des fonctions du fichier de configuration

Pour la base de données, il est possible de définir les paramètres de connexion à celle-ci. Les paramètres disponibles sont :

- L'host
- Le port
- Le nom de la base de données
- L'utilisateur pour se connecter à elle
- Le mot de passe de l'utilisateur

```

"database":{
  "host": "localhost",
  "port": 8889,
  "db_name": "meetus",
  "user": "meet",
  "password": "Super"
},

```

22

Figure 22: Paramètres de la base de données du fichier de configuration

---

<sup>21</sup> Source : Code de l'application

<sup>22</sup> Source : Code de l'application

## **2.6. FONCTIONNEMENT**

---

La complexité de l'application se trouve au moment où nous traitons avec les utilisateurs. À chaque fois qu'un utilisateur est ajouté, modifié ou supprimé, une mise à jour du rendez-vous sera effectuée.

### **2.6.1. Utilisateurs**

Pour l'ajout, la modification et la suppression, seules les requêtes vers la base de données sont différentes, seul l'ajout demande une fonction en plus. Étant donné qu'un utilisateur renseigne le nom de la ville et que le centre géographique est calculé avec les coordonnées de chaque utilisateur, il faut les ajouter, en les récupérant grâce à une requête à l'API Nominatim, dans les données de l'utilisateur.

Dans n'importe quel cas, une fois que la base de données sera mise à jour, le rendez-vous le sera lui aussi. Chaque modification apportée aux données du rendez-vous est susceptible de modifier la date, l'heure ou bien le lieu de rendez-vous optimal, il est donc important d'effectuer cette mise à jour à chaque fois.

### **2.6.2. Choix de la date/heure**

La date de rendez-vous est choisie à partir des données des utilisateurs. Nous allons la choisir en prenant la date entrée par le plus grand nombre de personnes. Si nous n'avons pas au minimum 80% de personnes disponibles à cette date, nous enverrons un mail à chaque membre inscrit au rendez-vous prévu, en signalant qu'il y aura un grand nombre d'absents pour ce jour-ci.

L'heure de rendez-vous correspondra à l'heure la plus tardives parmi les utilisateurs ayant entré le jour choisi.

### **2.6.3. Calcul du lieu de rendez-vous**

Le choix du lieu de destination va être fait par rapport aux données des utilisateurs et de données externes. Dans un premier temps, on va utiliser les coordonnées des emplacements de chacun, pour calculer le centre géographique entre tous. Une fois ce point calculé, les villes autour de ce point et correspondant aux paramètres prédéfinis, vont être récupérées. Les paramètres sont :



le rayon dans lequel il faut se trouver, le nombre maximum de ville récupéré, la population minimum à avoir et la source dans laquelle chercher ces données. Dans notre cas elles le sont depuis l'API de Geodb.

Une fois cette liste de villes obtenu on va calculer le temps de trajet entre l'emplacement de chaque utilisateur et chaque ville de la liste. On va remplir un tableau de résultat contenant pour chaque ville, son nom et la différence de temps mis par le plus rapide et le plus lent. Le temps est indiqué en minutes.

```
{ city: 'Bern', diff: 48 }  
{ city: 'Köniz', diff: 119 }  
{ city: 'Biel/Bienne', diff: 62 }  
{ city: 'Thun', diff: 110 }  
{ city: 'Fribourg', diff: 61 }  
{ city: 'Emmen', diff: 142 }  
{ city: 'Kriens', diff: 130 }
```

Figure 23: Tableau des villes et leur différence

Une fois le tableau rempli on va parcourir les villes et chercher celle ayant la différence la plus faible. Elle sera choisie comme ville de destination.

#### 2.6.4. Récupéré les trajets

Nous allons maintenant obtenir les trajets pour chaque utilisateur et mettre à jour toutes les données dans la base de données. Le trajet est obtenu par rapport à la ville de destination ainsi que la date et l'heure de rendez-vous. Il ne sera calculé qu'une fois les données du rendez-vous fixé.

Les données récupérées seront adaptées à la structure de notre fichier JSON avant d'être ajouté dans la base de données.

---

<sup>23</sup> Source : Résultat dans le terminal

## Chapitre 3: TECHNOLOGIES

Ce chapitre évoque le développement technique du site et revient sur des points importants de celui-ci et notamment certaines complications.

### 3.1. ASYNCHRONICITÉ

---

Utilisant Node Js et donc JavaScript du côté serveur nous allons rencontrer certaines subtilités dues à son utilisation et dans notre cas principalement l'asynchronicité. Ce n'est pas un problème en soi, mais il va falloir la gérer pour faire ce dont on a besoin.

#### 3.1.1. Qu'est-ce que l'asynchronicité

L'asynchronicité permet au moteur JavaScript de gérer d'autres tâches tout en gardant une interface active. C'est-à-dire que le programme va continuer d'être exécuté sans attendre qu'un résultat soit retourné par une fonction asynchrone. Par exemple pour une fonction exécutant une requête http. Ces fonctions peuvent dans de nombreux cas être très pratiques, car le programme continue de fonctionner même si d'autres n'ont pas terminé.

Dans notre cas lorsque nous faisons des requêtes HTTP ou à la base de données JavaScript ne va pas attendre d'avoir reçu une réponse avant de continuer d'exécuter le code, ce qui évidemment pose un problème lorsque la suite du programme nécessite d'avoir reçu les données précédentes. Heureusement JavaScript propose une solution à ce problème, les callbacks dans un premier temps et plus récemment les promesses.

#### 3.1.2. Promesse/ Callback

Les promesses et les callbacks vont résoudre le problème d'asynchronicité de JavaScript. Dans notre cas, il est nécessaire d'y avoir recours pour l'ajout d'un utilisateur par exemple. Lorsqu'on ajoute un utilisateur on va mettre à jour les données du rendez-vous il est donc nécessaire de prendre en compte ce nouvel utilisateur. Il va donc falloir attendre que celui-ci soit ajouté dans la base de données. En utilisant un callback, par exemple, on va spécifier qu'on veut être sûr que l'utilisateur ait été ajouté dans la base avant de continuer. Si on ne lui précise pas, l'exécution du programme continuera après l'envoi de la requête sans attendre que celle-ci soit terminée.

Un autre problème est celui des boucles. Dans une boucle JavaScript, un callback ne peut être utilisé, car l'incrémentación n'attendra pas qu'il ait été rendu pour continuer, il va donc falloir bloquer le processus après la boucle et attendre qu'elle ait fini de nous rendre ce que l'on attend.

La solution pour faire cela est d'utiliser les promesses et plus précisément un tableau de promesses. À chaque incrémentación nous allons lancer notre requête qui va nous rendre une promesse que nous allons mettre dans un tableau.

### Boucle for avec tableaux de promesses

```
var promises = []
for (var i = 0; i < result.length; i++) {
  city = result[i]

  var validation = validate_city(idMeeting, result[i], date)

  promises.push(validation)

  validation.then(function(data){
    table_cities.push(data)
  })
}
```

24

*Figure 24: Boucle avec des promesses*

La boucle va continuer de tourner sans attendre la fin de chaque promesse, une fois qu'elle sera arrivée au bout on va lui dire qu'il faut attendre que chaque promesse du tableau soit terminée pour continuer. Ce fonctionnement ne va pas bloquer la boucle, mais lui faire effectuer les promesses en parallèle. Cette façon de faire est très intéressante, car elle nous permet d'avoir d'une part un code fonctionnant en parallèle et d'autre part d'attendre que notre contenu ait bien été récupéré.

---

<sup>24</sup> Source : Code de l'application

```

Promise.all(promises).then(function(){

    // take the shortest path
    if(table_cities.length != 0){

        city = table_cities[0]

        for (var i = 0; i < table_cities.length; i++) {

            if(table_cities[i].diff < city.diff){
                city = table_cities[i]
            }
        }
    } else {
        console.log("There's no city available");
    }
    callback(city.city)
})

```

25

Figure 25: Attente que les promesses est finies

## 3.2. LIMITATIONS DES API

---

L'application est tributaire des ressources proposées par les API existantes et du choix de celles-ci. Chacune d'entre elles fournit des services différents avec des données différentes parfois gratuites parfois payantes. Nous avons choisi de faire une application avec des ressources gratuites et ce choix nous limite dans les données exploitables. Mais le fait d'avoir choisi des API gratuites ne nous limite pas dans les fonctionnalités de l'application, ce choix impacte simplement les données utilisées comme le choix des villes de rendez-vous. La liste de villes rendue par une autre API pourrait être différente.

Chacune des API que nous utilisons possède des restrictions d'utilisation parfois dues à notre utilisation de leur formule gratuite ou simplement à cause de leur fonctionnement naturel. Les restrictions monétaires ne sont pas un réel problème, car nous pouvons y remédier. Dans le cadre de ce travail, nous avons fait le choix de créer une application gratuite, mais en cas de mise en service de l'application, nous pourrions très facilement établir un budget plus ou moins, important. Il permettrait d'obtenir de meilleurs résultats en utilisant des jeux de données plus

---

<sup>25</sup> Source : Code de l'application

conséquents. En revanche une limitation propre au fonctionnement de l'API peut être un frein à l'évolution de l'application surtout si c'est une des seules sur le marché ou qu'elle est déjà la plus complète. Une des solutions serait de changer d'API, mais ce n'est pas toujours possible.

### **3.2.1. API utilisées**

Voici la liste des API utilisées actuellement :

#### **Geodb cities API :**

Elle est utilisée afin de recevoir une liste de villes les plus proches d'un point géographique. On lui renseigne les coordonnées géographiques de notre point, le rayon de recherche, la taille minimale de la ville et elle nous donne les villes correspondantes.

Geodb nous propose plusieurs formules pour utiliser ses services, une formule gratuite, que nous utilisons ici, mais aussi d'autres qui sont payantes. Les résultats obtenus avec la formule gratuite sont satisfaisants dans une grande majorité des cas. Ils le sont lorsqu'il s'agit de cas plus complexes ce qui parfois peut représenter une contrainte réelle pour optimiser au maximum notre algorithme.

Le fait qu'une ville en dessous de 19'999 habitants ne sera pas retournée ou qu'il y ait un maximum de ville retournée peut poser un problème dans certains cas. Imaginons que des personnes se trouvent entre Genève et Lausanne, on ne recevra jamais la solution Nyon, car la ville est en dessous des 19'999 habitants. On aura le choix entre Genève et Lausanne.

Il y a aussi les limitations de requêtes par seconde et par jour qui vont poser problème si plusieurs personnes utilisent l'application en même temps.

#### **Transport API :**

C'est une API non officielle couvrant le réseau des transports publics en suisse. Nous l'utilisons pour calculer les trajets de chaque utilisateur vers la ville de destination à l'heure choisie. Elle nous retourne par exemple, les horaires, les moyens de transport et le nom de l'arrêt.

Utilisant elle-même le service web de search.ch les limitations sont donc celles de search.ch. Dans notre cas actuel la limitation de requêtes par jour s'élève à 1'000, ce qui est très peu. Néanmoins, il est possible d'entrer en contact avec eux afin de trouver une solution.

### **Nominatim :**

Nominatim est un moteur de recherche pour OpenStreetMap. Il nous permet de recevoir les coordonnées géographiques pour une ville. Les coordonnées vont nous servir lors du calcul du centre géographique pour trouver une ville de destination. Nominatim permet de réaliser une requête par seconde.

### **3.2.2. Normalisations**

Étant donné l'utilisation de données provenant de plusieurs sources différentes, les données utilisées par chaque API peuvent signifier la même chose, mais avoir une appellation légèrement différente : dans une autre langue, avec des accents, etc. Le problème, c'est que quelqu'un habitant dans une ville dont le nom comporte des accents par exemple, il se peut qu'une des API ne trouve pas de correspondance.

C'est une complication qui est difficilement solvable. Par exemple, si lors des calculs du trajet, la ville de destination choisie n'a pas la même signification pour les différentes API, il sera difficile de vérifier que le trajet ne correspond pas, à moins qu'on nous rende une erreur.

## Chapitre 4: RÉSULTAT

Dans ce dernier chapitre, nous allons voir les résultats obtenus lors de l'utilisation du site. Quelle est la complexité de ceux-ci et pourquoi on obtient ces résultats.

### 4.1. SCALABILITÉ

---

La scalabilité est la capacité d'un produit à s'adapter à une forte montée en charge (nombre de demande) et de maintenir ses fonctionnalités opérationnelles.

#### 4.1.1. Problématique

La scalabilité n'a pas été testée pour le moment, mais étudiée. En regardant les limitations actuelles des API utilisées, on se rend vite compte que lors d'une utilisation intensive du site, il serait impossible de répondre à la demande. En l'état nous ne pourrions ajouter qu'un utilisateur par seconde à cause des réglementations de Nominatim. Aussi, nous ne pourrions afficher que très peu de trajets par jour sachant que nous n'avons le droit qu'à 1'000 requêtes par jour pour les transports et que notre application a besoin de recalculer les trajets à chaque modification de la destination, pour chaque utilisateur. Imaginons que la recherche de villes par rendez-vous soit d'une par seconde, nous ne pourrions avoir que cinq rendez-vous fonctionnant à la seconde.

Imaginons que les demandes techniques citées avant soient contentées, nous aurions toujours des problèmes d'optimisation pour le choix des villes. Le fait d'être bloqué aux villes au-dessus de 20'000 est un frein certain. S'ajoute à cela le fait qu'un maximum de 10 villes soit reçu il est difficile de répondre aux cas les plus singuliers.

#### 4.1.2. Idées de solutions

En ce qui concerne les transports publics, peu de solutions s'offrent à nous, nous pourrions utiliser une autre API ou bien contacter transport API pour connaître les solutions disponibles.

Pour ce qui est de la recherche des villes, on pourrait abandonner l'idée de l'open source et utiliser les services d'une autre API comme Google, leurs données étant plus fournies et leurs services plus importants il nous serait plus facile d'atteindre nos standards. Une autre solution serait de télécharger une liste des villes suisses avec toutes leurs données. Actuellement le fichier de config nous permet de choisir un mode « local » qui utilise déjà un fichier JSON avec

une liste de villes. Malheureusement ce fichier ne contient pas toutes les informations nécessaires, cependant les fonctions pour choisir les villes existent déjà à l'intérieur de ce fichier. On pourrait donc imaginer changer ou améliorer le contenu de ce fichier pour que les résultats soient déjà plus précis.

## 4.2. PERFORMANCES

---

Sans prendre en compte les limitations des API, j'ai testé le site avec deux jeux de données. Un provenant de l'API Geodb et l'autre d'un fichier JSON contenant une liste de villes téléchargées sur le site « [simplemaps.com](https://simplemaps.com) <sup>26</sup> ». Je me suis rendu compte que, que ce soit dans le premier cas ou dans le deuxième, les temps étaient équivalents et que récupérer les villes d'un fichier en local (dans le format de celui-ci) ou depuis l'API de Geodb, ne faisait pas une grande différence. La seule différence que l'on observe est que le temps a plus souvent tendance à changer quand on utilise l'API de Geodb à cause de la connexion internet qui peut varier.

|   | Geodb | Fichier local |
|---|-------|---------------|
| <b>5 villes vérifiées</b><br><b>10 utilisateurs</b>                 | ~3s   | ~3s           |
| <b>10 villes vérifiées (max de geodb)</b><br><b>20 utilisateurs</b> | ~3s   | ~3s           |

Tableau 1: Résultats du test Geodb/fichier local

Le facteur ayant le plus grand impact est donc le temps de calcul des temps de trajets des utilisateurs. Pour améliorer les performances, il faudrait davantage se concentrer sur cette partie.

Pour ce qui est de l'optimisation du choix de la ville, tester la liste locale ou bien celle obtenu depuis l'API de Geodb n'aurait pas beaucoup de sens sachant que le fichier JSON actuel n'est pas suffisamment fourni. Les résultats sont donc plutôt similaires avec un léger avantage à Geodb évidemment. Pour améliorer cette partie, il faudra donc obtenir une liste de villes plus

---

<sup>26</sup> Source : <https://simplemaps.com/data/ch-cities>



fournie afin de voir si la ville choisie avec ces nouvelles données serait toujours la même qu'avec les données actuelles.

# CONCLUSION

## ÉTAT ACTUEL

---

Le but de ce projet était d'obtenir une application prévoyant un rendez-vous pour un grand nombre de personnes se trouvant à distance les uns des autres. L'objectif était d'avoir un algorithme fonctionnel sur un serveur et de pouvoir l'utiliser depuis un prototype d'interface web.

Afin de réaliser ce projet une étude des technologies web a été effectué ce qui a permis de choisir les plus optimales pour ce projet. Le site a été conçues selon les réflexions faites aux préalables et adapté selon les problèmes rencontrés.

En l'état le site correspond aux objectifs fixé au départ. On retrouve un prototype d'interface interagissant avec l'algorithme sur le serveur. Les fonctions prévues au départ ont été réalisé avec succès quant à l'algorithme de rendez-vous, il nous renvoie bien les résultats attendus lorsqu'on entre ses données comme utilisateur. On obtient bien en retour les données d'un rendez-vous. Le calcul de ce rendez-vous se fait par le biais de données d'API et de données propres aux membres de celui-ci.

Les résultats obtenus sont encourageant et entrevois un futur positif pour cette application. Grace à la réalisation de ce prototype, j'ai pu m'apercevoir que les résultats étaient très satisfaisants et permettrait même une utilisation en l'état.

## RÉFLEXION

---

En réalisant ce projet, je me suis rendu compte de la difficulté et de la variété des choix pour créer une application. Étant entièrement libre quant aux choix des technologies utilisées et de l'architecture à employer, j'ai voulu créer ce site de la manière la plus optimal possible. J'ai aussi fait le choix d'utiliser des technologies récentes, comme Node JS, afin qu'il corresponde à notre époque. Ayant fait ce choix, cela a été l'occasion d'apprendre et de me perfectionner dans ce domaine.

Le fait de créer seul ce projet aussi abouti, est l'occasion de découvrir le temps que ça peut prendre de le réaliser dans son entièreté et que tous les choix que nous faisons vont façonner la suite du projet, en partant de son analyse à sa conception.

Il aussi fallu faire des choix sur ce que je devais implémenter en premier. Plonger dans un projet qui me tient à cœur cela me motive à vouloir optimiser la moindre fonctionnalité et m'interroge constamment sur les décisions que j'ai prises, sont-elles les plus judicieuse ? Malgré tout il faut rester concentré sur les objectifs premiers et penser à les résoudre en premier avant de vouloir ajouter une nouvelle fonctionnalité.

Des compromis ont donc été faits notamment sur l'utilisation des API. Au départ j'étais persuadé de trouver assez facilement les données nécessaires et pouvoir me concentrer sur la création de l'algorithme. Ceci n'étant pas le cas il a fallu faire en sorte que son fonctionnement ne souffre pas des ressources actuelles. C'est pourquoi, étant tributaire de leurs ressources il était préférable que l'application puisse évoluer et changer facilement des sources de données sans que l'algorithme n'en soit affecté.

En somme ce travail est une expérience importante pour un ingénieur. Être confronté à ses choix et les délais imposés nous pousse à faire les choses dans l'ordre et de la manière la plus professionnelle possible. Étant constamment confronté à des problèmes et à des imprévus on se voit obligé de revoir nos priorités afin que le produit final soit le plus proche possible de ce que l'on souhaitait au début.

## AMÉLIORATIONS

---

Pour finir, la réalisation de ce projet m'a vraiment intéressé et pendant sa création je n'ai pu m'empêcher de penser à ce qu'il pourrait devenir. Actuellement l'application n'est que le prototype de ce qu'elle pourrait être. L'objectif était de créer un algorithme de rendez-vous afin de voir où étaient les complications et qu'elles étaient les possibilités.

Au vu du travail effectué, je pense qu'il est possible de voir une belle perspective d'avenir pour cette application. Pour ce faire il faudrait consolider la base et principalement revoir les sources de données. Il serait aussi intéressant d'ajouter des fonctionnalités ainsi que de perfectionné l'interface.

Voici quelques-unes des idées de fonctionnalités :

- Une des premières fonctionnalités à ajouter serait celle qui permettrait d'avoir accès à plus de paramètres lors de la création d'un rendez-vous. Par exemple avoir le choix de cacher ou non le code du rendez-vous ou pouvoir ajouter des emails auxquels un code est envoyé directement. Nous pourrions aussi proposer des critères à choix pour décider de la ville de destination, tel que la présence de certains bâtiments (bibliothèque, cathédrale, etc.), ou des trajets n'occasionnant que peu de changements, pas plus que deux par exemple.
- Avoir plus de choix lors des résultats tels que plusieurs villes et plusieurs dates qui conviendraient au plus grand nombre de personnes avec des temps de trajets les plus satisfaisants possible. Cela laisserait la possibilité de voter pour nos résultats préférés et le favori serait donc sélectionné comme définitif.
- Pouvoir indiquer son adresse afin que le trajet reçu parte depuis celle-ci plutôt que depuis sa ville
- Créer un compte personnel qui garderait enregistré ses informations et qui éviterait à l'utilisateur de les mentionner à chaque rendez-vous. Nous pourrions aussi avoir notre propre calendrier avec nos rendez-vous et ainsi recevoir des rappels. Nous pourrions créer des listes de personnes auxquelles nous enverrions le code. Nous pourrions recevoir le code directement dans l'application plutôt que par mail.

Pour terminer, je me suis demandé s'il était possible d'atteindre tous ces objectifs ? Et si oui, combien de temps il serait nécessaire pour finir son développement ? Est-ce que le public cible visé est le bon ? Et est-ce que les choix techniques et technologiques conviennent pour l'avenir de ce projet ?

# ANNEXES

## ANNEXE 1 : MANUEL D'INSTALLATION

*Le guide suivant montre comment préparer le site en local dans les mêmes conditions qu'il a été développé. Il se peut qu'en modifiant certaines des actions le site ne fonctionne pas correctement.*

### Préparation/ configuration

#### Base de données

Avant de commencer, nous allons télécharger notre gestionnaire de base de données MySQL, Mamp pour macOS. Une fois installé il va falloir importer la base de données.

Lancez Mamp, allez sur la page d'accueil et allez dans l'onglet « tools/phpMyAdmin ».

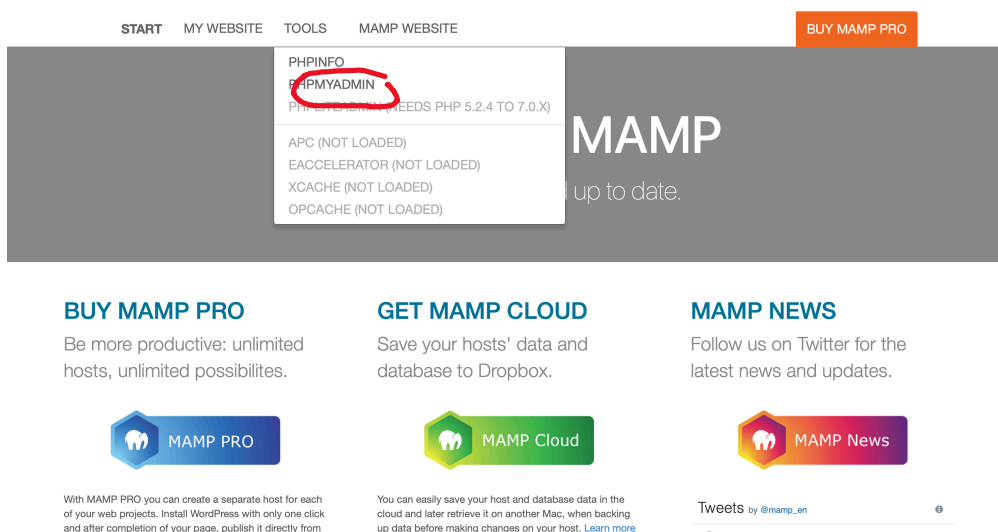
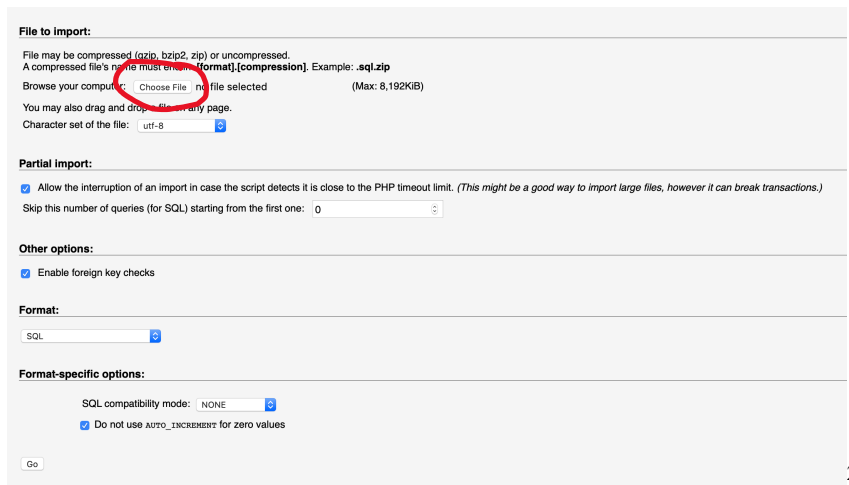


Figure 26: Accéder à phpMyAdmin

<sup>27</sup> Source : Interface de Mamp

Une fois, dedans sélectionner l'onglet « import ». Vous allez ensuite choisir la base de données « db\_meetus.sql » dans le dossier « db » du projet et cliquer sur go.



**File to import:**

File may be compressed (gzip, bzip2, zip) or uncompressed.  
A compressed file's name must end in **[format].[compression]**. Example: **.sql.zip**

Browse your computer:  No file selected (Max: 8,192KiB)

You may also drag and drop a file on any page.

Character set of the file:

**Partial import:**

☒ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. (This might be a good way to import large files, however it can break transactions.)

Skip this number of queries (for SQL) starting from the first one:

**Other options:**

☒ Enable foreign key checks

**Format:**

**Format-specific options:**

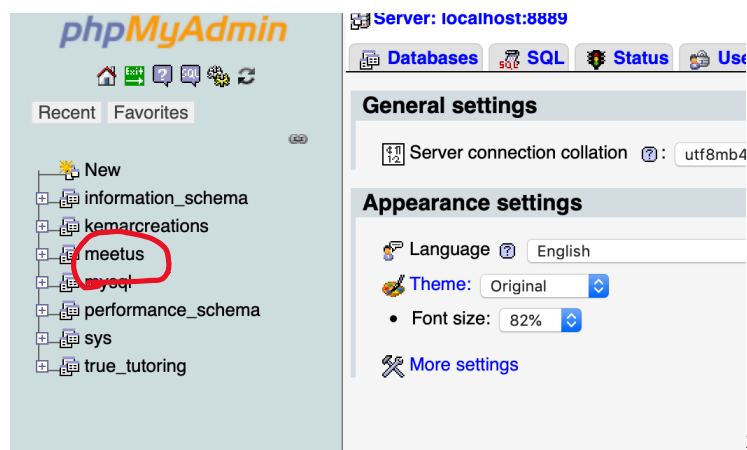
SQL compatibility mode:

☒ Do not use AUTO\_INCREMENT for zero values

28

Figure 27: Insérer une base de données

Vous devriez avoir « meetus » dans la barre de navigation de gauche.



29

Figure 28: Navigation de phpMyAdmin

<sup>28</sup> Source : Interface de Mamp

<sup>29</sup> Source : Interface de Mamp

Il va maintenant falloir créer un utilisateur pour la base de données. Dans l'onglet « user accounts » et créer un nouvel utilisateur avec les informations suivantes :

**Nom d'utilisateur :**

meet

**Mot de passe :**

Super

Cocher la case Check all

30

Figure 29: Création d'un utilisateur sur phpMyAdmin

Les paramètres de connexions à la base de données du serveur Node JS, sont réglés avec les paramètres par défaut de Mamp. Si vous souhaitez les changer c'est facilement faisable dans le fichier de configuration du code : « config/config.json »

```
{
  "node_port": 8080,
  "database": {
    "host": "localhost",
    "port": 8889,
    "db_name": "meetus",
    "user": "meet",
    "password": "Super"
  },
  "city_search": {
    "method": "geodb",
    "population": 10000,
    "range": 30,
    "limit": 10
  },
  "transport_search": {
    "method": "transport_api"
  }
}
```

31

Figure 30: Fichier de configuration

<sup>30</sup> Source : Interface de Mamp

<sup>31</sup> Source : Code de l'application

## **Serveur**

Il faudra ensuite télécharger et installer Node JS et npm.

Une fois installer vous allez devoir installer les middlewares utilisés avec la commande « npm install ». Toutes les librairies nécessaires au fonctionnement du site vont se télécharger.

## **Lancement**

Maintenant que tout est prêt, il suffit de lancer Mamp, si ce n'est pas déjà fait, et de lancer le serveur. Pour lancer le serveur il faut exécuter la commande suivante à la racine du dossier du programme :

***node server.js***

On peut maintenant se rendre dans notre navigateur, à l'adresse du site qui est par défaut :

***localhost :8080***



## ANNEXE 2 : CONFIGURATION

---

Il est possible de configurer certains paramètres dans le fichier de config. Voici les paramètres disponibles et les valeurs par défaut. Toutes ces valeurs sont modifiables, mais peuvent entraîner des erreurs si elles ne correspondent pas à ce qui est demandé.

| Paramètres           |            | Valeur        | Description  |
|----------------------|------------|---------------|--|
| <b>node_port</b>     |            | 8080          | Port sur lequel va s'exécuter le serveur.  |
| <b>database</b>      | host       | localhost     | Adresse ip de la base de données. « localhost » pour une base de données locale. |
|                      | port       | 8889          | Numéro de port de la base de données.  |
|                      | db_name    | meetus        | Nom de la base de données.   |
|                      | user       | meet          | Utilisateur de la base de données.   |
|                      | password   | Super         | Mot de passe de l'utilisateur de la base de données.                             |
| <b>city_search</b>   | method     | geodb, local  | Quelles données vont être utilisées pour chercher la ville.                      |
|                      | population | 10000         | La population minimale des villes sélectionnées.                                 |
|                      | range      | 25            | Le rayon dans lequel chercher les villes à vérifier.                             |
|                      | limit      | 10            | Le maximum de villes dans lesquels chercher.                                     |
| <b>transport_api</b> | method     | transport_api | Quelles données vont être utilisées pour trouver les trajets.                    |

Tableau 2: Données du fichier de configuration

## RÉFÉRENCES DOCUMENTAIRES

NOM DE L'AUTER Prénom ou ORGANISME, « Titre de la page », in *Nom du site*, Adresse Internet, date de la consultation

OPENSTREETMAP NOMINATIM, « OpenStreetMap Nominatim search », in *Nom du site*, [HTTPS://nominatim.openstreetmap.org](https://nominatim.openstreetmap.org), date de la consultation

GEODB CITIES API, « Find Cities & Town | Geodb API », in *Nom du site*, <http://geodb-cities-api.wirefreethought.com>, date de la consultation

TRANSPORT API, « Titre de la page », in *Nom du site*, [HTTPS://transport.opendata.ch](https://transport.opendata.ch), date de la consultation

SIMPLE MAPS, « Titre de la page », in *Nom du site*, <https://simplemaps.com/data/ch-cities>, date de la consultation

BOOTSTRAP, « Titre de la page », in *Nom du site*, A <https://getbootstrap.com/docs/4.3/getting-started/introduction/>, date de la consultation

MDN WEB DOCS, « MDN JavaScript », in *MDN web docs*, <https://simplemaps.com/data/ch-cities>, date de la consultation

NOM DE L'AUTER Prénom ou ORGANISME, « Titre de la page », in *Nom du site*, <https://sass-lang.com>, date de la consultation

NADEL Ben, « Exploring Recursive Promises In JavaScript », in *Ben Nadel*, <https://www.bennadel.com/blog/3201-exploring-recursive-promises-in-javascript.htm>