

DOSSIER DE PROJET PROFESSIONNEL

REALISATION DE L'APPLICATION WEB ET MOBILE

FOU2FOOT

PAR GAËLICK RIGOUX

Sommaire du projet professionnel

1. Introduction générale

- a) Contexte et objectifs du projet
- b) Enjeux utilisateurs et métier
- c) Périmètre et limites du projet
- d) Méthode et organisation

2. Vue d'ensemble technique

2.1. Pile technologique

2.2. Architecture de l'application

- a) Architecture générale
- b) Schéma de fonctionnement
- c) Schéma global d'architecture
- d) Schéma de base de données simplifié
- e) Explication des relations

2.3. Qualité et sécurité

- a) Principaux mécanismes de sécurité
- b) Utilisation d'Eloquent ORM
- c) Séparation des responsabilités

3. Développement de la partie front-end sécurisée

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

- a) Outils et versions utilisées
- b) Organisation des sources
- c) Finalités offertes par la configuration

3.2. CP2 - Maquette une application

- a) Parcours et zones principales
- b) Wireframe

c) Maquette

3.3. CP3 - Réaliser une interface utilisateur statique et adaptable

- a) Pages statiques
- b) Composants réutilisables
- c) Présentation des pages front-end
- d) Outils et technologies utilisés
- e) Accessibilité
- f) Lighthouse

3.4. CP4 - Développer une interface utilisateur dynamique

- a) Navigation dynamique
- b) Interaction avec le panier
- c) Mises à jour côté client
- d) Sécurité front-end

3.5. Accessibilité et éco-conception

- a) Principes appliqués
- b) Score Lighthouse
- c) Compétences mobilisées
- d) Éco-conception

3.6. Bilan du front-end

4. Développement back-end P58

4.1. CP5 - Mettre en place une base de données relationnelle P58

- a) Méthode Merise(MCD, MLD,MPD) P58
- b) Outils et technologies utilisés
- c) Conception et organisation des données
- d) Mise en œuvre technique

4.2. CP6 - Développer des composants d'accès aux données

- a) Outils et technologies utilisés
- b) Mise en œuvre technique
- c) Définition des relations entre modèles
- d) Mise en place de requêtes typiques
- e) Sécurité et performance

4.3. CP7 - Développer la logique métier

- a) Règles métier principales

- b) Encapsulation de la logique dans les contrôleurs
- c) Calcul du total panier et suppléments
- d) Gestion et sécurité des sessions
- e) Évolutions prévues et pertinentes MVP

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

- a) Environnement cible
- b) Procédure de déploiement
- c) Automatisation et perspectives

4.5. Bilan du back-end

5. Conclusion et perspectives

- a) Bilan global
- b) Difficultés rencontrées et solutions proposées
- c) Perspectives d'évolution

6. Annexes

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

J'ai circonscrit le périmètre de l'application à :

- 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, viderage).
- la gestion d'un compte utilisateur (authentification, adresses).
- la simulation d'un processus de commande.

En revanche, je n'ai pas pu implémenter certaines fonctionnalités 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 Wishlist.

1.4. Méthode et organisation

J'ai mené le projet de manière incrémentale et agile :

- J'ai procédé au 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 :

- J'ai mis en place progressivement le front-end et le back-end.
- J'ai Utilisé Git pour la gestion des versions (versioning) et la collaboration (création de branches afin de procéder à des essais sans affecter le code initial).
- J'ai effectué des 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) : framework¹ 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(SGBDR).

¹ 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).

Exemple de routes :

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

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 induit :

- 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

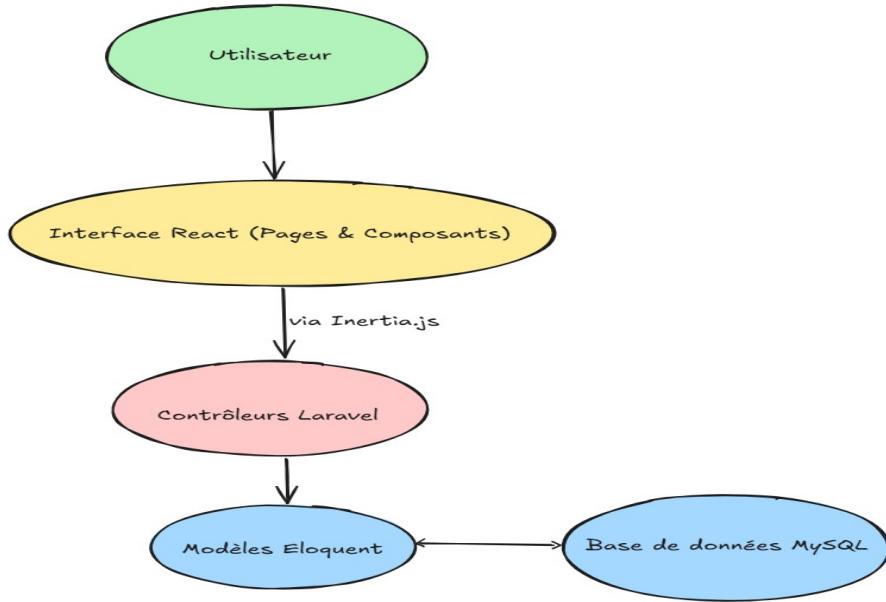
Simplicité de déploiement (une seule application à héberger), **Sécurité renforcée, Performances** (Vite et React assurent un rendu rapide et interactif), **Cohérence des données** (Laravel gère la logique métier et la validation, React se concentre sur l'affichage).

Bilan

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**.

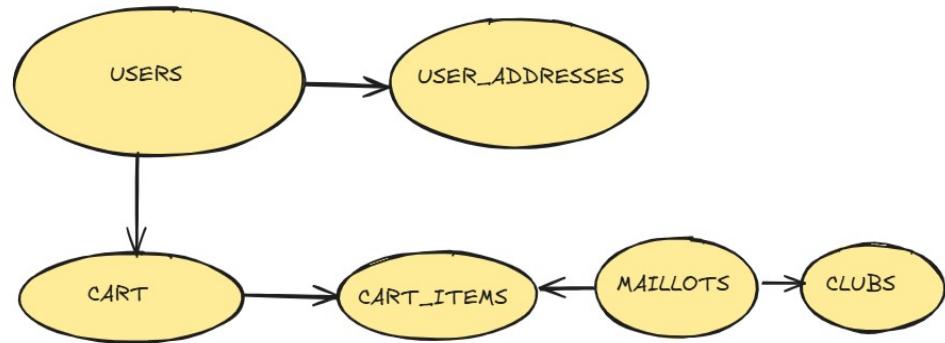
Laravel centralise les données, la validation et les règles métier, tandis qu'Inertia.js fait office de **passerelle** vers une interface fluide et moderne.

c) Schéma global d'architecture



Le **Front-end (React)** : gère l'affichage et l'interaction utilisateur, tandis qu'**Inertia.js** : transmet les données entre Laravel et React sous forme de props. Le **Back-end (Laravel)** : applique la logique métier, valide les données et interagit avec la base de données. Quant à la **Base de données (MySQL)**, elle 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         }
32     }
33 }
```

- **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';
27 }
```

- **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. 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 app/Models/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     }

```

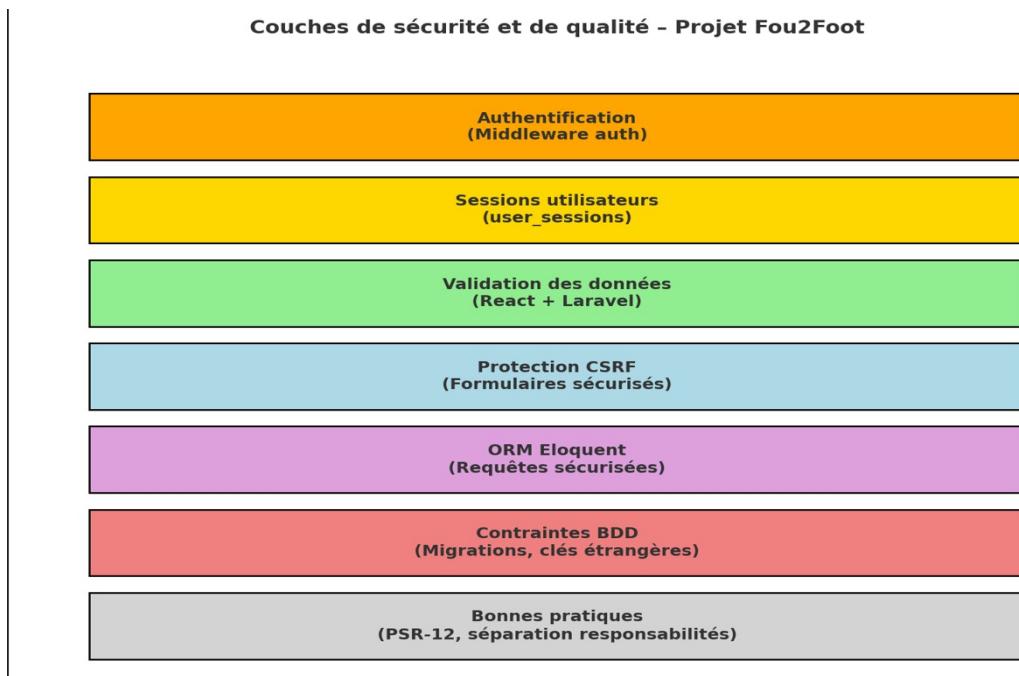
`$club -> maillots` : permet de récupérer tous les maillots associés à un club.

Dans app/Models/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 garantissant 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 Replacement²), offrant un rechargement instantané et fluide.
- npm run build génère les fichiers optimisés pour la production, garantissant des performances maximales grâce à la minification³ et à la suppression des classes inutilisées.

b) Organisation des resources

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

c) Finalités offertes par la configuration

Grâce à cette configuration, j'ai pu :

- développer efficacement avec **React + Tailwind** sans surcharge CSS.

² recharge à chaud sans recharge complet de page.

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

- 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, 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 afin de 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 couleur ni style définitif. 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'enchaînement des pages, les éléments à insérer dans le tableau de bord, la page d'accueil, les pages du catalogue, la page 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 (voir annexes)

PAGE ACCUEIL

Version desktop



Connexion/ Inscription

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

Tableau de bord

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 Livraisons Contact Newsletter

Liste Maillots

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 Livraisons Contact Newsletter

This screenshot shows a user interface for a football apparel website. At the top, there's a search bar labeled 'recherche produit' and a navigation bar with links like 'FOOT EN STOCK/ CULTURE FOOT', 'compte', and 'Déconnexion'. Below that is a secondary navigation bar with categories such as 'Equipes nationales', 'Ligue 1', 'premier league', 'Bundesliga', 'liga', 'série A', and 'Panier'. The main content area features a grid of nine placeholder boxes for products, each with a blue header bar containing the text 'NOM CLUB/ EQUIPE NATIONALE'. Below the grid, there's a section labeled 'descriptions succinctes du produit'. At the bottom, there's another navigation bar with links for 'mentions légales', 'Livraisons', 'Contact', and 'Newsletter'.

Détail maillot

Frame 26

recherche produit FOOT EN STOCK/ CULTURE FOOT compte Déconnexion

Equipes nationales Ligue 1 premier league Bundesliga liga série A Panier

Taille

Personnalisation

Quantité

Ajouter au panier

Description

Avis

Frame 32

mentions légales Livraisons Contact Newsletter

This screenshot shows a detailed view of a football jersey product. It includes fields for 'Taille' (size) and 'Personnalisation' (personalization), both with input fields. There's also a 'Quantité' (quantity) field and a prominent red 'Ajouter au panier' (Add to cart) button. Below the product details, there are sections for 'Description' and 'Avis' (reviews). At the bottom, there's a navigation bar with links for 'mentions légales', 'Livraisons', 'Contact', and 'Newsletter'.

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

tableau de bord

The wireframe illustrates the structure of the Fou2Foot dashboard. At the top, there's a header bar with the Fou2Foot logo, a search bar, and account-related links like 'Mon compte' and 'Déconnexion'. Below the header is a navigation menu with links to '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 a 'Se déconnecter' link. The main content area features a 'Bienvenue, marion!' message with the user's email. It then displays a section titled 'Mon Tableau de Bord' with four cards: 'Mes Commandes' (track purchases), 'Adresse' (manage delivery address), 'Détails du compte' (edit personal information), and 'Ma wishlist' (view favorite items). Below this is an 'Activité récente' section showing a purchase history and an address modification. The footer contains the Fou2Foot logo, a 'Informations' section with legal links, a 'Service client' section with contact info, and social media links for 'Nous suivre'.

Bienvenue, marion !
Votre Email : marion@hotmail.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

Vous avez passé une commande le 21 mai.

Adresse modifiée le 18 mai.

Fou2Foot
La passion du football au cœur de vos équipements

Informations

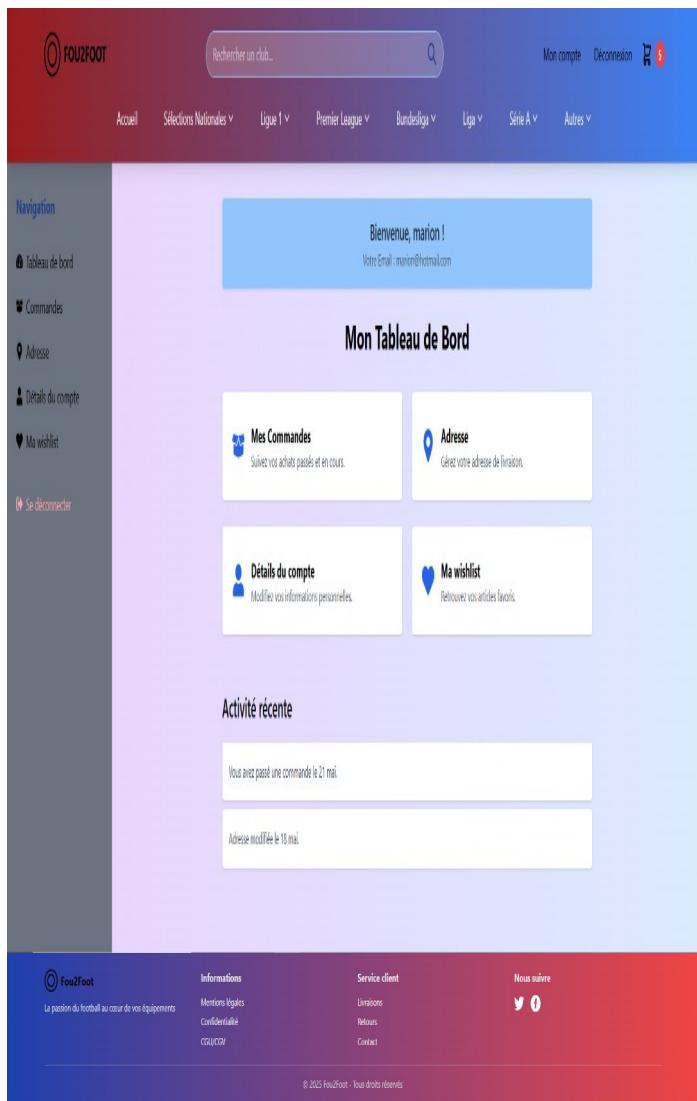
Mentions légales
Confidentialité
CGU/CGV

Service client

Livraisons
Retours
Contact

Nous suivre

version mobile



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.

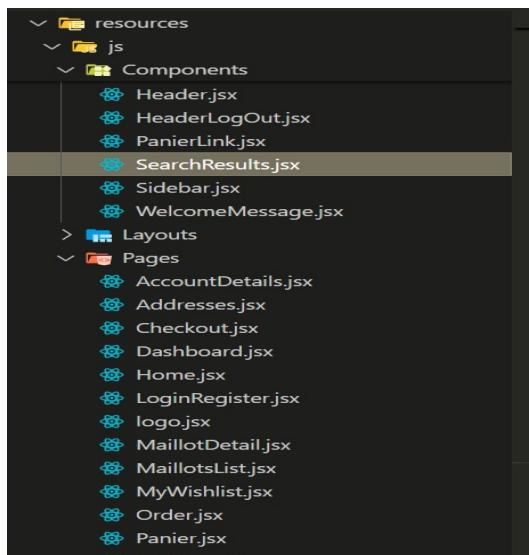
Cette étape correspond à la mise en œuvre concrète des maquettes, sans intégrer encore la logique dynamique ou les traitements métier.

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 header with the text "Maillots de Football Officiels" and a subtext "Retrouvez les tenues des plus grands clubs et des plus grandes sélections". Below this, there is a paragraph of text about the service offered. The main content area is titled "Nos Maillots Phares" and displays four football jerseys in separate boxes: a blue and white jersey, a white and blue jersey, a grey and red jersey with "Emirates FLY BETTER" and "OL" logos, and a dark blue and red striped jersey with "Emirates FLY BETTER" and "OL" logos.

Pages responsives voir annexes

Page inscription

Connexion Inscription

Nom d'utilisateur

Email

Mot de passe

Confirmer le mot de passe

S'inscrire

Fou2Foot Informations Service client Nous suivre

Page connexion

Connexion Inscription

Nom d'utilisateur ou Email

Mot de passe

Se connecter

Fou2Foot Informations Service client Nous suivre

Tableau de bord

The screenshot shows the FOU2FOOT dashboard at 127.0.0.1:8000/dashboard. The top navigation bar includes a logo, a search bar, and links for Mon compte, Déconnexion, and a shopping cart with 7 items. The main content area features a welcome message "Bienvenue, mina !" with your email "Votre Email : mina@carp.com". Below it is a section titled "Mon Tableau de Bord" with four cards: "Mes Commandes" (track purchases), "Adresse" (manage delivery address), "Détails du compte" (edit personal information), and "Ma wishlist" (find favorite items). A sidebar on the left lists navigation options: Tableau de bord, Commandes, Adresse, Détails du compte, Ma wishlist, and Se déconnecter.

Détails du compte

The screenshot shows the account details page at 127.0.0.1:8000/accountdetails. It features a "Mon compte" section with a subtitle "Gérez vos informations personnelles et vos préférences". Below this is a "Informations personnelles" card with fields for Identifiant (minna), Nom complet (MINA BELLA), Email (minna@carp.com), and Téléphone (77 33 11 88 66). At the bottom is a "Sécurité" card with a "Changer le mot de passe" link. The left sidebar remains the same as the dashboard, listing the same navigation options.

Adresses

127.0.0.1:8000/addresses

FOU2FOOT Rechercher un club... Mon compte Déconnexion 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

Mes adresses

Adresse de facturation

Facturation Par défaut

MINA BELLA
rue gandolphe
06210 Mandelieu
FR
77 33 11 88 66

[Modifier](#) [Supprimer](#)

Liste Maillots d'un club

127.0.0.1:8000/clubs/olympique-lyonnais/maillots

FOU2FOOT Rechercher un club... Mon compte Déconnexion 4

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

Maillots de Olympique Lyonnais

Découvrez notre collection de maillots Olympique Lyonnais - 8 maillots disponibles

Domicile 2024

Extérieur 2024

maillot third

Maillot 75 ans

Présentation Maillot

The screenshot shows a web browser displaying a product page for a football jersey. The URL in the address bar is 127.0.0.1:8000/maillots/35. The page title is "Olympique Lyonnais Domicile 2024". The jersey is white with blue and red accents, featuring the "Emirates FLY BETTER" logo. The price is listed as 20€. The user has selected size S and quantity 1. There are checkboxes for adding a number (+2€) and a name (+3€). A blue button at the bottom right says "AJOUTER AU PANIER". The top navigation bar includes links for Accueil, Sélections Nationales, Ligue 1, Premier League, Bundesliga, Liga, Serie A, and Autres.

Panier de l'utilisateur

The screenshot shows a web browser displaying a user's shopping cart. The URL in the address bar is 127.0.0.1:8000/panier. The title of the page is "Mon panier". The table lists five items:

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

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

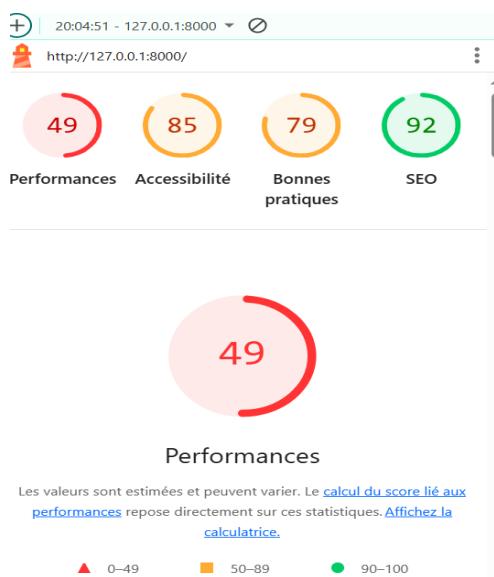
```
152      <button
153        type="button"
154        onClick={() => setShowConfirm(!showConfirm)}
155        className="absolute inset-y-0 right-0 px-3 flex items-center"
156        aria-label={showConfirm ? "Masquer la confirmation du mot de passe" : "Afficher la confirmation"
157      }
158      >
159        <EyeIcon show={showConfirm} />
160      </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).

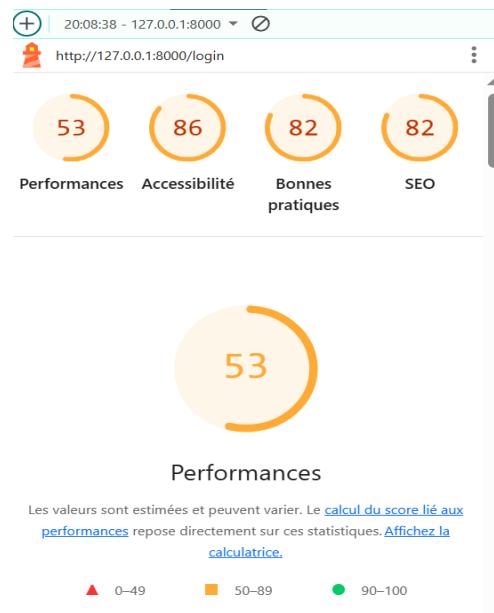
Les scores obtenus sont les suivants :

Score lighthouse accueil



Interprétation et axes d'amélioration page accueil (voir annexes)

Score lighthouse connexion/ inscription



Interprétation et axes d'amélioration page connexion /inscription (voir annexes)

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

```

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 :

```

20
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    {
30      onSuccess: () => alert("Maillot ajouté au panier !"),
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 } public function add(Request $request) { 96     $cart = Cart::firstOrCreate(['user_id' => Auth::id()]); 97 98     $item = $cart->items() 99         ->where('maillot_id', $request->maillot_id) 100        ->where('size', $request->size) 101        ->where('numero', $request->numero ?? '') 102        ->where('nom', $request->nom ?? '') 103        ->first(); 104 105     if ($item) { 106         $item->increment('quantity', $request->quantity); 107     } else { 108         $cart->items()->create([ 109             'maillot_id' => $request->maillot_id, 110             'size' => $request->size, 111             'quantity' => $request->quantity, 112             'numero' => $request->numero, 113             'nom' => $request->nom, 114             ]); 115     } 116 117     return redirect()->route('cart.show')->with('success', 'Article ajouté au panier'); 118 } 119 } 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 (MailotDetail.jsx)

```

6  export default function MailotDetail({ 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             className="mr-2"
71     />
```

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 rechargeement 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  ˜   {
30  ˜     onSuccess: () => alert("Maillot ajouté au panier !"),
31  ˜   });
32 }
```

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             ];
89     ]:
```

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
10     <main className="flex-1">
11       |   {children}
12     </main>
13     <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>
67      </div>
68    </main>
```

- 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:scale-105">
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.

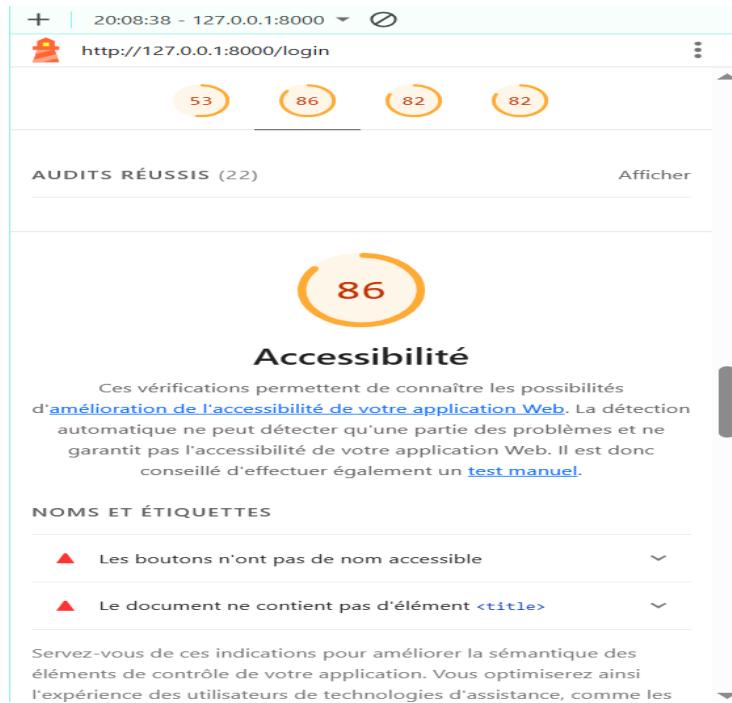
resources/js/Pages/LoginRegister.jsx -label lié par htmlFor + messages d’erreur

```
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 ${isLogin ? "text-blue-600 border-b-2 border-blue-600"}`}
53 |               role="tab"
54 |               aria-selected={isLogin}
55 |             >
56 |               Connexion
57 |             </button>
58 |             <button
59 |               onClick={() => setIsLogin(false)}
60 |               className={`${px-4 py-2 font-medium text-lg ${!isLogin ? "text-blue-600 border-b-2 border-blue-600"}`}
61 |               role="tab"
62 |               aria-selected={!isLogin}
63 |             >
64 |               Inscription
65 |             </button>
66 |           </div>
```

b) Lightscore page login (connexion/inscription)



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 et 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 également 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, mes pages front-end sont désormais en capacité d'interagir efficacement avec la partie back-end. Le front-end doit offrir 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 Fou2Foot, 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 le courant des 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 de facto un outil analytique qui facilite la création de base de données et de projets informatiques.

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 :

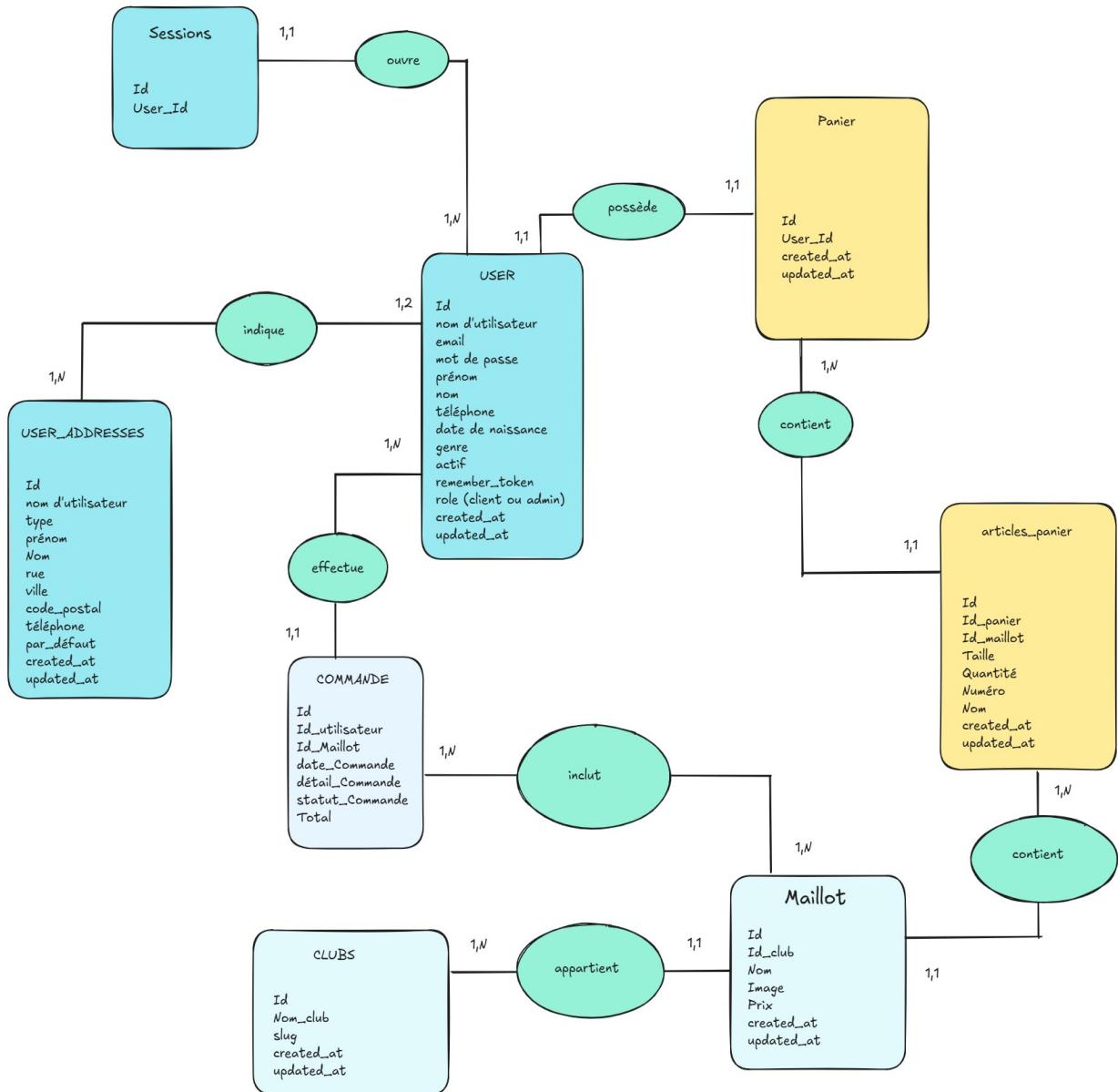
- des **migrations Laravel**, assurant la création et la version des tables ;
- des **modèles Eloquent**, pour gérer les relations (hasMany, belongsTo,hasOne) ;
- des **seeders**, pour insérer des données de test (clubs, maillots, utilisateurs) et valider les fonctionnalités.

Exemple de migration simplifiée :

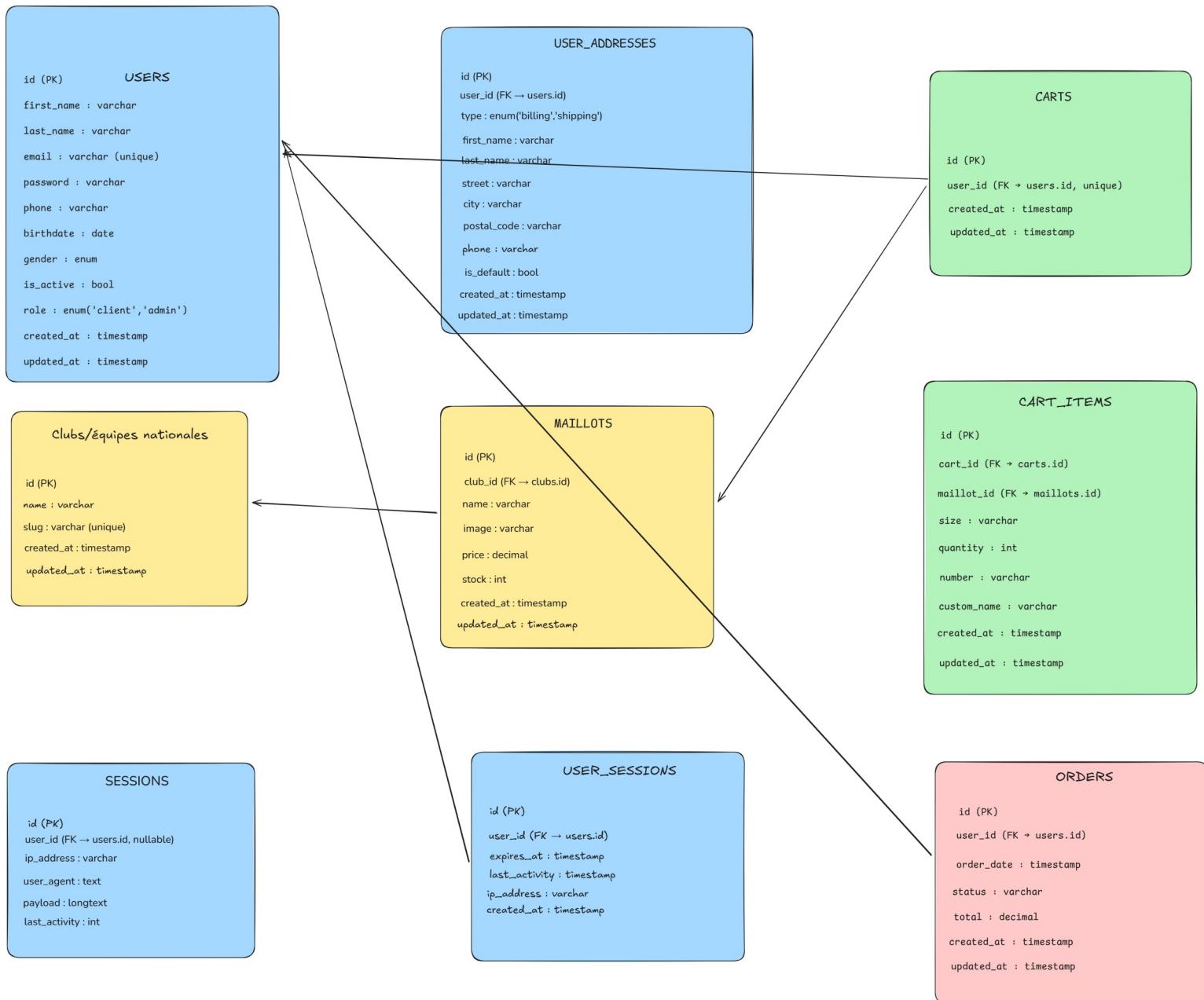
fichier :\www\Lam_Maillot_de_foot\database\migrations
2025_07_10_112614_create_maillots_table.php

```
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         });
21     }
22
23     public function down(): void
24     {
25         Schema::dropIfExists('maillots');
26     }
27 }
```

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 (voir annexes)

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, les Migrations Laravel pour créer, versionner et maintenir la structure de la base de données de manière automatisée, les 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 (voir annexes)

localhost/phpmyadmin/index.php?route=/database/structure&db=maillot

phpMyAdmin

Structure SQL Rechercher Requête Exporter Importer Opérations Priviléges Plus

Filtres

Contenant le mot :

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
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	21,0 kio	10,0 kio
users	Parcourir Structure Rechercher Insérer Vider Supprimer	18	MyISAM	utf8mb4_unicode_ci	18,5 kio	20 kio
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 kio 10,5 kio

Tout cocher / Vérifier les tables non optimisées Avec la sélection :

Console de requêtes SQL de données

Table users

The screenshot shows the phpMyAdmin interface for the MySQL server at localhost. The left sidebar lists the database structure, including tables like Nouvelle base de données, Nouvelle table, cart, cart_items, clubs, maillots, migrations, sessions, users, and user_addresses. The main area displays the 'users' table from the 'maillot' database. The table has columns: id, username, email, password, first_name, and last_name. The data is as follows:

		id	username	email	password	first_name	last_name		
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	Albundy	albundy@gmail.com	\$2y\$12\$Uvn5QMkeIwZ9jKtIn5codO.qKMrfFCFvNc8KwPsMJuRZ...	AL	BU
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	prasanna	prasann@yahoo.in	\$2y\$12\$O2AMY2P67ahq57agiSkXeWHCfnoEdhRbszeQsN.wA...	Prasanna Lakshmi	Dai
<input type="checkbox"/>	Éditer	Copier	Supprimer	3	Lolo	lolo@gmail.com	\$2y\$12\$BKwgXMX9dDzQOKZH1nxw.uq24jqBFQggGMKFnivcxr...	Lolo	Rig
<input type="checkbox"/>	Éditer	Copier	Supprimer	4	Sophie69	sophie@gmail.com	\$2y\$12\$U6nJRH09.yTiXoE3rdgeD6e6.M400Mm5GDogmpkQw...	NULL	NU
<input type="checkbox"/>	Éditer	Copier	Supprimer	5	marion	marion@hotmail.com	\$2y\$12\$desYkbbsYL9MII1r0YB6eu7Yzd7.LdCmhX.j69F4W...	Marion	LAf
<input type="checkbox"/>	Éditer	Copier	Supprimer	6	magnum	magnum@yahoo.us	\$2y\$12\$JGfAGqsS.cuglCQviXDLfOram.WaSjCidCnMP6.xsyX...	thomas	Ma
<input type="checkbox"/>	Éditer	Copier	Supprimer	7	rais	rais@yahoo.fr	\$2y\$12\$SS5cQ9VlH7dG.A0maBwL..WaUcnDClou/qFTm0qnZ5...	Rais	Pac

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	id	bigint		UNSIGNED	Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
2	username	varchar(50)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
3	email	varchar(191)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
4	password	varchar(191)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
5	first_name	varchar(100)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
6	last_name	varchar(100)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
7	phone	varchar(20)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
8	birth_date	date			Oui	NULL			Modifier Supprimer Plus
9	gender	enum('male', 'female', 'other')	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
10	email_verified_at	timestamp			Oui	NULL			Modifier Supprimer Plus
11	is_active	tinyint(1)			Non	1			Modifier Supprimer Plus
12	remember_token	varchar(100)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
13	created_at	timestamp			Oui	NULL			Modifier Supprimer Plus
14	updated_at	timestamp			Oui	NULL			Modifier Supprimer Plus

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

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).

Bilan

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

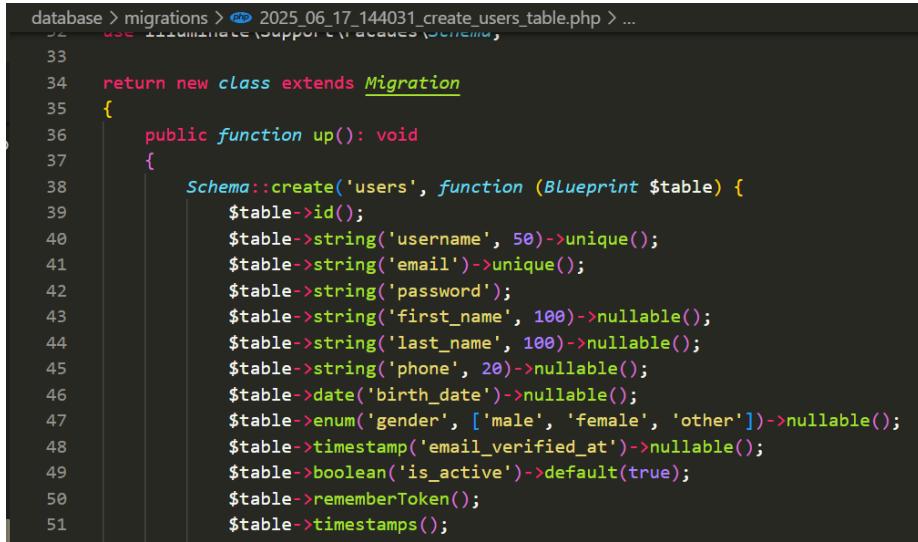
PHPMyAdmin

The screenshot shows the PHPMyAdmin interface for a MySQL database named 'maillot'. The left sidebar lists various tables: Nouvelle base de données, fou2foot, maillot (containing Nouvelle table, carts, cart_items, clubs, maillots, migrations, sessions, users, user_addresses), and others like Nouvelle base de données, Nouvelle table, etc. The 'migrations' table is selected in the 'maillot' database. The main area displays the data from the 'migrations' table:

		id	migration	batch
<input type="checkbox"/>		1	2025_05_14_122644_create_sessions_table	1
<input type="checkbox"/>		2	2025_06_07_001044_add_type_to_addresses_table	1
<input type="checkbox"/>		3	2025_06_17_144031_create_users_table	1
<input type="checkbox"/>		4	2025_06_17_144035_create_user_sessions_table	1
<input type="checkbox"/>		5	2025_06_17_144036_create_user_addresses_table	1
<input type="checkbox"/>		6	2025_06_26_123828_create_addresses_table	1
<input type="checkbox"/>		7	2025_07_10_112612_create_clubs_table	1
<input type="checkbox"/>		8	2025_07_10_112614_create_maillots_table	1
<input type="checkbox"/>		9	2025_07_13_034211_create_carts_table	2
<input type="checkbox"/>		10	2025_07_13_225109_update_cart_item_maillot_foreign	3
<input type="checkbox"/>		11	2025_07_14_025414_add_numero_nom_to_cart_items_table	4
<input type="checkbox"/>		12	2025_07_16_124954_add_price_to_maillots_table	5

Below the table, there are buttons for selecting all rows, performing bulk operations (Éditer, Copier, Supprimer), and exporting data. At the bottom, there are links for Imprimer, Copier dans le presse-papiers, Exporter, Afficher le graphique, and Crée une vue.

Migrations dans VS code



A screenshot of a code editor in VS Code displaying a PHP migration file. The file is named `2025_06_17_144031_create_users_table.php`. The code defines a new migration class that extends `Migration`. The `up()` method creates a table named `users` with various fields: `id`, `username` (unique), `email` (unique), `password`, `first_name`, `last_name`, `phone`, `birth_date`, `gender` (enum with values 'male', 'female', 'other'), `email_verified_at`, `is_active` (boolean with default value true), `rememberToken`, and `timestamps`.

```
database > migrations > 2025_06_17_144031_create_users_table.php > ...
33
34     return new class extends Migration
35     {
36         public function up(): void
37         {
38             Schema::create('users', function (Blueprint $table) {
39                 $table->id();
40                 $table->string('username', 50)->unique();
41                 $table->string('email')->unique();
42                 $table->string('password');
43                 $table->string('first_name', 100)->nullable();
44                 $table->string('last_name', 100)->nullable();
45                 $table->string('phone', 20)->nullable();
46                 $table->date('birth_date')->nullable();
47                 $table->enum('gender', ['male', 'female', 'other'])->nullable();
48                 $table->timestamp('email_verified_at')->nullable();
49                 $table->boolean('is_active')->default(true);
50                 $table->rememberToken();
51                 $table->timestamps();
52             });
53         }
54     }
55 }
```

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

```
app > Models > Club.php > ...
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
```

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

```

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, nom).

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

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

Bilan

La mise en œuvre des composants d'accès aux données m'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 indiquent que de l'application est en capacité de 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 la partie suivante (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 fonctionnalité 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 'carts' table structure in MySQL Workbench. The table has four columns: 'id' (primary key, auto-increment), 'user_id' (foreign key to the 'users' table), 'created_at' (timestamp), and 'updated_at' (timestamp). The 'user_id' column is defined as a foreign key.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	id 📃	bigint		UNSIGNED	Non	Aucun(e)		AUTO_INCREMENT
2	user_id 📃	bigint		UNSIGNED	Non	Aucun(e)		
3	created_at	timestamp			Oui	NULL		
4	updated_at	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 ;
- la **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 prédéfini.

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 et prépare les **extensions futures** (commandes, paiement, rôles administrateur).

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

Cette étape a eu pour objectif de valider la fiabilité du projet et la cohérence entre les données, les interfaces et les règles métier. Les tests ont été menés en local sur l'environnement Laravel, à partir d'un jeu de données généré automatiquement par les *seeders*.

a) Mise en place des données de test

Afin de disposer d'un environnement de démonstration cohérent, j'ai créée ddes *seeders*.

UserSeeder : création automatique de deux comptes utilisateurs (administrateur et client).

```
database > seeders > UserSeeder.php > UserSeeder > run
10  class UserSeeder extends Seeder
11      public function run(): void
12
13          // Utilisateur test
14          $testUser = User::create([
15              'username' => 'testuser',
16              'email' => 'test@fou2foot.com',
17              'password' => Hash::make('password123'),
18              'first_name' => 'Test',
19              'last_name' => 'User',
20              'phone' => '+33987654321',
21              'is_active' => true,
22          ]);
23
24
25
26
27
28
29
30
31
32
33
34
35
```

Le *UserSeeder* illustre le principe de génération de comptes de test. Il insère un utilisateurs type, permettant de simuler le parcours “client”.

Ces données automatisées ont permis de valider les pages principales sans ressaissie manuelle : affichage des maillots, ajout au panier, connexion et consultation du compte utilisateur.

b) Vérification fonctionnelle

Des tests manuels ont ensuite été effectués pour valider les principaux scénarios utilisateurs :

Connexion / Inscription : test de la double validation front (React) et back (Laravel).

Affichage des produits : contrôle du bon chargement des maillots et de leur association au club correct.

Ajout au panier : vérification du calcul dynamique des totaux dans *MaillotDetail.jsx* et de la persistance des données via *CartController.php*.

Protection des formulaires : validation du fonctionnement du jeton CSRF et du hachage des mots de passe.

c) Résultats et bilan

Les tests réalisés montrent une cohérence complète entre les données stockées en base MySQL et les données affichées côté interface. L'utilisation de Laravel et d'Eloquent ORM garantit l'intégrité des relations et la sécurité des échanges. Les seeders facilitent la reproductibilité des tests et assurent une base stable pour les futures évolutions du projet.

Cette phase de contrôle permet de valider la qualité du MVP avant tout déploiement futur et démontre la fiabilité du code, de la base et de l'expérience utilisateur.

4.5. Bilan du back-end

Le développement back-end m'a aidé à 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.

Bilan : 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

J'ai pu mettre en pratique par le biais de mon projet Fou2Foot 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**, grâce à ce projet j'ai 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. Perspectives d'évolution

Bien que le projet soit fonctionnel, j'envisage plusieurs améliorations et extensions :

- **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.
- **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 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.

ANNEXES

3.2. CP2 -Maquetter une application



Connexion/ Inscription

Version mobile

Frame 27

FOOT EN STOCK/ CULTURE FOOT

recherche produit

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 horizontal split layout. The left side is for registration, containing fields for email, password, and a 'remember me' checkbox, along with a 'Register' button. The right side is for logging in, containing fields for identifier or email and password, along with a 'Login' button. The top navigation bar includes links for football stock, culture, search product, account, and logout. The bottom navigation bar includes links for legal mentions, delivery, contact, and newsletter.

Version desktop

Frame 27

FOOT EN STOCK/ CULTURE FOOT

recherche produit

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 split-screen layout to the mobile version. The left side is for registration, and the right side is for logging in. Both sides include fields for identifier or email and password, along with a 'Validate' button. The top navigation bar and bottom footer are identical to the mobile version, providing links for football stock, culture, search product, account, logout, legal mentions, delivery, contact, and newsletter.

Tableau de bord

Version mobile

Frame 30

FOOT EN STOCK/ CULTURE FOOT

recherche produit

compte Déconnexion

Equipes nationales Ligue 1 premier league Bundesliga liga série A Panier

NAVIGATION

Ma Commande

Adresses

Détails du compte

Ma wishlist

Déconnexion

TABLEAU DE BORD

Ma Commande Adresses

Détails du compte Ma wishlist

Activités récentes

Frame 31

mentions légales Livraisons Contact Newsletter

Version desktop

Frame 30

FOOT EN STOCK/ CULTURE FOOT

recherche produit

compte Déconnexion

Equipes nationales Ligue 1 premier league Bundesliga liga série A Panier

NAVIGATION

Ma Commande

Adresses

Détails du compte

Ma wishlist

Déconnexion

TABLEAU DE BORD

Ma Commande Adresses

Détails du compte Ma wishlist

Activités récentes

Frame 31

mentions légales Livraisons 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:

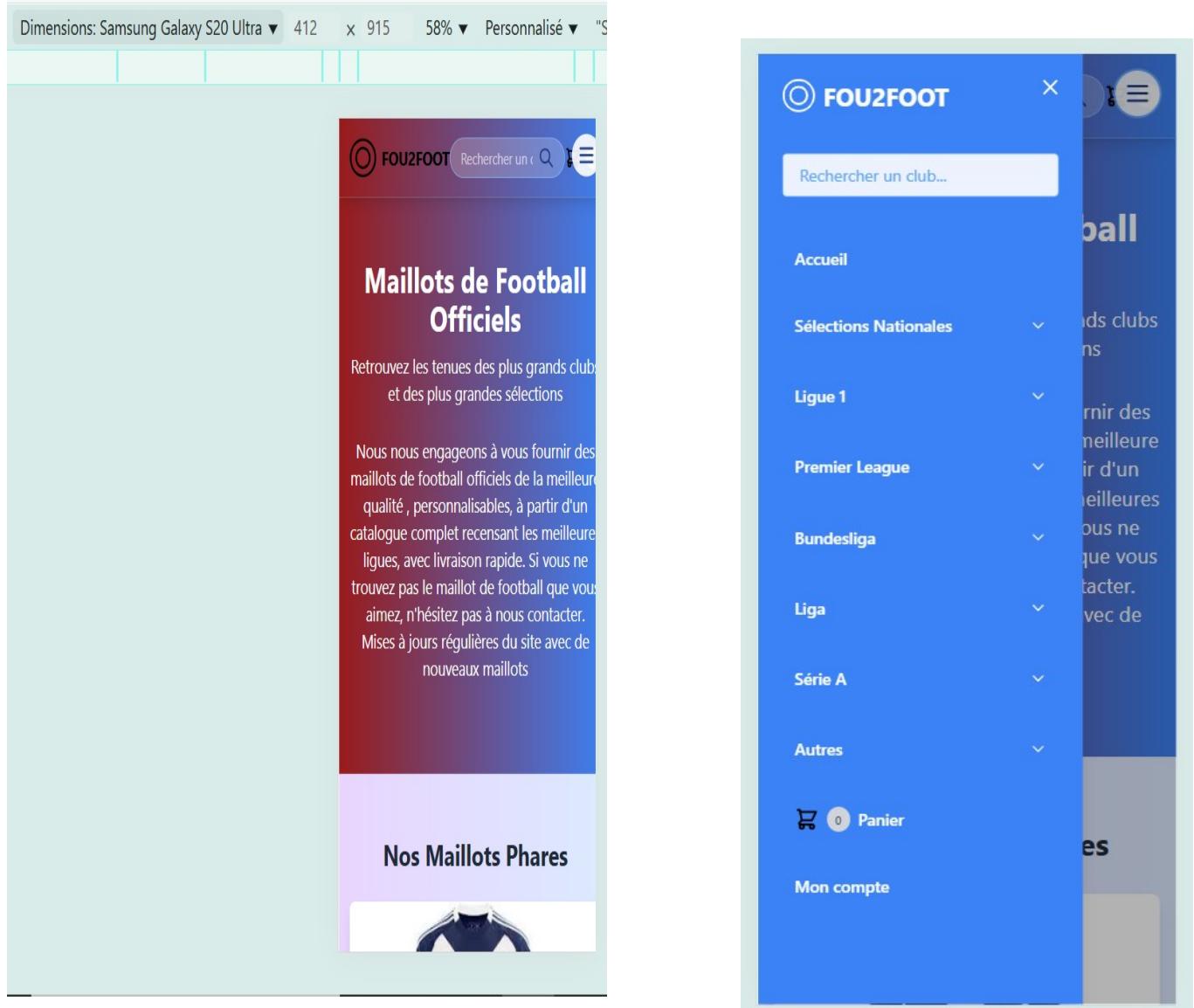
- Header:** A purple header bar with the text "Frame 28" and "recherche produit" on the left, and "FOOT EN STOCK/ CULTURE FOOT", "compte", and "Déconnexion" on the right.
- Navigation:** A green navigation bar below the header containing links for "Equipes nationales", "Ligue 1", "premier league", "Bundeliga", "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

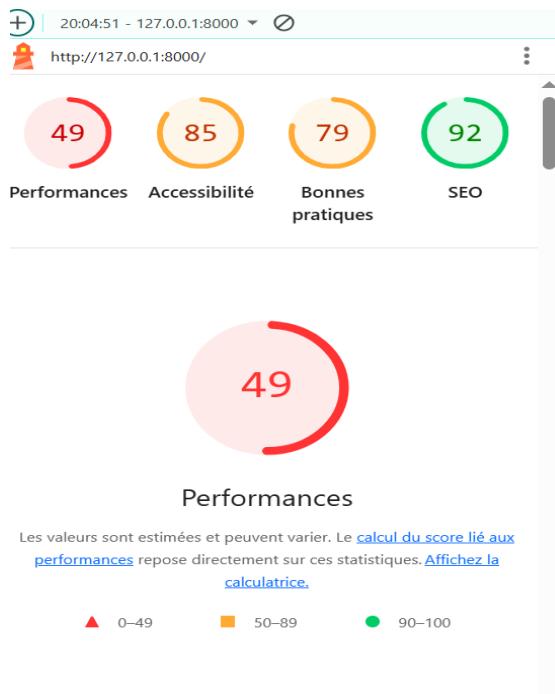
et facilité l'étape suivante du maquettage que l'on doit théoriquement pouvoir présenter au client afin que celui-ci puisse la valider.

Design responsive avec menu burger



*Autres pages responsives voir annexes

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).

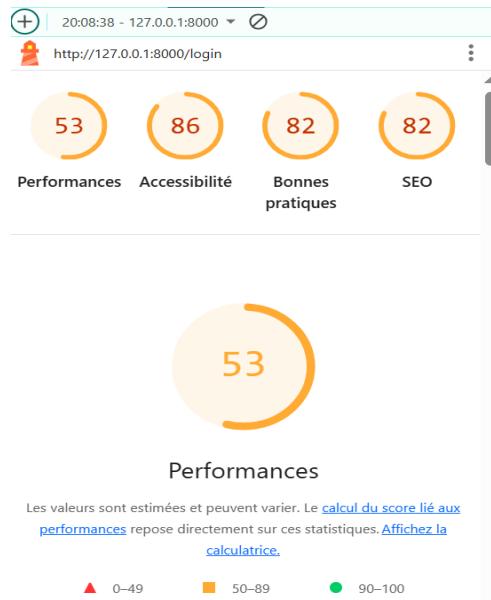
Bonnes pratiques (79)

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.

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

Relations et contraintes

Origine	Cible	Foreign Key	Cardinalité
user_addresses	users	user_id → users.id	n,1
carts	users	user_id → users.id	1,1
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

c) Conception et organisation des données

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
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	21,0 kio	10,0 kio
users	Parcourir Structure Rechercher Insérer Vider Supprimer	18	MyISAM	utf8mb4_unicode_ci	18,5 kio	20
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 kio 10,5 kio

Table users

The screenshot shows the 'users' table in the 'maillot' database. The table has 18 rows of data. The columns are: id, username, email, password, first_name, and last_name. The data includes various user entries such as Albundy, prasanna, lolo, Sophie69, marion, magnum, rais, etc.

	id	username	email	password	first_name	last_name
1	1	Albundy	albundy@gmail.com	\$2y\$12\$UvN5QMkelWZ9jKhn5codO.qKMFCFvNc8KwPsMjnRZ...	AL	BU
2	2	prasanna	prasann@yahoo.in	\$2y\$12\$02AMY26P67ahq57aqjSkXeWHCfnoEdhRbse3QsN.wA...	Prasanna	Dai
3	3	Lolo	lolo@gmail.com	\$2y\$12\$BKwgXMX9dDzQOKZH1nxw.uq24jq8BfQggGMKFnivcxr...	Lolo	Rig
4	4	Sophie69	sophie@gmail.com	\$2y\$12\$U6nJRH09y.TXoE3rdgeD6e6.M400Mm5GDogmpkQw...	NULL	NU
5	5	marion	marion@hotmail.com	\$2y\$12\$desYkbbsYL9MI/1r0YB6eu97Yzd7.LdCmhXj69F4W...	Marion	LAI
6	6	magnum	magnum@yahoo.us	\$2y\$12\$JGfAGqsS.cuglCQviXDLfOraM.waSjCidCnMP6.xsyX...	thomas	Ma
7	7	rais	rais@yahoo.fr	\$2y\$12\$S5cQ9VHc7dG.A0maBwL..WaUcnDClou/qFTm0qnZ5...	Rais	Pa

Structure table users

The screenshot shows the structure of the 'users' table. It contains 14 columns:

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

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

A screenshot of the phpMyAdmin interface for the 'cart_items' table in the 'maillot' database. The table has 59 rows. The columns are: id, cart_id, maillot_id, size, quantity, created_at, updated_at, numero, and nom. The data includes various items such as ZORRO, LULOL, LUBLEU, ULYSSE, AL BUNDY, BUNDY FRANCE, AL FRANCE, AL OL, Marion OL, and Zorro le renard.

	id	cart_id	maillot_id	size	quantity	created_at	updated_at	numero	nom
1	138	1	188	XL	3	2025-07-22 11:59:49	2025-07-25 11:17:50	33	ZORRO
2	114	2	185	S	1	2025-07-20 02:22:00	2025-09-27 00:46:56	NULL	NULL
3	7	4	177	XL	2	2025-07-15 07:59:45	2025-07-25 09:19:18	NULL	LULOL
4	8	4	188	XL	3	2025-07-15 08:35:53	2025-07-25 09:19:29	NULL	LUBLEU
5	140	17	2	XL	3	2025-07-22 13:33:39	2025-07-22 13:55:37	31	ULYSSE
6	153	3	389	XL	3	2025-07-26 00:32:44	2025-07-26 01:19:55	88	AL BUNDY
7	154	3	1	XL	1	2025-07-26 00:42:28	2025-07-26 00:42:28	97	BUNDY FRANCE
8	19	3	188	XL	3	2025-07-15 09:40:00	2025-07-26 01:20:06	0	AL FRANCE
9	20	3	36	XL	1	2025-07-15 09:41:37	2025-07-26 01:20:47	99	AL OL
10	129	7	36	S	3	2025-07-21 12:05:35	2025-07-21 13:26:54	11	Marion OL
11	146	1	371	XL	2	2025-07-25 10:09:09	2025-07-25 10:09:09	00	Zorro le renard

Maillots

A screenshot of the phpMyAdmin interface for the 'maillots' table in the 'maillot' database. The table has 214 rows. The columns are: id, club_id, nom, image, created_at, updated_at, and price. The data includes various items such as Domicile 2024, Extérieur 2024, Domicile 2024, Extérieur 2025, Domicile 2024, Extérieur 2024, Domicile 2024, Extérieur 2024, Domicile 2024, and Extérieur.

	id	club_id	nom	image	created_at	updated_at	price
1	1	1	Domicile 2024	images/maillot/images_maillot/france.jpg	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
2	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
3	3	2	Domicile 2024	images/maillot/images_maillot/bresil.jfif	2025-07-11 00:51:44	2025-07-11 00:51:44	20.00
4	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
5	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
6	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
7	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
8			Extérieur		2025-07-11	2025-07-11	

User_Addresses

The screenshot shows the phpMyAdmin interface for a MySQL database named 'mailot'. The left sidebar lists various tables: 'fou2foot', 'mailot' (selected), 'Nouvelle table', 'cart', 'cart_items', 'clubs', 'maillots', 'migrations', 'sessions', 'users', and 'user_addresses'. The main area displays the 'user_addresses' table with the following data:

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

[Connexion](#) [Inscription](#)

Nom d'utilisateur ou Email

testuser

Identifiants incorrects ou compte inactif.

Mot de passe

.....|



[Se connecter](#)