Dossier de projet



Le Champ du Voisin

En vue de l’obtention du Titre Professionnel

Développeur Logiciel

Soutenu par

Jonathan Mourier

Tuteur professionnel Benjamin Cordier | Centre de Formation O’clock

Sommaire

**Remerciements p.4**

**Abstract p.5**

**Liste des compétences du référentiel couvertes par le projet p.6**

Développer application Client - Serveur p.6

Développer une application web p.7

**Cahier des charges p.8**

**Spécifications fonctionnelles p.17**

Introduction p.17

Users Stories p.19

**Spécifications techniques p.21**

Versionning et stockage p.21

Technologies p.21

**Réalisation p.23**

Présentation équipe p.23

Rythme p.23

Planification p.24

Déroulement d’un sprint p.24

Outils de communication p.25

**Déroulement du projet p.26**

Mon rôle p.26

Nos réalisations à chaque sprint p.26

**Focus sur mes réalisation p.30**

Documents de conception & Recherches p.30

Code & Fonctionnalités p.32

Test, Correction, Commentaire & Refactorisation p.34

Interface Graphique p.40

Refonte graphique, Test & Correction p.47

**Difficultés rencontrées p.48**

Relationnelles p.48

Techniques p.48

**Conclusion p.49**

**Glossaire p.50**

**Annexes p.52**

Routes p.52

Modèle Conceptuel de Données version 1 p.56

Modèle Conceptuel de Données version 2 p.57

Modèle Logique de Données Final p.58

Dictionnaire de données p.59

Carnet de bord p.61

# 

# Remerciements

Je tiens à remercier en premier lieu toute l’école d’ O’clock qui nous donne la possibilité d’apprendre de chez soi, une expérience personnelle unique où l’on apprend à mieux se connaître.

Je remercie également mes formateurs Julien, Claire, Lucie, JC et les différents Ben pour leur patience et leur pédagogie, qui m’ont permis de m’ouvrir plus sereinement à ces nouveaux horizons et Maylee pour ses courageuses relectures.

Je remercie ma promotion Invaders qui avec son petit nombre reste une promotion avec un très bon niveau me poussant à toujours me remettre en question dans la bonne ambiance.

Je remercie mon équipe qui a vécu avec moi les péripéties de ce projet dans le calme et l’envie d’avancer, me permettant de donner le maximum.

Abstract

Today it’s important to eat quality food. To find this food the best way is the local food with ‘AMAP’s and stores with drive-in services but not a lot of information about local producers.

For these reasons i decided to build an app allowing to directly buy to local producers. As in a supermarket we can choose products, add them to a cart, pay them and have them delivered. The application is the link between consumer and producer. We reduce intermediaries to a minimum.

The first step to turn this idea into an application was to identify our users and their roles. Visitor, Consumer, Producer and Administrator are the 4 roles we defined.

We then defined a Minimum Viable Product (MVP) with the most important functions and parameters of this project.

We had one month on the project, as a team of three O’Clock students : one student working with React and two others working with Symfony.

For the communication between team members we use Slack and Discord.

For product versioning we use Git and GitHub.

We planned our work trying to respect agile principles and SCRUM method. So we worked in week-long sprints, with defined goals to produce faster and adjust our plan over time and results.

The first sprint was mainly dedicated to conception, the second to the technical frame of the application and starting to code, the third to more advanced functionalities and the fourth for tests and debug.

Front-End technology : React

Back-End technologies : PHP with Symfony framework, MySQL database.

Liste des compétences du référentiel couvertes

Introduction

Dans cette partie il vous sera présenté les compétences que nous avons pu déployer pour la réalisation de ce projet.

Développer une application client-serveur

* **Maquetter une application** : Avec l’aide du Product Owner nous avons définis des *wireframes¹* simples (tous les termes sont définis dans le glossaire p. 49) , un logo, ainsi qu’une gamme de couleurs rappelant les valeurs du site.

Nous avons produit et validé les wireframes en équipe.

* **Concevoir une base de données** : En tant que développeur spécialisé en back j’ai pris en charge une bonne partie du travail de conception du modèle de données : du recensement des données à stocker, au Modèle Conceptuel de Données. Le Modèle Logique de Données, qui fait référence à un langage de base de données, a été généré directement par *l’ORM²*que nous avons utilisé : Doctrine.

Vous pourrez trouvez en Annexe 4 la structure de base de données obtenue.

* **Mettre en place une base de données** : La mise en place concrète s’est fait à l’aide de Doctrine et de son outil de gestion en ligne de commande. J’ai également conçu à l’aide de librairies dédiées (*AppFixturesBundle*³ et *Faker*⁴) des jeux de données automatisés. Ce sont des données de test couvrant les cas identifiés dans le cahier de charges et qui sont utilisés en présentation pour la démo de l’application.

**Développer une interface utilisateur** : Nous avons créé l’interface grâce au moteur de template Twig, CSS et dynamisé l’interface à l’aide de JavaScript. J’ai construit les templates de façon hiérarchique pour modulariser la construction des pages. J’ai également utilisé les formulaires proposés par Symfony.

* **Développer des composants d’accès aux données** : J’ai développé les classes de la couche *Model*⁵ de l’application, (soit les classes Entités et Dépôts de Doctrine).

Développer une application web

* **Développer des pages web en lien avec une base de données** : Toutes les pages du site puisent leur variables dans la base de données pour ensuite les restituer à l’utilisateur.

L’application est développée avec le framework Symfony selon l’architecture Model-View-Controler⁶. La mise en place de cette architecture est détaillée dans la description du Sprint 0 page 26.

* **Utiliser l’anglais dans son activité professionnelle en informatique** : La recherche constante d’information sur Symfony m’a plongé dans la documentation Symfony qui est en anglais. J’ai du comprendre les notions mais également effectuer des recoupements entre les différentes parties de code expliquées car la documentation n’est quelquefois pas linéaire.

Le Champ du Voisin

Cahier des charges

# Sommaire

1. Présentation du projet.

Objectifs.

1. Déroulement du projet.

Description des rôles.

Gestion du temps.

1. Fonctionnalités.

Utilisateurs.

Versions.

Navigation.

1. Choix techniques.

# Présentation du projet

### Objectifs

* Le champ du voisin à pour de but de mettre en relation, directement les consommateurs et les producteurs locaux.
* Nous souhaitons simplifier la connexion entre les consommateurs et les producteurs de la région.
* En effet, bien souvent, il est difficile de se fournir en bon produit directement à la source, entre les appels téléphoniques à répétition et les déplacements dans toutes les fermes avoisinantes, bien nombreux sont ceux qui abandonnent par manque de temps.
* C'est là que, le champ du voisin intervient, nous offrons de la visibilité aux producteurs locaux sur leurs produits, tarifs, nous leur assurons un système de paiement sécurisé et fiable.
* Un gain de temps conséquent aux consommateurs pas d'appels, de déplacement inopportun, un large choix de produits issus du territoire français.
* Au niveau de la réception des commandes pour le consommateur, il peut directement les récupérer chez le producteur.
* Un véritable circuit-court 2.0.

# 

# Déroulement du projet

Le délai étant de 21 jours de développement , une version “Minimum Viable Product” (MVP) permettant de tester les fonctionnalités principales sera mis en place. Il s’agira de pouvoir rapidement tester l’application pour savoir si elle est viable et évaluer plus facilement les problèmes potentiels à venir.

Plusieurs sprints de développement successifs permettront de mettre en place les différents aspects de l’application et de commencer à les tester au fur et à mesure.

**Sprint 0 : Organisation.**

Durant cette phase nous définirons nos rôles, notre cahier des charges, nos objectifs puis nos documents de conception comme les wireframes, dictionnaire de données, les routes, ect…

**Sprint 1 : Mise en place de la structure du site.**

Durant cette phase nous commencerons à coder la structure du site et principalement la partie back-end et quelques composants de l’interface.

**Sprint 2 : Mise en place des fonctionnalités avancés.**

Cette phase sera consacrée à la consolidation du code, la connexion entre les différents éléments et l’ajout des dernières fonctionnalités.

**Sprint 3 : Tests fonctionnelles et refactorisation.**

Durant celle-ci nous ferons du test fonctionnel notamment en effectuant le

parcours d’un utilisateur.

### 

### Description des Rôles

L’équipe se compose de :

* Remy de spécialité React : 22 ans, élève d’une précédente promotion et ayant déjà développé un projet full stack auparavant, nous utilisons son retour d’expérience tout au long du projet notamment sur Git et la mise en forme avec CSS.

* Pierre-Alain de spécialité Symfony : 24 ans avec des études dans le domaine de la psychologie et le comportement humain nous apporte une vision tournée autour de l’utilisateur, ce qui nous a aidé à être critique vis-à-vis de notre code.
* Jonathan de spécialité Symfony : 28 ans , étant le product owner a une vision plus précise du projet et propose les idées et les actions à mener afin d’avancer dans le projet.

Nous avons défini des rôles afin que chacun soit responsable d’une partie et acteur du projet :

* Rémy de par son précédent projet a plus d’expérience que nous avec l’outil très important qui est Git, nous lui attribuons donc les rôles de Git master & Lead Front.
* Jonathan est Product Owner et comme il a une appétence pour les documents de gestion de projet il se propose pour le Lead Back.
* Pierre-Alain sera l’élément qui gardera du recul et une vision d’ensemble sur l'application et les tâches à effectuées, il sera Product Manager.

### 

### Gestion du temps

Nous nous inspirons des méthodes agiles⁷ et du SCRUM notamment avec la mise en place de sprints qui rythme cette période de développement.

Les événements prévus afin de respecter le SCRUM sont le **découpage du temps en** **sprint**,

les **daily SCRUM** qui sont des réunions journalières et rapides où chacun indique l’avancement de son travail et les difficultés rencontrées,

la **réunion de planification du sprint** qui pour notre part s'effectue à chaque début de sprint,

la **revue de sprint** qui est normalement faite avec le client qui permet de valider le travail effectuée dans notre cas les points de présentation effectués chaque semaine devant la promotion,

puis la **rétrospective du sprint** qui elle est faite en interne afin de tirer une leçon du sprint précédent et ainsi améliorer le travail d’équipe, nous l’avons effectué via les points faits avec nos référents ce qui a notamment permis de redistribuer les ressources et de prendre de bonnes décisions.

# 

# 

# Fonctionnalités

### Utilisateurs

Afin de définir les fonctionnalités nécessaires pour l'application nous avons défini les rôles des utilisateurs

* **Visiteur** : Personne ne s’étant pas connectée qui souhaite voir les produits et producteurs présents sur le site.
* **Consommateur** : Personne étant connectée qui souhaite acheter des produits.
* **Agriculteur** : Personne étant connectée pouvant proposer des produits.

* **Admin** : Personne étant connectée ayant tous les droits sur les données du site.

Voici ci-contre toutes les fonctionnalités du produit final regroupé par versions.

Version 1 - Minimum Viable Product

Cette version est la plus simple possible de l’application. Voici les fonctionnalités proposées :

* Afficher la liste des produits et producteurs.
* Connexion et Inscription.
* Envoie de mail via Contact à l’administrateur.
* Voir fiche produits et fiche agriculteurs.
* Modifier son profil.
* Créer, modifier, supprimer un produit.
* Historique de commande.
* Système de panier.
* Back office pour l’admin.
* Valider une commande.
* Gestion des rôles par l’application.

Version 1.1 (cette version représente la version initialement prévu en Sprint 0)

* Commander le panier en drive ou en livraison.
* Laisser un commentaire sur l’agriculteur.
* Echange de service.
* Bon de réduction.
* Newsletter.
* Vente flash.
* Envoyer des messages.
* Noter l’agriculteur.
* Paiement en ligne.

Version 2

* statistiques du site (nombre de visiteurs, achat, paniers en attente, paniers refusés ect...).
* liste des produits et tri par localisation.
* liste des producteurs et tri par localisation.

Version 3

* outils d’aide à la livraison permettant aux producteurs d’avoir un itinéraire calculé. en direct suivant les commandes qu’il a en attente.

Version 4

* outils de gestion de stock comme par exemple des prévisionnels sur les récoltes ou encore sur les ventes d’après les données du site.
* outils d’aide à la production en partenariat avec des gens du métiers afin de faciliter la culture.
* outils d’aide à la comptabilité afin par exemple d’avoir une idée rapide des bénéfices qu’il peut tirer de tel ou tel commandes, prenant en compte le prix de revient, le coût de la livraison ect...

### 

### 

### 

### Navigation

**Visiteur**

* Produits par catégorie.
* Inscription.
* Connexion.

**Consommateur**

* Liste Produits.
* Profil.
* Logout.

**Agriculteur**

* Mes produits.
* Liste Produits.
* Mes ordres de paiement.
* Mes commandes.
* Logout.

**Admin**

* Back-office.
* Logout.

# 

# Choix techniques

Nos ressources sont constituées de 2 personnes back-end spécialisées Symfony et une personne front-end spécialisé React.

Nous avons donc naturellement choisi pour l’application d’utiliser Symfony 4 pour le routage et la base de données et de React pour la partie front-end.

La problématique de la connexion entre ces 2 technologies sera l’objet de nos premières recherches.

Partie Back-end nous utiliserons

**PHP 7.2** - la version la plus récente nécessaire pour Symfony 4.

**Composer** - qui est l’installateur de dépendances PHP.

**SensioFrameworkBundle** - le coeur de Symfony grâce auquelle on accède aux principales fonctions du framework.

**Doctrine** - ORM qui permet de travailler avec la base de données en objet.

**MySQL** - qui sera notre base de données en langage SQL.

**Twig** - Moteur de template.

**EasyAdminBundle** - permet de mettre en place un back-office rapidement et simplement.

**SwiftMailer** - permet l’envoie de mail.

**FixturesBundle** - Librairie Symfony qui facilite la création de fausses données.

**Faker** - Librairie qui génère des données de test proche de la réalité.

Partie Front-end nous utiliserons

**Node.js** - certain modules sont nécessaires pour React.

**Yarn** - qui est l’installateur de dépendance node.js.

**Webpack Encore** - qui est un module bundler⁸ simplifié.

Spécifications fonctionnelles

Introduction

Dans un premier temps avec l’aide du Product Owner nous avons défini les objectifs du site ainsi que les types d’utilisateurs. Il a fallu réfléchir à quelles fonctionnalités nous voulions implanter dans l’application. C’est une étape importante car de celle-ci découle tous les wireframes avec les zones d’interaction ainsi que la structure de la base de donnée.

Lors de la gestion du projet il a fallu prioriser les fonctionnalités afin d’en dégager un Minimum Viable Product qui a été l’objet de ce mois de projet.

Nous avons définis 4 familles d’utilisateurs

* **Visiteur** , c’est l’utilisateur qui vient sans se connecter, a donc aucun droit mais veut se renseigner sur les produits présents. L’idéal serait de lui permettre de pouvoir commander sans s’inscrire.
* **Consommateur** , c’est l’utilisateur qui s’est connecté afin d’avoir un suivi de ses données de profil et de commande.
* **Producteur** , c’est l’utilisateur qui est inscrit pour vendre ses produits.
* **Admin** , c’est la personne qui a tous les droits pour modérer un commentaire ou modifier des données.

A l’aide de ces profils nous avons pu hiérarchiser les fonctionnalités

**Visiteur**

**Pages Liste produits & producteurs** → permet au client de naviguer

dans le panel de produits et de

producteur sans inscription. Il peut y

voir

les quantités disponibles.

**Page Contact** → permet d’envoyer un mail à l’administrateur via un formulaire

de contact.

**Page Inscription** → permet de s’inscrire. Tout d’abord un premier formulaire

où l’on s’inscrit en tant que personne puis un second

formulaire obligatoire où l’on enregistre son adresse.

**Page Connexion** → permet de se connecter.

**Consommateur**

**Page Mes commandes** → accéder à son historique de commande.

**Page Mon profil** → pouvoir modifier mon profil.

**Page Mon panier** → modifier, supprimer son panier ou passer à l’étape

du paiement.

**Producteur**

**Page Mes produits** → pour un producteur pouvoir gérer son stock et produits

disponibles.

**Page Mes ordres de paiement** → permettre au producteur d’effectuer des

demandes pour récupérer son solde.

**Admin**

**Page Back-Office** → pouvoir gérer toutes les données

du site.

Users Stories

Voici les Users Stories le document qui nous a servi pour définir les fonctionnalités et les routes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **V.** | **ID** | **Thème** | **En tant que** | **J’ai besoin de pouvoir ...** | **Afin de ...** |
| 1 | 0 | Présentation | Visiteur | Me rendre sur la page d'accueil | Découvrir les fonctionnalités proposées |
| 1 | 1 | Présentation | Visiteur | Consulter la liste des agriculteurs | Connaître les agriculteurs présents |
| 1 | 2 | Présentation | Visiteur | Consulter la liste des produits | Connaître les produits proposés |
| 1 | 3 | Présentation | Visiteur | M’inscrire | Commander des produits |
| 1 | 4 | Présentation | Visiteur | Envoyer un mail à l’admin | poser une question , une réclamation |
| 1 | 5 | Achat | Utilisateur | Mettre des produits dans le panier | Commander des produits |
| 1 | 6 | Achat | Utilisateur | Payer un panier | Recevoir les produits |
| 1 | 7 | Achat | Utilisateur | Choisir le mode de livraison | Optimiser ma livraison |
| 1 | 8 | Achat | Utilisateur | Voir le récapitulatif de ma commande | Faciliter mon expérience d’achat |
| 1 | 9 | Gestion de données | Utilisateur | Modifier, supprimer mes données personnelles | Gérer mes données personnelles |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | Gestion de données | Producteur | Créer,Modifier,Supprimer mes produits | Gérer mon offre |
| 1 | 11 | Gestion de données | Producteur | Lister mes commandes | Livrer ses produits |
| 1 | 12 | Gestion de données | Admin | Accéder à un back office | Gérer les données du site |
| 1 | 13 | Sécurité | Utilisateur | Me connecter | Accéder aux fonctionnalités du site |
| 1 | 14 | Sécurité | Utilisateur | Me deconnecter | Supprimer le cookie de session |
| 2 | 15 | Fonctionnalité | Utilisateur | Envoyer un message | Communiquer entre utilisateur |
| 2 | 16 | Fonctionnalité | Utilisateur | Poster des commentaires | Améliorer les services |
| 2 | 17 | Fonctionnalité | Utilisateur | Avoir des producteurs favoris | Retrouver facilement mes producteurs favoris |
| 2 | 18 | Fonctionnalité | Admin | Créer,Modifier,Supprimer des réductions | Prospecter des clients |
| 2 | 19 | Fonctionnalité | Admin | Créer des newsletters | Prospecter des clients |

Spécifications techniques

Versionning et stockage

Pour nous permettre une mise en commun du code et un versionningpermanent nous avons utilisé Git et le site GitHub comme dépôt distant.

Cette outil de contrôle de version est devenu l’outil indispensable du développeur car il permet de travailler plus efficacement en équipe et d’avoir une persistance du code.

Nous avons dans un premier temps organisés l’arborescence en 4 branches.

* une branche **Master :** sur cette branche se trouve l’application finale que l’on *push⁹* une fois que la version en développement est correcte.
* une branche **Dev** : sur laquelle se trouve toute les fonctionnalités back-end et front-end qui fonctionnent que l’on *merge¹⁰* à partir des branches back et front afin de vérifier qu’elle fonctionnent ensemble.

* une branche **Front** : sur laquelle notre Lead Front dépose ses fonctionnalités.
* une branche **Back** : commune aux 2 développeurs back.

Cette architecture fut amenée à changer en fin de projet au vu de nos changements techniques nous avons finalement tout centré sur la branche **Dev**.

**Technologies**

**Front** :

* **Yarn** qui permet l’installation de paquets nécessaires au développement ou aux fonctionnements de l’application.
* **Webpack Encore** qui est une version simplifiée: de Webpack permettant la compilation de l’ES6 vers des versions antérieurs.
* **Node.js** car React nécessite certaines de ses modules pour fonctionner.

**Back** :

* **PHP 7.2** qui est la version la plus récente nécessaire pour le fonctionnement de Symfony 4.
* **Composer** qui est l’installateur de paquets PHP.
* **SensioFrameworkBundle** qui est le bundle de base utilisé par le framework Symfony permettant des fonctionnalités comme le modèle MVC, routage, les formulaires ect...
* **Doctrine** qui est utilisé pour transformer la base de données en objet et ainsi faciliter la sauvegarde, mise à jour et maintenance de celle-ci.
* **MySQL** est la base de données que nous avons utilisé pour ce projet.
* **Twig** qui est un moteur de template.
* **EasyAdminBundle** qui nous a permis de facilement mettre en place un back-office.
* **SwiftMailer** est un bundle facilitant l’envoi de mail.
* **FixturesBundle** est un bundle utilisé pour générer des fausses données.
* **Faker** est une dépendance qui permet d’utiliser de façon aléatoire des données complexes proche de la réalité comme le téléphone, les codes postaux, ect….. .

Réalisations

Présentation de l’équipe

Avant de commencer à expliquer plus en détails comment nous avons travailler laissez moi vous présenter les membres de mon équipe.

Jonathan - Product Owner & Lead Back - De nature curieuse et motivé j’ai toujours de nombreuses idées ce qui m’a permis de facilement en proposer une pour les projets. Appréciant particulièrement la conception de projet j’ai naturellement pris le rôle de Lead Back et produit les documents de conception nécessaires.

Remi - Lead Front & Git Master - Ayant déjà effectué un projet un peu plus innovant Rémi nous a apporté un plus pour la gestion de projet avec Git et ses astuces front-end notamment en CSS.

Pierre-Alain - Project Manager - De nature posée Pierre-Alain a su nous poser les bonnes questions et proposer des idées pour avancer. Certainement de par son cursus en psychologie il a pu garder un oeil attentif à l’expérience utilisateur.

Rythme

Nous nous sommes fixé un rythme théorique de 9h-17h du lundi au vendredi soit 7h30 par jour avec une pause d’1/2 heure. Rythme qui a pu se modifier suivant les impératifs de chacun. Et même quelques week end y sont passés.

Le matin nous nous retrouvions pour une réunion surnommé daily scrum où chacun expliquait ses problématiques et ses résultats et l’on fixait les prochains objectifs.

Planification

Afin de planifier à l’avance chacun des sprints nous avons utilisé le site Trello qui est un site de *Kanban¹¹* permettant la création et le déplacement de note. Nous avons défini pour chaque sprint des tâches à effectuer en leur affectant un état (todo, current, done).

Cette méthode a été très efficace car cela nous a permis de toujours savoir où nous en étions, ce qui nous restait à faire.

C’est un point qui me semble impératif dans le développement d’un projet. Nous n’étions jamais perdu dans nos tâches.

Cela nous a également permis de nous coordonner et d’avoir une vision d’ensemble, ce qui est motivant je trouve car nous avons une vue sur le projet en entier et non juste sur une fonctionnalité.

Déroulement d’un sprint

Ayant seulement moins de 4 semaines nous avons dû modifier les délais des sprints préconisés par la méthode SCRUM. Chaque sprint dure 1 semaine et se finalise par la présentation de nos avancées à la promotion correspondant à la revue de sprint.

Nous avions une mise au point obligatoire avec notre référent comme un rendez vous de consulting ce qui équivalait à la rétrospective de sprint. Ce rendez-vous nous a permis au premier sprint de recalibrer la charge de travail car nous avions beaucoup trop de fonctionnalités et nous n’aurions pas pu finir notre MVP.

Chaque Sprint commence par la distribution des tâches prévues.

Puis chacun crée ses fonctionnalités.

Une fois fonctionnelles nous les ajoutons sur le dépôt GitHub dans la branche Dev et chacun les récupère afin de valider son fonctionnement sur chaque poste.

Quand nous avons validé la fonctionnalité nous la rajoutons dans la branche master.

Outils de communication

Afin de communiquer au mieux entre nous nous avons utilisé un panel d’outils

* Slack : ce canal était utilisé pour toute nos communications asynchrones (souvent suivant l’heure de la communication) ou comme support persistant pour des liens ou documents.

* Discord : utilisé en vocal pour nos réunions ou nos discussions en temps réel sur le projet.
* Google Drive et Google Doc : un dossier partagé regroupe toute nos réalisation de document principalement de conception ainsi que notre journal d’équipe (qui nous permet d’avoir un recul sur notre travail) et nos carnets de bord (permettant à chacun de noter son travail, ses recherches, ses idées ect).

Très utile pour être certain de ne pas avoir de problème de format, avoir une sauvegarde permanente des dossiers et avoir tous le temps la dernière version du document car il est mis à jour en direct.

* Git et GitHub : l’outil de versionning indispensable qui nous a permis de mettre en commun notre code.
* Atom : Environnement de développement que j’ai utilisé.

Déroulement du projet

Mon rôle

En tant que Product Owner j’ai essayé au maximum de proposer des idées claires et je prend en compte chacune des remarques pour améliorer l'expérience utilisateur ou la lisibilité du code.

En tant que Lead Back ma responsabilité était de produire des documents de conception de qualité ainsi que superviser le travail effectué en back-end.

Nos réalisations

***Sprint 0***

* **Objectifs**

Concevoir le cahier des charges et le documents de gestion de projet.

Recherche sur la communication React/Symfony.

* **Durée 4 jours**

C’est pour moi le sprint le plus important car il permet de gagner énormément de temps en concevant correctement l’application dès le début, c’est pour cela que la phase de conception nous a pris 4 jours environ afin de produire des documents les plus complets et clairs possible. En plus je suis le Lead Back donc responsable de la conception de la base de données et de ses composants d’accès.

Durant ce sprint nous avons produit le cahier des charge, le dictionnaire de donnée, les routes, le Modèle Conceptuel de Données.

J’ai également aidé Remi pour l'intégration de Webpack Encore car ne connaissant pas Symfony il lui a fallu des explications sur le fonctionnement de Symfony.

Suite à la rétrospective de sprint nos référents nous ont conseillés de réduire les ambitions et de prévoir moins de fonctionnalités car cela serait impossible de les tenir dans nos délais. Cela a impliqué une refonte partielle de tous nos documents. C’est une décision important prise dès le départ car nous aurions pu être submergé et découragé par l’ampleur de la tâche sans cela.

***Sprint 1***

* **Objectifs**

Compléter les documents de communication format JSON.

Mettre en place la base de donnée.

Coder les premières fonctionnalités et interface.

* **Durée 6 jours**

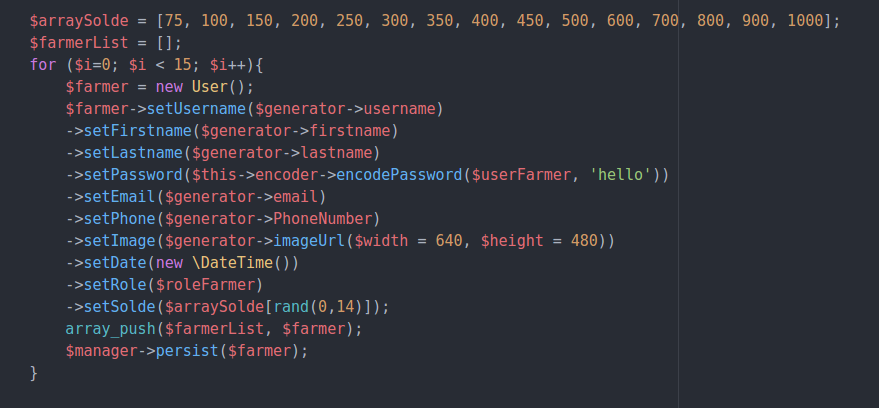
Sur la base des documents de conception Pierre Alain et moi avons créer la base de données ensemble grâce à Doctrine (pour éviter tout problème) et ensuite mettre en place les contrôleurs afin d’effectuer le traitement et l’envoi des données sous format JSON.

Les fixtures sont également créées afin de pouvoir intégrer des données de test dans notre base de données. Ce sprint se passe sans réelle problème technique car nous faisions du code que nous connaissions.

**Code source instanciation de l’objet pour les fixtures**



**Exemple de boucle qui crée des objets User**



**Explication du code**

Dans un premier temps nous installons Faker et AppFixturesBundle puis nous créons son instance dans la classe AppFixtures de Symfony. On fait un appel statique de la classe Factory utilisant la méthode create en arguments fr\_FR qui permet d’utiliser les données francaise pré-écrite par faker.

fr\_FR est un code ISO des langues, le premier fr correspondant à langue et le deuxième correspondant au pays.

L’exemple dessous montre une boucle permettant de générer les faux utilisateurs.

Le premier tableau correspondant à des soldes aléatoires.

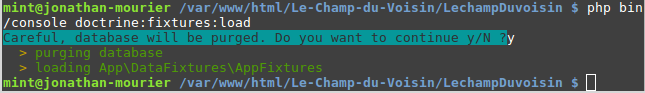
Le second à celui réunissant tous les objets User crées.

Puis avec les *mutateurs*¹² nous modifions les champs de notre nouvel objet User grâce à l’objet generator, nous l’ajoutons aux tableaux puis nous utilisons la méthode de l’EntityManagerInterface persist pour que cela soit validé par Doctrine.

Ceci est fait pour toute les fausses données que l’on veut créer puis nous utilisons la méthode flush qui permet d’effectuer réellement l’enregistrement des données dans la base de donnée.

Symfony nous permet de générer ces données à volonté grâce à une commande, ce qui est vraiment un gain de temps pour la mise en place de version de démonstration. Il nous faut coder les fixtures une seule fois pour notre base de donnée.

Le seul inconvénient peut être que la base de données sera purgée à chaque fois, ce qui serait peut être gênant si nous voulons intégrer des valeurs de test sur une base de données déjà en production.



***Sprint 2***

* **Objectifs**

Mise en place des interfaces.

Mise en place des formulaires Symfony & des contraintes.

Finition des fonctionnalités.

* **Durée 6 jours**

Suite à perte de temps et recherche nous changeons de cap en Front et décidons d’utiliser Twig avec seulement quelque composants React, ce qui correspond peut être mieux à nos ressources.

Suite à cela nous devons modifier tous nos controllers car nous avions prévu de renvoyer du JSON et pas des rendus de templates Twig.

Pour l’organisation nous nous divisons les routes et chacun code la fonctionnalité correspondante.

***Sprint 3***

* **Objectifs**

Revue du code & debug.

Mise en ligne.

* **Durée 3 jours**

Ce sprint voit la finalisation du design ainsi qu’un travail un peu plus fin sur les interfaces utilisateurs et les formulaires. Pour cela nous avons effectué des tests fonctionnels où nous imitons le comportement de chaque utilisateurs pour trouver les éventuels bugs ou améliorations possible. Nous passons également du temps pour organiser la présentation finale face à notre promotion.

Focus sur mes réalisations

Durant l’élaboration de ce dossier, j’ai pu vous faire part des différentes étapes de construction de notre application, néanmoins je ne suis pas rentré dans les détails de mon travail afin de pouvoir l’isoler dans cette partie.

*1er étape - Les documents de conception & recherches*

Dans un premier temps étant le Product Owner et le Lead back j’ai rédigé les documents de conceptions ci-contre en gardant bien en vue que la concision et la précision de ces documents permettra une bonne répartition du travail.

Une fois le cahier des charges rédigé ensemble je me suis attelé au dictionnaire de données en listant toute les données nécessaires aux fonctionnalités. Puis je leur ai attribué un type et une entité (table).

Cela m’a permis de créer le Modèle Logique de Données qui un modèle entité-association illustrant les relations entre les différentes entités. Ce document est la base de la création d’une base de donnée.

Ensuite j’ai rédigé un tableau regroupant toute les routes possibles ainsi que leur url et le contrôleurs auxquels elles sont rattachées.

**Vous trouverez en annexe le cahier des charges, les routes, le**

**dictionnaires de donnée ainsi que les MCD pour les différentes versions.**

Il est convenu que Symfony s’occupera de toute la partie base de données et routage et React toute l’interface graphique.

Utiliser Symfony a l’avantage de pouvoir intégrer une architecture de dossier de type MVC ‘modèle-view-controller’ de façon simple et efficace. Cette architecture présente de nombreux avantages.

Tout d’abord en terme de développement grâce à une séparation des concepts qui permet d’organiser notre code de façon structurée et l’interchangeabilité de ses composants.

Il utilise également l’*url rewriting¹³*.

L’outil de debug de Symfony est aussi très utile car il est détaillé et permet même quelque fois d’orienter sur l’erreur commise.

Nos recherches sur la communication entre React et Symfony est une des priorités de ce premier sprint voici le résultat de nos recherches.

*Suite à nos recherches nous avons trouver 2 solutions techniques possible*

* utiliser Symfony et React indépendamment. React consommant l’*API*¹⁴ fonctionnant sous Symfony aux travers d’appels *Ajax¹⁵*.
* intégrer React dans l’arborescence Symfony à l’aide de WEBPACK ENCORE version simplifié de WEBPACK.

Nous avons retenu la 2e solution car cela permettrait de nous partager un seul et même dossier et l’API a peu d'intérêt car le site n’est pas destiné à être utilisé par plusieurs applications front.

*Dans cette solution technique il y a encore plusieurs choix*

* Nous plaçons les variables dans des éléments HTML en utilisant l’attribut *data-id¹⁶* à travers les templates.

Choix non retenue car cela nous ferait écrire beaucoup de code et créer des éléments HTML inutile.

* Nous plaçons nos variables à travers les balises scripts du template.

Choix non retenue car cela nous fait travailler intervenir dans le contrôleur et dans la vue.

* Nous plaçons une unique div d’id ‘root’ dans le template de layout et React l’utilise pour placer tous les composants graphiques, les données dont il aura besoin seront disponible via des appels AJAX.

Nous avons retenu la dernière solution car la plus propre. Coté Symfony il est simplement nécessaire de renvoyer une réponse JSON et côté React de faire des appels AJAX. La principale difficulté à anticiper sera de bien standardiser les noms des variables pour les échanges. Cela se rapproche d’un fonctionnement API.

J’effectue également l’installation de Symfony 4 sur notre dépôt afin que chacun ait la même version et architecture de base.

J’ai également aidé notre dev front car il n’arrivait pas à intégrer React en le guidant à travers Symfony, je me suis rendu compte à ce moment là de l’importance d’aller vers les membres de son équipe car ce qui est évident pour nous ne l’est pas pour tout le monde.

Le souci technique provenait des paramètres de configuration de Webpack Encore qui ne compilait pas les fichiers CSS.

A noter qu’à cette étape du projet nous avions essayer de l’imaginer le plus complet possible, nos référents nous ont conseillés de réduire le nombre de fonctionnalités car le délai était à priori trop court, cela a aboutit à des modifications afin de produire un MVP plus simple.

2e étape - Code des fonctionnalités

Sur la base des documents créés en Sprint 0, nous créons la base de donnée à l’aide de Doctrine.

Grâce à Symfony cela est simplifié grâce à la création d’entité (les tables) et de propriétés (les entrées) en ligne de commande. Et rajoutez à cela Doctrine gère également les relations!

L’automatisation par la ligne de commande n’exclut pas de savoir ce qui se passe derrière mais permet d’éviter des erreurs d’inattention et fait gagner énormément de temps.

Quand nous créons une entité cela génère dans le dossier Entity un fichier avec une classe du nom de l’entité qu’un fichier dans le dossier Repository associé à cette entité.

Le fichier repository est celui qui regroupe les appels pour la base de donnée.

Lorsque nous créons un champ dans une table nous créons une propriété dans une entité. Lors de la création Doctrine met également en place les accesseurs et mutateurs.

Ces méthodes dans la classe nous permettent de pouvoir installer des contrôles si besoin, les propriétés étant protected ou private. C’est une bonne pratique de les utiliser car sans cela les propriétés serait public et sans contrôle sur leur modification.

Pour une relation One-To-Many Doctrine va créer un tableau dans l’entité qui peut avoir plusieurs relations et pour l’entité qui ne peut avoir qu’une relation unique, une simple propriété sera créée.

Pour une relation Many-To-Many il créera un tableau dans chacune des entités.

Cela ne n’exclut pas de vérifier grâce à PHPMyAdmin si la base de données est créée correctement et si le modèle produit correspond au MCD.

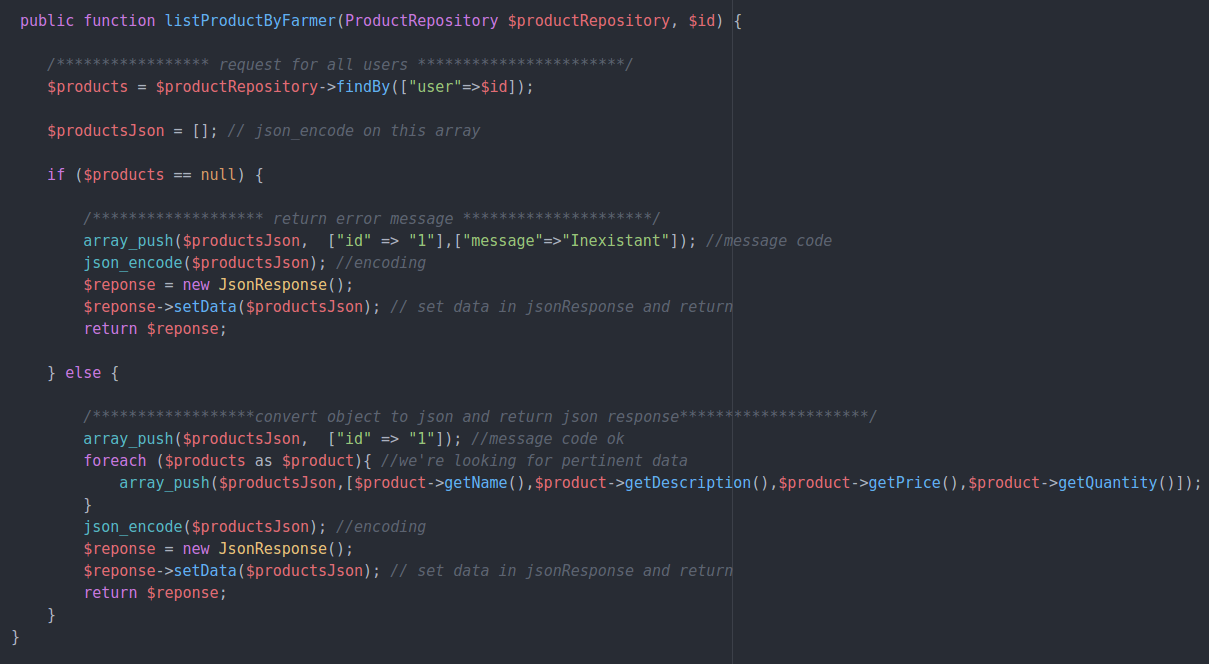
De la même manière nous créons les contrôleurs. Nous avons créé un contrôleur par entité, chaque méthode du contrôleur représente une route. Nous pouvons configurer les routes soit par des commentaires appelés ‘annotation’ soit dans un fichier yaml. Symfony crée également un fichier twig correspondant.

Enfin nous commençons chacun à coder les fonctionnalités pour chaque routes. La démarche consistait pour chaque route à faire la requête dans la base de données suivant les besoins et renvoyer les données sous format JSON.

Nous avons choisi le format JSON car c’est un format de donnée structuré très simple à manipuler pour React.

J’ai utiliser la méthode native de PHP qui encode un tableau en json (afin d'être certain de notre format) puis celle qui permet de renvoyer une réponse JSON.

**Exemple du contrôleur qui renvoie la liste des produits par producteur**



**Explication du code**

Nous voyons que cette méthode accepte des arguments typés. Cela nous permet une fonctionnalité de Symfony appelée injonction de dépendance qui nous permet de faciliter notre code en évitant de créer une instance de l’objet. Dans ce cas on remarque que le $productRepository n’a pas été instancié.

Nous stockons le résultat de la recherche en base de donnée.

Nous déclarons un tableau vide qui nous permettra d’y insérer les données à retourner.

Puis nous entrons dans une condition qui nous permet de gérer le cas de figure où il n’y a pas de produits en base de données qui correspondent à la recherche afin d’éviter l’affichage d’une erreur disgracieuse.

Si les produits n’existent pas nous insérons des paramètres que nous avons prédéfini dans le tableau. Nous transformons le tableau au format JSON grâce à une fonction native de PHP. Puis nous l’encapsulons dans un objet de type Response.

Si les produits existent, nous insérons les données recherchées en parcourant les objets grâce à une boucle puis nous transformons le tableau au format JSON et nous l’encapsulons dans un objet Response.

A l’écriture de ce code je voyais déjà quelques pistes d’amélioration comme la factorisation de la partie qui renvoie les donnée au format JSON.

Suite à cela voulant anticiper au maximum les problèmes de communications j’ai ajouté dans le tableau des routes chaque réponse JSON que l’on devait renvoyer. Cela m’a pris quelque heures et je me suis rendu compte que des fois il ne faut pas trop anticiper car cela correspond à de la sur-qualité.

*3e étape - Test, correction, commentaire & Refactorisation*

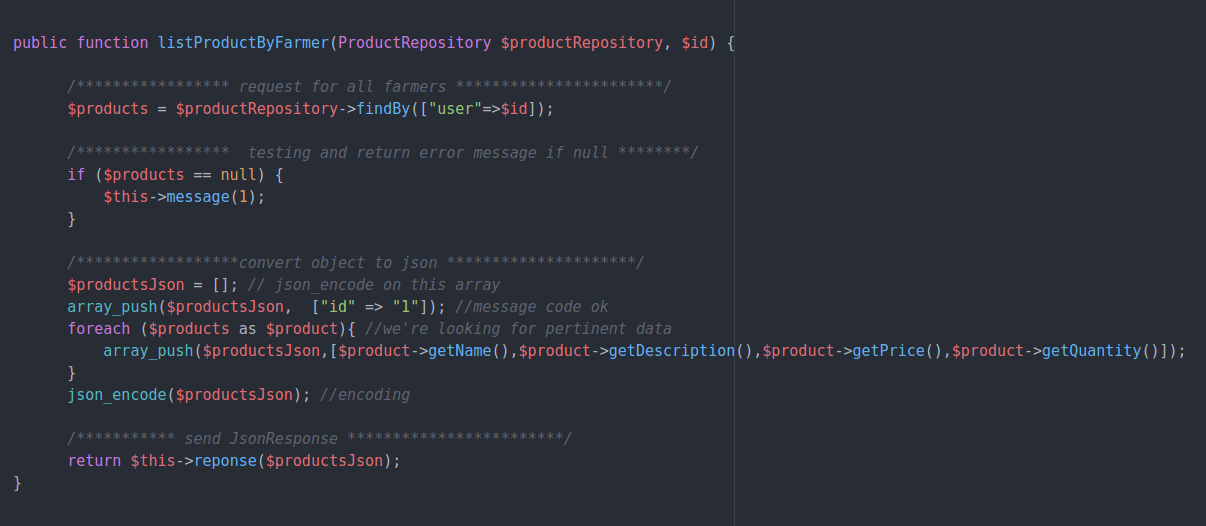
Une fois fini cette tâche et la partie front n’étant pas assez avancée pour nous

connecter j’ai refactoriser le code pour en améliorer la lisibilité.

Dans un premier temps j’ai modifié l’abstract class Controller dans les fichiers de Symfony mais je pense n’étant pas une bonne pratique car le fichier ‘vendor’ dans laquelle sont les fichiers du framework ne sont pas mis sur les dépôts Git de façon à ne pas télécharger des données inutiles donc j’ai créer une abstract class intermédiaire mais il possible que l’on puisse encore optimiser cela.

**Exemple du contrôleur qui renvoie la liste des produits par producteur**

**suite à refactorisation**



**Explication du code**

Ce code a exactement le même comportement.

De manière identique à la version précédente la méthode commence par une recherche en base de données qu’elle stocke dans une variable.

Puis s’il n’y a pas d’objet nous renvoyons les même données que précédemment en utilisant une méthode qui est défini dans une abstract class Custom dont hérite notre contrôleur. Cela a le mérite de simplifier le code mais nous oblige de concevoir un document afin que les autres membres ou les futurs développeurs travaillant sur ce projet puissent réutiliser les mêmes codes.

La deuxième amélioration consiste à instancier l’objet JsonResponse et y encapsuler les données à envoyer dans la méthode reponse de mon abstract class Custom.

Arrivé à ce point du projet notre partie front ayant de grande difficulté nous avons tous ensemble fais le point sur la partie React prenant le temps de bien comprendre les difficultés, notre front nous donnant des explications sur le fonctionnement de React mais nous aboutissons à la conclusion qu’avec le retard que nous avions pris et les ressources humaines nous ne pourrions afficher quelque chose si nous ne changions pas de choix technique. Nous avons décidé de coupler le template twig avec React.

Suite à la décision de changer de stratégie en utilisant Twig je me charge de reprendre les controllers afin de retourner un template Twig et non plus une réponse JSON.

**Exemple du contrôleur qui renvoie la liste des produits par producteur**

**suite au passage à Twig**

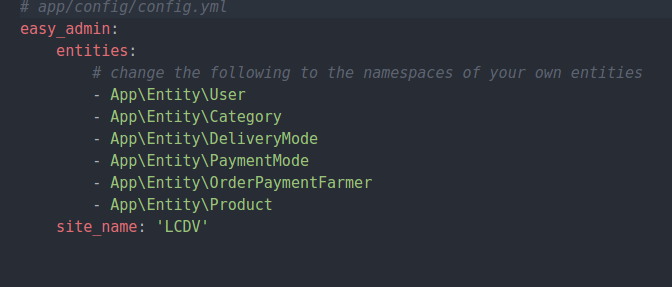


**Explication du code**

Cette version exécute une demande en base et envoie les données au template Twig.

Pour la création du back-office afin de gagner du temps j’ai installer le bundle Symfony EasyAdminBundle qui permet très rapidement d’avoir une interface d’administration simple mais efficace.

**Fichier de configuration permettant de personnaliser l’interface**



**Explication de code**

Ce fichier nous permet de configurer le back office en y ajoutant les entités que l’on veut pouvoir modifier.

Une des dernières fonctionnalités que j’ai produite est l’envoi de mail via le formulaire de contact. Pour cela j’ai utilisé le bundle de Symfony SwiftMailer que je ne connaissais pas. La documentation étant bien faite j’ai eu peu de souci.

**Méthode qui gère l’envoi d’email**

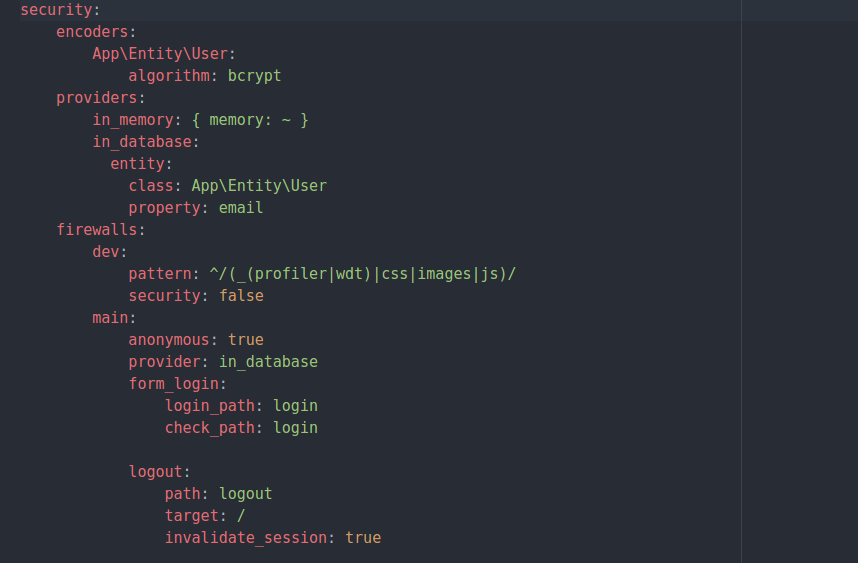


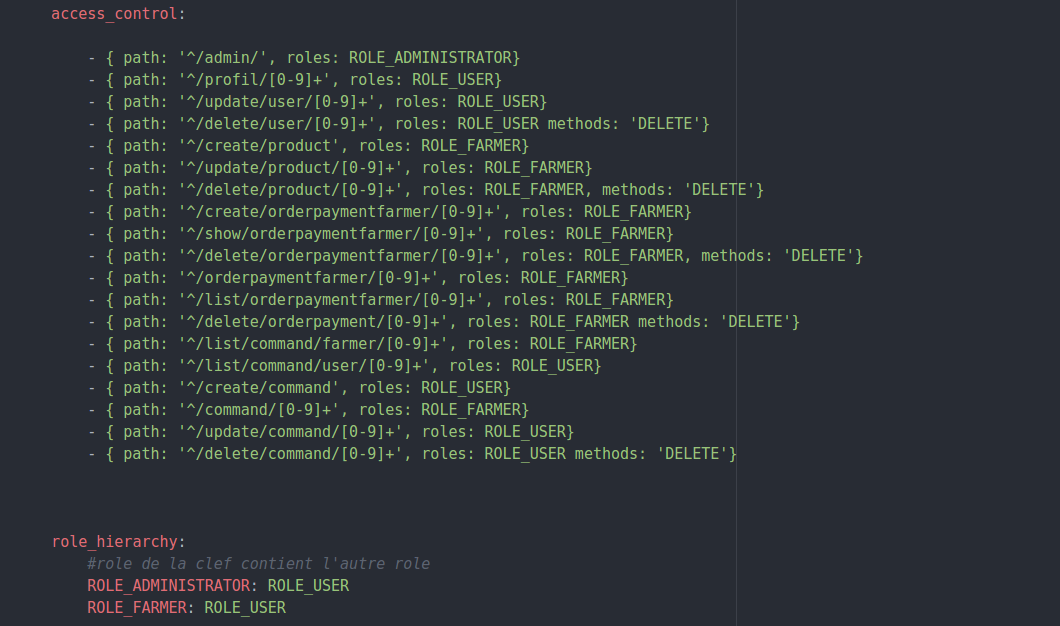
**Explication du code**

En suivant le code trouvé dans la documentation voici la méthode qui instancie un objet Swift\_Message puis intègre un body, une adresse d’envoi, de destination et un en-tête. Puis l’email est envoyé.

Un flash message est enregistré pour indiquer à l’utilisateur que son message a bien été réceptionné puis redirige vers le formulaire de contact où l’information que le message a bien été envoyé ne s'affiche qu’une seule fois même en cas de rafraîchissement de la page.

**Code source du fichier security.yaml**





**Explication du code**

Ce fichier nous permet de configurer différentes notions de sécurité.

Dans un premier temps il indique quelle algorithme nous utilisons pour crypter les mots de passe puis nous définissons quelle entité et quelle champ de cette entité sera utilisé comme identifiant.

Puis nous pouvons gérer l’accès des url grâce à l’access control où l’on indique des patterns d’url en utilisant les *expressions régulières¹⁷*, suivi du rôle nécessaire.

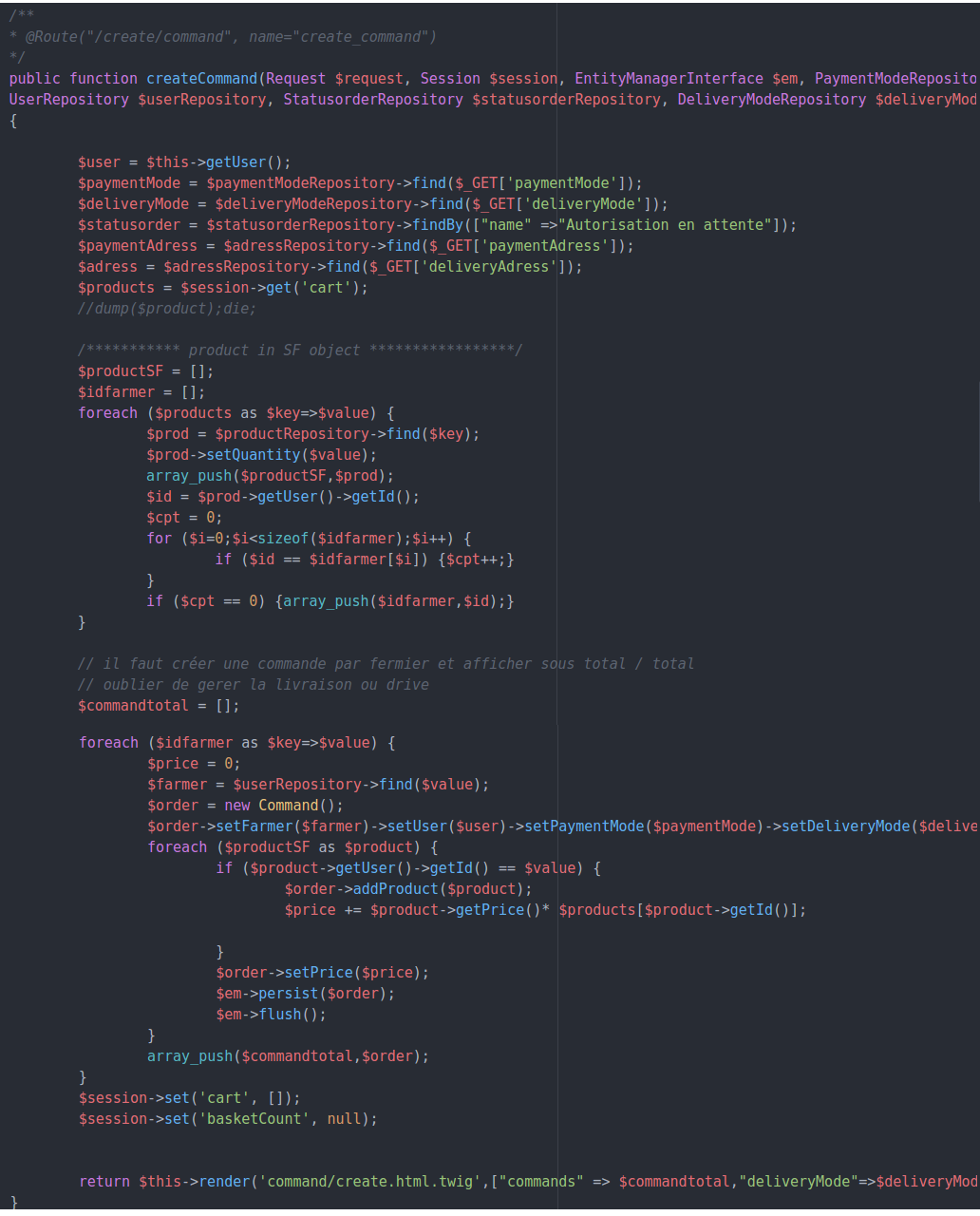
A la fin Symfony nous permet de pouvoir établir une hiérarchie, ici cela indique que l’administrateur et le producteur ont également accès aux espaces réservés aux utilisateurs.

Durant cette étape j’ai également mis en place la gestion du panier. Lors de l’ajout d’un produit au panier on le stocke dans la session du navigateur.

Lors de la validation du panier une problématique de logique métier nous demandait de créer autant de commande que de producteurs.

J’ai réussi à répondre à cette problématique grâce à une algorithme situé dans le contrôleur suivant.

**Contrôleur responsable de la création de commande**



**Explication du code**

Ce contrôleur effectue le traitement des données présentes dans la session du client. Il les trie et creé les commandes correspondantes.

Puis je continue par le squelette graphique du site.

4e étape - Création de l’interface graphique

Cette étape m’a permis de faire de l’intégration en utilisant au maximum Bootstrap.

**Exemple d’un template**



**Explication du code**

La première ligne correspond au layout de base que nous utilisons. Dans celui-ci on retrouve le menu de navigation et le footer.

Le code que nous voyons s’intègre dans un block body et title du layout.

L’utilisation d’un moteur de template nous permet de dynamiser notre HTML comme nous le voyons avec le lien qui est généré suivant les données en session de l’utilisateur.

La méthodologie que j’ai employé dans cette partie est de produire une interface simple pour chacune des actions que peut faire l’utilisateur en m’aidant des Users Stories et des routes.

Dans un premier temps je crée le gabarit HTML pour concevoir la partie fonctionnel pure et quand le test fonctionnel je passe au suivant.

Par exemple voici à la page suivante le code de la barre de navigation ou l’on peut voir également la gestion des rôles qui se fait en partie grâce à Twig car il génère des liens suivant le rôle de l’utilisateur.

**Partie incomplète de la barre de navigation**



**Explication du code**

Sur ce code le point pertinent est la condition qui gère les liens suivant le rôle de l’utilisateur. Nous pouvons donc facilement permettre une gestion des rôles de cette façon car nous générons les liens à la volée.

J’ai également beaucoup utiliser les formulaires Symfony qui permettent de mettre en place assez rapidement des champs suivant l’entité voulu et des contraintes standards.

**Exemple de formulaire Symfony**



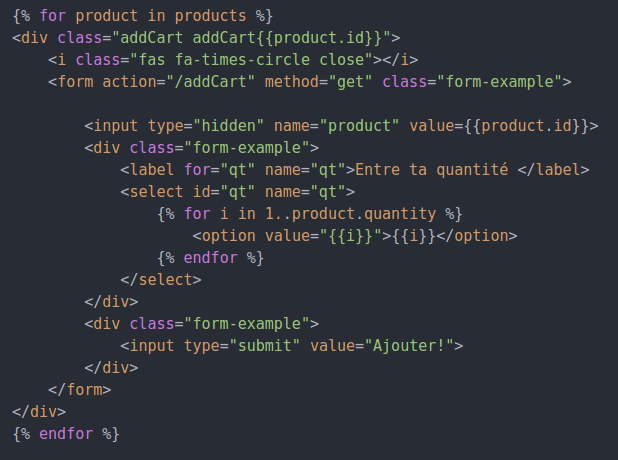
**Explication du code**

Ce code illustre la structuration d’un formulaire par Symfony. Cela permet une séparation des concepts car la gestion de l’affichage des champs du formulaire n’est pas faite par Twig et nous facilite dans nos contraintes.

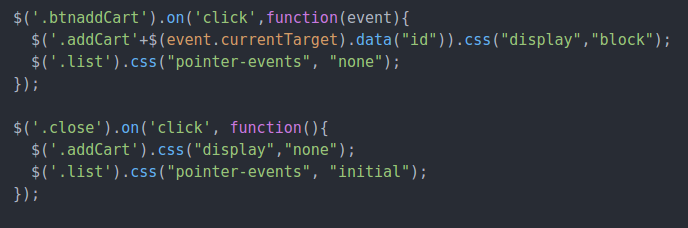
Comme pour les entité Symfony envisage les formulaires comme des objets auquel on ajoute des propriétés et des méthodes.

Afin de dynamiser l’ajout au panier et créer une pop-up permettant de rajouter un produit au panier j’ai utiliser un fichier JavaScript séparé qui manipule des div cachées qui s’affichent suivant l’action de l’utilisateur.

**Elément servant de pop-up généré par le template, chaque produit génère un pop-up caché**



**Code javascript qui gère les pop-up d’ajout de produit en panier**



**Explication du code**

La première portion de code montre comment est généré via Twig dans une boucle des éléments HTML. Chaque produits est généré avec en data-id son id de base de donnée.

La génération des listes de produits et d’agriculteurs sont générées de la même façon.

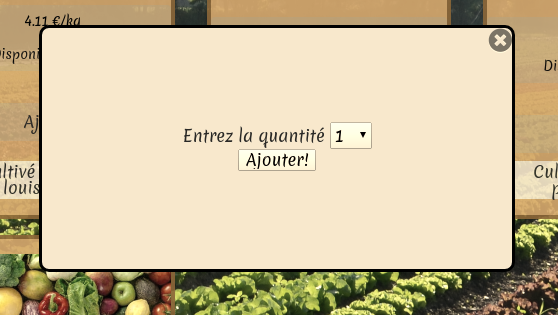
On peut voir que pour chaque produit nous générons une div qui a un nom de classe dynamique suivant l’id du produit. Cette élément est invisible.

Grâce à cela nous pouvons cibler avec javascript l'élément HTML qui correspond au produit et l'afficher.

Dans la deuxième portion de code nous voyons des listeners qui permettent lorsque nous cliquons sur un produit de récupérer son id puis de faire apparaître la div dite ‘pop-up’ correspondante et d'empêcher toute interaction sur le reste de la page.

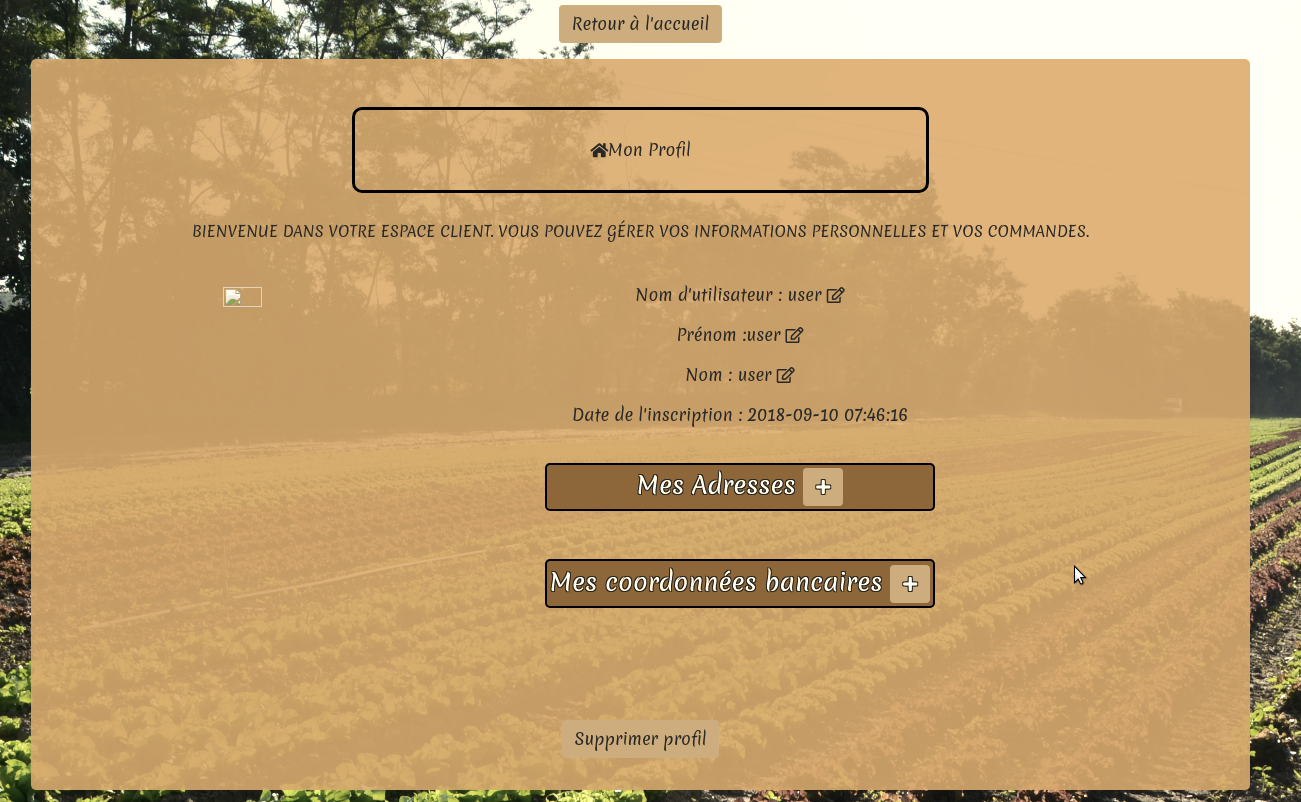
A cela s’ajoute la croix en haut à droite qui permet de faire disparaître cette div.

**Pop-up apparaissant lors de l’ajout du panier**



D’une manière identique j’ai incorporé les formulaires de modification de profil de façon ergonomique.

**Exemple de modification de profil**



**Code javascript qui gère les modification de profil**



**Méthode du contrôleur**



**Explication du code**

Ayant des soucis d’ordres technique pour incorporer des formulaires Symfony dans ce template, j’ai du utiliser des formulaires simples.

Pour cela on voit dans le code source du template que pour chaque champ de la table User un formulaire est créé avec tous les champs possibles cachés et avec une valeur nulle hormis le champ que l’on souhaite modifier. Ceci afin de pouvoir le traiter simplement côté serveur par le contrôleur ci-dessus.

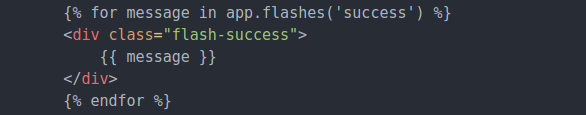
Par injection de dépendance la méthode instancie l’objet correspondant à l’utilisateur puis grâce aux setters modifie les champs si les données entrantes ne sont pas nuls.

Ensuite grâce à l’objet Doctrine EntityManagerInterface les données sont mis à jour dans la base de donnée.

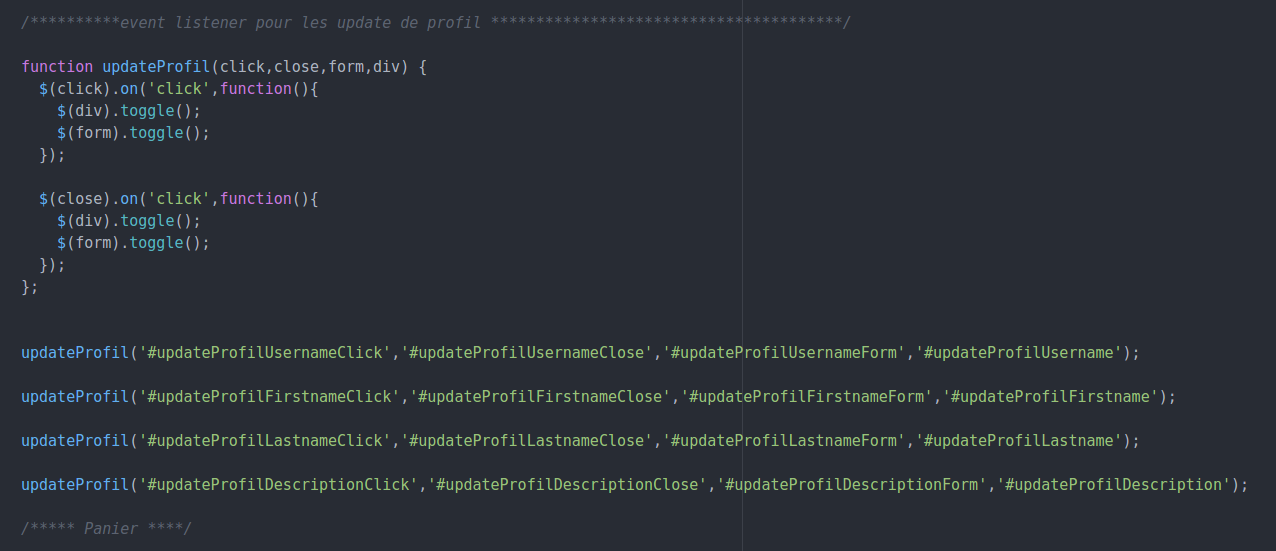
On remarquera ce qu’on appelle un message flash. C’est une fonctionnalité de Symfony.

On ajoute dans la session des données qui ne pourront être utilisé qu’une fois. Ce qui est très pratique pour l’utiliser comme message de confirmation ou d’erreur améliorant l’expérience utilisateur. Le code ci-dessous illustre la façon d’utiliser le flash message coté vue. C’est un code proposé par Symfony dans sa documentation.

**Code se trouvant dans le fichier twig**



**Code javascript qui gère l’interaction de modification du profil**



L’interaction est gérée par le code javascript ci-dessus permettant une interaction intuitive.

***5e étape - Refonte graphique, Test, Correction***

Pour finir et essayer d’améliorer notre travail j’ai voulu faire des test fonctionnels sur l’ensemble de l’application en reprenant chaque comportement de l’utilisateur ce qui m’a permis de relever quelques incohérences.

Pour effectuer les tests fonctionnels j’ai était méthodique en utilisant chaque fonctionnalité listées dans le cahier des charges afin d’en relever des incohérences ou dysfonctionnement.

Par exemple lors de l’inscription de l’utilisateur on ne lui demandait pas d’indiquer une adresse or s’il commandait il était obligatoire d’en avoir une ce qui m’a contraint à rajouter une étape d’enregistrement d’adresse dans le processus d’inscription.

**Difficultées rencontrées**

Relationnelles

En tant que Product Owner je me devais d’arriver à transmettre la même image clair du projet tel que je l’ai imaginé au début. Cela implique de s’appliquer à être le plus clair possible ce qui n’est pas toujours évident. De cette expérience j’ai compris qu cela pouvait être contre-productif de ne pas échanger ou de le faire tardivement avec son équipe lorsque l’on rencontre un problème technique. Chaque personne a une façon différente d’aborder les choses.

Techniques

Grâce au framework Symfony de nombreuses problématiques chronophage comme le routage, sauvegarde de la BDD ect…

Notre difficulté première a été de mettre en communication Symfony et React.

Pour ma part j’ai perdu beaucoup de temps sur le ciblage d’un tableau en session car la mise en session des données était faite par Symfony.

Et j’ai appris à réfléchir aux programmes dans sa globalité, non uniquement sur quelque fonctionnalités.

J’ai également appris à ne pas trop anticiper car travailler trop en amont sur une solution qui ne sera finalement pas mis en place fait perdre beaucoup de temps inutilement.

Vous trouverez en Annexe n°6 mon carnet de bord où j’ai noter succinctement chaque jour les avancements du projet.

Conclusion

Ce projet constitue le premier projet complet pour moi que j’ai pu prendre en charge de la conception à la livraison. J’ai beaucoup appris sur la gestion personnel du temps qui donne l’impression de devoir être étirable, sur la ténacité à surmonter une difficulté ou simplement à la laisser de côté pour y revenir et sur la gestion en équipe.

J’ai pu confirmer ma certitude sur le bien fondé de bien prendre son temps à produire des documents de conception les plus complets possible.

Egalement sur le fait que l’on veut toujours faire plus que ce que l’on a le temps de faire.

En terme d’expérience technique j’ai dû faire des recherches sur React et sur sa communication avec les frameworks PHP ce qui a enrichi mes compétences mais également sur des bundles Symfony tel que SwiftMailer. J’ai conforter mon apprentissage du framework Symfony et de l’architecture MVC.

La formation d’O’clock ne peut pas tout nous apprendre car on se rend bien compte que le domaine du développement web est très vaste, les technologies nombreuses et personnes ne peut tout maîtriser. Mais elle nous a donnée les bases nécessaires pour apprendre, savoir où chercher et comprendre les notions élémentaires. Les outils évoluent extrêmement rapidement, nous avons besoin d’être constamment en veille technologique, c’est bien le point principal que je retiendrais de ma formation.

C’est avec plaisir que j’ai entrepris ce projet, plaisir qui n’a pas disparu durant ce mois de pratique et il me tarde de pouvoir confronter et améliorer mes compétences dans le milieu professionnel.

Merci d’avoir lu ce dossier de projet!

Glossaire

***¹ Wireframe*** est une maquette fonctionnelle utilisé afin de définir les zones d’interaction avec les utilisateurs.

***² ORM*** (Objet Relational Mapping) est un type de programme informatique qui se place en interface entre un programme applicatif et unebase de données relationnelles pour simuler une [base de données orientée objet](https://fr.wikipedia.org/wiki/Base_de_données_orientée_objet).

**³ Faker** est une librairie PHP utilisé pour générer des données de test.

***⁴ AppFixturesBundle*** est une librairie Symfony permettant de générer de fausses donnée et les intégrer dans la base de données.

***⁵ Model*** est un élément de l’architecture de fichier back-end Model-View-Controller.

Cette architecture est mis en place automatiquement par le framework Symfony. Il permet une séparation des concepts, l’url rewriting, une barre de debug, ect…

Soin principal intérêt est de ne pas réinventer la roue et disposer d’un code éprouvé aux problématiques les plus communes.

***⁶ Architecture Model-View-Controller*** est un modèle d’architecture de fichier composé de 3 modules ayant chacun des responsabilités différents. Le modèle pour la gestion de la donnée, le contrôleur pour la logique concernant les actions effectuées par les utilisateurs et la vue pour l’interface graphique.

***⁷ Méthodes Agile*** sont des méthodes de pilotages et de réalisation de projet basées sur un manifeste Agile rédigé en 2001 qui énumère 12 principes à suivre pour améliorer son développement en équipe.

Le SCRUM inspiré des principes du manifeste Agile est un schéma d’organisation de développement de produits complexes. Il repose sur 3 caractéristiques , **la transparence** qui doit permettre à tous les membres de l’équipe de parler le même langage afin de tous avoir une vue d’ensemble du projet, **l’inspection** par les différents événements qui produisent des retours d’expérience afin de permettre le dernier pilier qui est **l’adaptation** c’est à dire que si l’on voit une déviation à un moment donnée nous adaptons le flux de travail afin de reprendre une bonne direction. Cette méthode est une démarche empirique qui prône l’adaptation continue.

***⁸ Module Bundler -*** Service qui permet de compiler les fichiers nécessaires d’un site web en versions compréhensible pour la plus grande partie des sites web.

***⁹ Push -*** commande Git permettant de déplacer les modifications de code du dépôt local au dépôt distants.

***¹⁰ Merge*** - commande Git permettant de rassembler les modifications de code d’une branche à l’autre.

***¹¹ Kanban*** - est une méthode de gestion de projet basé sur l’utilisation de liste et de tâches auxquelles on applique des statuts (à faire, en cours, terminé).

***¹² mutateurs et accesseurs -*** ce sont des méthodes qui permettent d'accéder ou modifier les propriétés privées ou protégées de la classe.

***¹³ Url rewriting*** - Symfony utilise l’URL rewriting ou réécriture d’URL qui est nécessaire pour l'expérience utilisateur et le référencement. Grâce à un fichier .htaccess Apache redirige toutes les requêtes vers le fichier index.php de l’application. Celui-ci nommé Front Controller dans le MVC traite la requête et redirige vers le contrôleur adéquat. Cela a l’avantage d’avoir des URL plus propre, plus facile à indexer et renforce la sécurité.

***¹⁴ API*** (application programming interface) est une interface de programmation qui est utilisé pour délivrer des fonctionnalités. Il est nécessaire d’avoir le manuel d’utilisation afin de savoir comment utiliser les services de l’API.

***¹⁵ Ajax*** (Asynchronous Javascript et XML) technologie combinant Javascript et le XMLHttpRequest afin de créer des interfaces riches sans nécessité de rafraîchir la page a chaque fois. Cela entraîne de meilleurs performance coté serveur et client.

***¹⁶ Data-id*** est un attribut que l’on peut donner à un élément HTML. Cette attribut est souvent utilisé car facilement manipulable par le javascript. Il permet l’échange de donnée facilement.

¹⁷ Les **expressions régulières** issu du monde mathématique permettent de décrire avec concision un ensemble d’url possible.

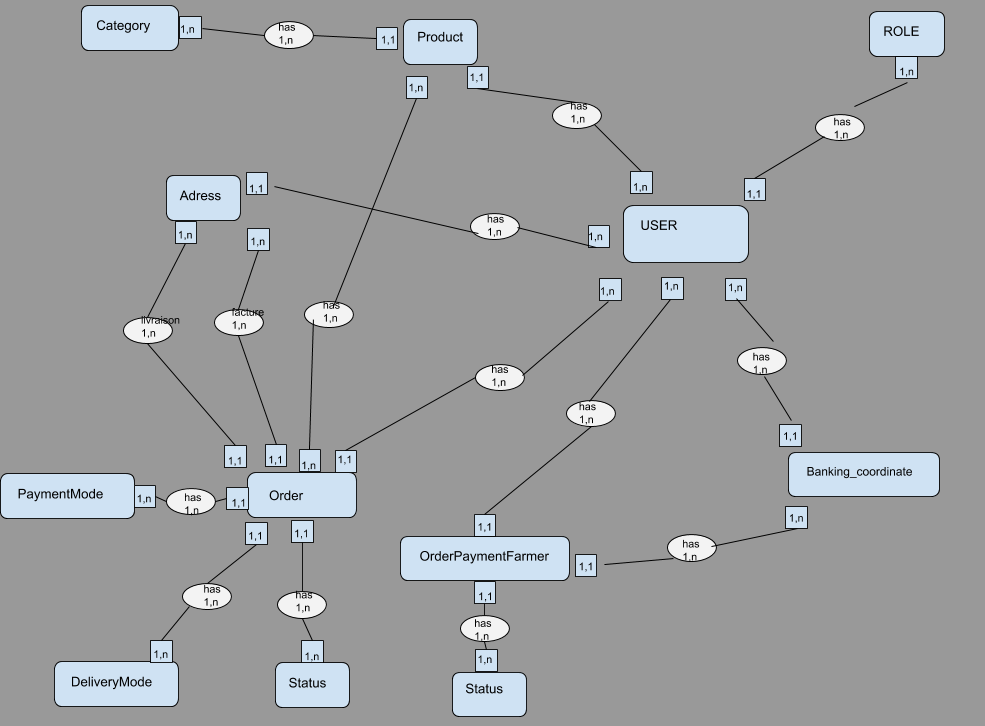
Annexe 1

Routes Version finale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nom | Url | Description | Controller | Security |
| Acceuil | / |  | MainController | / |
| Login | /login |  | MainController | / |
| Contact | /contact |  | MainController | / |
| Logout | /logout |  | MainController | / |
|  | | | | |
| Liste Farmer | /list/farmer |  | UserController | / |
| Liste Farmer by CP | /list/farmer/{code postal} |  | UserController | / |
| List User | /admin/user |  | UserController | ROLE\_ADMIN |
| Create User | /inscription |  | UserController | / |
| Show User | /profil/{id} |  | UserController | ROLE\_USER |
| Update User | /update/user/{id} |  | UserController | :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::!; |
| Delete User | /delete/user/{id} |  | UserController | ROLE\_USER |
|  | | | | |
| Admin | /admin | board | AdminController | ROLE\_ADMIN |
| Stat | /admin/stat |  | AdminController | ROLE\_ADMIN |
| Mail | /admin/mail | Bundle? | AdminController | ROLE\_ADMIN |
| Newsletter | /admin/newsletter | ?? | AdminController | ROLE\_ADMIN |
| Creer Newsletter | /admin/newsletter/create | ?? | AdminController | ROLE\_ADMIN |
| ::::::::::::: | | | | |
| Create Message | /create/message |  | MessageController | ROLE\_USER |
| Show Message | /message/{id} |  | MessageController | ROLE\_USER |
| Update message | /update/message/ {id} |  | MessageController | ROLE\_USER |
| Delete Message | /delete/message/{id} |  | MessageController | ROLE\_ADMIN |
| Read Message | /message/{id}/read |  | MessageController | ROLE\_USER |
| Unread | /message/{id}/unread |  | MessageController | ROLE\_USER |
|  | | | | |
| List Service | /list/service |  | ServiceController | / |
| List Service by Farmer | /list/service/farmer/{id} |  | ServiceController | / |
| List Service by Type | /list/service/category/{id} |  | ServiceController | / |
| Liste Service Proche | /list/service/{code postale} |  | ServiceController | / |
| Create service | /create/service |  | ServiceController | ROLE\_FARMER |
| Show service | /service/{id} |  | ServiceController | / |
| Update service | /update/service/ {id} |  | ServiceController | ROLE\_FARMER |
| Delete service | /delete/service/{id} |  | ServiceController | ROLE\_FARMER |
|  | | | | |
| List Product | /list/product |  | ProductController | / |
| List Product by Farmer | /list/product/farmer/{id} |  | ProductController | / |
| List Product by Category | /list/product/category/{id} |  | ProductController | / |
| Liste Produit Proche | list/products/{code postale} |  | ProductController | / |
| Create product | /create/product |  | ProductController | ROLE\_FARMER |
| Show Produit | /product/{id} |  | ProductController | / |
| Update Produit | /update/product/ {id} |  | ProductController | ROLE\_FARMER |
| Delete Produit | /delete/product/{id} |  | ProductController | ROLE\_FARMER |
|  | | | | |
| Liste OrderPaymentFarmer | admin/list/orderpaymentfarmer |  | OrderPaymentFarmerController | ROLE\_ADMIN |
| List OrderPaymentFarmer by Farmer | admin/list/orderpaymentfarmer/farmer/{id} |  | OrderPaymentFarmerController | ROLE\_ADMIN |
| Create OrderPaymentFarmer | /create/orderpaymentfarmer |  | OrderPaymentFarmerController | ROLE\_FARMER |
| Show OrderPaymentFarmer | /orderpaymentfarmer/{id} |  | OrderPaymentFarmerController | ROLE\_FARMER |
| Delete OrderPaymentFarmer | /delete/orderpaymentfarmer/{id} |  | OrderPaymentFarmerController | ROLE\_FARMER |
|  | | | | |
| List Command | /list/command |  | CommandController | ROLE\_ADMIN |
| List Command by Farmer | /list/command/farmer/{id} |  | CommandController | ROLE\_FARMER |
| List Command by User | /list/command/user/{id} |  | CommandController | ROLE\_USER |
| Create Command | /create/command |  | CommandController | ROLE\_USER |
| Show Command | /command/{id} |  | CommandController | ROLE\_FARMER |
| Update Command | /update/command/ {id} |  | CommandController | ROLE\_USER |
| Delete Command | /delete/command/{id} |  | CommandController | ROLE\_USER |
|  | | | | |
| List Reduction by User | /reduction/user/{id} |  | ReductionController | ROLE\_USER |
| Create Reduction | /create/reduction |  | ReductionController | ROLE\_ADMIN |
| Show Reduction | /reduction/{id} |  | ReductionController | ROLE\_USER |
| Update Reduction | /update/reduction/{id} |  | ReductionController | ROLE\_ADMIN |
| Delete Reduction | /delete/reduction/{id} |  | ReductionController | ROLE\_ADMIN |
| Reduction Available | /available/reduction/{id} |  | ReductionController | ROLE\_ADMIN |
| Reduction Disable | /disable/reduction/{id} |  | ReductionController | ROLE\_ADMIN |
|  | | | | |
| List Category | admin/list/category |  | CategoryController | ROLE\_ADMIN |
| Create Category | admin/create/category |  | CategoryController | ROLE\_ADMIN |
| Update Category | admin/update/category/{id} |  | CategoryController | ROLE\_ADMIN |
| Delete Category | admin/delete/category/{id} |  | CategoryController | ROLE\_ADMIN |
|  | | | | |
| List Role | admin/list/role |  | RoleController | ROLE\_ADMIN |
| Create Role | admin/create/role |  | RoleController | ROLE\_ADMIN |
| Update Role | admin/update/role/{id} |  | RoleController | ROLE\_ADMIN |
| Delete Role | admin/delete/role/{id] |  | RoleController | ROLE\_ADMIN |
|  | | | | |
| List DeliveryMode | admin/list/deliverymode |  | DeliveryModeController | ROLE\_ADMIN |
| Create DeliveryMode | admin/create/deliverymode |  | DeliveryModeController | ROLE\_ADMIN |
| Update DeliveryMode | admin/update/deliverymode/{id} |  | DeliveryModeController | ROLE\_ADMIN |
| Delete DeliveryMode | admin/delete/deliverymode/{id} |  | DeliveryModeController | ROLE\_ADMIN |
|  | | | | |
| List PaymentMode | admin/list/paymentmode |  | PaymentModeController | ROLE\_ADMIN |
| Create PaymentMode | admin/create/paymentmode |  | PaymentModeController | ROLE\_ADMIN |
| Update PaymentMode | admin/update/paymentmode/{id} |  | PaymentModeController | ROLE\_ADMIN |
| Delete PaymentMode | admin/delete/paymentmode/{id} |  | PaymentModeController | ROLE\_ADMIN |
|  | | | | |
| List TypeOfferService | /list/typeofferservice |  | TypeOfferServiceController | ROLE\_ADMIN |
| Create TypeOfferService | /create/typeofferservice |  | TypeOfferServiceController | ROLE\_ADMIN |
| Update TypeOfferService | /update/typeofferservice/{id} |  | TypeOfferServiceController | ROLE\_ADMIN |
| Delete TypeOfferService | /delete/typeofferservice/{id} |  | TypeOfferServiceController | ROLE\_ADMIN |

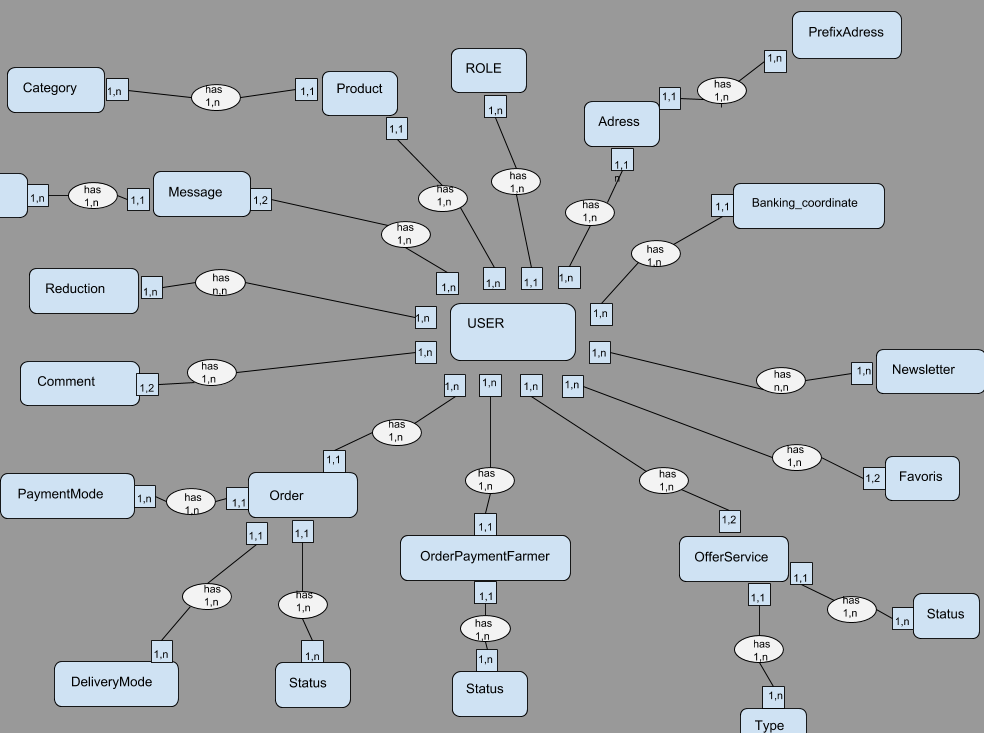
Annexe 2

MCD Version 1



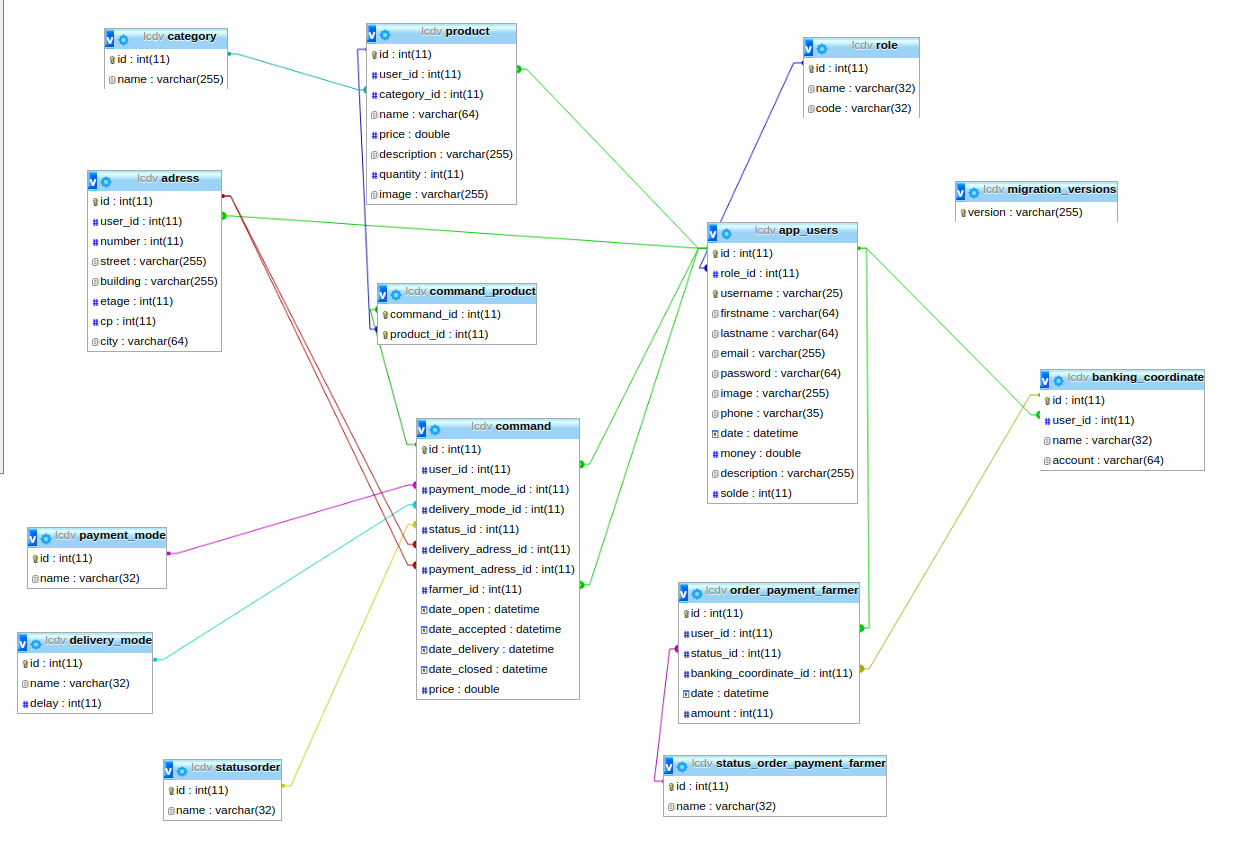
Annexe 3

MCD Version Final



Annexe 4

Modèle Logique de Données Final



Annexe 5

Dictionnaire de donnée

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Entity** |
| firstname |  |  | User |
| lastname |  |  | User |
| email |  |  | User |
| password |  |  | User |
| description |  |  | User |
| phone |  |  | User |
| registration\_date |  |  | User |
| name |  | USER/FARMER/ADMIN | Role |
| name |  |  | BankingCoordinate |
| account |  |  | BankingCoordinate |
| number |  |  | Adress |
| street |  |  | Adress |
| building |  |  | Adress |
| etage |  |  | Adress |
| CP |  |  | Adress |
| City |  |  | Adress |
| name |  |  | Product |
| price |  |  | Product |
| description |  |  | Product |
| flashSale |  |  | Product |
| harvestDate |  |  | Product |
| quantity |  |  | Product |
| name |  |  | Category |
| content |  |  | Commentary |
| date |  |  | Commentary |
| stars |  |  | Commentary |
| title |  |  | Message |
| body |  |  | Message |
| date |  |  | Message |
| name |  | read/unread/trash | StatusMessage |
| dateOpen |  |  | Command |
| dateAccepted |  |  | Command |
| dateDelivery |  |  | Command |
| dateClosed |  |  | Command |
| price |  |  | Command |
| name |  | waiting/accepted/delivery/closed | StatusCommand |
| name |  | emporter/aDomicile | DeliveryMode |
| delay |  |  | DeliveryMode |
| name |  | CB/Paypal | PaymentMode |
| date |  |  | OrderPaymentFarmer |
| amount |  |  | OrderPaymentFarmer |
| name |  | Waiting/Done | StatusOrderPaymentFarmer |
|  | | | |
| title |  |  | OfferService |
| body |  |  | OfferService |
| date |  |  | OfferService |
| price |  |  | OfferService |
| name |  | recolte/terassement/planter | TypeOfferService |
| description |  |  | TypeOfferService |
| name |  | waiting/done/closed | StatusOfferService |
| title |  |  | Newsletter |
| body |  |  | Newsletter |
| date |  |  | Newsletter |
| Quantity |  |  | Reduction |
| Status |  |  | Reduction |

Annexe 6

Carnet de Bord

2 Août

Le matin nous avons le lancement des projets avec les détails de mise en place et l’après-midi

lancement du projet nous créons les premiers documents délimitant le projet.

* Création du cahier des charges en commun.
* Création des user stories en commun.
* Création du dictionnaire de données seul étant le lead back.
* Création du fichier de routes seul étant le lead back.
* Création d’un tableau sur le site Trello ou nous mettons toute nos idées.

Nous nous concentrons exclusivement sur le cahier des charges afin d’avoir la vision la plus précise possible de l’application

3 Août

Nous continuons à rédiger les documents tout en commençant à réfléchir, lire de la documentation sur l’utilisation de React et Symfony dans le même projet.

Une fois le cahier des charges fait nous pouvons commencer à le traduire en document plus technique tel que le Modèle Logique de donnée, le dictionnaire de donnée ou encore faire le listing de toute les routes de l’application.

Cela demande un long travail méthodique si on veut éviter d’oublier quelque chose.

4 Août

Les documents sont terminés. Nous assistons le matin à une présentation du titre professionnel expliquant les attentes et les documents nécessaire.

L’après-midi nous réussissons à installer webpack dans Symfony nécessitant toute l’équipe.

Le problème qu’il nous reste à régler est la compilation du fichier .css qui ne s’effectue pas.

De mon côté j'ai effectué des tests réussis afin de renvoyer nos données en format JSON. Mais je n’arrive pas à les afficher à travers un composant React.

J’effectue également un test avec un bundle symfony d’envoi de mail et lis de la documentation sur les moyens de paiements.

Références

Recherche sur bundle d’envoi de mail.

<https://symfony.com/doc/current/email.html> pour SwiftMailerBundle

Recherche sur les moyen de paiement

<https://github.com/schmittjoh/JMSPaymentPaypalBundle>

<https://github.com/Payum/PayumBundle>

<https://github.com/alcalyn/payplug-bundle>

<https://afsy.fr/avent/2017/17-bien-demarrer-avec-symfony-et-react>, lecture

7 Août

Suite à réunion avec nos référents , le projet est trop compliqué pour y arriver dans le temps imparti.

Nous simplifions donc au maximum toute les fonctionnalités et les documents back-end.

Le MCD est remanié. La base du projet avec la base de données est créé et mis en commun sur GitHub.

Ensuite je code 2 controllers et un en cours.

Ce jour-là nous avons perdu un peu de temps avec git car nous n'arrivons pas à récupérer le dépôt.

Ce fut réglée avec cette commande git commit -m -a ‘’.

8 Août

Pour anticiper la communication entre React et Symfony je décide de rajouter 2 colonnes dans les Routes qui indiquent les datas sortantes et entrantes.

Je démarre un petit entrainement sur ce site afin de m'améliorer sur l’utilisation de git.

<https://learngitbranching.js.org/>

Démarrage des fonctionnalités des routes

- pensez comment tester si la bdd ne répond pas pour pas avoir une grosse erreur moche

- affichage des producteurs réfléchir à la gestion de leur adresse

9 Août

Aucune difficulté particulière. Route après route on récupère les bonnes datas et on les renvoie en json.

10 Août

Même travail que la veille.

13 Août

Afin d’avoir un code propre et factorisé et après quelque tests il m’a semblé plus propre de créer un CoreCustomController qui va permettre aux controllers d'hériter des méthodes liées au response json et au trycatch de base de donnée.

14 Août

Ce jour correspond au rdv n°2 avec l'équipe référente. Coté back les fixtures et les controllers sont faits on n’attend plus que le front pour se connecter mais le front n'avance pas du coup il est nécessaire de changer notre gestion des ressources. Soit en aidant le front en creusant React soit en passant Twig/React.

15 Août

Petit jour de flottement notre dev front nous donnent des indications sur React on effectue des recherches de notre coté notamment sur le fonctionnement d’ajax en react.

<https://www.javascriptstuff.com/react-ajax-best-practices/>

<https://reactjs.org/tutorial/tutorial.html>

En essayant de donner un coup de main j’ai proposé une arborescence pour le front.

Puis pour ma part j’ai continué mes lectures sur React.

on a parler de react native et electron pour l’avantage d’un projet react/symfo

// S’exercer sur JS

<https://exercism.io/tracks/javascript>

<https://www.pluralsight.com/paths/javascript>

//

Event listener react

<https://hackernoon.com/basic-concepts-and-applications-for-event-listeners-in-react-js-e616fbd22827>

16 Août

Ce matin nous avons présenté l’avancement du projet. Nous n’avions rien a présenter hormis du code Symfony.

N’avançant pas du tout coté front il a été décidé de passer en mode Twig/React car notre dev front ne se sent pas de faire tout le front seul.

Test pour effectuer la communication via Twig est un succès en nommant les variables pour React dans un boc twig javascript.

17 Août

Mise en place d’un design sobre et rapide, une barre de navigation et une liste de produit.

Souci rencontré pour la création du panier. Il est enregistrer en Session avec Symfony et j’ai perdu beaucoup de temps pour arriver à cibler le tableau avec Twig.

18 Août

Le panier est en phase final au niveau fonctionnel.

On peut afficher le profil , la liste des commandes et des produits, gestion des produits.

20 Août

Mise en place de la gestion des commandes pour le producteur.

Avec SwifMailer je suis arriver à implanter la fonctionnalité Contact.

Avec PA on s’occupe de récupérer du code que l’on a égaré (ne sachant pas ou)

Problème au niveau de la gestion du stock, lorsque qu’un panier est lancé même si le cookie expire la quantité n’est pas remis disponible.

21 Août

Refonte totale du design.

Rajout du favicon ;p

Aucune difficulté particulière à part tatonner en css.

22 Août

Ce jour consiste en petit peaufinage et ajout visuel, notamment avec le CRUD sur le profil.

23 Août

Le matin il y a la présentation des projets.