

# **DOSSIER DE PROJET**

**J'TMC**

**J'Trouve Mon Commerçant**

Retrouvez en quelques clics vos commerces locaux !



**PRÉSENTÉ ET SOUTENU PAR**

**LAURA SULLY-ALEXANDRINE**

**En vue de l'obtention du Titre Professionnel**

**Développeur Web et Web Mobile de niveau III**

**Référent de formation**

**Claire Fortin**

**Centre de formation**

**O'Clock**

**Promo**

**Marty 2020 - 2021**



# SOMMAIRE

<b>REMERCIEMENTS</b>	<b>3</b>
<b>A. INTRODUCTION</b>	<b>4</b>
<b>B. COMPÉTENCES COUVERTES PAR LE PROJET</b>	<b>5</b>
a. Développer la partie front-end d'une application en intégrant les recommandations de sécurité	5
b. Développer la partie back-end d'une application en intégrant les recommandations de sécurité	6
<b>C. RÉSUMÉ DU PROJET</b>	<b>8</b>
<b>D. CAHIER DES CHARGES</b>	<b>9</b>
a. La conceptualisation du projet	9
i. Le Minimum Viable Product version 1 et les fonctionnalités	9
ii. Le Minimum Viable Product envisagé	10
b. Les wireframes	10
c. Utilisateurs et User stories	11
i. Public visé	11
ii. User stories version 1	12
iii. User stories envisagées	13
d. Roadmap de l'application	14
<b>E. SPÉCIFICATIONS TECHNIQUES</b>	<b>16</b>
a. Le versionning	16
b. Technologies du Back	16
c. Technologies du Front	19
d. La sécurité de l'application	22
<b>F. GESTION DE PROJET</b>	<b>27</b>
a. Présentation de l'équipe	27
b. Répartitions et détail des rôles	28
c. Organisation des tâches	28
d. Outils utilisés	29
e. Le workflow de l'application	30
<b>G. PRODUCTION</b>	<b>31</b>
a. Réalisations personnelles	31
i. Sprint 0	31
ii. Sprint 1 et 2	32
iii. Sprint 2 et 3	36



iv. Sprint 4 -----	44
b. Jeu d'essai représentatif -----	48
<b>H. DIFFICULTÉS RENCONTRÉES ET VEILLE -----</b>	<b>56</b>
a. Veille sur la vulnérabilité technologique -----	56
b. Résolution de problème -----	58
c. Problématique : recherche et traduction d'extrait sur un site anglophone-----	59
<b>I. CONCLUSION -----</b>	<b>61</b>
<b>J. ANNEXES -----</b>	<b>62</b>
a. Wireframes -----	62
b. Modèle Conceptuel de Données -----	64
c. Modèle Physique de Données -----	65
d. Dictionnaire de Données -----	66



## REMERCIEMENTS

Je veux dans un premier temps remercier les membres de l'équipe avec qui j'ai partagé ce mois de pratique que l'on appelle communément chez O'clock 'Apothéose', Benjamin Papy, Guy Guyon et Tayeb Nehari, pour leur patience, leur bonne humeur et leur professionnalisme. De ce fait, nous avons pu mettre en pratique notre apprentissage, coder et développer ce projet qui me trottait dans la tête depuis quelque temps déjà. Donc, après ces cinq mois riches d'expériences intenses et beaucoup de remise en question, nous avons pu faire de cette fin de formation, un moment enrichissant tant humainement que techniquement.

Un grand MERCI aussi à Jérémy Matthew et Mickaël pour leur aide précieuse sur ce projet.

Je remercie toute l'équipe pédagogique dont Claire Fortin ma référente durant la formation, je salue son implication sans faille, sa gentillesse, son optimisme elle a su croire en moi et me mettre en confiance, Juliane qui a toujours été disponible, Jérémy Guédé mon référent d'admission qui a tout mis en oeuvre pour que j'intègre la promotion Marty.

Je remercie également l'équipe enseignante qui fait un gros travail au quotidien pour que les cours soient aussi dynamiques et ludiques qu'en présentiel.

Dans un second temps, je veux mettre en lumière mes ami.es et ma famille, sans qui je pense, que je n'aurai pas eu le courage de me lancer dans cette aventure émotionnelle.

Je remercie mon amie Béa, elle est depuis des années mon leitmotiv, car elle a aussi su se booster pour réussir ce changement de vie professionnelle, également mes anciennes collègues qui sont le témoin depuis un moment de mon envie de retaper à la porte du world wide web.

Pour finir, ma famille, mon conjoint qui me soutient, me motive et m'écoute quand je pars dans mes délires de code et mes garçons, c'est en grande partie pour eux que je fais cela pour leur montrer qu'on peut avec du travail faire ses propres choix de vie.

Merci à vous tous.



## A. INTRODUCTION

Cette idée de projet est née suite à de petites expériences du quotidien, au cœur de la crise sanitaire donc, c'est grâce ou à cause de cette situation que je me suis rendu compte que je connaissais très peu, l'activité locale de ma ville.

J'Trouve Mon Commerçant qui fut rebaptisé au cours du développement **J'TMC** par mon coéquipier Benjamin, un acronyme qui veut dire, J'T'aiMe Commerçant, est un site internet qui permet d'obtenir des informations sur ses commerces de proximités, pas seulement les informations basiques, mais des renseignements complémentaires comme leur mode de paiement, par exemple les différents titres de repas qui ne sont pas acceptés par tous les commerces car il s'agit de partenariat.

De plus, s'ils proposent des services partenaires, comme le Too Good To Go ou Phenix ou encore le Click & Collect pour faire ses courses en ligne ou les points relais comme Relais Colis, mais c'est aussi une plateforme qui permettra aux petits commerçants qui n'ont pas encore de site internet d'être visibles et d'être trouvés facilement, ou ceux qui en ont un d'avoir une meilleure visibilité.



## B. COMPÉTENCES COUVERTES PAR LE PROJET

### a. Développer la partie front-end d'une application en intégrant les recommandations de sécurité

➤ Maquetter une application

Avant de se lancer dans la conception des **wireframes**, l'équipe et moi avons pris un temps de réflexion pendant lequel, j'ai essayé de partager la vision que j'avais du projet à l'instant T, étant le **Product Owner** de JTMC et ayant une idée claire de l'expérience utilisateur que je souhaitais voir réaliser.

Nous avons chacun maquetté deux wireframes, à chaque étape, nous nous sommes concertés pour les corriger.

Ces **wireframes** comportent une version desktop (bureau) et mobile de :

- la page d'accueil
- la page inscription du commerçant
- la page connection du commerçant
- la page compte du commerçant
- la page du commerce
- la page des mentions légales
- la page contact

➤ Réaliser une interface utilisateur web statique et adaptable

Il était primordial de concevoir une version mobile, car je souhaitais que le site soit consultable de n'importe quel endroit, que l'on soit dans les transports, ou devant son ordinateur. Donc, notre site s'adapte progressivement aux écrans de mobile.

Comme nous étions une équipe entièrement originaire de la spécialisation **Symfony**, par conséquent, nous n'avions pas de **Lead Dev Front**.

J'ai été toute désignée, pour faire l'intégration de la partie **Front** et de ce fait le **responsive web design**.

J'ai utilisé partiellement le framework **Bootstrap** que pour les formulaires, je précise 'partiellement' parce que je les ai personnalisés, de manière à réaliser l'idée bien définie de ce que je voulais. Ainsi, l'intégration a été créée de toutes pièces, avec le moteur de templates **Twig**, le langage **HTML (HyperText Markup Language)** et **CSS (Cascading Style Sheets)** et les **media queries**, qui m'ont permis d'adapter les contenus de notre application en une version mobile.



### ➤ Développer une interface utilisateur web dynamique

Lors de la consultation de l'application sur un écran mobile le menu latéral d'origine statique sur la version **desktop**, passe en **responsive web design**, de manière à pouvoir être actionné par l'utilisateur via un bouton burger. Il a été entièrement conçu avec des fonctions natives de **JavaScript**. Elles interagissent avec le **DOM (Document Object Model, interface de programmation)** depuis les classes **CSS**.

Lors de nos premières réflexions pour la gestion de la localisation des commerces locaux, nous avions décidé, pour l'**API (Application Programming Interface)** de consommer l'**API Google Maps**, mais pour sa facilité d'utilisation, sa gratuité et pour les besoins du projet, nous avons opté pour l'**API OpenStreetMap**. Elle est gérée avec **Leaflet**, l'une des bibliothèques de **JavaScript**. Pour l'interaction avec le projet **Symfony** et la base de données nous avons utilisé la technologie **AJAX** qui est l'acronyme du **JavaScript et XML asynchrones**. Il combine le **JavaScript** et le **DOM** et utilise l'objet **XMLHttpRequest** pour envoyer des **requêtes HTTP** au **serveur**.

## b. Développer la partie back-end d'une application en intégrant les recommandations de sécurité ➤ Créer une base de données

Pour la création de la base de données, nous avons commencé par la modéliser en réalisant dans un premier temps le **MCD (Modèle Conceptuel de Données** en annexe), la difficulté était de définir les quantités minimum et maximum que nous souhaitions attribuer à chaque relations entre deux entités à travers les **cardinalités**.

Par la suite, le **dictionnaire de données** (en annexe) avec **Visual Studio Code**, dans le but de pouvoir ici déterminer les métadonnées et les données nécessaires à la conception de notre base de données relationnelle..

Pour finir, après concertation et correction en équipe, de nos deux documents, nous avons pu lancer la construction de notre projet dans le framework **Symfony** dont nous étions tous originaires.

Lors du développement du projet, nous avons administré, notre **Base de Données** avec le **SGBD** relationnel (**Système de Gestion de Base de Données**) **MySQL** sous **Adminer** et pour l'évolution du projet, dont je parlerai par la suite, j'ai utilisé **PhpMyAdmin**. Donc, avec **Symfony**, nous avons pu mettre en place une base de données avec un accès sécurisé, dans laquelle nous avons migré nos différentes entités grâce à l'**ORM (Object Relational Mapping)** **Doctrine**. Nous avons pu ensuite en sortir notre **MPD (Modèle Physique de Données** en annexe) qui met en évidence les relations entre nos tables caractérisées par les clés étrangères et des tables d'association.



## ➤ Développer les composants d'accès aux données

Grâce à l'**ORM Doctrine** qu'utilise **Symfony** les modèles créés à partir des entités permettent d'accéder à notre base de données via leurs **Repositories** respectifs. Ils gèrent entre autre les quatres méthodes de bases **find(\$id)**, **findAll()**, **findBy()** et **findOneBy()** et ils permettent aussi de créer des requêtes **SQL (Structured Query Language)** dynamiques avec la classe **QueryBuilder** qui utilise le langage **PHP (Hypertext Preprocessor)**, lorsqu'il s'agit de faire jouer une relation entre deux tables. Dans le cadre de notre projet, il nous a permis de créer une requête **custom (personnalisée)**, pour obtenir les données stockées en base, quand l'utilisateur du site fait une requête par le biais du moteur de recherche ou encore les filtres par activité et ou service partenaire.

Pour la partie **Front**, notamment l'affichage des points géographiques représentant les commerces inscrits sur l'application **J'TMC**, ils sont gérés avec le langage **JavaScript**, dans le but de pouvoir récupérer les coordonnées de nos commerçants, nous avons utilisé la technologie **AJAX**, pour instancier l'objet **XMLHttpRequest**.

Elle permet de récupérer les informations stockées dans la **BDD (Base De Données)** par le biais d'une fonction créée dans un contrôleur, alimenté à son tour par la méthode **custom** créée dans le **Repository** de l'entité **Store**.

## ➤ Développer la partie back-end d'une application web et web mobile

Pour la mise en place du **back-end**, nous avons tout d'abord fait appel au **bundle maker user** et au **bundle maker auth** de **Symfony**. Il permet de créer une interface d'authentification sécurisée d'un utilisateur comme un commerçant ou d'un administrateur en utilisant la méthode de protection de **Symfony** contre la faille de sécurité **CSRF (Cross-Site Request Forgery)**.

Pour le **back-office**, nous avons utilisé le **bundle EasyAdmin**, afin d'avoir une interface de gestion d'administration dans laquelle nous avons pu déterminer les entités de notre projet, que nous souhaitions administrer.



## C. RÉSUMÉ DU PROJET J'TMC

**J'Trouve Mon Commerçant** est une application à titre informatif et dans cette optique, je souhaite avec ce projet **J'TMC**, pouvoir réunir sur une seule application et sous réserve d'inscription, tous commerces de proximité qui souhaiteraient prendre part à notre aventure.

Permettre à un utilisateur lambda, sans qu'il soit obligé de créer un compte sur notre site, de pouvoir trouver leurs commerces locaux, de consulter toutes leurs informations administratives basiques comme leurs jours et horaires d'ouvertures, leurs coordonnées. Également, leurs modes de paiements acceptés, des services partenaires comme l'anti-gaspillage alimentaire ou encore les services de dépôt ou retrait de colis.

De donner à un utilisateur commerçant sans vitrine virtuelle une visibilité sur la toile **WWW (World Wide Web)**.

## D. CAHIER DES CHARGES

### a. La conceptualisation du projet

#### i. Le Minimum Viable Product version 1 et les fonctionnalités

Au début du développement, l'ambition de ce projet était grande, un compte visiteur, un compte commerçant, des mails de suivi, un système de notation à étoile, des opérations promotionnelles transmises en temps réel... Mais au vu du temps imparti au cours de ce mois de pratique et notre jeune expérience dans le domaine du développement web, Nous sommes partis du principe que le compte du commerçant et la localisation de son commerce était le minimum viable que nous devions produire, partant du fait qu'un utilisateur lambda n'a pas besoin de créer un compte pour utiliser l'application.

Ce projet est donc composé dans un premier temps :

- D'une page 'home' (accueil) : Cette page comporte un en-tête avec le logo du site, le nom de l'application, un intitulé secondaire qui exprime l'objectif du projet, des boutons **HTML** agissant comme des liens vers la page d'accueil, de connexion et d'inscription.
- D'un menu latéral statique pour la version desktop et **responsive** pour la version mobile, il comporte :
  - Une première partie composée d'un moteur de recherche, dont l'utilisateur peut se servir pour filtrer sur sa ville ou toute autre ville de son choix ce qui lui permet de trouver des commerces locaux.
  - Une deuxième partie est consacrée à une sélection de filtre par secteur d'activité du commerce.
  - Une troisième et dernière partie est consacrée aux services partenaires proposés par le commerce.
- Une 'carte de localisation' qui permet de positionner les commerces géographiquement via un marqueur adapté à la charte graphique du site et une popup avec les informations essentielles et un lien vers la fiche du commerce.
- Une page 'inscription' : Cette page contient, un formulaire permettant la création du compte du commerçant avec son courriel comme identifiant et un mot de passe pour sécuriser son authentification.



- Une page ‘connexion’ : Cette page contient, un formulaire permettant de se connecter avec le courriel et le mot de passe du commerce renseignés lors de son inscription, le bouton de connexion et un checkbox (case à cocher) Se souvenir de moi.
- Une page ‘ajouter un commerce’ : Cette page contient, un formulaire permettant la création d’un commerce avec toutes les informations que l’on retrouve dans la page ‘fiche commerce’.
- Une page ‘fiche du commerce’ : Cette page contient toutes les informations du commerce :
  - ses coordonnées
  - son activité
  - l’adresse de son site internet (s’il y en a un)
  - son adresse mail
  - ses jours et horaires d’ouvertures
  - les modes de paiements qu’il accepte
  - les services partenaires qu’il peut proposer
  - un rapide descriptif de son activité.

Sur chaque page, la navigation principale reste accessible, ainsi qu’un lien en bas de page donnant accès aux mentions légales et un lien permettant de contacter les administrateurs du site en cas de problème.

## ii. Le Minimum Viable Product envisagé

Des fonctionnalités sont prévues pour l’évolution de l’application :

Comme je l’ai évoqué précédemment, un utilisateur lambda n’a pas besoin de créer un compte pour se servir de l’application, mais je souhaite pouvoir mettre en place le même système de création de compte personnel que pour le commerçant et de ce fait fidéliser le visiteur, en lui permettant de sélectionner ses commerces favoris.

De lui permettre de recevoir de ces derniers, des alertes par mail en cas de changement de leurs informations et de suivre leurs offres promotionnelles.

- Pour le visiteur :
  - Créer un compte
  - Calculer la distance entre la position du commerce et celle du visiteur.



- Indiquer un itinéraire routier, pédestre, transport en commun ou cyclable pour atteindre le commerce.
  - Une page d'aide aux fonctionnalités de l'application J'TMC.
  - Mettre en place un système de notation à étoile du commerçant par l'utilisateur connecté ou non connecté.
- 
- Pour le commerce :
    - Ajouter une fonctionnalité 'commerce vérifié'.
    - Ajouter les informations d'accessibilité aux personnes à mobilités réduites.
    - Ajouter une fonctionnalité d'alerte destinées aux utilisateurs, les informant de tout changement sur les commerçants inscrit sur le site.
    - Une page d'aide aux fonctionnalités de l'application J'TMC.

### b. Les wireframes (en annexe)

Les **wireframes** nous ont permis de mettre en évidence les différentes fonctionnalités que nous voulions et pouvions construire lors de ce mois de pratique. Comme je l'avais déjà mentionné précédemment, nous souhaitions absolument une utilisation sur écran mobile, car il était primordial de pouvoir accéder à J'TMC de n'importe quel endroit, c'est pourquoi l'application s'adapte aux écrans mobiles donc, elle est **responsive web design**.

### c. Utilisateurs et User stories

#### i. Public visé

Cette application est destinée aux utilisateurs qui souhaitent s'informer rapidement sur leurs commerces de proximité et les différents services qu'ils peuvent proposer. Également permettre à ces commerces locaux de se faire connaître du grand public ou d'améliorer leur visibilité.



## ii. Users stories version 1

En tant que	Je veux pouvoir	Afin de (si besoin/si nécessaire)
Visiteur	accéder rapidement à la page d'accueil	faciliter ma navigation
Visiteur	accéder rapidement à la carte de localisation	faciliter ma navigation
Visiteur	savoir sur quelle page je me trouve dans l'application	faciliter ma navigation
Visiteur	visualiser les commerces locaux sur la carte	-
Visiteur	localiser la ville de mon choix	filtrer ma recherche
Visiteur	trouver les commerces locaux par secteur d'activité	filtrer ma recherche
Visiteur	trouver les commerces locaux en fonction de leurs services partenaires	filtrer ma recherche
Visiteur	consulter les informations du commerce	obtenir les informations souhaitées
Visiteur	consulter les mentions légales du site	être rassuré sur la fiabilité de l'administrateur et les informations diffusées
Visiteur	contacter l'administrateur du site	me permettre d'adresser directement mes réclamations et m'assurer de trouver un contact facilement
Commerçant	accéder rapidement à la page d'accueil	faciliter ma navigation
Commerçant	accéder rapidement à la carte de localisation	faciliter ma navigation
Commerçant	savoir sur quelle page je me trouve dans l'application	faciliter ma navigation
Commerçant	visualiser tous les commerces locaux sur la carte	-
Commerçant	créer un compte facilement sur le site	m'authentifier en tant que commerçant
Commerçant	me connecter rapidement à mon compte	éviter d'avoir à renseigner mes identifiants à chaque nouvelle connexion
Commerçant	gérer facilement mes informations personnelles	modifier mes informations personnelles
Commerçant	accéder à mon ou mes commerce(s)	gérer les informations de mon ou mes commerce(s)
Commerçant	ajouter mon ou mes commerce(s) sur le site	être visible sur la carte de localisation
Commerçant	consulter les mentions légales du site	être rassuré sur la fiabilité de l'administrateur et les informations diffusées



<b>Commerçant</b>	contacter l'administrateur du site	me permettre d'adresser directement mes réclamations et m'assurer de trouver un contact facilement
<b>Administrateur</b>	visualiser les commerces locaux sur la carte	-
<b>Administrateur</b>	accéder rapidement à la page d'accueil	faciliter ma navigation
<b>Administrateur</b>	créer un compte facilement sur le site	m'identifier en tant qu'administrateur
<b>Administrateur</b>	accéder facilement à la partie administrateur	gérer tous les comptes des utilisateurs
<b>Administrateur</b>	accéder au compte du commerçant	modifier ses informations en cas de problème
<b>Administrateur</b>	accéder aux informations du commerce	modifier ses informations en cas de problème ou supprimer des contenus illégaux

### iii. Users stories envisagées

En tant que	Je veux pouvoir	Afin de (si besoin/si nécessaire)
Visiteur	créer un compte facilement sur le site	m'authentifier en tant qu'utilisateur
Visiteur	me connecter rapidement à mon compte	éviter d'avoir à renseigner mes identifiants à chaque nouvelle connexion
Visiteur	créer une liste de commerces favoris	retrouver mes commerces préférés facilement
Visiteur	revenir rapidement sur les 5 derniers commerces visités	faciliter ma navigation
Visiteur	choisir mon mode de transport	adapter mon itinéraire
Visiteur	calculer la distance entre ma position et le commerce	connaître le temps et la distance à parcourir
Visiteur	visualiser l'itinéraire entre ma position et le commerce	suivre le chemin à suivre
Visiteur	consulter la page d'aide à la navigation du site	faciliter ma navigation
Commerçant	alerter les visiteurs des changements concernant mon commerce	informer le visiteur
Commerçant	alerter les visiteurs de mes promotions flashs	informer et fidéliser une clientèle
Commerçant	consulter la page d'aide à la navigation du site	faciliter ma navigation



Administrateur	envoyer un mail à l'utilisateur commerçant lors de la création de son compte	rassurer l'utilisateur commerçant sur la prise en compte de sa demande
Administrateur	envoyer un mail à l'utilisateur visiteur lors de la création de son compte	rassurer l'utilisateur visiteur sur la prise en compte de sa demande
Administrateur	vérifier le bien fondé du commerce	fiabiliser les informations diffusées sur mon site
Administrateur	accéder au compte du visiteur	modifier ses informations en cas de problème
Administrateur	accéder aux articles diffusés par le visiteur	les modifier ou les supprimer en cas de contenus diffamatoire ou illégaux

#### d. Roadmap de l'application

- Le chemin d'un visiteur :
  - Arrivée sur la page d'accueil avec la carte de localisation
    - Lien mentions légales qui permet de consulter les mentions légales
    - Lien contact qui permet de contacter les administrateurs du site
    - Menu latéral permet de filtrer la recherche d'un commerce par ville, code postal, secteur d'activité ou par service partenaire.
    - Carte de localisation avec les marqueurs des commerces inscrits sur l'application permettent de visualiser la position du commerce sur la carte.
    - Les marqueurs de localisation permettent d'afficher une popup du commerce avec un lien vers la fiche du commerce.
      - La fiche du commerce avec un lien retour à la page d'accueil ou le bouton Accueil ou encore le logo du site lié à la page d'accueil.
- Le chemin de l'utilisateur commerçant qui souhaite s'inscrire sur notre site :
  - Arrivée sur la page d'accueil avec la carte de localisation
    - Page d'inscription pour une première visite
    - Page de connexion après inscription.
      - Page compte du commerçant l'invitant à ajouter un commerce ou de modifier son mot de passe
        - Page de modification de mot de passe
        - Page d'ajout de commerce
        - Page de gestion du ou des commerces
          - Page fiche commerce



- Lien de connexion pour un commerçant déjà inscrit
  - Page de connexion
    - Page compte
      - Page de gestion du ou des commerces
      - Page de modification de mot de passe
      - Page d'ajout de commerce
        - Page fiche commerce
- Lien de déconnexion du commerçant. Il sera directement redirigé vers la page d'accueil.



## E. SPÉCIFICATIONS TECHNIQUES

### a. Le versioning

Pour gérer notre code de façon fluide et travailler ensemble lors du développement de notre application, nous avons utilisé l'éditeur de texte **Visual Studio Code** et l'outil de **versioning Git**, comme au cours de notre formation. Il nous a permis de travailler directement sur notre code source tout en gardant la version de base sur laquelle nous avions installé le framework **Symfony** et mis en place le gestionnaire de dépendances **Composer**.

Nous avons dans un premier temps déterminé qu'il était nécessaire d'avoir deux branches principales : une **Front** représentant l'intégration, le **responsive design (conception réactive)** et une autre pour la consommation de l'**API** de géolocalisation. Ainsi qu'une branche **Back**, elle serait scindée en plusieurs branches déterminants les différentes fonctionnalités **back-end** de notre projet, soit la mise en place des entités, des contrôleurs, des formulaires et des templates.

### b. Les technologies du Back

Comme je l'ai mentionné précédemment, nous étions une équipe entièrement **Symfony**, alors pour le **back**, nous avons utilisé essentiellement les outils mis à disposition dans le framework, géré avec le langage de programmation **PHP**.

- Pour le **back-office**, nous avons utilisé **easyAdmin**, il permet d'administrer les entités de notre choix dont l'entité **User**. Compte tenu du fait que la gestion d'un utilisateur 'commerçant' et d'un utilisateur 'administrateur' sont créés à partir de la même entité, nous devions mettre en place la gestion des droits d'accès à la partie administration du site. Ainsi nous avons configuré dans le fichier **security.yaml**, des contrôles d'accès en fonction des rôles des utilisateurs. Au regard des fonctionnalités que nous voulions développer lors de l'apothéose, nous avons déterminé qu'il serait plus productif de garder un rôle **User** pour les utilisateurs commerçants et de créer un rôle **ADMIN** pour les administrateurs.  
Donc, le rôle **User** a accès, après son authentification, à toutes les routes qui commencent par 'compte'. Étant donné que l'administrateur a, par défaut, le rôle **User** (j'y reviendrai par la suite), il a accès aux mêmes routes que le rôle **User** en plus de celles qui commencent par 'admin'.



extrait du code yaml security.yaml : création des contrôles d'accès avec les expressions régulières

```
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/compte, roles: ROLE_USER }
```

- Pour notre base de données relationnelle, nous avons utilisé le **SGBDR MySQL** avec le gestionnaire **Adminer** durant l'apothéose et par la suite avec le gestionnaire **PhpMyAdmin** pour l'évolution du projet.
- Pour l'accès à la base de données, nous nous sommes servi de l'**ORM (Object Relational Mapping) Doctrine**. Cet **ORM** fait le lien entre les entités du projet gérées en **POO (Programmation Orientée Objet)** avec **DBAL (Database Abstraction Layer)** par le biais des **repositories** de **Symfony** dans lesquelles sont gérées les méthodes de base **find(\$id)**, **findAll()**, **findBy()** et **findOneBy()** et les tables en base de données. Ainsi que la connexion **PDO (PHP Data Object)** via le fichier point d'accès appelé le **DSN (Data Source Name) '.env'**.
- Pour la persistance des données d'un commerce en base de données, nous avons utilisé les quatres opérations de base **CRUD (CREATE READ UPDATE DELETE)**. Donc nous pouvons créer, afficher, modifier et supprimer un commerce. Chaque composant de l'acronyme **CRUD** peut être associé à un type de requête en **SQL** ainsi qu'à une **méthode HTTP**.



## extrait du code PHP StoreController.php : fonction add => create

```
/** * @Route( "/compte/ajouter-votre-commerce", name="account_store_add" ) */ public function add(Request $request, FileUploader $fileUploader ): Response { $store = new Store(); $form = $this->createForm(StoreType::class, $store); $form->handleRequest($request); if ($form->isSubmitted() && $form->isValid()) { $store->setUser($this->getUser()); $this->entityManager->persist($store); $picture = $form->get('picture')->getData(); $fileUploader->moveStorePicture($picture, $store); $this->entityManager->flush(); $this->addFlash('success', 'Votre commerce à bien été ajouté à votre compte'); return $this->redirectToRoute('account_store'); } return $this->render('account/store_add.html.twig', [ 'form' => $form->createView() ]); }
```

- Grâce à la **POO** je peux créer un nouvel **objet** qui hérite des méthodes et propriétés de sa **classe**.
- La fonction **add()** qui équivaut au **create()** de l'acronyme **CRUD**, prend en paramètre la requête du commerçant, en l'occurrence ici le formulaire d'ajout de commerce, ainsi que le **service FileUploader**. Le principe du **service** dans **Symfony** est de pouvoir mettre en place une fonctionnalité qui sera réutilisable dans tout le code de l'application si nécessaire de manière à éviter de réécrire du code.
- Le **service FileUploader** a été mis en place, afin de permettre au commerçant de télécharger l'image de son commerce et de contrôler ce téléchargement.
  - Dans un premier temps, je crée une instance de la **classe Store** et je rends l'objet créé à la vue avec la méthode **render()**.



- Ensuite, je crée une instance de la **classe StoreType** qui hérite elle-même de la **classe Store**.
- Puis, avec la méthode **handlerRequest()** je gère le traitement de la saisie du formulaire en lisant les données :
  - Si il y a des erreurs de saisie, le formulaire est affiché avec les alertes d'erreur déterminées au préalable.
- Si le formulaire est soumis et validé => **isSubmitted()**, **isValid()**
- Je récupère et modifie l'utilisateur en **session**.
- Je fais appelle à **Doctrine** avec l'**entityManager()** ensuite, avec la méthode **persist()**, je lui indique qu'un nouvel objet doit être enregistré en base de données.
- Je soumets l'image du commerce aux contraintes déterminées avec le service **FileUploader**.
- La méthode **flush()** met à jour la base de données avec l'objet signalés à **Doctrine**
- Si la soumission et l'enregistrement sont validés, j'affiche un message de succès avec la méthode **addFlash()**.
- Pour finir, je redirige vers la page 'Mes commerces' du commerçant en session.

### c. Les technologies du Front

Comme nous étions entièrement **Symfony**, nous n'avions pas de **Lead Dev Front**, par conséquent, le rendu du navigateur est réalisé sous **Symfony**, via le moteur de templates **Twig**. L'un des avantages de **Twig**, c'est qu'il intègre un système d'**héritage**. Il décompose une **vue** en plusieurs blocs distincts, ce qui permet d'éviter la surcharge de notre code en étendant une **vue** à une autre. Typiquement, pour notre site le **header (entête)**, il contient le logo et la navigation principale. Le **footer (bas de page)**, il contient les liens vers les mentions légales, le lien de contact, une ligne de texte résumant le principe du site et pouvant être utile au référencement (**SEO Search Engine Optimization => référencement naturel**). Ainsi, avec ce système d'**héritage**, les **landmarks (zones)** présents dans le template parent peuvent être étendus au template enfant.

- L'intégration a été faite avec les langages **HTML** et **CSS**. (cf: **réalisations personnelles**).
- La carte de localisation et les marqueurs sont gérés avec **OpenStreetMap** et une des **bibliothèques Leaflet de JavaScript**. (cf: **réalisations personnelles**).
- L'Interrogation du serveur se fait avec la technologie **AJAX (Asynchronous Javascript And XML)** par l'instanciation de l'objet **XMLHttpRequest**. Il permet d'avoir un affichage dynamique sans avoir à rafraîchir la page. (cf: **réalisations personnelles**).



- Pour pouvoir transcrire les coordonnées, longitude et latitude, des commerçants inscrits sur notre application, nous avons consommé l'**API Adresse** du gouvernement français. Elle fonctionne avec **cURL** qui signifie **client URL(Uniform Resource Locator)** cela permet de récupérer le contenu d'une ressource accessible par réseau informatique. Cette API a été consommée dans **Symfony** via un **service** utilisé avec l'interface **HTTP Client** de manière à éviter de réécrire du code, comme je l'ai expliqué précédemment, dans tous les contrôleurs qui font un appel à l'**API Adresse**.
  - Dans un premier temps, nous avons créé une classe **Geoloc** qui intègre une fonction permettant de récupérer les coordonnées du commerce inscrit.
    - On demande à ce que soit stocké dans la variable '**\$response**' la requête transmise par le client après traitement via l'**API Adresse** et dans 'query' on lui indique la chaîne de caractère à rechercher sur cette **API Adresse** et de se limiter à un seul résultat qui sera retourné en **GeoJSON**.

#### extrait de la documentation technique de l'API Adresse du gouvernement français

Utiliser le paramètre **q** pour faire une recherche plein texte:

```
curl "https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port"
```

Avec **limit** on peut contrôler le nombre d'éléments retournés:

```
curl "https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port&limit=15"
```

#### □ Documentation technique

GET /search Point d'entrée pour le géocodage ^

Nom	Description	Type
<b>q</b>	Chaîne de caractère recherchée	string
<b>limit</b>	Contrôle le nombre d'éléments retournés	number



## extrait du code PHP Classe Geoloc.php

```
● ● ●  
public function getCoordinates($store,$limit = 1)  
{  
    $response = $this->client->request("GET","https://api-adresse.data.gouv.fr/search/", [  
        "query" => [  
            "q" => $store["road_number"] . " " . $store['address'] . " " . $store['postal_code'] .  
            " " . $store['city'],  
            "limit" => $limit  
        ]  
    ]);  
}
```

- Ensuite le contenu retourné est décodé sous forme de variable **PHP** par la fonction **json\_decode**, découpé en 4 variables. Elles permettent le placement des marqueurs sur la carte de localisation.

## extrait du code PHP Classe Geoloc.php

```
● ● ●  
  
$content = $response->getContent();  
$coordinate = json_decode($content);  
  
$result = $coordinate->features[0];  
$data['lat'] = $result->geometry->coordinates[1];  
$data['long']= $result->geometry->coordinates[0];  
$data['city']= $result->properties->city;  
$data['street']= $result->properties->street;  
  
return ($data);
```



## d. La sécurité de l'application

Pour la sécurité de l'application nous avons utilisé le **bundle maker user** de **Symfony**. Il instaure, par défaut, à tout nouvel utilisateur un rôle **User**. Il permet de créer une gestion d'authentification d'un utilisateur aux données enregistrées en base. Ainsi que le **bundle maker auth**, il génère un formulaire de **login (connexion)** sécurisé avec une gestion de route d'authentification. Il récupère les identifiants tout en créant un **token (jeton)** de sécurité. Il vérifie que la route est bien celle définie dans le **firewall (pare-feu)**. A partir des identifiants, il récupère l'utilisateur, vérifie que l'utilisateur correspond bien aux identifiants enregistrés en base. Lors de la validation de la connexion, l'utilisateur sera redirigé vers son compte.

extrait du code PHP Classe **LoginFormAuthenticator.php** généré par le **bundle maker auth** : redirige après authentification

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $providerKey)
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {
        return new RedirectResponse($targetPath);
    }

    return new RedirectResponse($this->urlGenerator->generate('account'));
}
```

Donc ces bundles, permettent de créer un environnement utilisateur avec un système d'authentification sécurisé par le biais du composant **Security**.

Ce composant **Security** met à jour un fichier nommé **security.yaml** composé de trois parties :

- **Le Firewall** : il permet de définir les parties de l'application que l'on souhaite protéger et comment les protéger.
  - La partie **dev** : représente l'environnement de développement de **Symfony** et elle est accessible en anonyme et non protégée, afin que le chemin vers les dossiers css/images/js reste accessible au cours du développement du projet.
  - La partie **main** regroupe tout ce qui sera protégé dans notre application.
    - “anonymous = true” : il est possible d'y être en tant qu'anonyme.
    - provider : app\_user\_provider : c'est le chemin de l'entité d'où proviennent les utilisateurs authentifiés.
    - guard : authenticators : App\Security\LoginFormAuthenticator : c'est le chemin de la classe **LoginFormAuthenticator**. Il prend en charge les différents contrôles de sécurité.



- `logout : path : app_logout` : c'est le chemin de déconnexion.

```
firewalls:  
    dev:  
        pattern: ^/(_(profiler|wdt)|css|images|js)/  
        security: false  
    main:  
        anonymous: true  
        lazy: true  
        provider: app_user_provider  
        guard:  
            authenticators:  
                - App\Security\LoginFormAuthenticator  
        logout:  
            path: app_logout
```

extrait du code `yaml security.yaml` : mis à jour par le bundle maker auth

- **Providers** : il permet de définir l'endroit où se trouve nos données à protéger et de comparer les données en base avec celles soumises par un utilisateur.
  - En l'occurrence ici on demande à ce que la tentative de connexion d'un utilisateur, soit redirigée vers l'entité **User** et le chemin sécurisé défini dans le **Firewalls main** : `app_user_provider`, de manière à comparer les données à celles en base de données.

extrait du code `yaml security.yaml` : mis à jour par le bundle maker auth

```
providers:  
    # used to reload user from session & other features (e.g. switch_user)  
    app_user_provider:  
        entity:  
            class: App\Entity\User  
            property: email
```

- Ici c'est la propriété **email** qui fait l'objet d'une comparaison et si les données ne correspondent pas alors, la tentative de connexion est avortée et un message d'alerte stipulant que l'email ne correspond pas, est notifié à l'utilisateur.



## extrait du code PHP Classe LoginFormAuthenticator.php générée par le bundle maker auth : mise en place du CsrfToken

```
● ● ●  
public function getUser($credentials, UserProviderInterface $userProvider)  
{  
    $token = new CsrfToken('authenticate', $credentials['csrf_token']);  
    if (!$this->csrfTokenManager->isTokenValid($token)) {  
        throw new InvalidCsrfTokenException();  
    }  
  
    $user = $this->entityManager->getRepository(User::class)->findOneBy(['email' =>  
$credentials['email']]);  
  
    if (!$user) {  
        // fail authentication with a custom error  
        throw new CustomUserMessageAuthenticationException('L\'email ne correspond pas !');  
    }  
  
    return $user;  
}
```

```
● ● ●  
security:  
    encoders:  
        App\Entity\User:  
            algorithm: auto
```

## extrait du code : security.yaml mis à jour par le bundle maker auth

- **Encoders** : il permet de définir un algorithme d'encodage et pour notre application. Comment nous souhaitons hacher le mot de passe de l'utilisateur enregistré dans notre base de données.
- Ainsi avant d'enregistrer le mot de passe de l'utilisateur en base de données, il sera encodé donc haché.

## extrait du code PHP Classe LoginFormAuthenticator.php généré par le bundle maker auth : processus d'authentification

```
/*
 * @Route("/inscription", name="register")
 */
public function index(Request $request, UserPasswordEncoderInterface $encoder)
{
    $user = new User();
    $form = $this->createForm(RegisterType::class, $user);

    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) { // If form is submitted and valid

        // I tell him to fill in the fields of user entity with the fields entered in the form
        $user = $form->getData();

        // Before saving the user I want to hashed the password that comes from the user entity.
        $password = $encoder->encodePassword($user, $user->getPassword());

        // Change the password and instead I put the one I just hached
        $user->setPassword($password);

        // I tell him to keep it in memory
        $this->entityManager->persist($user);

        // and save it in the database
        $this->entityManager->flush();

        $this->addFlash('succes', 'Votre compte a bien été créé, Vous pouvez dès à présent vous connecter !');

        return $this->redirectToRoute('app_login');
    }
}
```

- En plus d'avoir sur le mot passe l'**algorithme auto**, géré par la classe **LoginFormAuthenticator** généré par le **bundle maker auth**, nous déterminons, lors de l'inscription de l'utilisateur une double vérification, de manière à ce que l'utilisateur puisse contrôler la saisie de son mot de passe, introduite par la contrainte **RepeatedType** de **Symfony**. Cette contrainte sert à contrôler l'équivalence des mots de passe saisis et affiche un message d'erreur lors de la non-conformité.



## extrait du code PHP Register.type : contrainte RepeatedType

```
    ->add('password', RepeatedType::class, [
        'type' => PasswordType::class,
        'invalid_message' => 'Les mots de passe doivent être identiques',
        'label' => 'Mon mot de passe',
        'required' => true,
        'first_options' => ['label' => 'Mot de passe :',
        'attr' => [
            'placeholder' => 'Saisir le mot de passe :'
        ]
    ],
    'second_options' => ['label' => 'Confirmez le mot de passe',
    'attr' => [
        'placeholder' => 'Confirmer le mot de passe'
    ]
],
```

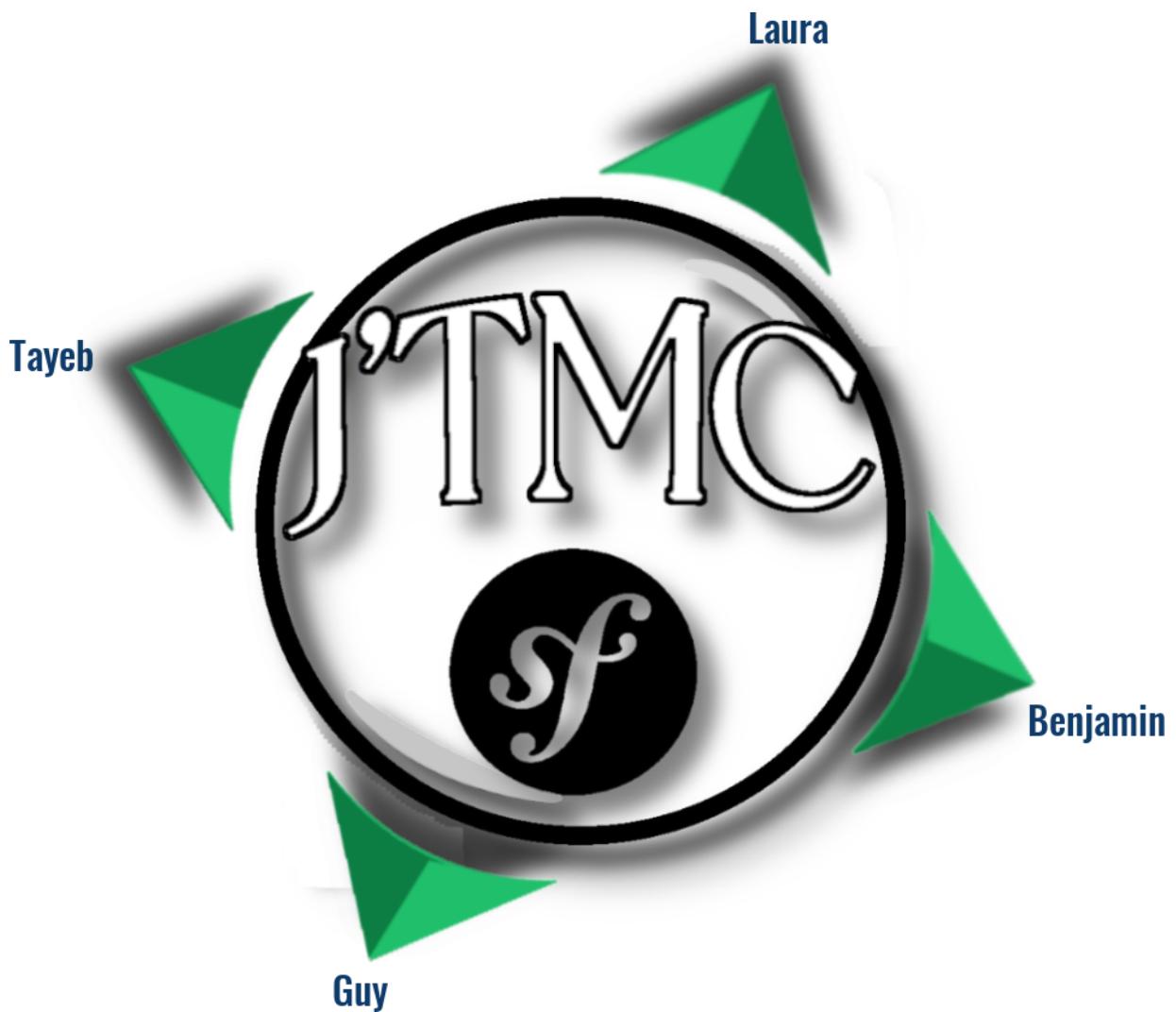
- Pour finir il n'est pas possible de créer deux utilisateurs identiques grâce à l'annotation par défaut **unique=true** stipulée dans l'entité **User**. Par la suite je montrerai comment j'ai personnalisé spécification en y ajoutant un message d'erreur. (cf: **réalisations personnelles**).

## extrait du code entité User.php : contrainte unique

```
    /**
     * @ORM\Column(type="string", length=180, unique=true)
     * @Assert\Length(
     *     min = 2,
     *     max = 30,
     *     minMessage = "Votre email doit contenir au moins {{ limit }} caractères",
     *     maxMessage = "Votre email doit contenir au maximum {{ limit }} caractères"
     *
     */
    private $email;
```

## F. GESTION DE PROJET

### a. Présentation de l'équipe



Comme j'ai déjà évoqué, l'équipe J'TMC est issue de la spécialisation **Symfony** donc, nous étions une équipe entièrement composée de développeurs **back-end**. Cela n'a pas été un gros problème en soit car nous avions, au cours de la formation, vu les bases des langages du **Front** et le framework **Symfony** nous a permis par le biais du moteur de templates **Twig** de faire l'intégration de l'application.



## b. Répartition des rôles

- Product Owner et Lead dev front :

- Laura Sully-Alexandrine

- Scrum master et Dev Back :

- Benjamin Papy

- Git Master et Dev Back :

- Guy Guyon

- Lead dev back :

- Tayeb Nehari

## c. Organisation des tâches

### ➤ Les sprints

Au cours de notre formation chez O'clock, nous avons appris à développer un projet, appliquant la méthode gestion de projet **Scrum**, elle est basée sur la méthode de développement **Agile**, dont le principe de base est d'être toujours prêt à réorienter le projet au fil de son avancement en fonction de besoins du Client et des difficultés rencontrées. Cette méthode est découpée en plusieurs petites phases essentielles au développement du produit et dont le principe est de se fixer des objectifs à court terme, donc en plusieurs petits projets que l'on appelle dans le jargon de cette méthode **Agile** des **sprints**. C'est donc sur ces phases que nous avons basé la **roadmap** de notre projet lors de ce mois d'apothéose.

Ce mois de développement a été découpé en 4 **sprints** d'une durée de sept jours.

À la fin de chaque période nous avons présenté à notre promotion à tour de rôle l'avancée de notre projet.

### 1. Sprint 0

Le sprint 0 consistait à mettre en place l'organisation et la structure du projet. Nous devions rédiger un certain nombre de documents, le **cahier des charges**, les **wireframes**, documents de veille et carnets de bord et notre outil de gestion de projet, arborescence, **Modèle Conceptuel de Données**, **dictionnaire de données**, et **Modèle Physique de Données**. Nous devions aussi penser aux technologies que nous souhaitions utiliser, sachant que nous



étions une équipe **full Symfony**, notre grosse problématique était de savoir comment nous allions consommer une **API** de géolocalisation dans notre projet développer sous le framework au langage **PHP**.

Lors de ce sprint j'ai exposé à l'équipe ma conception de l'application. Nous avons commencé à établir le cahier des charges ensemble et nous avons défini les rôles de chacun.

## 2. Sprint 1 et 2

Pendant ces deux sprints, nous avons redéfini les enjeux au fur et à mesure du développement du projet : revu nos objectifs et réadapter les documents produits pendant le sprint 0.

## 3. Sprint 3

Nous devions terminer de coder les fonctionnalités commencées pendant les sprints 1 et 2, refactoriser et nettoyer notre code et pour finir de mettre en place et préparer notre présentation sur YouTube.

### ➤ Journée type

Pendant ce mois de pratique, nous avons mis en place l'organisation d'une journée dite 'type'. Nous nous réunissions à 9 heures pour faire un point rapide (**daily scrum**) sur le moral des troupes, l'avancée de chacun, les prévisions du jour, les problèmes rencontrés et les solutions envisagées, la distribution des tâches.

### d. Outils utilisés

Pour le développement de notre site nous avons utilisé l'éditeur de texte **Visual Studio Code** qui permet entre autres un Live Share. Cette fonctionnalité consiste à partager son code et de coder en équipe.

Nous avions aussi à notre disposition d'autres outils collaboratifs, comme :

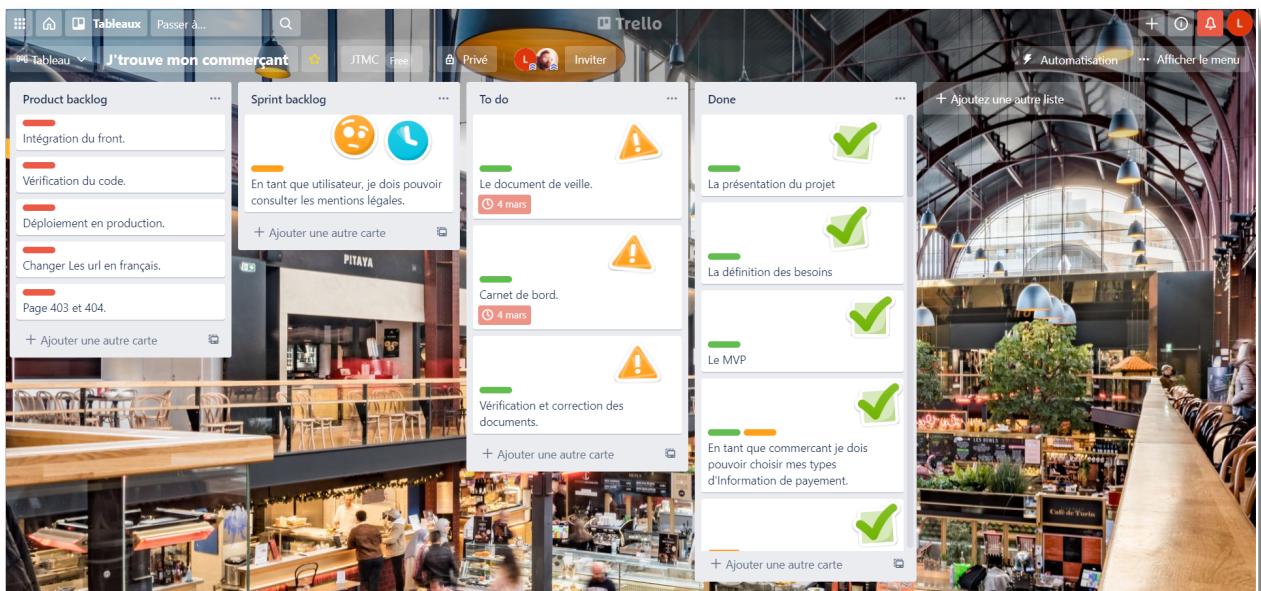
- **Slack**, est un outil de messagerie instantanée, sous la forme de canaux de discussion. Elle permet le partage d'informations, fichiers, documents. Nous avions à disposition un canal dédié à notre projet.
- Ainsi que **Discord**, qui est un logiciel de messagerie instantanée dans laquelle nous disposons de salons. Ils permettent d'échanger à l'oral ou à l'écrit, de partager son écran d'ordinateur et visualiser le code en cours d'analyse.
- Nous avions aussi la **suite de Google** soit le **Drive** pour stocker et partager des documents : **Docs** pour établir le cahier des charges, le document de vieille, le carnet de bord de l'équipe et le personnel.
- **Git**, outil de versionning. Il nous permet d'envoyer du code et d'en recevoir, dans le but de synchroniser le projet.

## e. Le Workflow de l'application

Pour le **workflow** de l'application nous avons utilisé l'outil **Trello**. Il permet de mettre en place, sous la forme d'un tableau, des séquences de tâches à accomplir du début à la fin. Le principe étant de déplacer les cartes sur les listes au fur et à mesure de leur progression. Donc, nous retrouvons ici la méthodologie **Scrum Agile**, car les tâches sont structurées sous forme de **sprints**. Ils sont mis en zone d'attente, sous le nom de **Product Backlog produit par le Product Owner**, elle répertorie toutes les tâches et user stories à effectuer pour le développement du projet et le déploiement.

Au fur et à mesure de l'avancement de notre projet, nous passons les cartes de **sprint Backlog à To Do (à faire)** pour finir à sur la colonne **Done (fait)**.

vue de l'outil Trello utilisé lors du développement du site JTMC





## G. PRODUCTION

### a. Réalisations personnelles

#### i. Sprint 0

J'ai réalisé en collaboration avec un de mes coéquipiers, les **routes** de notre application et par la suite, j'ai pris en charge le **dictionnaire de données** et le **MCD (Modèle Conceptuel de Données)**.

J'ai maquetté la page d'inscription du commerçant en version mobile et desktop. Ces documents ont beaucoup évolué au cours du développement du projet.

- Le **dictionnaire de données** (en annexe) :

Il m'a permis de déterminer les métadonnées pour chaque champ qui compose une table, son type, cela peut-être un nombre, du texte, un **booléen** se sont des valeurs de logique à deux état true(vrai) false(faux), sa spécificité, elle peut-être une clé primaire, en auto-incrémentation, non nul ou nul **UNSIGNED** indiqué pour des valeurs qui sont toujours positive.

- Le **MCD** (en annexe) est construit sur la base de six entités principales :

- **COMMERCIAL\_SERVICE** : elle correspond aux services partenaires proposés par les commerçants.
- **INFORMATION\_PAYMENT** : elle définit les types de paiements acceptés par les commerçants.
- **OPEN\_DAYS** : elle représente les jours d'ouverture du commerce.
- **USER**: elle représente dans cette version 1, l'utilisateur commerçant et l'administrateur.
- **ROLE**: elle devra introduire, lors de l'évolution du projet, l'attribution des rôles des administrateurs du site au moment de leur inscription, afin de gérer les routes correspondantes à leur future attribution. Ainsi, elle existe mais elle n'est pas utilisée dans la version actuelle.
- **STORE** qui représente le commerce.
  - Ces différentes relations sont verbalisées et quantifiées par des cardinalités multiples. Comme on peut le voir dans le **MCD** les cardinalités max. entre les entités **STORE**, **COMMERCIAL\_SERVICE**, **INFORMATION\_PAYMENT** et **OPEN\_DAYS** sont toutes en **n** ce qui crée dans le **MPD (Modèle Physique de Données** en annexe) trois tables supplémentaires appelées **tables d'association** :
    - **store\_commercial\_service**. Elle détermine la multiplicité entre les tables **store** et **commercial\_service**.

- **store\_information\_payment**. Elle détermine la multiplicité entre les tables **store** et **information\_payment**.
- **store\_open\_days**. Elle détermine la multiplicité entre les tables **store** et **open\_days**.

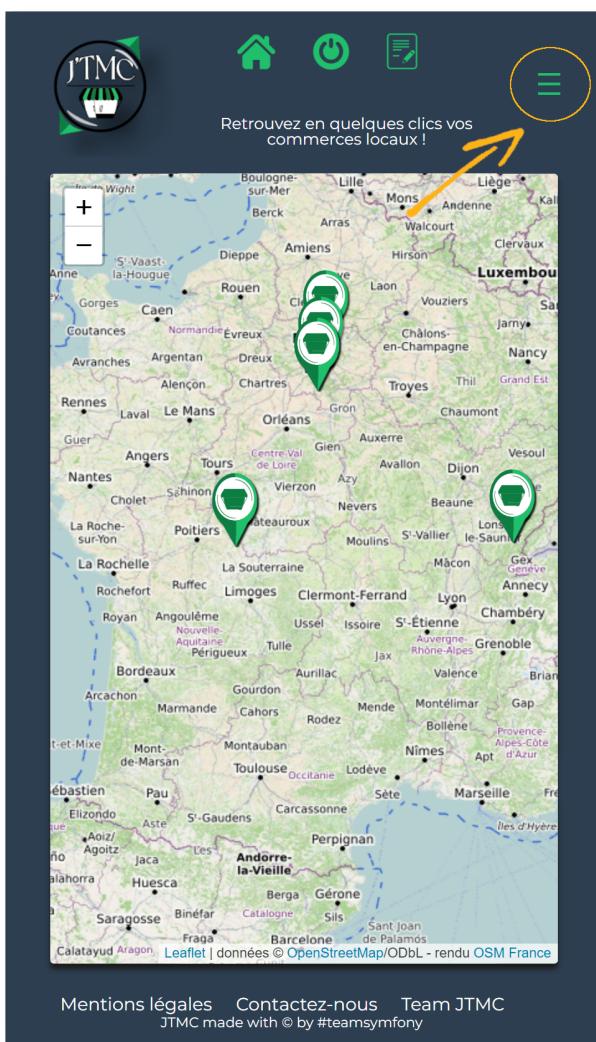
## ii. Sprint 1 et 2

### ➤ L'intégration de l'application

Comme je l'avais déjà précisé auparavant, je suis la porteuse de projet et c'est dans ce sens que je me suis proposée en tant que **Lead Dev Front**, car j'avais une idée assez précise de ce que je souhaitais pour l'application **J'TMC**. Donc, j'ai élaboré un graphisme à partir des **wireframes**, et en fonction des besoins que je souhaitais développer pour l'utilisateur.

L'intégration du site est entièrement personnalisée, réalisée sans bibliothèque, comme les boutons de lien et leur micro interaction ainsi que le logo réalisé sous **Illustrator** et le marqueur de la carte. Les icônes ont été téléchargées d'une part sur le site [icnon8.com](http://icnon8.com) et d'autre part récupérées sur internet et adaptées sur **GIMP** à la **charte graphique** du site.

J'ai choisi de partir sur un mode sombre, car personnellement cela était moins fatigant à travailler également plus esthétique. De plus, de nos jours de plus en plus d'applications proposent un mode sombre, car moins **énergivore** et comme je le disais, plus reposant pour les yeux.



### ➤ Le responsive design (conception réactive)

#### vue du site en version mobile page d'accueil

De manière à pouvoir consulter le site depuis un téléphone mobile, le contenu est adapté sans framework et entièrement avec les **media queries**, module du **CSS3** ainsi que les **breakpoints** adaptés à la taille d'un Smartphone (téléphone intelligent).

Le menu latéral gauche, en version desktop est statique et contenu dans une div **.hamburger-content**,

représente désormais, un menu **responsive** en version mobile, il est actionné par l'utilisateur par le biais d'un **bouton burger**, ce qui permet d'ouvrir le menu.

La structure du système se trouvera dans la partie **main**, **balise sémantique** du **HTML**, puisque c'est là que se



trouve le menu, permettant de filtrer la recherche de l'utilisateur. J'ai créé une div **.hamburger** qui sert de conteneur parent, elle contient le **bouton burger** dans une div **.hamburger-bouton** avec son **entité HTML (caractères spéciaux)**. En version **desktop**, elle est cachée au navigateur par la propriété **Display** en lui appliquant la valeur **none**

- Une div **.search-engine** : elle contient le moteur de recherche
- Une div **.hamburger-content** avec les filtres par activité et par service partenaire
- Une div nommée **.hamburger-sidebar**, cachée également, qui servira de nouveau conteneur à notre menu en mode **responsive design**, elle sera découpée en deux parties.
- Enfin une div nommée **.overlay** : elle sert de masque en vue de recouvrir une partie de l'application de manière à ce que l'utilisateur ne soit pas contraint de cliquer sur le **bouton burger** pour fermer le menu.

Pour mettre en place cette action, j'ai utilisé le langage **JavaScript** qui m'a permis de modifier le contenu du navigateur via le **DOM (Document Object Model)**.

extrait du code HTML template base.html.twig : représente le bouton burger et le formulaire de recherche

```
<main class="wrapper">

    <div class="hamburger">
        <!--Responsive Button-->
        <button class="hamburger-button">☰</button>
        <!--Search Engine-->
        <div class="search-engine">
            <form class="form-search-engine" action="{{ path('home') }}" method="post">
                <div class="form-search-engine-city">
                    <label for="city" class="search-city">Quelle ville ?</label><br>
                    <input type="text" name="city" placeholder="Code Postal / Ville" />
                    <button class="submit">Par ici !</button><br>
                    <a href="{{ path('home', {reset: "1"}) }}" class="menu-search-link">
                        &nbsp; Supprimer le filtre</a>
                </div>
            </form>
        </div>

        <!--Select filters-->
        <div class="hamburger-sidebar sidebar">
            <div class="hamburger-sidebar-header"></div>
            <div class="hamburger-sidebar-body"></div>
        </div>

        <div class="hamburger-overlay overlay"></div>
    </div>
```



C'est pourquoi, dans mon code **JavaScript**, j'ai initialisé plusieurs variables dans le but de pouvoir récupérer avec la méthode **querySelector**, les éléments du **DOM** correspondant à des classes, cette méthode retourne le premier élément dans le document en rapport avec le sélecteur.

- Dans un premier temps, je stocke dans deux variables ce qui sera les parties de ma div **.hamburger-sidebar**

extrait du code JavaScript hamburger.js : récupère des éléments du DOM.

```
● ● ●  
// here we create many variables to then build our function js  
// firt time  
let searchEngine = document.querySelector('.search-engine');  
let sidebarHeader = document.querySelector('.hamburger-sidebar-header');  
  
let content = document.querySelector('.hamburger-content');  
let sidebarBody = document.querySelector('.hamburger-sidebar-body');
```

- Dans le but de pouvoir modifier les éléments du **DOM**, lors de l'adaptation de l'écran, avec la propriété **innerHTML** de **JavaScript**, je stocke aussi dans des variables :
  - la div **.hamburger-button**
  - la div **.hamburger-sidebar**
  - la div **.hamburger-overlay**

extrait du code JavaScript hamburger.js : récupère des éléments du DOM.

```
● ● ●  
// second time  
let btn = document.querySelector('.hamburger-button');  
let sidebar = document.querySelector('.hamburger-sidebar');  
let overlay = document.querySelector('.hamburger-overlay');
```



- Donc, dans le contenu de ma div `.hamburger-sidebar-header`, j'insère le moteur de recherche `.search-engine`  
et dans la div `.hamburger-sidebar-body`, j'insère la div `.content`, soit les filtres par activité et service partenaire.

extrait du code JavaScript hamburger.js : modifie les éléments du DOM.

```
● ● ●  
// Here the hamburger-sidebar-body id will replace the hamburger-content id same think for search-engine id with sidebar-header id.  
sidebarHeader.innerHTML = searchEngine.innerHTML;  
sidebarBody.innerHTML = content.innerHTML;
```

- Ensuite, je crée une fonction dans le but d'activer les classes `sidebar-open` et `overlay-open` défini dans le CSS, à l'action `onclick` (propriété représentant un gestionnaire d'événement de l'élément courant) de l'utilisateur sur le bouton burger stocké dans la variable `btn`.

extrait du code JavaScript hamburger.js : fonction anonyme onclick.

```
● ● ●  
// Here we create two functions that make it possible to boost the responsive interface  
btn.onclick = function() {  
    sidebar.classList.toggle('sidebar-open');  
    overlay.classList.toggle('overlay-open');  
    // console.log(btn);  
}
```

---

extrait du code CSS : style.css fonction `translateX()`.

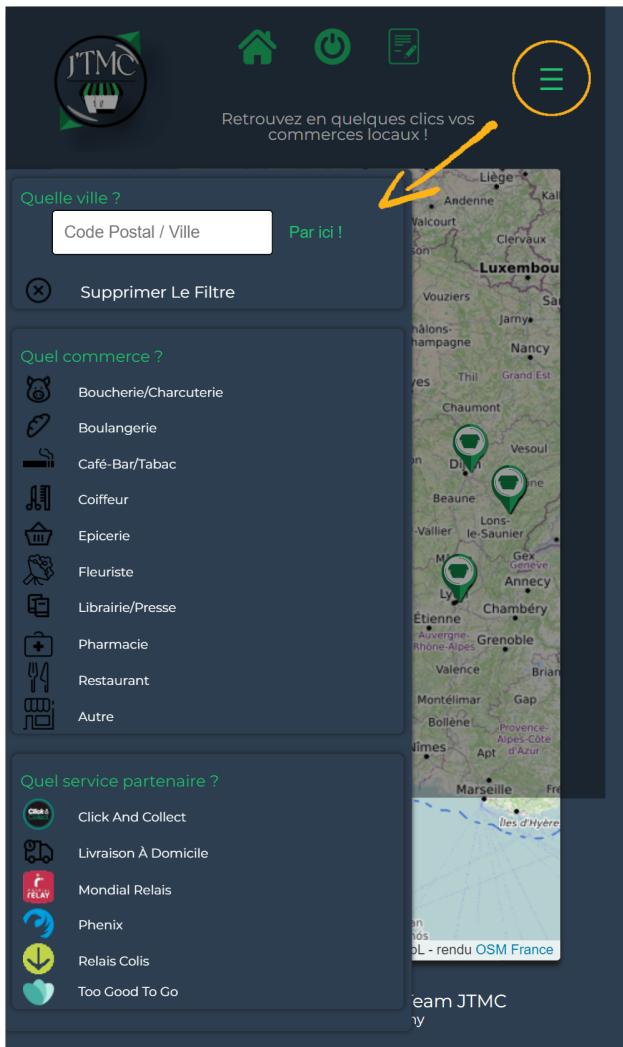
Après avoir rendu visible les éléments initialement cachés au DOM, avec la propriété **Display** en valeur **block**.

J'utilise la propriété **transform** du CSS avec la fonction `translateX()`.

Elle permet sur un axe horizontal, l'activation de la classe `sidebar-open`, en modifiant l'espace de coordonnées de ma div `.sidebar` de 0% vers une mise en forme visuelle de 100%.

Je réitère l'opération avec la classe `overlay-open`, dans le sens inverse.

```
● ● ●  
.sidebar {  
    transform: translateX(0);  
}  
.sidebar-open {  
    transform: translateX(100%);  
}  
.overlay {  
    transform: translateX(100%);  
}  
.overlay-open {  
    transform: translateX(0);  
}
```



vue du site en version mobile : page d'accueil menu responsive et masque ouverts

Pour finir, je crée une fonction qui permet, au clique de l'utilisateur sur le bouton burger, ou sur la div **.hamburger-overlay** de supprimer les classes **sidebar-open** et **overlay-open** ce qui a pour effet de refermer le menu et le masque, si la div **#overlay** est active.

extrait du code JavaScript : **hamburger.js** fonction anonyme onclick.

```
overlay.onclick = function() {
    if(overlay)
    {
        sidebar.classList.remove('sidebar-open');
        overlay.classList.remove('overlay-open');
    }
    // console.log(overlay);
}
```

### iii. Sprint 2 et 3

#### ➤ L'API de géolocalisation côté Front

Au cours de ces sprints, j'ai commencé à mettre en place l'API de géolocalisation avec **OpenStreetMap** gérée par **Leaflet** l'une des bibliothèques de **JavaScript** ainsi que les filtres par activité. Le principe est d'intégrer les données provenant de ma base de données **JTMC**, représentées par des points géographiques à l'intérieur de la carte.

**Côté Front**, comme j'utilise une bibliothèque **JavaScript**, pour accéder à ma **base de données**, je dois passer par la technologie **AJAX (JavaScript et XML asynchrones)** via l'objet **XMLHttpRequest**. Il me permet d'interagir avec mon serveur, sans avoir à recharger la page, en récupérant les données à partir de l'**URL (Uniform Resource**



**Locator**) que je définis dans la fonction dédié à la carte, dans le contrôleur **PHP** qui gère ma page d'accueil. (cf : API géolocalisation côté Back)

- Afin, de pouvoir initialiser la carte dans le navigateur, je passe par quatres étapes en me servant de plusieurs méthodes de **Leaflet** :
  1. Je crée dans en premier lieu, une div **#map** dans le template `home.html.twig` et je lui attribue des dimensions depuis mon fichier `style.css`, pour que la carte puisse s'afficher.
  2. Je crée une constante nommée **map** dans mon fichier `map.js`, dans laquelle :
    - 2.1. J'indique en paramètre de la méthode **map**, le nom de ma div **#map**, pour que ma carte s'affiche à l'intérieur.
    - 2.2. Je concatène avec la méthode **setView** en déclarant en paramètre :
      - 2.2.1. Les coordonnées géographiques d'un emplacement de mon choix.
      - 2.2.2. et la valeur du zoom par défaut au chargement de la page, qui contient la carte.
  3. J'appelle la méthode **tileLayer**, je lui donne en paramètre l'adresse du serveur **OpenStreetMap France**.
  4. Je l'ajoute, avec la méthode **addTo**, en indiquant en paramètre, ma constante **map**.

extrait du code JavaScript `map.js` : initialise la carte

```
● ● ●

// We initialize the map in the div map that I find in the home template
const map = L.map('map').setView([46.23219299999995, 2.20966699999996], 6);

// We load the "tiles"
L.tileLayer('https://{s}.tile.openstreetmap.fr/osmfr/{z}/{x}/{y}.png', {
    // It is always good to leave the link to the data source
    attribution: 'données © <a href="//osm.org/copyright">OpenStreetMap</a>/ODbL - rendu <a href="//openstreetmap.fr">OSM France</a>',
    // Add the parameters defined in the map constant to the map
}).addTo(map);
```

- Comme il sera indiqué dans la partie '**côté Back**', pour transmettre mon objet **PHP** récupéré de ma **base de données**, à mon objet **XMLHttpRequest**, je le transcris en **JSON (JavaScript Object Notation)**, dans le but de boucler et parcourir cet objet **JavaScript**.
- 1. J'ouvre **l'URL** que j'ai défini au préalable dans mon fichier `MainController.php` avec la méthode **open**.
  - 1.1. Elle se compose de trois arguments : le type de la méthode : **GET, POST**
  - 1.2. le chemin relatif de la cible, ou le nom de la ressource, du serveur.



- 1.3. et 'true' pour valider la requête asynchrone.
2. et la méthode **send** qui dit avec 'null' que je reçois, mais que je n'envoie rien (**méthode POST**).  
extrait du code JavaScript map.js : suite initialisation de la carte

```
● ● ●  
} // 'open' method of the XMLHttpRequest object which contains 3 arguments the type of the request,  
// the relative path of the target and 'true' to validate the asynchronous request.  
xmlhttp.open("GET", "http://localhost:8080/get", true);  
// 'send' method of the XMLHttpRequest object which says with 'null' that I don't have to send.  
xmlhttp.send(null);
```

3. Je déclare une variable **xmlhttp** dans laquelle j'instancie un objet **XMLHttpRequest**.
4. J'utilise l'attribut **onreadystatechange** de mon objet **XMLHttpRequest** qui s'apparente à un événement, car il écoute l'état de retour du serveur.
  - 4.1. Je crée une fonction fléchée anonyme, dans laquelle, je vérifie avec des conditions que :
    - 4.1.1. l'état de la transaction avec l'attribut **readyState** soit égale à 4. Cela signifie que la réponse est présente sur le **Client (navigateur de l'utilisateur)** et qu'elle peut-être traitée en **JavaScript**.
    - 4.1.2. Ainsi que le statut code des entêtes, avec l'attribut **status**, s'il est égale à 200, cela équivaut à un succès.

extrait du code JavaScript map.js : interroge le serveur avec l'objet XMLHttpRequest

```
● ● ●  
let xmlhttp = new XMLHttpRequest();  
xmlhttp.onreadystatechange = () =>  
{  
    // The transaction is finished so it went through all the statuses from 0 to 4 we arrive at 4  
    // which means  
    // hat the response is present on the Client and that it can be processed in JS.  
    if (xmlhttp.readyState == 4)  
    {  
        // If the transaction is successful.  
        if (xmlhttp.status == 200)  
        {
```

- 4.2. Alors, je déclare une constante nommée **donnees** dans laquelle je récupère les données reçues du serveur avec l'attribut **responseText**, que je transcris en un objet **JavaScript** avec la méthode **parse**, que je parcours avec une boucle **for()**(elle permet de répéter des actions rapidement et simplement), dans le but de récupérer les données dont j'ai besoin pour afficher les points géographiques de mes commerces et les informations qui servent à alimenter ma **popup**.



extrait du code JavaScript map.js : récupère les données du serveur transcris en JSON

```
● ● ●  
// We process the data received  
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\_Objects/JSON/parse  
const donnees = JSON.parse(xmlhttp.responseText)  
console.log(donnees);
```

- 4.3. Ces données sont également transmises à ma **vue Twig**, de manière à alimenter les filtres des activités et des services partenaires.

extrait du code JavaScript map.js : boucle for() pour positionner les marqueurs et alimenter les popups

```
● ● ●  
for (let i = 0; i < donnees.length; i++) {  
    let name = donnees[i].name  
    let id = donnees[i].id  
    let picture = donnees[i].picture  
    let marker = L.marker([donnees[i].latitude, donnees[i].longitude], { icon: icone  
}).addTo(carte)  
    marker.bindPopup('<div class="popup"><h1 class="popup-title">' + name + '</h1> <br><a class="popup-link"  
href="http://0.0.0.0:8080/' + id + '/'>Voir le commerce</a></div>')  
}
```

5. Si le chargement de la carte est trop long, j'affiche un message d'attente. Je récupère du **DOM** ma div **#map** et je remplace la carte par le message d'attente.

extrait du code JavaScript map.js : message d'attente de chargement de la carte

```
● ● ●  
} // if the map takes too long to load, a waiting message is displayed  
else {  
    document.getElementById("map").innerHTML = '<p class="loading">Chargement en cours ...  
</p>';  
}
```

- Pour afficher les points géographiques, soit les marqueurs, je déclare une variable nommée **marker** dans laquelle :

  1. J'ajoute en paramètre de la méthode **marker**, aussi de **Leaflet**, les données géographiques des différents commerces, que j'ai récupéré de la base de données, par le biais de la fonction **map** codée dans mon fichier



**MainController.php**. Je déclare également, la variable **icone** porté par la propriété **icon** de la méthode **icon**, dont j'explique l'activation quelques lignes plus bas.

1.1. J'initialise avec la méthode **addTo** en ajoutant en paramètre ma constante **map**.

extrait du code JavaScript map.js : initialise le marker



```
let marker = L.marker([donnees[i].latitude, donnees[i].longitude], { icon: icone }).addTo(carte)
```



```
let icone = L.icon({
  iconUrl: "images/marker-jtmc.png",
  iconSize: [30, 40],
  iconAnchor: [10, 40],
  popupAnchor: [4, -40]
})
```

extrait du code JavaScript map.js : suite initialisation marker

- Pour me permettre de personnaliser mon marqueur créé sous **GIMP**, je déclare une variable **icone** avec en paramètre de la méthode **icon** de **Leaflet**, 4 propriétés :
  1. l'Url de l'image
  2. la taille de l'icône
  3. Les valeurs d'ancre de l'icône [X, Y]

4. Les valeurs d'ancre de la popup [X, Y]

1.2. J'ajoute à ma variable **marker**, une **popup** avec la propriété **bindPopup**, qui lie la **popup** au marqueur et s'affiche seulement au click de l'utilisateur sur le marqueur, dans laquelle j'affecte, à l'aide de balise **HTML**, les informations choisies du commerce.

extrait du code JavaScript map.js : initialise la popup



```
marker.bindPopup('<div class="popup"><h1 class="popup-title">' + name + '</h1> <br><a class="popup-link" href="http://0.0.0.0:8080/' + id + '/>Voir le commerce</a></div>')
```



## ➤ L'API de géolocalisation côté Back

Dans le fondement du fonctionnement de **Symfony** la connexion à la base de données se fait via l'**ORM Doctrine** et le **Repository** géré avec **DBAL (Database Abstraction Layer)** ainsi, que la **POO (Programmation Orientée Objet)** qui instaure **l'héritage** entre les **classes**.

Partant du principe que les points géographiques de la carte doivent correspondre aux commerces inscrits sur notre application, j'ai ajouté deux nouvelles propriétés à notre entité **Store**, soit latitude et longitude.

Ainsi, **côté Back**, pour que je puisse récupérer les informations de la table **Store** en base de données et rendre fonctionnel les filtres par activité, il me faut interroger la base en passant par le **Repository** de l'entité **Store**.

- Le principe de cette fonction est de récupérer en **session**, originaire du composant **HttpFoundation** de **Symfony**, toutes les informations du commerce stockées dans notre base de données.

Cette fonction nommée **findByActivity()** est construite avec la classe **QueryBuilder**, originaire de l'**ORM Doctrine**, qui permet de créer des requêtes **SQL** dynamiques en utilisant le langage **PHP**.

- En premier lieu, je lui passe en paramètre une variable nommée **\$session** dans laquelle je récupérer les éléments filtrés par l'utilisateur :
  - la ville et/ou le code postal
  - l'activité

```
// create a custom function for active activity filters
public function findByActivity($session)
{
    $citySearch = $session->get('search-city');
    $activity = $session->get('activity');
```

extrait du code PHP **StoreRepository.php** :

fonction **findByActivity()**

- Je déclare les variables **\$activity** et **\$citysearch**. Avec la méthode **get()** je récupère, dans la variable **\$session** les différents filtres stockés en **session**.

```
$qb = $this->createQueryBuilder('s');
```

extrait du code PHP **StoreRepository.php** :

fonction **findByActivity()**

- Je déclare une variable nommée **\$qb**. Elle me permet de faire une nouvelle instance de l'objet **QueryBuilder** avec la méthode **createQueryBuilder()** de manière

à donner un **alias** à l'entité **Store**, que je passe en argument.



```

if($activity !== null)
{
    $qb->andWhere("s.storeActivity = :activity")
    ->setParameter('activity', $activity);
}

if($citySearch !== null)
{
    $qb->andWhere("s.city = :citySearch")
    ->orWhere("s.postalCode = :citySearch ")
    ->setParameter('citySearch', $citySearch);
}

```



```
return $qb->getQuery()->getResult();
```

**extrait du code PHP StoreRepository.php :  
fonction findByActivity()**

- Ensuite, le filtre que je récupère en **session**, je le gère sous forme de condition et j'y associe les propriétés de l'entité **Store**.
- Puis, avec la méthode **setParameter()**, je constraint la valeur en **session** aux propriétés de l'entité en question et je modifie la variable **\$activity** et **\$citySearch**.

**extrait du code PHP StoreRepository.php :  
fonction findByActivity()**

- Pour finir, je retourne le résultat de la requête avec les méthodes **getQuery()** et **getResult()**.
- Cette fonction **findByActivity()**, je la mets en application dans mon fichier **MainController.php**, car c'est avec lui et la route, que je détermine en **annotation** via le **Routing** de **Symfony**, que je gère la page d'accueil et donc la carte de localisation.

- C'est également avec ce routage que je transmets les données issues de ma base de données à ma requête **XMLHttpRequest** définit dans mon fichier **map.js**.

**extrait du code PHP MainController.php : fonction map Routing**



```

/**
 * This function display map markers from database and use custom function for active activity
filters
 * @Route("/get", name="api_geo")
*/

```



- Je crée une fonction nommée **map** dans laquelle je passe en paramètre mon **StoreRepository**, la classe **SessionInterface** issue du composant **HttpFoundation** en appliquant le système **héritage** issu de la **POO**.  
extrait du code PHP MainController.php : fonction map

```
● ● ●  
public function map(StoreRepository $storeRepository, SessionInterface $session): Response  
{
```

- Je déclare une variable nommée **\$store** dans le but d'y stocker les données reçues à l'action de l'utilisateur dans la fonction **findByActivity()** codée dans mon fichier **StoreRepository.php**.  
extrait du code PHP MainController.php : fonction map

```
● ● ●  
$store = $storeRepository->findByActivity($session);  
// dump($store);
```

- Afin d'envoyer et de transformer mon objet **PHP** en objet **JavaScript**, je me sers de la méthode **json()** du composant **Serializer** de **Symfony**. La problématique avec le composant **Serializer** ce sont les entités en relation avec l'entité transformée, car tous les attributs sont inclus par défaut lors de la **sérialisation**. Le **Serializer** analyse et transforme l'entité lorsqu'il arrive à une relation, il s'arrête et analyse l'entité en relation, il arrive de nouveau sur la même relation en sens inverse, il s'arrête de nouveau et analyse l'entité en relation. Donc, ceci s'apparente à une boucle sans fin, ce qui provoque une erreur dans le projet **Symfony** que l'on appelle une **référence circulaire**.



Afin d'éviter cette erreur, on utilise la clé **IGNORED\_ATTRIBUTE** et on lui indique sous la forme d'un tableau, les entités en relation à ignorer soit '**user, openDays, InformationPayment, CommercialService.**'

extrait du code PHP MainController.php : fonction map

```
return $this->json($store, 200, [], [
    AbstractNormalizer::IGNORED_ATTRIBUTES => [
        'User',
        'OpenDays',
        'InformationPayment',
        'CommercialService',
    ]
]);
```

#### iv. Sprint 4

##### ➤ L'API de géolocalisation côté Back

Après la formation j'ai continué à coder, au cours d'un sprint 4, dans le but de mettre en place les services partenaires proposés par les commerçants, un des fondamentaux de cette application.

- Ainsi, dans le **StoreRepository**, j'ai mis en place une **jointure** de manière à associer la table **store** et la table **commercial\_service** en relation **ManyToMany** (une instance de l'entité A peut avoir un nombre quelconque d'instance de l'entité B).
  - Après avoir renommé la fonction par **findByInformation()**, j'ai ajouté une variable **\$service** de manière à récupérer en **session** le service partenaire sélectionné.

```
// create a custom function for active activity filters
public function findByInformation($session)
{
    $citySearch = $session->get('search-city');
    $activity = $session->get('activity');
    $service = $session->get('service');
```

extrait du code PHP  
StoreRepository.php : fonction  
findByActivity() modifié



extrait du code PHP StoreRepository.php :  
fonction **findByActivity()** modifiée

- Avec la méthode d'assistance **addSelect()**, je récupère mon entité **Store** de manière à associer les deux tables avec une **jointure**, construite à l'aide de la méthode d'assistance **leftJoin()**.

```
$qb->addSelect('c');
$qb->leftjoin('s.commercialService', 'c');
```

extrait du code PHP StoreRepository.php :  
fonction **findByActivity()** modifiée

- Ensuite, le service que je récupère en **session**, je lui applique la condition au même titre que les variables **\$citySearch** et **\$activity**. Je lui associe la propriété **serviceTypes** de l'entité

```
if($service !== null)
{
    $qb->andWhere("c.serviceTypes = :service")
        ->setParameter('service', $service);
}
```

### CommercialService.

- Puis, avec la méthode **setParameter()**, je constraint la valeur en **session** aux propriétés de l'entité en question et je modifie la variable **\$service**.

- Dans le fichier **MainController.php**, je renomme ma fonction **findByActivity()** par **findByInformation()** modifiée dans le fichier **StoreRepository.php**.

extrait du code PHP MainController.php :  
fonction **map** modifiée

```
return $qb->getQuery()->getResult();
```

```
/**
 * This function display map markers from database and use custom function for active
commercial filters
 * @Route("/get", name="api_geo")
 */
public function map(StoreRepository $storeRepository, SessionInterface $session): Response
{
    $store = $storeRepository->findByInformation($session);
```



```
return $this->json($store, 200, [], [
    AbstractNormalizer::IGNORED_ATTRIBUTES => [
        'User',
        'OpenDays',
        'InformationPayment',
        'CommercialService',
        'stores'
    ]
]);
```

extrait du code PHP MainController.php :  
fonction map modifiée

- Pour finir, mes modifications **côté Back**, j'ajoute à la méthode **json()** l'attribut de l'entité **Store** à ignorer. Maintenant, j'ai dans mon objet **JavaScript** toutes les données nécessaires à transmettre à ma carte par le biais de mon objet **XMLHttpRequest**.

## ➤ L'API de géolocalisation côté Front

Dans l'objet **XMLHttpRequest**, j'ai mon nouveau **JSON** modifié.

Afin de récupérer mes nouvelles données, je complète ma requête avec une nouvelle boucle **for()** pour obtenir les services partenaires associés aux commerces. J'y déplace également les informations du marqueurs et de la popup.

extrait du code JavaScript map.js modifiée

```
// We make a loop to browse the JS object.
for(let s = 0; s < donnees.length; s++)
{
    // We collect the stores
    let id = donnees[s].id;
    let name = donnees[s].name;
    // console.log(name);
    let picture = donnees[s].picture;
    let latitude = donnees[s].latitude;
    // console.log(latitude);
    let longitude = donnees[s].longitude;

    // We collect too the commercial services in a sub table that we will have to browse
    let servicePartenaire = donnees[s].commercialService;
    // console.log(servicePartenaire);
    for(let c = 0; c < servicePartenaire.length; c++)
    {
        let serviceTypes = servicePartenaire[c].serviceTypes;
        // console.log(serviceTypes);
        let marker = L.marker([donnees[s].latitude, donnees[s].longitude], {icon: icon}).addTo(map)
        marker.bindPopup('<div class="popup"><h1 class="popup-title">' + name + '</h1>
<p class="popup-service">' +
serviceTypes + '</p><a class="popup-link" href="http://localhost:8080/' + id +
'/">Voir le commerce</a></div>')
    }
}
```



## ➤ Entité User : adresse email unique

Dans le principe de construction de l'entité **User** par le **bundle marker user** de **Symfony**, l'email est initialement unique.

Afin d'empêcher un nouvel utilisateur de s'inscrire en utilisant une adresse e-mail qui existe déjà sur notre application et l'alerter par le biais d'un message.

J'ai par suite ajouté, d'après la documentation **Symfony**, une fonction **loadValidatorMetadata()** :

- Elle prend en paramètre la classe **ClassMetadata** issue de l'**ORM Doctrine** que j'associe à la variable **\$metadata**. Cette classe permet de fournir des métadonnées **ORM** sous forme de code **PHP** simple. Elle permettra de comparer l'email saisi avec ceux stockés en base.

The screenshot shows a mobile browser interface. On the left, there is a form field labeled "E-mail \*". Inside the field, the email "laura@jtmc.fr" is typed. To the right of the field, an error message is displayed in orange text: "L'email indiqué est déjà utilisé !". Below the form, there is some descriptive text in white: "vue du site en version mobile page inscription : message d'alerte".

- J'appelle la méthode **addConstraint**, je lui passe en paramètre un nouvel objet de la classe **UniqueEntity**.
- J'indique en paramètre sous la forme d'un tableau
  - le champs qui doit être unique : 'email'
  - et le message d'alerte.

extrait du code PHP entité User.php : modifiée

The screenshot shows a code editor with a snippet of PHP code. The code defines a static public method **loadValidatorMetadata** that takes a **ClassMetadata** object as a parameter. Inside the method, it calls **\$metadata->addConstraint** on the **ClassMetadata** object, passing a new **UniqueEntity** constraint. This constraint is an array with two key-value pairs: 'fields' set to 'email', and 'message' set to the string "L'email indiqué est déjà utilisé !".

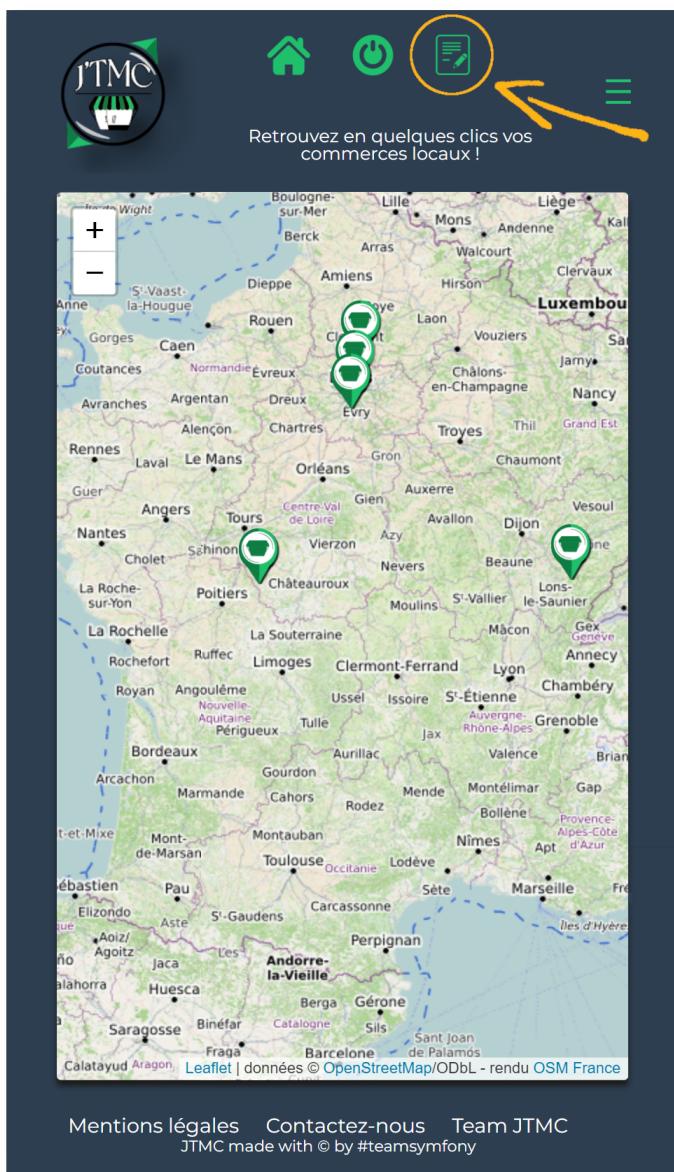
```
public static function loadValidatorMetadata(ClassMetadata $metadata)
{
    $metadata->addConstraint(new UniqueEntity([
        'fields' => 'email',
        'message' => "L'email indiqué est déjà utilisé !",
    ]));
}
```

## b. Jeu d'essai représentatif

### ➤ L'inscription et la connexion du commerçant

Au stade actuel du développement du site, seuls l'utilisateur commerçant et l'administrateur peuvent créer un compte. Le commerçant, afin de pouvoir y enregistrer, son ou ses commerce(s) et l'administrateur pour gérer le **back-office**.

Toujours dans l'optique d'une application adaptable aux écrans de mobile, le **Jeu d'essai** présenté ici sera le chemin d'un utilisateur commerçant en version mobile.



**vue du site en version mobile template : page d'accueil**

1. Arrivée sur la page d'accueil le commerçant pour créer son compte, clique sur l'icône de lien d'inscription.
2. Il est redirigé sur la page contenant le formulaire d'inscription dans lequel il renseigne ses informations.



Retrouvez en quelques clics vos commerces locaux !

Si vous êtes commerçant de proximité et que vous voulez prendre part à notre aventure...

Commencez par créez votre compte !

**Inscription**  
\* champs obligatoires

Prénom \*

Nom \*

E-mail \*

Mot de passe \*

Confirmez le mot de passe \*

**S'inscrire**

Mentions légales   Contactez-nous   Team JTMC  
JTMC made with © by #teamsymfony

vue du site en version mobile template : page d'inscription

3. Les champs ont tous l'obligation d'être renseignés avant la soumission du formulaire, dans le cas contraire une **tooltip (info-bulle)** s'affiche avec un message d'alerte.

Commencez par créez votre compte !

**Inscription**  
\* champs obligatoires

Prénom \*

Nom \*

E-mail \*

Mot de passe \*

**Veuillez renseigner ce champ.**

**S'inscrire**

Mot de passe \*      **Les mots de passe doivent être identiques**

Confirmmez le mot de passe \*

Afin que l'utilisateur ne fasse pas d'erreur de saisie dans son mot de passe, il doit le confirmer avec une double saisie, si à la soumission du formulaire, il y a une erreur alors un message d'alerte s'affiche.



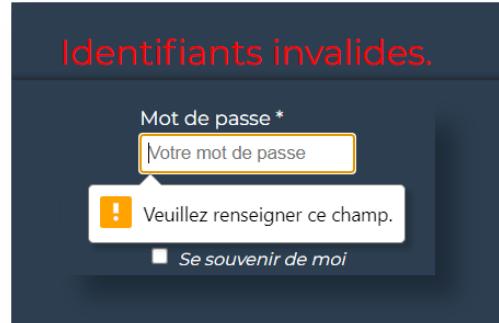
The screenshot shows the J'TMC mobile template's connection page. At the top, there is a logo for "J'TMC" with a green checkmark icon, followed by three navigation icons: a house, a power button, and a document. Below these is a message: "Retrouvez en quelques clics vos commerces locaux !". A yellow arrow points to the text "Votre compte a bien été créé, Vous pouvez dès à présent vous connecter !". The main form area has a green header "Merci de vous connecter" and a note "\* champs obligatoires". It contains fields for "E-mail \*" (placeholder "Votre email") and "Mot de passe \*" (placeholder "Votre mot de passe"). Below the fields are a green "Connection" button and a "Se souvenir de moi" checkbox. At the bottom of the page, there are links for "Mentions légales", "Contactez-nous", and "Team J'TMC", with the note "J'TMC made with © by #teamsymfony".

vue du site en version mobile template : page de connexion

4. Lorsque l'inscription est validée, le commerçant est redirigé vers la page de connexion dans laquelle s'affiche un message de confirmation.

5. Le commerçant peut, à présent, saisir son email et son mot de passe renseignés lors de son inscription.

6. Si le commerçant fait des erreurs de saisie à la connexion, un message d'erreur s'affiche, ainsi qu'une **tooltip**, s'il oublie de renseigner un champ obligatoire.





The screenshot shows the 'Mon compte' (My Account) section of the mobile template. At the top, there is a logo for 'JTMC' with a small shopping cart icon. To the right are three green icons: a house, a person, and a power button. Below the logo, a message reads: 'Retrouvez en quelques clics vos commerces locaux !'. A large green hand icon is centered on the page. Below it, a welcome message says: 'Bienvenue Martin sur votre compte.' A note below states: 'C'est dans cet espace que vous allez pouvoir gérer votre compte ainsi que vos commerces.' Under the heading 'Mon compte', there are two links: 'Gérer mes commerces' and 'Modifier mon mot de passe'. At the bottom, there are links for 'Mentions légales', 'Contactez-nous', and 'Team JTMC', followed by the text 'JTMC made with © by #teamsymfony'.

**vue du site en version mobile template : page Mon compte**

7. Après connexion, le commerçant est redirigé vers son compte, d'où il peut gérer ses informations personnelles, comme la modification de son mot passe, ainsi que la gestion de son ou ses commerce(s).

8. Pour pouvoir enregistrer un commerce, le commerçant clique sur le lien 'Gérer mes commerces'. Lors de sa première visite, il est redirigé vers la page '**Mes commerces**' l'invitant à ajouter un commerce via le lien 'Cliquez ici'.

**vue du site en version mobile template : mes commerces**

The screenshot shows the 'Mes commerces' (My Businesses) section of the mobile template. It features the same header and logo as the previous page. A message encourages users to 'Retrouvez en quelques clics vos commerces locaux !'. Below, there is a green storefront icon and the text 'Mes commerces'. A 'Retour' link is also present. A note says: 'C'est dans cet espace que vous allez pouvoir gérer vos commerces.' Another note below states: 'Vous n'avez pas encore ajouté de commerce dans votre compte.' A prominent yellow-outlined button labeled 'Cliquez ici' is located at the bottom.



## vue du site en version mobile template : Gérer mes commerce

9. Le commerçant est redirigé vers un formulaire d'ajout de commerce où il est invité à compléter les champs. Ici également, si il y a des erreurs ou des oubliers de saisie, des messages d'erreur et des **tooltips** sont affichés.

Merci de renseigner les informations relatives à votre commerce.

\* champs obligatoires

Télécharger la photo de votre commerce

Choisir un fichier Aucun fichier choisi

Activité du commerce \*

Boucherie/Charcuterie

Nom de votre commerce \*

Ex: O'feurs

Numéro de rue \*

Ex: 352

Nom de rue \*

Ex: Rue des lilas

Code postal \*

Ex: 60700

Ville \*

Ex: Pont Sainte-Maxence

Téléphone \*

Ex: 0344531894

E-mail \*

Ex: ofeur@gmail.com

Site web

Ex: www.ofeur.com

Siret \*

Ex: 159357651 48562

Horaires d'ouverture \*

Ex: 9h00 - 18h30

Jours d'ouverture \*

- Lundi
- Mardi
- Mercredi
- Jeudi
- Vendredi
- Samedi
- Dimanche

Modes de paiement acceptés \*

- Carte Bancaire
- Chèque
- Ticket restaurant
- Chèque déjeuner
- Chèque restaurant
- Chèque de table
- Chèque-Vacances ancv

Services partenaires proposés \*

- click-and-collect
- livraison-a-domicile
- mondial-relais
- phenix
- relais-colis
- too-good-to-go

Description de votre activité

Décrivez les spécialités de :

Enregister

Mentions légales Contactez-nous Team JTMC  
JTMC made with © by #teamsymfony

Merci de renseigner les informations relatives à votre commerce.

\* champs obligatoires

Télécharger la photo de votre commerce Erreur Le fichier est trop volumineux (26,08 MB). Sa taille ne doit pas dépasser 2 MB.

Choisir un fichier Aucun fichier choisi

10. Lorsque l'enregistrement du commerce est un succès, le commerçant est redirigé vers la page de son compte avec un message de validation et l'ajout de son commerce sous forme de carte.

11. A partir de cette carte il a la possibilité de :

- 11.1. modifier les informations de son commerce.
- 11.2. afficher son commerce pour vérifier le rendu, qui sera le même que pour les visiteurs.

11.3. supprimer son commerce.

Votre commerce a bien été ajouté à votre compte

Mes commerces Retour

C'est dans cet espace que vous allez pouvoir gérer vos commerces.

Ajouter un commerce

Tang Marche

Rue de l'Arquebuse

Modifier Supprimer Afficher

Mentions légales Contactez-nous Team JTMC  
JTMC made with © by #teamsymfony

Tang Marche Retour

Epicerie

27, Rue de l'Arquebuse  
21000 Dijon

Tel : 03 80 47 24

Site internet : [tang-marche@gmail.com](mailto:tang-marche@gmail.com)

Jours D'ouverture

- Mardi
- Mercredi
- Jeudi
- Vendredi
- Samedi
- Dimanche

Horaires D'ouverture

10h00 - 21h00

Mode De Paiement Accepté

- Carte Bancaire
- Chèque
- Ticket restaurant
- Chèque de table

Service Commercial Proposé

- relais-colis

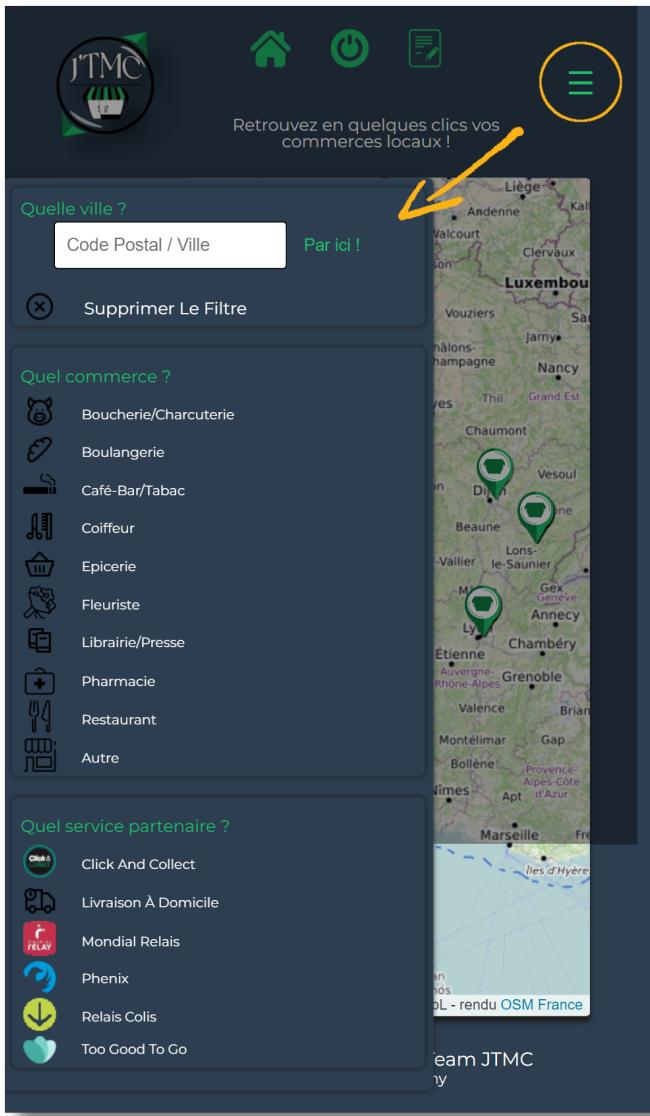
Description Du Commerce

Masque obligatoire : Les employés portent des masques et les clients doivent porter un masque et désinfecter les surfaces entre chaque client

Mentions légales Contactez-nous Team JTMC  
JTMC made with © by #teamsymfony

## ➤ Le moteur de recherche

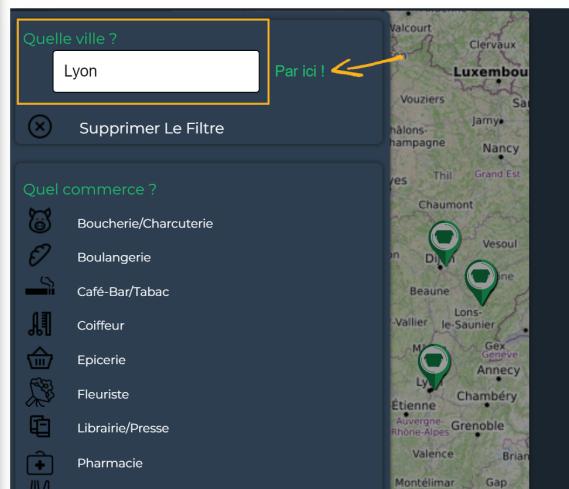
Lorsqu'un utilisateur lambda arrive sur notre application, il souhaite s'informer sur des commerces de proximité. Pour se faire, il doit saisir en mot clé, une ville et ou son code postal, afin que sa requête interagisse avec notre base de données.

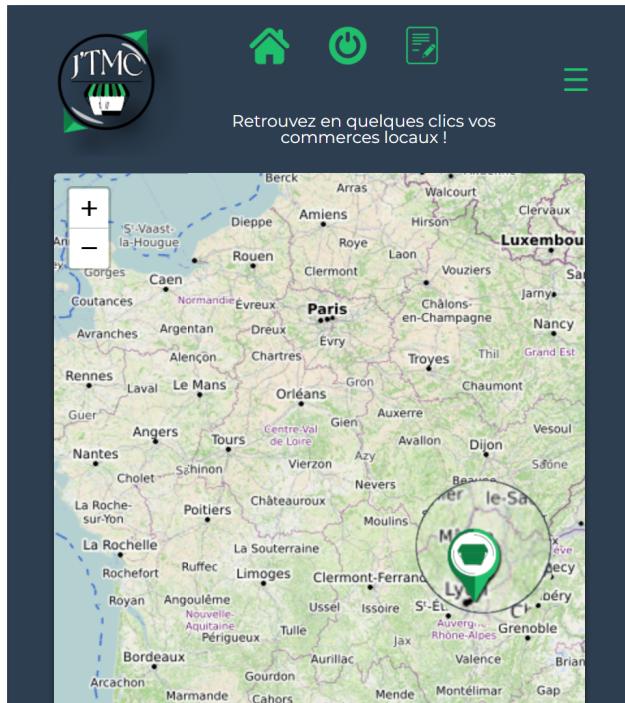


**vue du site en version mobile template : page d'accueil  
menu responsive et masque ouverts**

1. Le visiteur ouvre le menu responsive.
2. Dans le moteur de recherche, il saisit la ville et/ou le code postal souhaité.
3. il valide avec le lien Par ici !.

**vue du site en version mobile template : page d'accueil  
menu responsive et masque ouverts saisie et validation du l'utilisateur**





**J'TMC**

Retrouvez en quelques clics vos commerces locaux !



**Hiep Hung Asie** [Retour](#)

**Epicerie**  
7, Rue Passet  
69007 Lyon  
Tel : 004 72 73 19 20  
Site internet : <http://asie-marche.fr>  
[hung.asie-marche.fr](http://hung.asie-marche.fr)

**Jours D'ouverture**  
Lundi  
Mardi  
Mercredi  
Jeudi  
Vendredi  
Samedi  
Dimanche

**Horaires D'ouverture**  
6h30 - 21h00

**Mode De Paiement Accepté**  
Carte Bancaire  
Chèque restaurant

**Service Partenaire Proposé**  
livraison-a-domicile  
mondial-relais

**Description Du Commerce**  
Ce petit supermarché propose des produits alimentaires, pâtes, épices, condiments et légumes de toute l'Asie.

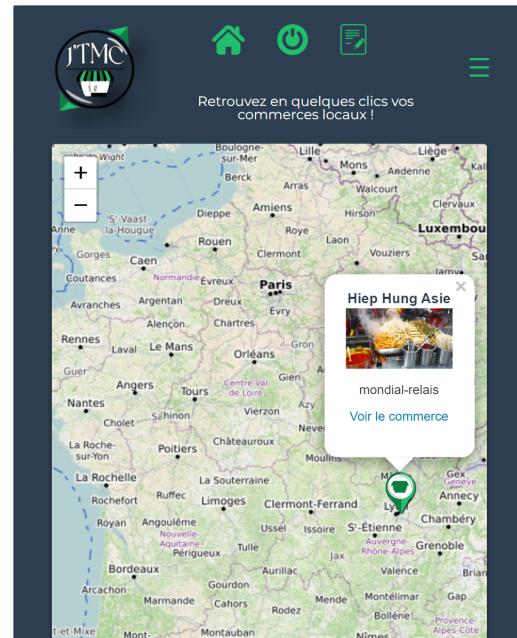
Mentions légales Contactez-nous Team JTMC  
JTMC made with © by #team Symfony

**vue du site en version mobile template : page d'accueil après validation de la requête utilisateur**

**4. Le moteur de recherche cible sur la ville demandée.**

**5. Sur la carte il ne reste plus que les commerces de la ville recherchée.**

**6. En cliquant sur le marqueur, la popup d'information du commerce s'affiche avec entre autres, un lien voir le commerce.**



**vue du site en version mobile template : fiche du commerce**

**7. Le visiteur est redirigé vers la fiche d'information du commerce.**



## ➤ Les filtres de sélection

Le visiteur peut également clarifier sa recherche, à l'aide des filtres de sélection.

The screenshot shows the JTMC mobile template homepage. At the top right is a green circular menu icon with three horizontal lines. Below it is a map of France with several green location markers. To the left of the map are three filter selection boxes:

- Quelle ville ?**: A search bar labeled "Code Postal / Ville" with a placeholder "Par ici !". Below it is a button "Supprimer Le Filtre".
- Quel commerce ?**: A list of commerce types with icons: Boucherie/Charcuterie, Boulangerie, Café-Bar/Tabac, Coiffeur, Epicerie, Fleuriste, Librairie/Presse, Pharmacie, Restaurant, and Autre. The "Click And Collect" option is highlighted with a yellow arrow pointing to it.
- Quel service partenaire ?**: A list of delivery partners with icons: Click & Collect (highlighted with a yellow arrow), Livraison À Domicile, Mondial Relais, Phenix, Relais Colis, and Too Good To Go.

vue du site en version mobile template : page d'accueil  
menu responsive et masque ouverts filtre par service partenaire

Après avoir ouvert le menu responsive :

1. Le visiteur sélectionne un filtre par activité ou par service partenaire
2. Sur la carte il ne reste plus que les commerces du filtre sélectionné.

This screenshot shows the same mobile template homepage after a filter has been applied. The map now only displays green location markers for Click & Collect services, specifically in Paris and surrounding areas like Evry, Châtenay-Malabry, and Nanterre. The other filters and the original map labels are no longer visible.

Mentions légales   Contactez-nous   Team JTMC  
JTMC made with © #team Symfony



## H. DIFFICULTÉS RENCONTRÉES ET VEILLE

### a. Veille sur la vulnérabilité technologique

Pour se prémunir des failles de sécurité le Framework **Symfony** intègre plusieurs systèmes de protection :

#### ➤ CSRF (Cross-Site Request Forgery)

Contre la faille de sécurité du système d'authentification d'un utilisateur **CSRF**, le **bundle maker auth** crée, avec la classe **LoginFormAuthenticator**, un **token de sécurité (jeton)**.

1. Dans un premier temps, l'action du **token** est implémentée avec **CsrfTokenManagerInterface**.

Elle permet de créer une clé de sécurité unique à chaque nouvelle authentification d'un utilisateur.  
extrait du code PHP classe **LoginFormAuthenticator** : génère le token de sécurité

```
● ● ●  
  
public function getUser($credentials, UserProviderInterface $userProvider)  
{  
    $token = new CsrfToken('authenticate', $credentials['csrf_token']);  
    if (!$this->csrfTokenManager->isTokenValid($token)) {  
        throw new InvalidCsrfTokenException();  
    }  
}
```

2. Ce jeton est créé et récupéré du formulaire de login par le biais d'un champ caché.  
extrait du code PHP template **login.html.twig** : génère le token de sécurité

```
● ● ●  
  
<label class="label-con" for="inputEmail">E-mail </label>  
<input class="login-input-con" type="email" value="{{ last_username }}" name="email"  
id="inputEmail" class="form-control" placeholder="Votre email" required autofocus>  
  
<label class="label-con" for="inputPassword">Mot de passe </label>  
<input class="login-input-con" type="password" name="password" id="inputPassword" class="form-  
control" placeholder="Votre mot de passe" required>  
</div>  
<button class="btn-con" type="submit">Connection</button>  
  
<input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}>
```



## ➤ SQLi (Injection SQL)

Contre les méthodes d'exploitation de failles de sécurité d'une application interagissant avec une base de données. **Symfony**, nous donne ici la possibilité de protéger les divers formulaires de notre application.

```
->add('email', EmailType::class, [
    'label' => 'email :',
    'attr' => [
        'placeholder' => 'Ex: ofleur@gmail.com',
    ],
    'constraints' => [
        new NotBlank(),
    ],
])
```

extrait du code PHP form StoreType.php :

contrainte email

1. Il intègre un système de validation par contraintes. Ces contraintes permettent d'astreindre la donnée attendue dans nos champs de formulaire, comme une adresse email. Elle permet d'imposer le format attendu à l'email.

```
if($activity !== null)
{
    $qb->andWhere("s.storeActivity = :activity")
    ->setParameter('activity', $activity);
}

if($citySearch !== null)
{
    $qb->andWhere("s.city = :citySearch")
    ->orWhere("s.postalCode = :citySearch ")
    ->setParameter('citySearch', $citySearch);
}
```

extrait du code PHP repository StoreRepository.php :

requête paramétrées

2. Il permet de paramétriser les **requêtes SQL**. Ainsi, pour notre formulaire permettant de faire une recherche par ville et/ou par code postal dans notre base de données en retournant les enregistrements correspondants à des critères établis par l'utilisateur, le **Paramater de Symfony** met en place une collection de paramètres et

permet de contraindre le type de données attendues.

## ➤ XSS (Cross Site Scripting)

En plus du paramétrage et des contraintes de **Symfony**, contre la faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web. Nous avons également le moteur de templates **Twig** :

1. Il applique un filtre par défaut sur toutes les variables que nous affichons, afin de les protéger de balises HTML malencontreuses.



extrait du code HTML & PHP template base.html.twig : variable activity échappée avec les crochets {}

```
<a href="{{ path('home', {activity: "boucherie-charcuterie"})}}>&nbsp;  
Boucherie/Charcuterie</a>
```

## b. Résolution de problème

### ➤ Serializer

Afin de pouvoir récupérer les données enregistrées en base pour que les points géographiques correspondent aux commerces enregistrés en base, donc faire coïncider le **Front** et le **Back**, j'ai utilisé la technologie **AJAX** pour interroger le serveur.

Ma problématique était d'envoyer et de transformer mon objet **PHP**, obtenu de ma base de données, en un objet **JavaScript**, de manière à ce qu'il soit exploitable dans la requête **XMLHttpRequest**. Pour cela j'ai utilisé, comme expliqué dans la partie **réalisations personnelles**, le composant **Serializer** de **Symfony**.

### ➤ HTTP Client

Le pilier majeur de notre application c'est la localisation des commerces sur la carte.

La problématique était que les données géographiques renseignées lors de nos tests n'étaient pas fiables et les marqueurs sur la carte n'étaient pas correctement géolocalisés.

Donc, pour résoudre ce problème et avoir des données adresses fiables, nous avons consommé l'**API Adresse** du gouvernement français.

Pour l'intégrer à notre projet **Symfony**, il a fallu mettre en place un service par le biais du composant **HTTP Client** de **Symfony**, de manière à consommer l'**API** gouvernementale.

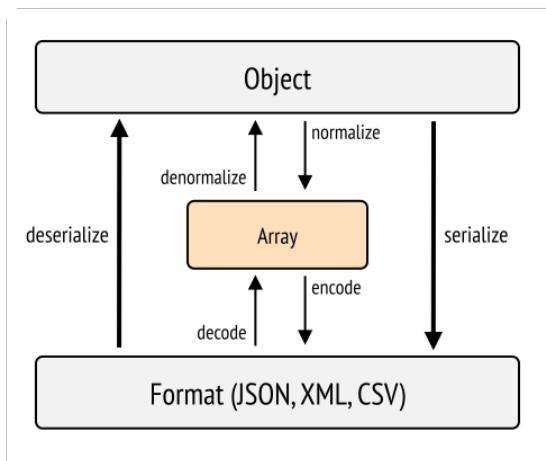
### c. Problématique : recherche et traduction d'extrait sur un site anglophone

#### ➤ Serializer

**mots clé : serializer component symfony**

extrait issu du site anglophone Symfony :

The Serializer component is meant to be used to turn objects into a specific format (XML, JSON, YAML...) and the other way around(...)



As you can see in the picture above, an array is used as an intermediary between objects and serialized contents. This way, encoders will only deal with turning specific formats into arrays and vice versa. The same way, Normalizers will deal with turning specific objects into arrays and vice versa.

#### Ignoring Attributes :

All attributes are included by default when serializing objects.

Pass an array with the names of the attributes to ignore

using the `AbstractNormalizer::IGNORED_ATTRIBUTES` key in the context of the serializer method.

#### Traduction en français :

Le composant est utilisé pour entre autres choses, pour transformer les objets en un format spécifique (XML, JSON, YAML...).

Comme vous pouvez le voir sur le schéma ci-dessus, on utilise en tableau comme intermédiaire entre les objets et sérialiser les contenus.

Normaliser ne sert qu'à transformer les objets en tableau et inversement. Encoder sert qu'à transformer le tableau en un format spécifique.

#### Ignorer les Attributs :

Lorsque l'objet est sérialisé tous les attributs sont inclus.

Pour les ignorer il faut utiliser la clé `AbstractNormalizer::IGNORED_ATTRIBUTES` et passer les noms des attributs dans un tableau.



## ➤ HTTP Client

**mots clé : component api asynchronous symfony**

extrait issu du site anglophone Symfony :

The **HttpClient** component is a low-level HTTP Client with support for both PHP stream wrappers and cURL. It provides utilities to consume APIs and supports synchronous and asynchronous operations(...)

Use the `Symfony\Component\HttpClient\HttpClient` class to make requests. In the Symfony framework this class is available as the `http_client` service. This service will be autowired automatically when type-hinting for `Symfony\Contracts\HttpClient\HttpClientInterface`.

Traduction en français :

Le composant **HttpClient** est un client **HTTP** qui supporte les flux **PHP** et **cURL**. Il permet de consommer des **API** qui fonctionnent avec des flux synchrones ou asynchrones.

L'utilisation de la classe `Symfony\Component\HttpClient\HttpClient` permet de faire des requêtes. Le framework **Symfony** intègre cette classe tout comme le service `http_client`. Il sera implémenté lors de l'autocomplétion.



## I. CONCLUSION

Ce mois d'apothéose ne nous permet pas seulement de pratiquer le code, il nous plonge en situation semi-réelle de développement d'un projet. Il nous fait prendre conscience de l'importance de ne pas se lancer tête la première dans le code. De prendre le temps de définir les bases, l'organisation de la structure du projet.

Présenter l'avancée de son projet est aussi un exercice formateur.

J'ai principalement développé la partie **Front** du projet **J'TMC**, cela m'a permis de revoir la base des langages du **Front**. J'ai, tout de même éprouvé, une certaine frustration de ne pas pouvoir revoir et pratiquer les fondements du framework **Symfony**, appris lors de ma spécialisation.

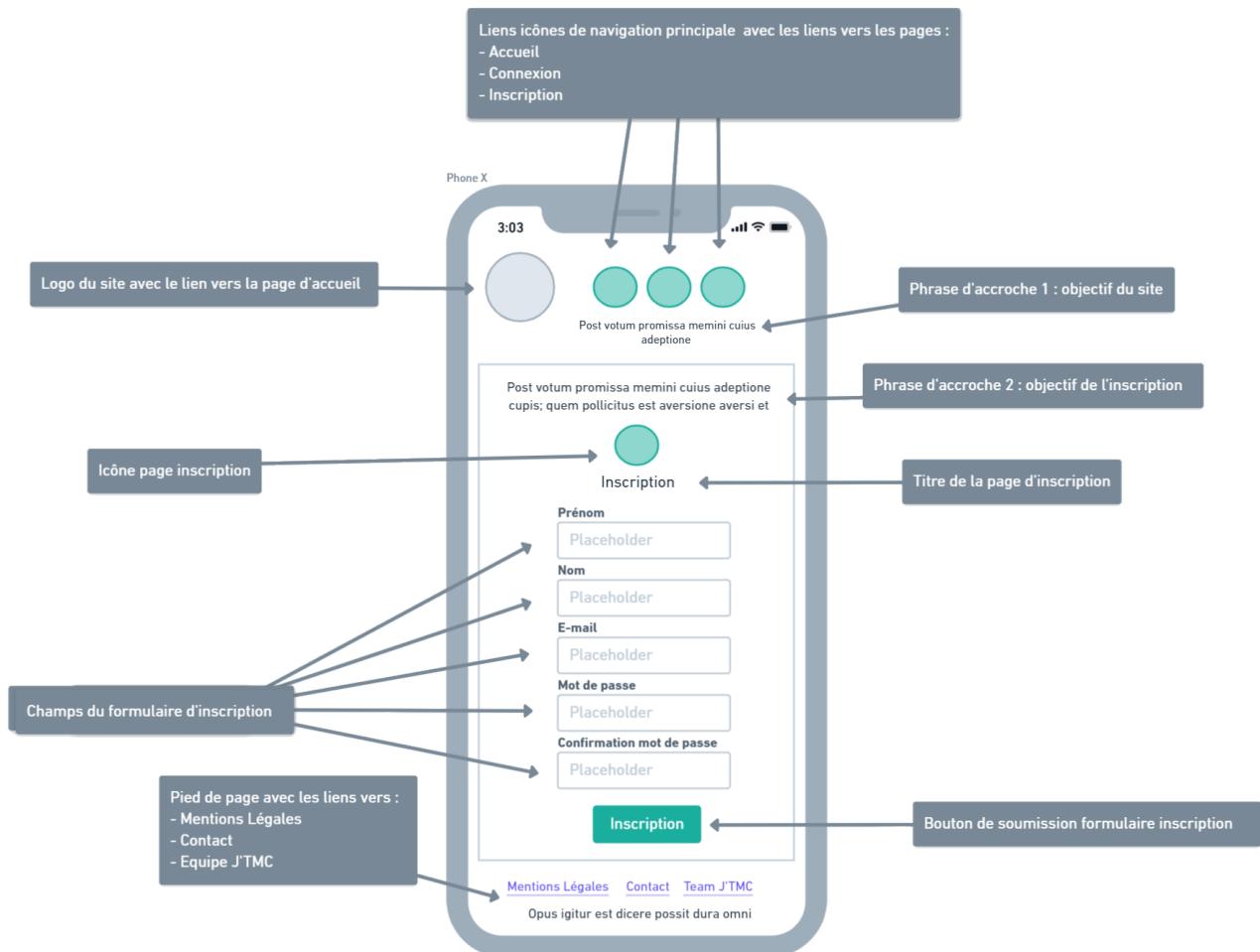
Mais, à la rédaction de ce dossier, j'ai pu, avec intérêt, revoir et mieux comprendre des éléments qui restaient flous et j'ai continué à le développer avec beaucoup de difficultés, mais aussi pas mal de satisfaction.

Le projet **J'TMC (J'Trouve Mon Commerçant)** est une plateforme d'échange d'information gratuite. Il est voué à évoluer, ne serait-ce que le déployer dans cette première version. Continuer la pratique et apprendre de nouvelles technologies. Mettre en place les users stories envisagées et dans second temps sûrement très lointain, développer une partie pour les associations Elle aurait pour but d'avoir à portée de main leurs informations administratives mais également la possibilité pour elles de diffuser l'essentiel de leur actualité comme par exemple leurs jours d'inscription.

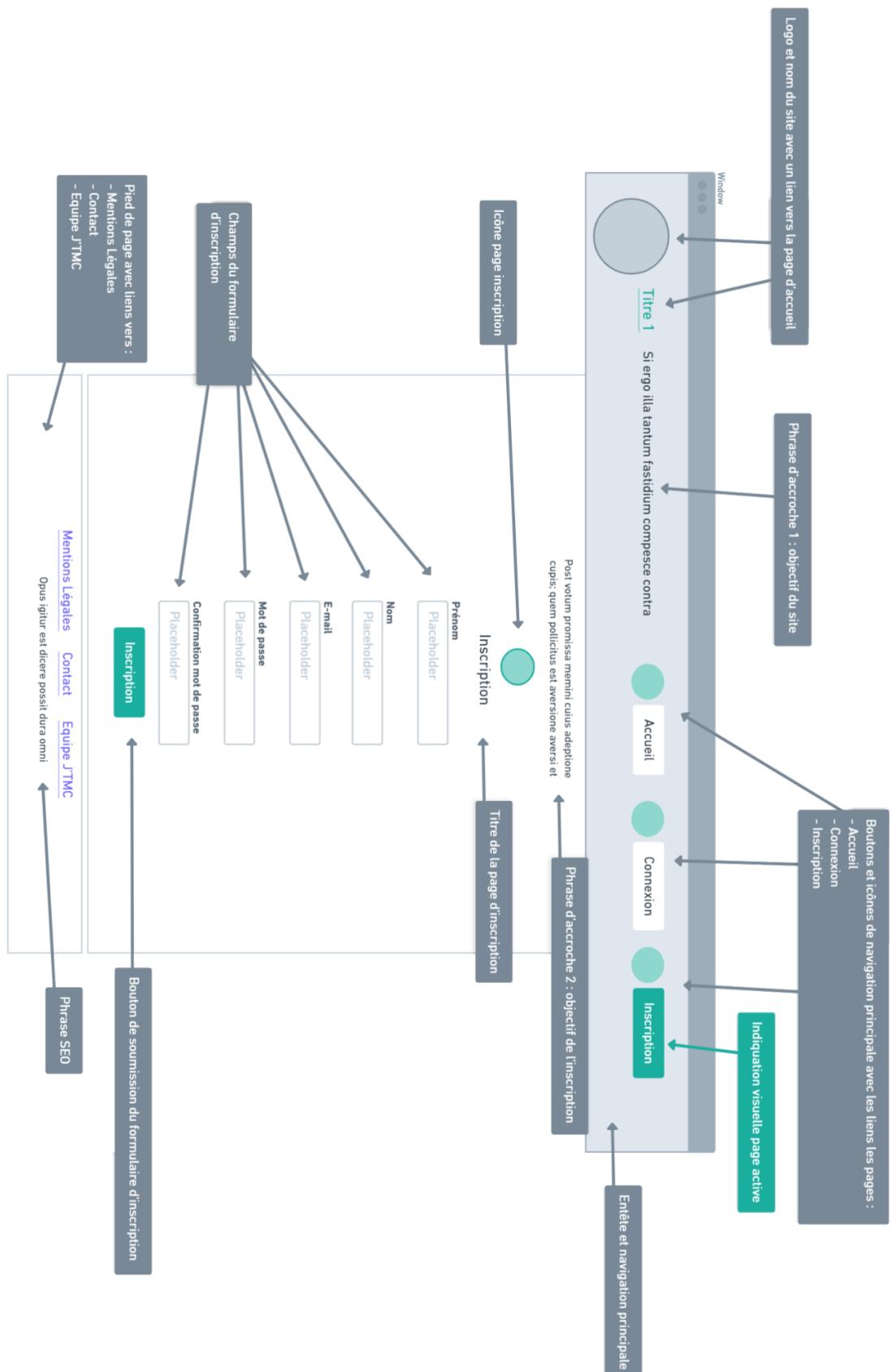
## J. ANNEXES

### a. Wireframes

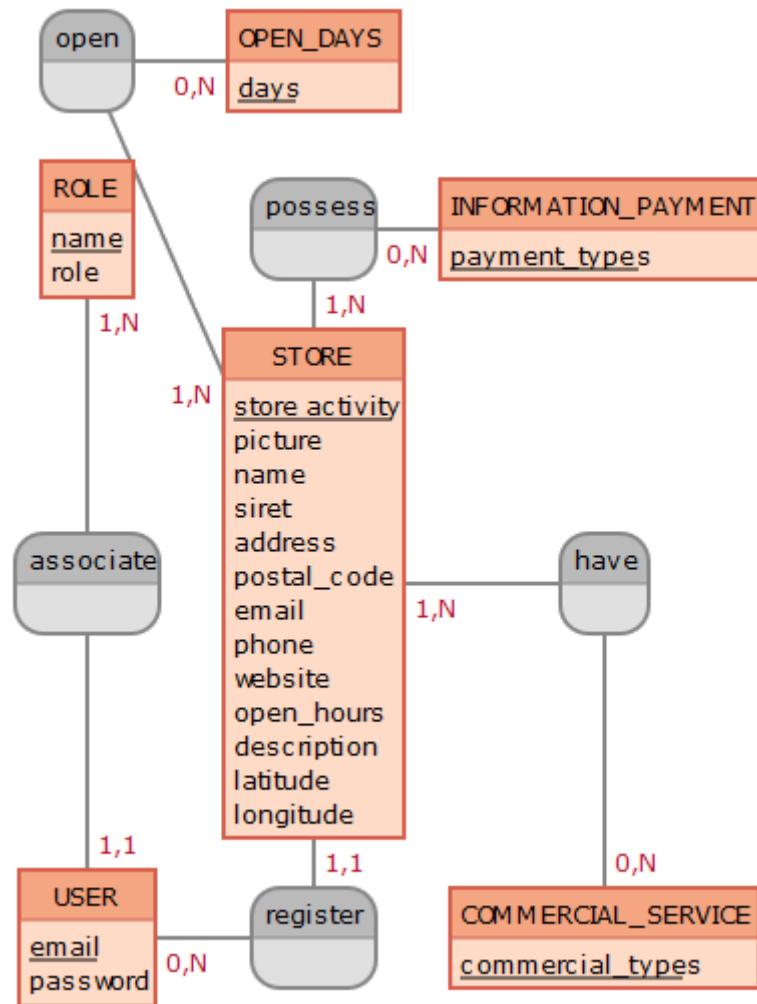
#### ➤ Page d'inscription du commerçant version mobile



## ➤ Page d'inscription du commerçant version desktop

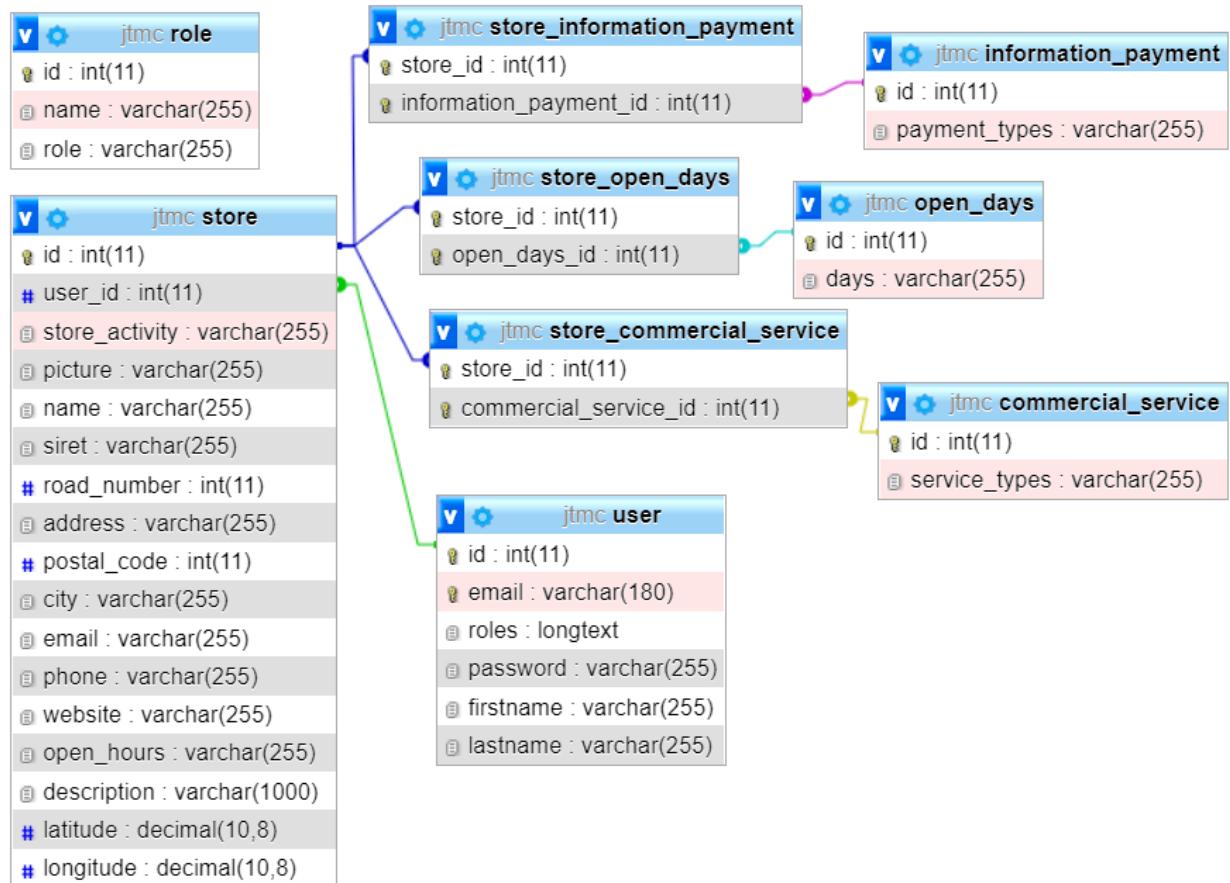


## b. Modèle Conceptuel de Données



Dans le MCD on parle **d'entité** qui sont représentées par des rectangles ainsi que de **relations** qui sont représentées par des rectangles arrondis et nommées par un verbe et de **cardinalités** qui sont les quantités **minimum** et **maximum** qui peuvent exister entre deux entités.

### c. Modèle Physique de Données du gestionnaire PhpMyAdmin





## d. Dictionnaire de Données



# Dictionnaire de données

## User (`user`)

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant de l'utilisateur
email	VARCHAR(255)	NOT NULL	email de l'utilisateur
password	VARCHAR(255)	NOT NULL	mot de passe de l'utilisateur

## Store (`store`)

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant du commerce
store_activity	VARCHAR(255)	NOT NULL	L'activité du commerce
name	VARCHAR(255)	NOT NULL	nom de l'enseigne
picture	VARCHAR(255)	NULL	L'URL de l'image du commerce
siret	INT	NOT NULL	Le siret du commerce
road_number	INT	NOT NULL	Le numéro de la rue
address	VARCHAR(255)	NOT NULL	L'adresse du commerce
postal_code	INT	NOT NULL	Le code postal
store_email	STRING	NOT NULL	L'email du commerce
phone	VARCHAR(255)	NOT NULL	Le numéro du commerce
website	VARCHAR(255)	NULL	L'adresse du site internet du commerce
open_hours	JSON	NOT NULL	Les horaires d'ouverture du commerce
description	VARCHAR(255)	NOT NULL	La description du commerce
longitude	DECIMAL(10,8)	NOT NULL	longitude du commerce
latitude	DECIMAL(10,8)	NOT NULL	latitude du commerce
user	entity	NOT NULL	Le commerçant

## Information\_payment (`information\_payment`)

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant de l'information de paiement
payment_types	VARCHAR(255)	NULL	Les différents modes de paiement du commerce

## Commercial\_service (`commercial\_service`)

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant du service commercial
service_types	VARCHAR(255)	NULL	Les différents services commerciaux

## Open\_days (`open\_days`)

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant du jour d'ouverture du commerce
days	VARCHAR(255)	NOT NULL	Jours d'ouverture du commerce

## Role (`role`)

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant du role
name	VARCHAR(255)	NOT NULL	Le nom du role
role	VARCHAR(255)	NOT NULL	Le type de role