

# **Chapitre 1 -Introduction générale**

## **1.1. Contexte et objectifs du projet**

Le développement du e-commerce connaît une croissance continue, notamment dans des secteurs comme la vente de maillots de football personnalisables.

Le projet « Maillot de Foot » s'inscrit dans le cadre de ma formation de Développeur Web et Web Mobile. Il a pour objectif de mettre en pratique l'ensemble des compétences du référentiel REAC à travers la conception et le développement d'une application web moderne.

L'application a été pensée comme une boutique e-commerce spécialisée dans les maillots de football. J'ai effectué ce choix car il m'est apparu en tant qu'utilisateur de ce type de site, qu'il me serait plus aisés à reproduire, tout d'abord en m'inspirant d'eux pour comprendre la logique dans la conception d'un site e-commerce mais également parce que par leur fréquentation j'ai pu cerner les limites de certains d'entre eux ce qui me semblait une promesse d'améliorations possibles.

Ce choix thématique répond aussi à un cas d'usage courant (site marchand), tout en offrant une base pédagogique suffisamment conséquente pour manipuler des concepts variés : catalogue de produits, gestion du panier, comptes utilisateurs, adresses. L'enjeu principal était de créer un site permettant aux utilisateurs :

- de consulter un catalogue de clubs et de maillots.
- d'afficher les détails de chaque produit.,
- d'ajouter des articles dans un panier.
- de gérer un compte utilisateur et ses adresses.
- de simuler un processus de commande.

Ce projet nécessite la maîtrise conjointe du front-end et du back-end, ainsi que l'intégration des aspects liés à la sécurité, à l'ergonomie et à la gestion de projet.

## **1.2. Enjeux utilisateurs et métier**

Du point de vue métier, l'application doit offrir une expérience de type e-commerce adaptée à un public ciblé (supporters et/ou amateurs de football, collectionneurs de maillots).

Du point de vue utilisateur, les enjeux étaient :

- Simplicité d'utilisation : navigation intuitive, interface claire, parcours d'achat fluide.
- Rapidité et réactivité : grâce à Inertia et React, qui évitent les rechargements complets de page.
- Accessibilité : compatibilité avec divers supports (desktop et mobile).
- Sécurité : gestion fiable des comptes et sessions utilisateurs, protection des données personnelles.

## **1.3. Périmètre et limites du projet**

Le périmètre de l'application comprend :

- la gestion des clubs et maillots (catalogue et fiches produits).
- l'affichage des fiches produits avec images.
- la gestion du panier (ajout, mise à jour, suppression, vidage).
- la gestion d'un compte utilisateur (authentification, adresses).
- la simulation d'un processus de commande.

En revanche, certaines fonctionnalités n'ont pas pu être implémentées dans cette version par manque de temps :

- La gestion avancée d'un espace administrateur.
- L'intégration d'un véritable système de paiement en ligne.
- La mise en place d'un module de livraison complet.
- La possibilité pour l'utilisateur d'ajouter des maillots dans une wish list.

## **1.4. Méthode et organisation**

Le projet a été mené de manière incrémentale et agile :

- Découpage en fonctionnalités prioritaires (MVP : catalogue, panier, compte utilisateur). J'ai dans un premier temps concentré mon effort à la mise en place des fonctionnalités de base. J'ai débuté par la création du compte utilisateur côté client (inscription , connexion, détails du compte, adresses). J'ai ensuite travaillé sur la partie catalogue pour terminer avec le panier.

Le **MVP** (*Minimum Viable Product*, ou « produit minimum viable » en français) est une méthode de développement qui consiste à **définir et réaliser uniquement les fonctionnalités essentielles** pour que l'application soit utilisable par les premiers utilisateurs. L'idée consiste à ne pas tout développer dès le départ mais à se concentrer sur un noyau fonctionnel minimal. Par la suite, on pourra enrichir le projet de fonctionnalités supplémentaires.

Cette façon de procéder permet d'obtenir rapidement une première version utilisable. Les utilisateurs peuvent tester le site et émettre des retours. J'ai ainsi pu valider les choix technologiques et la faisabilité technique de mon projet :

- Mise en place progressive du front-end et du back-end.

- Utilisation de Git pour la gestion des versions (versioning) et la collaboration (avec moi-même il est vrai mais il m'est apparu nécessaire de créer des branches afin de procéder à des essais sans affecter le code initial).

Tests manuels réguliers pour valider chaque étape du parcours des utilisateurs.

Cette méthodologie m'a permis de conserver une vision produit claire tout en respectant les contraintes pédagogiques.

J'ai également scindé la conception de mon travail en deux parties. Je me suis dans un premier temps concentré sur la partie liée au compte utilisateur côté client. Quand il m'a paru que les fonctionnalités essentielles du compte utilisateur côté client (connexion, inscription , détails du compte, adresse) avaient été développées, j'ai estimé que je devais désormais m'occuper de la partie produit du site, avec le développement de la pages listant les maillots des clubs et des équipes nationales et celui de la page référençant le maillot sélectionné par l'utilisateur et la page panier.

Je n'ai pas encore développé la partie administrateur cependant je pense établir un rôle en utilisant un middleware (prévu dans le MCD, MLD,MPD) avec des autorisations étendues par rapport à un utilisateur lambda, lui offrant la possibilité de gérer le site (commandes, stock, ajout/retrait des produits, modification produit, accès aux comptes utilisateurs).

L'utilisation d'un middleware dans le cadre de mon projet contribuera pour ce rôle d'administrateur :

- **à la centralisation de la sécurité** : Un middleware permet d'appliquer un contrôle d'accès à tous les points d'entrée (contrôleurs, routes) de la partie administrateur sans devoir répéter les vérifications dans chaque fonction. Cela constitue la solution la plus sûre et la plus propre.
- **Évolutif et maintenable** : Je pourrai faire évoluer la gestion des rôles facilement (plusieurs rôles ou permissions) sans toucher à la logique métier de chaque page.
- **Bonne pratique Laravel** : Cela suit les conventions et recommandations et offre la possibilité de réutiliser la structure pour d'autres rôles/parties privées ultérieurement.

## Chapitre 2 -Vue d'ensemble technique

### 2.1. Stack technologique

Mon projet Fou2Foot repose sur une architecture monolithique **Laravel** intégrant **Inertia.js** et **React**.

Cette approche me permet de bénéficier de la réactivité d'une application moderne, le front et le back cohabitent dans un même environnement applicatif.

- Laravel 11.9 (PHP 8.3.14) : framework<sup>1</sup> back-end, gestion du routage, logique métier, accès BDD via Eloquent ORM.
- Inertia.js : pont entre Laravel et React.
- React : framework front-end, création d'interfaces dynamiques et réactives.
- Tailwind CSS : framework CSS utilitaire, garantissant un design responsive et homogène.
- Vite : outil de build rapide et moderne pour le front-end.
- MySQL : système de gestion de base de données relationnelle.

---

<sup>1</sup> Framework : un framework est un ensemble de composants logiciels réutilisables qui permettent de développer de nouvelles applications plus efficacement.

## 2.2. Architecture de l'application

### a) Architecture

L'architecture suit le modèle MVC (modèle, vue, contrôleur) de Laravel, enrichi par l'intégration Inertia/React :

Côté serveur (Laravel)

- *Models* : représentent les entités métier (User, Club, Maillot, Cart, Address).
- *Controllers* : assurent la logique métier (ex. CartController pour gérer le panier).
- *Migrations* : définissent la structure des tables et leurs relations.
- *Routes* : exposent les points d'entrée (parcours public, parcours utilisateur connecté).

Côté client (React via Inertia)

- *Pages* : chaque route est liée à une page React (Home, MaillotDetail, Cart, AccountDetails...).
- *Composants* : éléments réutilisables (Header, Footer, WelcomeMessage, PanierLink).
- *Styles* : Tailwind pour la mise en forme et la responsivité.

Cette organisation permet :

- une séparation claire entre logique serveur et interface,
- une expérience fluide côté utilisateur (navigation sans rechargement complet),
- une évolutivité (possibilité d'ajouter de nouvelles pages et composants facilement).

### b) Schéma de fonctionnement

#### 1. Requête utilisateur

Lorsqu'un utilisateur accède à une page (ex. /maillots), la route Laravel correspondante est appelée via le fichier routes/web.php.

Exemple :

```
// Routes pour les maillots
Route::get('/maillots/{id}', [MaillotController::class, 'show'])->name('maillots.show');
```

## 2. Traitement côté serveur

Le contrôleur exécute la logique métier (récupération des maillots, pagination, filtrage par club, etc.), via les **modèles Eloquent** :

MaillotController.php

```
14 |     $maillot = Maillot::with('club')->findOrFail($id);
15 |
16 |     // ajouter les tailles, quantités
17 |     $tailles = ['S', 'M', 'L', 'XL'];
18 |     $quantite = 1200; // ou récupère depuis la BDD
19 |
20 |     return Inertia::render('MaillotDetail', [
21 |         'maillot' => $maillot,
22 |         'tailles' => $tailles,
23 |         'quantite' => $quantite,
24 |         'nom' => $maillot->nom,
25 |         'numero' => $maillot->numero,
26 |         'prix' => 20,
27 |         'prix_nom' => 3,
28 |         'prix_numero' => 2,
29 |     ]);

```

## 3. Transmission des données via Inertia.js

Inertia agit comme une “passerelle” entre Laravel et React.

Les données envoyées par le contrôleur sont automatiquement transformées en **props React** et injectées dans le composant MaillotsList.jsx.

## 4. Rendu et interactions côté client (React)

Le composant React affiche les données et gère l’interactivité (filtrage, clics, ajouts au panier...).

Toute action utilisateur (soumission de formulaire, suppression d'article, etc.) est renvoyée vers les routes Laravel via Inertia, sans rechargement complet de page :

```
21  function handleAddToCart() {
22    router.post('/cart/add', [
23      maillot_id: maillot.id,
24      size: taille,
25      quantity: qte,
26      numero: personnalisation.numero ? numero : null,
27      nom: personnalisation.nom ? nom : null,
28    ], {
29      onSuccess: () => alert("Maillot ajouté au panier !"),
30    });
31 }
```

## 5. Mise à jour dynamique

Inertia ne recharge que le **composant concerné**, simulant un comportement SPA (Single Page Application) tout en restant dans un **monolithe Laravel unique**.

Cela offre la possibilité d'obtenir une **navigation fluide**, la **cohérence des sessions Laravel** (authentification, middleware) et une **sécurité unifiée**.

## 6. Les avantages de cette approche

**Simplicité de déploiement** : une seule application à héberger.

**Sécurité renforcée**.

**Performances** : Vite et React assurent un rendu rapide et interactif.

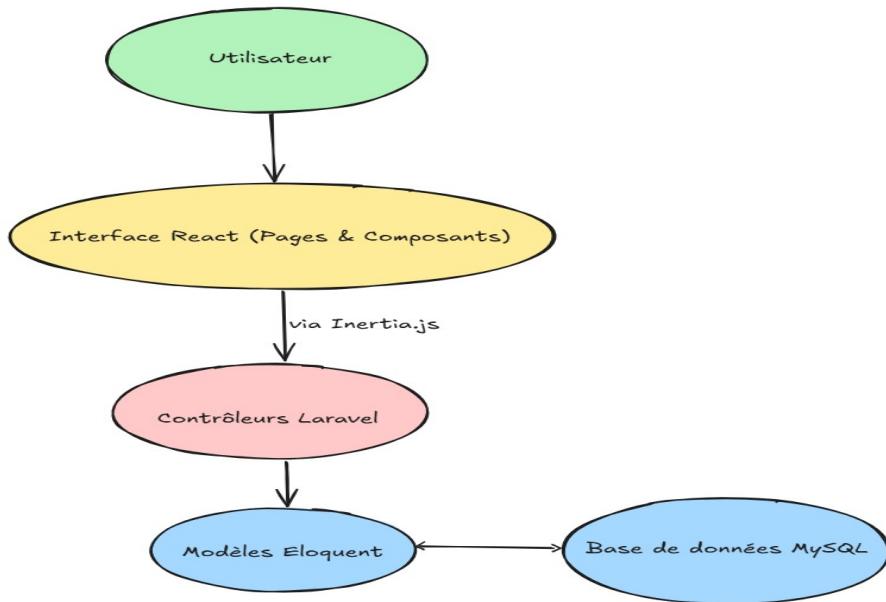
**Cohérence des données** : Laravel gère la logique métier et la validation, React se concentre sur l'affichage.

## Conclusion

Ce choix architectural permet de bénéficier de la **souplesse du front React** tout en conservant la **stabilité et la sécurité d'un framework back-end complet**.

Laravel centralise les données, la validation et les règles métier, tandis qu'Inertia.js fait office de **passerelle** vers une interface fluide et moderne.

### c) Schéma global d'architecture



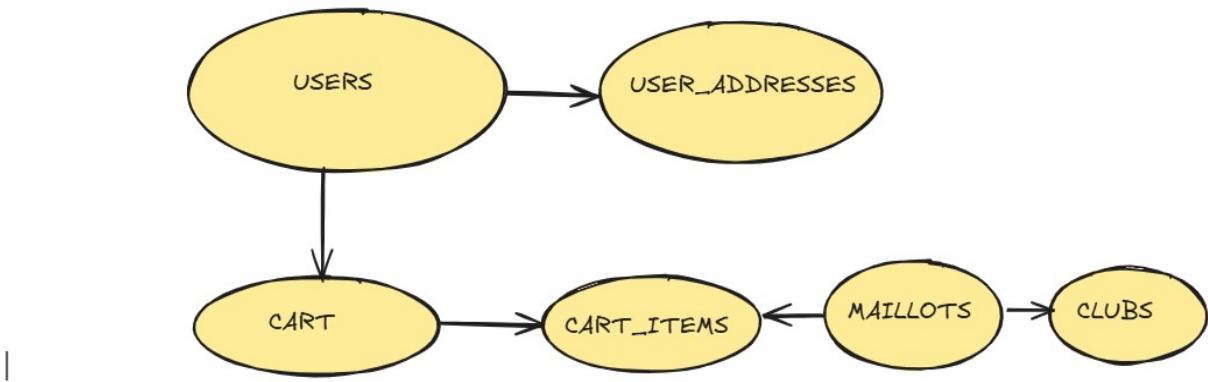
**Front-end (React)** : gère l'affichage et l'interaction utilisateur.

**Inertia.js** : transmet les données entre Laravel et React sous forme de props.

**Back-end (Laravel)** : applique la logique métier, valide les données et interagit avec la base.

**Base de données (MySQL)** : stocke les informations persistantes (utilisateurs, clubs, maillots, adresses, paniers).

### d) Schéma de base de données (simplifié)



### Explication des relations :

- Un User peut avoir plusieurs Adresses (relation N-N via user\_addresses).
- Un User possède un Cart (panier).
- Un Cart contient plusieurs CartItems (articles panier), chacun lié à un Maillot.
- Un Maillot appartient à un Club.

### 2.3. Qualité et sécurité

La sécurité fait partie des enjeux prioritaires de la conception d'une application. Plusieurs mécanismes intégrés à Laravel et aux outils front-end m'ont aidé à assurer la fiabilité du système :

- **Authentification** : le middleware auth protège l'accès aux zones privées (compte, adresses, panier). Seuls les utilisateurs connectés peuvent accéder à ces fonctionnalités.

```

app > Http > Middleware > AuthSessionMiddleware.php > ...
6  use Illuminate\Http\Request;
7  use Symfony\Component\HttpFoundation\Response;
8
9  class AuthSessionMiddleware
10 {
11     /**
12      * Handle an incoming request.
13      *
14      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next
15      */
16     public function handle(Request $request, Closure $next): Response
17     {
18         $sessionId = $request->cookie('session-id');
19
20         if (!$sessionId) {
21             return redirect()->route('Login');
22         }
23
24         $session = \App\Models\UserSession::with('user')
25             ->where('id', $sessionId)
26             ->where('expires_at', '>', now())
27             ->first();
28
29         if (!$session) {
30             return redirect()->route('Login')->withoutCookie('session-id');
31         }

```

- **Sessions utilisateurs** : un suivi des connexions est assuré grâce à la table dédiée user\_sessions, permettant de tracer les activités et de renforcer le contrôle d'accès.

```
app > Models > UserSession.php > UserSession
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6  use Illuminate\Database\Eloquent\Relations\BelongsTo;
7
8  class UserSession extends Model
9  {
10     protected $fillable = [
11         'id',
12         'user_id',
13         'expires_at',
14         'ip_address',
15         'user_agent',
16         'last_activity',
17     ];
18
19     protected $casts = [
20         'expires_at' => 'datetime',
21         'last_activity' => 'datetime',
22         'created_at' => 'datetime',
23     ];
24
25     public $incrementing = false;
26     protected $keyType = 'string';
```

- **Validation des données** : les formulaires sont systématiquement validés côté serveur (Laravel) et côté client (React). Cela réduit les risques d'erreurs et empêche l'injection de données malveillantes.

- **Protection CSRF** : elle est activée par défaut dans Laravel ; elle sécurise tous les formulaires en générant un jeton unique associé à chaque session.

- **Gestion des relations en base** : les contraintes d'intégrité référentielle (clés primaires et étrangères) sont mises en place via les migrations, garantissant la cohérence des données.

Au-delà de ces mécanismes techniques, j'ai tenté dans la conception de mon projet d'appliquer les pratiques de développement destinées à renforcer la maintenabilité et la lisibilité du code :

- **Utilisation d'Eloquent ORM** : Plutôt que d'écrire des requêtes SQL brutes, mon projet s'appuie sur Eloquent ORM (Object Relational Mapping).

Les Avantages d'Eloquent ORM :

- **Maintenance** : les évolutions de la base (ex. ajout d'un champ) sont facilement prises en charge via les migrations et les modèles.
- **Sécurité** : les injections SQL sont évitées grâce à la protection native de Laravel.
- **Lisibilité** : les relations sont exprimées sous forme d'objets (\$user->useraddresses, \$club → maillots, \$cart->items ), ce qui rend le code plus compréhensible.

\$user → addresses: permet de récupérer toutes les adresses liées à un utilisateur. Dans modèle User.php :

```
app > Models > User.php > User
16  class User extends Authenticatable
39      protected $casts = [
40          'email_verified_at' => 'datetime',
41          'birth_date' => 'date',
42          'is_active' => 'boolean',
43          'created_at' => 'datetime',
44          'updated_at' => 'datetime',
45      ];
46
47      // Relations
48      public function sessions(): HasMany
49      {
50          return $this->hasMany(UserSession::class);
51      }
52
53      public function addresses(): HasMany
54      {
55          return $this->hasMany(UserAddress::class);
56      }
```

\$club → maillots : permet de récupérer tous les maillots associés à un club.

Dans modèle Club.php :

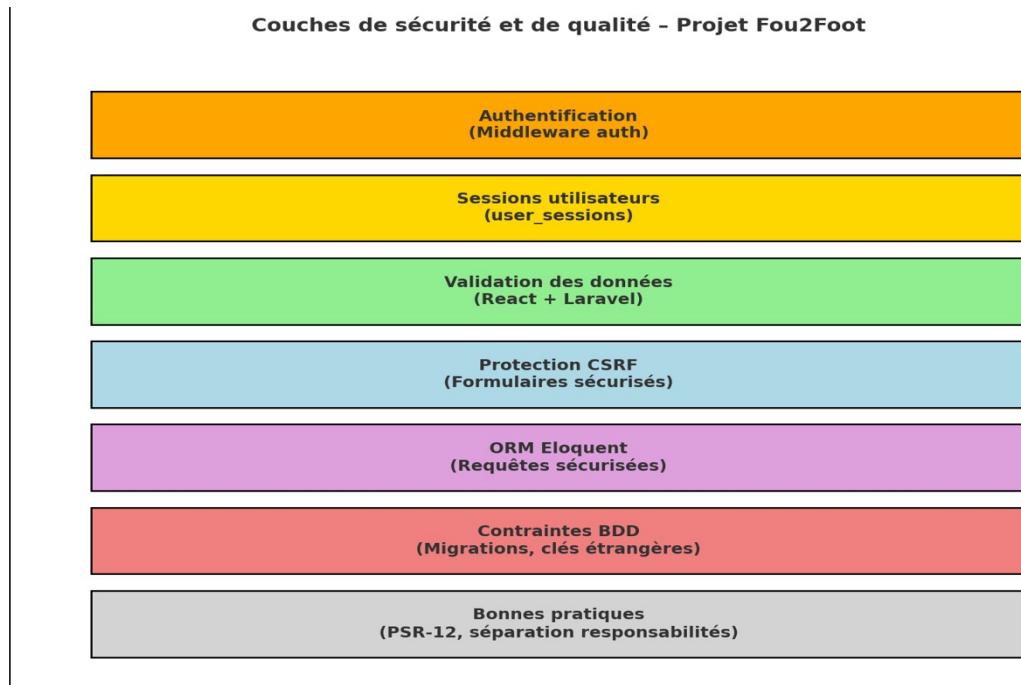
```

app > Models > Club.php > Club
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Club extends Model
8  {
9      public function maillots()
10     {
11         return $this->hasMany(Maillot::class);
12     }
13 }
14

```

- **Séparation des responsabilités** : l'architecture du projet suit le principe *Separation of Concerns*. Les contrôleurs gèrent la logique applicative, les modèles manipulent les données, et les composants React/Tailwind assurent la présentation. Cette structuration rend le projet plus facile à maintenir et à tester.

En combinant **mécanismes de sécurité natifs** et **bonnes pratiques de développement**, l'application bénéficie d'un socle fiable, sécurisé et durable, garantissant à la fois la protection des données et la facilité d'évolution du projet.



# **Chapitre 3 -Développement de la partie front-end sécurisée**

## **3.1. CP1 -Installer et configurer l'environnement de travail**

Notre formateur nous a conseillé différents outils afin de mettre en place un environnement qui devait garantir un développement fluide et conforme aux standards. Cette étape initiale était déterminante pour la structuration du projet.

### **a) Outils et versions utilisées**

#### **Outil / Technologie Version**

PHP	8.3.14
Laravel	11.9
Node.js	20.17.0
npm	10.8.2
React	18.3.1
React-DOM	18.3.1
TailwindCSS	3.4.10
Vite	5.0
MySQL	8.0
Inertia.js	1.3.0

### **Étapes d'installation**

#### **1. Clonage du dépôt Git :**

```
git clone https://github.com/utilisateur/Lam_Maillot_de_Foot.git  
cd Lam_Maillot_de_Foot
```

#### **2. Configuration des variables d'environnement :**

```
cp .env.example .env
```

Personnalisation des accès base de données.

#### **3. Installation des dépendances backend :**

```
composer install
```

#### 4. Installation des dépendances frontend :

```
npm install
```

#### 5. Génération de la clé d'application Laravel :

```
php artisan key:generate
```

#### 6. Migration et alimentation initiale de la base :

```
php artisan migrate --seed
```

#### 7. Lancement des serveurs de développement :

```
php artisan serve (serveur Laravel)
```

```
npm run dev (build et hot reload côté front)
```

En production, la commande suivante est utilisée :

```
npm run build
```

Les scripts npm permettent de piloter la compilation du front-end :

- npm run dev lance le serveur de développement avec **HMR** (Hot Module Replacement<sup>2</sup>), offrant un rechargement instantané et fluide.
- npm run build génère les fichiers optimisés pour la production, garantissant des performances maximales grâce à la minification<sup>3</sup> et à la suppression des classes inutilisées.

### b) Organisation des sources

- resources/js/Pages : pages React (ex. MaillotsList.jsx, MaillotDetail.jsx).
- resources/js/Components : composants réutilisables (boutons, formulaires, liens).
- resources/css : styles globaux centralisés.

---

<sup>2</sup> recharge à chaud sans rechargement complet de page.

<sup>3</sup> Minification : processus de suppression de tous les caractères inutiles du code source JavaScript sans altérer sa fonctionnalité

### c) Finalités offertes par la configuration

Cette configuration m'a offert la possibilité de :

- développer efficacement avec **React + Tailwind** sans surcharge CSS.
- profiter de la navigation fluide via **Inertia.js**.
- compiler et tester rapidement grâce à **Vite** (HMR :Hot Module Replacement ) : avec **Vite**, dès la modification d'un fichier (JS/TS/React/Tailwind), celui-ci **réinjecte** uniquement le module modifié dans le navigateur **sans recharger toute la page**. Cela me permet d'obtenir un cycle de développement ultra-rapide avec un état des composants conservé et un retour immédiat.
- disposer d'une base de données prête dès les premiers tests grâce aux migrations et seeders Laravel.

### 3.2. CP2 -Maquetter une application

J'ai réfléchi à la maquette en amont afin de me guider le développement. Elle s'appuie sur les **principaux parcours utilisateurs** :

- **Accueil** : vitrine du site et accès rapide au catalogue.
- **Catalogue des maillots** : affichage en grille, filtrage par clubs.
- **Fiche produit** : visuels du maillot, description, prix, disponibilité, bouton "ajouter au panier".
- **Panier** : liste des articles ajoutés, quantités modifiables, total calculé automatiquement.
- **Authentification** : page de connexion et d'inscription.
- **Espace utilisateur/ tableau de bord** : gestion des adresses, suivi des commandes.

L'outils de prototypage utilisé ( Figma en l'occurrence) m'a aidé à orienter mes choix pour présenter une interface ergonomique. J'avais pour objectif d'obtenir une **navigation simple** (header, footer), des **cartes produit homogènes** (visuel, nom, prix), des **formulaires clairs** (connexion, adresse).

La phase de conception visuelle a été réalisée afin de structurer l'interface et de préparer le travail front-end. Cette étape a suivi un processus progressif allant du **zoning à la maquette**.

- **Le zoning** est la représentation schématique de la page en zones fonctionnelles (header, menu, contenu, footer). L'objectif est d'organiser les blocs principaux sans se soucier du design graphique.
- **le wireframe** constitue l'étape suivante du maquettage. Il s'agit d'une version plus détaillée du zoning, dans laquelle on affine et on place les éléments (boutons, images, textes, formulaires) sans couleurs ni styles définitifs. Le wireframe précise la disposition et les interactions prévues.
- La dernière étape du processus consiste à réaliser la **maquette**. Elle est la représentation graphique finalisée, incluant couleurs, polices, images et styles. C'est une projection fidèle du rendu attendu dans l'application

J'ai suivi les étapes de conception préconisées ci-dessus. J'ai débuté la réalisation de mon maquettage par un zoning en utilisant Figma.

Avec le Zoning, j'ai identifié les zones principales de l'application :

- une zone header (logo + navigation),
- une zone dédiée au contenu (liste des maillots, fiches produits, panier),
- une zone footer (informations légales, contacts).

Cette étape a été cruciale pour me servir de base à mon wireframe.

Avec le Wireframe, j'ai réalisé des croquis fonctionnels ( en utilisant Figma) pour représenter la disposition des éléments, notamment une liste de produits en grille, une fiche produit avec photo, prix et boutons d'action, un panier accessible depuis chaque page.

Les choix des couleurs ne correspondaient pas encore à cette étape à des choix définitifs mais avaient pour fonction d'établir un visuel de ma future structure. Néanmoins en concevant mon site, j'ai procédé à de nombreux réajustements et surtout à de nouvelles orientations esthétiques.

De plus, ainsi que je l'ai souligné précédemment, je me suis inspiré de plusieurs sites de vente en ligne de maillots de football notamment le site footbebe. J'ai repris l'enchainement des pages, les éléments à insérer dans le tableau de bord, l'accueil, le catalogue, le détail d'un maillot.

Leur site me paraissait plus cohérent que ceux de leurs concurrents, la navigation entre les pages répondait à une logique et se révélait fluide. L'ergonomie de leur site m'a également semblé un garant de leur sérieux, de leur professionnalisation.

L'exhaustivité de leur catalogue m'a également paru un indicateur de fiabilité car si les produits proposés sont nombreux, il est vraisemblable que le site est capable de répondre à la demande et la prend en charge.

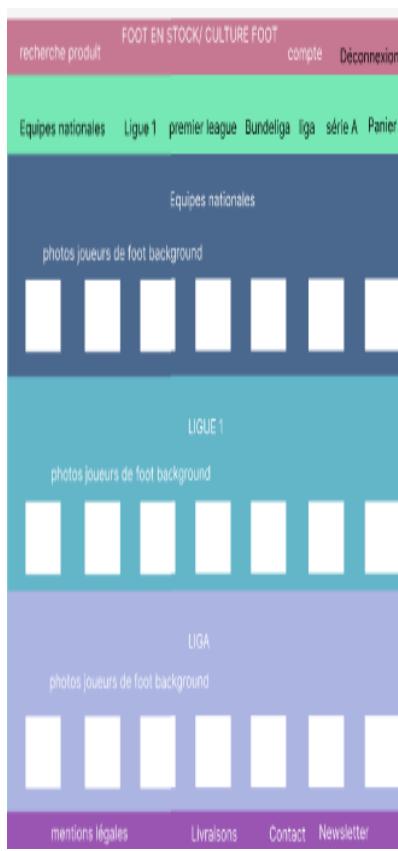
Le nom de mon site n'était pas encore déterminé au moment du wireframe. Je l'ai modifié ultérieurement pour Fou2Foot lors de l'étape finale du maquettage.

Avec les maquettes Figma j'ai enfin créé des interfaces graphiques intégrant la palette de couleurs choisie (liée à l'univers du football), la typographie adaptée à une lecture claire et dynamique, les icônes (HeroIcons) et composants réutilisables (boutons, formulaires).

## WIREFRAME

### PAGE ACCUEIL

Version mobile



Version desktop



## Connexion/ Inscription

Version mobile

Frame 27

recherche produit FOOT EN STOCK/ CULTURE FOOT compte Déconnexion

Equipes nationales Ligue 1 premier league Bundesliga liga série A Panier

S'enregistrer Connexion

email  identifiant ou email

identifiant  Mdp

Mdp  Valider

se souvenir de moi

Valider

Frame 34

mentions légales Livraisons Contact Newsletter

This mobile version of the login page features a registration section on the left and a login section on the right. Both sections include input fields for email/username and password, and a 'Valider' button. The registration section also includes a 'se souvenir de moi' checkbox. The entire form is set against a red background.

Version desktop

Frame 27

recherche produit FOOT EN STOCK/ CULTURE FOOT compte Déconnexion

Equipes nationales Ligue 1 premier league Bundesliga liga série A Panier

S'enregistrer Connexion

email  identifiant ou email

identifiant  Mdp

Mdp  Valider

se souvenir de moi

Valider

Frame 34

mentions légales Livraisons Contact Newsletter

This desktop version of the login page follows a similar layout to the mobile one, with registration on the left and login on the right. It includes the same input fields and buttons, with the 'Valider' button being blue. The registration section has a 'se souvenir de moi' checkbox. The desktop version uses a green header and a yellow footer.

## Tableau de bord

Version mobile

Version desktop

frame 30

recherche produit	FOOT EN STOCK/ CULTURE FOOT	compte	Déconnexion
Equipes nationales	Ligue 1 premier league Bundesliga liga série A Panier		
NAVIGATION	TABLEAU DE BORD		
Ma Commande	Ma Commande	Adresses	
Adresses			
Détails du compte	Détails du compte	Ma wishlist	
Ma wishlist			
Déconnexion	Activités récentes		
Frame 31			
mentions légales	Livrasons	Contact	Newsletter

frame 30

recherche produit	FOOT EN STOCK/ CULTURE FOOT	compte	Déconnexion
Equipes nationales	Ligue 1 premier league Bundesliga liga série A Panier		
NAVIGATION	TABLEAU DE BORD		
Ma Commande	Ma Commande	Adresses	
Adresses			
Détails du compte	Détails du compte	Ma wishlist	
Ma wishlist			
Déconnexion	Activités récentes		
Frame 31			
mentions légales	Livrasons	Contact	Newsletter

## Liste Maillots

Version desktop

Frame 25

recherche produit

FOOT EN STOCK/ CULTURE FOOT

compte

Déconnexion

Equipes nationales   Ligue 1   premier league   Bundesliga   liga   série A   Panier

NOM CLUB/ EQUIPE NATIONALE

descriptions succinctes du produit

Frame 33

mentions légales

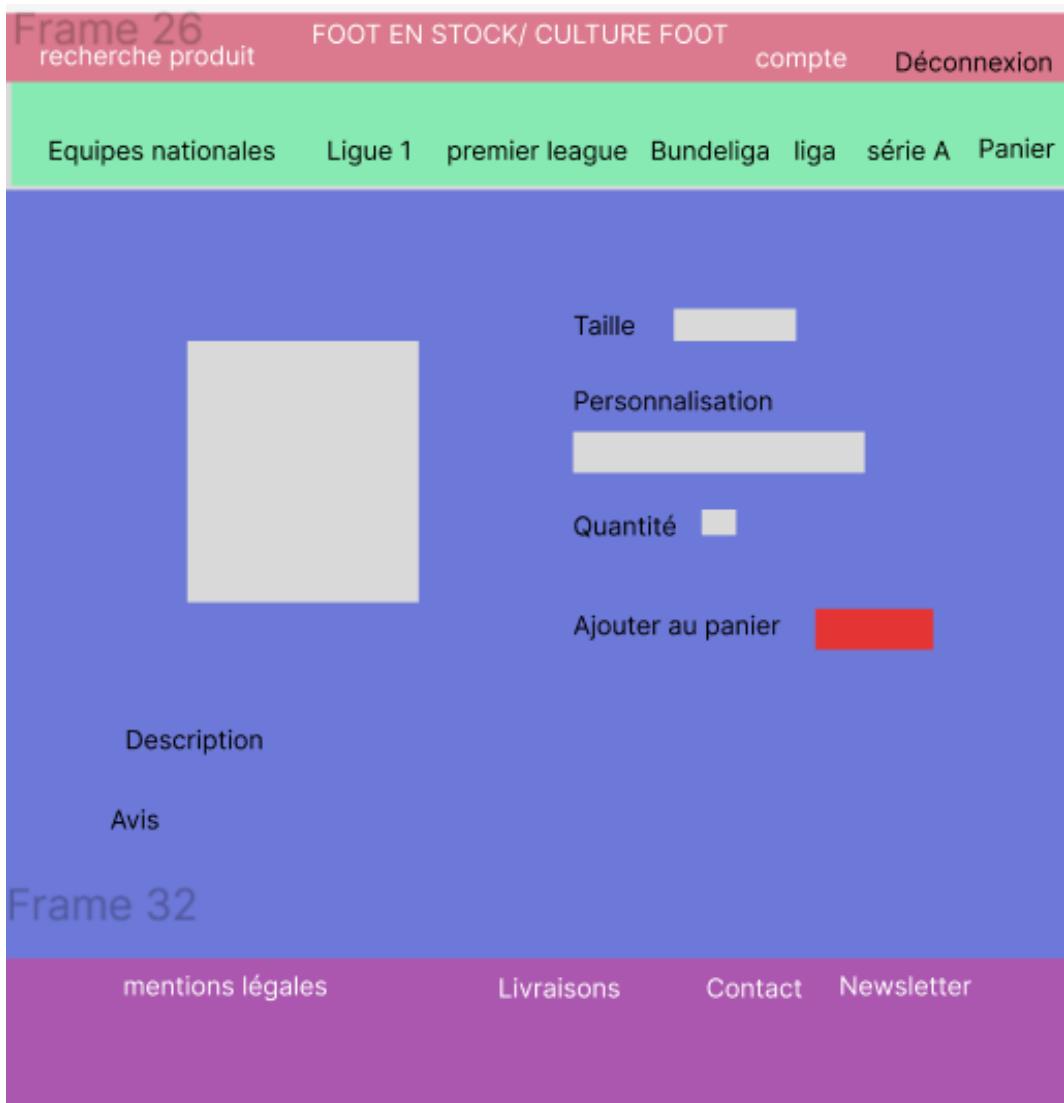
Livrasons

Contact

Newsletter

## Détail maillot

Version Desktop



## Commande

J'avais initialement pensé à réaliser une page récapitulant les commandes effectuées par un utilisateur, cependant la conception de mon site s'est avérée plus chronophage que je l'avais escomptée, si bien que je n'ai produit cette page qu'en front.

Je présente néanmoins la page prévue ci-dessous et que je compte réaliser ultérieurement avec le code couleur de mon site e-commerce.

**Commande.** Version desktop

The wireframe for the 'Commande' page is divided into several sections:

- Header:** A purple header bar with the text "Frame 28" and "recherche produit" on the left, and "FOOT EN STOCK/ CULTURE FOOT", "compte", and "Déconnexion" on the right.
- Navigation:** A green navigation bar below the header containing links for "Equipes nationales", "Ligue 1", "premier league", "Bundeliga", "liga", "série A", and "Panier".
- Left Column (Blue Area):** Labeled "Détails facturations". It contains fields for "ID", "NOM" (with placeholder "Prénom"), "adresse", "email", "téléphone", "N° Carte de paiement" (with placeholder "Expiration" and "N° CVC"), and a "total" field.
- Right Column (Green Area):** Labeled "Détails commande". It contains a "articles" section.
- Bottom Navigation:** A red footer bar with the text "Frame 35" and links for "mentions légales", "Livraisons", "Contact", and "Newsletter".

Initialement prévue, je n'ai pas également encore réalisé la page WishList car elle ne m'a pas paru faire partie des fonctionnalités à développer dans le cadre du MVP.

La réalisation de mon wireframe achevée, j'avais une idée plus précise quant à la structuration de mon projet et à la navigation de l'utilisateur sur le site. Elle a confirmé mes choix, guidé mon travail

et facilité l'étape suivante du maquettage que l'on doit théoriquement pouvoir présenter au client afin que celui-ci puisse la valider.

## MAQUETTE

version desktop

tableau de bord

The screenshot shows the Fou2Foot desktop dashboard. At the top, there's a navigation bar with the Fou2Foot logo, a search bar, and links for 'Mon compte' and 'Déconnexion'. Below the navigation bar is a main content area titled 'Mon Tableau de Bord'. This area includes sections for 'Mes Commandes' (with a link to 'Suivez vos achats passés et en cours.'), 'Adresse' (with a link to 'Gérez votre adresse de livraison.'), 'Détails du compte' (with a link to 'Modifiez vos informations personnelles.'), and 'Ma wishlist' (with a link to 'Retrouvez vos articles favoris.'). On the left side, there's a sidebar titled 'Navigation' containing links for 'Tableau de bord', 'Commandes', 'Adresse', 'Détails du compte', 'Ma wishlist', and 'Se déconnecter'. At the bottom, there's a footer with the Fou2Foot logo, a 'Informations' section (links to 'Mentions légales', 'Confidentialité', and 'CGU/CGV'), a 'Service client' section (links to 'Livraisons', 'Retours', and 'Contact'), and a 'Nous suivre' section with social media icons for Twitter and Facebook. A copyright notice at the very bottom reads '© 2025 Fou2Foot - Tous droits réservés'.

## version mobile

The screenshot shows the mobile version of the Fou2Foot website's user dashboard. At the top, there is a navigation bar with the Fou2Foot logo, a search bar, and account links for "Mon compte" and "Déconnexion". Below the navigation, a horizontal menu includes "Accueil", "Sélections Nationales", "Ligue 1", "Premier League", "Bundesliga", "Liga", "Série A", and "Autres". On the left, a sidebar titled "Navigation" lists "Tableau de bord", "Commandes", "Adresse", "Détails du compte", "Ma wishlist", and "Se déconnecter". The main content area features a blue header box with the text "Bienvenue, marion !" and "Votre Email : marion@free.fr". Below this is a section titled "Mon Tableau de Bord" with four cards: "Mes Commandes" (Follow your past and current purchases), "Adresse" (Manage your shipping address), "Détails du compte" (Edit your personal information), and "Ma wishlist" (Find your favorite articles). Further down, a section titled "Activité récente" shows a message about a recent purchase and another about an address update. At the bottom, there is a footer with links for "Fou2Foot", "Informations" (Mentions légales, Confidentialité, CGU/CCV), "Service client" (Livraisons, Retours, Contact), and "Nous suivre" (social media icons for YouTube and Twitter).

### **3.3. CP3 -Réaliser une interface utilisateur statique et adaptable**

Après la phase de maquettage, j'ai décidé de commencer par la partie front du projet suivant les recommandations de notre formateur.

Il a donc fallu transformer ces maquettes en interfaces statiques fonctionnelles, en respectant les principes d'accessibilité, de responsive design. et de cohérence graphique définis lors de la conception (notamment pour l'architecture de la page). Concernant ce dernier point, mes aspirations ont évolué. Alors qu'initialement je comptais utiliser des couleurs unies pour le wireframe, j'ai finalement opté pour l'utilisation d'un dégradé rouge à bleu pour le header et un dégradé violet à bleu pour l'arrière-plan (background). Ce code couleur me paraissait plus moderne, plus frais, moins monotone qu'un fond blanc.

Cette étape correspond à la mise en œuvre concrète des maquettes, sans intégrer encore la logique dynamique ou les traitements métier.

#### **a) Pages statiques :**

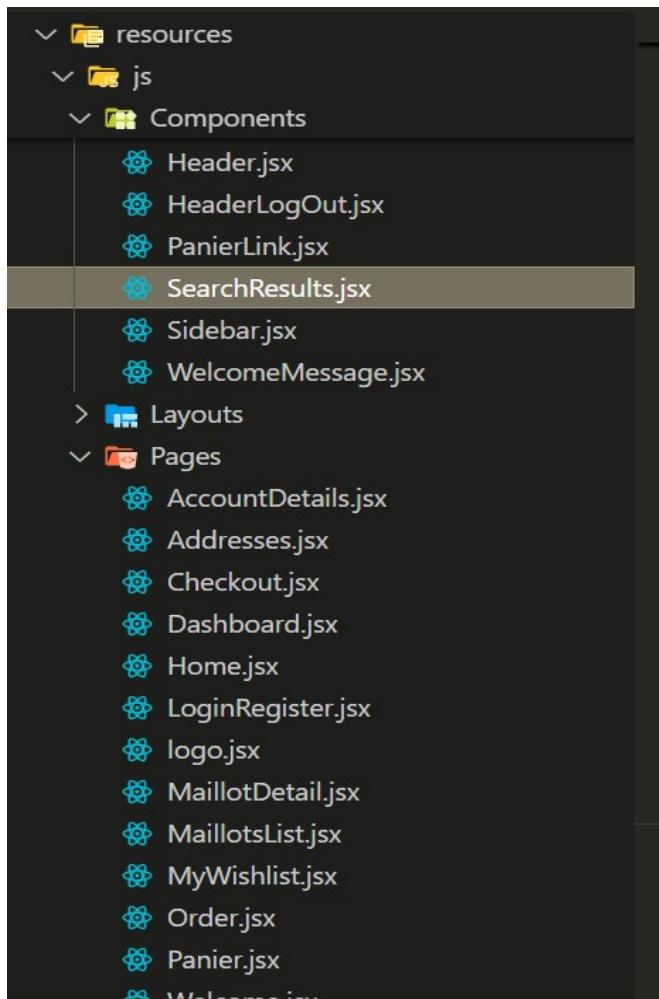
- Home.jsx (page d'accueil qui correspond à la vitrine du site, accessible hors connexion).
- LoginRegister.jsx et Register.jsx. (connexion et inscription d'un nouvel utilisateur)
- Dashboard.jsx (Tableau de bord).
- Addresses.jsx (facturation + livraison car un utilisateur peut commander et faire livrer un maillot en cadeau à une adresse différente de la sienne).
- AccountDetails.jsx (détails du compte).
- MaillotsList.jsx (catalogue filtré) pour la visualisation du catalogue.
- MaillotDetail.jsx (fiche produit) pour l'affichage des maillots.
- Panier.jsx (produit sélectionné par l'utilisateur).

#### **b) Composants réutilisables :**

- Header.jsx et Footer.jsx pour la navigation,
- WelcomeMessage.jsx afin d'afficher le nom et l'email de l'utilisateur dans le tableau de bord,
- Sidebar.jsx pour avoir accès au dashboard sur les pages de l'utilisateur.

- PanierLink.jsx pour le résumé du panier accessible depuis toutes les pages.

- SearchResult.jsx (pour la barre de recherche),



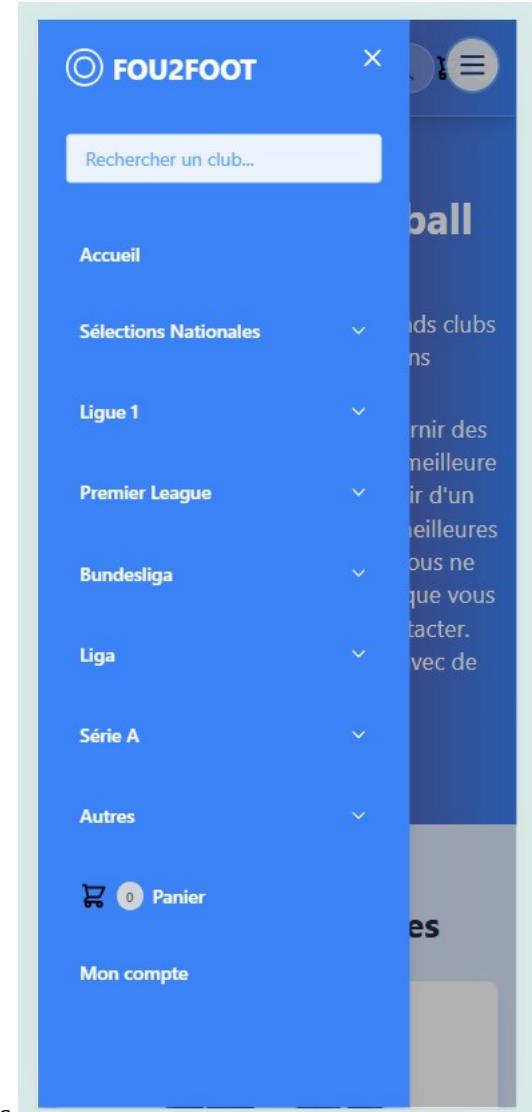
### c) Présentation des pages front-end

On peut simuler ici la navigation future d'un utilisateur à partir des pages front-end

**Page Accueil** (ne nécessite pas de connexion car c'est la vitrine du site)

The screenshot shows a web browser window with the URL 127.0.0.1:8000 in the address bar. The page has a blue header with the title "Maillots de Football Officiels" and a subtext "Retrouvez les tenues des plus grands clubs et des plus grandes sélections". Below the header, there is a paragraph of text: "Nous nous engageons à vous fournir des maillots de football officiels de la meilleure qualité , personnalisables, à partir d'un catalogue complet recensant les meilleures ligues, avec livraison rapide. Si vous ne trouvez pas le maillot de football que vous aimez, n'hésitez pas à nous contacter. Mises à jours régulières du site avec de nouveaux maillots". The main content area features four football jerseys displayed in separate boxes, labeled "Nos Maillots Phares". The jerseys shown are: 1. A dark blue jersey with a large white chevron on the chest. 2. A white jersey with a large dark blue chevron on the chest. 3. A grey jersey with red and blue stripes on the sleeves and a red Emirates logo on the chest. 4. A dark blue jersey with red stripes on the sleeves and a red Emirates logo on the chest.

Design responsive avec menu burger



pages responsives voir annexes

\*Autres

## Page inscription

The screenshot shows a web browser window with the URL 127.0.0.1:8000/login. The 'Inscription' tab is active. The form contains fields for 'Nom d'utilisateur', 'Email', 'Mot de passe', and 'Confirmer le mot de passe'. Each password field has an eye icon to its right. A large blue 'S'inscrire' button is at the bottom.

Nom d'utilisateur	<input type="text"/>
Email	<input type="text"/>
Mot de passe	<input type="password"/> ⚀
Confirmer le mot de passe	<input type="password"/> ⚀
<b>S'inscrire</b>	

Fou2Foot

Informations

Service client

Nous suivre

## Page connexion/ Inscription

The screenshot shows a web browser window with the URL 127.0.0.1:8000/login. Both the 'Connexion' and 'Inscription' tabs are visible. The form is identical to the one in the previous screenshot, containing fields for 'Nom d'utilisateur', 'Email', 'Mot de passe', and 'Confirmer le mot de passe'. A large blue 'S'inscrire' button is at the bottom.

Nom d'utilisateur	<input type="text"/>
Email	<input type="text"/>
Mot de passe	<input type="password"/> ⚀
Confirmer le mot de passe	<input type="password"/> ⚀
<b>S'inscrire</b>	

Fou2Foot

Informations

Service client

Nous suivre

## Tableau de bord

127.0.0.1:8000/dashboard

FOU2FOOT

Rechercher un club...

Mon compte | Déconnexion | Panier (7)

Accueil | Sélections Nationales | Ligue 1 | Premier League | Bundesliga | Liga | Série A | Autres

**Navigation**

- Tableau de bord
- Commandes
- Adresse
- Détails du compte
- Ma wishlist
- Se déconnecter

Bienvenue, mina !  
Votre Email : mina@carp.com

## Mon Tableau de Bord

**Mes Commandes**  
Suivez vos achats passés et en cours.

**Adresse**  
Gérez votre adresse de livraison.

**Détails du compte**  
Modifiez vos informations personnelles.

**Ma wishlist**  
Retrouvez vos articles favoris.

Activité récente

## Détails du compte

127.0.0.1:8000/accountdetails

Bienvenue, mina !  
Votre Email : mina@carp.com

## Mon compte

Gérez vos informations personnelles et vos préférences

**Informations personnelles** Modifier

Identifiant mina	Nom complet MINA BELLA
Email mina@carp.com	Téléphone 77 33 11 88 66

**Sécurité** Changer le mot de passe

## Adresses

The screenshot shows the FOU2FOOT website interface. The top navigation bar includes links for Accueil, Sélections Nationales, Ligue 1, Premier League, Bundesliga, Liga, Série A, and Autres. A search bar and a 'Mon compte' link are also present. On the left, a sidebar titled 'Navigation' lists Tableau de bord, Commandes, Adresse (which is selected), Détails du compte, Ma wishlist, and Se déconnecter. The main content area is titled 'Mes adresses' and displays a section for 'Adresse de facturation' (Billing address) for a contact named 'MINA BELLA' with the details: rue gandolphe, 06210 Mandelieu, FR, 77 33 11 88 66. Buttons for 'Modifier' and 'Supprimer' are shown at the bottom of this section.

## Liste Maillots d'un club

The screenshot shows the FOU2FOOT website interface for the Olympique Lyonnais club page. The top navigation bar and sidebar are identical to the previous screenshot. The main content area is titled 'Maillots de Olympique Lyonnais' and features a sub-header: 'Découvrez notre collection de maillots Olympique Lyonnais - 8 maillots disponibles'. Below this, four jerseys are displayed in a row: 'Domicile 2024' (white jersey with red and blue accents), 'Extérieur 2024' (black jersey with red and white accents), 'maillot third' (white jersey with red accents), and 'Maillot 75 ans' (light grey jersey with red accents).

## Présentation Maillot



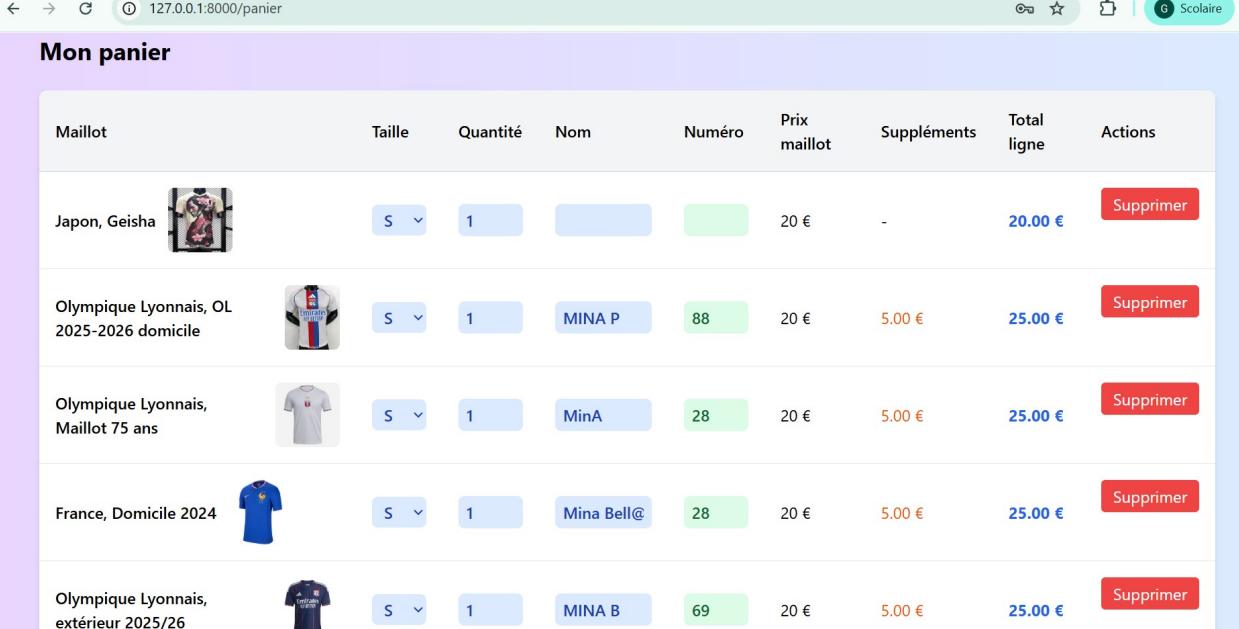
**Olympique Lyonnais**  
Domicile 2024

Prix : 20 €  
Type : Maillot  
Quantité disponible : 1200  
Taille : S  
Quantité : 1  
 Ajouter un numéro (+2 €)  
 Ajouter un nom (+3 €)

**Total: 20 €**

**AJOUTER AU PANIER**

## Panier de l'utilisateur



Maillot	Taille	Quantité	Nom	Numéro	Prix maillot	Suppléments	Total ligne	Actions
Japon, Geisha	S	1			20 €	-	20.00 €	<b>Supprimer</b>
Olympique Lyonnais, OL 2025-2026 domicile	S	1	MINA P	88	20 €	5.00 €	25.00 €	<b>Supprimer</b>
Olympique Lyonnais, Maillot 75 ans	S	1	MinA	28	20 €	5.00 €	25.00 €	<b>Supprimer</b>
France, Domicile 2024	S	1	Mina Bell@	28	20 €	5.00 €	25.00 €	<b>Supprimer</b>
Olympique Lyonnais, extérieur 2025/26	S	1	MINA B	69	20 €	5.00 €	25.00 €	<b>Supprimer</b>

#### d) Outils et technologies utilisés

- **React 18.3.1** pour la création de composants modulaires et réutilisables.
- **TailwindCSS 3.4.10** pour appliquer rapidement une mise en forme responsive et homogène. TailwindCSS permet aux classes utilitaires (flex, grid, p-4, md:w-1/2, etc.) de garantir un affichage adapté sur mobiles, tablettes et ordinateurs.
- **Inertia.js 1.3.0** pour intégrer les vues React au sein du projet Laravel sans rechargement complet de la page.

#### e) Accessibilité :

- Utilisation de balises sémantiques (<nav>, <main>, <footer>),
- Respect du contraste entre texte et fond,
- Champs de formulaires correctement labellisés avec **aria label**.

Exemple dans C:\wamp64\www\Lam\_Maillot\_de\_foot\resources\js\Pages\LoginRegister.jsx

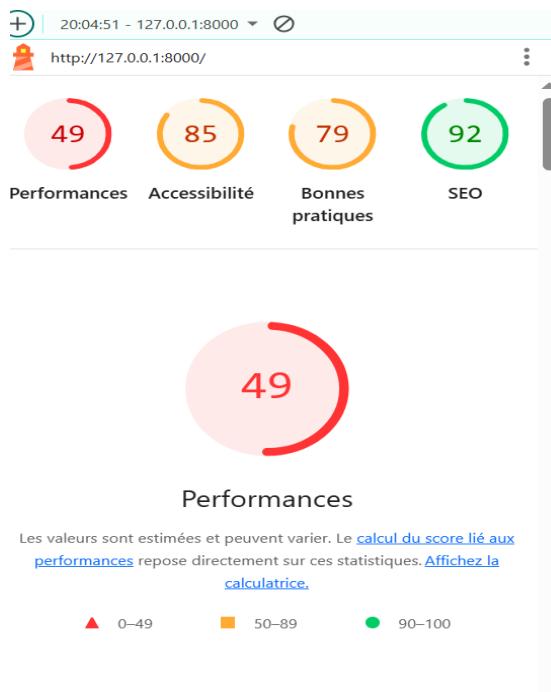
```
152           <button
153             type="button"
154             onClick={() => setShowConfirm(!showConfirm)}
155             className="absolute inset-y-0 right-0 px-3 flex items-center"
156             aria-label={showConfirm ? "Masquer la confirmation du mot de passe" : "Afficher la confirmation"
157           }
158             <EyeIcon show={showConfirm} />
159           </button>
```

#### f) Lighthouse

Un audit **Lighthouse** a été réalisé via Google Chrome sur la page d'accueil de l'application (<http://127.0.0.1:8000/>), ainsi que sur la page connexion/ inscription (<http://127.0.0.1:8000/login> ). L'audit est destiné à évaluer la qualité technique et l'accessibilité de l'interface. L'audit a été exécuté en environnement local (mode développement, sans optimisation de build ni compression serveur).

Les scores obtenus sont les suivants :

## Score lighthouse accueil



## Interprétation et axes d'amélioration page accueil

### Performances (49)

Le score moyen est lié à l'exécution en environnement de développement. En **mode production** (npm run build), les fichiers sont minifiés et purgés par **Vite** et **TailwindCSS**, ce qui réduit considérablement le poids des ressources. Le déploiement sur un serveur **Nginx** avec **compression gzip** et **mise en cache** devrait améliorer le score au-delà de 85.

### Accessibilité (85)

Le site répond déjà à plusieurs critères du **RGAA** : navigation clavier fluide, contraste suffisant, structure sémantique cohérente (header, main, footer), alt descriptifs sur les images, **aria- label** sur les icônes.

Les axes d'amélioration concernent le renforcement du focus sur certains éléments interactifs et la hiérarchisation des titres (h1 à transformer en h3).

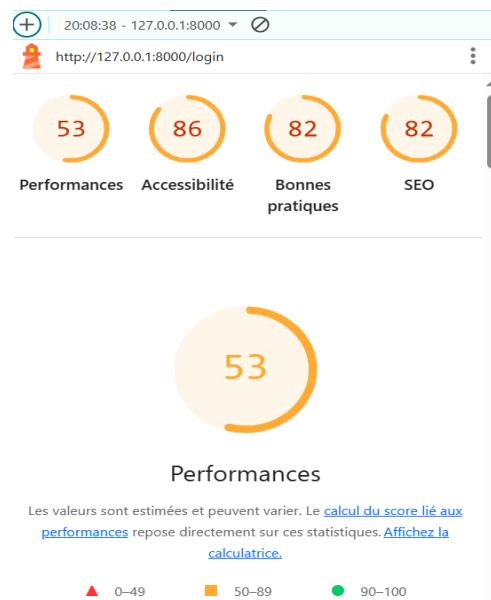
## Bonnes pratiques (79)

Quelques recommandations de **Lighthouse** concernent la mise à jour de dépendances et la sécurisation de liens externes. Ces points seraient corrigés lors du passage en production.

## SEO (92)

Les balises structurantes (title, description, lang) et la hiérarchie du contenu permettent une très bonne indexabilité du site. L'ajout d'un sitemap XML et de balises OpenGraph pourrait encore l'améliorer.

## Score lighthouse connexion/ inscription



## Interprétation et axes d'amélioration page connexion /inscription

## **Performances (53)**

Ce score reste satisfaisant en phase de développement. Les principales améliorations à prévoir :

- exécuter un **build de production** (npm run build) pour minifier les fichiers JS et CSS,
- activer la **mise en cache du navigateur**,
- compresser les ressources statiques via **gzip** sur le serveur (Nginx ou Apache).

Ces optimisations devraient porter le score de performance au-delà de 85 en production.

## **Accessibilité (86)**

Les formulaires de connexion et d'inscription respectent les critères RGAA :

- tous les champs sont associés à un label,
- le focus clavier est visible,
- Les erreurs de validation sont signalées en texte clair.

Une légère amélioration du contraste sur certains boutons renforcerait la lisibilité.

## **Bonnes pratiques (82)**

L'audit recommande :

- de mettre à jour certaines dépendances NPM.
- d'ajouter rel="noopener noreferrer" aux liens externes,
- d'éviter les images trop volumineuses.

## **SEO (82)**

Le SEO est solide pour une page de type « authentification », même s'il est vrai que cette page n'est généralement pas indexée. L'ajout de métadonnées cohérentes et d'un robots.txt adapté renforcerait la cohérence SEO globale du site.

### 3.4. CP4 -Développer une interface utilisateur dynamique

Après la mise en place des interfaces statiques, cette étape a consisté à rendre les pages interactives et connectées aux données réelles de l'application. L'objectif était d'offrir une expérience fluide, intuitive et réactive, en exploitant les capacités combinées de React et Inertia.js. Les composants gèrent leur état local, orchestrent les requêtes asynchrones vers Laravel, affichent des messages d'erreur cohérents entre le front et le back, et respectent les principes de sécurité front-end (validation, CSRF, XSS).

#### a) Navigation dynamique

Grâce à **Inertia.js**, la navigation entre les pages s'effectue **sans rechargeement complet du navigateur**, tout en conservant la logique et la sécurité Laravel côté serveur.

Exemple : un clic sur un maillot dans MaillotsList.jsx déclenche l'appel suivant :

```
42     <Link
43       href={`/maillots/${mailLot.id}`}
44       className="block group"
45       aria-label="Voir les détails du maillot ${mailLot.nom} "
46     >
47       <div className="aspect-square overflow-hidden">
48         <img
49           src={`/ ${mailLot.image}`}
50           alt="Maillot ${mailLot.nom} - ${club.name}"
51           className="w-full h-full object-cover group-hover:scale-105 transition-transform duration-300"
52           loading="lazy"
53         />
54       </div>
55
56       <div className="p-4">
57         <h2 className="font-semibold text-lg text-gray-900 mb-2 group-hover:text-blue-600 transition-colors">
58           ${mailLot.nom}
59         </h2>
60
61         ${mailLot.type} && (
62           <p className="text-sm text-gray-600 mb-2">
63             ${mailLot.type}
64           </p>
```

```

64          </p>
65      )}
66
67      {maillot.prix && (
68          <p className="text-lg font-bold text-green-600">
69              {maillot.prix} €
70          </p>
71      )}
72
73      <div className="mt-3 flex items-center text-sm text-blue-600 group-hover:text-blue-800">
74          <span>Voir les détails</span>
75          <svg
76              className="ml-2 w-4 h-4 group-hover:translate-x-1 transition-transform"
77              fill="none"
78              stroke="currentColor"
79              viewBox="0 0 24 24"
80              aria-hidden="true"
81          >
82              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9 5l7 7-7 7" />
83          </svg>
84      </div>
85  </div>
86  </Link>
87  </div>

```

La route correspondante est définie dans routes/web.php :

```

75
74  // Routes pour les maillots
75  Route::get('/maillots/{id}', [MaillotController::class, 'show'])->name('maillots.show');
76
77

```

Et la méthode show() du MaillotController prépare les données avant l'envoi à React :

```

10  class MaillotController extends Controller
11 {
12     public function show($id)
13     {
14         $maillot = Maillot::with('club')->findOrFail($id);
15
16         // ajouter les tailles, quantités
17         $tailles = ['S', 'M', 'L', 'XL'];
18         $quantite = 1200; // ou récupère depuis la BDD
19
20         return Inertia::render('MaillotDetail', [
21             'maillot' => $maillot,
22             'tailles' => $tailles,
23             'quantite' => $quantite,
24             'nom' => $maillot->nom,
25             'numero' => $maillot->numero,
26             'prix' => 20,
27             'prix_nom' => 3,
28             'prix_numero' => 2,
29         ]);
30     }
31 }

```

Ainsi, le passage entre la liste des produits et la fiche détail s'effectue instantanément, sans rupture de navigation.

## Gestion des formulaires

Les formulaires d'authentification utilisent Inertia useForm : l'état du formulaire, l'envoi et les erreurs sont gérés de manière idiomatique, sans validation manuelle ad-hoc côté front. Côté serveur, Laravel valide les champs ; Inertia remonte ensuite les erreurs automatiquement dans errors, affichées au plus près des inputs.

### État et soumission avec useForm

```
12  ↴  const { data, setData, post, processing, errors } = useForm({
13    login: "",
14    username: "",
15    email: "",
16    password: "",
17    password_confirmation: ""
18  })
19
20 ↴  const handleChange = (e) => {
21   setData(e.target.name, e.target.value)
22 }
23
24 ↴  const handleSubmit = (e) => {
25   e.preventDefault()
26   post(isLogin ? "/login" : "/register")
27 }
28
```

### Affichage des erreurs et désactivation du bouton

```
100 ↴  {isLogin && (
101    <div>
102      <label htmlFor="login" className="block text-sm font-medium text-gray-700">Nom d'utilisateur ou Email</label>
103      <input
104        id="login"
105        name="login"
106        value={data.login}
107        onChange={handleChange}
108        required
109        className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm mt-1"
110      />
111      {errors.login && <p className="text-sm text-red-600">{errors.login}</p>}
112    </div>
113  )}
```

```

114
115      <div>
116          <label htmlFor="password" className="block text-sm font-medium text-gray-700">Mot de passe</label>
117          <div className="relative">
118              <input
119                  id="password"
120                  name="password"
121                  type={showPassword ? "text" : "password"}
122                  value={data.password}
123                  onChange={handleChange}
124                  required
125                  className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm pr-10 mt-1"
126              />
127              <button
128                  type="button"
129                  onClick={() => setShowPassword(!showPassword)}
130                  className="absolute inset-y-0 right-0 px-3 flex items-center"
131                  aria-label={showPassword ? "Masquer le mot de passe" : "Afficher le mot de passe"}
132              >
133                  <EyeIcon show={showPassword} />
134              </button>
135          </div>
136          {errors.password && <p className="text-sm text-red-600">{errors.password}</p>}
137      </div>

```

**Accessibilité et UX intégrées** : contrôle de la visibilité du mot de passe, focus ring, et labels explicites dans le formulaire (voir ci-dessus)

## Bilan

La validation front repose sur useForm (gestion d'état, envoi, processing) et l'affichage automatique des erreurs renvoyées par Laravel (errors). La validation back reste la source de vérité (règles côté contrôleur/FormRequest). Ce couplage garantit une UX fluide (sans rechargement) et une cohérence stricte des règles métier et sécurité.

## b) Interaction avec le panier

L'ajout au panier se fait depuis la page MaillotDetail.jsx. Lorsque l'utilisateur choisit la taille et la quantité, le composant envoie une requête Inertia vers Laravel :

```
21 |     function handleAddToCart() {
22 |       router.post('/cart/add', {
23 |         maillot_id: maillot.id,
24 |         size: taille,
25 |         quantity: qte,
26 |         numero: personnalisation.numero ? numero : null,
27 |         nom: personnalisation.nom ? nom : null,
28 |       }, {
29 |         onSuccess: () => alert("Maillot ajouté au panier !"),
30 |       });
31 |
32 }
```

Cette requête appelle la route définie dans routes/web.php :

```
93 | Route::post('/cart/add', [CartController::class, 'add'])->name('cart.add');
```

Et le contrôleur CartController.php crée ou met à jour l'article dans la base de données :

```
95 ▼ }
96   public function add(Request $request)
97   {
98     $cart = Cart::firstOrCreate(['user_id' => Auth::id()]);
99
100  $item = $cart->items()
101    ->where('maillot_id', $request->maillot_id)
102    ->where('size', $request->size)
103    ->where('numero', $request->numero ?? '')
104    ->where('nom', $request->nom ?? '')
105    ->first();
106
107  if ($item) {
108    $item->increment('quantity', $request->quantity);
109  } else {
110    $cart->items()->create([
111      'maillot_id' => $request->maillot_id,
112      'size' => $request->size,
113      'quantity' => $request->quantity,
114      'numero' => $request->numero,
115      'nom' => $request->nom,
116    ]);
117  }
118
119  return redirect()->route('cart.show')->with('success', 'Article ajouté au panier');
120}
```

L'utilisateur voit immédiatement le panier mis à jour, sans rechargement complet, grâce à **Inertia.js**.

Le composant Panier.jsx s'appuie ensuite sur ces données pour calculer et afficher le total en temps réel :

```
// Total global du panier (somme des totaux lignes)
const prixTotal = cartItems.reduce((sum, item) => sum + (item.total || 0), 0)
```

Et un composant global PanierLink.jsx maintient le compteur dynamique :

```
6  export default function PanierLink() {
7    const [cartCount, setCartCount] = useState(0)
8    const [isLoading, setIsLoading] = useState(false)
9    const { auth } = usePage().props
10
11   // Fonction pour récupérer le nombre d'articles depuis le serveur
12   const fetchCartCount = async () => {
13     try {
14       setIsLoading(true)
15     }
```

### c) Mises à jour côté client

La page MailotDetail.jsx gère dynamiquement l'état des champs (taille, quantité, nom, numéro) et le calcul du total en temps réel. L'objectif que je m'étais fixé est de permettre à l'utilisateur de personnaliser son maillot et d'obtenir instantanément le prix mis à jour, sans rechargement de la page.

Le composant exploite les hooks React useState pour suivre l'état local de chaque champ et recalculer le total en fonction des choix effectués.

## Exemple (MaillotDetail.jsx)

```
6  export default function MaillotDetail({ maillot, tailles, quantite, prix, prix_numero, prix_nom }) {
7    const [taille, setTaille] = useState(tailles[0]);
8    const [qte, setQte] = useState(1);
9    const [numero, setNumero] = useState("");
10   const [nom, setNom] = useState("");
11   const [personnalisation, setPersonnalisation] = useState({ numero: false, nom: false });
12
13   // Calcul du supplément pour personnalisation
14   const supplement =
15     (personnalisation.numero && numero ? prix_numero : 0) +
16     (personnalisation.nom && nom ? prix_nom : 0);
17
18   // Multiplie bien par la quantité choisie
19   const total = (prix + supplement) * parseInt(qte || 1, 10);
20
```

Chaque interaction (ex. changement de taille, de quantité ou activation d'une personnalisation) provoque une **mise à jour immédiate du rendu**. Le composant React réévalue total à chaque modification de l'état, garantissant une expérience fluide et cohérente.

## Interaction utilisateur

Le champ de quantité ou les cases à cocher déclenchent une modification directe de l'état :

```
53 <div className="mb-2">
54   Quantité :
55   <input
56     type="number"
57     min="1"
58     max={quantite}
59     value={qte}
60     onChange={e => setQte(e.target.value)}
61     className="ml-2 border rounded px-2 py-1 w-16"
62   />
63 </div>
64 <div className="mb-2">
65   <label>
66     <input
67       type="checkbox"
68       checked={personnalisation.numero}
69       onChange={e => setPersonnalisation(p => ({ ...p, numero: e.target.checked }))}>
70     </label>
71   </div>
```

Les valeurs sont répercutées immédiatement sur l'affichage du prix total :

```
</div>
<div className="text-xl font-bold mt-4">Total : {total} €</div>
<button
```

Cette logique supprime toute latence entre l'action et le retour visuel, améliorant fortement la perception de réactivité.

## Envoi au back-end

Enfin, le bouton d'ajout déclenche un appel asynchrone vers Laravel, sans rechargement complet :

```
21  function handleAddToCart() {
22    router.post('/cart/add', {
23      maillot_id: maillot.id,
24      size: taille,
25      quantity: qte,
26      numero: personnalisation.numero ? numero : null,
27      nom: personnalisation.nom ? nom : null,
28    },
29    onSuccess: () => alert("Maillot ajouté au panier !"),
30  });
31}
```

Cette approche connecte directement la logique front à Laravel via Inertia, garantissant la persistance des données et la cohérence du panier tout en maintenant un comportement de SPA fluide.

## d) Sécurité front-end

L'UI (interface utilisateur) est pensée pour **empêcher l'injection** côté navigateur et **garantir** que toutes les règles de sécurité sont ré-appliquées côté serveur. Les points clés :

### 1) Anti-XSS par design (React)

Le rendu des champs saisis par l'utilisateur (**nom**, **numéro**, etc.) est **échappé par défaut** par React.

Aucun dangerouslySetInnerHTML n'est utilisé.

Exemple : fiche produit- seules des valeurs texte sont insérées et renvoyées au serveur via Inertia (pas d'HTML injecté).

**Extrait :** envoi des valeurs « brutes »

```
21  < function handleAddToCart() {
22    router.post('/cart/add', {
23      maillot_id: maillot.id,
24      size: taille,
25      quantity: qte,
26      numero: personnalisation.numero ? numero : null,
27      nom: personnalisation.nom ? nom : null,
28    }, {
29      onSuccess: () => alert("Maillot ajouté au panier !"),
30    });
31  }
```

## 2) CSRF / intégration Inertia

Les **formulaires** envoyés par Inertia (router.post) bénéficient de la **protection CSRF de Laravel** (token généré par le middleware et le layout).

Aucun jeton n'est manipulé côté front, cela induit une **réduction du risque d'erreur**.

**Fichier côté front :** resources/js/Pages/LoginRegister.jsx

```
12  const { data, setData, post, processing, errors } = useForm({
13    login: "",
14    username: "",
15    Click to add a breakpoint
16    password: "",
17    password_confirmation: "",
18  })
19
20  const handleChange = (e) => {
21    setData(e.target.name, e.target.value)
22  }
23
24  const handleSubmit = (e) => {
25    e.preventDefault()
26    post(isLogin ? "/login" : "/register")
27  }
28
```

### 3) Validation serveur systématique

Chaque entrée utilisateur est **validée côté Laravel**.

Trois exemples :

#### A. Panier - mise à jour d'un article (taille, quantité, nom, numéro)

Fichier : app/Http/Controllers/CartController.php, méthode update()

```
126     $data = $request->validate([
127         'size'      => 'required|string|max:10',
128         'quantity'  => 'required|integer|min:1',
129         'nom'        => 'nullable|string|max:50',
130         'numero'    => 'nullable|string|max:3',
131     ]);
132
133     $item->update($data);
```

Cela empêche taille/quantité invalides et limite la longueur de saisie.

#### B. Adresse - création / modification

Fichier : app/Http/Controllers/AddressController.php, méthode store() :

```
39  $validated = $request->validate([
40      'type'      => 'required|in:billing,shipping',
41      'first_name' => 'required|string|max:100',
42      'last_name'  => 'required|string|max:100',
43      'street'     => 'required|string',
44      'city'       => 'required|string|max:100',
45      'postal_code'=> 'required|string|max:20',
46      'country'   => 'required|string|max:2',
47      'phone'     => 'nullable|string|max:20',
48      'is_default'=> 'boolean',
49  ]);
50
```

Types stricts / longueurs / ensembles de valeurs autorisées.

## C. Compte -mise à jour du mot de passe

Fichier : app\Http\Controllers\AccountDetailController.php

```
75     $request->validate([
76         'current_password' => 'required',
77         'password' => 'required|confirmed|min:8',
78     ]);
79
80     if (!Hash::check($request->current_password, $user->password)) {
81         return back()->withErrors(['current_password' => 'Le mot de passe actuel est incorrect.']);
82     }
83
84     $user->password = Hash::make($request->password);
85 }
```

Évite le changement non autorisé et hachage obligatoire du nouveau mot de passe.

## 4) Autorisations / cloisonnement des données (403 si hors périmètre)

Avant modification d'un item de panier, on vérifie la propriété de l'élément.

Si l'objet n'est pas reconnu, on obtient un code **403** immédiat.

Fichier : app\Http\Controllers\CartController.php, méthode update() :

```
123 {
124     if ($item->cart->user_id !== Auth::id()) abort(403);
125 }
```

C'est une barrière simple et efficace pour empêcher l'accès horizontal (IDOR).

## 5) Réduction de surface d'exposition (données envoyées côté UI)

Dans la vue panier, on façonne les données renvoyées à React (mapping) pour exposer uniquement le nécessaire et recalculer le total côté serveur (source de vérité).

Fichier : app\Http\Controllers\CartController.php, méthode show() (mapping + recalculs) :

```

59     return inertia('Panier', [
60         'cartItems' => $cart->items->map(function($item) {
61             $maillot = $item->maillot;
62             $price = $maillot ? $maillot->price : 0;
63             $image = $maillot ? $maillot->image : null;
64             $name = $maillot ? $maillot->name : '????';
65
66             // Suppléments personnalisation
67             $suppNom = $item->nom ? 3 : 0;
68             $suppNumero = $item->numero ? 2 : 0;
69             $supplement = $suppNom + $suppNumero;
70
71             // Total ligne
72             $total = ($price + $supplement) * $item->quantity;
73
74             return [
75                 'id' => $item->id,
76                 'club_name' => $maillot->club->name ?? 'Club inconnu',
77                 'maillot_name' => $maillot->nom ?? $maillot->name ?? 'Maillot',
78                 'maillot_id' => $item->maillot_id,
79                 'name' => $name,
80                 'image' => $image,
81                 'size' => $item->size,
82                 'quantity' => $item->quantity,
83                 'price' => $price,
84                 'nom' => $item->nom,
85                 'numero' => $item->numero,
86                 'supplement' => $supplement,
87                 'total' => $total,
88             ];

```

Règle métier (surcoûts nom/numéro) appliquée côté back en cohérence cohérence avec l'UI (interface utilisateur).

## 6) Authentification & hachage des mots de passe

À l'inscription, les mots de passe sont **hachés** (Hash::make) et les e-mails **validés / uniques**.

**Fichier :** app/Http/Controllers/AuthController.php, méthode register() :

```

33     public function register(Request $request)
34     {
35         $validated = $request->validate([
36             'username'      => 'required|string|max:50|unique:users',
37             'email'        => 'required|email|unique:users',
38             'password'      => 'required|string|min:8|confirmed',
39             'first_name'    => 'nullable|string|max:100',
40             'last_name'     => 'nullable|string|max:100',
41             'phone'         => 'nullable|string|max:20',
42             'birth_date'    => 'nullable|date',
43             'gender'        => 'nullable|in:male,female,other',
44         ]);
45
46         $user = User::create([
47             'username'      => $validated['username'],
48             'email'        => $validated['email'],
49             'password'      => Hash::make($validated['password']),
50             'first_name'    => $validated['first_name'] ?? null,
51             'last_name'     => $validated['last_name'] ?? null,
52             'phone'         => $validated['phone'] ?? null,
53             'birth_date'    => $validated['birth_date'] ?? null,
54             'gender'        => $validated['gender'] ?? null,
55             'is_active'     => true, // Valeur par défaut ajoutée ici
56         ]);
57

```

## Bilan

L'interface React limite l'injection par échappement automatique, et n'insère pas d'HTML arbitraire. Les formulaires passent par Inertia (CSRF natif Laravel). Toutes les règles (tailles, quantités, formats, longueurs) sont revalidées côté serveur avec contrôle d'accès (403) et hachage des mots de passe. Le panier n'expose au client que les champs nécessaires, et les totaux sont recalculés côté back pour conserver la source de vérité.

## 3.5. Accessibilité et éco-conception

L'accessibilité a été prise en compte dès la conception de l'interface afin de garantir une expérience utilisateur inclusive, conformément aux recommandations du **RGAA (Référentiel Général d'Amélioration de l'Accessibilité)**.

### a) Principes appliqués

Les principaux critères respectés sont :

#### - Navigation clavier :

Toutes les pages et formulaires sont accessibles à la navigation au clavier grâce aux attributs HTML natifs et à la gestion du focus dans les composants React (tabIndex, focus:ring de TailwindCSS).

:

#### - Contrastes :

Les couleurs ont été choisies pour offrir un contraste suffisant entre le texte et l'arrière-plan (minimum 4.5:1 pour le texte normal). J'ai effectué les vérifications avec l'outil intégré Lighthouse et le testeur RGAA.

### - Structures sémantiques :

Les balises <header>, <main>, <section>, <footer> et <nav> ont été utilisées pour structurer les pages, permettant aux technologies d'assistance (lecteurs d'écran) de mieux comprendre la hiérarchie des contenus.

**resources/js/Layouts/MainLayout.jsx** utilisation de <main> (avec header/footer)

```
 8   <Header />
 9   <main className="flex-1">
10     {children}
11   </main>
12   <Footer />
```

**resources/js/Pages/LoginRegister.jsx** <main> + attributs ARIA pour une zone d'onglets

```
46   <main className="min-h-screen flex items-center justify-center bg-gradient-to-r from-purple-200 to-blue-200 py-12 px-4 sm:px-6 lg:px-8">
47     <div className="max-w-md w-full space-y-8">
48       /* Tabs */
49       <div className="flex justify-center mb-6" role="tablist" aria-label="Navigation formulaire">
50         <button
51           onClick={() => setIsLogin(true)}
52           className={`px-4 py-2 font-medium text-lg ${isLogin ? "text-blue-600 border-b-2 border-blue-600" : "text-gray-500"}`}
53           role="tab"
54           aria-selected={isLogin}
55         >
56           Connexion
57         </button>
58         <button
59           onClick={() => setIsLogin(false)}
60           className={`px-4 py-2 font-medium text-lg ${!isLogin ? "text-blue-600 border-b-2 border-blue-600" : "text-gray-500"}`}
61           role="tab"
62           aria-selected={!isLogin}
63         >
64           Inscription
65         </button>
66       </div>
```

### - Alternatives textuelles :

Les images des maillots disposent toutes d'un attribut alt descriptif.

## Carte d'équipe (Home) – TeamCard dans resources/js/Pages/Home.jsx

```
252 <div className="relative rounded-lg overflow-hidden hover:transform hover:-translate-y-1">
253   <img src={image} alt={team} className="w-full h-64 object-cover" />
254 </div>
```

Pour les cartes “nouveaux maillots” :

```
158 <img
159   src={maillot.image}
160   alt={`Maillot ${maillot.team}`}
161   className="w-full h-64 object-contain rounded-t-lg"
162 />
```

## - Labels et formulaires :

Chaque champ de formulaire est associé à un <label> ou un attribut aria-labelledby. Les messages d’erreurs sont visibles et compréhensibles, tant visuellement qu’à la lecture vocale.

## resources/js/Pages/LoginRegister.jsx -label lié par htmlFor + messages d’erreur

```
85 <label htmlFor="email" className="block text-sm font-medium text-gray-700">Email</label>
86 <input
87   id="email"
88   name="email"
89   type="email"
90   value={data.email}
91   onChange={handleChange}
92   required
93   className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm mt-1"
94   />
95   {errors.email && <p className="text-sm text-red-600">{errors.email}</p>}
96 </div>
97 </>
```

## Navigation par onglets accessible (même fichier)

```
49   <div className="flex justify-center mb-6" role="tablist" aria-label="Navigation formulaire">
50     <button
51       onClick={() => setIsLogin(true)}
52       className={`px-4 py-2 font-medium text-lg ${isLogin ? "text-blue-600 border-b-2 border-blue-600"
53         : ""} role="tab"
54         aria-selected={isLogin}>
55       Connexion
56     </button>
57     <button
58       onClick={() => setIsLogin(false)}
59       className={`px-4 py-2 font-medium text-lg ${!isLogin ? "text-blue-600 border-b-2 border-blue-600"
60         : ""} role="tab"
61         aria-selected={!isLogin}>
62       Inscription
63     </button>
64   </div>
```

## b) Lightscore page login ( connexion/inscription )

The screenshot shows the Lightscore accessibility audit interface. At the top, there's a header with a plus sign, the date and time (20:08:38 - 127.0.0.1:8000), a refresh icon, and a URL field containing 'http://127.0.0.1:8000/login'. Below the header, there are four orange circular progress indicators with the numbers 53, 86, 82, and 82.

The main content area has a title 'AUDITS RÉUSSIS (22)' and a 'Afficher' button. Below this, there's a large orange circle with the number '86' in red, followed by the heading 'Accessibilité'. A descriptive text follows: 'Ces vérifications permettent de connaître les possibilités d'[amélioration de l'accessibilité de votre application Web](#). La détection automatique ne peut détecter qu'une partie des problèmes et ne garantit pas l'accessibilité de votre application Web. Il est donc conseillé d'effectuer également un [test manuel](#)'.

Under the heading 'NOMS ET ÉTIQUETTES', there are two expandable sections:

- ▲ Les boutons n'ont pas de nom accessible
- ▲ Le document ne contient pas d'élément `<title>`

At the bottom, a note reads: 'Servez-vous de ces indications pour améliorer la sémantique des éléments de contrôle de votre application. Vous optimiserez ainsi l'expérience des utilisateurs de technologies d'assistance, comme les'.

## Bilan

Le score **Lighthouse Accessibilité : 86/100** traduit une bonne conformité générale aux critères RGAA.

Quelques points d'amélioration subsistent, notamment :

- homogénéiser les contrastes dans certaines zones secondaires,
- renforcer les indications de focus sur les liens de navigation.

Ces ajustements pourront être intégrés lors de la phase d'optimisation visuelle.

## c) Compétences mobilisées :

J'ai appris à réaliser une interface utilisateur web statique et adaptable (prise en compte des normes d'accessibilité, design responsive).

J'ai pu Développé une interface utilisateur web dynamique (intégration d'attributs ARIA, gestion de l'interactivité accessible).

## d) Éco-conception

**Optimisation des médias** : conversion des images en formats légers (WebP) et compression systématique pour limiter le poids des pages.

**Code CSS optimisé** : grâce à **TailwindCSS**, les classes inutilisées sont supprimées lors de la phase de build (purgeCSS), ce qui réduit la taille du code distribué.

**Chargement progressif des composants** : avec Inertia.js, seuls les composants nécessaires sont chargés (lazy loading), ce qui limite la consommation de bande passante et améliore les performances côté client. Mon site de e-commerce profite du lazy chargement pour les images et pour les relations entre entités, ce qui améliore la rapidité et la réactivité.

## Compétences mobilisées :

Développer des composants d'accès aux données (gestion optimisée des ressources et des performances).

- Préparer et exécuter le déploiement d'une application web (intégration d'optimisations d'éco-conception avant la mise en production).

### **3.6. Bilan du front-end**

La phase front-end du projet a permis de mettre en œuvre l'ensemble des compétences professionnelles CP1 à CP4 : installation et configuration de l'environnement de travail, maquettage des interfaces, réalisation statique puis dynamisation des pages utilisateur.

Ce travail s'est traduit par :

- la création d'une interface moderne et responsive, respectant les bonnes pratiques de **conception web** et d'**ergonomie** ;
- l'intégration de fonctionnalités dynamiques (navigation fluide via Inertia.js, gestion des formulaires, mise à jour du panier en temps réel).
- l'application des recommandations en matière d'**accessibilité (RGAA :Référentiel Général d'Amélioration de l'Accessibilité )** et d'**éco-conception** (optimisation des images, réduction du code superflu, chargement progressif des composants).
- la mise en place de mécanismes garantissant la **sécurité** côté client (validation des entrées, protection CSRF via Laravel).

En résumé, le front-end est désormais en capacité d'interagir efficacement avec la partie back-end. Il offre aux utilisateurs finaux une expérience fluide, intuitive et sécurisée, conforme aux standards attendus dans une boutique en ligne moderne.

## **Chapitre 4 -Développement Back-end**

Après avoir conçu et dynamisé l'interface utilisateur côté front-end, l'étape suivante a consisté à mettre en place la partie back-end de l'application. Cette couche a pour rôle de gérer la logique métier, l'accès aux données et la sécurité globale du système. Elle repose sur le framework **Laravel**, qui fournit un environnement robuste pour interagir avec la base de données, traiter les requêtes des utilisateurs et garantir l'intégrité des informations.

L'objectif de ce chapitre est de présenter la mise en œuvre progressive des compétences professionnelles **CP5 à CP8**, allant de la création de la base de données relationnelle jusqu'à la documentation du déploiement de l'application web.

## 4.1. CP5 -Mettre en place une base de données relationnelle

Afin de permettre la gestion cohérente et sécurisée des données de l'application **Fou-2-Foot**, une base de données relationnelle a été conçue puis déployée. Cette étape visait à formaliser le modèle conceptuel, à le traduire en schéma relationnel et à l'implémenter dans le projet au moyen des outils intégrés de Laravel.

### a) la méthode Merise

Avant la mise en œuvre technique de la base de données, j'ai choisi d'appliquer la méthode Merise.

Merise est une méthode informatique dédiée à la modélisation qui analyse la structure à informatiser en terme de systèmes. L'indéniable avantage de cette méthode est de pouvoir cadrer le projet informatique. Créée dans les années 1970 sur commande de l'État français et destinée aux gros projets informatiques de l'époque, la méthode a perduré jusqu'à aujourd'hui. Son utilisation reste très répandue et constitue un socle difficilement contournable lorsque l'on entame la création de bases de données.

Merise est donc en fait un outil analytique qui facilite la création de base de données et de projets informatique.

Merise s'appuie sur trois niveaux de représentation :

- **Le MCD (Modèle Conceptuel de Données)** : représentation graphique des entités, associations et cardinalités, sans considération technique.

Exemple : un *utilisateur* possède une ou plusieurs *adresses*, un *club* regroupe plusieurs *maillots*.

- **Le MLD (Modèle Logique de Données)** : traduction du MCD en tables et relations logiques adaptées à un SGBD relationnel.

Exemple : les entités deviennent des tables (users, clubs, maillots) reliées par des clés étrangères.

- **Le MPD (Modèle Physique de Données)** : version finale du modèle, avec les types de champs, contraintes, index et clés primaires/étrangères utilisés dans MySQL.

### Application de la méthode Merise

La modélisation Merise a permis de structurer les différentes entités nécessaires au fonctionnement du site e-commerce :

- **Users** : représente les comptes utilisateurs (authentification, rôle, etc.) ;

- **UserAddress** : gère les adresses de facturation et de livraison (relation 1-n avec users) ;

- **UserSession** : enregistre les connexions actives et renforce la sécurité des sessions ;
- **Clubs** : regroupe les clubs de football (nom, logo, pays) ;
- **Maillots** : référence les produits vendus, avec leurs caractéristiques (taille, prix, image, club associé) ;
- **Carts** et **CartItems** : assurent la gestion du panier d'achat de chaque utilisateur ;
- **Orders** : table prévue pour les commandes et paiements (extension future du projet).

## Implémentation dans Laravel

La base a été implémentée via :

- des **migrations Laravel**, assurant la création et la version des tables ;
- des **modèles Eloquent**, pour gérer les relations (hasMany, belongsTo,hasOne) ;
- des **seeders**, pour insérer des données de test (clubs, maillots, utilisateurs) et valider les fonctionnalités.

Exemple de migration simplifiée :

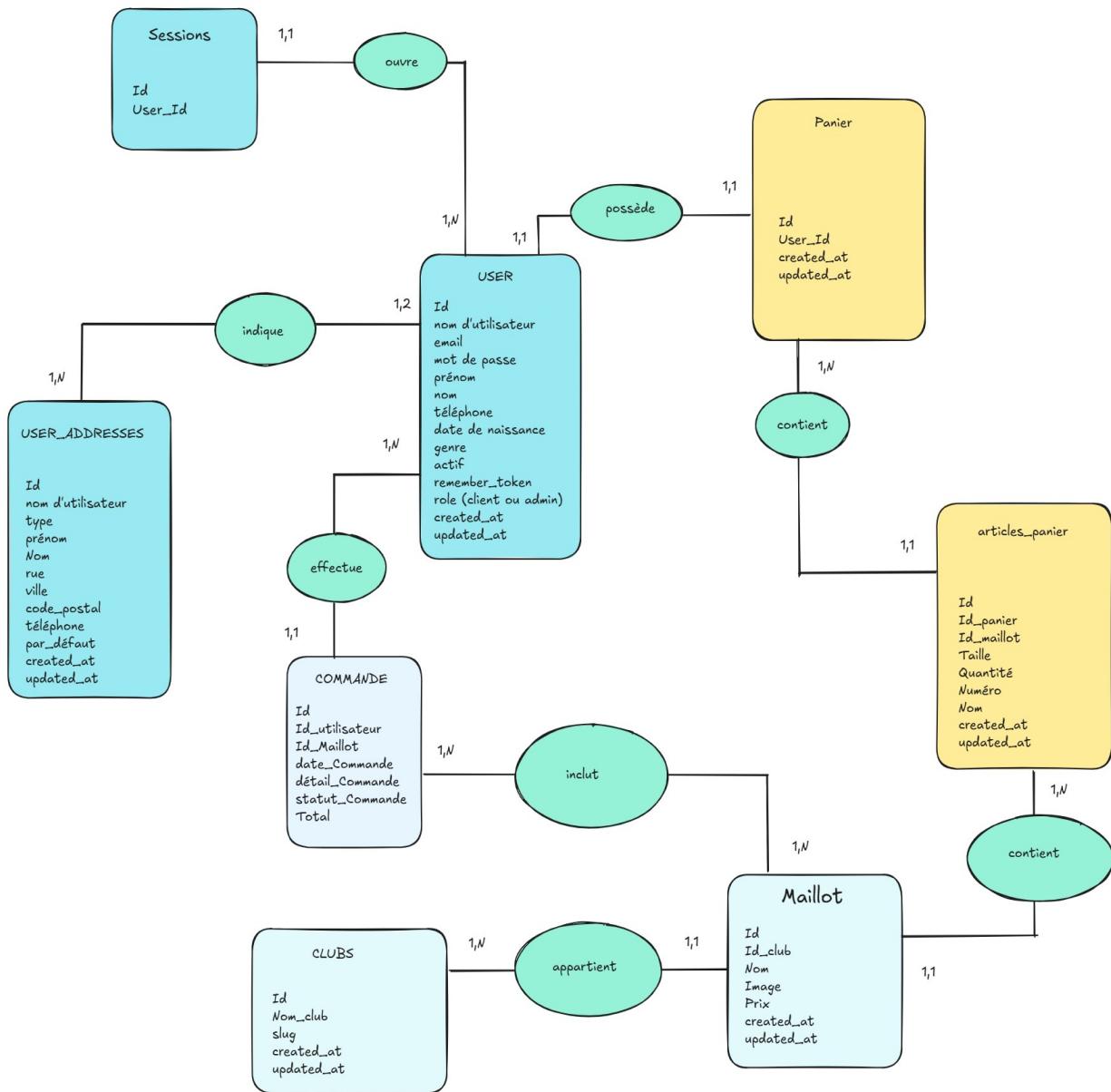
fichier :\www\Lam\_Maillot\_de\_foot\database\migrations  
2025\_07\_10\_112614\_create\_maillots\_table.php

```

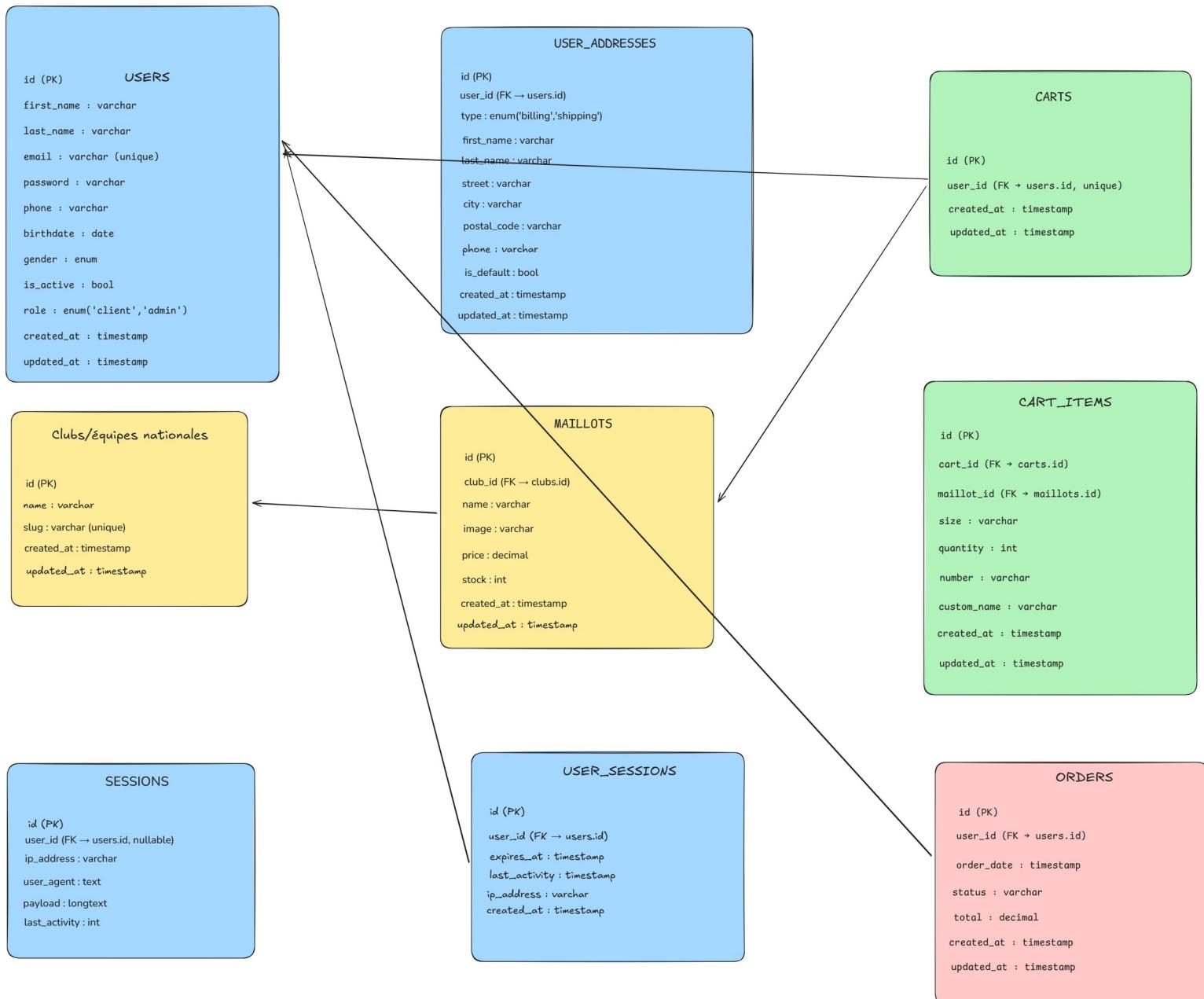
12  public function up(): void
13  {
14      Schema::create('maillots', function (Blueprint $table) {
15          $table->id();
16          $table->foreignId('club_id')->constrained()->onDelete('cascade');
17          $table->string('nom');
18          $table->string('image'); // chemin ou URL de l'image
19          $table->timestamps();
20      });

```

# MCD (Modèle conceptuel de données)



# MLD (Modèle Logique de Données)



\* J'ai utilisé Excalidraw plutôt que db diagram pour réaliser mon MCD et mon MLD parce que le rendu de db diagram était moins lisible.

## Structure du MPD (Modèle Physique de Données)

### Utilisateur (users)

- **Id** : bigint UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT

- **username** : varchar(50), UNIQUE
- **email** : varchar(191), UNIQUE
- **password** : varchar(191)
- **first\_name** : varchar(100) (nullable)
- **last\_name** : varchar(100) (nullable)
- **phone** : varchar(20) (nullable)
- **birth\_date** : date (nullable)
- **gender** : enum('male','female','other') (nullable)
- **email\_verified\_at** : timestamp (nullable)
- **is\_active** BOOLEAN DEFAULT TRUE,
- **role** ENUM('client','admin') NOT NULL DEFAULT 'client',
- **remember\_token** : varchar(100) (nullable)
- **created\_at/updated\_at** : timestamp (nullable)
- **Indexes** : email, username, is\_active

### **Adresse utilisateur (user\_addresses)**

- **id** : bigint UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT
- **user\_id** : bigint UNSIGNED, FOREIGN KEY (users.id)
- **type** : enum('billing','shipping')
- **first\_name/last\_name** : varchar(100)
- **street** : text
- **city** : varchar(100)
- **postal\_code** : varchar(20)
- **country** : varchar(2) DEFAULT 'FR'
- **phone** : varchar(20) (nullable)
- **is\_default** : tinyint(1) DEFAULT 0

- **created\_at/updated\_at** : timestamp (nullable)
- **Indexes** : user\_id, type, is\_default

## Club (clubs)

- **id** : bigint UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT
- **name** : varchar(191)
- **slug** : varchar(191), UNIQUE
- **created\_at/updated\_at** : timestamp (nullable)

## Maillot (maillots)

- **id** : bigint UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT
- **club\_id** : bigint UNSIGNED, FOREIGN KEY (clubs.id)
- **nom** : varchar(191)
- **image** : varchar(191)
- **price** : decimal(8,2) DEFAULT 0.00
- **created\_at/updated\_at** : timestamp (nullable)
- **Indexes** : club\_id

## Panier (carts)

- **id** : bigint UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT
- **user\_id** : bigint UNSIGNED, UNIQUE, FOREIGN KEY (users.id)
- **created\_at/updated\_at** : timestamp (nullable)

## Article du panier (cart\_items)

- **id** : bigint UNSIGNED, PRIMARY KEY, AUTO\_INCREMENT

- **cart\_id** : bigint UNSIGNED, FOREIGN KEY (carts.id)
- **maillot\_id** : bigint UNSIGNED, FOREIGN KEY (maillots.id)
- **size** : varchar(191) (nullable)
- **quantity** : int DEFAULT 1
- **numero** : varchar(191) (nullable)
- **nom** : varchar(191) (nullable)
- **created\_at/updated\_at** : timestamp (nullable)
- **Indexes** : cart\_id, maillot\_id

## Sessions (sessions)

- **id** : varchar(191), PRIMARY KEY
- **user\_id** : bigint UNSIGNED, FOREIGN KEY (users.id)
- **ip\_address** : varchar(45) (nullable)
- **user\_agent** : text (nullable)
- **payload** : longtext
- **last\_activity** : int
- **Indexes** : user\_id, last\_activity

## Relations et contraintes

Origine	Cible	Foreign Key	Cardinalité
user_addresses	users	user_id → users.id	n,1
carts	users	user_id → users.id	n,1
cart_items	carts	cart_id → carts.id	n,1
cart_items	maillots	maillot_id → maillots.id	n,1
maillots	clubs	club_id → clubs.id	n,1
sessions	users	user_id → users.id	n,1

Le MPD ci-dessus correspond à la structure physique des tables SQL : types de champs, clés primaires, étrangères et index sont explicitement listés pour une implémentation optimale en base MySQL.

### b) Outils et technologies utilisés

- **MySQL** comme système de gestion de base de données relationnelle, offrant robustesse et compatibilité avec Laravel.
- **Eloquent ORM** pour manipuler les données sous forme de modèles objets et faciliter les relations entre entités.
- **Migrations Laravel** pour créer, versionner et maintenir la structure de la base de données de manière automatisée.
- **Seeders** pour insérer des données de test (utilisateurs, clubs, maillots) et accélérer les phases de développement et de validation.

### c) Conception et organisation des données

Le modèle conceptuel a été décliné en tables relationnelles dans le respect des principes de normalisation (éviter les redondances, garantir la cohérence) :

- **users** : stocke les informations de compte et d'authentification.
- **user\_addresses** : gère les adresses de facturation et de livraison. J'ai jugé essentiel de permettre à un utilisateur de définir deux adresses distinctes, car ce type de produit peut être acheté pour soi ou offert en cadeau.
- **clubs** : référence les clubs de football afin d'associer chaque maillot à son équipe.
- **maillots** : contient le catalogue de produits (nom, image, prix, club associé).
- **carts** et **cart\_items** : permettent la gestion du panier (articles sélectionnés, quantités, options de personnalisation).
- **orders** (*prévu en extension*) : table destinée au suivi des commandes et des paiements.

- **UserSession** : enregistre les connexions actives et renforce la sécurité des sessions ;

## Base de données

Table	Action	Lignes	Type	Interclassement	Taille	Perte
carts	Parcourir Structure Rechercher Insérer Vider Supprimer	18	MyISAM	utf8mb4_unicode_ci	3,4 kio	-
cart_items	Parcourir Structure Rechercher Insérer Vider Supprimer	59	MyISAM	utf8mb4_unicode_ci	14,6 kio	560 c
clubs	Parcourir Structure Rechercher Insérer Vider Supprimer	87	MyISAM	utf8mb4_unicode_ci	14,8 kio	-
maillots	Parcourir Structure Rechercher Insérer Vider Supprimer	214	MyISAM	utf8mb4_unicode_ci	29,5 kio	-
migrations	Parcourir Structure Rechercher Insérer Vider Supprimer	12	MyISAM	utf8mb4_unicode_ci	2,7 kio	-
sessions	Parcourir Structure Rechercher Insérer Vider Supprimer	2	MyISAM	utf8mb4_unicode_ci	21,8 kio	10,0 kic
users	Parcourir Structure Rechercher Insérer Vider Supprimer	18	MyISAM	utf8mb4_unicode_ci	18,5 kio	20 c
user_addresses	Parcourir Structure Rechercher Insérer Vider Supprimer	21	MyISAM	utf8mb4_unicode_ci	7,1 kio	-

8 tables Somme 431 MyISAM utf8mb4\_general\_ci 111,5 10,5 kio kic

## Table users

	id	username	email	password	first_name	last_name
<input type="checkbox"/>	1	Albundy	albundy@gmail.com	\$2y\$12\$UvN5QMkeIWZ9jKhn5codO.qKMfFCFvNc8KwPsMjnRZ...	AL	BU
<input type="checkbox"/>	2	prasanna	prasann@yahoo.in	\$2y\$12\$O2AMY26P67ahq57agjSkXeWHCfnoEdhRbse3QsN.w...	Prasanna	Lakshmi
<input type="checkbox"/>	3	Lolo	lolo@gmail.com	\$2y\$12\$BKwgMX9dDzQOKZH1nxw.uq24jq8BfQggGMKFnivcx...	Lolo	Rig
<input type="checkbox"/>	4	Sophie69	sophie@gmail.com	\$2y\$12\$U6nJRH09y.TiXoE3rdgeD6e6.M400Mm5GDogmpkQw...	NULL	NU
<input type="checkbox"/>	5	marion	marion@hotmail.com	\$2y\$12\$desYkbbsYL9Ml/1r0YB6eu97Yzd7.LdCmhXj69F4W...	Marion	LAf
<input type="checkbox"/>	6	magnum	magnum@yahoo.us	\$2y\$12\$JGfaGqsS.cuglCQviXDLfOraM.waSiCidCnMP6.xsyX...	thomas	Ma
<input type="checkbox"/>	7	rais	rais@yahoo.fr	\$2y\$12\$S5cQ9VHc7dG.A0maBwL..WaUcnDCIou/qFTm0qnZ5...	Rais	Pa

## Structure table users

The screenshot shows the phpMyAdmin interface for the 'users' table in the 'maillot' database. The table has 14 columns:

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	<code>id</code>	bigint		UNSIGNED	Non	Aucun(e)		AUTO_INCREMENT	<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
2	<code>username</code>	varchar(50)	utf8mb4_unicode_ci		Non	Aucun(e)			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
3	<code>email</code>	varchar(191)	utf8mb4_unicode_ci		Non	Aucun(e)			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
4	<code>password</code>	varchar(191)	utf8mb4_unicode_ci		Non	Aucun(e)			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
5	<code>first_name</code>	varchar(100)	utf8mb4_unicode_ci		Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
6	<code>last_name</code>	varchar(100)	utf8mb4_unicode_ci		Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
7	<code>phone</code>	varchar(20)	utf8mb4_unicode_ci		Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
8	<code>birth_date</code>	date			Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
9	<code>gender</code>	enum('male','female','other')	utf8mb4_unicode_ci		Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
10	<code>email_verified_at</code>	timestamp			Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
11	<code>is_active</code>	tinyint(1)			Non	1			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
12	<code>remember_token</code>	varchar(100)	utf8mb4_unicode_ci		Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
13	<code>created_at</code>	timestamp			Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>
14	<code>updated_at</code>	timestamp			Oui	NULL			<a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Plus</a>

Etant encore en phase évolutive, il est vraisemblable que je modifierai à terme les valeurs des types pour que celles-ci soient plus restrictives.

## CartItem

The screenshot shows the phpMyAdmin interface for the 'cart\_items' table in the 'maillot' database. The table has 11 columns:

	<code>id</code>	<code>cart_id</code>	<code>maillot_id</code>	<code>size</code>	<code>quantity</code>	<code>created_at</code>	<code>updated_at</code>	<code>numero</code>	<code>nom</code>
	138	1	188	XL	3	2025-07-22 11:59:49	2025-07-25 11:17:50	33	ZORRO
	114	2	185	S	1	2025-07-20 02:22:00	2025-09-27 00:46:56	NULL	NULL
	7	4	177	XL	2	2025-07-15 07:59:45	2025-07-25 09:19:18	NULL	LULOL
	8	4	188	XL	3	2025-07-15 08:35:53	2025-07-25 09:19:29	NULL	LUBLEU
	140	17	2	XL	3	2025-07-22 13:33:39	2025-07-22 13:55:37	31	ULYSSE
	153	3	389	XL	3	2025-07-26 00:32:44	2025-07-26 01:19:55	88	AL BUNDY
	154	3	1	XL	1	2025-07-26 00:42:28	2025-07-26 00:42:28	97	BUNDY FRANCE
	19	3	188	XL	3	2025-07-15 09:40:00	2025-07-26 01:20:06	0	AL FRANCE
	20	3	36	XL	1	2025-07-15 09:41:37	2025-07-26 01:20:47	99	AL OL
	129	7	36	S	3	2025-07-21 12:05:35	2025-07-21 13:26:54	11	Marion OL
	146	1	371	XL	2	2025-07-25 10:09:09	2025-07-25 10:09:09	00	Zorro le renard

## Maillots

The screenshot shows the phpMyAdmin interface for the 'maillot' database. The left sidebar lists tables such as 'fou2foot', 'maillot', 'cart\_items', 'clubs', 'maillots', 'migrations', 'sessions', 'users', and 'user\_addresses'. The 'maillots' table is selected, displaying 214 rows. The table structure includes columns: id, club\_id, nom, image, created\_at, updated\_at, and price. The data shows various items like 'Domicile 2024', 'Extérieur 2024', and 'Bresil 2025', each with a corresponding image path and timestamp.

	id	club_id	nom	image	created_at	updated_at	price
<input type="checkbox"/>	1	1	Domicile 2024	images/maillot/images_maillot/france.jpg	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
<input type="checkbox"/>	2	1	Extérieur 2024	images/maillot/images_maillot/france_ext.jfif	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
<input type="checkbox"/>	3	2	Domicile 2024	images/maillot/images_maillot/bresil.jfif	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
<input type="checkbox"/>	4	2	Extérieur 2025	images/maillot/images_maillot/bresil26_exterieur.w...	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
<input type="checkbox"/>	5	3	Domicile 2024	images/maillot/images_maillot/espagne_24_domicile....	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
<input type="checkbox"/>	6	3	Extérieur 2024	images/maillot/images_maillot/espagne_24_exterieur....	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
<input type="checkbox"/>	7	4	Domicile 2024	images/maillot/images_maillot/pays_bas_24_domicile...	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00

## User\_Addresses

The screenshot shows the phpMyAdmin interface for the 'user\_addresses' table. The left sidebar lists the same tables as the previous screenshot. The 'user\_addresses' table is selected, displaying 21 rows. The table structure includes columns: id, user\_id, type, first\_name, last\_name, street, city, postal\_code, country, phone, and is\_c. The data shows various addresses for users, including 'AL BUNDY' at 'macadam av 93 CHICAGO BULLS' and 'Rig' at 'chemin le soleil levant Vernaison'.

	id	user_id	type	first_name	last_name	street	city	postal_code	country	phone	is_c
<input type="checkbox"/>	1	1	billing	AL	BUNDY	macadam av 93	CHICAGO BULLS	45GOR99	FR	22 84 89 21 69	
<input type="checkbox"/>	2	2	billing	Prasanna	Lakshmi	Darbla	Champs Elysées	75000	FR	07 84 55 43 88	
<input type="checkbox"/>	3	3	billing	Lolo	Rig	chemin le soleil levant	Vernaison	69390	FR	04 78 46 05 93	
<input type="checkbox"/>	4	6	billing	thomas	Magnum	Robin master street	Hawaï 53	8234	FR	1234567890	
<input type="checkbox"/>	5	7	billing	Rais	Paqueta	781 BD de la croisette	Vence	06400	FR	34 54 64 74 94	
<input type="checkbox"/>	6	8	billing	Diego	De la Vega	65 ROUTE DE la soie	MONTEREY California	56734	FR	88 14 29 52 44	
<input type="checkbox"/>	8	9	billing	alicia	chabane	33 BD de la mourrachone	Grasse	06355	FR	07 25 50 75 11	

## Mise en œuvre technique

Les **migrations** ont permis de générer automatiquement les tables et leurs contraintes (clés primaires (PK), clés étrangères(FK), index).

Les relations entre modèles ont été définies via Eloquent, par exemple :

- User → UserAddress (relation *one-to-many*),
- Club → Maillot (relation *one-to-many*),
- Cart → CartItem (relation *one-to-many*).

Les **seeders** ont permis d'initialiser des données de démonstration, facilitant les tests fonctionnels des pages de liste (MaillotsList.jsx) et de détail (MaillotDetail.jsx).

## Résultat

La mise en place de la base de données relationnelle offre plusieurs avantages concrets :

une gestion structurée et centralisée des informations,

l'assurance de l'intégrité référentielle (par exemple, un maillot appartient toujours à un club existant),

la traçabilité et la cohérence des données utilisateurs (par exemple, chaque panier est lié à un compte),

un socle fiable et évolutif pour les étapes suivantes : développement des composants d'accès aux données (CP6) et mise en œuvre de la logique métier côté serveur (CP7).

## Migrations

## Migrations

localhost/phpmyadmin/index.php?route=/sql&pos=0&db=maillot&table=migrations

phpMyAdmin

Parcourir Structure SQL Rechercher Insérer Exporter Importer Priviléges Opérations Plus

	id	migration	batch
<a href="#">Éditer</a>	1	2025_05_14_122644_create_sessions_table	1
<a href="#">Éditer</a>	2	2025_06_07_001044_add_type_to_addresses_table	1
<a href="#">Éditer</a>	3	2025_06_17_144031_create_users_table	1
<a href="#">Éditer</a>	4	2025_06_17_144035_create_user_sessions_table	1
<a href="#">Éditer</a>	5	2025_06_17_144036_create_user_addresses_table	1
<a href="#">Éditer</a>	6	2025_06_26_123828_create_addresses_table	1
<a href="#">Éditer</a>	7	2025_07_10_112612_create_clubs_table	1
<a href="#">Éditer</a>	8	2025_07_10_112614_create_maillots_table	1
<a href="#">Éditer</a>	9	2025_07_13_034211_create_carts_table	2
<a href="#">Éditer</a>	10	2025_07_13_225109_update_cart_item_maillot_foreign	3
<a href="#">Éditer</a>	11	2025_07_14_025414_add_numero_nom_to_cart_items_table.php	4
<a href="#">Éditer</a>	12	2025_07_16_124954_add_price_to_maillots_table	5

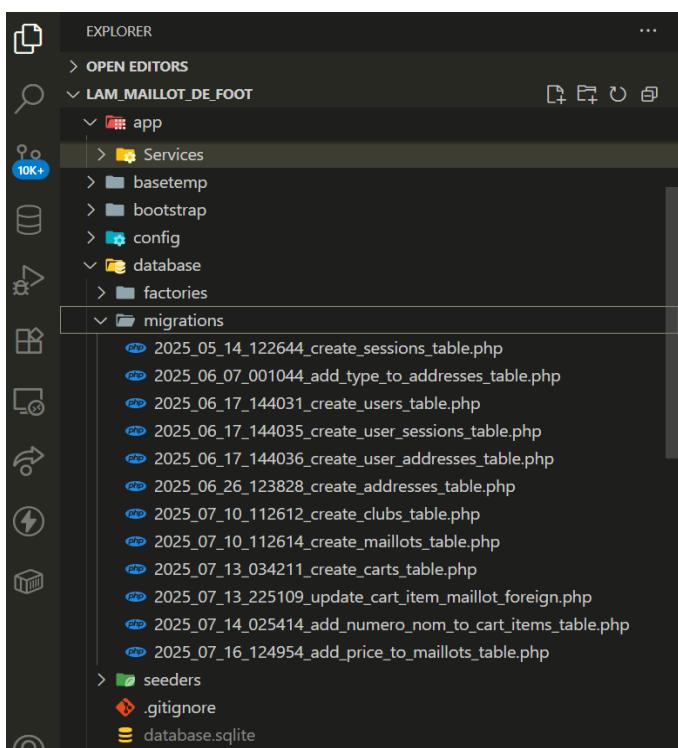
Tout cocher Avec la sélection : Éditer Copier Supprimer Exporter

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

Opérations sur les résultats de la requête

Imprimer Copier dans le presse-papiers Exporter Afficher le graphique Crée une vue Console de requêtes SQL

## Migrations dans VS code



## 4.2. CP6 -Développer des composants d'accès aux données SQL et NoSQL

Alors que le CP5 a consisté à concevoir et mettre en place la base de données relationnelle (création des tables, relations et contraintes), le CP6 s'attache à exploiter cette base au sein de l'application grâce aux modèles Eloquent et aux requêtes SQL. Autrement dit, le CP5 correspond à la construction de la structure, tandis que le CP6 concerne l'accès et la manipulation des données pour alimenter les fonctionnalités du projet.

La mise en place des composants d'accès aux données a constitué une étape clé du développement back-end. L'objectif était de permettre à l'application de communiquer de manière fiable et sécurisée avec la base de données relationnelle, afin de fournir au front-end les informations nécessaires (maillots, clubs, utilisateurs, paniers).

#### **a) Outils et technologies utilisés**

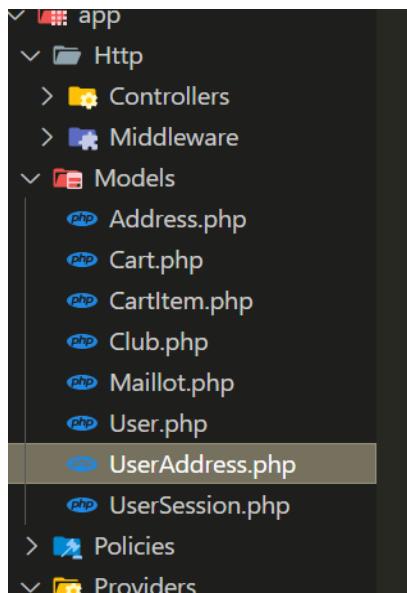
- **Eloquent ORM (Laravel)** pour représenter chaque table de la base sous forme de modèle objet et simplifier les requêtes.
- **Migrations et seeders** pour assurer la cohérence et la reproductibilité des données entre environnements.

#### **b) Mise en œuvre technique**

Création des modèles principaux :

- User et UserAddress pour la gestion des comptes et des adresses,
- Club et Maillot pour le catalogue de produits,
- Cart et CartItem pour la gestion des paniers,
- UserSession : permet d'assurer la gestion et la traçabilité des connexions utilisateurs, renforçant ainsi la sécurité et le suivi des activités. UserSession.php est un modèle Eloquent côté back-end, qui s'appuie sur une table (user\_session). Dans le chapitre précédent (accès aux données) on valorise les modèles, car ils sont la traduction du schéma SQL en objets manipulables dans Laravel.
- Order pour le suivi des commandes (pas encore conçu mais prévu en extension).

#### **Models**



#### c) Définition des relations entre modèles avec Eloquent :

- User et UserAddress : un utilisateur peut avoir plusieurs adresses,
- Club et Maillot : un club peut proposer plusieurs maillots,
- Cart et CartItem : un panier contient plusieurs articles,
- User et Cart : un utilisateur possède un ou plusieurs paniers.

#### d) Mise en place de requêtes typiques :

##### Récupération des maillots d'un club

(dans app/Http/Controllers/ClubController.php, méthode maillots(\$slug)), **ligne 12** :

```
12     $club = Club::where('slug', $slug)->firstOrFail();
13     $maillots = $club->maillots()->get();
```

Pour afficher la liste des maillots associés à un club donné, le contrôleur interroge la relation définie dans le modèle Club.

Ici, firstOrFail() garantit que le club existe bien en base de données, et renvoie une erreur 404 dans le cas contraire. La méthode maillots() exploite la relation *one-to-many* entre Club et Maillot afin de récupérer directement tous les maillots associés.

Ce mécanisme illustre l'un des avantages d'Eloquent : transformer une relation SQL en une méthode simple et lisible dans le code.

## Affichage du détail d'un maillot

(Dans app/Http/Controllers/MaillotController.php, méthode show(\$id)), **ligne 14** :

```
13  ↴      {  
14      $maillot = Maillot::with('club')->findOrFail($id);  
15
```

Lorsqu'un utilisateur consulte la fiche d'un maillot, le contrôleur utilise findOrFail() pour en récupérer les informations.

Dans cet exemple, findOrFail(\$id) permet de charger un maillot précis à partir de son identifiant, tout en gérant l'éventualité où il n'existe pas (erreur 404). La méthode with('club') charge en même temps les informations du club associé, évitant une requête supplémentaire.

Cette approche assure à la fois **efficacité** (requête optimisée) et **sécurité** (gestion des erreurs), tout en rendant le code plus expressif et compréhensible.

**Ajout d'un article au panier** (dans app/Http/Controllers/CartController.php, méthode add(Request \$request), :

```

app > Http > Controllers > CartController.php > CartController > add
13  class CartController extends Controller
96  public function add(Request $request)
97
98      $cart = Cart::firstOrCreate(['user_id' => Auth::id()]);
99
100     $item = $cart->items()
101         ->where('maillot_id', $request->maillot_id)
102         ->where('size', $request->size)
103         ->where('numero', $request->numero ?? '')
104         ->where('nom', $request->nom ?? '')
105         ->first();
106
107     if ($item) {
108         $item->increment('quantity', $request->quantity);
109     } else {
110         $cart->items()->create([
111             'maillot_id' => $request->maillot_id,
112             'size' => $request->size,
113             'quantity' => $request->quantity,
114             'numero' => $request->numero,
115             'nom' => $request->nom,
116         ]);
117     }
118
119     return redirect()->route('cart.show')->with('success', 'Article ajouté au panier');
120 }
121

```

Ce code indique deux points importants :

- Création ou récupération du panier : la méthode firstOrCreate() vérifie si l'utilisateur connecté dispose déjà d'un panier. Si ce n'est pas le cas, un nouvel enregistrement est automatiquement créé en base.
- Recherche d'un article existant : grâce à la relation items(), Eloquent interroge la table cart\_items pour vérifier si le panier contient déjà un produit avec les mêmes caractéristiques (maillot, taille, numéro).

Selon le résultat, l'application choisit soit d'incrémenter la quantité de l'article trouvé, soit de créer une nouvelle ligne avec \$cart->items()->create([...]).

Cet exemple illustre bien l'apport du **CP6** : un accès aux données simplifié, sécurisé et expressif, qui masque la complexité des requêtes SQL derrière des méthodes orientées objet.

Afin de rendre le code plus lisible et de simplifier la manipulation des données, les relations entre les différentes entités de la base ont été définies directement dans les modèles Eloquent. Le tableau ci-dessous récapitule l'ensemble de ces relations, leur type et leur utilisation au sein du projet.

### Tableau -Relations Eloquent du projet Fou2Foot

<b>Modèle</b>	<b>Méthode relation</b>	<b>Type de relation</b>	<b>Cible</b>	<b>Exemple d'utilisation</b>
<b>User</b>	addresses()	hasMany	UserAddress	\$user->addresses
	carts()	hasMany	Cart	\$user->carts()->latest()->first()
	sessions()	hasMany	UserSession	\$user->sessions
	orders()	hasMany	Order	\$user->orders
<b>UserAddress</b>	user()	belongsTo	User	\$address->user
<b>Club</b>	maillots()	hasMany	Maillot	\$club->maillots
<b>Maillot</b>	club()	belongsTo	Club	\$maillot->club
	cartItems()	hasMany	CartItem	\$maillot->cartItems
<b>Cart</b>	user()	belongsTo	User	\$cart->user
	items()	hasMany	CartItem	\$cart->items
<b>CartItem</b>	cart()	belongsTo	Cart	\$item->cart
	maillot()	belongsTo	Maillot	\$item->maillot
<b>Order</b> (prévu)	user()	belongsTo	User	\$order->user
	items()	hasMany	OrderItem	\$order->items
<b>UserSession</b>	user()	belongsTo	User	\$session->user

### e) Sécurité et performance

- Validation des données saisies grâce aux Form Requests de Laravel.
- Protection native contre les injections SQL via les méthodes préparées d'Eloquent.
- Gestion des clés étrangères et des contraintes d'intégrité pour garantir la cohérence des enregistrements.

### Résultat

L'implémentation des composants d'accès aux données assure :

- une interaction fiable entre le back-end et la base de données.
- un code plus lisible et maintenable grâce à l'ORM (Object-Relational Mapping ).
- une intégration fluide avec le front-end via Inertia.js, permettant de charger et manipuler les données (maillots, panier, adresses) en temps réel,
- une base technique solide pour la mise en œuvre de la logique métier côté serveur (CP7).

## Conclusion

La mise en œuvre des composants d'accès aux données a permis de relier efficacement l'application à la base MySQL, tout en tirant parti des fonctionnalités offertes par Eloquent ORM. Grâce aux relations entre modèles (Club → Maillot, Cart → CartItem, etc.), il est possible d'écrire un code expressif, lisible et sécurisé pour gérer l'ensemble des opérations : récupération de listes, affichage de détails ou ajout d'articles au panier.

Ces exemples démontrent la capacité de l'application à manipuler les données de manière fiable, en respectant les contraintes d'intégrité et en simplifiant la logique d'accès. Ce socle technique constitue une étape essentielle avant la mise en place de la logique métier propre à l'application, qui sera abordée dans le **CP7**.

### 4.3. CP7 -Développer la logique métier

Cette étape du projet a consisté à définir et implémenter la logique métier de l'application Fou2Foot, c'est-à-dire les règles qui régissent le fonctionnement du site au-delà de la simple présentation des données. L'objectif principal était de garantir la cohérence des traitements entre les actions de l'utilisateur, la base de données et l'affichage côté client.

Laravel, via son ORM Eloquent, a permis d'encapsuler efficacement cette logique dans les contrôleurs, tout en appliquant des règles métier explicites liées au commerce en ligne : gestion du panier, personnalisation des maillots, suivi des sessions et calcul dynamique des totaux.

#### a) Règles métier principales

Les principales règles fonctionnelles du projet sont les suivantes :

- Panier : un utilisateur ne peut avoir **qu'un seul panier actif** à la fois. La règle métier est implémentée.
- Panier : si un maillot déjà présent (même taille, même personnalisation) est ajouté, la **quantité est incrémentée** au lieu de créer un doublon. La règle métier est implémentée.

- Personnalisation : le nom et le numéro sur le maillot entraînent un **supplément tarifaire** (+3 € pour le nom, +2 € pour le numéro).
- Commandes : Les commandes sont **maquettées** sur le front-end uniquement pour le MVP (pas encore reliées au back-end). La règle métier est prévue mais n'est pas encore réalisée.
- Wishlist : Fonctionnalité de favoris, prévue pour une version ultérieure. La règle métier est prévue mais n'est pas encore réalisée.

## b) Encapsulation de la logique dans les contrôleurs

L'ensemble des traitements métier est regroupé dans les **contrôleurs Laravel**, garantissant la séparation des responsabilités (principe MVC).

Exemple d'implémentation : **ajout d'un article au panier**.

Fichier: app/Http/Controllers/CartController.php

```

96     public function add(Request $request)
97     {
98         $cart = Cart::firstOrCreate(['user_id' => Auth::id()]);
99
100        $item = $cart->items()
101            ->where('maillot_id', $request->maillot_id)
102            ->where('size', $request->size)
103            ->where('numero', $request->numero ?? '')
104            ->where('nom', $request->nom ?? '')
105            ->first();
106
107        if ($item) {
108            $item->increment('quantity', $request->quantity);
109        } else {
110            $cart->items()->create([
111                'maillot_id' => $request->maillot_id,
112                'size' => $request->size,
113                'quantity' => $request->quantity,
114                'numero' => $request->numero,
115                'nom' => $request->nom,
116            ]);
117        }

```

Cette logique répond à deux règles métier :

- **Un panier unique par utilisateur** : la méthode firstOrCreate() évite toute duplication de panier.
- **Pas de doublon d'article** : les conditions sur maillot\_id, size, nom, numero garantissent que l'article est simplement incrémenté.

table carts montrant la clé étrangère user\_id et les relations.

The screenshot shows the MySQL Workbench interface with the 'carts' table selected. The table has four columns: id, user\_id, created\_at, and updated\_at. The 'id' column is defined as a bigint with an unsigned attribute and an AUTO\_INCREMENT extra attribute. The 'user\_id' column is also a bigint. The 'created\_at' and 'updated\_at' columns are timestamp types.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	<b>id</b>	bigint		UNSIGNED	Non	Aucun(e)		AUTO_INCREMENT
2	<b>user_id</b>	bigint		UNSIGNED	Non	Aucun(e)		
3	<b>created_at</b>	timestamp			Oui	NULL		
4	<b>updated_at</b>	timestamp			Oui	NULL		

### c) Calcul du total panier et suppléments

Le calcul du total prend en compte le prix unitaire, la quantité et les éventuelles personnalisations :

```
59    return inertia('Panier', [
60      'cartItems' => $cart->items->map(function($item) {
61        $maillot = $item->maillot;
62        $price = $maillot ? $maillot->price : 0;
63        $image = $maillot ? $maillot->image : null;
64        $name = $maillot ? $maillot->name : '????';
65
66        // Suppléments personnalisation
67        $suppNom = $item->nom ? 3 : 0;
68        $suppNumero = $item->numero ? 2 : 0;
69        $supplement = $suppNom + $suppNumero;
70
71        // Total ligne
72        $total = ($price + $supplement) * $item->quantity;
```

Ce calcul est effectué côté serveur afin de garantir une cohérence stricte entre l'affichage front-end et les données réelles stockées en base. L'approche empêche également toute manipulation frauduleuse des prix côté client.

## d) Gestion et sécurité des sessions

Une table user\_sessions complète le mécanisme d'authentification Laravel. Chaque connexion active est enregistrée avec un identifiant unique et une date d'expiration. Ce suivi permet de renforcer la **sécurité** (éviter les connexions multiples non désirées), de **tracer les activités utilisateurs** et, à terme, de préparer un **tableau de bord administrateur**.

## e) Évolutions prévues et périmètre MVP

Certaines fonctionnalités mentionnées dans les maquettes ne sont pas encore connectées au back-end.

Il s'agit notamment de :

- la page **Commandes**, actuellement en maquette front-end ;
  - la **Wishlist**, non implémentée dans cette version ;
  - la **gestion de paiement Stripe**, en cours d'intégration ;
- 1- a **distinction des rôles (admin / utilisateur)** à finaliser via le champ rôle de la table users.

Ces éléments appartiennent au périmètre d'évolution du projet et ne font pas partie du MVP présenté à la soutenance.

## Bilan

Grâce à cette organisation, la logique métier :

- garantit la **cohérence des traitements** et l'intégrité des données ;
- offre une **expérience fluide** entre le front-end React et le back-end Laravel via Inertia ;
- assure une **sécurité accrue** grâce aux vérifications côté serveur ;
- prépare les **extensions futures** (commandes, paiement, rôles administrateur).

## **4.4. CP8 -Documenter le déploiement d'une application web ou web mobile**

Afin de rendre le projet exploitable en conditions réelles, un processus de déploiement a été défini et documenté. L'objectif était de pouvoir installer l'application sur un serveur distant de manière reproductible et sécurisée, en garantissant la disponibilité du front-end et du back-end.

### **Environnement cible**

- **Serveur Linux (Ubuntu 22.04 LTS)** hébergé sur un VPS.
- **Stack LEMP** (MySQL, PHP 8.2) adaptée à Laravel.
- **Node.js** pour compiler les assets front-end via Vite.
- **Composer** pour gérer les dépendances back-end.

### **Procédure de déploiement**

#### **1. Préparation du serveur :**

Installation des dépendances système (php-cli, php-mysql, mysql-server, nodejs, npm, composer).

Création d'un utilisateur dédié non-root pour sécuriser les opérations.

#### **2. Récupération du projet :**

- Clonage du dépôt Git (git clone ...).
- Configuration des droits d'accès sur storage/ et bootstrap/cache/.

#### **3. Configuration de l'application :**

- Copie et adaptation du fichier .env pour définir les paramètres (connexion MySQL, clés API, variables d'environnement).
- Génération de la clé applicative Laravel :

```
php artisan key:generate
```

#### **4. Base de données :**

- Création de la base MySQL et d'un utilisateur dédié.

- Exécution des migrations et des seeders pour créer les tables et insérer des données initiales :

```
php artisan migrate --seed
```

## 5. Compilation front-end :

- Installation des dépendances front-end :

```
npm install
```

- Build de production avec Vite :

```
npm run build
```

## 6. Configuration du serveur web :

- Création d'un virtual host pointant sur le dossier public/ de Laravel.
- Activation de HTTPS avec **Certbot** (Let's Encrypt).

## 7. Mise en production :

- Lancement de Laravel via **PHP-FPM**.
- Mise en place de **supervision** (fail2ban, UFW firewall) pour renforcer la sécurité.

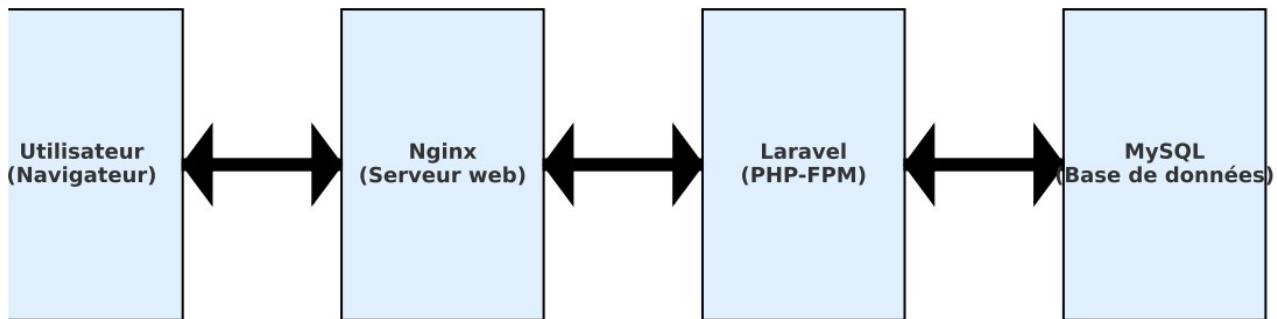
## Conclusion

Grâce à cette procédure, l'application peut être déployée sur un serveur distant en suivant des étapes claires et reproductibles. La documentation garantit que tout développeur ou administrateur système peut installer et configurer le projet de manière identique, assurant ainsi la portabilité et la maintenabilité du système.

## Tableau -Checklist de déploiement

Étape	Commande / Action	Résultat attendu
1. Préparation du serveur	<code>sudo apt update &amp;&amp; sudo apt install php-cli php-mysql mysql-server nginx nodejs npm composer</code>	Serveur prêt avec PHP, MySQL, Node.js, Composer
2. Récupération du projet	<code>git clone &lt;repo&gt; &amp;&amp; cd projet</code>	Code source disponible sur le serveur
3. Permissions	<code>chmod -R 775 storage bootstrap/cache</code>	Laravel peut écrire dans ses répertoires
4. Configuration	<code>cp .env.example .env puis adaptation (DB, APP_KEY, etc.)</code>	Variables d'environnement définies
5. Génération de la clé	<code>php artisan key:generate</code>	Clé d'application générée
6. Base de données	<code>php artisan migrate --seed</code>	Tables et données initiales créées
7. Dépendances front-end	<code>npm install &amp;&amp; npm run build</code>	Assets front compilés en mode production
8. Dépendances back-end	<code>composer install --optimize-autoloader --no-dev</code>	Packages PHP installés et optimisés
9. Configuration Nginx	Virtual host pointant vers public/	Site accessible via HTTP
10. Sécurisation HTTPS	<code>sudo certbot --nginx -d domaine.com</code>	Certificat SSL actif
11. Mise en production	Lancer PHP-FPM, configurer UFW et fail2ban	Application sécurisée et disponible en ligne

## **schéma d'architecture simplifié du déploiement :**



Ce schéma illustre le flux de communication lors de l'exécution de l'application : l'utilisateur interagit via son navigateur, les requêtes sont d'abord traitées par le serveur web qui redirige vers **Laravel (PHP-FPM)** pour la logique applicative. Laravel interroge ensuite **MySQL** pour accéder aux données, et les réponses suivent le chemin inverse afin d'être affichées à l'utilisateur.

### **4.5. Bilan du back-end**

Le développement back-end a permis de doter l'application d'un socle robuste, assurant la cohérence des données et l'application des règles métier. Les différentes étapes ont couvert les compétences professionnelles attendues :

- **CP5** : conception et mise en place d'une base de données relationnelle normalisée, garantissant l'intégrité des informations.
- **CP6** : implémentation de composants d'accès aux données via Eloquent ORM, facilitant la manipulation sécurisée et lisible des tables.
- **CP7** : intégration de la logique métier, avec gestion complète du panier, règles de tarification (suppléments pour la personnalisation), autorisations et préparation de l'extension vers la gestion des commandes.

- **CP8** : documentation du processus de déploiement, permettant la reproductibilité et l'industrialisation du projet dans un environnement serveur sécurisé.

**Résultat** : le back-end fournit désormais un environnement fiable et évolutif, garantissant la cohérence des interactions entre utilisateurs, interface et base de données. Couplé au front-end, il assure une expérience fluide, sécurisée et conforme aux objectifs initiaux du projet.

## Chapitre 5 -Conclusion et perspectives

### 5.1. Bilan global

Le projet Fou2Foot a permis de mettre en pratique l'ensemble des compétences visées par le **Titre Professionnel Développeur Web et Web Mobile (DWWM)**.

Sur le plan **technique**, le développement a couvert :

- la création d'une interface utilisateur moderne et responsive avec React, TailwindCSS et Inertia.js ;
- la mise en place d'une base de données relationnelle complète et normalisée (MySQL) ;
- l'implémentation de la logique métier côté serveur via Laravel et Eloquent ;
- la préparation d'un processus de déploiement reproductible et sécurisé.

Sur le plan **méthodologique**, ce projet a renforcé des compétences clés : analyse d'un besoin, structuration d'un projet, gestion des versions avec Git, documentation technique et présentation professionnelle.

L'application constitue ainsi une **preuve de concept fonctionnelle** : un site e-commerce spécialisé dans la vente de maillots personnalisés, répondant aux attentes des utilisateurs cibles (choisir un produit, le personnaliser, le mettre au panier et préparer une commande).

### 5.2. Difficultés rencontrées et solutions apportées

Comme tout projet de développement, plusieurs obstacles ont dû être surmontés :

- **Gestion des dépendances** : la coexistence entre Laravel, React et TailwindCSS a nécessité un temps d'adaptation, résolu grâce à l'utilisation de Vite et à une structuration claire des répertoires.

- **Accès aux données** : la mise en place correcte des relations entre tables (ex. panier ↔ articles) a représenté un défi. L'utilisation d'Eloquent ORM a simplifié l'écriture des requêtes complexes.

- **Logique métier du panier** : fusionner des articles similaires (même maillot, taille et personnalisation) demandait une réflexion particulière. La solution a consisté à encapsuler cette règle dans le contrôleur puis à envisager un déplacement futur dans le modèle.

- **Sécurité et sessions** : la gestion des connexions utilisateurs et la mise en place de politiques d'accès ont été essentielles pour éviter toute manipulation indue des paniers.

Ces difficultés ont permis de progresser en autonomie et de consolider la compréhension des bonnes pratiques de développement.

### 5.3. Perspectives d'évolution

Bien que le projet soit fonctionnel, plusieurs améliorations et extensions sont envisageables :

- **Finalisation du module Commandes** : implémenter le modèle Order, gérer le passage en caisse et l'historique des commandes.

- **Intégration des paiements en ligne** : connecter l'application à une API tierce (Stripe, PayPal) pour finaliser le processus d'achat.

- **Mise en place de la partie Administrateur** : l'administrateur doit pouvoir accéder à l'ensemble des données utilisateurs, gérer les stocks disponibles et pouvoir ajouter ou supprimer des maillots.

- **Amélioration des performances** : mise en cache des pages catalogue, optimisation SQL et intégration d'un CDN pour les images.

- **Renforcement de la sécurité** : authentification à deux facteurs (2FA), gestion avancée des sessions utilisateurs.

- **Tests automatisés** : mise en place de tests unitaires et fonctionnels avec PHPUnit et Laravel Dusk, afin de valider le bon fonctionnement en continu.

- **Déploiement évolutif** : automatisation via Docker ou CI/CD (GitHub Actions, GitLab CI) pour simplifier la maintenance et la montée en charge.

## Conclusion générale

Ce projet a constitué une expérience formatrice, permettant de passer de la conception à la mise en œuvre complète d'une application web. L'association entre **technologies modernes** (Laravel, React, Tailwind, Inertia, Vite) et **bonnes pratiques de développement** (ORM, migrations, sécurité, documentation, déploiement) illustre pleinement les compétences acquises dans le cadre du TP DWWM.

Il offre une base solide pour des évolutions futures, mais aussi un exemple concret de mise en situation professionnelle, préparant à intégrer des projets réels en entreprise.

## Résumé – Conclusion

Le projet **Fou2Foot** a permis de concevoir et de développer une application e-commerce spécialisée dans la vente de maillots personnalisés. Il s'inscrit dans le cadre du **Titre Professionnel Développeur Web et Web Mobile (DWWM)** et a couvert l'ensemble des compétences attendues, du front-end au back-end jusqu'au déploiement.

Sur le plan **technique**, l'application repose sur une architecture moderne :

- **Front-end** : React, TailwindCSS et Inertia.js pour une interface responsive, dynamique et accessible.
- **Back-end** : Laravel et MySQL pour la gestion des données, Eloquent ORM pour simplifier les requêtes, et une logique métier structurée (gestion du panier, règles de tarification, sécurité).
- **Déploiement** : documentation complète du processus d'installation et de mise en production sur un serveur Linux avec Nginx et MySQL.

Sur le plan **méthodologique**, ce projet a renforcé la capacité à analyser un besoin, à structurer un projet logiciel, à travailler sur Git et à documenter les étapes techniques de manière claire et reproductible.

Plusieurs **difficultés** ont jalonné le développement (gestion des dépendances, relations de données, fusion d'articles similaires dans le panier), mais elles ont été surmontées grâce à la maîtrise progressive de Laravel, d'Eloquent et des bonnes pratiques de sécurité.

Enfin, des **perspectives d'évolution** sont identifiées : finalisation du module Commandes et du paiement en ligne, amélioration des performances, mise en place de tests automatisés et intégration d'outils modernes de déploiement (Docker, CI/CD).

En conclusion, ce projet constitue une **preuve de concept fonctionnelle** et une expérience professionnalisante. Il démontre la capacité à conduire un projet web complet, depuis la conception jusqu'au déploiement, et constitue une base solide pour des évolutions futures en contexte professionnel.

## ANNEXES

The screenshot shows a login form with a light blue header containing the words "Connexion" and "Inscription". Below the header is a white input field labeled "Nom d'utilisateur ou Email" which contains the text "testuser". Underneath this field is a red error message: "Identifiants incorrects ou compte inactif.". Below the input field is another input field labeled "Mot de passe" containing several dots. To the right of this input field is a small circular icon with an eye symbol. At the bottom of the form is a large blue button with the text "Se connecter" in white.