

DOSSIER DE PROJET



CRÉATION D'UN SITE INTERNET

Nom du projet : Stars of football

Fait par : Cyril Valette

Fait pour : Le titre de développeur Web et Mobile

SOMMAIRE

INTRODUCTION :	3
Contexte :	3
Présentation du projet :	3
Technologies utilisées :	4
Compétence du référentiel couvertes :	4
CAHIER DES CHARGES :	6
Objectif :	6
Client cible :	6
Type d'application :	6
Spécification technique du projet :	7
Contrainte :	7
Réalisation :	8
Création de la maquette :	8
Base de données :	10
Composant Livewire :	13
Page tactique du club :	15
LES PROPRIÉTÉS :	15
LES PROPRIÉTÉS CALCULÉES :	15
LES ACTIONS :	Error! Bookmark not defined.
FLASH MESSAGE :	16
VALIDATION EN TEMPS RÉEL :	18
VEILLE :	30
Sécurité :	30
RECHERCHE ET TRADUCTION :	31
ANNEXE :	34

INTRODUCTION :

Contexte :

La formation « Développeur web et web mobile » que propose le Callac Soft Collège se déroule sur un an. Elle est divisée en deux parties de six mois chacune, une première théorique et une seconde de production pendant laquelle j'ai réalisé ce projet.

Présentation du projet :

Stars of football est un jeu sur navigateur qui permet de créer son club et le développer et gérer pour qu'il devienne le meilleur.

L'utilisateur pourra donner le nom de son club ainsi que celui du stade qu'il souhaite, il pourra aussi développer son club via l'amélioration des bâtiments. Il devra aussi garder les joueurs de son club en forme grâce aux entraînements. Il pourra acheter et vendre des joueurs pour améliorer son club mais devra faire attention à son argent. Il commencera dans la ligue la plus basse et devra au fil des saisons monter jusqu'au sommet.

L'utilisateur pourra avoir accès à un récapitulatif de ses derniers mois niveau finance ainsi que sa dernière dépense, le montant et le descriptif. Il pourra avoir la liste de ses joueurs de son club ainsi que leurs statistiques. Il aura accès à sa feuille de tactique et donc de choisir ses titulaire et le dispositif de son équipe ainsi que la mentalité de l'équipe : offensive, défensive, équilibré.

Pendant les matchs entre club se feront avec les règles originales du foot.

Technologies utilisées :

BACKEND :

PHP via le framework Laravel

Livewire

MySQL

Versionnage GIT

FRONT :

HTML5

CSS 3 Tailwind

JavaScript Livewire, AlpineJS, ChartJS

CONCEPTION :

Visual studio code

Figma

JMerise

Versionnage GIT

Compétence du référentiel couvertes :

FRONT :

Maquetter une application :

Étant à l'origine de la conception de ce projet, j'ai réalisé ces maquettes pour concrétiser visuellement l'interface graphique pensée. Capture d'écran : **cf. Annexe**

Réaliser une interface utilisateur adaptable :

L'ensemble des écrans réalisés est compatible sur smartphone et sur ordinateur. J'ai eu recours aux variables de tailwind css : « MD : » pour adapter mes pages en fonction du support utilisé. Capture d'écran : **cf. Annexe**

Réaliser une interface utilisateur de gestion de contenu :

Mes pages de gestion du club sont dynamiques car le contenu est un composant Livewire ce qui me permet de la rendre dynamique en fonction des actions de l'utilisateur.

BACK :

Créer une base de données :

Pour ce projet, j'ai créé une base de données afin de pouvoir gérer les utilisateurs, les clubs, les joueurs, stades, centre d'entraînement, finances... Capture du MCD : **cf. Annexe**

Développer les composants d'accès aux données :

Au cours de ce projet, j'ai développé des composants en lien avec la base de données, notamment le CRUD permettant à la création de club, modification du dispositif, titulaire, mentalité et le transfert de joueurs. Pour cela j'ai utilisé l'ORM Eloquent qui est un objet-relational mapping (mapping objet-relationnel) qui permet de gérer la base de données et au travers des Model, chaque table aura un Model et cela permet donc des les manipuler au travers d'un objet.

Développer la partie back-end d'un application web :

Ayant utilisé le Framework Laravel pour développer cette application qui utilise le motif d'architecture logicielle MVC(Modèle-vue-contrôleur), j'ai donc dû procéder à l'installation de ce dernier ainsi qu'à la configuration des composants serveur tels que le routage et la création de mes Model et Controller.

Elaborer et mettre en œuvre des composants dans une application de gestion de contenu :

Ce projet incluant plusieurs mécanismes complexes tel que la gestion des 11 premiers joueurs titulaires et remplaçants dans l'ordre. Il m'était donc nécessaire de créer des composants me permettant de gérer tout cela.

CAHIER DES CHARGES :

Objectif :

Créer un site où les personnes peuvent avoir leur propre club et joueurs et le développer petit à petit jusqu'à devenir le meilleur et aussi affronter pleins d'autres joueurs.

Le but serait que les personnes viennent quotidiennement sur le site pour s'amuser et gérer toutes les tâches qu'il faut faire.

Client cible :

Je vise un public qui aime le foot en premier lieu car ce seront les plus gros joueurs mais aussi les personnes qui aiment les jeux de gestion et de stratégie.

Le site sera principalement destiné au Français, il pourra avoir une boutique en jeu par la suite, le site sera disponible en premier lieu pour un accès facile sur téléphone mais pourra ensuite évoluer en application mobile.

Le site exigera la création d'un compte, puis la création du club après sera obligatoire pour jouer.

Type d'application :

Jeu sur navigateur.

Spécification technique du projet :

Caractéristiques des joueurs :

ID , Prénom, Nom, ID Club, Poste, Vitesse , Dribble, Tir, Passe , Défense, Physique.

Menu de navigation (Accueil, stade, classement, effectif, centre des jeunes, finances).

Page de non connecté et page de connecté au site.

Gérer le club entier.

Système de transfert de joueurs.

Système de Ligue et récompense au classement.

L'algorithme du jeu et l'économie du club.

Faire travailler les stats des joueurs en match pour avoir une stratégie.

Système de match.

Système d'évolution des joueurs jusqu'à 30 ans.

Baisse du niveau après 30 ans.

Évolution des jeunes joueurs dans le centre des jeunes.

Contrainte :

Gérer toutes les structures qui seront rattachées au club.

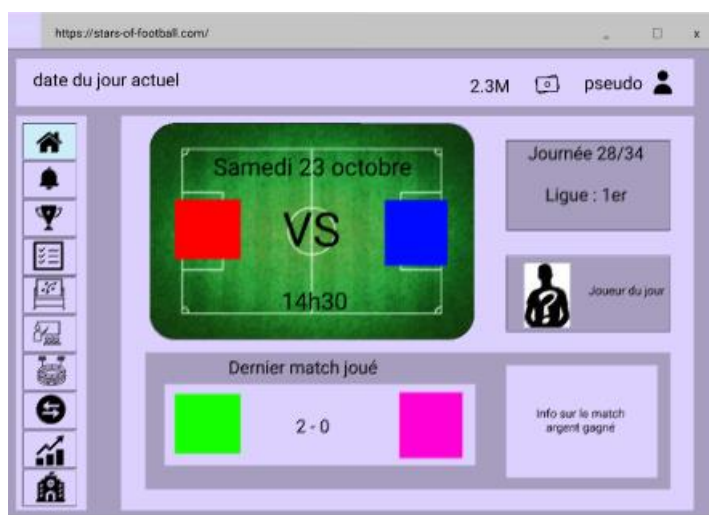
Mettre en place un système de tactique dans le club.

Avoir des bases solides comme tous les autres jeux de gestion et stratégie.

Réalisation :

Création de la maquette :

J'ai tout d'abord commencé mon projet en maquant les pages de mon application afin d'avoir une idée plus précise de ce à quoi allait ressembler mon site, Pour ce faire j'ai utilisé l'éditeur graphique Figma.



Mon idée était de faire une interface simple avec le plus d'info pratique mise en avant pour que l'utilisateur ne soit pas trop perdu non plus.

Cette page la est l'accueil quand l'utilisateur possède un club.

On peut par exemple retrouver l'argent que le club possède actuellement, son pseudo et les infos du match futur, ainsi que la journée actuelle dans la ligue et son classement puis un récapitulatif du dernier match joué.

Sur cette page qui est la page tactique on peut encore retrouver les infos pratiques comme l'argent du club et son pseudo. Mais la page reste simple car les 11 premiers joueurs sont les titulaires et aussi affichés sur le terrain donc on peut voir leurs postes.



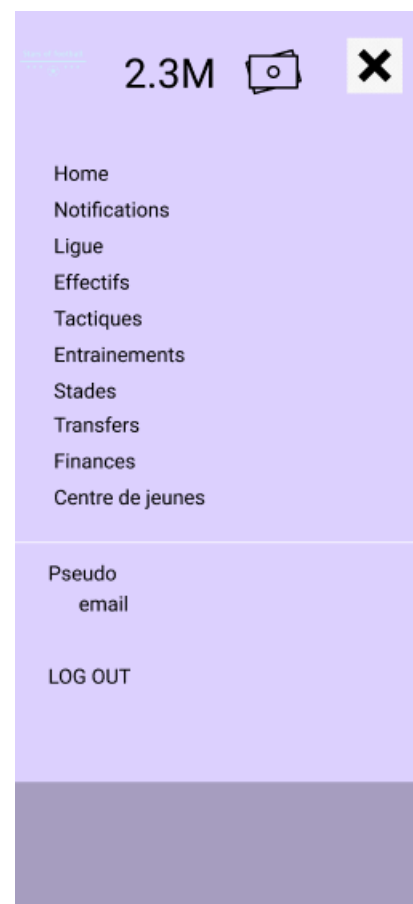
Version Mobile :



La version mobile du site sera une version très simplifiée avec des infos les plus pratiques possible mis à l'écran les infos seront mises à la suite de haut en bas pour que l'utilisateur ne soit pas submergé et qu'il a temps de descendre dans la page si besoin.

La navigation entre les pages sur la version mobile passe par un hamburger qu'on peut retrouver en haut à droite de la page dans la barre d'info.

Lorsque l'utilisateur ouvre le hamburger on peut voir la navigation entre les pages du site ainsi que des infos sur le compte de l'utilisateur et s'il veut se déconnecter ou non.

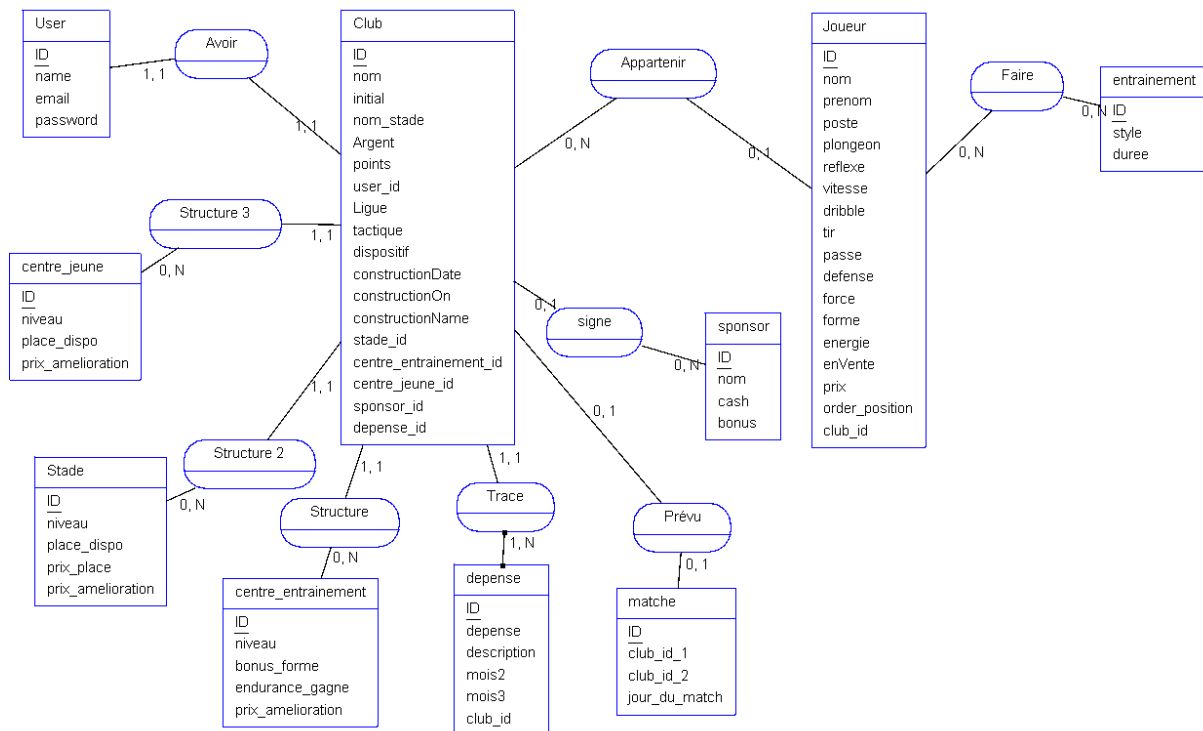


Base de données :

Pour la création de ma base de données j'ai commencé par concevoir le MCD afin d'organiser au mieux ma base de données.

J'ai utilisé le logiciel JMerise qui me permet de faire ce type de schéma qui peut être très utile pour que par la suite on ne se perde pas car un à un plan précis.

Pour ce schéma on peut regarder la relation entre le Club et le Joueur, un club peut très bien posséder plusieurs joueurs d'où le choix de : **0, N**. Mais un joueur quant à lui peut très bien ne pas avoir de club ou alors appartenir qu'à UN seul club d'où le choix de : **0, 1**.



Ma base de données tourne autour de trois tables : user, club et joueur.

Un **user** va créer son **club** en le nommant et donnant un nom au stade puis ces initiales, ensuite son **club** va obtenir des **joueurs** de départ généré aléatoirement et avoir un **centre de jeune**, **stade** et **centre d'entraînement**. Puis cela va créer la traçabilité de ses **dépenses**. Puis il pourra ensuite faire des **entraînements** et améliorer ses structures grâce à son argent de départ. Il aura l'occasion au début de la saison de signer chez un **sponsor** et d'ensuite faire des **matches**.

APPLICATION DE LA BASE DE DONNÉES DANS LARAVEL :

Les migrations dans Laravel permettent de facilement créer et modifier ces schémas de base de données. On peut aussi déployer facilement la base de données sur un nouveau serveur.

Pour créer une migration avec Laravel, on utilise la commande Artisan suivante :

```
-php artisan make:migration create_users_table
```

Les migrations sont comme un contrôle de version pour votre base de données, permettant de définir la définition du schéma de base de données de l'application. Elle possède une fonction **up** qui permet de créer la table et une fonction **down** qui permet de la supprimer si la table existe déjà.

Laravel met en place une convention de nommage qui lui permet d'effectuer un lien directement entre la table et le model (composant d'accès aux données) associé en nommant la table comme le model mais en ajoutant un **s** à la fin du nom.

Un Model est une classe qui va permettre de manipuler la base de données en entrant des valeurs, en supprimant...

Le Model est une classe ce qui nous permet de faire cela en manipulant via un objet.

En connaissant cette convention de nommage et parcourant la doc de Laravel j'ai pu trouver une meilleure façon de mettre en place la base de données par Laravel. Comme je savais quel Model j'allais avoir dans mon projet j'ai utilisé cette commande Artisan :

```
-php artisan make:model Club -m
```

En utilisant ce petit drapeau **-m** je demande à Laravel de créer la migration correspondant au model donc grâce à cela j'obtiens le Model et la migration.

```

public function up()
{
    Schema::create('clubs', function (Blueprint $table) {
        $table->id();
        $table->String('nom',40);
        $table->String('initial',4);
        $table->String('nom_stade',40);
        $table->bigInteger('Argent')->default(50);
        $table->bigInteger('Points')->default(0);
        $table->foreignId('user_id')->constrained();
        $table->enum('Ligue', ['Bronze', 'Argent', 'Or'])->
>default('Bronze');
        $table->enum('tactique',['défensif','équilibré','offensif'])->
>default('équilibré');
        $table->enum('dispositif',['4-4-2','4-3-3','4-5-1'])->default('4-
4-2');

        $table->dateTime('constructionDate')->nullable();
        $table->boolean('constructionOn')->default('0');
        $table->string('constructionName',40)->nullable();
        $table->foreignId('stade_id')->default(1)->constrained();
        $table->foreignId('centre_entrainement_id')->default(1)-
>constrained();
        $table->foreignId('centre_jeune_id')->default(1)->constrained();
        $table->foreignId('sponsor_id')->nullable()->constrained();
        $table->foreignId('depense_id')->nullable()->constrained();
        $table->timestamps();
    });
}

```

Ci-dessus j'ai pris exemple de la migration clubs. Dans cette migration les colonnes sont indiquées via « \$table-> » ensuite j'indique par une méthode de la classe **Blueprint** le type de donnée, par exemple pour la colonne qui contiendra le **user_id** je précise le type de donnée « foreignId ». Je précise ensuite le nom de la colonne « ('user_id') ». Après cela, je définis des paramètres supplémentaires comme « constrained » qui applique à la colonne les contraintes d'une clef étrangère. Il faut bien mettre dans le nom de la colonne pour une clef étrangère le nom du modèle « user » suivi de la clef « id » pour que le lien entre les deux soit bien fait.

Composant Livewire :

LIVEWIRE, COMPOSANT DE LA LIBRAIRIE DE LARAVEL :

Livewire est une librairie du framework PHP Laravel destiné au développement web. Pour rappel, le framework PHP Laravel est une plateforme d'outils de développement web qui vous proposeront des architectures en fonction de vos besoins afin de pouvoir y attacher votre code tout en le personnalisant.

Le but de Livewire est d'augmenter la rapidité du développement web tout en le rendant plus fluide. Livewire simplifie la tâche de développement web avec un design minimaliste pour pouvoir se concentrer sur le plus important et vous fera gagner beaucoup de temps.

Livewire vous permet d'exécuter des interrogations de votre base de données en temps réel, mais aussi de créer des composants réactifs avec Laravel sans même à avoir à écrire des lignes de code avec Javascript.

Le temps de réponse en recherche sera également grandement diminué et propose même des recherches automatiques simples et rapides à effectuer.

INSTALLATION :

Livewire est un package pour Laravel qui s'installe classiquement avec composer :

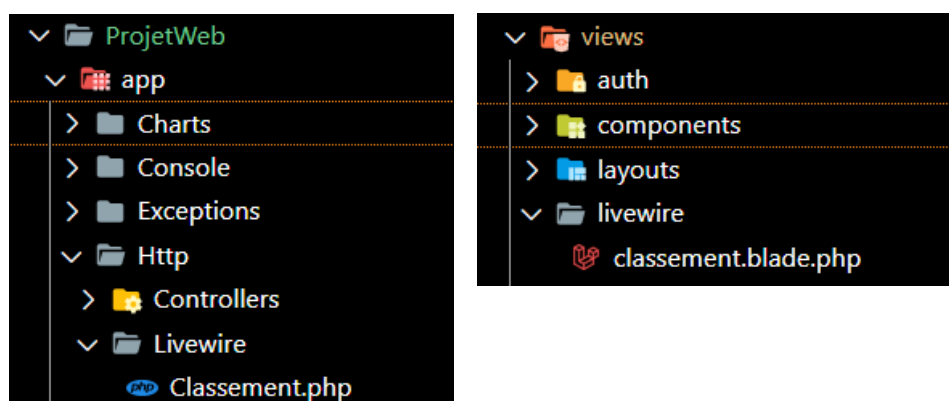
-composer require livewire/livewire

CRÉER UN COMPOSANT :

Livewire fonctionne avec des composants. Pour en créer un c'est tout simple :

-php artisan make:livewire Classement

On se retrouve avec deux fichiers créés (et deux dossiers qui n'existaient pas auparavant) :



Le premier comporte intendance (la classe) qui fonctionne comme un Controller possédant du coup une fonction `render()` et le second s'occupe de l'aspect (la vue).

Un Controller est une classe qui permet principalement de Controller la vue elle se place entre la vue et la Route elle permet de faire des fonctions en lien avec la vue et avoir un fichier de route simple.

Voyons le code généré dans la classe :

```
<?php

namespace App\Http\Livewire;

use Livewire\Component;

class Classement extends Component
{
    public $clubuser;

    public function render()
    {
        return view('livewire.classement');
    }
}
```

On a une méthode **render** et on retourne la vue générée.

Dans la vue on a :

```
<div>
    {{-- If you look to others for fulfillment, you will never truly be
    fulfilled. --}}
</div>
```

Maintenant qu'on sait créer un composant voyons comment l'utiliser. Il y a deux façons de le faire :

-dans une vue avec une nouvelle balise blade :

```
<livewire:classement :clubuser="$clubuser" />
```

Cela permet de mettre le composant dans la vue comme une balise et ensuite on peut lui passer une valeur.

-dans une vue complète, dans ce cas le composant est la vue

Pour mon projet j'ai choisi la première option car la plus facile au vu de l'avancement de mon projet quand j'ai installé Livewire dessus.

Pour que les composants Livewire fonctionne il faut que dans le Layouts il y a une insertion des scripts de Livewire : **@livewireScripts**

Page tactique du club :

Voyons un peu comment j'ai appliqué les fonctionnalités de Livewire dans ma page tactique du club.

LES PROPRIÉTÉS :

Les composants de Livewire ont besoin de données. On peut créer des propriétés dans la classe :

```
public $poste1 = ['', 'GC', 'DD', 'DC', 'DC', 'DG', 'MD', 'MC', 'MC', 'MG',  
'BU', 'BU', 'REM1'];
```

Une propriété **public** dans la classe est automatiquement disponible dans la vue :

```
<th scope="row">{{ $poste1[$loop->iteration] }}</th>
```

Là je parcours et affiche la propriété **\$poste1** en fonction de ma boucle et de son itération.

LES PROPRIÉTÉS CALCULÉES :

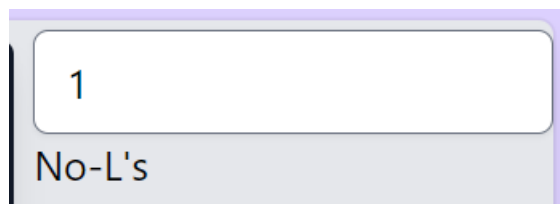
On peut aussi avoir des propriétés calculées (comme avec Vue.js), alors là évidemment ça devient plus intéressant parce qu'on a besoin d'information sur le serveur. Imaginons que nous avons deux utilisateurs dans notre table **users** (il vous suffit d'utiliser le formulaire d'enregistrement). Alors on peut créer cette propriété calculée (computed) :

```
class Tactique extends Component  
{  
  
    public $poste1 = ['', 'GC', 'DD', 'DC', 'DC', 'DG', 'MD', 'MC', 'MC', 'MG',  
'BU', 'BU', 'REM1'] ;  
  
    public $index = 1;  
  
    public function getUserProperty()  
    {  
        return User::find($this->index);  
    }  
}
```

Et dans la vue :

```
<div>
  {{-- If you look to others for fulfillment, you will never truly be
fulfilled. --}}
  <input wire:model="index" class="form-input rounded-md shadow-sm block mt-
1 w-full" type="text" />
  <p>{{ $this->user->name }}</p>
</div>
```

Maintenant quand on entre l'index dans la zone de texte le nom de l'utilisateur correspondant apparaît au-dessous :

A screenshot of a web form. It features a white text input field with rounded corners and a light gray border, containing the number '1'. Below the input field is a light gray rectangular box containing the text 'No-L's' in a dark gray font.

FLASH MESSAGE :

Voyons un peu comment peut-on mettre en pratique dans mon projet la fonctionnalité d'action.

Une action dans Livewire c'est la capacité à écouter une action sur la page web et d'appeler une méthode dans le composant pour modifier la page. C'est donc un pas de plus dans l'interactivité par rapport aux propriétés calculées vues ci-dessus.

Ici j'ai un composant Livewire Flash avec plusieurs particularités. Elle possède plusieurs valeurs qui sont **\$message** et **\$type** et **\$styleByType**.

```
class Flash extends Component
{
  public $message;
  public $type;

  public $styleByType = [
    'success' => 'bg-green-100 border-green-700 text-green-700',
    'info' => 'bg-blue-100 border-blue-700 text-blue-700',
    'warning' => 'bg-yellow-100 border-yellow-700 text-yellow-700',
    'error' => 'bg-red-100 border-red-700 text-red-700',
  ];
}
```



```

public $listeners = ['flash' => 'setFlashMessage'];

public function setFlashMessage($message, $type)
{
    $this->message = $message;
    $this->type = $type;

    $this->dispatchBrowserEvent('flash-message');
}

public function render()
{
    return view('livewire.flash');
}
}

```

Ici on écoute une action d'une page web via **\$listeners** et la valeur flash qui est rattachée à une méthode **setFlashMessage**.

La façon de déclencher la méthode en l'appelant est comme ce ci :

```

$this->emit('flash', 'Tu n\'a pas l\'argent necessaire pour acheter ce joueur
!', 'error');

```

On émit un événement sur la page nommée **flash** où y est ajouté le **\$message** puis le **\$type**.

Quand la méthode est appelée elle va ensuite récupérer les **\$message** puis le **\$type** et lancer un **dispatchBrowserEvent** qui est un événement sur la page permettant de l'écouter en Javascript et ici on va le faire avec AlpineJS qui est dans Laravel grâce à Livewire ou Breeze.

```

<div x-data="{ open: false }"
    @flash-message.window="open = true; window.scrollTo(0, 0); setTimeout(()
=> open = false, 7000);">

    <div x-show.transition.opacity="open" x-cloak class="flex {{$type ?
$styleByType[$type] : ''}} p-5 border rounded m-3">
        <svg class="h-6 w-6 mr-1" fill="none" viewBox="0 0 24 24"
stroke="currentColor">
            <path stroke-linecap="round" stroke-linejoin="round" stroke-
width="1"
                d="M15 17h5l-1.405-1.405A2.032 2.032 0 0118 14.158V11a6.002
6.002 0 00-4-5.659V5a2 2 0 10-4 0v.341C7.67 6.165 6 8.388 6 11v3.159c0 .538-
.214 1.055-.595 1.436L4 17h5m6 0v1a3 3 0 11-6 0v-1m6 0H9" />
        </svg>
        <p class="text-lg">{{ $message }}</p>

    </div>
</div>

```

Ici on écoute l'événement nommé **@flash-message.window** donc **flash-message** envoyé par le **dispatchBrowserEvent**.

On donne une valeur **x-data='{ open : false }'** qui est pour l'affichage et quand flash-message est appelé on passe à **'open=true'**.

Le Flash message est un composant qui est très utilisé et assez dynamique, je me suis tourné vers un composant comme celui-là pour que lorsque l'utilisateur fait une action qui n'est pas possible ou qui pose problème et donc lui transmettre une information.

VALIDATION EN TEMPS RÉEL :

Livewire va plus loin en permettant une validation en temps réel lors de la saisie de la valeur sans attendre la soumission.

On choisit la propriété qu'on veut valider avec **validateOnly** (sinon on validerait toutes les valeurs).

Maintenant quand on entre une valeur on aura la validation instantanément sans attendre la soumission.

Validation en temps réel pour l'achat des joueurs :

```
<form wire:submit.prevent="achat({{ $joueur }})">
    <a class="
        href="{{ route('transfert.joueur') }}">

        <div>
            {{ $joueur->nom }}
        </div>
        <div>
            {{ $joueur->age }}
        </div>
        <div>
            {{ $joueur->poste }}
        </div>
        <div>
            {{ $joueur->prix }}M
        </div>
    </a>
    <button
        class=" "
        type="submit">
        Acheter
    </button>
</form>
```

Ici dans la page je génère autant de form qu'il y a de joueurs en vente donc je passe la valeur **\$joueur** via le form dans ma fonction.

Donc ensuite dans mon composant Marche je peux récupérer le joueur en question et ainsi lorsque je vais Validate je vérifie qu'il n'y a pas de joueur ajouté.

```
class Marche extends Component
{
    public $joueur;

    public function achat($joueur)
    {
        $clubuser = Club::where('user_id', Auth::id())->first();
        $clubJoueur = Club::where('id', $joueur['club_id'])->first();
        $joueurSell = Joueur::where('id', $joueur['id'])->first();

        if ($clubuser->id != $clubJoueur->id) {
            if ($clubuser->Argent >= $joueurSell->prix) {
                $data = $this->validate(['joueur' => '']);
                $data['joueur'] = $joueur;
            }
        }
    }
}
```

Je récupère donc la valeur passé \$joueur et ensuite au début de ma fonction je récupère le club qui achète le joueur **\$clubuser** et le club du joueur vendu **\$clubJoueur** et aussi le joueur qui ai vendu est reçu sous forme d'array que grâce à Eloquent je récupère ensuite en objet **\$joueurSell**.

Ensuite je vérifie si le club qui achète ne possède déjà pas le joueur puis s'il a assez d'argent pour l'acheter. Et je vais donc faire une **Validate** qui regarde bien si un joueur en plus n'a pas été rajouté dans le form autre que celui actuel.

OPTIMISATION DES DONNÉES TRANSMISE DANS LE COMPOSANT :

Plus j'ai avancé dans le développement du composant Livewire je me suis rendu compte que la façon de passer des données depuis la balise rendait le composant moins dynamique alors je suis allé sur la doc de Livewire pour chercher un moyen de le rendre plus dynamique.

A Livewire component's render method gets called on the initial page load AND every subsequent component update.

The render() method is expected to return a Blade view, therefore, you can compare it to writing a controller method.

Ce que la doc veut dire c'est que dans un composant Livewire la méthode render est appelé au premier chargement et à chaque changement dans le composant par exemple un changement dans la base de données.

La méthode render() est attendu à ce qu'elle retourne une view Blade mais on peut la comparer à une méthode dans un controller.

Après avoir lu cela j'ai fait comme dans la méthode d'un controller et j'ai donc passé les propriétés au rendu :

```
public function render()
{
    $clubuser = Auth::user()->clubUser;
    $joueurs = Joueur::where('club_id', $clubuser->id)
        ->where('enVente', '0')
        ->orderBy('order_position', 'asc')
        ->get();

    $joueur1 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',1)->first();
    $joueur2 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',2)->first();
    $joueur3 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',3)->first();
    $joueur4 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',4)->first();
    $joueur5 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',5)->first();
    $joueur6 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',6)->first();
    $joueur7 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',7)->first();
    $joueur8 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',8)->first();
    $joueur9 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',9)->first();
    $joueur10 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',10)->first();
    $joueur11 = Joueur::where('club_id', $clubuser->id)-
>where('order_position',11)->first();

    return view('livewire.tactique',
compact('clubuser','joueurs','joueur1','joueur2','joueur3','joueur4','joueur5'
,'joueur6','joueur7','joueur8','joueur9','joueur10','joueur11'));
}
```

MISE EN PLACE DES TITULAIRES DE FAÇON DYNAMIQUE :

Pour mettre en place dans la page tactique et mettre en place les titulaires j'ai voulu mettre un système de drag and drop pour que ce soit plus facile pour l'utilisateur, j'ai donc cherché sur la documentation de laravel s'il y avait un moyen d'émettre cela en place mais je n'ai pas trouvé et en cherchant plus loin j'ai trouvé un plug-in Livewire fait par la communauté qui se nomme : Livewire sortable.

J'ai donc rajouté une balise de script à la fin de ma page :

```
<script  
src="https://cdn.jsdelivr.net/gh/livewire/sortable@v0.x.x/dist/livewire-  
sortable.js" defer></script>
```

Cette balise me permet d'avoir accès au script de Livewire sortable et toutes ces fonctionnalités.

Une fois le script mis en place j'obtiens plusieurs nouvelles valeurs très importantes :

```
<tbody wire:sortable="updateJoueurOrder">  
    @foreach ($joueurs as $joueur)
```

Cette première valeur permet de relier avec une fonction les items. Cela met les limites du drag and drop et donc tout ce qui se trouve dans la balise peut être un item.

Ces deux prochaines me permette d'obtenir par la suite de nouvelles données qui vont me servir ensuite dans la fonction après :

```
<tr wire:sortable.item="{{ $joueur->id }}" wire:key="joueur-{{ $joueur->id }}"
```

J'utilise ses deux valeurs pour manipuler mes items qui sont les joueurs donc en premier lieu j'attache l'item à l'id du joueur puis je lui donne une clef qui est 'joueur-\$joueur->id' cela me permettra d'identifier par la suite les joueurs.

Ensuite je mets en place sur la balise le fait qu'elle puisse être attrapé :

```
wire:sortable.handle style="width: 10px; cursor: move;" class="hover:shadow-  
md">
```

Ensuite j'obtiens une autre valeur qui est quand je tiens l'item et donc je peux lui ajouter un style css lorsque c'est le cas. Je lui ajoute donc une taille précise et le fait que le curseur change.

Ensuite mon tableau qui liste tous mes joueurs de l'équipe se dispose comme ce ci :

```
<tbody wire:sortable="updateJoueurOrder">
    @foreach ($joueurs as $joueur)
        <tr wire:sortable.item="{{ $joueur->id }}"
wire:key="joueur-{{ $joueur->id }}"
            wire:sortable.handle style="width: 10px;
cursor: move;" class="hover:shadow-md">

                <th scope="row">{{ $poste3[$loop-
>iteration] }}</th>
                <th scope="row">{{ $loop->iteration
}}</th>
                <td><a class="hover:underline" href="{{
route('joueur', $joueur) }}">
                    {{ $joueur->nom }} {{ $joueur->
>prenom }}
                    </a></td>
                <td>{{ $joueur->poste }}</td>
                <td>{{ $joueur->energie }}</td>
                <td class="invisible hidden md:visible
md:table-cell">{{ $joueur->forme }}</td>

            </tr>
        @endforeach
    </tbody>
```

Ceci est le body de mon tableau, comme vu avant j'ai attaché les valeurs de Livewire sortable à plusieurs endroits et pour les joueurs dans ma page blade je reçois la totalité des joueurs du club sous \$joueurs et donc je fais une boucle qui parcourt tous les joueurs un par un ce qui donne \$joueur et donc pour chaque joueur j'aurai une ligne dans mon tableau déplaçable.

Maintenant que j'ai tout cela en place je me tourne vers la méthode qui va me permettre de rendre tout cela dynamique et toujours à jours qui se trouve dans le composant Livewire et cette méthode reçoit une valeur à chaque fois que le drag and drop est utilisé qui est : **\$items**

```
public function updateJoueurOrder($items)
{
    dd($items);
}
```

Avant d'essayer de voir si cela marche je dois avant tout rajouter une colonne dans ma base de données des joueurs qui est order position.

```
$table->bigInteger('order_position')->default(0);
```

Maintenant que j'ai tout cela en place je peux aller regarder ce que je reçois dans ma méthode.

```
array:15 [▼
  0 => array:2 [▼
    "order" => 1
    "value" => "1"
  ]
  1 => array:2 [▼
    "order" => 2
    "value" => "4"
  ]
  2 => array:2 [▶]
  3 => array:2 [▶]
  4 => array:2 [▶]
  5 => array:2 [▶]
  6 => array:2 [▶]
  7 => array:2 [▶]
  8 => array:2 [▶]
  9 => array:2 [▶]
  10 => array:2 [▶]
  11 => array:2 [▶]
  12 => array:2 [▶]
  13 => array:2 [▶]
  14 => array:2 [▶]
]
```

A la réception j'obtiens un array du nombre de joueurs dans le club.

On peut remarquer deux valeurs **order** et **value**, la première est la position de l'item dans la liste actuelle et puis la seconde est l'id du joueur.

Je n'ai donc qu'à parcourir cet array et modifier l'order_position des joueurs que je trouve grâce à la **value** et ensuite je modifie leur order_position grâce à la **order**.

Ce qui donne cela par la suite :

```
public function updateJoueurOrder($items)
{
    foreach ($items as $item) {
        Joueur::where('id', $item['value'])
            ->update([
                'order_position' => $item['order']
            ]);
    }
}
```

Ici j'applique ce que j'ai dit je cherche les joueurs dont l'id est égale à la value et j'update dans la base de données l'order_position grâce à la valeur order.

Maintenant je n'ai plus qu'à trier les joueurs afficher de façon à ce qu'il soit affiché dans l'ordre ascendant de order_position.

```
$joueurs = Joueur::where('club_id', $clubuser->id)
    ->where('enVente', '0')
    ->orderBy('order_position', 'asc')
    ->get();
```

MISE EN PLACE DE L’AFFICHAGE DES JOUEURS TITULAIRES SUR LE TERRAIN DYNAMIQUEMENT :

Comme j’arrive à récupérer l’ordre des joueurs du premier au 11^{ème} je n’ai juste à récupérer les joueurs un par un et ensuite les placer sur le terrain et comme je souhaite que ce soit dynamique et que cela s’adapte grâce au drag and drop je les récupère pareillement que la liste de tous les joueurs du club, j’ai choisi de faire une requête pour chaque joueur via Eloquent car je voulais manipuler des objets plutôt que un array cela me permet donc de pouvoir afficher plus d’info sur le joueur :

```
$joueur1 = Joueur::where('club_id', $clubuser->id)->where('order_position',1)->first();
$joueur2 = Joueur::where('club_id', $clubuser->id)->where('order_position',2)->first();
$joueur3 = Joueur::where('club_id', $clubuser->id)->where('order_position',3)->first();
$joueur4 = Joueur::where('club_id', $clubuser->id)->where('order_position',4)->first();
$joueur5 = Joueur::where('club_id', $clubuser->id)->where('order_position',5)->first();
$joueur6 = Joueur::where('club_id', $clubuser->id)->where('order_position',6)->first();
$joueur7 = Joueur::where('club_id', $clubuser->id)->where('order_position',7)->first();
$joueur8 = Joueur::where('club_id', $clubuser->id)->where('order_position',8)->first();
$joueur9 = Joueur::where('club_id', $clubuser->id)->where('order_position',9)->first();
$joueur10 = Joueur::where('club_id', $clubuser->id)->where('order_position',10)->first();
$joueur11 = Joueur::where('club_id', $clubuser->id)->where('order_position',11)->first();
```

Ici dans mes requêtes je filtre chaque joueur en fonction de ça position dans l’effectif et donc les joueurs afficher sur la page et le terrain des titulaires ce met a jour constamment si la personne modifie l’ordre des titulaires avec le drag and drop.

Ce qu'y donne quelque chose comme cela en premier lieu :

```
<div class="z-1 col-span-3">
    <div class="flex justify-end">
        <div class="flex flex-col items-center">
            
            {{ $joueur11->nom }}
        </div>
    </div>
</div>
<div class="z-1 col-start-4 col-span-3">
    <div class="flex justify-start">
        <div class="flex flex-col items-center">
            
            {{ $joueur10->nom }}
        </div>
    </div>
</div>
<div class="z-1 col-start-1 col-span-2">
    <div class="flex justify-center">
        <div class="flex flex-col items-center">
            
            {{ $joueur9->nom }}
        </div>
    </div>
</div>
<div class="z-1 col-start-3">
    <div class="flex justify-center">
        <div class="flex flex-col items-center">
            
            {{ $joueur8->nom }}
        </div>
    </div>
</div>
</div>
```

DISPOSITIF AFFICHE SUR LE TERRAIN :

Ensuite j'utilise une fonctionnalité des vues Blade qui est que l'on peut utiliser du php dans la vue avec le html : @if - @endif, @foreach - @endforeach.

Pour le système de dispositif je vais utiliser les @if et @endif qui vont me permettre de mettre une condition du dispositif choisi pour l'affichage.

Comme un composant Livewire se rafraîchit à chaque fois que l'on fait un changement, cela va être très pratique de mettre en place des conditions d'affichage justement.

Je commence par ajouter un enum, un enum est une valeur qui peut uniquement avoir les valeurs prédéfinies à la création dans la base de données. Ensuite je mets donc plusieurs boutons pour changer justement le dispositif dans la base de données.

```
<button
    class=" "
    wire:click="dispositif1">
    4-4-2
</button>
<button
    class=" "
    wire:click="dispositif2">
    4-3-3
</button>
<button
    class=" "
    wire:click="dispositif3">
    4-5-1
</button>
```

Le **wire:click** de Livewire permet de relier une méthode à un bouton au click.

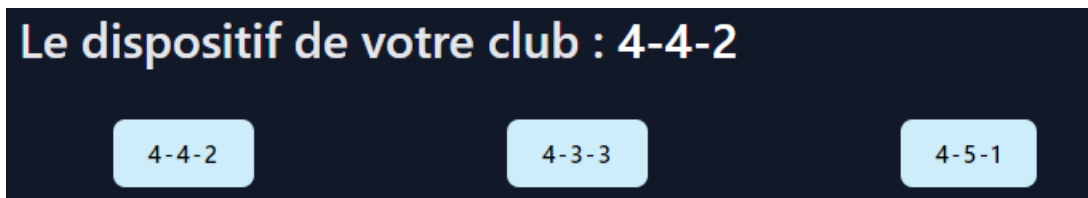
Par la suite, je construis mes méthodes qui sont simples pour changer le dispositif du club dans la base de données.

```
public function dispositif1(){
    Club::where('id', Auth::id())
    ->update([
        'dispositif' => '4-4-2',
    ]);
}
```

```
public function dispositif2(){
    Club::where('id', Auth::id())
    ->update([
        'dispositif' => '4-3-3',
    ]);
}
```

```
public function dispositif3(){
    Club::where('id', Auth::id())
    ->update([
        'dispositif' => '4-5-1',
    ]);
}
```

Maintenant que j'ai accès à plusieurs dispositifs dans le club dans la base de données et que les boutons modifient cela au click le composant se rafraîchit donc pour afficher le dispositif choisi.



Je mets en place donc les @if en fonction du dispositif du club pour que le dispositif avec les joueurs affiché corresponde au dispositif choisi par le club et que lorsque l'utilisateur change le dispositif la vue suivent.



```
{{-- dispositif 4-4-2 --}}
@if ($clubuser-
>dispositif === '4-4-2')
    <div class="z-1
col-span-3">
```



```
{{-- dispositif 4-3-3 --}}
@if ($clubuser-
>dispositif === '4-3-3')
    <div
class="z-1 col-start-1 col-span-2">
```

J'ai ensuite pris le même procédé pour ajouter un style de jeu au club de type offensif ou défensif ou équilibré.

La tactique de votre club : équilibré

OFFENSIF

DÉFENSIF

EQUILIBRÉ

FINALITÉ :

Avec toutes ces fonctionnalités de développ   j'ai atteint   la fin un rendu comme cela de la page :



La tactique de votre club : **equilibr  **

OFFENSIF DEFENSIF EQUILIBR  

Le dispositif de votre club : **4-3-3**

4-4-2 4-3-3 4-5-1

Liste de l'effectif du club :

!	#	NOM DU JOUEURS	POSTE	ENDURANCE	FORME
GC	1	Pagac Dan	GC	80	17
DD	2	Aufderhar Margaret	DD	100	10
DC	3	Hill Gennaro	DG	100	10
DC	4	Leuschke Michelle	DC	100	10
DG	5	Barrows Karon	DC	100	10
MC	6	Ledner Chanelle	MG	100	10
MC	7	Hegmann Nettie	MD	100	10
MC	8	Gutmann Juanita	MC	100	10
AG	9	Dickens Nathanael	MC	100	10
AD	10	Powlowski Jaden	MC	100	10
BU	11	Prohaska Lauriane	BU	100	10
REM1	12	Sanford Kayla	DC	100	10
REM2	13	Herzog Moshe	BU	100	10
REM3	14	Jack Jack	BU	10	10
REM4	15	Little Bernie	GC	100	10

VEILLE :

Sécurité :

Laravel intègre et gère pour nous beaucoup de points de sécurité.

CSRF :

La protection CSRF (Cross-Site Request Forgery) protège des actions non autorisées qui sont faites à l'encontre d'un utilisateur authentifié. Laravel va automatiquement générer un Token CSRF pour chaque utilisateur actif pour vérifier que c'est bien lui qui effectue la requête vers l'application. Dans chacun de mes formulaires j'ai ajouté un Token CSRF, avec la syntaxe Blade il suffit de faire @CSRF pour générer un input caché avec le Token.

```
<form method="POST" action="{{ route('Creation') }}">
    @csrf
```

LES INJECTIONS SQL :

Laravel nous protège contre les injections SQL, mis à part le cas où on rentre ou on exécute une requête dite «Raw» sur la base de données. Dans ce cas-là, c'est à nous de bien vérifier ce qui va être entré dans cette requête.

Grâce à Eloquent on obtient une protection supplémentaire car lorsque on passe par cela toute les requête passe donc par les Models mis en place par Eloquent qui sont relié à la base de données.

Le fait de passer par les Models créer donc un objet et un utilisateur ne peut donc pas modifier la base de donnée vu qu'il n'aura accès que à l'objet.

XSS ATTAQUE :

XSS Attack est une attaque sur l'affichage d'un résultat venant de la base de données par exemple une personne a réussi à écrire dans un input « <script>alert('XSS') </script> »et que l'on réaffiche ce résultat plus tard le script va être exécuté par le navigateur, pour éviter cela j'utilise la syntaxe Blade {{.....}} qui va échapper le tout et afficher le résultat en html.

RECHERCHE ET TRADUCTION :

COMPOSANT DYNAMIQUE :

Lorsque j'ai découvert Livewire et créer mes premiers composant pour mes différentes pages je passais mes valeurs dans la page grâce au controller ce qui n'aidait pas le composant à être dynamique vu qu'il ne refaisait pas les requêtes pour afficher ce qui avait changé vu que dans le page les valeurs n'avaient pas changé. Donc je suis allé voir sur la documentation de Livewire et la fonction `render()` et j'ai pu apprendre que elle pouvait embarquer des requêtes et les passer dans la page car c'est cette fonction qui est appeler à chaque changement dans la base de données.

Avec ce que j'ai appris sur la docs j'ai donc modifier ma méthodes `render()` :

```
public function render()
{
    $clubuser = Auth::user()->clubUser;
    $joueurs = $clubuser->joueurs;
    $centre_entrainement = Centre_entrainement::where('id', $clubuser->centre_entrainement_id)->first();

    return view('livewire.training', compact('joueurs', 'clubuser', 'centre_entrainement'));
}
```

The Render Method

A Livewire component's `render` method gets called on the initial page load AND every subsequent component update.



In simple components, you don't need to define a `render` method yourself. The base Livewire component class has a dynamic `render` method included.

Returning Blade Views

The `render()` method is expected to return a Blade view, therefore, you can compare it to writing a controller method. Here is an example:



Make sure your Blade view only has ONE root element.

```
1 class ShowPosts extends Component
2 {
3     public function render()
4     {
5         return view('livewire.show-posts', [
6             'posts' => Post::all(),
```

Ce que la doc veut dire c'est que dans un composant Livewire la méthode `render` est appelé au premier chargement et à chaque changement dans le composant par exemple un changement dans la base de données.

La méthode `render()` est attendu à ce qu'elle retourne une view Blade mais on peut la comparer à une méthode dans un controller.

CONCLUSION ET REMERCIMENT :

CONCLUSION :

Pour finir ce dossier,

Ce projet m'a permis d'approfondir mes connaissances sur Laravel qui est un Framework très riche par ces librairies et permet beaucoup de libertés. J'ai pu améliorer mes compétences en développement web que ce soit en PHP, HTML, CSS, JS. J'ai pu apprendre une nouvelle extension pour le css qui est Tailwind css que j'ai trouvé plutôt pratique. J'ai pu aussi enrichir mes techniques de développement et acquérir de bonnes pratiques.

Grâce à ce projet j'ai pu progresser en anglais car les forums et aides sont principalement tous en anglais.

Le projet n'étant pas totalement fini, il me reste encore beaucoup de chose à implémenter tel qu'un algorithme de jeu pour gérer les matchs des clubs entre eux et la mise en place des ligues de football aussi installer un système de notification de la part des développeurs pour une trace des mises à jour.

REMERCIEMENT :

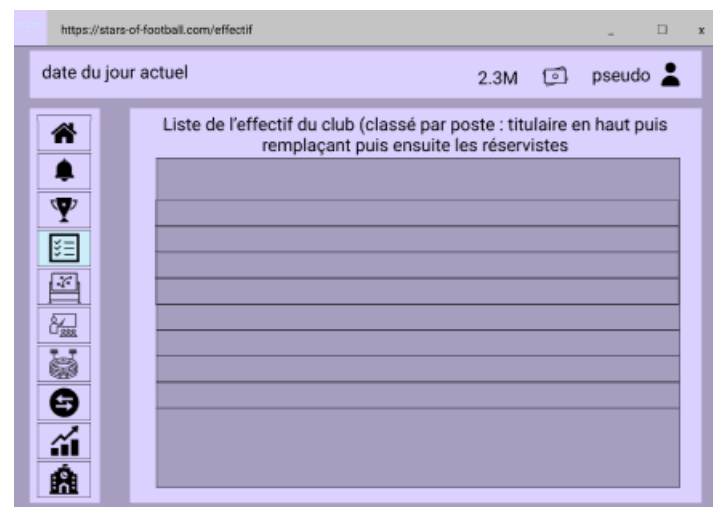
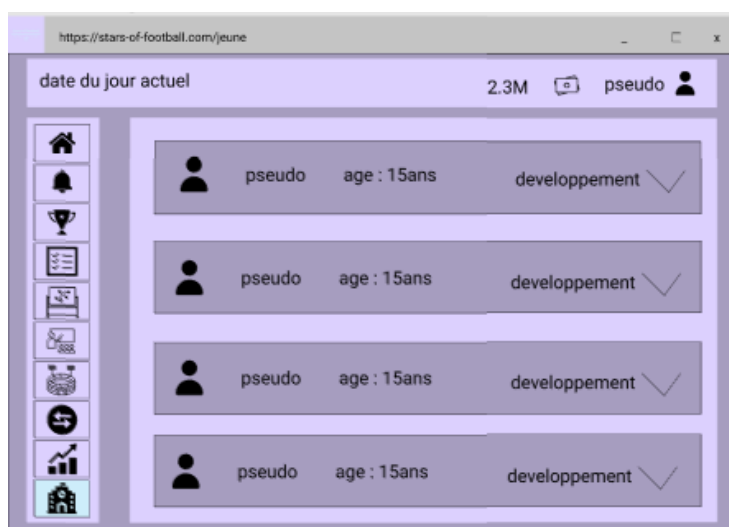
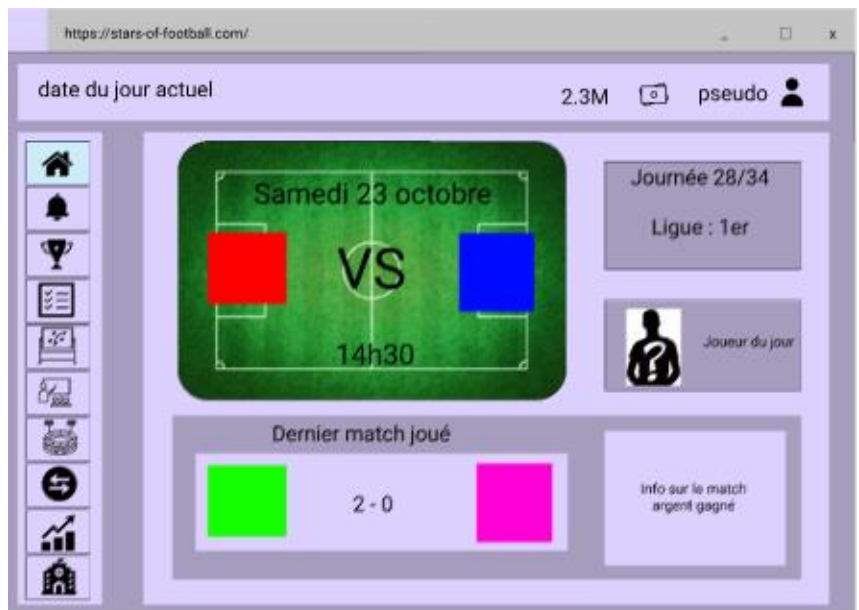
Enfin je tiens à remercier le Callac Soft Collège pour avoir proposé cette formation de développeur web et web mobile ainsi que pour m'avoir accueilli au sein de leur établissement durant cette année.

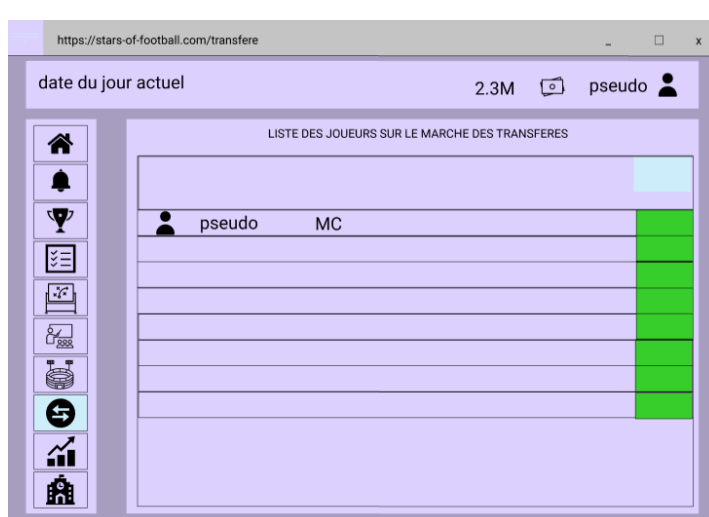
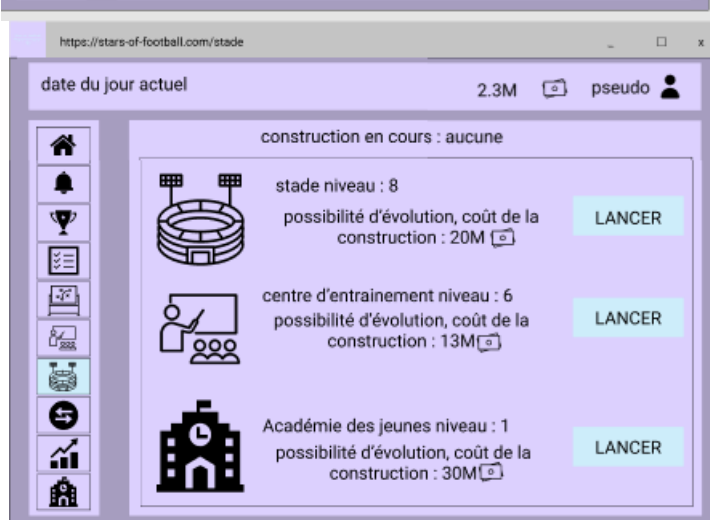
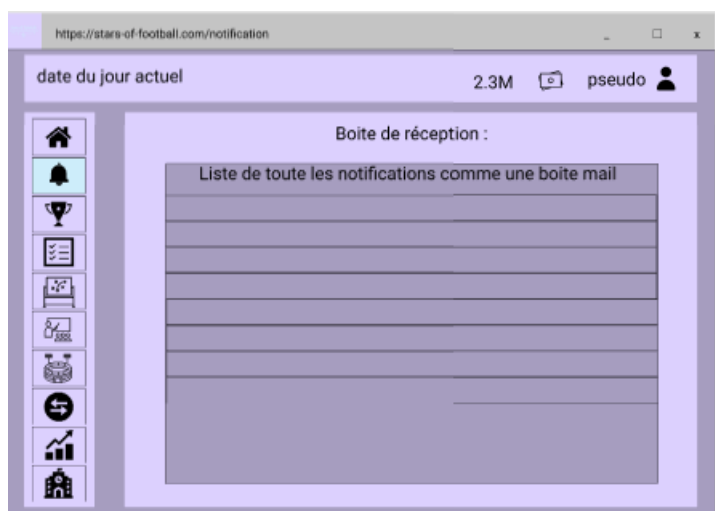
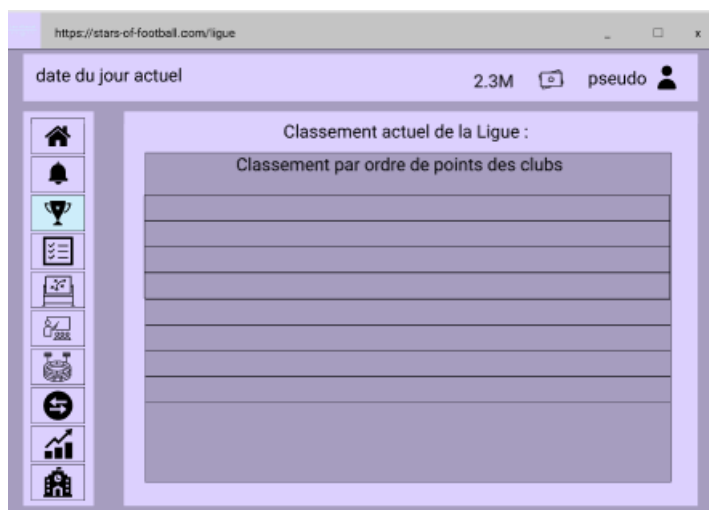
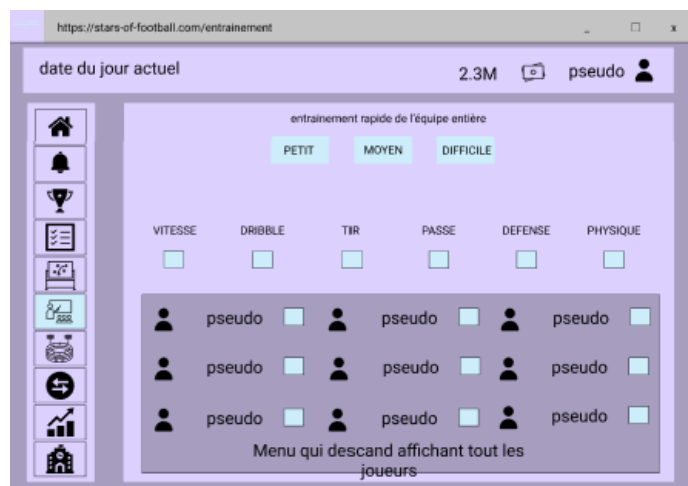
Je remercie particulièrement les deux formateurs Yannick Favreau et Christopher Corbin pour leur implication tout au long de mon apprentissage que ce soit pendant la phase de formation ou en production et m'avoir permis mon entrée dans le monde du développement informatique ainsi que pour leur cours, pour leurs explications passionnées et leur patience.

Je remercie également Henri-Pierre Tison et Christian Rondel pour avoir créé cet établissement qui m'a permis d'apprendre à travailler dans un nouveau domaine auquel j'avais peu de connaissances auparavant et qui dorénavant sera le choix de mon orientation professionnelle.

ANNEXE :

DESKTOP :






MOBILE :

Stars of Football

Email Password

☐ Remember me ☐ No account ? [Forgot your password ?](#)



Rejoignez pleins d'autres joueurs et monter votre club au sommet !

Footer

Stars of Football 2.3M  



Journée 28/34
Ligue : 1er



Joueur du jour

Info sur le match argent gagné



Footer

Stars of Football

Nom du club



Initiale du club

Nom du stade

Stars of Football 2.3M  

Liste de l'effectif

Footer

Stars of Football 2.3M  

PETIT

MOYEN

DIFFICILE

VITESSE ☐


DRIBBLE ☐


TIR ☐


PASSE ☐


DEFENSE ☐


PHYSIQUE ☐


 pseudo ☐

 pseudo ☐

 pseudo ☐

 pseudo ☐

 pseudo ☐

 pseudo ☐

Footer

Stars of Football 2.3M  



2M   pseudo

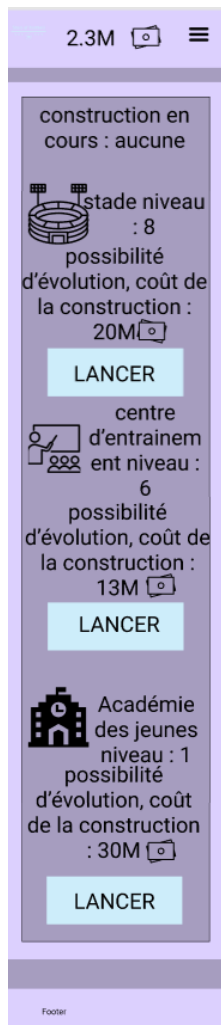
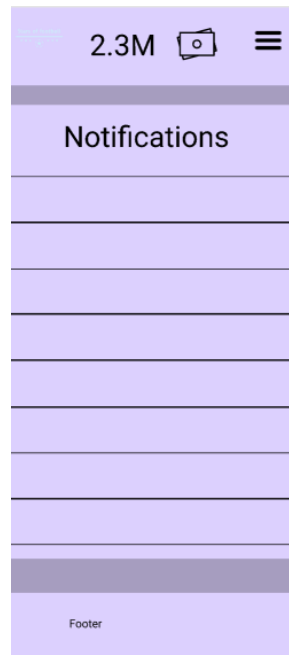
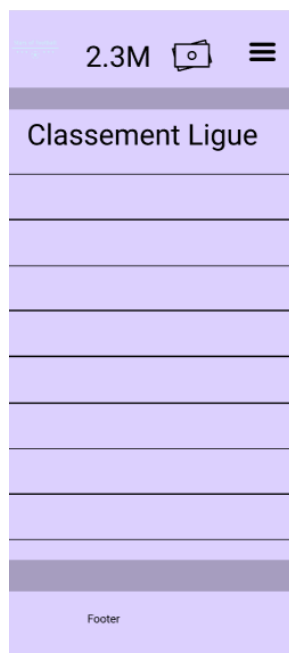
SPONSOR :

2M à la signature
Bonus :
5M si 2eme place au classement de Ligue

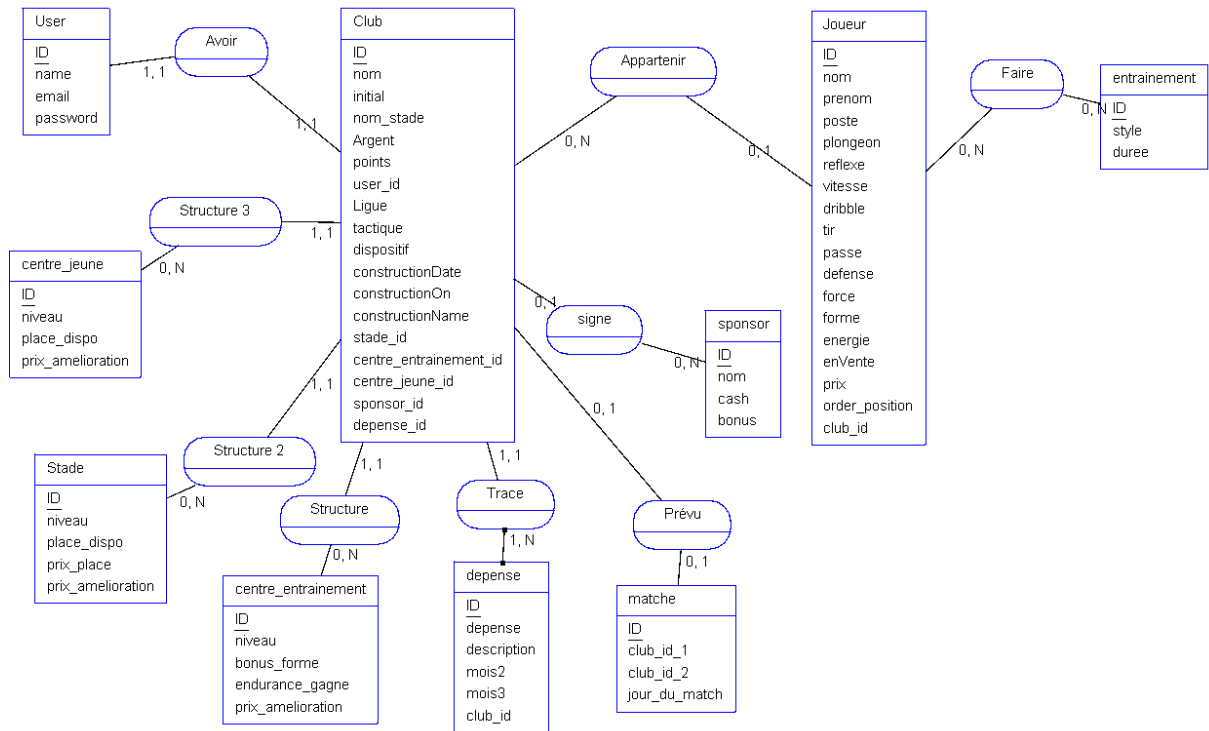
5M à la signature

1M à la signature
Bonus :
7M si 1er au classement de Ligue

Footer



MCD :



MLD :

