

DOSSIER DE PROJET

O'PETITS OIGNONS



PRÉSENTÉ ET SOUTENU PAR
AYMERIC MARTINACHE

EN VUE DE L'OBTENTION DU
TITRE PROFESSIONNEL DE DÉVELOPPEUR WEB
ET WEB MOBILE

CENTRE DE FORMATION O'CLOCK
PROMOTION NEM 2024



SOMMAIRE

1. REMERCIEMENTS.....	5
2. INTRODUCTION.....	6
3. CONTEXTE DU PROJET.....	7
3.1. Présentation de l'application et du service.....	7
3.2 . Objectifs.....	7
3.3 . Présentation de l'équipe.....	7
👉 Equipe Back-end.....	7
👉 Equipe Front-end.....	8
3.4 . Méthodologie de travail.....	8
👉 Sprint 0 :.....	8
👉 Sprint 1 :.....	8
👉 Sprint 2 :.....	9
👉 Sprint 3 :.....	9
3.5 . Versionning et gestion des branches.....	9
3.6 . Rétrospectives et amélioration continue.....	9
4. COMPÉTENCES COUVERTES PAR LE PROJET.....	10
4.1 Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité.....	10
👉 Installer et configurer son environnement de travail en fonction du projet web ou web mobile.....	10
👉 Maquetter des interfaces utilisateurs.....	10
👉 Réaliser des interfaces utilisateur statiques.....	10
👉 Développer la partie dynamique des interfaces utilisateur.....	11
4.2 Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité.....	12
👉 Mettre en place une base de données relationnelle.....	12
👉 Développer des composants d'accès aux données SQL.....	12
👉 Développer des composants métier côté serveur.....	13
5. CAHIER DES CHARGES.....	14
5.1 . MVP (Minimum Vitale Production).....	14
👉 Authentification et gestion des comptes:.....	14
👉 Recherche et navigation:.....	14
👉 Gestion producteur:.....	14
👉 Interface consommateur.....	14
👉 Panier et Commande.....	15
👉 Administration du site.....	15
5.2 . Workflow.....	15

👉 Inscription/Connexion.....	15
👉 Recherche de producteurs et produits (pour les consommateurs).....	15
👉 Gestion des produits (pour les producteurs) :	16
👉 Ajout au panier et gestion des commandes :	16
👉 Gestion des commandes (pour les consommateurs) :	16
5.3 . User stories.....	16
5.4 . Les routes.....	17
5.5 . Arborescence.....	17
👉 Page d'accueil.....	17
👉 Recherche de producteurs et produits.....	18
👉 Inscription / Connexion.....	18
👉 Profil Utilisateur.....	18
👉 Panier.....	18
👉 Informations légales et support.....	18
5.6 . MCD (Modèle Conceptuel de Données).....	19
5.7 . Dictionnaire de données.....	20
5.8 . MLD (Modèle Logique de Données).....	21
5.9 . Wireframes.....	22
5.10 . Charte graphique.....	24
5.11 . Maquettes.....	25
6. SPECIFICATIONS TECHNIQUES.....	26
6.1 . Front-office.....	26
👉 Technologies Utilisées.....	26
👉 Appels API.....	30
👉 Exemples d'appels API :	30
👉 Gestion de l'État et Middleware.....	31
👉 Gestion des Tokens et Sécurité.....	33
👉 Schémas de Validation.....	34
👉 Optimisation et Performance.....	34
👉 Responsive Design.....	35
6.2 . Back-office.....	36
👉 Technologies Utilisées.....	36
👉 Gestion des Utilisateurs et des Rôles.....	38
👉 Sécurité et Authentification.....	39
👉 Doctrine ORM et Gestion des Données.....	42
7. RÉALISATIONS PERSONNELLES.....	42
👉 Carrousel d'images.....	43
👉 Formulaires.....	46
8. RÉALISATIONS PERSONNELLES.....	49
👉 Sécurité côté Front-End.....	49
👉 Sécurité côté Back-End.....	51
9. DIFFICULTÉS RENCONTRÉES.....	54
👉 Gestion du z-index.....	54
👉 Conflits Github.....	54

👉 Géolocalisation.....	55
10. CONCLUSION.....	55
👉 Wireframes et maquettes.....	58
👉 User stories.....	62
👉 Charte graphique.....	65
👉 Dictionnaire de données.....	66
👉 Modèle Logique de Données (MLD).....	69
👉 Modèle Conceptuel de Données (MCD).....	70
👉 Routes.....	71
👉 Arborescence des pages.....	75
👉 Exemples de vues de l'application.....	76



1. REMERCIEMENTS

Ce qu'on appelle "l'apothéose" porte bien son nom. Un mois de projet qui concrétise 6 mois de formation intense. 6 mois durant lesquels je me suis totalement investi et durant lesquels j'ai pris beaucoup de plaisir. J'ai élargi mes compétences professionnelles et j'ai pu les mettre en œuvre durant ce projet. Tout cela ne s'est pas fait sans difficultés mais le challenge a été relevé en équipe ! J'ai vécu cette expérience avec une grande intensité et implication grâce au soutien de nombreuses personnes. Sans eux, cette immersion n'aurait pas pu se dérouler avec des conditions aussi optimales pour suivre la formation. C'est avec une grande gratitude que je souhaite remercier...

... **mon entourage** : c'est long 6 mois. Ils ont su m'accompagner lors de moments où le doute s'installait. Je n'ai pas été très présent pour eux durant cette période mais ils ne m'en ont pas tenu rigueur, bien au contraire. Vous m'avez rassuré et encouragé à tenir bon jusqu'au bout.

... **l'équipe O'clock** : comment cette formation aurait pu se faire sans eux ? Dès le départ, avant même d'intégrer la formation, on sait que nous allons être entre de bonnes mains. Nous avons la réponse à nos différentes questions et différents canaux de communication pour trouver des infos.

Une fois la formation démarrée, nous avons une équipe pédagogique aux petits soins avec les apprenants. Toujours présent quand on en a besoin. Tous les jours un petit message pour nous accompagner à passer une bonne journée.

Et quelle équipe de professeurs ! Du début à la fin, vous avez été géniaux. Le format télé présentiel n'est pas forcément évident mais vous avez réussi à nous tenir en haleine au fur et à mesure des saisons. C'est grâce à vous si j'en suis là aujourd'hui et c'est également grâce à vous si je n'ai plus de doute sur le fait de continuer ma carrière en tant que développeur.

... **la promotion Nem** : une promotion formidable. Une équipe où l'entraide et la bienveillance étaient de mise. Sans oublier cette excellente ambiance que vous avez réussi à distiller.

Un GRAND merci à vous tous !



2. INTRODUCTION

Cette formation O'clock m'a donné l'opportunité d'acquérir de nombreuses compétences du métier de développeur web full-stack. Comprendre le fonctionnement d'une application web et arriver à en créer une grâce aux notions apprises est plutôt gratifiant et satisfaisant. Même si ma préférence va plutôt vers le frontend...

C'est pour cela que je me suis tourné vers une spécialisation React pour finaliser ma formation.

Ce projet associe React et NextJS pour le frontend et Symfony pour le backend. Ce projet d'équipe nous a permis de vivre une mise en situation professionnelle durant laquelle nous avons mis en pratique plusieurs mois d'apprentissage, et de passer du statut d'apprenant à celui de développeur web.



3. CONTEXTE DU PROJET

3.1. Présentation de l'application et du service

L'application O'Petits Oignons est conçue pour promouvoir la consommation locale en mettant en relation les consommateurs et les producteurs locaux. Ce service permet aux utilisateurs de commander en ligne des produits frais directement auprès des agriculteurs et autres producteurs, avec un retrait en mode **Click & Collect**. L'objectif est de simplifier l'accès aux produits locaux tout en soutenant les circuits courts, réduisant ainsi les intermédiaires entre les producteurs et les consommateurs.

3.2 . Objectifs

L'objectif principal de l'application est de créer une plateforme intuitive où les consommateurs peuvent facilement trouver et acheter des produits locaux parmi une liste de producteurs. En facilitant cette interaction directe, l'application vise à :

- Promouvoir les producteurs locaux en leur offrant une vitrine numérique où ils peuvent présenter et vendre leurs produits.
- Faciliter l'accès aux produits locaux en offrant un système de commande en ligne avec retrait rapide et pratique.
- Soutenir les circuits courts en réduisant les intermédiaires, ce qui permet de garantir des prix justes pour les producteurs et les consommateurs.

3.3 . Présentation de l'équipe

Le développement a été mené par une équipe pluridisciplinaire, répartie en deux pôles : une équipe **Back-end** et une équipe **Front-end**. Chaque membre a joué un rôle clé dans l'aboutissement du projet, apportant son expertise spécifique pour garantir la réussite de l'application.

👉 Equipe Back-end

Emmanuel Diop - Product Owner : En tant que Product Owner, Emmanuel a été le lien entre les besoins des utilisateurs et l'équipe de développement. Il a défini les fonctionnalités clés de l'application et a assuré la priorisation des tâches pour maximiser la valeur du produit final.

Sarika Ambinintsoa - Git Master Back : Responsable de la gestion des versions du code côté back-end, Sarika a veillé à ce que toutes les contributions soient intégrées de manière fluide et organisée, garantissant ainsi la stabilité du projet.

Michael Degorre - Scrum Master & Lead Dev Back : En tant que Scrum Master, Michael a coordonné les sprints de développement, tout en supervisant l'ensemble du développement back-end, en assurant l'intégration des différentes fonctionnalités et la gestion des bases de données.

👉 Equipe Front-end

David Vasseur - Lead Dev Front : David a dirigé le développement des interfaces utilisateur, en s'assurant que l'application soit réactive, ergonomique, et conforme aux exigences. Il a également veillé à l'intégration des composants front-end avec les API back-end.

Aymeric Martinache - Git Master Front : J'ai géré le versioning pour l'équipe front-end, en m'assurant que les différentes branches du projet étaient bien organisées et en corrigeant les éventuels conflits, permettant ainsi une livraison continue et de qualité.

3.4 . Méthodologie de travail

Le développement a été réalisé en suivant une **méthodologie agile**, plus spécifiquement en utilisant le cadre **Scrum**, pour assurer une gestion efficace du projet et une livraison continue des fonctionnalités.

Le projet a été divisé en **4 sprints**, chacun d'une durée d'une semaine, permettant une progression itérative du produit. Chaque sprint se composait des éléments suivants :

👉 Sprint 0 :

Mise en place du projet : Identification des besoins, définition des objectifs, et mise en place de l'environnement de développement.

User Stories : Établissement des **user stories** pour structurer le développement des fonctionnalités.

Création du MCD/MLD : Conception des modèles de données (**MCD** et **MLD**) pour structurer la base de données.

Wireframes et maquettes : Conception des **wireframes** et du **design graphique** de l'application.

👉 Sprint 1 :

Développement des premières fonctionnalités front-end : Mise en place des éléments essentiels comme le **header**, le **footer**, et les formulaires de connexion/inscription.

Mise en place de la base de données : Création des **entités** principales et initialisation de la **base de données**.

👉 Sprint 2 :

Connexion et sécurisation : Implémentation du système de login avec le **token JWT** et sécurisation des pages via des **middleware**.

Développement de l'API : Création et finalisation des **routes API** nécessaires à l'interaction front-back.

Pages utilisateur : Développement des pages profil utilisateur, commande, et produit.

👉 Sprint 3 :

Finalisation des fonctionnalités : Mise en place du back-office, gestion des commandes, et affichage des données utilisateurs.

Mise en production : Préparation de la version finale de l'application pour le déploiement.

3.5 . Versionning et gestion des branches

Pour assurer une bonne gestion des versions du projet, nous avons utilisé **Git** et **Github**. Cela nous a permis de structurer le développement en utilisant les branches suivantes :

- **Main** : Branche de production contenant le code stable.
- **Develop** : Branche principale de développement où les nouvelles fonctionnalités sont intégrées avant d'être testées et fusionnées dans la branche **Main**.
- **Feature branches** : Branches dédiées au développement de nouvelles fonctionnalités ou à la correction de bugs spécifiques.

3.6 . Rétrospectives et amélioration continue

Pour assurer une communication fluide et une coordination efficace entre les membres de l'équipe, des **daily meetings** étaient organisés. Ces réunions quotidiennes ont permis à chacun de partager les progrès réalisés, d'identifier les obstacles éventuels, et de réajuster les priorités si nécessaire.

À la fin de chaque sprint, une rétrospective était organisée pour discuter des succès et des axes d'amélioration. Cela nous a permis d'ajuster les priorités, de résoudre les problèmes rencontrés, et d'améliorer la collaboration au sein de l'équipe.



4. COMPÉTENCES COUVERTES PAR LE PROJET

4.1 Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- 👉 Installer et configurer son environnement de travail en fonction du projet web ou web mobile.

L'écriture du code a été faite via l'IDE **Visual Studio Code** de Microsoft.

Afin de produire un code plus propre et maintenable, le **linter ESLint** ainsi que le code formateur **Prettier** ont été utilisés.

Des procédures de versionning ont été mises en place avec **Git** et **Github**. Le développement a été fait sur une branche "Develop" avant d'être "mergé" sur la branche principale.

- 👉 Maquetter des interfaces utilisateurs

La création des **wireframes** et **maquettes** a été réalisée conjointement avec les différents membres de l'équipe. Nous avons échangé régulièrement afin de pouvoir fournir des documents cohérents. Ils ont été réalisés avec l'outil **Figma**.

Le **responsive design** a été défini lors de la réalisation des maquettes graphiques pour l'intégration.

Une charte graphique simple a également été mise en place afin d'harmoniser tous les éléments de l'application

Pour pouvoir mettre en place ces **wireframes**, les **user stories** de l'application ont été établies en prenant en compte le parcours des différents utilisateurs.

Le **MVP** (Minimum Viable Product) a été défini pour inclure les fonctionnalités essentielles telles que la gestion des utilisateurs, la recherche de producteurs, et la commande de produits. Cela a permis de focaliser les efforts de développement sur les éléments les plus critiques, en utilisant les wireframes et les user stories comme guides.

- 👉 Réaliser des interfaces utilisateur statiques

À partir des prototypes et de la charte graphique établie, les différentes interfaces utilisateurs ont été implémentées en suivant une approche "**mobile first**",

garantissant ainsi une expérience utilisateur optimale sur tous les types d'appareils, notamment mobiles et ordinateurs.

Afin de maximiser l'accessibilité et améliorer le référencement (**SEO**), nous avons utilisé les balises sémantiques **HTML** adéquates. (<header>, <nav>, <main>...).

Lors de la conception des interfaces utilisateur, une attention particulière a été portée à la sécurisation des champs de formulaire comme par exemple l'utilisation de champs de type "password" pour le mot de passe.

Ces balises permettent aux moteurs de recherche de mieux comprendre la structure de la page et contribue également à une meilleure compatibilité avec les lecteurs d'écran, améliorant ainsi l'accessibilité.

Grâce à **NextJS**, nous avons pu tirer parti du **Server-Side Rendering (SSR)**, qui permet de générer des pages **HTML** côté serveur. Cela a un impact direct sur le **SEO**, car les moteurs de recherche peuvent facilement indexer le contenu.

Nous avons utilisé **Tailwind CSS** pour la gestion du style, ce qui nous a permis de créer des interfaces réactives. **Tailwind** offre des classes prédéfinies qui facilitent l'application directe des styles, sans avoir à écrire du **CSS** personnalisé pour chaque composant.

Les MediaQueries, indispensables pour le responsive design, sont gérées directement par **Tailwind**, rendant l'adaptation du contenu aux différentes tailles d'écrans fluide et intuitive. Cette approche a permis de prioriser le "**mobile first**", en garantissant une expérience utilisateur cohérente et optimisée, indépendamment de l'appareil utilisé.

👉 Développer la partie dynamique des interfaces utilisateur

Pour ajouter des fonctionnalités dynamiques aux interfaces utilisateur de l'application, nous avons utilisé **React** en combinaison avec **NextJS**. Cela a permis d'améliorer l'interaction utilisateur en rendant l'application plus réactive et interactive.

Les formulaires d'inscription et de connexion ont été développés avec **Formik** pour gérer l'état des champs et **Yup** pour la validation en temps réel des données saisies par l'utilisateur. Par exemple, lors de la saisie du mot de passe, le champ est automatiquement validé et les erreurs sont affichées instantanément, améliorant ainsi l'expérience utilisateur.

Nous avons également intégré des fonctionnalités telles que la **géolocalisation** pour afficher les producteurs locaux sur une carte interactive, en utilisant **React-Leaflet**. Ces fonctionnalités dynamiques sont essentielles pour offrir une expérience utilisateur fluide, où les données sont mises à jour en temps réel en fonction des interactions de l'utilisateur.

Le **state management** a été assuré par **Redux Toolkit**, avec un **store** centralisé qui gère les données utilisateur, les produits, et les commandes. Cette gestion centralisée permet de synchroniser les données entre les différentes parties de l'application, garantissant une expérience cohérente.

4.2 Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

👉 Mettre en place une base de données relationnelle

La base de données relationnelle de "O'Petits Oignons" a été conçue en utilisant **Doctrine ORM**. Le Modèle Conceptuel de Données (**MCD**) et le Modèle Logique de Données (**MLD**) ont été élaborés pour structurer les entités principales de l'application, telles que les utilisateurs, les producteurs, les produits, et les commandes.

Les entités sont définies dans le répertoire `src/Entity`, et chaque entité est reliée par des relations bien définies (one-to-many, many-to-many). Par exemple, une entité **Product** est associée à plusieurs **OrderLine**, permettant de suivre les produits dans les différentes commandes.

Les migrations de base de données, gérées via **Doctrine Migrations**, assurent que toutes les modifications de schéma sont appliquées de manière cohérente et sécurisée dans tous les environnements. Cela garantit que la base de données reste "alignée" avec les besoins du projet au fur et à mesure de son évolution.

Le **dictionnaire de données** documente chaque table et chaque champ de la base de données, offrant une référence claire pour l'équipe de développement et facilitant la maintenance future du projet.

Le **dictionnaire de données** est disponible en annexe.

👉 Développer des composants d'accès aux données SQL

Pour permettre l'interaction avec la base de données SQL, des composants d'accès aux données ont été développés en utilisant **Doctrine ORM**. Chaque entité, comme **User**, **Product**, et **Order**, dispose d'un repository dédié dans le répertoire `src/Repository`.

Ces repositories contiennent les méthodes nécessaires pour exécuter des opérations CRUD (Create, Read, Update, Delete) sur les données. Par exemple, **UserRepository.php** permet de récupérer les informations utilisateur, de vérifier leur existence lors de la connexion, et de mettre à jour leurs données.

Les **requêtes SQL** sont optimisées pour garantir des performances élevées, même avec un grand nombre d'utilisateurs ou de produits. De plus, des transactions sont utilisées pour assurer que les opérations complexes, comme le traitement des commandes, se déroulent de manière atomique, évitant ainsi les incohérences dans la base de données.

Bien que ce projet soit principalement basé sur une base de données SQL, l'architecture est conçue pour être extensible et pourrait intégrer des bases NoSQL pour des besoins spécifiques, comme la gestion de grandes quantités de données non structurées.

👉 Développer des composants métier côté serveur

Les composants métier ont été développés avec **Symfony** pour gérer la logique d'application côté serveur. Cela inclut la gestion des utilisateurs, l'authentification, la création des commandes, et la gestion des produits.

Les **APIs RESTful**, implémentées dans [src/Controller](#), permettent une communication fluide entre le front-end et le back-end. Chaque API est sécurisée avec des tokens JWT, implémentés via le **LexikJWTAuthenticationBundle**, pour garantir que seules les requêtes authentifiées peuvent accéder aux données sensibles.

Les contrôleurs utilisent ces tokens pour vérifier les permissions, et chaque **endpoint** est protégé contre les accès non autorisés. Par exemple, les opérations sur les commandes ne sont accessibles qu'aux utilisateurs ayant les droits appropriés, garantissant ainsi la sécurité des transactions.

Les **services** côté serveur, situés dans [src/Service](#), gèrent des tâches spécifiques comme la génération de numéros de commande uniques, assurant ainsi la cohérence des données tout au long du cycle de vie de l'application.

5. CAHIER DES CHARGES

Le projet O'Petits Oignons a été conçu pour répondre aux besoins des consommateurs souhaitant accéder à des produits locaux et des producteurs souhaitant vendre leurs produits en ligne. Ce projet a nécessité une architecture solide, tant au niveau du front-end que du back-end, pour garantir une expérience utilisateur optimale, sécurisée, et performante.

5.1 . MVP (Minimum Vitale Production)

Le MVP du projet représente la version minimale du produit qui peut être lancée pour obtenir un retour des utilisateurs tout en offrant des fonctionnalités essentielles. Il inclut :

Authentification et gestion des comptes:

Inscription / connexion : Les utilisateurs peuvent créer un compte ou se connecter à un compte existant. Ils peuvent choisir entre un compte consommateur ou un compte producteur.

Edition de profil : Les utilisateurs peuvent gérer leurs informations personnelles.

Recherche et navigation:

Consommateur : géolocalisation des producteurs grâce à une carte interactive. Une liste des producteurs est disponible avec des détails comme les produits disponibles.

Gestion producteur:

Les **producteurs** peuvent ajouter des produits.

Informations détaillées sur le producteur (présentation, liste des produits).

Suivi des commandes : Les producteurs peuvent voir et gérer les commandes reçues.

Interface consommateur

• **Disponible sans inscription:**

Voir la fiche de présentation du producteur. Voir la liste des produits disponibles avec descriptions, prix, unité de vente et disponibilité (en stock ou pas en stock).

- **Uniquement si un compte consommateur est créé:**

Pouvoir commander des produits et voir ses commandes en cours et passées

👉 Panier et Commande

Panier : Les utilisateurs peuvent ajouter des produits au panier, modifier les quantités, et supprimer des produits. Le panier est temporaire et associé à la session de l'utilisateur. Si l'utilisateur n'a pas de compte, il doit s'inscrire pour valider son panier.

Commande : Processus de commande simple. Le consommateur envoie la commande et la récupère chez le producteur quand elle est prête.

👉 Administration du site

Création d'un **back-office** pour les administrateurs du site.

5.2 . Workflow

Le **workflow** décrit la manière dont les utilisateurs interagissent avec l'application, de l'inscription à la gestion des commandes.

👉 Inscription/Connexion

L'utilisateur choisit son rôle (consommateur ou producteur) et s'inscrit via un formulaire.

Après l'inscription, l'utilisateur peut se connecter pour accéder à son profil.

👉 Recherche de producteurs et produits (pour les consommateurs)

Le consommateur effectue une recherche pour trouver des produits. Il peut consulter les fiches des producteurs et voir les produits disponibles.

👉 Gestion des produits (pour les producteurs) :

Le producteur ajoute les produits qu'il souhaite vendre. Il peut également modifier la disponibilité de ses produits.

👉 Ajout au panier et gestion des commandes :

Le consommateur ajoute des produits à son panier. Il peut consulter le panier, ajuster les quantités, et passer la commande.

Le producteur reçoit la commande, la traite, et met à jour le statut (en préparation, prête, terminée).

👉 Gestion des commandes (pour les consommateurs) :

Le consommateur peut consulter l'état de ses commandes en cours et son historique d'achat.

5.3 . User stories

Les **user stories** représentent les besoins des utilisateurs en termes de fonctionnalités. Voici quelques exemples :

- **En tant que consommateur**, je veux pouvoir rechercher des producteurs locaux afin de découvrir les produits disponibles dans ma région.
- **En tant que consommateur**, je veux ajouter des produits à mon panier et consulter ce dernier pour vérifier les produits que j'ai sélectionnés avant de passer commande.
- **En tant que producteur**, je veux pouvoir gérer mes produits (ajout, modification, suppression) pour tenir à jour mon catalogue en ligne.
- **En tant que producteur**, je veux consulter et gérer les commandes reçues pour assurer le bon déroulement des ventes.
- **En tant qu'administrateur**, je veux accéder à un backoffice pour gérer les produits, les commandes et les utilisateurs afin de maintenir la qualité du service.

La liste des **user stories** est disponible en annexe.

5.4 . Les routes

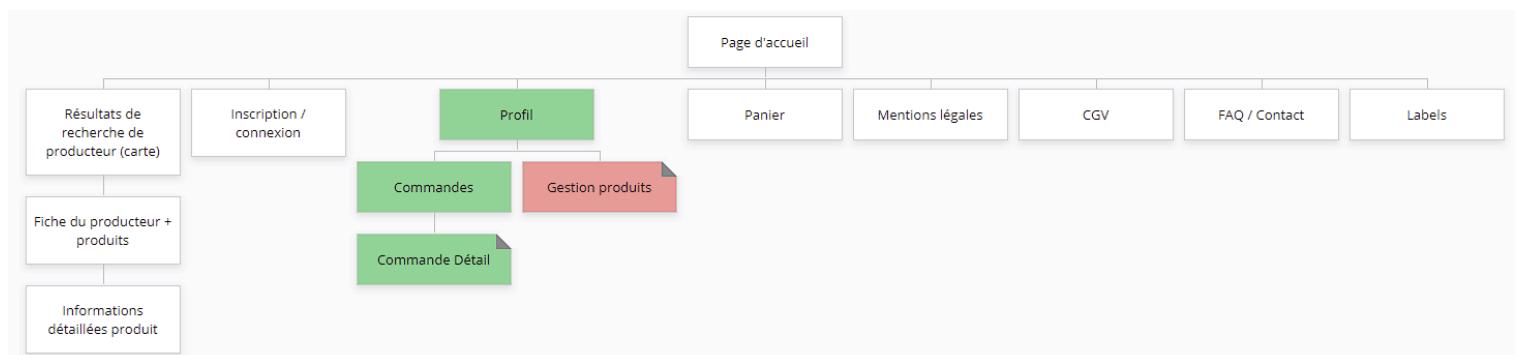
Les routes définissent les chemins d'accès pour les différentes fonctionnalités de l'application.

Voici un aperçu des principales routes :

- Front-end (consommateurs) :
 - `/` : Page d'accueil.
 - `/result` : Recherche de producteurs.
 - `/producer/{id}` : Détails du producteur et liste des produits.
 - `/product/{id}` : Détails d'un produit spécifique.
 - `/cart` : Panier.
 - `/profile` : Gestion du profil utilisateur.
- Front-end (producteurs) :
 - `/profile` : Tableau de bord du producteur.
 - `/product/add` : Ajout d'un nouveau produit.
- Back-office (administrateurs) :
 - `/backoffice/orders` : Gestion des commandes.
 - `/backoffice/users` : Gestion des utilisateurs.

5.5 . Arborescence

L'arborescence de l'application est structurée pour permettre une navigation intuitive et organisée.



👉 Page d'accueil

La page d'accueil sert de point de départ pour les utilisateurs et offre un accès rapide aux différentes sections du site.

Recherche de producteurs et produits

- Résultats de recherche de producteurs (carte) :
 - Présente une carte interactive permettant aux utilisateurs de rechercher des producteurs locaux.
- Fiche du producteur + produits :
 - Affiche les détails d'un producteur ainsi que les produits qu'il propose.
- Informations détaillées produit :
 - Fournit des informations complètes sur un produit spécifique sélectionné par l'utilisateur.

Inscription / Connexion

- Inscription / Connexion :
 - Permet aux utilisateurs de s'inscrire sur la plateforme ou de se connecter pour accéder à leur compte.

Profil Utilisateur

- Profil :
 - Donne accès aux informations personnelles de l'utilisateur et à la gestion de son compte.
- Commandes :
 - Liste les commandes passées par l'utilisateur, avec la possibilité de consulter les détails de chaque commande.
 - Commande Détail : Affiche les détails spécifiques d'une commande sélectionnée.
- Gestion produits (pour les producteurs) :
 - Permet aux producteurs de gérer leurs produits.

Panier

- Affiche les produits que l'utilisateur a ajoutés à son panier avec la possibilité de finaliser la commande.

Informations légales et support

- Mentions légales :

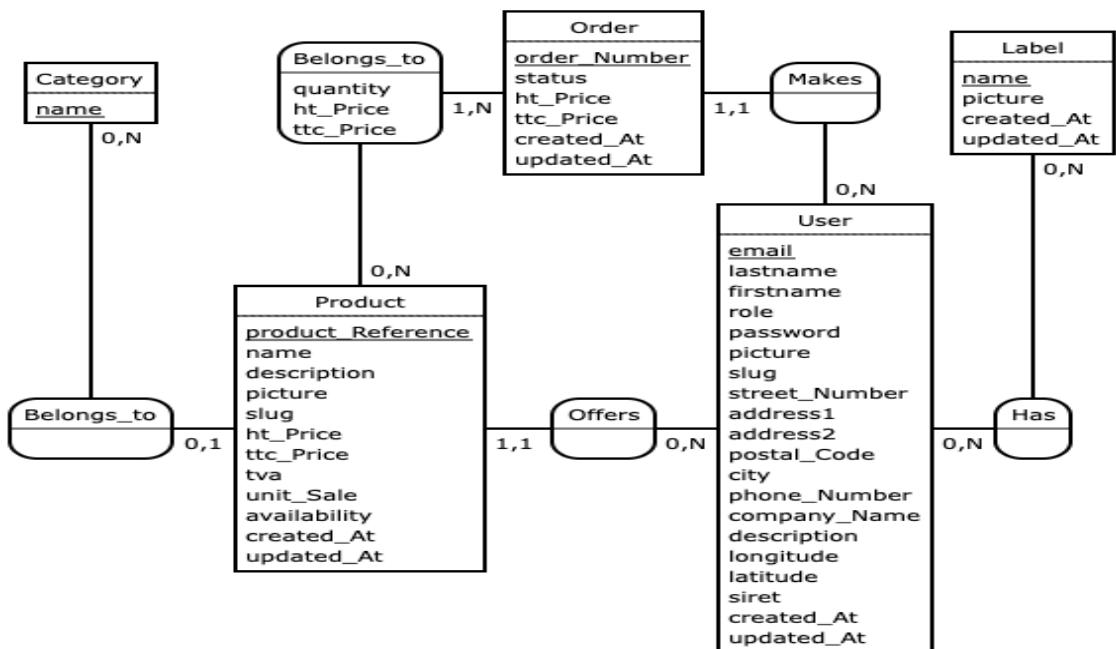
- Présente les informations légales liées à l'utilisation de la plateforme.
- **CGV (Conditions Générales de Vente) :**
 - Liste les conditions générales de vente auxquelles sont soumis les utilisateurs.
- **FAQ / Contact :**
 - Fournit des réponses aux questions fréquemment posées et offre un moyen de contact pour les utilisateurs.
- **Labels :**
 - Explique les différents labels qui peuvent être associés aux producteurs disponibles sur la plateforme.

5.6 . MCD (Modèle Conceptuel de Données)

Le **MCD** de l'application est un schéma représentant les différentes entités de la base de données ainsi que leurs relations. Les principales entités sont :

- **User (Utilisateur)** : Contient les informations des consommateurs et des producteurs (nom, prénom, email, rôle, etc.).
- **Product (Produit)** : Regroupe les produits mis en vente par les producteurs (nom, description, prix HT, prix TTC, TVA, disponibilité, etc.).
- **Order (Commande)** : Enregistre les commandes passées par les consommateurs, y compris le statut de la commande.
- **OrderLine (Ligne de commande)** : Détaille les produits et quantités associés à chaque commande.
- **Label (Label)** : Associe des labels aux producteurs pour indiquer des certifications ou des qualités spécifiques.

Chaque entité est reliée par des associations clairement définies, comme la relation entre un producteur et ses produits, ou une commande et ses lignes de commande. Le **MCD** assure que toutes les données nécessaires au bon fonctionnement de l'application soient correctement organisées et accessibles.



5.7 . Dictionnaire de données

Le **dictionnaire de données** décrit en détail les champs, types de données, contraintes et relations entre les différentes tables de la base de données. Ce dictionnaire sert de référence pour assurer l'intégrité et la cohérence des données tout au long du cycle de vie de l'application.

Exemple de définitions de données :

Utilisateur

Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id de l'utilisateur
<code>lastname</code>	VARCHAR(64)	NOT NULL	Le nom de l'utilisateur
<code>firstname</code>	VARCHAR(64)	NOT NULL	Le prénom de l'utilisateur
<code>email</code>	VARCHAR(64)	UNIQUE, NOT NULL	L'email de l'utilisateur

Produit

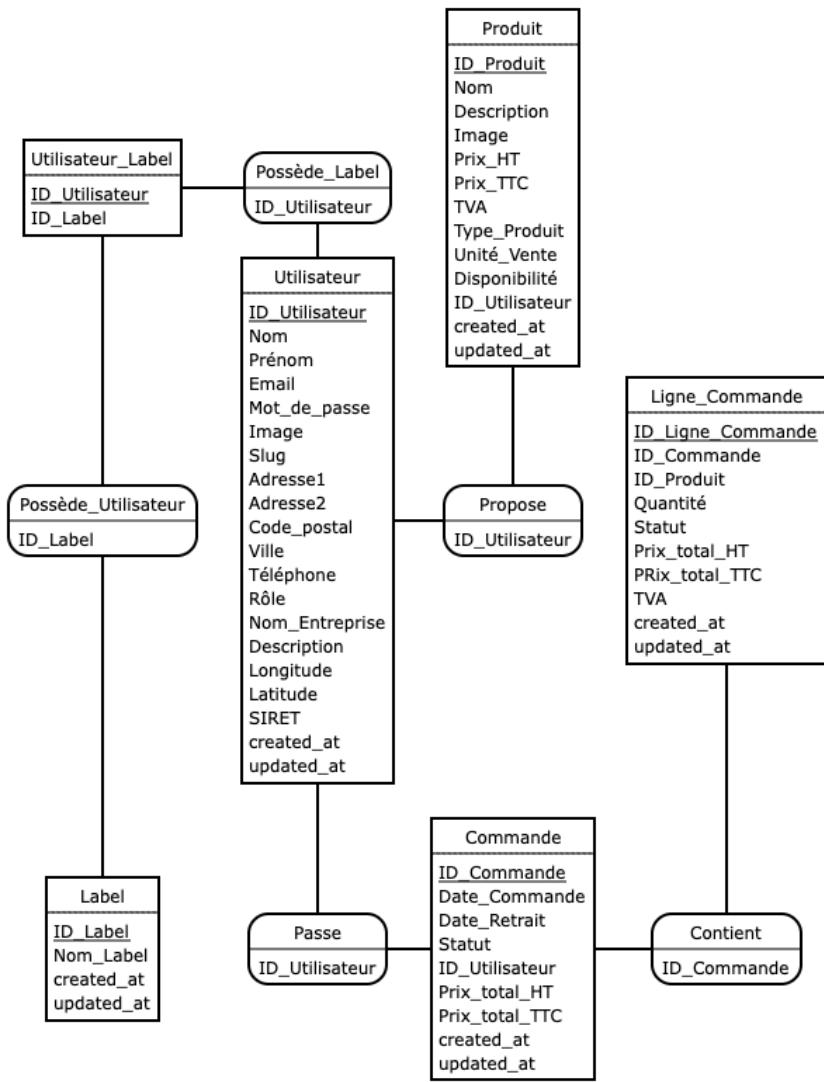
Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id du produit
<code>id_utilisateur</code>	INT	FOREIGN KEY	L'id du producteur
<code>name</code>	VARCHAR(64)	NOT NULL	Le nom du produit
<code>description</code>	MEDIUMTEXT	NOT NULL	La description du produit
<code>picture</code>	VARCHAR(255)	NULL	L'image du produit remplacée par une image générique si le producteur n'en mets pas

5.8 . MLD (Modèle Logique de Données)

Le Modèle Logique de Données (**MLD**) formalise les entités et leurs relations, ainsi que les contraintes d'intégrité applicables aux données de l'application. Le **MLD** est une version détaillée et structurée du Modèle Conceptuel de Données (**MCD**), et sert de base pour la création du schéma de base de données physique.

Principales entités et relations :

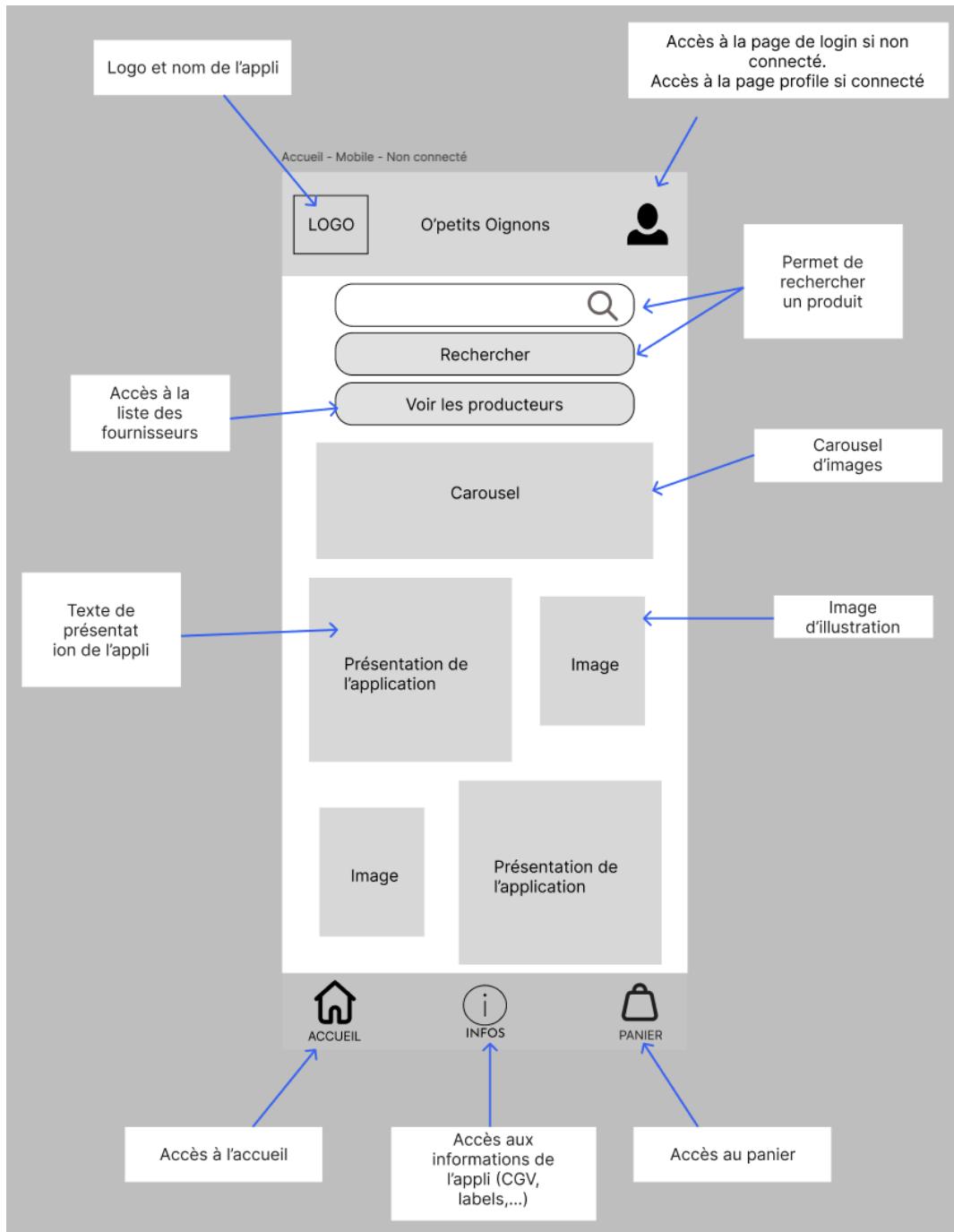
- **User** :
 - Relation **1,N** avec **Order** (un utilisateur peut passer plusieurs commandes).
 - Relation **1,N** avec **Product** (un producteur peut proposer plusieurs produits).
- **Product** :
 - Relation **1,N** avec **OrderLine** (un produit peut apparaître dans plusieurs lignes de commande).
- **Order** :
 - Relation **1,N** avec **OrderLine** (une commande contient plusieurs lignes de commande).
 - Relation **N,1** avec **User** (une commande est associée à un utilisateur).

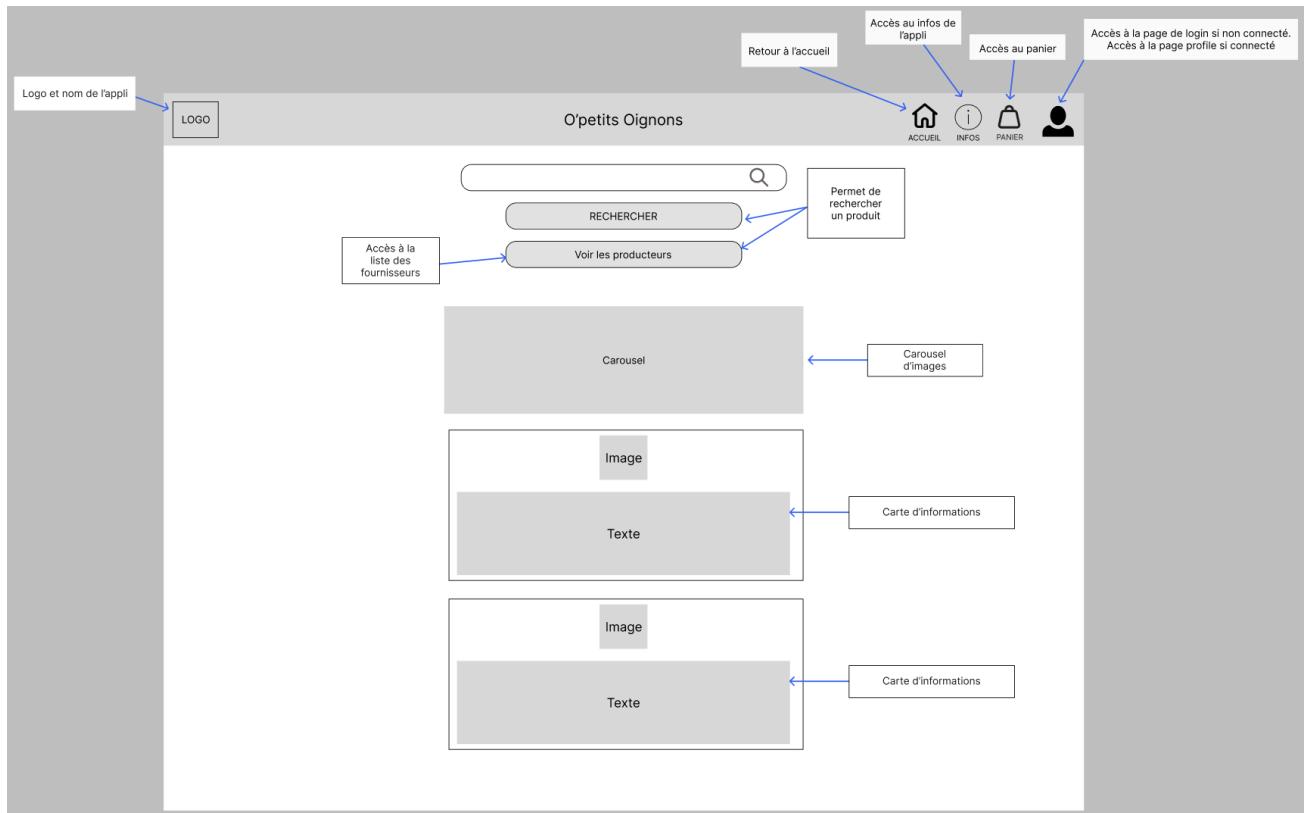


5.9 . Wireframes

Les **wireframes** sont des schémas simplifiés représentant la structure des différentes pages de l'application. Ils définissent l'emplacement des éléments principaux (menus, boutons, champs de formulaire, etc.) et servent de guide pour le développement des interfaces utilisateur.

Exemple de pages:





5.10 . Charte graphique

La **charte graphique** définit l'identité visuelle de la plateforme. Elle inclut les choix de couleurs, de typographies, de styles de boutons, ainsi que les règles d'utilisation des logos et des images. Cette charte garantit une cohérence visuelle à travers toutes les interfaces de l'application.

Exemple pour les couleurs:

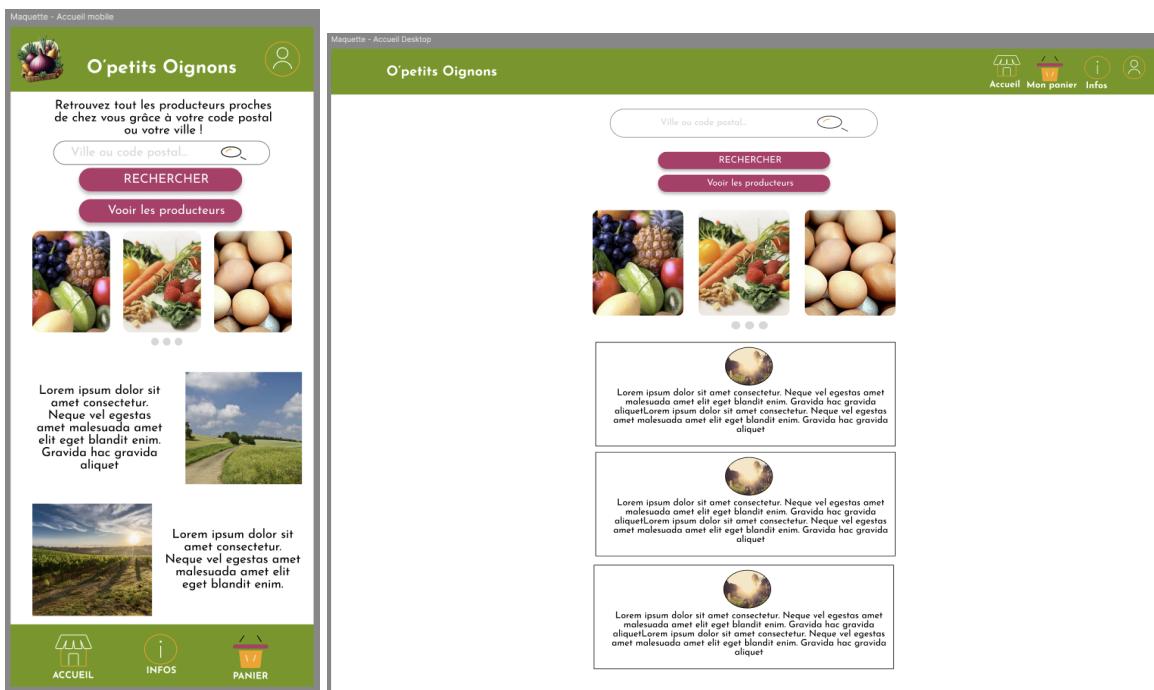
Couleurs	
Principale Active : #709a0b	
Principale Inactive : #91A16A	
Secondaire Active : #EA6F0E	
Secondaire Inactive : #EEAC78	
Purple Active : #BB2F6B	
Purple Inactive : #BD6F90	
Background : #FEF4EC	

5.11 . Maquettes

Les maquettes représentent les interfaces finales de l'application, telles qu'elles seront perçues par les utilisateurs.

Elles sont le fruit d'une itération sur les wireframes et intègrent la charte graphique de manière complète. Les maquettes permettent de valider l'ergonomie et l'esthétique de chaque page avant leur développement.

Exemples de maquettes :



Ces éléments fournissent une vue complète et structurée des aspects techniques et visuels du projet, facilitant ainsi le développement, la maintenance, et l'évolution future de l'application.



6. SPECIFICATIONS TECHNIQUES

6.1 . Front-end

Le front-office de l'application est conçu pour offrir une expérience utilisateur optimale grâce à une architecture moderne, utilisant des technologies telles que **Next.js**, **React**, **Formik**, **Yup**, **Leaflet**, et **Redux Toolkit**.

Voici une description des technologies utilisées et des fonctionnalités implémentées pour garantir la performance, la réactivité et la sécurité de l'application.

👉 Technologies Utilisées

- **Next.js**

Next.js est utilisé comme framework principal pour le développement du front-end. Il fournit un rendu côté serveur (**SSR**), une génération statique de pages (**SSG**) et une pré-récupération des données, ce qui améliore considérablement les performances, l'optimisation **SEO** et le temps de chargement des pages.

Grâce à son système de routage intégré, il permet de gérer facilement les routes dynamiques pour les pages, offrant ainsi une navigation fluide et rapide.

- **React**

React, la bibliothèque **JavaScript** sous-jacente à **Next.js**, est utilisée pour construire les interfaces utilisateur dynamiques et réactives. Les composants **React** permettent de créer des vues modulaires, réutilisables et maintenables, ce qui simplifie le développement et l'évolution de l'application.

L'état local (**state**) des composants est géré efficacement, et des **hooks React** sont utilisés pour gérer les cycles de vie, les effets secondaires et la logique complexe.

- **TypeScript**

TypeScript est utilisé pour ajouter des types statiques au code **JavaScript**, ce qui améliore la qualité du code, aide à prévenir les erreurs courantes et facilite la maintenance du code à long terme.

L'utilisation de **TypeScript** permet de définir clairement les types de données utilisés dans l'application, tels que les interfaces pour les utilisateurs, les commandes, les produits, etc. Cela garantit une meilleure compréhension du code et réduit les erreurs lors du développement.

- **Formik et Yup**

Formik est utilisé pour la gestion des formulaires, simplifiant la gestion de l'état des formulaires, la validation, et la gestion des erreurs. Il permet de gérer des formulaires complexes de manière efficace tout en restant facile à utiliser.

```
● ● ●  
1  <Formik  
2      initialValues={user as IUser}  
3      validationSchema={validationSchema}  
4      enableReinitialize  
5      onSubmit={handleSubmit}  
6    >  
7    <Form>  
8      /* Email */  
9      <div className="flex flex-col gap-2 justify-between mb-4">  
10         <label htmlFor="email">Email : </label>  
11         <div className="flex items-center">  
12             <Field  
13                 className="rounded-[1rem] p-2 w-full mb-3 bg-gray-300"  
14                 name="email"  
15                 readOnly={isReadOnly.email}  
16                 placeholder={user?.email}  
17             />  
18         </div>  
19     </div>
```

Yup est utilisé en combinaison avec **Formik** pour la validation des formulaires. Il permet de définir des schémas de validation robustes, garantissant que les données saisies par l'utilisateur respectent les critères requis avant d'être soumises.

```
● ● ●  
1  // Schema de validation de Formik  
2  const validationSchema = Yup.object({  
3      email: Yup.string()  
4          .required('L'email est requis')  
5          .email("Veuillez saisir un email valide"),  
6      companyName: Yup.string().required(  
7          "Merci d'indiquer le nom de votre entreprise"  
8      ),  
9      lastName: Yup.string().required("Merci d'indiquer votre nom"),  
10     firstName: Yup.string().required("Merci d'indiquer votre prénom"),  
11     streetNumber: Yup.number().required(  
12         "Merci d'indiquer votre numéro de voie"  
13     ),
```

Ces outils sont essentiels pour des formulaires critiques tels que l'inscription, la connexion, ou la gestion du profil.

- **Leaflet** :

Leaflet est une bibliothèque JavaScript utilisée pour l'intégration de cartes interactives. Elle permet aux utilisateurs de visualiser les producteurs locaux sur une carte et de rendre l'expérience utilisateur plus interactive.

L'intégration de **Leaflet** avec **React** permet de créer des composants de carte réactifs qui s'adaptent dynamiquement aux changements d'état.



```
1  <MapContainer
2    center={[userLocation.latitude, userLocation.longitude]}
3    zoom={12}
4    className="h-[300px] w-full md:h-[500px] rounded-xl border-2 border-secondary_active shadow-md z-0"
5  >
6    <RecenterMap
7      lat={userLocation.latitude}
8      lng={userLocation.longitude}
9    />
10
11   <TileLayer
12     attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
13     url="https://s.tile.openstreetmap.org/{z}/{x}/{y}.png"
14   />
15   {users &&
16     Array.isArray(users) &&
17     users.map((user) => (
18       <Marker
19         key={user.id}
20         position={[user.latitude, user.longitude]}
21         icon={producerIcon}
22     >
```

- **API Imgbb** :

Pour la gestion des images, l'application utilise l'**API Imgbb**, qui permet aux utilisateurs d'uploader des images qui sont ensuite hébergées de manière sécurisée sur un serveur externe. Cette approche simplifie la gestion des fichiers tout en assurant une intégration facile avec l'interface utilisateur.

Exemple sur l'ajout d'une image sur un produit :

Lors du submit du formulaire, l'image est envoyée et on récupère l'url pour l'insérer dans la base de données.

```

1 // Fetch de l'upload d'une image sur ImgBB
2 export async function fetchImgbb(values: ISignUser | IProducts) {
3     let imageUrl = "";
4
5     if (values.picture !== "") {
6         const formData = new FormData();
7         formData.append("image", values.picture.substring(22));
8
9         try {
10             const responseImage = await fetch(
11                 "https://api.imgur.com/1/upload?key=e494e2154a4621708abb7ef9e65b07cc",
12                 {
13                     method: "POST",
14                     body: formData,
15                 }
16             );
17
18             if (responseImage.ok) {
19                 const dataImage = await responseImage.json();
20                 imageUrl = dataImage.data.url;
21                 console.log(imageUrl);
22             } else {
23                 console.log("erreur");
24             }
25         } catch (error) {
26             console.log(error);
27         }
28     }
29     return imageUrl;
30 }
31

```

```

1 const handleSubmit = async (values: { addProduct: IProducts[] }) => {
2     console.log("fetch en cours");
3
4     const imageUrlArray: string[] = [];
5
6     for (const product of values.addProduct) {
7         const imageUrl: string = await fetchImgbb(product);
8         imageUrlArray.push(imageUrl);
9     }
10
11     const newForm = values.addProduct.map((product, index) => {
12         return {
13             ...product,
14             id_user: userId,
15             availability: 1,
16             picture: imageUrlArray[index],
17             category: "",
18         };
19     });
20     console.log(newForm);
21
22     if (token) {
23         await fetchAddProduct(newForm, token);
24     }

```

👉 Appels API

Les appels **API** sont essentiels pour la communication entre le **front-end** et le **back-end** de l'application. Ils permettent de récupérer des données dynamiques, de soumettre des formulaires, et d'effectuer diverses actions telles que l'authentification, la gestion des produits, et le traitement des commandes.

- **Gestion des Appels API**

Les appels API sont effectués principalement via des requêtes **fetch** intégrées dans des composants **React**. Ces requêtes permettent d'interagir avec les **endpoints** définis dans le **back-end Symfony**.

Les appels API sont **asynchrones**, utilisant `async/await` pour gérer les promesses et assurer un traitement fluide des données, tout en maintenant l'application réactive.

👉 Exemples d'appels API :

- **Authentification**

Un appel API est effectué lors de la soumission du formulaire de connexion. L'utilisateur saisit ses identifiants, et une requête **POST** est envoyée pour vérifier les informations et récupérer un **token de connexion**.

```
● ● ●
1 // Fetch du login
2 export async function fetchLogin(log: ILoginUser): Promise<LoginResponse | undefined> {
3     try {
4         const response = await fetch("https://opeditoignons.io/api/v1/login", {
5             method: "POST",
6             headers: {
7                 "Content-Type": "application/json",
8             },
9             credentials: "include",
10            body: JSON.stringify(log),
11        });
12        const data = await response.json();
13        if (response.ok) {
14            const loginSuccess: LoginSuccess = {
15                token: data.token,
16                role: data.data.roles[0],
17                userId: data.data.user,
18            };
19            return loginSuccess;
20        } else if (!response.ok) {
21            const loginFailure: LoginFailure = { message: data.message };
22            return loginFailure;
23        }
24    } catch (error) {
25        const loginFailure: LoginFailure = { message: 'An error occurred while logging in.' };
26        return loginFailure;
27    }
}
```

- **Gestion des Commandes :**

Lors de la validation du panier, un appel API est effectué pour créer une nouvelle commande. Cette requête **POST** envoie les données du panier au **back-end** pour qu'elles soient traitées et enregistrées dans la base de données.

```
1 const handleSubmit = async () => {
2     // console.log("click");
3
4     if (token && total !== 0) {
5         const newOrder = items.map((item: any) => ({
6             quantity: item.qty,
7             productId: item.id,
8         }));
9
10        try {
11            const response = await fetch(
12                "https://opetitsoignons.io/api/v1/order/",
13                {
14                    method: "POST",
15                    headers: {
16                        Authorization: `Bearer ${token}`,
17                    },
18                    credentials: "include",
19                    body: JSON.stringify(newOrder),
20                }
21            );
22
23            if (response.ok) {
24                const data = await response.json();
25                console.log(data);
26                localStorage.removeItem("cart");
27                dispatch(resetCart());
28                dispatch(
29                    setModal({
30                        isOpen: true,
31                        content: {
32                            title: "Panier validé !",
33                            message:
34                                "Votre commande à bien été transmise au producteur.",
35                        },
36                        color: "text-primary_active",
37                        button: true,
38                    })
39                );
39            );
40        } catch (error) {
41            console.error(error);
42        }
43    }
44}
```

👉 Gestion de l'État et Middleware

- **Redux Toolkit :**

Redux Toolkit est utilisé pour la gestion globale du **state** de l'application. Il permet de centraliser le **state**, garantissant que toutes les parties de l'application accèdent aux données de manière cohérente.

Cela est crucial pour une application complexe où plusieurs composants doivent partager des données comme le panier ou l'état de connexion utilisateur.

- **Slices**

Les **slices** de **Redux Toolkit** sont utilisés pour organiser le **state** de l'application. Chaque slice gère un domaine spécifique, comme **CartSlice** pour le panier, **UserSlice** pour l'utilisateur, et **ModalSlice** pour les modals. Cela permet une gestion modulaire et maintenable du **state**.



```
1 const initialState: IOrder[] = [];
2
3
4 const OrderSlice = createSlice(
5   {
6     name: 'order',
7     initialState,
8     reducers: {
9       addOrder: ((state, action) => {
10         return action.payload;
11       })
12     }
13   }
```



```
1 const userSlice = createSlice({
2   name: 'user',
3   initialState,
4   reducers: {
5     setUser: (state, action) => {
6       state.token = action.payload.token;
7       state.userId = action.payload.userId;
8       state.role = action.payload.role;
9       state.picture = action.payload.picture;
10    },
11    clearUser: (state) => {
12      state.token = null;
13      state.userId = null;
14      state.role = null;
15      state.picture = null;
16    },
17  },
```

- **Middleware :**

Un **middleware** personnalisé pour **Redux** est utilisé pour gérer le **localStorage** afin de garder le panier entre les sessions utilisateur.

Ce middleware intercepte les actions liées au panier (**addToCart**, **removeFromCart**, **updateItemQuantity**) et met à jour le **localStorage** avec l'état actuel du panier.



```
1 import { Middleware } from '@reduxjs/toolkit';
2 import { addToCart, removeFromCart, updateItemQuantity } from '../feature/CartSlice';
3
4 const localStorageMiddleware: Middleware = storeAPI => next => action => {
5   const result = next(action);
6
7   // Interceptor les actions du panier
8   if (addToCart.match(action) || removeFromCart.match(action) || updateItemQuantity.match(action)) {
9     const state = storeAPI.getState();
10    localStorage.setItem('cart', JSON.stringify(state.cart));
11  }
12
13  return result;
14 };
15
```

Ce **middleware** garantit que les articles du panier sont sauvegardés dans le **localStorage** à chaque modification, assurant que les données ne sont pas perdues lors d'une actualisation de la page ou d'une nouvelle session.

👉 Gestion des Tokens et Sécurité

- **Session Storage et Tokens de Connexion :**

L'application utilise des **tokens** pour authentifier les utilisateurs. Lors de la connexion, le **token** reçu est stocké dans le **sessionStorage** afin de sécuriser l'accès aux ressources protégées de l'application, telles que les pages de profil ou la gestion des commandes.

Le code ci-dessous montre la soumission du formulaire, puis le stockage du token reçu après une connexion réussie.

```
 1 // Submit du form
 2 const handleSubmit = async (values: ILoginUser) => {
 3   const connect = {
 4     username: values.username,
 5     password: values.password,
 6   };
 7
 8   const connection = await fetchLogin(connect);
 9
10   if (connection && Object.keys(connection).length < 2) {
11     const message = connection
12     dispatch(setModal({
13       isOpen: true,
14       content: {
15         title: "ATTENTION",
16         message,
17       },
18       color: "text-red-400",
19       button: true
20     }))
21   }
22
23   else if (isLoginSuccess(connection) ) {
24     const loginSuccess = connection;
25     const newUser = {
26       token: loginSuccess.token,
27       userId: loginSuccess.userId,
28       role: loginSuccess.role,
29       picture: null,
30     };
31     dispatch(setUser(newUser));
32     router.push("/profil");
33   }
}
```

👉 Schémas de Validation

- **Validation des Formulaires :**

Les formulaires comme l'inscription, la connexion, et la gestion du profil utilisent des schémas de validation définis avec **Yup**. Cela garantit que les données saisies sont conformes aux attentes, ce qui réduit les erreurs et améliore la sécurité.

Par exemple, lors de la validation d'un formulaire de connexion, **Yup** vérifie que l'adresse email est valide et que le mot de passe n'est pas vide, fournissant des messages d'erreur adaptés en cas de non-conformité.



```
1 // Schéma de validation
2 const validationSchema = Yup.object({
3   username: Yup.string()
4     .required(`L'email est requis`)
5     .email("Veuillez saisir un email valide"),
6   password: Yup.string().required("Le mot de passe est requis"),
7 });
```

👉 Optimisation et Performance

- **Lazy Loading et Pre-rendering :**

Le **Lazy loading** est utilisé pour retarder le chargement des images et des composants non critiques, ce qui améliore les temps de chargement initial de l'application.

Grâce à **Next.js**, certaines pages sont pré-rendues au moment du build, permettant un chargement instantané lors de la première visite et améliorant les performances globales de l'application.

- **SEO :**

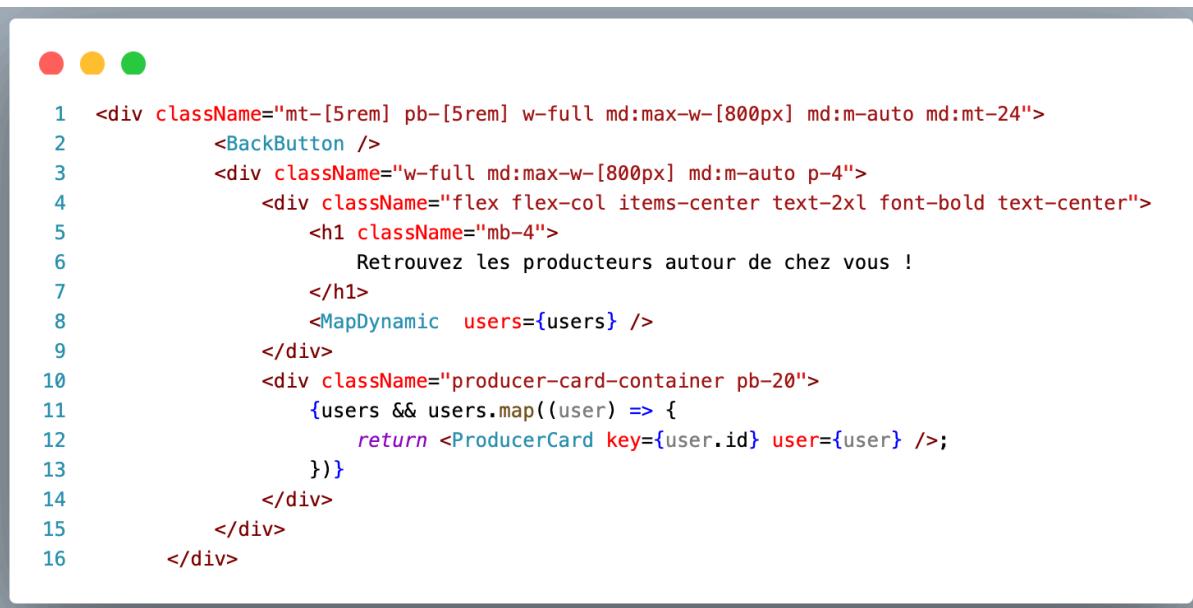
Le rendu côté serveur (**SSR**) et la génération statique de pages (**SSG**) de **Next.js** permettent d'optimiser les pages pour les moteurs de recherche, garantissant une meilleure visibilité et accessibilité de l'application.

👉 Responsive Design

- Tailwind CSS :

Tailwind CSS est utilisé pour le style de l'application. Il s'agit d'un **framework CSS** utilitaire qui permet de créer des interfaces élégantes et réactives sans écrire de CSS personnalisé.

Grâce à Tailwind, nous pouvons appliquer des classes directement dans les composants pour définir la disposition, la typographie, les couleurs, et d'autres styles. Cela permet de gagner du temps et d'assurer une cohérence visuelle à travers toute l'application.



```
1 <div className="mt-[5rem] pb-[5rem] w-full md:max-w-[800px] md:m-auto md:mt-24">
2   <BackButton />
3   <div className="w-full md:max-w-[800px] md:m-auto p-4">
4     <div className="flex flex-col items-center text-2xl font-bold text-center">
5       <h1 className="mb-4">
6         Retrouvez les producteurs autour de chez vous !
7       </h1>
8       <MapDynamic users={users} />
9     </div>
10    <div className="producer-card-container pb-20">
11      {users && users.map((user) => {
12        return <ProducerCard key={user.id} user={user} />;
13      })}
14    </div>
15  </div>
16</div>
```

Tailwind permet de définir rapidement des mises en page réactives. Par exemple, en utilisant des classes telles que **grid**, **flex**, **justify-center**, **items-center**, et des breakpoints comme **md**, **lg**, on peut facilement créer des interfaces utilisateur qui s'adaptent à différentes tailles d'écran.

- Approche Mobile First

L'application est conçue avec une approche **mobile first**, assurant que l'expérience utilisateur soit optimale sur les appareils mobiles. Les pages s'adaptent automatiquement aux différentes tailles d'écran grâce à l'utilisation de **media queries** et de classes Tailwind CSS pour les grilles et les flexbox.

Les composants sont conçus pour s'ajuster dynamiquement à la taille de l'écran, avec des listes de produits affichées en une seule colonne sur mobile et en grille sur desktop par exemple.



```
1 <section className="grid gap-6 grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-3 xl:grid-cols-3 justify-items-center mb-24 p-4">
2   {products.map((product: IItem) => (
3     <ProductCard key={product.id} product={product} />
4   )))
5 </section>
```

6.2 . Back-end

Le **back-end** de l'application constitue le cœur de la gestion des données et des processus métier. Il a été conçu pour être sécurisé, modulaire, et performant, tout en répondant aux exigences spécifiques de la mise en relation entre producteurs et consommateurs. Ce **back-end** s'appuie sur **Symfony**, un **framework PHP** qui suit le modèle de développement **MVC** (Model-View-Controller).

👉 Technologies Utilisées

- **Symfony**

Symfony est un **Framework PHP** servant de colonne vertébrale pour l'application. Il facilite le développement en offrant une structure claire et un ensemble d'outils et de **bundles** pour gérer les tâches courantes, telles que le routage, l'injection de dépendances, la gestion des formulaires, et plus encore.

- **Doctrine ORM**

Doctrine est l'**ORM** (Object-Relational Mapper) utilisé pour interagir avec la base de données. Il permet de manipuler les données à travers des entités PHP, évitant ainsi d'écrire des requêtes SQL manuellement. **Doctrine** facilite également la gestion des relations entre entités, comme les relations **ManyToOne**, **OneToMany**, etc.

- **JWT (JSON Web Tokens)**

JWT est utilisé pour l'authentification sécurisée des utilisateurs. Chaque utilisateur reçoit un **token JWT** lors de la connexion, qui est ensuite utilisé pour authentifier les requêtes API. Ce **token** est vérifié à chaque requête pour s'assurer de l'authenticité de l'utilisateur.

```
● ● ●
```

```
1 lexik_jwt_authentication:
2     secret_key: '%env(resolve:JWT_SECRET_KEY)%'
3     public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
4     pass_phrase: '%env(JWT_PASSPHRASE)%'
5
6     token_extractors:
7         cookie:
8             enabled: true
9             name: BEARER
10
11    set_cookies:
12        BEARER:
13            httpOnly: false
14            secure: false
15
16    remove_token_from_body_when_cookies_used: false
17
18
```

- **Twig**

Twig est utilisé principalement pour les vues dans le **back-office**, **Twig** est le moteur de **templates** de **Symfony** qui permet de générer des pages HTML dynamiques.

```
● ● ●
```

```
1 {% extends 'base.html.twig' %} 
2
3 {% block title %}Backoffice - O'Petits Oignons{% endblock %}
4
5 {% block body %} 
6
7     <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3 text-center">
8         <h1 class="h2">Bienvenue O'Petits Oignons &#128516;</h1>
9
10    </div>
11
12
13 {% endblock %}
14
```

- **LexikJWTAuthenticationBundle**

LexikJWT est un **bundle Symfony** utilisé pour la gestion des **JWT**. Il configure la génération, la validation ou l'invalidation des **tokens JWT**.

👉 Gestion des Utilisateurs et des Rôles

Le système de gestion des utilisateurs est conçu pour gérer différents types de rôles, chacun ayant des permissions spécifiques. Les principaux rôles sont :

- **ROLE_CONSUMER** : Ce rôle est attribué aux utilisateurs qui agissent en tant que consommateurs, leur permettant de parcourir les produits, de les ajouter à leur panier et de passer des commandes.
- **ROLE_PRODUCER** : Ce rôle est pour les producteurs qui gèrent leurs propres produits. Ils peuvent ajouter des produits et voir les commandes associées à leurs articles.
- **ROLE_ADMIN** : Ce rôle est réservé aux administrateurs de l'application, leur donnant un accès pour gérer les utilisateurs, les produits, les commandes, et superviser l'ensemble du système.

Chaque utilisateur a un rôle défini au moment de sa création, et les permissions sont gérées en conséquence.

```
1  /**
2   * @var list<string> The user roles
3   */
4  #[ORM\Column]
5  #[Assert\Choice(choices:[['ROLE_CONSUMER'], ['ROLE_PRODUCER'], ['ROLE_ADMIN']], max: 1)]
6  private array $role = [];
```

```
1  /**
2   * @see UserInterface
3   *
4   * @return list<string>
5   */
6
7  #[Groups(['user_browse', 'consumer_read', "producer_read", "order_read"])]
8  public function getRoles(): array
9  {
10
11     $role = $this->role;
12
13     // guarantee every user at least has ROLE_USER
14     // $roles[] = 'ROLE_USER';
15
16     return array_unique($role);
17 }
18
19 /**
20  * @param list<string> $roles
21  */
22 public function setRoles($roles): static
23 {
24     $this->role = $roles;
25
26     return $this;
27 }
```

👉 Sécurité et Authentification

Le système de sécurité est principalement géré via des **firewalls** configurés dans le fichier `security.yaml`.

Les **tokens JWT** sont utilisés pour authentifier les utilisateurs. Le **token** est vérifié pour chaque requête API protégée.

```
● ● ●
1 firewalls:
2   dev:
3     pattern: ^/(_profiler|wdt)|css|images|js/
4     security: false
5   login:
6     pattern: ^/api/v1/login
7     stateless: true
8     json_login:
9       check_path: /api/v1/login
10    success_handler: lexik_jwt_authentication.handler.authentication_success
11    failure_handler: lexik_jwt_authentication.handler.authentication_failure
12   api:
13     pattern: ^/api
14     stateless: true
15     jwt: ~
```

Les routes sont protégées en fonction des rôles des utilisateurs. Par exemple, les routes d'administration sont accessibles uniquement aux utilisateurs avec le rôle `ROLE_ADMIN`.

```
● ● ●
1 access_control:
2   - { path: ^/api/v1/user/id, roles: PUBLIC_ACCESS , methods: "GET" }
3   - { path: /api/v1/order/, roles: [ROLE_CONSUMER, ROLE_ADMIN], methods: "POST" }
4   - { path: ^/profile, roles: IS_AUTHENTICATED_FULLY }
5   - { path: ^/backoffice/login, roles: PUBLIC_ACCESS }
6   - { path: ^/backoffice, roles: ROLE_ADMIN }
```

👉 Event Listeners

Les **event listeners** permettent d'exécuter des actions spécifiques en réponse à des événements au sein de l'application.

AuthenticationSuccessListener intercepte les événements de succès d'authentification et ajoute des données supplémentaires à la réponse, telles que les rôles de l'utilisateur.

```
● ● ●
1 class AuthenticationSuccessListener {
2 /**
3 * @param AuthenticationSuccessEvent $event
4 */
5 public function onAuthenticationSuccessResponse(AuthenticationSuccessEvent $event)
6 {
7     $data = $event->getData();
8     $user = $event->getUser();
9
10    if (!$user instanceof UserInterface) {
11        return;
12    }
13
14    $data['data'] = array(
15        'roles' => $user->getRoles(),
16        'user' => $user->getId()
17    );
18
19    $event->setData($data);
20 }
21 }
```

AuthenticationFailureListener gère les échecs d'authentification en renvoyant un message d'erreur approprié.

```
● ● ●
1 <?php
2
3 namespace App\EventListener;
4
5 use Lexik\Bundle\JWTAuthenticationBundle\Event\AuthenticationFailureEvent;
6 use Lexik\Bundle\JWTAuthenticationBundle\Response\JWTAuthenticationFailureResponse;
7 use Symfony\Component\HttpFoundation\JsonResponse;
8
9
10 class AuthenticationFailureListener {
11     /**
12     * @param AuthenticationFailureEvent $event
13     */
14     public function onAuthenticationFailureResponse(AuthenticationFailureEvent $event)
15     {
16         $response = new JWTAuthenticationFailureResponse('Identifiant ou mot de passe invalide', JsonResponse::HTTP_UNAUTHORIZED);
17
18         $event->setResponse($response);
19     }
20 }
```

OrderNumberListener génère automatiquement un numéro de commande unique lors de la création d'une commande.



```
1  class OrderNumberListener
2  {
3      private $orderNumberGenerator;
4
5      public function __construct(OrderNumberGenerator $orderNumberGenerator)
6      {
7          $this->orderNumberGenerator = $orderNumberGenerator;
8      }
9
10     public function prePersist(LifecycleEventArgs $event)
11     {
12
13         $entity = $event->getObject();
14
15         // vérifier que l'entité est bien de type Order
16         if (!$entity instanceof Order) {
17             return;
18         }
19
20         if (!$entity->getOrderNumber())
21         {
22             $orderNumber = $this->orderNumberGenerator->generate();
23             $entity->setOrderNumber($orderNumber);
24         }
25     }
26
27 }
```

👉 Doctrine ORM et Gestion des Données

Doctrine ORM gère la persistance des données avec **MySQL**.

Les **repositories** sont des classes dédiées pour encapsuler les requêtes spécifiques, offrant une interface pour interagir avec la base de données.

```
1  class ProductRepository extends ServiceEntityRepository
2  {
3      public function __construct(ManagerRegistry $registry)
4      {
5          parent::__construct($registry, Product::class);
6      }
7
8      /**
9      * @return Product[] Returns an array of Product objects
10     */
11    public function findByExampleField($value): array
12    {
13        return $this->createQueryBuilder('p')
14            ->andWhere('p.exampleField = :val')
15            ->setParameter('val', $value)
16            ->orderBy('p.id', 'ASC')
17            ->setMaxResults(10)
18            ->getQuery()
19            ->getResult()
20        ;
21    }
}
```



7. RÉALISATIONS PERSONNELLES

Durant cette formation j'ai choisi de m'orienter vers une spécialisation **React**. La plupart des réalisations faites sur ce projet portent sur la partie **front-end**.

Exemples de réalisations :

👉 Carousel d'images

Sur la page d'accueil et sur la page de profil d'un producteur, un carrousel d'image à été mis en place. Plusieurs images défilent à intervalle régulier.

Un composant "Carrousel" a été créé et intégré à la page d'accueil.

```
● ● ●
1  /* Header section */
2      <div className="flex flex-col">
3          <div className="home-presentation text-center mt-6">
4              <h2 className="text-2xl font-bold mb-4">
5                  Bienvenue sur l'application Oignons !
6              </h2>
7              <p className="mb-3 mt-3">
8                  Retrouvez des producteurs grâce à notre chercheur de
9                  produits.
10             </p>
11         </div>
12         <div className="input-search">
13             /* Composant SearchInput */
14             <SearchInput />
15         </div>
16
17     /* Carousel d'images */
18     <Carousel />
19
```

Le composant commence par importer les modules nécessaires de **React** et **Next.js**, ainsi que les images qui seront affichées dans le carrousel.

```
● ● ●
1  /* --- REACT --- */
2  import { useEffect, useState } from "react";
3
4  /* --- NEXT --- */
5  import Image from "next/image";
6
7  /* --- IMPORT */
8  import img1 from "../../public/img/Media/HomePage/img1.jpg";
9  import img2 from "../../public/img/Media/HomePage/img2.jpg";
10 import img3 from "../../public/img/Media/HomePage/img3.jpg";
11 import img4 from "../../public/img/Media/HomePage/img4.jpg";
12 import img5 from "../../public/img/Media/HomePage/img5.jpg";
13 import img6 from "../../public/img/Media/HomePage/img6.jpg";
```

Le composant utilise le **hook useState** pour gérer l'index de la diapositive actuellement affichée. Cet index est stocké dans le state **currentSlide**.

```
● ○ ●  
1 const Carousel = () => {  
2   // État pour suivre l'index de la diapositive actuelle  
3   const [currentSlide, setCurrentSlide] = useState(0);  
4 }
```

Les images du carrousel sont stockées dans un tableau **slides**. Ce tableau est utilisé pour générer les diapositives du carrousel.

```
● ○ ●  
1 // Liste des images à afficher dans le carrousel  
2 const slides = [img1, img2, img3, img4, img5, img6];
```

Le hook **useEffect** est utilisé pour mettre en place un intervalle qui change automatiquement l'index de la diapositive toutes les 3 secondes. À chaque **itération**, l'index **currentSlide** est incrémenté de 1, et lorsque l'index atteint la fin du tableau **slides**, il revient à 0 grâce à l'opérateur **modulo**.

L'opérateur **modulo** est utilisé pour s'assurer que l'index de la diapositive **currentSlide** reste toujours dans les limites du tableau **slides**.

```
● ○ ●  
1 // Utilisation d'un useEffect pour changer la diapositive toutes les 3 secondes  
2 useEffect(() => {  
3   const interval = setInterval(() => {  
4     setCurrentSlide((prevSlide) => (prevSlide + 1) % slides.length);  
5   }, 3000);  
6  
7   // Nettoie l'intervalle à la fin du composant pour éviter les fuites de mémoire  
8   return () => clearInterval(interval);  
9 }, [slides.length]);  
10
```

C'est une **div** qui contient tout le carrousel. Il est stylisé pour occuper toute la largeur disponible. Chaque diapositive est un enfant de ce conteneur, et seules les diapositives correspondant à l'index **currentSlide** sont affichées grâce à la gestion de l'opacité.

```

1  return (
2      // Conteneur principal du carrousel
3      <div
4          id="default-carousel"
5          className="relative w-full p-2"
6          data-carousel="slide"
7      >
8          /* Conteneur pour les diapositives avec ratio d'aspect */
9          <div className="relative w-full overflow-hidden rounded-lg aspect-w-16 aspect-h-9">
10             {slides.map((slide, index) => (
11                 // Chaque diapositive du carrousel
12                 <div
13                     key={index}
14                     className={`${`absolute inset-0 transition-opacity duration-700 ease-in-out ${index === currentSlide ? "opacity-100" : "opacity-0"}`}
15                     data-carousel-item
16                 >
17                     /* Image de la diapositive */
18                     <Image
19                         src={slide}
20                         layout="fill"
21                         objectFit="cover"
22                         alt={`Slide ${index + 1}`}
23                         placeholder="blur"
24                         className="absolute inset-0 w-full h-full"
25                     />
26                 </div>
27             )))
28         </div>
29     )
30   </div>
31

```

Des boutons situés en bas du carrousel permettent à l'utilisateur de naviguer manuellement entre les diapositives. Chaque bouton correspond à une diapositive, et son apparence change en fonction de l'index actuel de la diapositive.

```

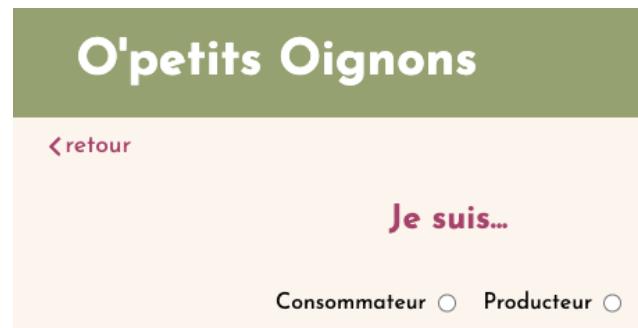
1  /* Indicateurs de navigation */
2      <div className="absolute z-10 flex -translate-x-1/2 bottom-5 left-1/2 space-x-3 rtl:space-x-reverse">
3          {slides.map(_, index) => (
4              // Boutons pour naviguer directement vers une diapositive
5              <button
6                  key={index}
7                  type="button"
8                  className={`${`w-3 h-3 rounded-full ${index === currentSlide ? "bg-white" : "bg-gray-400"}`}
9                      index === currentSlide ? "bg-white" : "bg-gray-400"
10                     `}
11                     aria-current={index === currentSlide}
12                     aria-label={`Slide ${index + 1}`}
13                     onClick={() => setCurrentSlide(index)}
14                 ></button>
15             )))

```

👉 Formulaires

Différents formulaires sont utilisés dans l'application. Ils sont utilisés notamment lors de la connection d'un utilisateur ou lors de son inscription.

Lors de l'inscription, les utilisateurs choisissent leur rôle (consommateur ou producteur) via un formulaire interactif. Le choix du rôle détermine les champs spécifiques à afficher dans le formulaire d'inscription. Cette sélection est gérée par un composant **React** avec un **state** contrôlé.



```
● ● ●  
1 const page = () => {  
2   /* STATE  
3   // Choix du formulaire  
4   // eslint-disable-next-line react-hooks/rules-of-hooks  
5   const [selectedOption, setSelectedOption] = useState("");
```

L'option sélectionnée détermine quels champs du formulaire s'affichent, en passant dynamiquement le rôle choisi au composant **SignFormUser**

```
● ● ●  
1 <section>  
2   {selectedOption && <SignFormUser role={selectedOption} />}  
3 </section>
```

Les formulaires utilisent **Formik** pour la gestion des états et des soumissions, tandis que **Yup** est utilisé pour la validation des données. La validation des champs est spécifique à chaque rôle et assure que les données respectent les formats attendus, comme les emails, les numéros de SIRET, etc.

Exemple de validation :



```
1 let validationSchema = Yup.object().shape({  
2     lastname: Yup.string().required("Ce champs est obligatoire"),  
3     firstname: Yup.string().required("Ce champs est obligatoire"),  
4     email: Yup.string()  
5         .required("Ce champs est obligatoire")  
6         .email("Veuillez saisir un email valide"),
```

Les utilisateurs peuvent modifier leur profil, y compris des champs spécifiques à leur rôle (comme le SIRET pour les producteurs). Les champs du formulaire sont initialement en lecture seule, mais peuvent être rendus modifiables individuellement, en cliquant sur une icône, grâce à la gestion des états avec React.



```
1 // Modification du formulaire  
2 const [isModified, setIsModified] = useState(false);  
3  
4 // Champs du formulaire en lecture seule  
5 const [isReadOnly, setIsReadOnly] = useState({  
6     email: true,  
7     companyName: true,  
8     lastName: true,  
9     firstName: true,
```



```
1 // Champs de formulaire en lecture seule  
2 const handleFieldEdit = (fieldName: ReadOnlyFields) => {  
3     setIsReadOnly((prevState) => ({  
4         ...prevState,  
5         [fieldName]: !prevState[fieldName],  
6     }));  
7     setIsModified(true);  
8 };
```



```
1  {/* Nom */}
2  <div className="mb-4">
3      <label htmlFor="lastName">Nom : </label>
4      <div className="flex items-center justify-center">
5          <div className="flex flex-col w-full">
6              <Field
7                  className={`rounded-[4rem] p-2 w-full ${isReadOnly.lastName ? "non-interactive" : ""}`}
8                  name="lastName"
9                  readOnly={isReadOnly.lastName}
10             />
11             <ErrorMessage
12                 name="lastName"
13                 className="text-red-800 text-sm ml-2"
14                 component="div"
15             />
16         </div>
17     <div className="ml-2">
18         {isReadOnly.lastName ? (
19             <FaPencil
20                 size={20}
21                 className="text-purple_active"
22                 onClick={() =>
23                     handleFieldEdit("lastName")
24                 }
25             />
26         ) : (
27             <FaCheck
28                 size={30}
29                 className="text-primary_active"
30                 onClick={() =>
31                     handleFieldEdit("lastName")
32                 }
33             />
34         )}
35     </div>
36     </div>
37 </div>
```

8. ASPECTS SÉCURITÉ

La sécurité est un aspect fondamental de toutes applications, tant du côté **front-end** que **back-end**. Diverses mesures ont été mises en place pour protéger les données des utilisateurs et garantir une communication sécurisée entre les différents composants de l'application.

Sécurité côté Front-End

Pour assurer la sécurité des données utilisateurs et se protéger contre les menaces courantes, plusieurs mécanismes ont été mis en œuvre.

- Validation des Formulaires avec **Yup** et **Formik**

Les formulaires utilisateur sont validés côté client avec **Yup** qui est une bibliothèque de validation schématique. Cela permet de prévenir les entrées invalides avant même qu'elles ne soient envoyées au serveur.

Exemples de validation : contrôle de la validité de l'adresse email, vérification de la longueur minimale des mots de passe, etc.

```
● ● ●  
1 // Schéma de validation  
2 const validationSchema = Yup.object({  
3   username: Yup.string()  
4     .required(`L'email est requis`)  
5     .email("Veuillez saisir un email valide"),  
6   password: Yup.string().required("Le mot de passe est requis"),  
7 });
```

- Gestion du **token JWT**

Après authentification, un **token JWT** (JSON Web Token) est stocké de manière sécurisée dans le **sessionStorage**. Cela permet de protéger le token contre les attaques **XSS** (Cross-Site Scripting) puisque les tokens ne sont pas accessibles via JavaScript, contrairement au **localStorage**.

Le token est envoyé avec chaque requête API protégée, assurant que l'utilisateur est bien authentifié.

```

1  if (isLoginSuccess(connection)) {
2      const loginSuccess = connection;
3      const newUser = {
4          token: loginSuccess.token,
5          userId: loginSuccess.userId,
6          role: loginSuccess.role,
7          picture: null,
8      };
9      dispatch(setUser(newUser));
10     router.push("/profil");
11 }

```

- HTTPS et CSP (Content Security Policy)

Toutes les communications entre le client et le serveur sont chiffrées via **HTTPS**, empêchant les attaques de type **Man-in-the-Middle**.

- **Imgbb** pour le téléchargement d'images

Les images téléchargées par les utilisateurs (les photos de profil par exemple) sont directement envoyées à **Imgbb** via une API sécurisée. Cela permet d'éviter les attaques liées aux fichiers malveillants qui pourraient être déposés sur le serveur.

```

1  if (values.picture !== "") {
2      const formData = new FormData();
3      formData.append("image", values.picture.substring(22));
4
5      try {
6          const responseImage = await fetch(
7              "https://api.imgur.com/1/upload?key=e494e2154a4621708abb7ef9e65b07cc",
8              {
9                  method: "POST",
10                 body: formData,
11             }
12         );
13
14         if (responseImage.ok) {
15             const dataImage = await responseImage.json();
16             imageUrl = dataImage.data.url;
17             console.log(imageUrl);
18         } else {
19             console.log("erreur");
20         }
21     } catch (error) {
22         console.log(error);
23     }
24 }

```

👉 Sécurité côté Back-End

Plusieurs mesures de sécurité ont également été implémentées pour protéger le serveur et les données des utilisateurs.

- Authentification **JWT** (JSON Web Token)

Le back-end utilise des **tokens JWT** pour l'authentification des utilisateurs. Chaque token est signé et peut être vérifié pour s'assurer de l'identité de l'utilisateur.

Les tokens sont valides pendant une durée déterminée, après quoi l'utilisateur doit se reconnecter.

```
● ● ●  
1 firewalls:  
2   dev:  
3     pattern: ^/(_(profiler|wdt)|css|images|js)/  
4     security: false  
5   login:  
6     pattern: ^/api/v1/login  
7     stateless: true  
8     json_login:  
9       check_path: /api/v1/login  
10      success_handler: lexik_jwt_authentication.handler.authentication_success  
11      failure_handler: lexik_jwt_authentication.handler.authentication_failure  
12   api:  
13     pattern: ^/api  
14     stateless: true  
15     jwt: ~
```

- Gestion des Rôles et des Permissions

Les utilisateurs sont catégorisés en trois rôles principaux : **Consommateur**, **Producteur**, et **Administrateur**. Chaque rôle a des permissions spécifiques, et l'accès aux routes est contrôlé en fonction de ces rôles.

Par exemple, seules les commandes effectuées par un utilisateur spécifique sont accessibles à ce dernier, et les opérations d'administration ne sont accessibles qu'aux administrateurs.

```
● ● ●  
1 access_control:  
2   - { path: ^/api/v1/user/id, roles: PUBLIC_ACCESS , methods: "GET" }  
3   - { path: /api/v1/order/, roles: [ROLE_CONSUMER, ROLE_ADMIN], methods: "POST" }  
4   - { path: ^/profile, roles: IS_AUTHENTICATED_FULLY }  
5   - { path: ^/backoffice/login, roles: PUBLIC_ACCESS }  
6   - { path: ^/backoffice, roles: ROLE_ADMIN }
```

- Validation des Données avec **Symfony Validator** :

Toutes les données envoyées par les utilisateurs sont strictement validées avant d'être traitées ou stockées en base de données. Cela inclut la validation des formulaires, la vérification des formats d'e-mail, des numéros de téléphone, etc.

```
● ● ●  
1 #[ORM\Column(length: 180)]  
2 #[Groups([  
3     'user_browse', 'consummer_read', 'order_read'  
4 ]])  
5 #[Assert\Email(message: 'The email {{ value }} is not a valid email.')]  
6 #[Assert\NotBlank]  
7 private ?string $email = null;  
8
```

- Protection contre les Attaques CSRF

Les formulaires du back-office sont protégés contre les attaques **CSRF** en utilisant des **tokens** générés par **Symfony**. Cela empêche les actions non autorisées depuis des sites externes.

- Encodage des mot de passe

Les mots de passe sont **hachés** avant d'être stockés en base de données à l'aide d'algorithmes sécurisés, garantissant que les mots de passe sont protégés même si la base de données est compromise.

```
● ● ●  
1 security:  
2     password_hashers:  
3         Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:  
4             algorithm: auto  
5             cost: 4 # Lowest possible value for bcrypt  
6             time_cost: 3 # Lowest possible value for argon  
7             memory_cost: 10 # Lowest possible value for argon
```

- Protection contre les Injections SQL :

Doctrine protège contre les injections SQL de plusieurs façons.

Requêtes préparées : Doctrine utilise des requêtes préparées pour toutes les interactions avec la base de données. Cela signifie que les paramètres des requêtes sont automatiquement échappés, ce qui empêche l'injection de code SQL malveillant.

Utilisation des DQL (Doctrine Query Language) : Le **DQL** est un langage spécifique à **Doctrine** qui ressemble à du **SQL**, mais qui fonctionne avec les entités et les relations au lieu des tables et colonnes **SQL** directement. Comme les **DQL** sont compilés en **SQL** par **Doctrine**, cela réduit le risque d'injection **SQL**.

- Gestion des CORS

Dans **Symfony**, le bundle **NelmioCorsBundle** est couramment utilisé pour gérer les politiques **CORS** de manière flexible et configurable. Ce **bundle** permet de définir les règles **CORS** au niveau de l'application, ce qui facilite le contrôle des accès cross-origin pour les différentes routes de votre API.

```

● ● ●

1  nelmio_cors:
2      defaults:
3          origin_regex: true
4          allow_origin: ['http://localhost:3000']
5          allow_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']
6          allow_headers: ['Content-Type', 'Authorization']
7          allow_credentials: true
8          expose_headers: ['Link']
9          max_age: 3600
10     paths:
11         '/': null
12         '^/api/':
13             allow_origin: ['*']
14             allow_headers: ['*']
15             allow_methods: ['POST', 'PUT', 'GET', 'DELETE']
16             max_age: 3600
17

```

9. DIFFICULTÉS RENCONTRÉES

👉 Gestion du z-index

Lors de la mise en place de la page d'accueil et de l'input, il était impossible d'écrire à l'intérieur ou de pouvoir valider avec le bouton "Rechercher".



Le dysfonctionnement venait du fait que dans les classes de **main**, il y avait le **-z-10** pour la gestion du **z-index**, afin que le header et le footer soient toujours au dessus des autres éléments.

L'input et le bouton fonctionnent correctement après avoir retiré le **z-index** sur cet élément.

👉 Conflits Github

Le fait de travailler en équipe nous oblige à vérifier que nous n'avons pas modifié d'éléments identiques de notre code lors de merge sur une branche commune (Develop par exemple).

La plupart des conflits sont gérés directement depuis l'interface web de **Github** quand cela ne concerne que peu de conflits. Si beaucoup de fichiers sont en conflits, les modifications ne peuvent pas se faire mais doivent être corrigées depuis notre **IDE**.

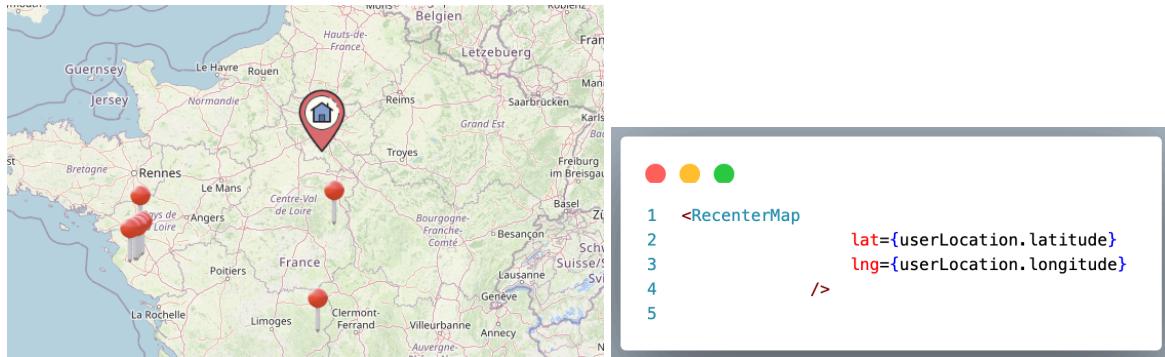
Nous avons utilisé la documentation **Github** fournie pour appliquer les correctifs sur nos conflits.

<https://docs.github.com/fr/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line>

👉 Géolocalisation

Après avoir récupéré les coordonnées GPS dans le **sessionStorage**, le centrage de la map ne se faisait pas dessus mais sur les coordonnées par défaut présentes dans le **state**.

Le point qui indique la localisation de l'utilisateur se fait correctement.



Ce dysfonctionnement a été corrigé avec l'utilisation d'un **useEffect**.

Dès le chargement du composant, un **useEffect** est déclenché pour vérifier si la position de l'utilisateur est déjà stockée dans le **sessionStorage**. Si c'est le cas, la position (latitude et longitude) est extraite et utilisée pour initialiser le **state userLocation** via **setUserLocation**.

```
● ● ●
1 const Map = ({ users }: MapProps) => {
2
3   const [userLocation, setUserLocation] = useState<{
4     latitude: number;
5     longitude: number;
6   }>({
7     latitude: 48.8566,
8     longitude: 2.3522,
9   });
10 // console.log(userLocation);
11
12 useEffect(() => {
13   if (typeof window !== 'undefined' && typeof localStorage !== 'undefined') {
14     const storedLocation = sessionStorage.getItem("userLocation");
15     if (storedLocation) {
16       try {
17         const parsedLocation = JSON.parse(storedLocation);
18         if (parsedLocation.latitude && parsedLocation.longitude) {
19           setUserLocation(parsedLocation);
20         }
21       } catch (error) {
22         console.error("Error parsing stored location:", error);
23       }
24     }
25   }
26 }, []);
```

Par défaut, la position est définie sur Paris (latitude : 48.8566, longitude : 2.3522) si aucune position n'est trouvée dans le **sessionStorage**.



10. CONCLUSION

La mise en place de ce projet (ainsi que la formation) m'a permis de comprendre le métier de développeur web en situation réelle. Ce fut une excellente expérience tant professionnelle qu'humaine. L'engagement de chacun des membres de l'équipe et l'entraide ont grandement participé à la réussite de ce projet.

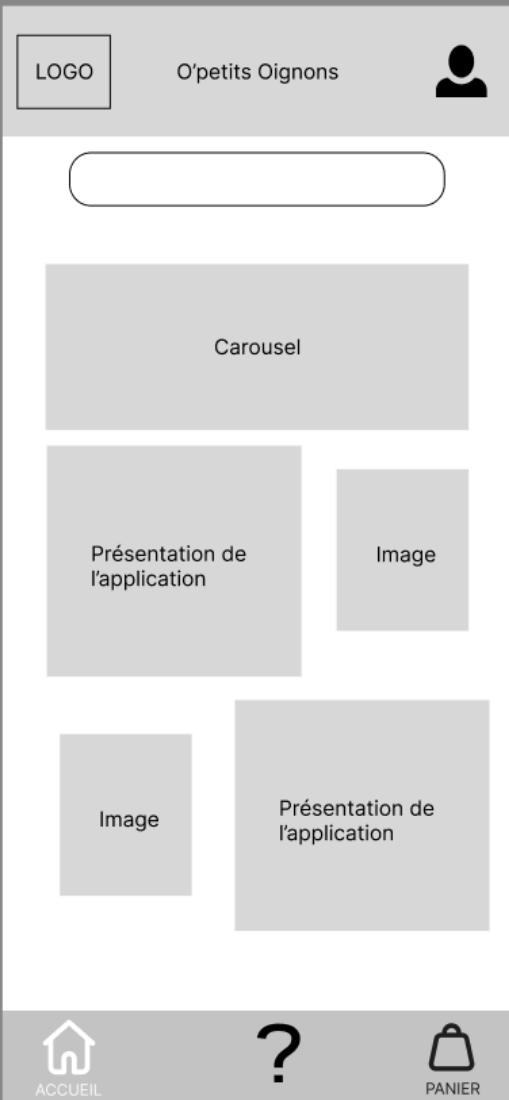
La mise en situation quasi-professionnelle sur un projet complet m'a permis de pratiquer toutes les facettes du développement web et aussi de monter en compétences, notamment sur le développement front.

Ces derniers mois passés à apprendre de nouvelles connaissances, imaginer et construire un projet de 0, se confronter à des problèmes et les résoudre, travailler en équipe et bien d'autres choses n'ont fait que confirmer mon envie d'exercer ce métier de développeur web.

 ANNEXES

👉 Wireframes et maquettes

Wireframe - Accueil mobile



Logo

O'petits Oignons

Carousel

Présentation de l'application

Image

Image

Présentation de l'application

ACCUEIL

?

PANIER

Maquette - Accueil mobile



O'petits Oignons

Retrouvez tout les producteurs proches de chez vous grâce à votre code postal ou votre ville !

Ville ou code postal...

RECHERCHER

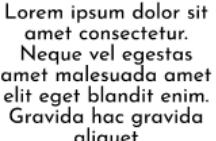
Voir les producteurs

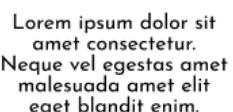
  

...





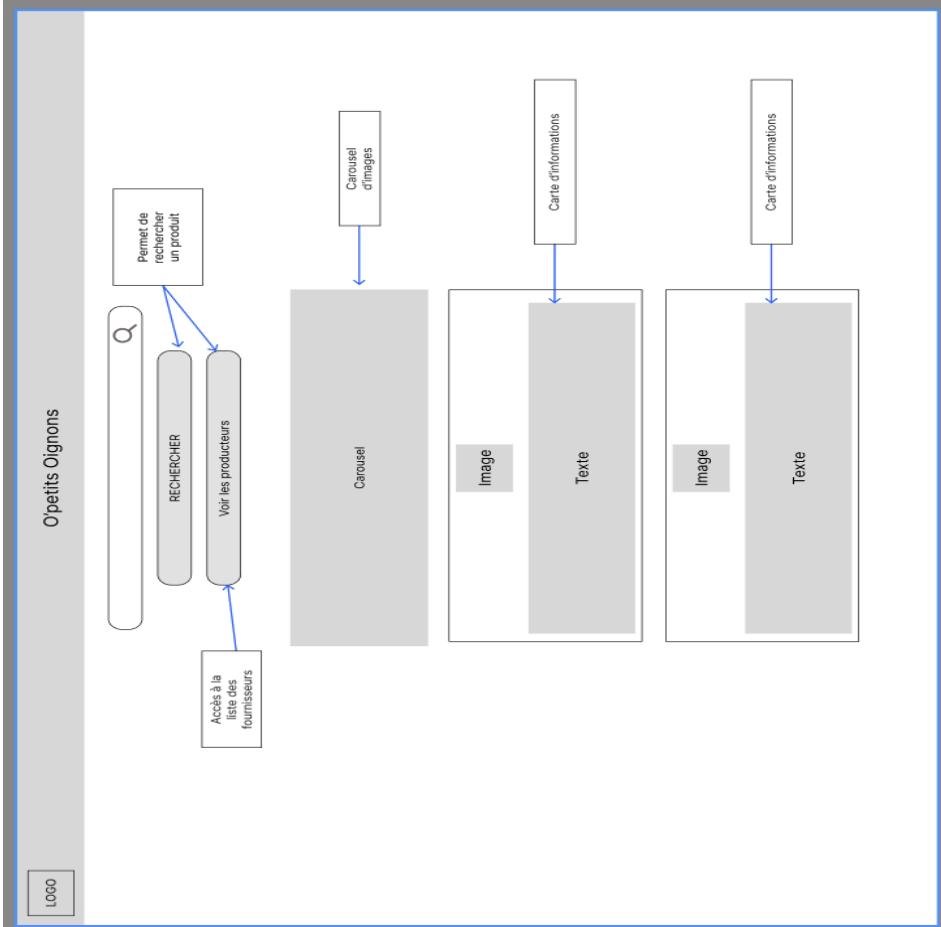




ACCUEIL

INFOS

PANIER



Inscription Client mobile



O'petits Oignons

INSCRIPTION

Nom :

Prénom :

Email :

Mot de passe :

Code postal :

Ville :

 ACCUEIL  FAQ  PANIER

Maquette - Inscription mobile



O'petits Oignons 

Inscrivez-vous !

Je suis... Consommateur Producteur

Nom :

Prénom :

Email :

Mot de passe :

Code postal :

Ville :

J'atteste que j'ai lu et accepté les [conditions générales d'utilisation](#)

 ACCUEIL  INFOS  PANIER

Maquette - Inscription Desktop

Opetits Oignons

ACCUEIL



Inscrivez-vous !

Consommateur Producteur

Je suis...:

Nom :

Prénom :

Email :

Mot de passe :

Confirmez le mot de passe :

Code postal :

Ville :

J'atteste que j'ai lu et accepté les conditions générales d'utilisation

Maquette - Inscription Desktop



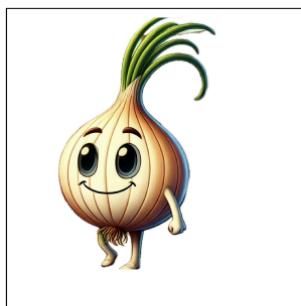
👉 User stories

Thème	En tant que...	j'ai besoin de ...	afin de
Accueil	Visiteur	d'une description de l'application	connaître le but principal du site
Accueil	Visiteur	rechercher les producteurs proches de chez moi	acheter des produits locaux
CGV	Visiteur	voir les conditions générales de vente	d'être informé des règles de vente du site
Mentions légales	Visiteur	voir les mentions légales	d'être informé des mesures légales du site
FAQ	Visiteur	voir la foire aux questions	trouver des réponses sur le fonctionnement du site
Contact	Visiteur	contacter l'administrateur du site	poser les questions dont je ne trouve pas les réponses autrement
Labels	Visiteur	voir la liste des labels	de mieux connaître la qualité des produits proposés.
Création de compte	Visiteur	créer un compte	pouvoir profiter de tous les services du site
Connexion avec compte existant	Consommateur ou producteur	me connecter à mon compte avec les erreurs éventuelles de connexion	pouvoir profiter des services du site
Connexion	Consommateur ou producteur	rester connecté	ne pas rentrer tout le temps ses informations de connexion
Compte	Consommateur ou producteur	récupérer / réinitialiser le mot de passe	
Déconnexion	Consommateur ou producteur	me déconnecter à mon compte	
Compte	Consommateur ou producteur	supprimer mon compte	

Recherche	Consommateur	chercher un producteur proche de chez moi	d'acheter des produits locaux et favoriser le circuit court
Recherche	Consommateur	une page de résultats	consulter la liste des producteurs.
Affichage	Consommateur	voir une page de présentation du producteur et de ses produits	avoir des détails sur le producteur
Affichage	Consommateur	voir une page d'informations détaillées du produit	avoir des détails sur le produit
Panier	Consommateur	pouvoir ajouter un produit dans mon panier	
Panier	Consommateur	pouvoir visualiser le contenu détaillé de mon panier	afin de visualiser les produits commandés (quantité et prix) et le montant total de la commande
Panier	Consommateur	pouvoir modifier les quantités de produit dans mon panier	
Panier	Consommateur	pouvoir supprimer un produit de mon panier	
Panier	Consommateur	pouvoir visualiser le contenu de mon panier à tout moment sur le site	
Panier	Consommateur	visualiser les dates de retrait	
Profil	Consommateur	de voir mes informations personnelles	modifier mon profil
Profil	Consommateur	consulter un historique des commandes (passées / en-cours)	
Profil	Producteur	de voir mes informations personnelles	modifier mon profil
Profil	Producteur	consulter un historique des commandes (passées / en-cours)	
Produits	Producteur	ajouter un produit	

Produits	Producteur	modifier un produit	
Produits	Producteur	supprimer un produit	
Backoffice	Administrateur	d'ajouter une commande	
Backoffice	Administrateur	de modifier une commande	
Backoffice	Administrateur	de supprimer une commande	
Backoffice	Administrateur	de voir une commande	
Backoffice	Administrateur	d'ajouter un compte utilisateur	
Backoffice	Administrateur	de modifier un compte utilisateur	
Backoffice	Administrateur	de supprimer un compte utilisateur	
Backoffice	Administrateur	d'ajouter un produit	
Backoffice	Administrateur	de modifier un produit	
Backoffice	Administrateur	de supprimer un produit	

👉 Charte graphique



🎨 Couleurs	
Principale Active : #709a0b	
Principale Inactive : #91A16A	
Secondaire Active : #EA6F0E	
Secondaire Inactive : #EEAC78	
Purple Active : #BB2F6B	
Purple Inactive : #BD6F90	
Background : #FEF4EC	

🔤 Polices	
Josefin Sans	
Sans Serif	

■ Boutons

[En savoir plus](#)

Icones

ACCUEIL PANIER INFOS

👉 Dictionnaire de données

Utilisateur

Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id de l'utilisateur
<code>lastname</code>	VARCHAR(64)	NOT NULL	Le nom de l'utilisateur
<code>firstname</code>	VARCHAR(64)	NOT NULL	Le prénom de l'utilisateur
<code>email</code>	VARCHAR(64)	UNIQUE, NOT NULL	L'email de l'utilisateur
<code>password</code>	VARCHAR(255)	NOT NULL	Le mot de passe de l'utilisateur
<code>picture</code>	VARCHAR(255)	NULL	L'image de profil de l'utilisateur
<code>slug</code>	VARCHAR(64)	NOT NULL	Le slug de l'utilisateur
<code>streetNumber</code>	SMALLINT	NULL, UNSIGNED	Le numéro de l'adresse de l'utilisateur
<code>address1</code>	VARCHAR(255)	NOT NULL	L'adresse de l'utilisateur
<code>address2</code>	VARCHAR(255)	NULL	Le complément d'adresse
<code>postalCode</code>	SMALLINT	NOT NULL	Le code postal de l'utilisateur
<code>city</code>	VARCHAR(64)	NOT NULL	La ville de l'utilisateur
<code>phoneNumber</code>	CHAR(10)	NULL	Le téléphone de l'utilisateur
<code>role</code>	LONGTEXT	NOT NULL	Le rôle de l'utilisateur
<code>companyName</code>	VARCHAR(255)	NULL	Le nom d'entreprise (si utilisateur est producteur)
<code>description</code>	MEDIUMTEXT	NULL	Description de l'entreprise
<code>longitude</code>	DECIMAL	NOT NULL	(se renseigner sur le type)
<code>latitude</code>	DECIMAL	NOT NULL	(se renseigner sur le type)
<code>siret</code>	INT	UNIQUE	Le SIRET
<code>openingHours</code>			(se renseigner)
<code>createdAt</code>	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	La date de création de l'utilisateur
<code>updatedAt</code>	TIMESTAMP	NULL	La date de mise à jour de l'utilisateur

Produit

Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id du produit
<code>id_utilisateur</code>	INT	FOREIGN KEY	L'id du producteur
<code>name</code>	VARCHAR(64)	NOT NULL	Le nom du produit
<code>description</code>	MEDIUMTEXT	NOT NULL	La description du produit
<code>picture</code>	VARCHAR(255)	NULL	L'image du produit remplacée par une image générique si le producteur n'en mets pas
<code>htPrice</code>	DECIMAL(3,2)	NOT NULL, UNSIGNED	Le prix unitaire du produit hors taxes
<code>ttcPrice</code>	DECIMAL(3,2)	NOT NULL, UNSIGNED	Le prix unitaire du produit TTC
<code>tva</code>	DECIMAL(2,2)	NOT NULL, UNSIGNED	La TVA
<code>slug</code>	VARCHAR(64)	NOT NULL	Le slug du produit
<code>unitSale</code>	ENUM('litre', 'kilogramme', 'pièce', 'lot')	NOT NULL	
<code>availability</code>	BOOLEAN	NOT NULL	0: non dispo ; 1: dispo
<code>createdAt</code>	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	La date de création du produit
<code>updatedAt</code>	TIMESTAMP	NULL	La date de mise à jour du produit

Label

Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id du label
<code>name</code>	VARCHAR(64)	NOT NULL	Le nom du label
<code>created_at</code>	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	La date de création du label
<code>updated_at</code>	TIMESTAMP	NULL	La date de mise à jour du label

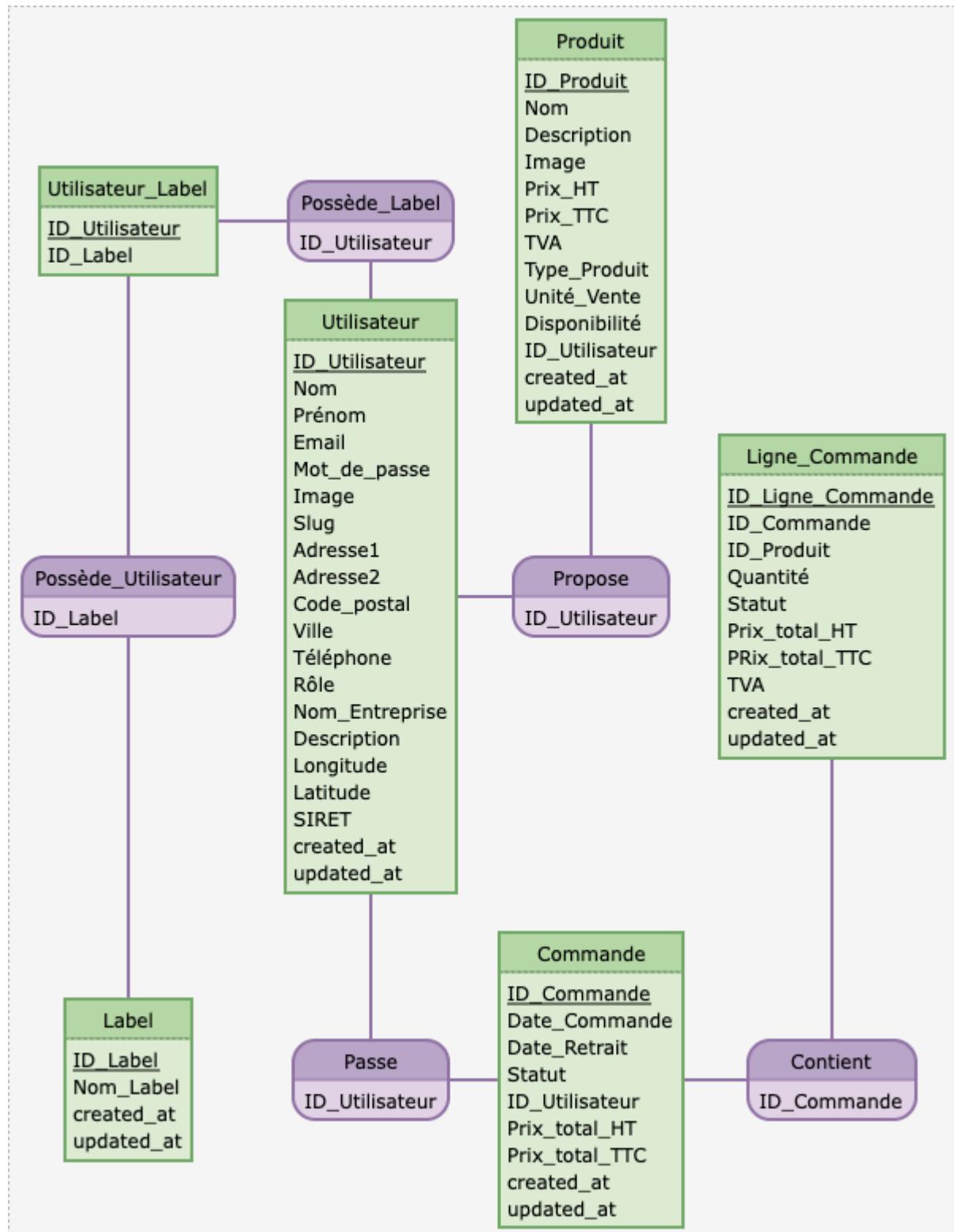
Commande

Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id de la commande
<code>id_utilisateur</code>	INT	FOREIGN KEY	L'id du consommateur
<code>status</code>	ENUM('En préparation', 'Prête', 'Récupérée')	NOT NULL	Le statut de la commande
<code>htPrice</code>	DECIMAL(4,2)	NOT NULL, UNSIGNED	Le prix totale HT de la commande
<code>ttcPrice</code>	DECIMAL(4,2)	NOT NULL, UNSIGNED	Le prix totale TTC de la commande
<code>createdAt</code>	TIMESTAMP	NOT NULL, [current_timestamp()]	

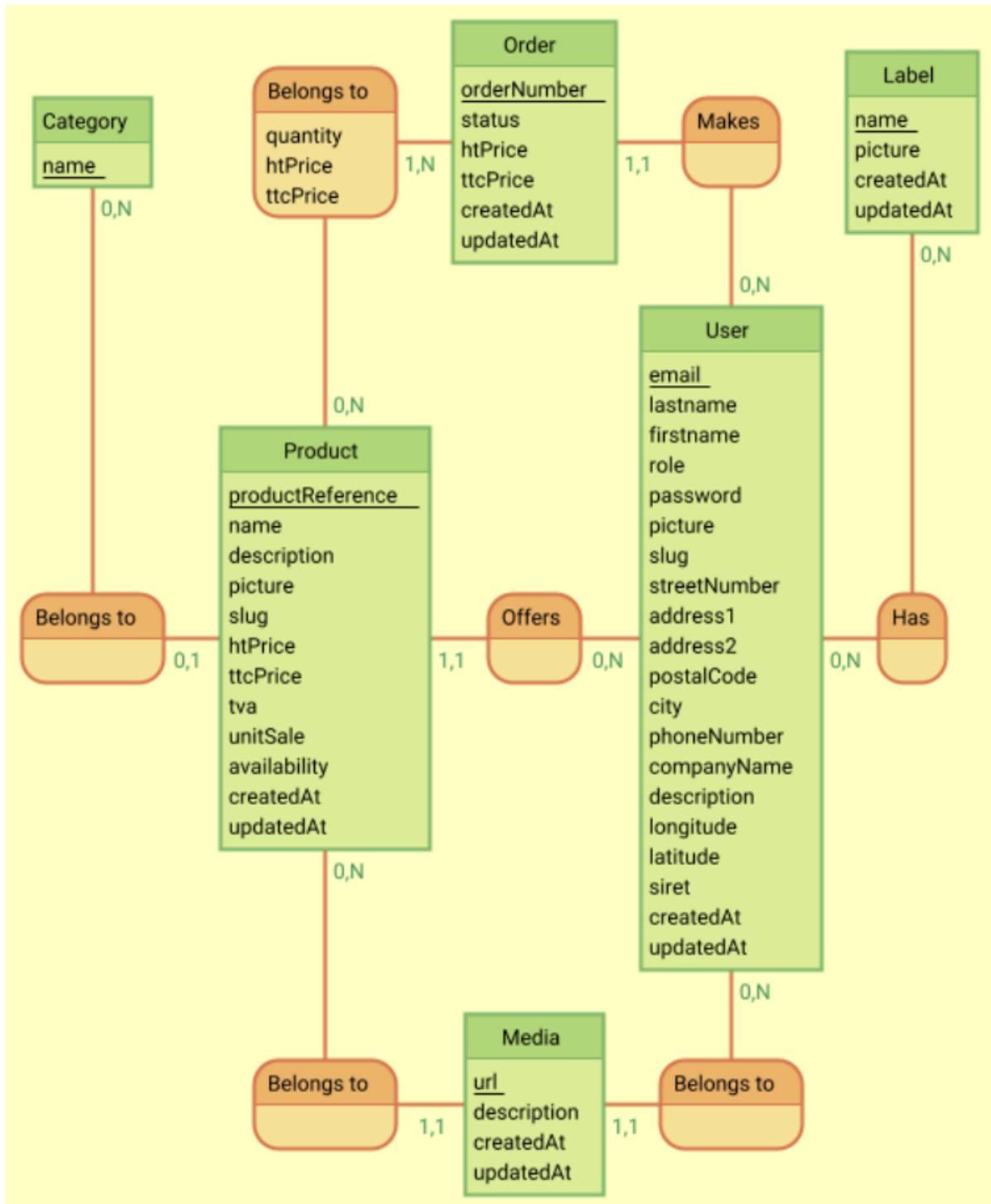
Ligne_Commande

Champ	Type	Spécificités	Description
<code>id</code>	INT	NOT NULL, AUTOINCREMENT, PRIMARY KEY, UNSIGNED	L'id de la ligne de commande
<code>id_commande</code>	INT	FOREIGN KEY	L'id de la commande
<code>id_produit</code>	INT	FOREIGN KEY	L'id du produit
<code>quantity</code>	TINYINT	UNSIGNED	La quantité
<code>htPrice</code>	DECIMAL(4,2)	NOT NULL, UNSIGNED	Le prix HT de la ligne
<code>ttcPrice</code>	DECIMAL(4,2)	NOT NULL, UNSIGNED	Le prix TTC de la ligne
<code>createdAt</code>	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	La date de création de la ligne
<code>updatedAt</code>	TIMESTAMP	NULL	La date de mise à jour de la ligne

👉 Modèle Logique de Données (MLD)



👉 Modèle Conceptuel de Données (MCD)



👉 Routes

Routes de l'application

URL	Méthode HTTP	Contrôleur	Méthode	Titre HTML	Commentaire
/		Bienvenue sur O'Petits Oignons	Page d'accueil: présentation du site avec un champ de recherche
/cgv		Conditions Générales de vente	-
/mentions-légales		Mentions légales	-
/faq		Foire aux questions / Contacter O'Petits Oignons	Affiche un formulaire d'envoi de message à l'administrateur et présente une foire aux questions
/labels		Les labels agricoles	Informations sur les labels en agriculture
/inscription		Créer un compte	Inscription de l'utilisateur
/connexion		Se connecter	Connexion de l'utilisateur
/profil		Mon profil	Profil
/commandes			Liste des commandes

Producteur

URL	Méthode HTTP	Contrôleur	Méthode	Titre HTML	Commentaire
/commande/{id}			Détail d'une commande
/produits			Gestion des produits

Consommateur

URL	Méthode HTTP	Contrôleur	Méthode	Titre HTML	Commentaire
/commande/{id}			Détail d'une commande
/{slug_du_producteur}/catalogue			Catalogue du producteur
/{slug_du_producteur}/produit/{id}		Détails d'un produit	Détails d'un produit
/panier		Panier	Panier

Back Office

Les controllers du back office sont dans un sous dossier back office

URL	Méthode HTTP	Contrôleur	Méthode	Titre HTML	Commentaire
/back/			home	Bienvenue sur le backoffice	Page d'accueil
/back/utilisateur			browse	Administration des utilisateurs	Liste des utilisateurs
/back/utilisateur/{id}			read		Visualisation d'un utilisateur
/back/utilisateur/{id}/edit			edit	Editer un utilisateur	Affiche / traite le formulaire d'édition
/back/utilisateur/{id}/delete			delete	Supprimer un utilisateur	Supprimer l'utilisateur
/back/commande/			browse	Administration des commandes	Liste des commandes
/back/commande/{id}			read		Visualisation d'une commande
/back/commande/{id}/edit			edit	Editer une commande	Affiche / traite le formulaire d'édition
/back/commande/{id}/delete			delete	Supprimer une commande	Supprimer la commande
/back/label/			browse	Administration des labels	Liste des labels de producteur
/back/label/{id}			read	Visualisation d'un label	Voir un label d'un producteur
/back/label/			add	Ajout d'un label	Crée un label
/back/label/{id}/edit			edit	Modifier d'un label	Modifier un label
/back/label/{id}/delete			delete	Supprimer un label	Supprimer un label
/back/produit/			browse	Administration des produits	Liste des produits
/back/produit/{id}			read		Visualisation d'un produit
/back/produit/{id}/edit			edit	Editer un produit	Affiche / traite le formulaire d'édition
/back/produit/{id}/delete			delete	Supprimer un produit	Supprimer le produit

API V1

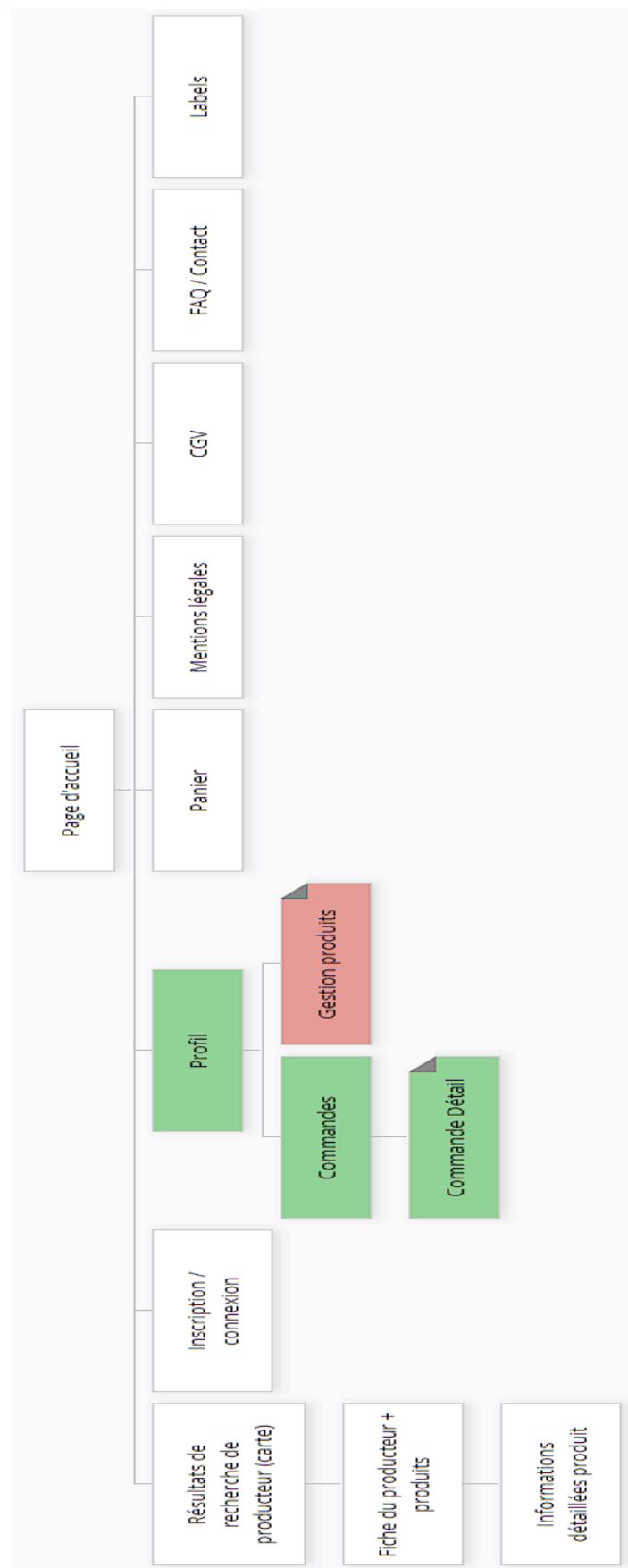
Les controllers de l'API sont dans un sous dossier api

Les URL de l'API vont être versionnée api/V1

endpoint	Méthode HTTP	Contrôleur	Méthode	Commentaire
User				
/api/V1/user/	GET	UserController	browse	Récupérer tous les utilisateurs
/api/V1/user/{id}	GET	UserController	read	Récupérer un utilisateur via son id
/api/V1/user/	POST	UserController	add	Crée un compte utilisateur
/api/V1/user/{id}	PUT , PATCH	UserController	edit	Modifie les infos du compte utilisateur
/api/V1/user/{id}	DELETE	UserController	delete	Supprime le compte utilisateur
Label				
/api/V1/label/	GET	LabelController	browse	Récupérer tous les labels
/api/V1/label/	POST	LabelController	add	Ajoute un label
/api/V1/label/{id}	DELETE	LabelController	delete	Supprime un label
Label producteur				
/api/V1/label/{labelId}/user/{userId}	POST	LabelController	addToUser	Ajoute un label à un producteur
/api/V1/label/{labelId}/user/{userId}	DELETE	LabelController	removeFromUser	Supprime un label à un producteur
Commande				
/api/V1/order/	GET	OrderController	browse	Récupérer toutes les commandes
/api/V1/order/	POST	OrderController	add	Crée une commande associée à un user
/api/V1/order/{id}	GET	OrderController	read	Récupérer une commande via son id
/api/V1/order/{id}	PUT , PATCH	OrderController	edit	Modifie une commande lorsque une ligne de commande est modifiée (à voir)
Commande détail				
/api/V1/order_detail	POST	OrderController	add	Insère une ligne de commande
/api/V1/order_detail/{id}	PUT , PATCH	OrderController	edit	Edite une ligne de commande

Produit				
/api/V1/product	GET	ProductController	browse	Récupérer tous les produits
/api/V1/product	POST	ProductController	add	Créer un produit
/api/V1/product/{user_id}	GET	ProductController	read	Récupérer les produits d'un producteur
/api/V1/product/{user_id}/{id}	GET	ProductController	read	Récupérer un produit via l'id du producteur et l'id produit
/api/V1/product/{user_id}/{id}	PUT , PATCH	ProductController	edit	Editer un produit via l'id du producteur et l'id produit
/api/V1/product/{user_id}/{id}	DELETE	ProductController	delete	Supprimer un produit via l'id du producteur et l'id produit

👉 Arborescence des pages



👉 Exemples de vues de l'application



The image shows two smartphones side-by-side, both displaying a mobile application interface.

Left Phone Screen:

- Header:** Oignons
- Section:** Retrouvez les producteurs autour de chez vous !
- Map:** A map of France showing several red location pins indicating producer locations. A zoom control (+/-) is visible in the top-left corner of the map area.
- Bottom Navigation:** ACCUEIL, INFOS, PANIER

Right Phone Screen:

- Header:** Oignons
- Image:** A large image of yellow onions.
- Section:** Oignons jaunes
- Description:** Des oignons fraîchement récoltés...
- Price:** 2.16 € - Kilos
- Status:** En Stock (green circle)
- Quantity Control:** Buttons for -1+ and a shopping cart icon.
- Bottom Navigation:** ACCUEIL, INFOS, PANIER