

LE PRIMEUR DE LAIGNEVILLE

RAPPORT DE STAGE

CONCEPTEUR DEVELOPPEUR
D'APPLICATIONS

Du 31/10/2023 au 05/01/2024



Association pour la formation
professionnelle des adultes.

30 Rue de Poulainville,

LOUNIS BENYAHIA

SOMMAIRE

1.INTRODUCTION	2
1.1 REMERCIEMENTS	2
1.2 PRESENTATION DE L'ENTREPRISE	3
1.3 OBJECTIF DU STAGE	3
2. OUTILS ET TECHNOLOGIES	4
3. CONCEPTION	9
3.1 LE DICTIONNAIRE DE DONNEES.....	9
3.2 LE MODELE CONCEPTUEL DE DONNEES (MCD)	9
3.3 DIAGRAMMES (UML).....	11
a. Diagramme des cas d'utilisation.....	11
b. Diagramme d'activité	12
c. Diagramme de séquences.....	13
d. Scénarios pour l'application	13
e. Graph de navigation : illustration des chemins de navigation.....	14
3.4 MAQUETTAGE.....	15
a. Zoning:.....	15
b. Wireframe:.....	16
c. Mockup.....	17
4. DEVELOPPEMENT DU PROJET	17
4.1 CREATION DU PROJET POUR LE PRIMEUR DE LAIGNEVILLE.	17
4.2 CREATION DE LA BASE DONNEES	18
4.3 CREATION DES PREMIERES ENTITES	18
4.4 INSTALLATION DE VICHUPLOADER	19
4.5 MISE EN PLACE DU FRONT POUR MON VISUEL.....	21
4.6 GESTION DES UTILISATEURS.....	27
4.7 GESTION DES CONTACTS	34
4.8 MISE EN PLACE D'UN SYSTEME D'ADMINISTRATION AVEC EASYADMIN	36
4.9 CREATION D'UNE LISTE COURSE.....	40
4.10 CREATION DES PROMOS.	42
4.11 CREATION DE SERVICES	44
4.12 BARRE DE RECHERCHE	44
4.13 MOT DE PASSE OUBLIE.....	45
5. TESTS.....	46
5.1 PRESENTATION	46
5.3 L'ENVIRONNEMENT	47
5.4 CRÉATION DE TESTS.	47
a. Test unitaire	48
b. Test fonctionnel.....	48
6. CONCLUSION	49

1.INTRODUCTION

1.1 Remerciements

En préambule, j'aimerais adresser mes remerciements à plusieurs personnes.

Tout d'abord, un grand merci à Medhi Benchou pour m'avoir offert cette opportunité de stage et pour sa confiance tout au long de cette expérience.

De plus, je tiens à remercier François-Régis Caumartin pour son accompagnement durant cette période de formation.

Je souhaite également exprimer ma reconnaissance envers mes camarades de promotion Pierre, Lucas, Mokhtar, avec qui nous avons partagé ce parcours, pas toujours facile, et qui a donné lieu à des moments de partage, d'entraide, et de travail collaboratif.

Et pour terminer, merci à vous, chers lecteurs et membres du jury, pour votre attention et votre curiosité envers le travail réalisé au cours de ce stage, et que cette nouvelle année soit bonne pour nous tous.



1.2 Présentation de l'entreprise

Le Primeur de Laigneville est un commerce de détail. Il vend des fruits et des légumes, des épices, et des produits de crèmerie.

Maraîcher de père en fils, ce commerce est né dans le contexte du Covid. Les marchés, étant des lieux de rencontre, n'avaient plus lieu. Et à part les grandes enseignes, il était très compliqué de trouver des fruits et légumes frais chez le détaillant.

L'entrepôt, basé sur Laigneville, est devenu, avec l'accord de la mairie, un lieu de vente pour les fruits et légumes, donnant la possibilité au gens de pouvoir continuer à s'approvisionner. Un fois le contexte passé, l'endroit a eu droit à un relooking, afin d'être mis en conformité, et de pouvoir recevoir les clients dans le nouveau magasin Primeur de Laigneville.

1.3 Objectif du stage

Ayant le projet de me mettre à mon compte, j'ai voulu aborder ce stage comme une prospection de potentiels clients, et commencer à développer de l'expérience.

Le commerce Le Primeur, avait un besoin de développer sa visibilité, et sa relation avec sa clientèle, et je devais donc réaliser un site Le Primeur de Laigneville pour répondre à ce besoin.

La tâche était complète, et j'ai pu voir toutes les étapes de la réalisation :

- Analyse des besoins, étude de la concurrence, et définition du cahier des charges,
- Conception : du dictionnaire de données, du MCD, les UML au maquettage, cette partie a été fondamentale, et a représenté environ les 2/3 du projet,
- Développement du site : le choix des outils, l'écriture du code pour le back-end et le front et la gestion de la base de données.



2. OUTILS ET TECHNOLOGIES



Visual Studio Code (VS code) est l'outil que j'ai utilisé pour mon projet de création du site. Développé par Microsoft, Vs code est un éditeur de code source, qui allie simplicité d'utilisation, performance et personnalisation. Il est largement reconnu dans le secteur du développement web, pour ses fonctionnalités et ses fonctionnalités robustes.

Présentation détaillée de Visual Studio Code :

VS Code est un éditeur de code léger mais puissant qui fonctionne sur votre ordinateur de bureau et est disponible pour Windows, macOS et Linux. Il est équipé d'un support intégré pour JavaScript, TypeScript et Node.js, et dispose d'un riche écosystème d'extensions pour d'autres langages tels que C++, C#, Java, Python, PHP, Go, etc. Cela en fait un outil particulièrement polyvalent pour les projets de développement logiciel.

L'un des points forts de VS Code est sa fonctionnalité d'IntelliSense, qui offre des complétions de code intelligentes basées sur des types de variables, des définitions de fonctions et des modules importés. Cela aide à écrire du code plus rapidement et avec moins d'erreurs.

Enfin, l'aspect personnalisable de VS Code m'a permis de configurer l'éditeur en fonction de mes besoins spécifiques, en ajoutant des extensions pour améliorer la productivité, comme PHP Intelephense pour une meilleure autocomplétion du code PHP, ou Symfony for VSCode pour des snippets de code Symfony prédéfinis.

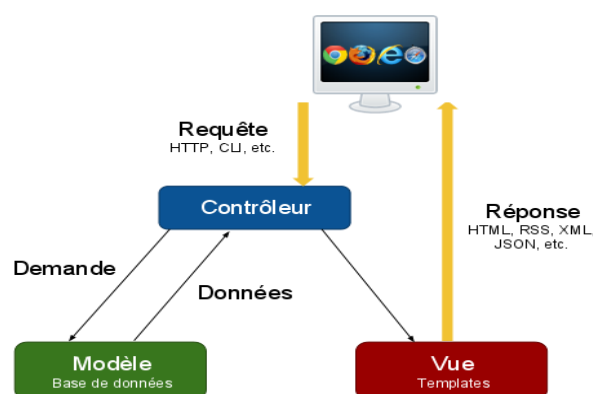
En somme, l'utilisation de VS Code dans le cadre du projet de refonte du LMS a non seulement facilité mon travail en tant que développeur, mais a également permis une progression plus efficace et organisée du projet.



Présentation détaillée de Symfony :

Symfony est un ensemble de composants PHP ainsi qu'un framework web à part entière, construit pour être rapide, flexible et plus stable. Il offre une architecture de projet solide et des composants réutilisables qui permettent de créer des applications web robustes et maintenables.

L'architecture de Symfony repose sur le modèle MVC (Modèle-Vue-Contrôleur), qui sépare les préoccupations de l'application en trois composants distincts. Le Modèle gère les données et la logique métier, la Vue gère l'affichage de l'interface utilisateur, et le Contrôleur gère l'interaction entre le Modèle et la Vue. Cette séparation des préoccupations favorise une meilleure organisation du code, une plus grande modularité et une maintenance plus aisée.



Doctrine est un ORM (Object-Relational Mapping) qui facilite l'interaction avec la base de données, Twig est un moteur de template pour la création de vues dynamiques, et les formulaires Symfony permettent de créer facilement des formulaires sécurisés et personnalisables.



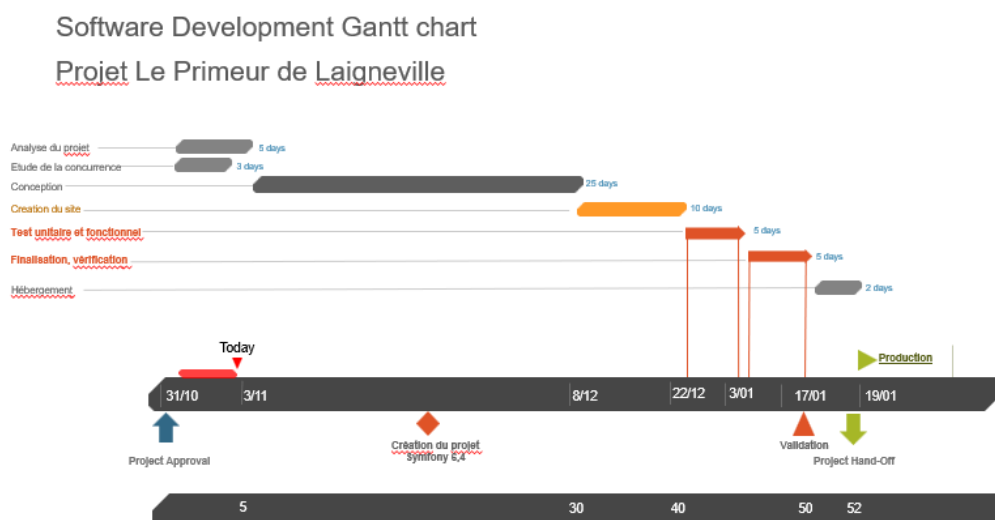
Git est une plateforme de travail collaboratif en ligne basée sur un système de gestion de versions. Elle permet aux développeurs de travailler ensemble, en mutualisant les efforts et gérer les contributions, elle permet aussi de sauvegarder son travail et de le partager.



Utilisation du diagramme de Gantt dans le projet :

Un diagramme de Gantt est un outil visuel précieux pour la gestion de projet, qui permet de planifier les tâches sur une ligne du temps et de visualiser leur déroulement. Les avantages de l'utilisation d'un diagramme de Gantt incluent la possibilité de voir clairement les dépendances entre les tâches, la durée de chaque tâche, ainsi que la progression globale du projet. C'est pourquoi j'ai choisi de l'utiliser.

Dans le diagramme de Gantt de mon projet, j'ai alloué du temps à différentes phases :



En somme, le diagramme de Gantt a été un outil précieux pour planifier et suivre mon projet, permettant de gérer mon temps de manière efficace et de garder une vue d'ensemble sur la progression du projet.



Draw.io est une application qui permet de faire des schémas et du dessin vectoriel. Elle a la particularité d'être une application web (qui tourne entièrement dans le navigateur). C'est donc un SaaS (Software as a Service).

J'ai réalisé mes diagrammes la plupart de mes schémas et diagrammes sur cet outil.



Figma est un outil collaboratif de création graphique en ligne qui permet de réaliser des maquettes de type UX/UI. Elle permet de faciliter la création de sites web et d'applications mobiles dans la réalisation de projets communs.

Cet outil permet ainsi de travailler à plusieurs sur un même projet en effectuant des modifications en direct sur différents "workspace" avec un système d'autorisations qui permet aux membres d'observer le travail réalisé (en tant que guest) ou alors de réaliser des modifications sur les "frames" pour faire avancer le projet.



Bootstrap est une librairie de templates disponible en ligne permettant aux utilisateurs de prélever directement des composants d'interface en ligne afin de réaliser leurs projets front-end.

Il permet la conception réactive d'une page ou d'une application Web tout en respectant les exigences du responsive.



Comme interface avec la base de données, je me suis servi de phpMyAdmin, qui est une application Web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB, réalisée principalement en PHP.



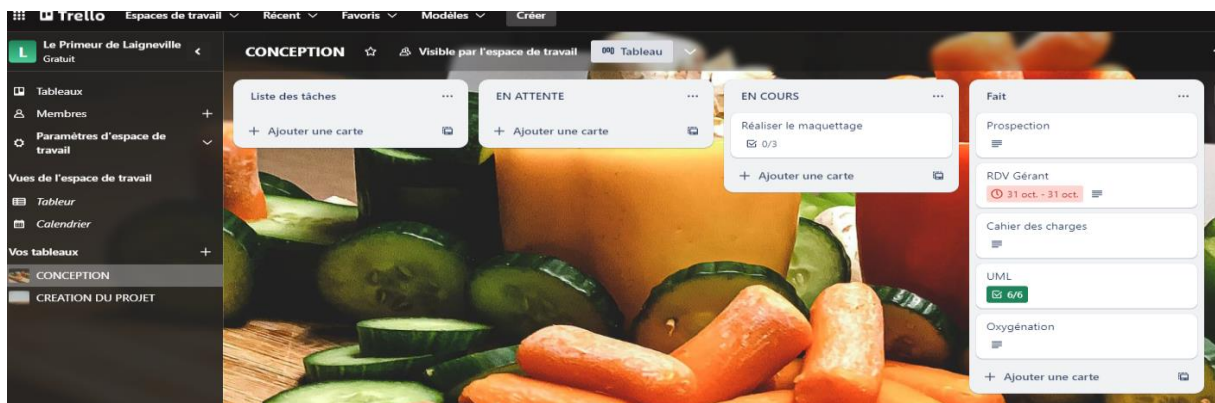
Afin de suivre l'évolution du projet, j'ai utilisé l'outil de gestion de projet Trello et découpé le projet en plusieurs étapes.

Trello est une plateforme de gestion de projets en ligne qui permet une organisation claire et efficace des tâches. Inspirée par la méthode Kanban de Toyota, Trello propose une approche visuelle et flexible de la planification des projets. Les utilisateurs peuvent créer des planches de projet, qui sont ensuite divisées en listes représentant les différentes étapes d'un projet. Chaque liste contient des cartes qui représentent les tâches individuelles.

Les principales étapes du projet ont été divisées en différentes colonnes :

- "Analyse", "Conception", "Création du site". Cette structure m'a permis de visualiser facilement l'ensemble du projet et de suivre son avancement.

En somme, l'outil Trello, grâce à sa flexibilité et à sa facilité d'utilisation, a joué un rôle clé dans mon organisation.



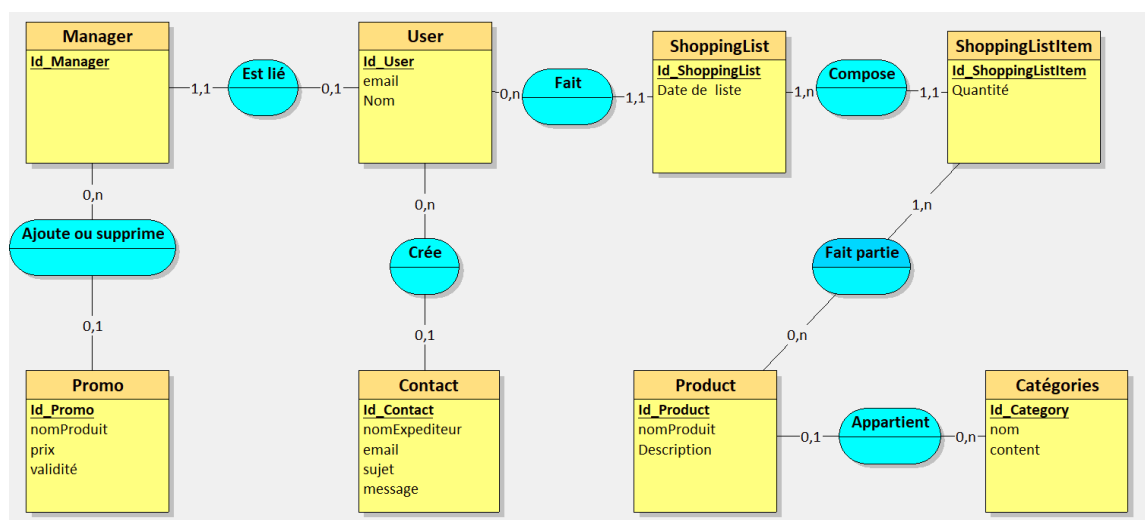
3. CONCEPTION

3.1 LE DICTIONNAIRE DE DONNEES

Description des données utilisées, leur type et leur rôle dans l'application.

Table	Code	Libelle	Type	Contraintes
USER	<u>Id Utilisateur</u>		INT AUTO INCREMENT	
	Firstname	Prénom de utilisateur	VARCHAR (50)	entre 2 et 50
	Mail	Adresse mail de utilisateur	VARCHAR (50)	entre 2 et 180
	Role		VARCHAR (50)	USER, ADMIN
SOPPINGLISTE	<u>Id ShoppingList</u>		INT AUTO INCREMENT	
	CreatedAt		DATETIME	
SHOPPING LISTE DETAIL	<u>Id ShoppingListItem</u>		INT AUTO INCREMENT	
	Quantite	Quantite	VARCHAR (50)	
PRODUCT	<u>Id Produit</u>		INT AUTO INCREMENT	
	Name	Nom du produit	VARCHAR (50)	entre 2 et 50
	Description	Description du produit	VARCHAR (50)	entre 2 et 50
	Image	Image du produit	VARCHAR (255)	entre 2 et 50
	CreatedAt	Date de création	DATETIME	
CONTACT	<u>Id Sous categorie</u>		INT AUTO INCREMENT	
	Fisrtname	Prénom du contact	VARCHAR (50)	entre 2 et 50
	Mail	Adresse mail de utilisateur	VARCHAR (180)	entre 2 et 180
	Subject	Sujet	VARCHAR (100)	entre 2 et 100
	Message	Texte du message	TEXT	
CATEGORY	<u>Id Categorie</u>		INT AUTO INCREMENT	
	Name	Nom de categorie	VARCHAR (50)	
	Content	Type	VARCHAR (50)	
	Imagesrc	Image	VARCHAR (50)	
PROMO	<u>Id Promo</u>		INT AUTO INCREMENT	
	Title	Nom de la promo		
	Description	Descriptif du produit		
	Prix	Prix		
	Image			

3.2 LE MODELE CONCEPTUEL DE DONNEES (MCD)



Le modèle conceptuel de données sert à conceptualiser l'application ! Il met en évidence deux éléments : les entités et les associations.

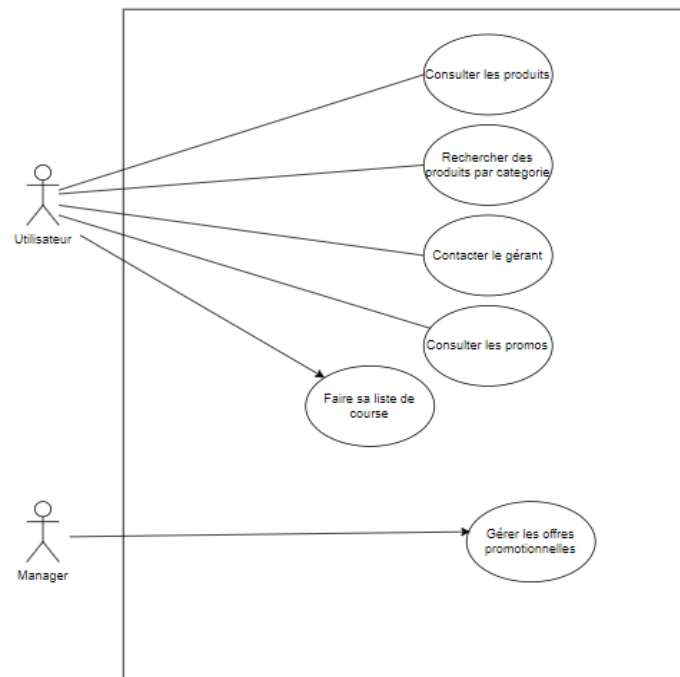
- L'entité est un objet que l'on souhaite modéliser par exemple : un produit, une commande, une catégorie. De plus, chaque entité possède des attributs, par exemple : un nom, une couleur, une longueur...
- Les associations illustrent le lien entre les entités : les relations sémantiques :
 1. **User** et **Manager** : Un **Manager** est associé à un seul **User**. Un **User** peut être lié à zéro ou un **Manager**.
 2. **User** et **Contact** : Un **Contact** est associé à un seul **User**. Un **User** peut avoir plusieurs **Contacts**.
 3. **Manager** et **Promotion** : Un **Manager** gère plusieurs **Promotions**. Une **Promotion** est gérée par un seul **Manager**.
 4. **Category** et **Product** : Une **Catégorie** peut avoir plusieurs **Produits**, mais chaque **Produit** appartient à une seule **Catégorie**.
 5. **_User** et **ShoppingList** : Un **_User** peut créer plusieurs **Listes de Courses**, mais chaque **Liste de Courses** est associée à un seul **_User**.
 6. **ShoppingList** et **ShoppingListItem** : Une **Liste de Courses** peut contenir plusieurs **Éléments de Liste de Courses**, mais chaque **Élément de Liste de Courses** appartient à une seule **Liste de Courses**.
 7. **Product** et **ShoppingListItem** : Un **Élément de Liste de Courses** est un **Produit**. Chaque **Produit** peut être associé à plusieurs **Éléments de Liste de Courses**.
 8. **Product** et **Fait_partie** : Un **Produit** peut être inclus dans plusieurs **Éléments de Liste de Courses**, et chaque **Élément de Liste de Courses** contient un ou plusieurs **Produits**.



logiciel de modélisation de données utilisé pour créer des Modèles Conceptuels de Données (MCD) et des Modèles Logiques de Données (MLD) conçu pour aider à concevoir et visualiser la structure des bases de données. Looping est entièrement gratuit et libre d'utilisation, et est développé à l'université Toulouse III.

3.3 Diagrammes (UML)

a. Diagramme des cas d'utilisation



Pour l'utilisateur, j'ai défini les parcours suivants :

- Consulter les produits** : permettant une vue d'ensemble de l'inventaire disponible.
- Rechercher des produits par catégorie** : offrant une méthode ciblée pour parcourir l'inventaire.
- Contacter le gérant** : un canal direct pour les demandes de service ou d'informations.
- Consulter les promos** : où les utilisateurs peuvent trouver les meilleures offres.
- Faire sa liste de course** : une fonctionnalité pour planifier et organiser les achats.

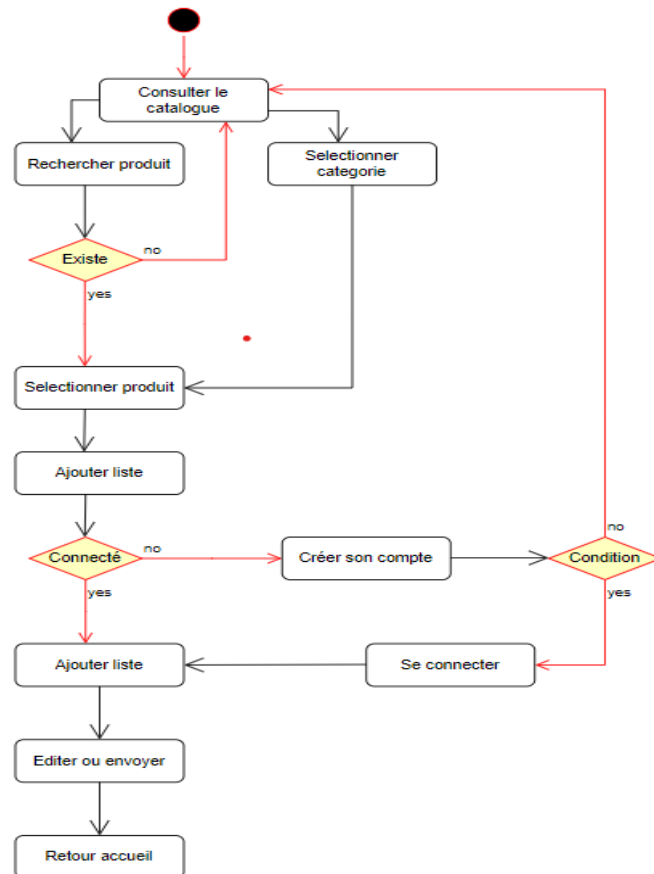
Pour le manager, j'ai conçu une fonctionnalité distincte :

- Gérer les offres promotionnelles** : ce qui permet au manager de créer et de modifier des promotions pour les utilisateurs.

Ce diagramme a non seulement facilité la compréhension des différents flux d'interaction.

L'élaboration de ce diagramme de flux a été un exercice méthodique qui m'a permis de consolider mes compétences en conception de système et en réflexion analytique. J'ai appris l'importance de visualiser les interactions des utilisateurs de manière à ce que les fonctionnalités soient à la fois complètes et efficaces, tout en étant simples à utiliser.

b. Diagramme d'activité



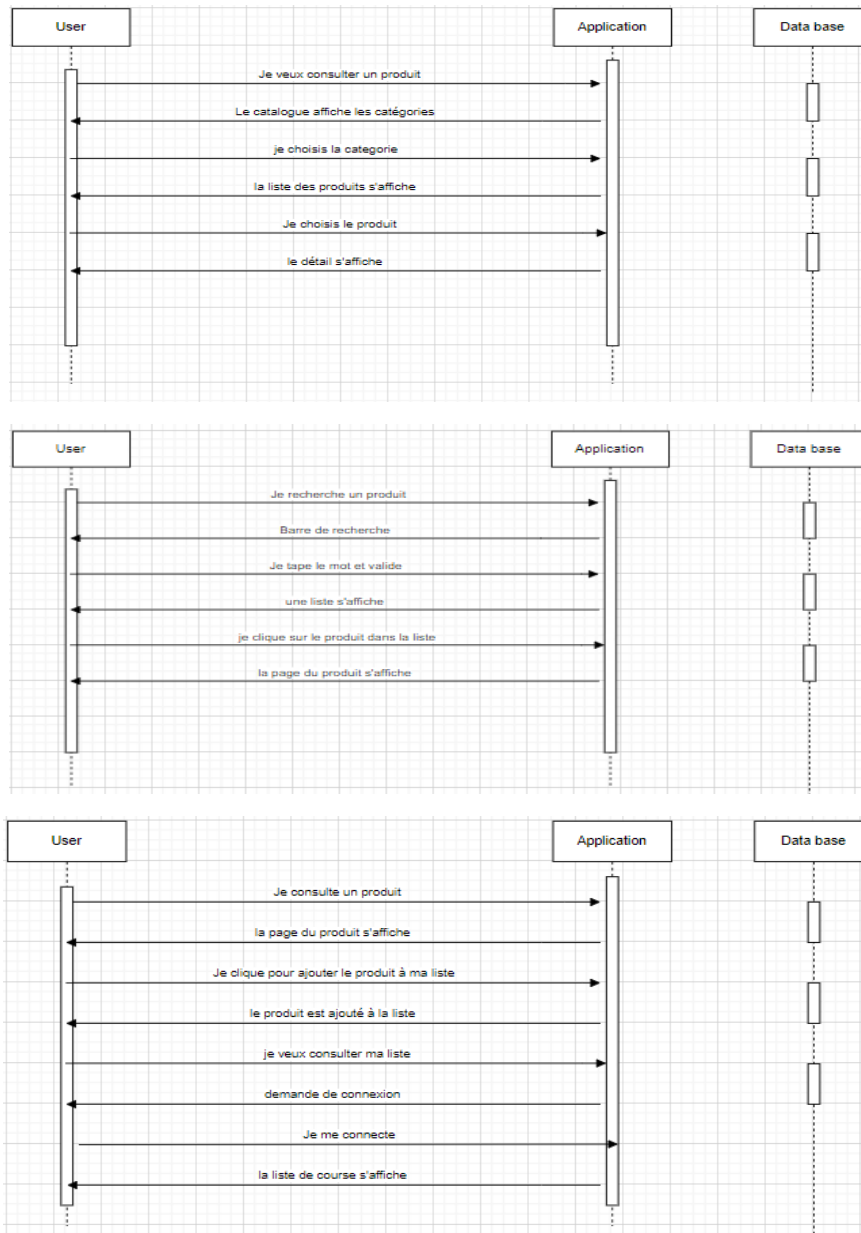
En langage de modélisation unifié (UML), un diagramme de cas d'utilisation peut servir à résumer les informations des utilisateurs de votre système (également appelés acteurs) et leurs interactions avec ce dernier. Il décrit le flux des processus dans l'application.

Le processus comprenait les étapes :

- de recherche de produit,
- de sélection de produit/catégorie,
- d'ajout à la liste ,
- de connexion/création de compte,
- et enfin, d'édition et d'envoi de la commande.

J'ai également intégré des conditions logiques pour gérer le flux en fonction de l'état de connexion de l'utilisateur et de la disponibilité du produit.

c. Diagramme de séquences



d. Scénarios pour l'application

Scénario 1 : Recherche de Produits par un Utilisateur

- Acteur : Utilisateur
- Déclenchement : L'utilisateur souhaite rechercher des produits.
- Actions :
 - L'utilisateur entre des termes de recherche.
 - Le système traite la requête de recherche.
 - Le système renvoie les résultats de la recherche à l'utilisateur.
- Fin : L'utilisateur visualise les résultats de la recherche.

Scénario 2 : Création d'un Compte Utilisateur

- Acteur : Utilisateur
- Déclenchement : L'utilisateur souhaite créer un compte utilisateur.
- Actions :
 - L'utilisateur sélectionne l'option "Créer un compte".
 - L'utilisateur entre ses informations (nom, adresse e-mail, mot de passe, etc.).
 - Le système vérifie la validité des informations.
 - Le système crée un compte utilisateur dans la base de données.
- Fin : Le compte utilisateur est créé, et l'utilisateur peut maintenant se connecter.

Scénario 3 : Consultation des Offres Promotionnelles

- Acteur : Utilisateur
- Déclenchement : L'utilisateur souhaite consulter les offres promotionnelles disponibles.
- Actions :
 - L'utilisateur accède à la section "Promotions" de l'application.
 - Le système récupère les offres promotionnelles en cours.
 - Le système affiche les offres promotionnelles à l'utilisateur.
- Fin : L'utilisateur peut consulter les offres promotionnelles actuelles.

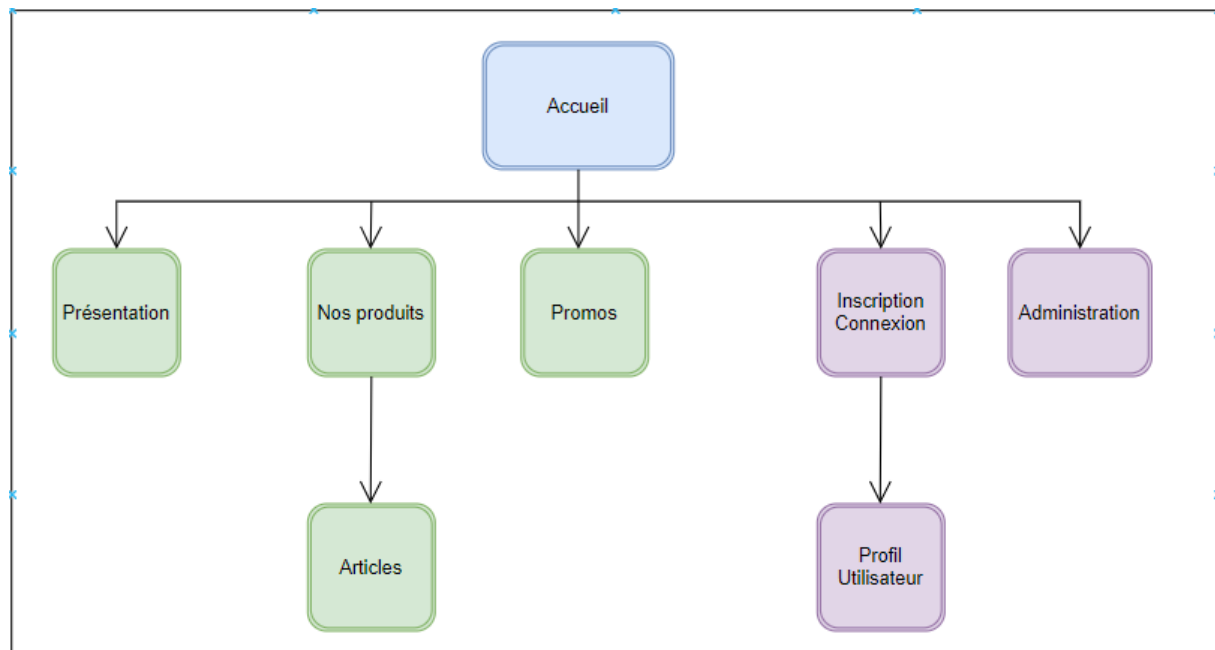
Scénario 4 : Création d'un contact avec le manager

- Acteur : Utilisateur
- Déclenchement : L'utilisateur souhaite créer une demande contact.
- Actions :
 - L'utilisateur sélectionne l'option "Demande de contact".
 - L'utilisateur clique sur le lien en pied de page.
 - L'utilisateur remplit le formulaire et valide l'envoi.
- La demande est envoyée et reçue par l'administrateur.

Scénario 5 : Gestion des Offres Promotionnelles par le Gérant

- Acteur : Gérant
- Déclenchement : Le gérant souhaite gérer les offres promotionnelles de produits.
- Actions :
 - Le gérant se connecte à l'interface d'administration.
 - Le gérant sélectionne l'option "Gérer les offres promotionnelles".
 - Le gérant ajoute, modifie ou supprime des offres promotionnelles.
 - Le système met à jour la base de données avec les modifications.
- Fin : Les offres promotionnelles sont gérées avec succès.

e. **Graph de navigation** : illustration des chemins de navigation



3.4 Maquettage

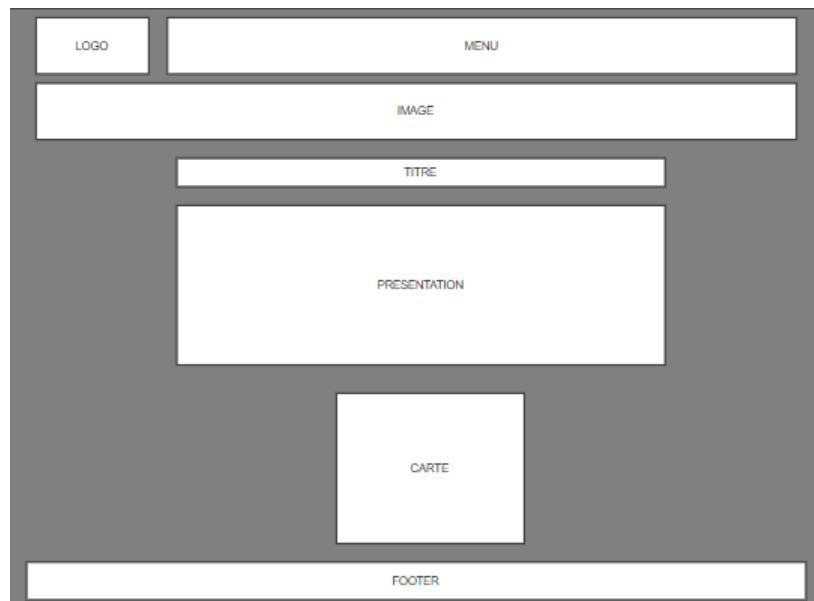
a. Zoning: Définition de l'organisation spatiale de la page.

Le zoning est une schématisation grossière de ce que sera la future page web. On utilise des blocs pour déterminer où se trouveront les contenus et fonctionnalités. Cette étape a généralement lieu après la création d'une arborescence, il arrive quelquefois qu'elle soit réalisée en parallèle.

Définir l'organisation générale des pages est l'occasion de présenter une première approche au client ou décideur. Celui-ci pourra alors valider ou réajuster les grands axes avant la réalisation des wireframes.

Les grandes zones de contenus et autres éléments doivent être cohérents sur la page. Il n'est pas rare que les souhaits initiaux soient inadaptés, par exemple une page d'accueil surchargée d'informations. C'est lors du zoning qu'est effectué ce premier débroussaillage.

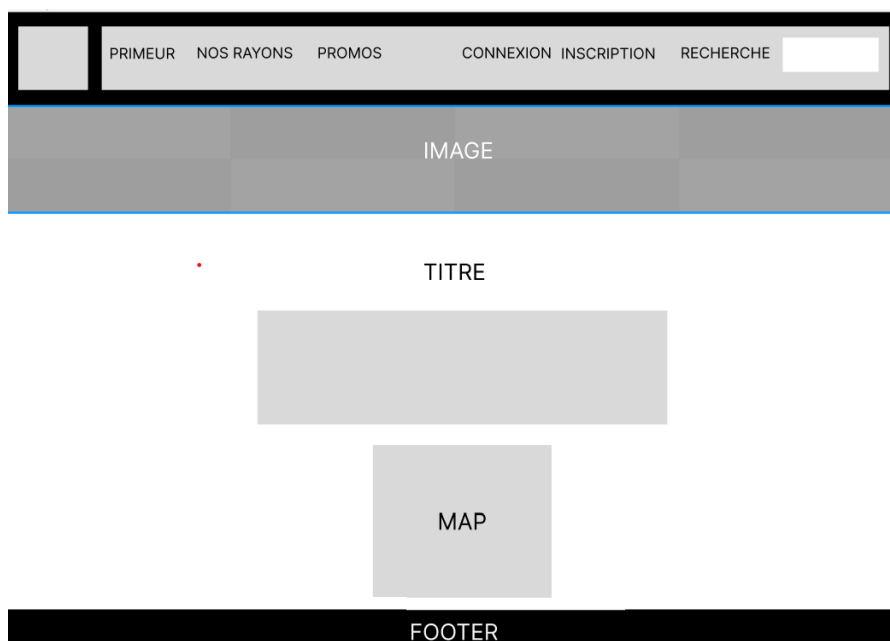




b. Wireframe: Schéma détaillé de l'interface utilisateur.

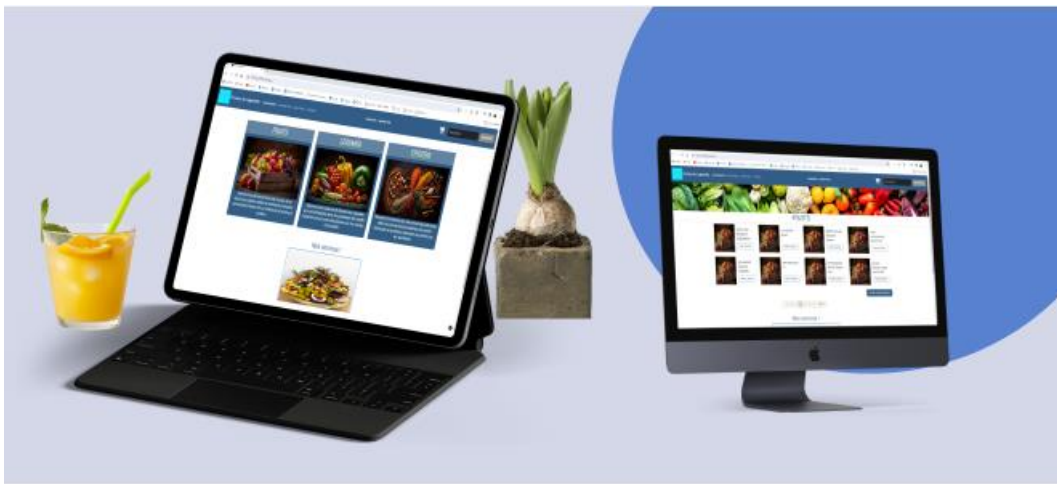
Ensuite le wireframe (on parle de « maquette fil de fer » en français) est la suite logique du zoning. L'objectif est de définir l'organisation des éléments et des formes sans travailler l'aspect visuel.

Là encore, un échange avec le client est nécessaire pour valider les avancées. Le wireframe, en bon outil de communication, l'aide à se projeter. Il évite surtout la rédaction d'un cahier des charges fonctionnel où les besoins peuvent être incomplets ou mal définis, ce qui entraînerait une refonte coûteuse de la plateforme finale. Les wireframes jouent le même rôle en présentant chaque fonctionnalité et spécification associée.



c. **Mockup**: Représentation visuelle détaillée du produit final.

Un Mockup est une maquette de la future interface web ou mobile. C'est un outil de conception graphique, d'évaluation et de communication . Il se fait déjà beaucoup plus esthétique que le wireframe. Bien sûr il reprend la structure esquissée dans le wireframe, mais il montre à quoi le produit final doit ressembler. Il n'est en revanche pas cliquable : c'est une simple image. Mais une image qui se veut le plus proche possible du but à atteindre. J'ai utilisé Artboard comme plugin sur Figma.



4. DEVELOPPEMENT DU PROJET

4.1 Création du projet pour Le Primeur de Laigneville.

J'ai créé un dossier PrimeurLaigneville, que j'ai ouvert dans VS code, et dans le terminal, avec la commande :

symfony new primeur --webapp

Le projet, étant créé, j'utilise la commande **cd primeur** pour travailler directement dans celui-ci, et je lance le serveur avec la commande **symfony server :start** pour confirmer que le projet est bien lancé, avec la page Symfony 6.4 qui s'affiche.

4.2 Création de la base données.

J'utilise Laragon pour la gestion de la base de données MySQL et configuration de l'environnement de développement.

Je crée un fichier **env.dev.local**

```
DATABASE_URL="mysql://root@127.0.0.1:3306/primeur?serverVersion=8.0.30&charset=utf8mb4"
```

Je peux créer ma base de données avec la commande

php bin/console doctrine:database:create

Je me connecte à PhpMyAdmin pour m'assurer que ma base de données a bien été créée.

4.3 Création des premières entités

J'utilise la commande **php bin/console make :entity** pour créer Category et Product, et pour User, j'utilise **php bin/console make :user**.

Pour chaque entité, deux fichiers sont créés :

- Entity,
- Repository.

Je complète mes entités avec

```
use Symfony\Component\Validator\Constraints as Assert;
```

des contraintes pour valider les données, pour une cohérence de la base données et pour les utilisateurs, comme **#[Assert\NotBlank()]**.

Ensuite je mets en place les relations entre les entités, tel qu'elles sont définies dans le MCD

Dans Category, je définis la relation OneToMany, une catégorie peut avoir plusieurs produits et un produit ne peut avoir qu'une seule catégorie.

Je lance la commande **php bin/console make :migration**, Symfony interprète l'objet et va générer le script SQL, et ensuite **php bin/console doctrine :migrations :migrate**, pour tout pousser sur la base de données. Mes entités, avec leurs attributs et les propriétés sont dans ma base de données.

4.4 Installation de VichUploader

J'ai choisi d'utiliser ce bundle pour importer les images de produits :

Composer req vich/uploader-bundle

Ajout dans bundles.php :

```
Vich\UploaderBundle\VichUploaderBundle::class => ['all' => true],
```

Dans Packages -> vich_uploader.yaml :

```
vich_uploader:
  db_driver: orm
  metadata:
    type: attribute

  mappings:
    products:
      uri_prefix: /images/produit
      upload_destination: "%kernel.project_dir%/public/images/produit"
      namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
```

Dans l'entité Product :

```
<?php

namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use App\Repository\ProductRepository;
use Symfony\Component\HttpFoundation\File\File;
use Vich\UploaderBundle\Mapping\Annotation as Vich;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

#[ORM\Entity(repositoryClass: ProductRepository::class)]
#[UniqueEntity('name')]
#[Vich\Uploadable]
class Product
{
    #[Vich\UploadableField(mapping: 'products', fileNameProperty: 'imageName')]
    private File $image;
```

```
private ?File $imageFile = null;

#[ORM\Column(type: 'string', nullable: true)]
private ?string $imageName = null;

#[ORM\Column(length: 255, nullable: true)]
private ?string $image = null;
```

Je relance la migration et le migrate, avant de créer mes fixtures. Pour cela, j'installe le bundle avec la commande :

composer req --dev orm-fixtures

et pour générer les fausses données :

composer req fakerphp/faker

Je crée un fichier ProductFixtures dans DataFixtures qui a été généré par mon composer :

```
$faker = Factory::create('fr_FR');
```

```
// PRODUCTS
for ($i = 1; $i <= 100; $i++) {
    $product = new Product();
    $product->setName($faker->unique()->words(4, true))
        ->setDescription($faker->realText(10));

    $randomCategory = $faker->randomElement([$Category1, $Category2,
$Category3]);
    $product->setCategory($randomCategory);

    $manager->persist($product);
}
```

Je fais de même avec Category, cette fois en inscrivant les données, pour 3 catégories.

A l'aide de manager

```
$manager->flush();
```

Pour envoyer dans la base de données les fixtures, j'utilise la commande

php bin/console doctrine:fixtures:load

Dans PhpMyAdmin :

SELECT * FROM `user`

☐ Profilage ☐ Éditer en ligne ☐ Éditer ☐ Expliquer SQL ☐ Créer le code source PHP ☐ Actualiser

☐ Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par clé : Aucun(e)

Options supplémentaires

		id	email	roles	password	firstname	created_at (OC2Type.datetime_immutable)
<input type="checkbox"/>	Éditer Copier Supprimer	1	admin@primeur.com	["ROLE_USER", "ROLE_ADMIN"]	\$2y\$13\$QX7PhKcLzE3VQKeSBj9qteG8gd7gVbtjSoQcRN.0J1...	admin	2023-12-16 10:28:01
<input type="checkbox"/>	Éditer Copier Supprimer	2	guillaume97@wanadoo.fr	["ROLE_USER"]	\$2y\$13\$5KqP0plx0FTSv6P86d6BweTsE9PJfgzXi7XPxvGQmCH...	Guy	2023-12-16 10:28:02
<input type="checkbox"/>	Éditer Copier Supprimer	3	wdesousa@gmail.com	["ROLE_USER"]	\$2y\$13\$D45L66wZuXnc6/Cq5rXReunV.EnZLdBN0luS6JwCkS6...	Emmanuel	2023-12-16 10:28:02
<input type="checkbox"/>	Éditer Copier Supprimer	4	zauger@maurice.fr	["ROLE_USER"]	\$2y\$13\$WgxhwE1KVdXZaDVkV9v.eTrib8hrWSMf62nXuhmf...	Mathilde	2023-12-16 10:28:03
<input type="checkbox"/>	Éditer Copier Supprimer	5	grenier.christiane@leconte.com	["ROLE_USER"]	\$2y\$13\$fxwBJ0xMF6p4uUkhETHB1OtuyGrDpWAQuT3E.jxeUGz...	Roland	2023-12-16 10:28:03
<input type="checkbox"/>	Éditer Copier Supprimer	6	margaux.albert@briand.fr	["ROLE_USER"]	\$2y\$13\$ziMdBnm8RS/xbrlb2v7ojus9GZpQ0M6hfm6JocDOaO...	Daniel	2023-12-16 10:28:03
<input type="checkbox"/>	Éditer Copier Supprimer	7	afpa@afpa.com	["ROLE_USER"]	\$2y\$13\$neCMxSfBGeceq7YTN06aeJl/A1Om8jGIFRL0W7IfAO...	Lou	2023-12-20 07:56:41

4.5 Mise en place du front pour mon visuel

Dans base.html.twig, j'importe les liens :

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">

{% block stylesheets %}
    <link rel="stylesheet" href="/bootstrap.min.css">
    <link rel="stylesheet" href="/_variables.scss">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.3/css/all.min.css">
    <link rel="stylesheet" href="/style.css">
{% endblock %}

{% block javascripts %}
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.min.js"
integrity="sha384-
Atwg2Pkwv9vp0ygtn1JAoJH0nYbwNjLPhwyoVbhoPwBhjQPR5VtM2+xf0Uwh9KtT"
crossorigin="anonymous"></script>
    <script src="https://kit.fontawesome.com/77b6e6294c.js"
crossorigin="anonymous"></script>
{% endblock %}
```

a) Création de ma page d'accueil

Je crée mon controller Home avec la commande :

Php bin/console make :controller

Qui va me générer 2 fichiers :

- Un controller HomeController.php
- Un template home.html.twig

Mon contrôleur Home :

```
class HomeController extends AbstractController
{
    #[Route('/', name: 'app_home')]
    public function index(): Response
    {

        return $this->render('home/index.html.twig', [

        ]);
    }
}
```

Mon template index :

```
{% extends 'base.html.twig' %}

{% block title %}Accueil{% endblock %}

{% block body %}
    <div class="content-container">

        <div class="container">

            <div class="row">
                <div class="col-12 text-center mt-4">
                    <h1 class="title-style">BIENVENUE CHEZ LE PRIMEUR DE
LAIGNEVILLE</h1>
                </div>
            </div>

            <div class="row">

                <div class="col-md-6">
```

```

        <p class="lead text-primary">
            <strong>Maraîcher, c'est une histoire de famille chez
nous. Nous vous proposons des fruits et légumes, de qualité à prix accessible
pour tous, crèmerie et épicerie.</strong>
            <br><br>
            <strong>Nous sommes un commerce de proximité, de
fruits et légumes, proposant également un rayon épicerie et des produits de
consommation courante pour vous faire gagner du temps.</strong>
        </p>
    </div>

```

b) Mise en place de la Navbar et du footer

Dans template, je crée un dossier partials, dans lequel je mets les fichiers _navbar.html.twig et _footer.html.twig, qui sont des composants et non des vues, que j'appelle dans mon fichier base.html.twig

Pour la navbar, je crée un controller

```

class NavController extends AbstractController
{
    public function index(): Response
    {
        return $this->render('partials/_navbar.html.twig', [

        ]) ;
    }
}

```

Et pour le footer, j'appelle le fichier avec la méthode include

```

{{ render(controller('App\\Controller\\NavController::index')) }}
    {% block body %}
    {% endblock %}

    {% block footer %}
        {% include "partials/_footer.html.twig" %}
    {% endblock %}

```


c) Création de ma page Catégorie

Dans mon controller Home, j'ajoute une route qui retournera ma vue catégorie :

```
#[Route('/categorie', name: 'app_category', methods: ['GET'])]
public function category(CategoryRepository $categoryRepository): Response
{
    $categories = $categoryRepository->findAll();

    return $this->render('home/category.html.twig', [
        'categories' => $categories,
    ]);
}
```

Et je crée mon template qui extends 'base.html', j'appelle les catégories dans une boucle **for**, et je mets en forme dans des cards de **Bootstrap avec le nom, le contenu et l'image**.

```
{% for cat in categories %}
    <div class="col-md-3 mt-4">
        <a href="{{ path('app_product', {id: cat.id}) }}">
            <div class="card text-white bg-primary mb-4 custom-card">
                <div class="card-header d-flex justify-content-center
h2">{{ cat.name }}</div>
                <div class="card-body d-flex flex-column align-items-
center justify-content-center">
                    
                    <p class="card-text text-center">{{ cat.content
}}</p>
                </div>
            </div>
        </a>
    </div>
{% endfor %}
```

Ce qui nous donne :



Dans mon composer.json, je crée un composer qui supprime, crée la BD, met à jour le schéma, et génère les fixtures afin de simplifier la commande :

```
"fixtures": [
    "php bin/console doctrine:database:drop --force",
    "php bin/console doctrine:database:create",
    "php bin/console doctrine:schema:update --force --complete",
    "php bin/console doctrine:fixtures:load --append"
]
```

d) Création de ma page produit

Dans mon controller Home, j'ajoute une route qui retournera ma vue produits. Ayant un grand nombre de produits, je mets en place une pagination en installant un bundle

composer req knplabs/knp-paginator-bundle

Je crée un fichier knp_paginator dans config/packages :

```
knp_paginator:
    page_range: 5                # number of links shown in the
                                # pagination menu (e.g: you have 10 pages, a page_range of 3, on the 5th page
                                # you'll see links to page 4, 5, 6)
    default_options:
        page_name: page         # page query parameter name
        sort_field_name: sort   # sort field query parameter name
        sort_direction_name: direction # sort direction query parameter name
        distinct: true          # ensure distinct results, useful when
                                # ORM queries are using GROUP BY statements
        filter_field_name: filterField # filter field query parameter name
        filter_value_name: filterValue # filter value query parameter name
    template:
        pagination:
            '@KnpPaginator/Pagination/bootstrap_v5_pagination.html.twig' # sliding
                                # pagination controls template
```

```

        sortable: '@KnpPaginator/Pagination/sortable_link.html.twig' # sort
link template
        filtration: '@KnpPaginator/Pagination/filtration.html.twig' # filters
template

```

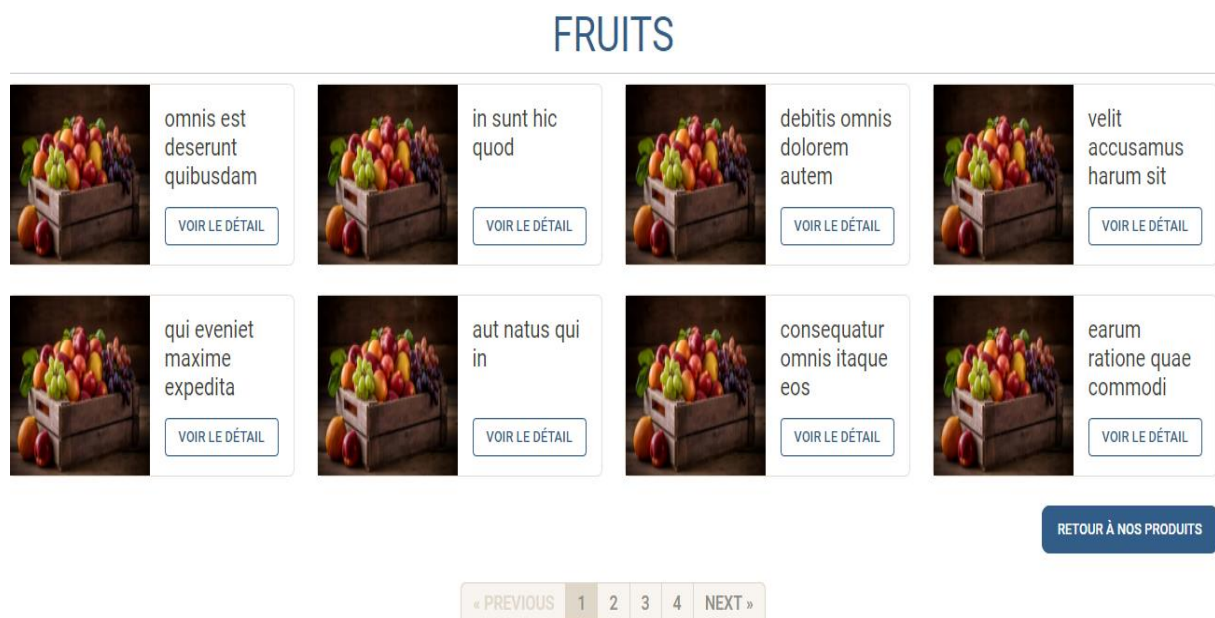
Je fais une injection de dépendances dans mon controller :

```

#[Route('/produit/{id}', name: 'app_product')]
public function product(Category $category, PaginatorInterface
$paginator, Request $request): Response
{
    $data = $paginator->paginate(
        $category->getProducts(),
        $request->query->getInt('page', 1),
        8
    );
    return $this->render('home/product.html.twig', [
        'category' => $category,
        'products' => $data,
    ]);
}

```

Ce qui nous donne :



4.6 Gestion des utilisateurs

Mon entité User, étant créée, je mets en place les contraintes avec **Assert** :

```
#[ORM\Column(length: 180, unique: true)]
#[Assert\Length(min: 2, max: 180)]
#[Assert\Email()]
private ?string $email = null;
```

Je rends le profil unique par son email

```
#[UniqueEntity('email', 'Cet email existe déjà en tant qu\'utilisateur')]
```

J'ajoute une propriété **plainpassword**, sans ORM, qui servira seulement à encoder le mot de passe :

```
private ?string $plainPassword = null;
```

Je crée une fonction pour construire la date de création et de mise à jour :

```
public function __construct()
{
    $this->createdAt = new \DateTimeImmutable();
    $this->updatedAt = new \DateTimeImmutable();
}
```

Et une fonction pour mettre à jour :

```
#[ORM\PreUpdate]
public function preUpdate()
{
    $this->updatedAt = new \DateTimeImmutable();
}
```

Je crée un fichier UserFixtures pour les données utilisateur, dans lequel on set le plainpassword qui va être encodé dans notre entityListener :

```
for ($j = 1; $j <= 5; $j++) {
    $user = new User();
    $user->setEmail($faker->email);
    $user->setRoles(["ROLE_USER"]);
    $user->setFirstname($faker->firstName);
    $user->setPlainPassword('password');

    $users[] = $user;
    $manager->persist($user);
}

$manager->flush();
```

Dans security.yaml, Symfony a modifié le provider, qui va permettre d'aller récupérer un utilisateur dans la base de données en fonction de la propriété **email** qui sera l'identifiant :

```
providers:
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

Dans services.yaml :

```
App\EntityListener\:
    resource: "../src/EntityListener/"
    tags: ["doctrine.orm.entity_listener"]
```

Je crée un dossier EntityListener, et un fichier UserListener qui va encoder le password :

```
<?php

namespace App\EntityListener;
use App\Entity\User;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;
class UserListener
{
    private UserPasswordHasherInterface $hasher;
    public function __construct(UserPasswordHasherInterface $hasher)
    {
        $this->hasher = $hasher;
    }
    public function prePersist(User $user)
    {
        $this->encodePassword($user);
    }
    public function preUpdate(User $user)
    {
        $this->encodePassword($user);
    }
    /**
     * Encode password based on plain password
     *
     * @param User $user
     * @return void
     */
    public function encodePassword(User $user)
```



```

{
    if($user->getPlainPassword() === null) {
        return;
    }
    $user->setPassword(
        $this->hasher->hashPassword($user, $user->getPlainPassword())
    );
    $user->setPlainPassword(null);
}

```

Un Entity Listener est une classe qui permet d'observer et de régir aux événements liés à une entité(classe) particulière dans Doctrine(ORM). Ces événements sont déclenchés lors de différentes phases du cycle d'une vie d'une entité, tels que la création, la mise à jour, la suppression...

Dans User.php :

```
#[ORM\EntityListeners(['App\EntityListener\UserListener'])]
```

Je vérifie dans ma base de données, mes utilisateurs sont bien créés, le mot de passe est encodé

Parcourir

Structure

SQL

Rechercher

Insérer

Exporter

Importer

Privileges

Opérations

Suivi

Déclencheurs

Affichage des lignes 0 - 6 (total de 7, traitement en 0.0004 seconde(s).)

SELECT * FROM `user`

Profilage

Éditer en ligne

Éditer

Expliquer SQL

Créer le code source PHP

Actualiser

Tout afficher

Nombre de lignes : 25

Filtrer les lignes: Chercher dans cette table

Trier par clé : Aucun(e)

Options supplémentaires

	id	email	roles	password	firstname
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	1	admin@primeur.com	["ROLE_USER", "ROLE_ADMIN"]	\$2y\$13\$QX7PhKcLzE3VQKeSBj9qteG8gd7gVbtrjSoQcRN.0J1...	admin
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	2	guillaume97@wanadoo.fr	["ROLE_USER"]	\$2y\$13\$5KqP0plx0FTSv6P86d6BweTsE9PJFgzXl7XPxGQmCH...	Guy
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	3	wdesousa@gmail.com	["ROLE_USER"]	\$2y\$13\$D45L66wZuXnc6/Cq5rXReunV.EnZLdBnOluS6JwCkS6...	Emmanuel
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	4	zauger@maurice.fr	["ROLE_USER"]	\$2y\$13\$WgxhwE1KVdXZaDVkTV9v.eTrib6hrWSMf62nXuhnfrl...	Mathilde
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	5	grenier.christiane@leconte.com	["ROLE_USER"]	\$2y\$13\$fxwBJ0xMF6p4uUkhETbB1OtuyGrDpWAQuT3E.jxeUGz...	Roland
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	6	margaux.albert@briand.fr	["ROLE_USER"]	\$2y\$13\$ziMdBnn8RS/xbrlb2v7ojus9GZpQ0M6hfm6JocDOOaO...	Daniel
<div><div></div><div>Éditer</div><div>Copier</div><div>Supprimer</div></div>	7	afpa@afpa.com	["ROLE_USER"]	\$2y\$13\$neCMxIsFbGeceq7YTN06aeJI/A1Om8jGIFRL0W7IfAO...	Lou

Je crée mon controller Security avec :

Php bin/console make:controller Security

Une route pour me connecter :

```
#[Route(path: '/connexion', name: 'app_login', methods: ['GET', 'POST'])]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    return $this->render('security/login.html.twig', [
        'last_username' => $authenticationUtils->getLastUsername(),
        'error' => $authenticationUtils->getLastAuthenticationError(),
    ]);
}
```

Et je crée mon formulaire de connexion dans login.html.twig, j'utilise un modèle de Bootswatch dans le thème choisi.

```
<form action="{{ path('app_login') }}" method="post" name="login">
    <div class="form-group">
        <label for="username" class="form-label mt-4">Adresse
email</label>
        <input type="email" class="form-control" id="username"
name="_username" placeholder="exemple@gmail.com" value="{{ last_username }}">
        <small id="emailHelp" class="form-text text-muted"></small>
    </div>
    <div class="form-group">
        <label for="password" class="form-label mt-4">Mot de
passe</label>
        <input type="password" class="form-control" id="password"
name="_password" placeholder="Entrez votre mot de passe">
    </div>
    <button type="submit" class="btn btn primary mt-4">Se
connecter</button>
</form>
```

Une route pour me déconnecter :

```
#[Route(path: '/deconnexion', name: 'app_logout')]
public function logout(): void
{
    //Nothing to do here..
}
```

Et je modifie mon fichier security.yaml, j'ajoute dans le firewalls les routes :

```
form_login:
    login_path: app_login
    check_path: app_login
logout:
    path: app_logout
```

Maintenant qu'on peut se connecter, je vais m'attaquer à l'inscription. Dans un premier temps, je vais créer un formulaire avec la commande

Php bin/console make : form Registration

Je complète les champs de fichier Registration.Type.php, en ajoutant les contraintes, les labels, pour les attributs Firstname, email, et deux champs pour le mot de passe pour confirmation, et un bouton pour valider.

```
'second_options' => [
    'attr' => [
        'class' => 'form-control',
        'placeholder' => 'Confirmer votre mot de passe',
    ],
    'label' => 'Confirmation du mot de passe',
    'label_attr' => [
        'class' => 'form-label mt-4'
    ]
],
'invalid_message' => 'Les mots de passe ne correspondent pas'
```

Dans le controller Security, je crée la route inscription, dans laquelle je crée un nouvel utilisateur en créant un formulaire, si celui-ci est soumis et valide, alors le **user** sera créée dans la base de données.

```
#[Route('/inscription', name: 'app_registration', methods: ['GET', 'POST'])]
public function registration(Request $request, EntityManagerInterface
$manager): Response
{
    $user = new User();
    $user->setRoles(['ROLE_USER']);
    $form = $this->createForm(RegistrationType::class, $user);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid())
    {
        $user = $form->getData();
        $this->addFlash(
            'success',
            'Bienvenue "'. $user->getFirstname() .'" , votre compte a bien
été créé !'
        );
        $manager->persist($user);
    }
}
```



```

        $manager->flush();
        return $this->redirectToRoute('app_home');
    }
    return $this->render('security/registration.html.twig', [
        'form' =>$form->createView()
    ]);
}

```

Je construis ma vue avec le fichier registration.html.twig :

```

{% extends "base.html.twig" %}
{% block title %}
BestBrico - Inscription
{% endblock %}
{% block body %}
<div class="content-container" style="margin-top: 80px;">
    <div class="container">
        <h1 class="mt-4">Formulaire d'inscription</h1>
        <hr class="my-2">
        {{ form(form, { 'attr': {
            "class": "mt-4"
        }}) }}
    </div>
</div>
{% endblock %}

```

J'effectue des tests de création d'utilisateurs, en simulant des erreurs, l'inscription est fonctionnelle.

Ensuite, je crée un formulaire de modification de mot de passe User.Type.php, une vue edit.html.twig où importe le formulaire, et pour que cela fonctionne, je crée le controller UserController.php.

Le contrôleur commence par vérifier si l'utilisateur est connecté. S'il ne l'est pas, il est redirigé vers la page de connexion.

Ensuite, il vérifie si l'utilisateur connecté est le même que celui dont le mot de passe doit être modifié. Cela empêche un utilisateur de modifier le mot de passe d'un autre utilisateur.

Gestion du Formulaire :

Un formulaire est créé et la requête est traitée. Cela permet à l'utilisateur de soumettre son mot de passe actuel et le nouveau mot de passe.

Validation et Mise à Jour du Mot de Passe :

- Si le formulaire est soumis et valide, le contrôleur vérifie si le mot de passe actuel est correct en utilisant le **UserPasswordHasherInterface**.
- Si le mot de passe est correct, le mot de passe de l'utilisateur est mis à jour, et un message de succès est affiché.
- Si le mot de passe actuel n'est pas correct, un message d'avertissement est affiché.

Les modifications de l'utilisateur sont enregistrées dans la base de données, et l'utilisateur est redirigé vers la page d'accueil.

```
#[Route('/utilisateur/edition/{id}', name: 'app_user', methods: ['GET', 'POST'])]
public function editPassword(User $user, Request $request,
EntityManagerInterface $manager, UserPasswordHasherInterface $hasher):
Response
{
    if (!$this->getUser())
    {
        return $this->redirectToRoute('app_login');
    }

    if($this->getUser() !== $user )
    {
        return $this->redirectToRoute('app_home');
    }

    $form = $this->createForm(UserType::class);

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()) {
        if ($hasher->isPasswordValid($user, $form-
>getData()['plainPassword'])) {
            $user->setUpdatedAt(new \DateTimeImmutable());
            $user->setPlainPassword(
                $form->getData()['newPassword']
            );
            $this->addFlash(
                'success',
                'Le mot de passe a bien été modifié. ');
            $manager->persist($user);
            $manager->flush();
            return $this->redirectToRoute('app_home');
        } else {
```

```

        $this->addFlash(
            'warning',
            'Le mot de passe est invalide.');
```

 }
}

 return \$this->render('user/edit.html.twig', [
 'form' => \$form->createView()
]);
}

4.7 Gestion des contacts

- Mise en place d'un formulaire de Contact ;
- Un controller ContactController dans le quel j'importe le formulaire, et les champs ;
- Deux vues dans contact, index.html.twig qui appelle le formulaire de contact, et contact.html.twig qui sera interprété dans la boîte email ;
- Des fixtures pour tester les données ;
- On configure notre environnement pour l'envoi de mails, j'utilise Mailtrap, je récupère le mailerDSN pour le fichier env :

```
MAILER_DSN=smtp://2f983e48ad9cc3:626ba7fd8b144f@sandbox.smtp.mailtrap.io:2525?
```

Symfony nous propose un grand choix de services que l'on peut voir avec la commande :

Php bin/console debug :autowiring --all

Et nous offre la possibilité de créer des services, ce qui permet de les réutiliser en allégeant la logique dans le controller. Je crée un fichier MailService.php dans un dossier Service/src

- Le service utilise TemplatedEmail de Symfony pour créer des e-mails basés sur des modèles Twig, et MailerInterface pour l'envoi des e-mails.
- Propriété \$mailer: Une instance de MailerInterface est stockée comme propriété privée. Cette interface est une partie du composant Mailer de Symfony et est utilisée pour envoyer des e-mails.
- Constructeur: Le service reçoit une instance de MailerInterface via l'injection de dépendances. Cette instance est ensuite assignée à la propriété \$mailer.

```
public function __construct(MailerInterface $mailer)
{
    $this->mailer = $mailer;
}
```

Méthode sendEmail :

```
public function sendEmail(
    string $from,
    string $subject,
    string $htmlTemplate,
    array $context,
    string $to = 'admin@primeur.com'
```

La méthode sendEmail prend plusieurs paramètres pour configurer l'e-mail :

```
$email = (new TemplatedEmail())
    ->from($from)
    ->to($to)
    ->subject($subject)
    ->htmlTemplate($htmlTemplate)
    ->context($context);
$this->mailer->send($email);
```

- Un objet TemplatedEmail est créé et configuré avec les paramètres fournis. Cette classe permet de créer facilement un e-mail basé sur un modèle Twig.
- Envoi de l'E-mail : L'e-mail configuré est envoyé en utilisant la méthode send de l'objet mailer.

J'importe la logique dans ContactController :

Tout ce qui est dans src peut être appelé comme un services (sauf exceptions), Symfony va créer un nouvel objet MailService

```
use App\Service\MailService;
```

```
$mailService->sendEmail(
    $contact->getEmail(),
    $contact->getSubject(),
    'contact/contact.html.twig',
    ['contact' => $contact]
);
```

L'entité Contact permettra aux utilisateurs d'envoyer un mail au gérant, un formulaire ContactType, avec un ReCAPTCHA pour protéger la boîte email d'envois de robots.

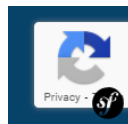
composer req karser/karser-recaptcha3-bundle

Je récupère les clés pour mon fichier :

```
RECAPTCHA3_KEY=6LeMyyspAAAAALZIBMiP30VtnvzNM2ORXP0NcOLL  
RECAPTCHA3_SECRET=6LeMyyspAAAAAGR3lP_8gfPXk5IzUF_PuVx2P_q_
```

J'ajoute un champ Recaptcha dans mon formulaire :

```
->add('captcha', Recaptcha3Type::class, [  
    'constraints' => new Recaptcha3(),  
    'action_name' => 'contact']);
```



4.8 Mise en place d'un système d'administration avec EasyAdmin

J'installe le bundle avec la commande :

composer req easycorp/easyadmin-bundle

Je crée mon dashboard :

Php bin/console make :admin:dashboard

Un dossier Admin est créé dans src/controller : DashboardController.php, je crée un template pour la vue dashboard.html.twig dans lequel on extends EasyAdmin

```
{% extends "@EasyAdmin/page/content.html.twig" %}  
{% block content %}  
<div class="container">  
    <div class="row">  
<h1>Bienvenue sur le panneau Administration</h1>  
<div>  
<a href="{{ path('app_home') }}" , class="badge bg-dark"  
title="Accueil">Revenir à l'accueil</a>  
    </div>  
    </div>  
</div>  
{% endblock %}
```

Ce **dashboard** va permettre à l'administrateur de gérer les produits, les contacts et les users.

```

public function configureMenuItems(): iterable
{
    yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
    yield MenuItem::linkToCrud('Utilisateurs', 'fas fa-user',
User::class);
    yield MenuItem::linkToCrud('Demandes de contact', 'fas fa-envelope',
Contact::class);
    yield MenuItem::linkToCrud('Produits', 'fa-solid fa-trowel-bricks',
Product::class);
    // yield MenuItem::linkToCrud('The Label', 'fas fa-list',
EntityClass::class);
}

```

Je crée les CrudControllers avec la commande :

php bin/console make:admin:crud

on choisit l'entité :

```

PS C:\Users\b_lou\Documents\workspace\Le Primeur de Laigneville\primeur> php bin/console make:admin:crud
Which Doctrine entity are you going to manage with this CRUD controller?:
[0] App\Entity\Category
[1] App\Entity>Contact

```

Je commence par Contact, dans ContactCrudController, je configure le Crud :

```

public function configureCrud(Crud $crud): Crud {
    return $crud
        ->setEntityLabelInPlural('Demandes de contact')
        ->setEntityLabelInSingular('Demande de contact')
        ->setPageTitle('index', 'Primeur de Laigneville - Administration
des demandes de contact')
        ->setPaginatorPageSize(15)
}

```

Je configure les champs :

```

public function configureFields(string $pageName): iterable {
    return [
        IdField::new('id')
            ->hideOnForm(),
        TextField::new('firstname'),
        TextField::new('email')
            ->setFormTypeOption('disabled', 'disabled'),
        TextField::new('subject'),
        TextareaField::new('message')
            ->setFormType(CKEditorType::class)
            ->hideOnIndex(),
        DateTimeField::new('createdAt'),
    ]
}

```

Mise en place d'un wysiwyg avec EasyAdmin :

composer req friendsofsymfony/ckeditor-bundle

php bin/console ckeditor:install --tag=4.22.1

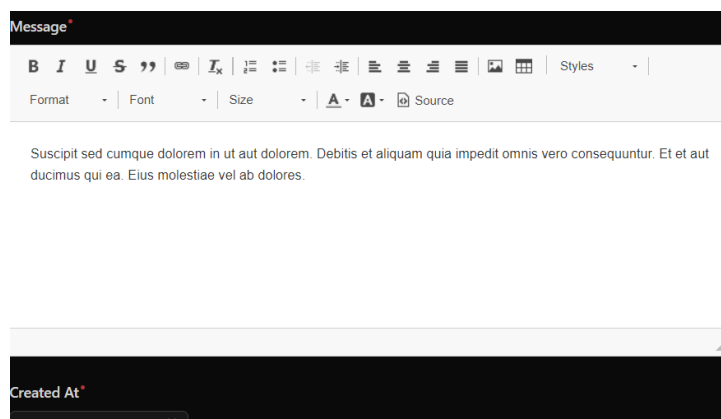
php bin/console assets:install public

Ajout dans configurecrud :

```
->addFormTheme('@FOSCKEditor/Form/ckeditor_widget.html.twig');
```

Ajout dans configureFields :

```
->setFormType(CKEditorType::class)
```



Je poursuis avec le crudcontroller pour les users et les products.

Je crée des fixtures pour l'admin :

```
$users = [];  
$admin = new User();  
$admin->setEmail('admin@primeur.com')  
->setRoles(["ROLE_USER", "ROLE_ADMIN"])  
->setFirstname('admin')  
->setPlainPassword('admin');  
  
$users[] = $admin;  
$manager->persist($admin);
```

Pour limiter l'accès et sécuriser les routes pour l'administrateur, dans DashboardController

```
use Symfony\Component\Security\Http\Attribute\IsGranted;  
class DashboardController extends AbstractDashboardController  
{  
    #[Route('/admin', name: 'admin')]  
    #[IsGranted('ROLE_ADMIN')]
```

Je teste l'accès pour un autre utilisateur.

Je modifie la navbar afin que seul l'administrateur puisse avoir les accès avec une condition

```
{% if 'ROLE_ADMIN' in app.user.roles %}
    <div class="dropdown-divider"></div>
    <a class="dropdown-item" href="{{ path("admin") }}"><font
style="vertical-align: inherit;"><font style="vertical-align:
inherit;">Panneau administrateur</font></font></a>
    </div>
{% endif %}
```

DASHBOARD

Primeur de Laigneville - Administration

Dashboard

Utilisateurs

Demandes de contact


Produits

Q Search

admin@primeur.com

Primeur - Administration des produits

Add Product

	ID ↑	Name ↕	Category ↕	Image ↕	
<input type="checkbox"/>	1	omnis est deserunt quibusdam	1 FRUITS		...
<input type="checkbox"/>	2	consequatur quibusdam eaque ratione	3 EPICERIE	Null	...
<input type="checkbox"/>	3	est adipisci debitis non	2 LEGUMES	Null	...
<input type="checkbox"/>	4	cupiōtate itaque et quae	2 LEGUMES	Null	...
<input type="checkbox"/>	5	in sunt hic quod	1 FRUITS	Null	...
<input type="checkbox"/>	6	debitis omnis dolorem autem	1 FRUITS	Null	...
<input type="checkbox"/>	7	in non nihil commodi	3 EPICERIE	Null	...
<input type="checkbox"/>	8	error fugit inventore illum	3 EPICERIE	Null	...
<input type="checkbox"/>	9	asperiores eum nihil vel	3 EPICERIE	Null	...
<input type="checkbox"/>	10	sint illum odio dolorum	3 EPICERIE	Null	...
<input type="checkbox"/>	11	velit accusamus harum sit	1 FRUITS	Null	...
<input type="checkbox"/>	12	odit voluptas sed queraat	2 LEGUMES	Null	...
<input type="checkbox"/>	13	qui eveniet maxime expedita	1 FRUITS	Null	...
<input type="checkbox"/>	14	maiores doloremque enim quo	2 LEGUMES	Null	...
<input type="checkbox"/>	15	minima asperiores nesciunt commodi	2 LEGUMES	Null	...

100 results

< Previous

1

2

3

4

5

6

7

Next >

4.9 Création d'une liste course

L'utilisateur pourra créer sa liste de course, il aura la possibilité de l'imprimer ou de l'envoyer par mail

Dans le MCD, j'ai défini 2 entités que nous allons utiliser `ShoppingList` et `ShoppingListItem`, avec la relation `OneToMany`. Je crée un formulaire `ShoppingListType`, que j'appellerai dans le contrôleur `ShoppingListController`, qui nous retournera la vue dans `list.html.twig`.

Fonctionnalités de la Liste de Courses

- **Gestion de la Liste:** Les utilisateurs peuvent ajouter, modifier, et supprimer des articles dans leur liste de courses.
- **Partage par Email:** Implémentation d'une fonctionnalité pour envoyer la liste de courses par email, utilisant le **MailService** de Symfony pour une communication efficace.
- **Impression de la Liste:** Intégration d'une option pour imprimer la liste de courses, augmentant la praticité de l'outil pour les utilisateurs.

Dans le contrôleur, il y aura plusieurs routes :

```
#[Route('/liste', name: 'app_liste')]  
public function liste
```

```
#[Route('/add/{id}', name: 'app_add')]  
public function add
```

```
#[Route('/remove/{id}', name: 'app_remove')]  
public function remove
```

```
#[Route('/delete/{id}', name: 'app_delete')]  
public function delete
```

```
#[Route('/empty', name: 'app_empty')]  
public function empty
```

Pour imprimer la liste ,

```
<button onclick="window.print()" class="btn btn-outline-primary me-4"  
title="Imprimer votre liste"><i class="fas fa-print cart-icon"></i></button>
```

Pour envoyer ma liste par mail, je créer une route dans le ContactController et j'utilise MailService que j'ai utilisé précédemment

```
#[Route('/envoyer-liste', name: 'envoyer_liste')]
public function MailListe(ProductRepository $productRepository, Request
$request, MailService $mailService, SessionInterface $session): Response
{
    $user = $this->getUser();
    if (!$user) {
        // Rediriger vers la page de connexion ou afficher un message
d'erreur
        $this->addFlash('error', 'Vous devez être connecté pour envoyer la
liste.');
```

```
        return $this->redirectToRoute('app_login');
    }
    // Utilisez l'email de l'utilisateur connecté
    $userEmail = $user->getEmail();

    // Récupérer liste de produits ici
    $listeSession = $session->get('liste', []);
    $listeProduits = [];

    foreach ($listeSession as $id => $quantity) {
        $product = $productRepository->find($id);
        if ($product) {
            $listeProduits[] = [
                'product' => $product,
                'quantity' => $quantity
            ];
        }
    }

    $mailService->sendEmail(
        'email@exemple.com',
        'Votre liste de produits',
        'liste/email.html.twig',
        ['liste' => $listeProduits],
        $userEmail
    );

    $this->addFlash('success', 'Votre liste a été envoyée par email.');
```

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

L'utilisateur doit être connecté pour envoyer sa liste, le mailservice récupère le tableau de la liste dans email.html.twig. Je teste l'envoi de mail avec Mailtrap, ça fonctionne.

4.10 Création des promos.

Promo.php : titre, description, prix et imagefile (avec vich)

Dans vich_uploader.yaml

```
promo_images:
  uri_prefix: /images/promos
  upload_destination: '%kernel.project_dir%/public/images/promos'
  namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
```

PromoType.php :formulaire pour créer, modifier les données

```
->add('imageFile', VichImageType::class, [
    'label' => 'Image de la promo',
    'label_attr' => [
        'class' => 'form-label mt-4'
    ],
    'required' => false
```

Pour la sécurité

```
class PromoType extends AbstractType
{
    private $token;

    public function __construct(TokenStorageInterface $token)
    {
        $this->token = $token;
    }
}
```

PromoController pour voir les promos pour tous les utilisateurs,

```
#[Route('/promo', name: 'app_promo', methods: ['GET'])]
```

créer, modifier et supprimer uniquement par l'admin

```
#[IsGranted('ROLE_ADMIN')]
#[Route('/promo/creation', 'new_promo', methods: ['GET', 'POST'])]
public function new
#[IsGranted('ROLE_ADMIN')]
#[Route('/promo/edition/{id}', 'edit_promo', methods: ['GET', 'POST'])]
public function edit
#[IsGranted('ROLE_ADMIN')]
#[Route('/promo/suppression/{id}', 'delete_promo', methods: ['GET',
'POST'])]
public function delete
```

Pour les vues, il y a 3 fichiers

- Index.html.twig, accessibles à tous les utilisateurs
- Edit.html.twig et new.html.twig, qui appellent le formulaire sécurisé avec

```
{{ form_row(form._token) }}
```

J'effectue plusieurs tests de création, de modification et de suppression.

J'ajoute un accès dans la navbar accessible par l'admin

```
<a class="dropdown-item" href="{{ path('new_promo') }}">Créer une Promo</a>
```

Et 2 boutons pour modifier ou supprimer qui ne s'affichent seulement si c'est l'administrateur qui est connecté.



4.11 Création de Services

Un aspect crucial du projet était de présenter les services offerts par le Primeur de Laigneville de manière claire et engageante.

Présentation des Services

- **Interface Utilisateur:** Création d'une section dédiée sur le site pour mettre en avant les différents services proposés, tels que la livraison à domicile, les offres spéciales, etc.
- **Design et Accessibilité:** Utilisation de cartes Bootstrap dans la conception pour une navigation intuitive et un design attrayant.

4.12 Barre de recherche

Je crée un nouveau dossier Model et un nouveau fichier SearchData.php

```
<?php

namespace App\Model;

class SearchData
{
    /** @var string */
    public $q = ''; // le terme de recherche
}
```

Dans ProductRepository, j'écris une nouvelle fonction qui fait appel à la nouvelle classe SearchData

```
public function findBySearch(SearchData $searchData)
{
    $query = $this->createQueryBuilder('p');

    if (!empty($searchData->q)) {
        $query = $query
            ->andWhere('p.name LIKE :q')
            ->setParameter('q', "%{$searchData->q}%");
    }

    return $query->getQuery()->getResult();
}
```

Dans le controller Home, j'ajoute une route /search et j'utilise la fonction findBySearch

```
#[Route('/search', name: 'app_search', methods: ['GET'])]
public function search(Request $request, ProductRepository
$productRepository): Response
{
    $searchData = new SearchData();
    $searchData->q = $request->query->get('q', '');

    $products = $productRepository->findBySearch($searchData);

    return $this->render('search/search.html.twig', [
        'products' => $products,
        'searchData' => $searchData,
    ]);
}
```

Je crée la vue search.html.twig qui renvoie le résultat de la recherche si elle trouve un résultat, sinon pas de produits trouvés.

Et j'ajoute le lien au bouton recherche dans la navbar

```
<form class="d-flex" action="{ path('app_search') }" method="get">
```

Je teste la fonctionnalité, et finalise le visuel du résultat.

4.13 Mot de passe oublié

En utilisant [MakerBundle](#) et [SymfonyCastsResetPasswordBundle](#), on peut créer une solution sécurisée prête à l'emploi pour gérer les mots de passe oubliés. Tout d'abord, j'installe SymfonyCastsResetPasswordBundle :

composer require symfonycasts/reset-password-bundle

Ensuite, j'utilise la **make:reset-passwordcommande**. Cela me pose quelques questions sur mon application et génère tous les fichiers dont j'ai besoin.

```
created: src/Controller/ResetPasswordController.php
created: src/Entity/ResetPasswordRequest.php
updated: src/Entity/ResetPasswordRequest.php
created: src/Repository/ResetPasswordRequestRepository.php
updated: src/Repository/ResetPasswordRequestRepository.php
updated: config/packages/reset_password.yaml
created: src/Form/ResetPasswordRequestFormType.php
created: src/Form/ChangePasswordFormType.php
created: templates/reset_password/check_email.html.twig
created: templates/reset_password/email.html.twig
created: templates/reset_password/request.html.twig
created: templates/reset_password/reset.html.twig

Success!
```

Next:

- 1) Run **"php bin/console make:migration"** to generate a migration for the new **"App\Entity\ResetPasswordRequest"** entity.
- 2) Review forms in **"src/Form"** to customize validation and labels.
- 3) Review and customize the templates in **"templates/reset_password"**.
- 4) Make sure your **MAILER_DSN** env var has the correct settings.
- 5) Create a **"forgot your password link"** to the **app_forgot_password_request** route on your login form.

Then open your browser, go to **"/reset-password"** and enjoy!

Je suis les remarques de Symfony :

- J'effectue la migration et le migrate pour mettre à jour la nouvelle entité ResetPasswordRequest ;
- Je modifie le formulaire de modification de mot de passe et lui intègre les contraintes avec les Regex :

```
'constraints' => [  
    new Regex([  
        'pattern' => "/^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{8,10}$/",  
        'message' => "Il faut minimum minimum huit et  
maximum 10 caractères, au moins une lettre majuscule, une lettre  
minuscule, un chiffre et un caractère spécial"  
    ]),  
],
```

- Je customise les templates pour un meilleur rendu sur les vues ;
- Je mets à jour le formulaire de connexion en ajouter un lien de **mot de passe oublié** et met à jour le chemin :

```
<a href="{{ path('app_forgot_password_request') }}" class="btn btn-link">Mot  
de passe oublié ?</a>
```

J'effectue plusieurs tests pour changer le mot de passe d'un utilisateur, avec des bonnes et des informations erronées pour être sûr que tout fonctionne.

5. TESTS

5.1 Présentation

Il existe deux différents types de tests automatisés dans symfony: les **tests unitaires** et les **tests fonctionnels**.

Les tests unitaires vérifient que chaque méthode et chaque fonction fonctionne correctement. Chaque test doit être aussi indépendante que possible des autres.

D'autre part, des tests fonctionnels vérifient que l'application résultante se comporte correctement dans son ensemble.

Tous les tests dans symfony sont situés sous le répertoire test/ du projet. Il contient deux sous-répertoires, un pour les tests unitaires (test/unit/) et un pour les tests fonctionnels (test/functional/).

- php bin/phpunit : pour lancer le test,
- php bin/console make:test : pour créer un test.

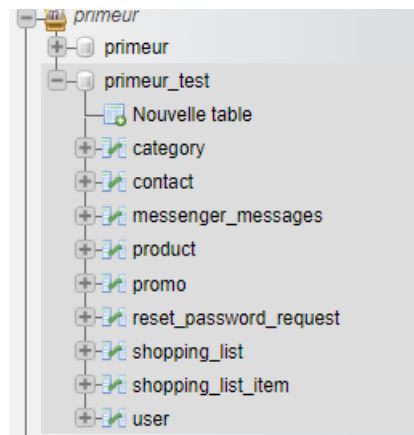
5.3 L'environnement

Pour effectuer des tests sur mon application, je configure le fichier `.env.test.local` :

```
DATABASE_URL="mysql://root@127.0.0.1:3306/primeur?serverVersion=8.0.30&charset=utf8mb4"
MAILER_DSN=smtp://2f983e48ad9cc3:626ba7fd8b144f@sandbox.smtp.mailtrap.io:2525?
RECAPTCHA3_ENABLED=0
```

Et je lance la commande pour créer une base de données de test :

php bin/console d:d:c --env=test



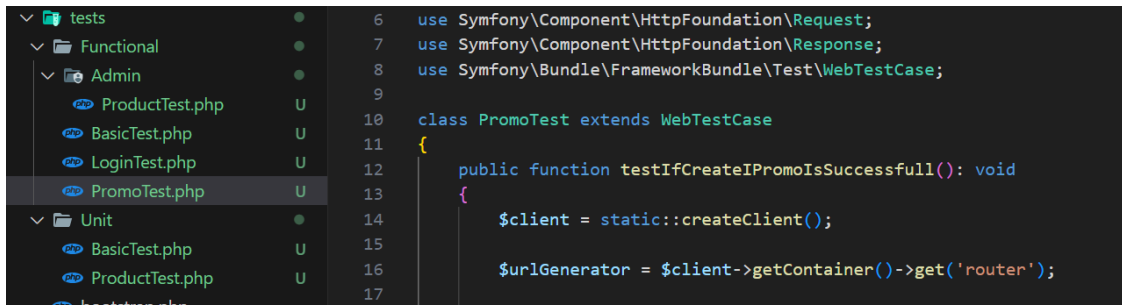
Je mets en place un script pour la commande « tests » :

```
"tests":
[
    "php bin/console d:d:d --force --if-exists --env=test",
    "php bin/console d:d:c --env=test",
    "php bin/console d:s:u --no-interaction --force --env=test",
    "php bin/console d:f:l --no-interaction --env=test",
    "php bin/phpunit --testdox tests/Unit/",
    "php bin/phpunit --testdox tests/Functional/"
]
```

5.4 Création de tests.

```
PS C:\Users\b_lou\Documents\workspace\Le Primeur de Laigneville\primeur> php bin/console make:test

Which test type would you like?:
[TestCase] basic PHPUnit tests
[KernelTestCase] basic tests that have access to Symfony services
[WebTestCase] to run browser-like scenarios, but that don't execute JavaScript code
[ApiTestCase] to run API-oriented scenarios
[PantherTestCase] to run e2e scenarios, using a real-browser or HTTP client and a real web server
> []
```

a. Test unitaire

1. Test de base pour vérifier que la configuration de test est fonctionnelle.

```

namespace App\Tests\Unit;
use PHPUnit\Framework\TestCase;
class BasicTest extends TestCase
{
    public function testSomething(): void
    {
        $this->assertTrue(true);
    }
}

```

2. Test sur l'entité Product.

b. Test fonctionnel

1. Test de base

```

namespace App\Tests\Functional;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
class BasicTest extends WebTestCase
{
    public function testSomething(): void
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/');
        $this->assertResponseIsSuccessful();
    }
}

```

2. La connexion.
3. New promo.
4. Produits dans EasyAdmin.

```

PS C:\Users\b_lou\Documents\workspace\Le Primeur de Laigneville\primeur> php bin/phpunit
PHPUnit 9.6.15 by Sebastian Bergmann and contributors.

Testing
.....
8 / 8 (100%)

Time: 00:04.190, Memory: 86.00 MB

OK (8 tests, 15 assertions)

```

6. CONCLUSION

À travers ce stage, j'ai pu appliquer mes compétences en développement web dans un contexte réel et sur un projet concret, en travaillant sur des fonctionnalités variées comme la liste de courses, la gestion des contacts, la barre de recherche, la mise en place d'un dashboard, et la présentation des services. J'ai utilisé un grand nombre d'outils, ce qui m'a permis de me familiariser un peu plus avec eux. Cette expérience a été enrichissante tant sur le plan professionnel que personnel, et a solidifié mon désir de poursuivre ma carrière dans le domaine du développement web.

Cela n'a pas été toujours facile, les problématiques faisaient partie de mon quotidien, et relever ces défis était un challenge intéressant et plaisant, l'envie d'apprendre et de maîtriser continuant de grandir en moi. J'ai pu vérifier l'adage « Il faut pratiquer », et c'est complètement avéré, plus on pratique, plus le sens et la compréhension prennent le pas.

Avec une liste de projets déjà en tête et une aspiration à l'indépendance professionnelle, je suis prêt à embrasser l'avenir avec enthousiasme et détermination.

La suite, c'est persévérer, et poursuivre mon apprentissage :

- Continuer de travailler mon anglais pour être plus à l'aise avec la documentation, et m'ouvrir quelques portes supplémentaires,
- Entamer des démarches pour lancer une activité à mon compte, après m'être libéré de mes obligations professionnelles actuelles.
- Valider une spécialisation, ce qui retient mon attention aujourd'hui, c'est la data qui offre un grand choix de possibilités, notamment le conseil. Je dois continuer à m'exercer, prendre de l'expérience au travers de missions, et aussi capitaliser sur mes expériences acquises durant toutes ces années.