CATÉGORIE TECHNIQUE

Parc des marêts — Rue Petermans, 80 – 4100 Seraing

H-F : UN CMS POUR L'UNION ATHLÉTIQUE DES HAUTES-FAGNES

Par Anthony Beaumecker

Travail de fin d'études en vue de l'obtention
du grade de Bachelier en techniques
graphiques orientation web

https://pfe.beaumecker.be

Année académique 2018-2019

REMERCIEMENTS

Je remercie Marine, ma compagne, pour le soutien qu'elle a pu m'apporter et les conseils qu'elle a pu me donner

Je remercie mes amis proches pour toute l'aide qu'ils ont pu m'apporter.

Je remercie ma famille de m'avoir soutenu tous les jours et pour tous les encouragements

Je remercie également mes professeurs sans qui je n'aurais pas pu réaliser un tel projet sans cette formation

Introduction **5**

Partie 1 : Cahier des charges **6**

1. Introduction....7

2. Définition du produit....7

3. Objectifs....7

3.1. Principaux....7

3.2 Secondaires....8

4. Techniques utilisées....8

Partie 2 : Réalisation du design **9**

1. Introduction....10

2. Création de la structure....10

3. Recherche d'inspirations....11

4. L'architecture de mon site....13

5. Création de la maquette graphique....15

5.1 Typographie....15

5.2 Grille....16

5.3 Création du design....16

5.4 Outils utilisés....20

Partie 3 : Développement **21**

1. Développement CSS....22

1.1 Technologies utilisées....22

1.2 Architecture....22

2. Développement côté serveur....24

2.1 Schématisation de la base de données....24

2.2 Présentation de Laravel Backpack....30

2.3 Création d'un CRUD d'un athlète....31

2.3.1 Générer les fichiers de base....31

2.3.2 Le fichier model Athlète....32

2.3.3 Utilisation de l'attribut casting....34

2.3.4 Génération des images....36

2.3.5 Création de la page d'administration d'un athlète....38

2.3.6 Gestion des erreurs....43
2.3.7 Création d'un contrôleur....43
2.4 Création d'un système de recherche....45
2.4.1 La fonction filter....45
2.4.2 La réalisation du select....46
2.4.3 La pagination....47
Conclusion **50**
Bibliographie **52**

INTRODUCTION

J'aimerais d'abord un peu parler des différents choix qui m'ont poussé à faire cette formation. Au niveau du domaine, je savais que je voulais me diriger vers l'informatique. J'ai donc essayé une première année en informatique de gestion qui finalement n'a pas été ce que je recherchais, je m'attendais à quelque chose de moins brut.

Après cela, j'ai voulu suivre une formation en Réseaux et Serveurs, mais la demande était forte et il ne restait plus beaucoup de place en tout cas pour ce qui était de faire un stage. Des connaissances m'ont ensuite conseillé de faire un test pour savoir quel type d'études me correspondrait le plus. Mon choix, c'est donc porté vers l'infographie. Je ne savais pas trop dans quoi je m'engageais, mais j'ai tenté le tout pour le tout.

Lors de ma première année, j'ai tout de suite apprécié les cours concernant le web, j'ai directement su que c'était vers là que j'allais me tourner en deuxième année. Même si j'ai eu beaucoup de mal à arriver jusqu'ici, cela montre néanmoins que j'ai eu l'envie et la passion pour me motiver et ne jamais baisser les bras.

Tout ce texte d'introduction n'a pas vraiment de rapport avec mon projet de fin d'études, mais c'était avant tout pour montrer les choix qui m'ont poussé à être ici.

PARTIE 1 : CAHIER DES CHARGES

1. INTRODUCTION

Avant de démarrer le projet, il était essentiel de définir les différents objectifs afin de savoir dans quelle direction j'allais m'engager. La bonne solution est donc de définir un cahier des charges complet et structuré.

2. DÉFINITION DU PRODUIT

Le produit se définit comme étant un CMS pour mon actuel club d'athlétisme **: L'Union Athlétique des Hautes-Fagnes** pour leur permettre d'avoir leur propre interface d'administration et ainsi de pouvoir mieux gérer la liste de leurs athlètes, entraîneurs et bien d'autres, ce qui n'était pas le cas sur leur site actuel (http://h-f.be).

3. OBJECTIFS

J'ai défini différents objectifs principaux et secondaires. Les objectifs principaux sont ceux qui représentent le site en lui-même. Tandis que les secondaires sont ceux qui permettent d'approfondir ces objectifs principaux.

3.1 OBJECTIFS PRINCIPAUX

- **Afficher la liste des athlètes**
- **Avoir une page de profil pour chaque athlète et entraîneur**
- **Afficher un calendrier pour les compétitions et résultats**
- **Afficher la liste des dirigeants par club (4)**
- **Avoir un planning des entrainements, des stages**
- **Avoir un planning spécifique d'un entraîneur pour un athlète**

3.2 OBJECTIFS SECONDAIRES

- **Afficher les palmarès du club**
- **Mise en avant des athlètes élites et suivi**
- **Afficher les records du club**
- **Tarifs des équipements du club**

4. TECHNIQUES UTILISÉES

Pour la réalisation de ce projet, j'ai décidé d'utiliser différentes technologies. Voici une liste des technologies utilisées pour la réalisation du projet :

- **HTML/CSS :** Les incontournables du web à savoir le langage **HTML** pour structurer sémantiquement mes pages, le langage **CSS,** et pour définir et mettre en forme le contenu visuellement
- **JavaScript :** JavaScript est souvent associé à la modernité. Je l'ai beaucoup utilisé afin d'améliorer le côté dynamique et expérience utilisateur.
- **Laravel et Backpack :** J'ai utilisé le framework **Laravel** comme je voulais obtenir quelque chose d'assez souple, je ne pouvais pas utiliser un CMS classique, j'ai donc décidé d'utiliser un groupe de packages pour me permettre de gérer simplement l'administration et de pouvoir créer mon propre CMS grâce à Laravel.

PARTIE 2 : RÉALISATION DU DESIGN

1. INTRODUCTION

Après avoir défini tous les objectifs de mon site web, j'ai commencé la partie création graphique. Pour commencer, je suis passé par plusieurs phases. J'ai commencé par rechercher plusieurs sources d'inspirations, puis en temps normal, j'aurais créé des maquettes que l'on appelle « wireframes », qui m'auraient permis de structurer comme il faut mon site, mais j'ai appris une approche différente que j'expliquerais plus tard. Pour terminer, après avoir terminé mon architecture, j'ai commencé la partie conception qui se base sur mon architecture. J'ai donc réalisé plusieurs maquettes graphiques qui m'aideront à déterminer l'aspect visuel de mon application.

2. CRÉATION DE LA STRUCTURE

Juste avant de commencer à réaliser mon architecture, j'ai dû établir un plan du site afin de mieux m'imaginer ma structure. Voici donc la structure imaginée :

- **Menu principal :**
    - Union Athlétique
        - À Propos
        - Nous rejoindre
        - Équipements et tarifs
    - Athlètes
    - Entraîneurs
    - Actualités
    - Agenda
        - Compétitions
        - Entrainements
        - Stages
    - Contact

3. RECHERCHE D'INSPIRATIONS

La recherche d'inspirations fait partie d'une des phases majeures de la réalisation d'un bon design. On arrive à un moment où l'on a fait un peu le tour. Il m'arrive souvent de rester pendant une heure devant un page, car je n'ai pas envie de repartir sur quelque chose que je connais, je veux élargir les possibilités. Voilà pourquoi je trouve que la recherche d'inspirations est tellement importante, elle permet de refaire le plein d'idées.

J'ai utilisé quelques sites pour m'inspirer. En voici la liste :

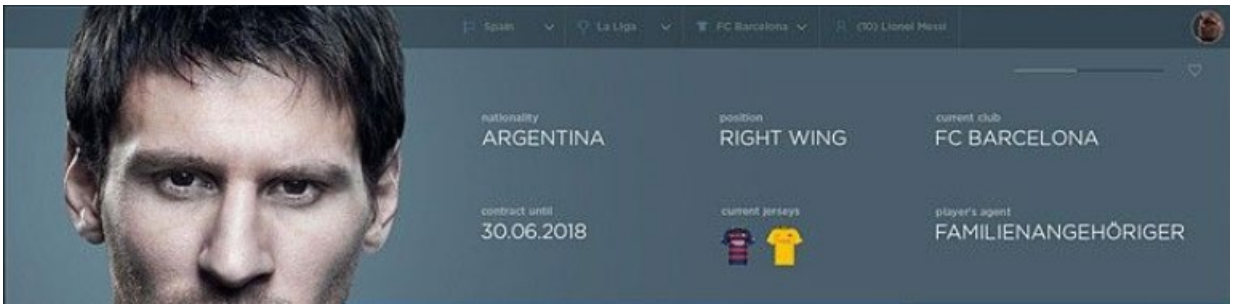
– Pinterest : C'est pour moi le site de référence pour ma recherche d'inspiration. Il regroupe des designs, idées d'autres sites très connus comme Behance. J'aime beaucoup l'idée de créer des tableaux pour trier ces inspirations par thème ou projet. Voici également le lien regroupant toutes mes inspirations de Pinterest (https://www.pinterest.fr/anthonybeaumeck/h-f/).

– Behance : Je suis un petit admiratif de Behance, mais je l'utilise si je ne trouve pas ce qui me convient comme inspiration dans Pinterest.

– Google : J'utilise plus Google pour rechercher des sites qui se rapprocheraient plus des idées de sites vers lesquelles j'aurais envie de me tourner par exemple des sites de clubs d'athlétismes professionnels.

Voici également une partie de mes inspirations qui ont pu m'aider à réaliser certaines parties de mon site :

EXEMPLE DU PROFIL D'UN SPORTIF :

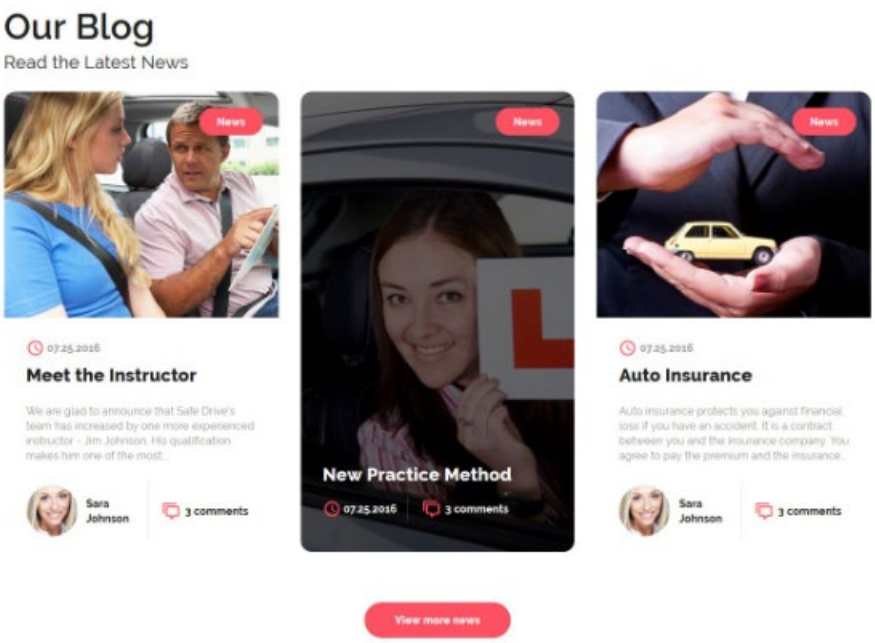


Je suis parti sur le même style pour afficher les informations générales d'un athlète.

lille-athletisme.fr/ :

J'aimais beaucoup l'ambiance générale du site. Je ne me suis pas spécialement inspiré de ce site au niveau graphique, mais plus dans le style moderne, épuré et sophistiqué.

EXEMPLE DES DERNIERS POSTS PUBLIÉS :



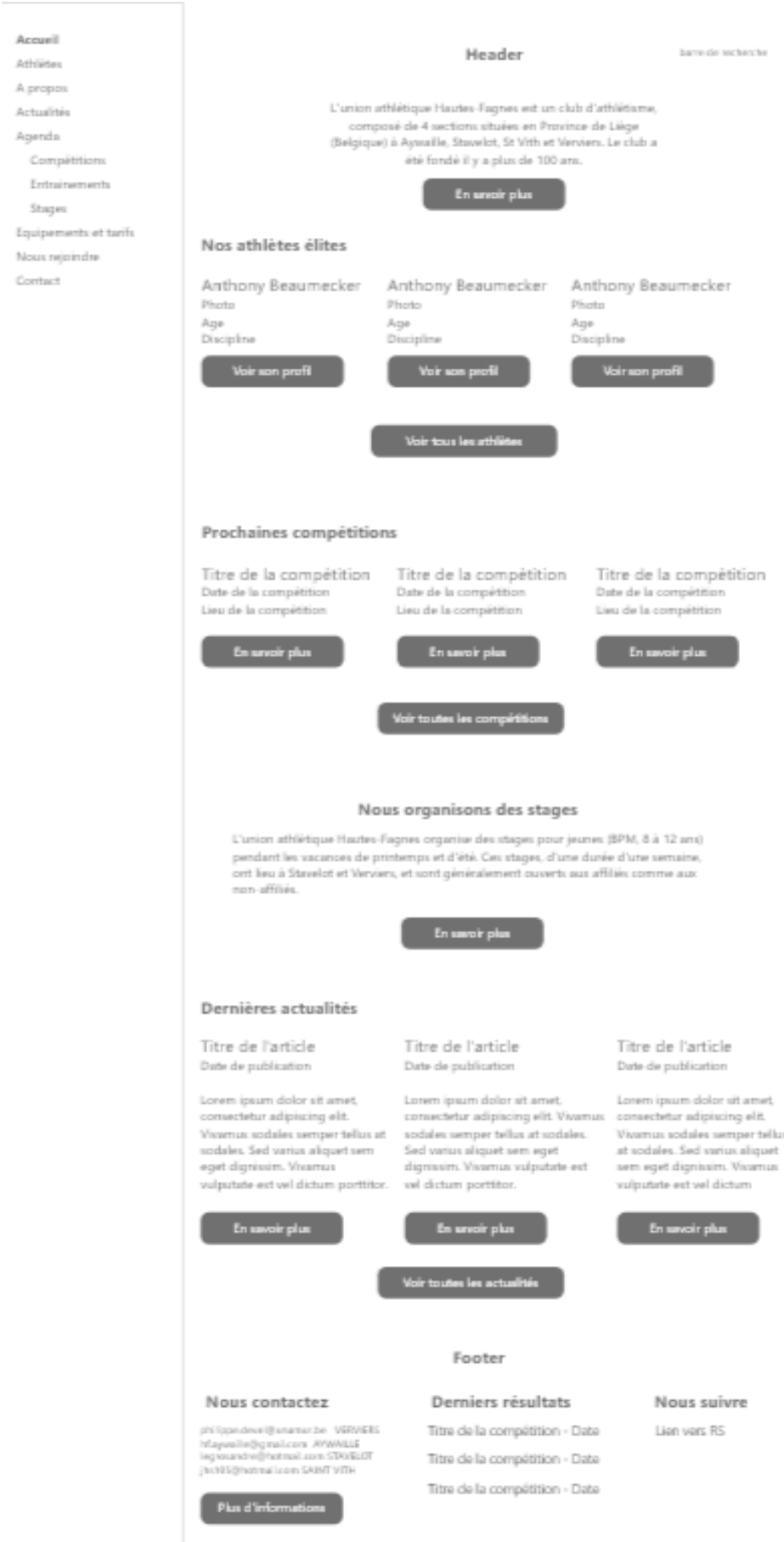
Je trouvais que la représentation d'un post correspondait bien ce que je cherchais. Le bloc regroupait toutes les informations nécessaires, mais ne faisait, pas trop charger et cela respirait dans bloc. J'ai donc repris à quelques choses près la même structure.

J'ai bien entendu consulté plusieurs autres sources d'inspirations, mais mes recherches sur des sites concernant des clubs d'athlétisme n'ont pas été un franc succès. La plupart des sites n'étant absolument pas à jour au niveau visuel.

4. L'ARCHITECTURE DE MON SITE

Pour réaliser mon architecture, j'ai utilisé le logiciel **Adobe Experience Design**, qui est un logiciel qui permet de créer rapidement des maquettes graphiques et autres. C'est une approche différente de celle de la réalisation de wireframes qui consiste juste à empiler différents blocs pour avoir une idée de la structure. Certes, cela est utile, je ne dis pas le contraire, mais réaliser une architecture sous forme de texte, reliant chaque page entre elles, m'a permis de mieux cerner l'ampleur du projet et du contenu que j'allais avoir.

Voici un exemple de l'architecture de ma page d'accueil :


Accueil
Athlètes
A propos
Actualités
Agenda
Compétitions
Entrainements
Stages
Equipements et tarifs
Nous rejoindre
Contact
Header
barre de recherche
L'union athlétique Hautes-Fagnes est un club d'athlétisme, composé de 4 sections situées en Province de Liège (Belgique) à Aywaille, Stavelot, St Vith et Verviers. Le club a été fondé il y a plus de 100 ans.
En savoir plus
Nos athlètes élites
Anthony Beaumecker
Photo
Age
Discipline
Voir son profil
Anthony Beaumecker
Photo
Age
Discipline
Voir son profil
Anthony Beaumecker
Photo
Age
Discipline
Voir son profil
Voir tous les athlètes
Prochaines compétitions
Titre de la compétition
Date de la compétition
Lieu de la compétition
En savoir plus
Titre de la compétition
Date de la compétition
Lieu de la compétition
En savoir plus
Titre de la compétition
Date de la compétition
Lieu de la compétition
En savoir plus
Voir toutes les compétitions
Nous organisons des stages
L'union athlétique Hautes-Fagnes organise des stages pour jeunes (BPM, 8 à 12 ans) pendant les vacances de printemps et d'été. Ces stages, d'une durée d'une semaine, ont lieu à Stavelot et Verviers, et sont généralement ouverts aux affiliés comme aux non-affiliés.
En savoir plus
Dernières actualités
Titre de l'article
Date de publication
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus sodales semper tellus at sodales. Sed varius aliquet sem eget dignissim. Vivamus vulputate est vel dictum porttitor.
En savoir plus
Titre de l'article
Date de publication
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus sodales semper tellus at sodales. Sed varius aliquet sem eget dignissim. Vivamus vulputate est vel dictum porttitor.
En savoir plus
Titre de l'article
Date de publication
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus sodales semper tellus at sodales. Sed varius aliquet sem eget dignissim. Vivamus vulputate est vel dictum
En savoir plus
Voir toutes les actualités
Footer
Nous contactez
philippedevel@smamar.be VERVIERS
hfaywaille@gmail.com AYWAILLE
leglorsandre@hotmail.com STAVELOT
jbch93@hotmail.com SAINT VITH
Plus d'informations
Derniers résultats
Titre de la compétition - Date
Titre de la compétition - Date
Titre de la compétition - Date
Nous suivre
Lien vers RS

C’est assez simpliste, mais cela m’a permis de savoir quel type de contenu j’allais intégrer, les différentes zones qui allaient apparaître et cela m’a surtout donné de nouvelles idées, car mes pages ont beaucoup évolué entre temps.

5. CRÉATION DE LA MAQUETTE GRAPHIQUE

Dans les phases de créations d’un site, c’est l’étape ou je prends le plus de plaisir ou l’on peut faire parler son imagination. C’est aussi l’occasion de faire des tests et de voir si des choses fonctionnent bien ou d’autres moins bien.

5.1 TYPOGRAPHIE

J’ai fait plusieurs tests de polices. J’ai surtout cherché dans les polices sans-sérif et sérif. J’ai finalement opté pour la sécurité en choisissant la police **Montserrat** pour les titres et **Open Sans** pour le corps que j’utilise pour écrire actuellement ce rapport.

Montserrat Bold

Open Sans Regular

J’ai choisi la sécurité dans le choix de mes polices. **Montserrat** et **OpenSans** s’accordent assez bien. Le seul défaut de la police **Montserrat**, c’est qu’elle ne possède pas de style italique.

5.2 GRILLE

Après avoir choisi mes tailles de polices, j'ai créé une grille à mon image. Voici les paramètres de la grille :

- Largeur totale : **1160px**
- Nombre de colonnes : **12**
- Largeur gouttières : **16**
- Largeur colonnes : **82**

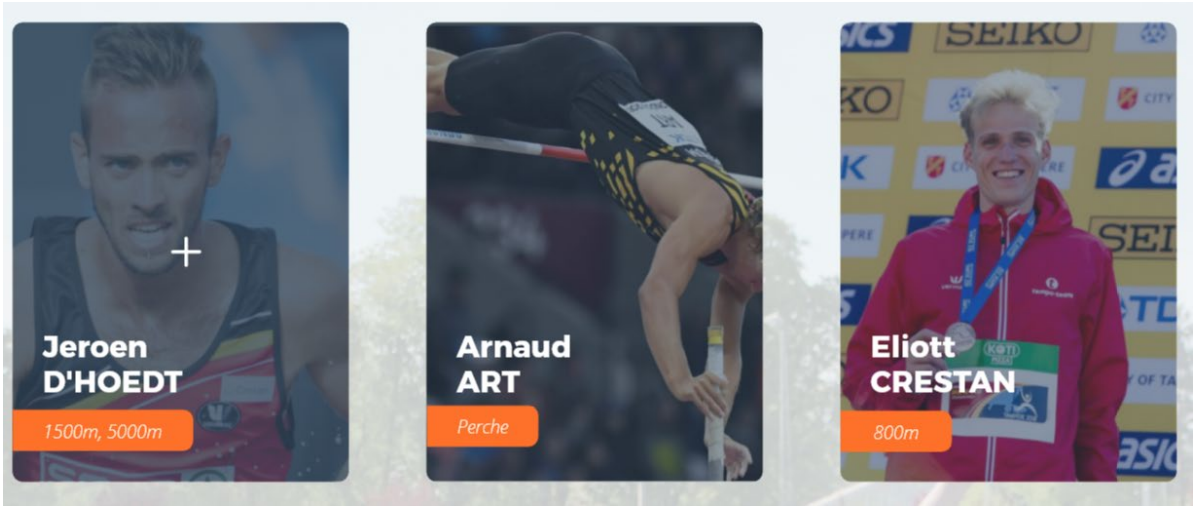
J'ai choisi ces paramètres-là, car cette grille prêtait au design que je voulais réaliser, c'est-à-dire quelque de moderne et très espacé. De plus, la plupart des écrans actuels rendent au moins 1200px.

5.3 CRÉATION DU DESIGN

Après avoir fini toutes mes recherches, j'ai pu commencer à réaliser mon design. La réalisation du design a été assez longue. J'ai dû reprendre de zéro certaines pages pour pouvoir atteindre un résultat que je trouverais satisfaisant. La page qui m'a demandé le plus de temps fut la page d'accueil. J'avais pas mal d'informations à afficher, j'ai dû choisir quelles informations serait les plus importantes.

Ce choix a joué par rapport aux utilisateurs. Certes, le but premier du site est de permettre au club d'avoir leur propre site et de pouvoir le gérer plus facilement, mais aussi de donner envie à d'autres personnes de rejoindre le club, d'où l'ajout d'un bloc pour commencer un stage dans la page d'accueil.

RÉALISATION DE LA SECTION ATHLÈTES POUR LA PAGE D'ACCUEIL



Pour cette section, lors de mes premiers essais, j'étais parti sur un style ou l'on avait un bloc divisé en deux. D'une part en haut, la photo de l'athlète et d'autre part les informations générales de l'athlète. Cependant, cela ne mettait pas assez en valeur les athlètes. Quand on arrive sur le site, c'est quand même la première section que l'on voit après le header. J'ai donc opté pour une image prenant tout l'espace et d'y déposer les informations de l'athlète sur celle-ci.

RÉALISATION DE LA FICHE D'UN ATHLÈTE

Records personnels

Discipline	Record	Lieu	Date
1500m	3.36.07	Oordegem	02/08/2011
5000m	13.33.09	Heusden	07/07/2012

Évolution

Année	Catégorie	1500m	5000m
2012	Espoir	3.40.97	-
2013	Senior	3.38.61	-
2014	Senior	3.38.20	13.34.90
2015	Senior	3.41.44	-
2016	Senior	-	13.52.11
2017	Senior	-	14.15.98
2018	Senior	3.38.61	-

L'une des sections les plus compliquées a été la fiche d'un athlète. Le gros point faible de l'ancien site est que les athlètes n'étaient pas où très peu mis en avant. J'ai donc voulu remédier à cela. Il allait donc de soi que je devais faire une fiche pour chaque athlète.

Pour la section des records personnels et l'évolution, j'avais dans l'optique et de prévoir une représentation graphique sous forme graphique pour marquer l'évolution de l'athlète. Mais, le problème était que les records établis ou l'évolution sont des chiffres assez précis et devoir écrire chaque fois ce temps sur le schéma pour tous ses records aurait donné un graphique chargé ou l'on aurait du mal à s'y retrouver. J'ai donc opté pour un tableau simple, mais très représentatif tout en donnant de l'espace au bloc, ce qui donne une sensation plus agréable à lire.

Voici quelques aperçus des autres sections de mon site.

RÉALISATION DE LA FICHE D'UNE COMPÉTITION

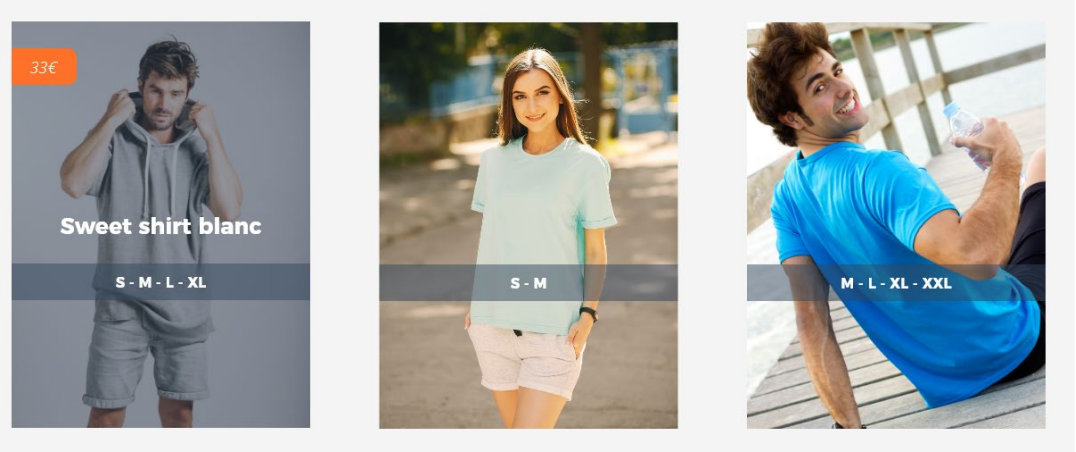
Pour les compétitions je ne voulais pas juste avoir un calendrier regroupant toutes les compétitions, je voulais que chaque compétition puisse avoir sa propre page afin que les utilisateurs soient suffisamment informés sur la compétition en question.

RÉALISATION DE LA SECTION DES ENTRAINEMENTS

En revanche pour les entrainements, je n'ai pas prévu de page pour chaque entrainement, car ici il ne s'agissait simplement que d'un planning et je n'avais pas prévu de fonctionnalités utiles pour cela. J'ai cependant trouvé l'idée intéressante à ce que l'utilisateur puisse savoir quel entraîneur donnera lieu à l'entrainement, afin aussi de mieux connaître sa carrière.

RÉALISATION DE LA SECTION DES STAGES

RÉALISATION DE LA SECTION DES ÉQUIPEMENTS



Pour la section des équipements, j'ai vraiment voulu rendre la page identique à un site de shopping sans pour autant que l'on puisse acheter, ce qui n'était pas le but, et que cela nécessiterait, de faire une application à part à développer, sachant que ce n'était pas le but premier du site.

5.4 OUTILS UTILISÉS

Pour m'aider dans la réalisation de mon design, je me suis servi principalement du logiciel de la suite Adobe : **Abode Experience Design**. Pour les utilisateurs de Windows, c'est un peu une variante de **Sketch**, que je regrette de ne pas avoir pu utiliser pour ce projet.

Cela n'empêche pas pour **Adobe Experience Design** d'être un très outil. Contrairement à Photoshop, son interface est simple, rapide et pensée pour la conception de maquettes graphiques. L'exportation est aussi très rapide, je peux sélectionner toutes mes icônes d'un seul coup en les sélectionnant toutes. Pour la compression de mes icônes SVG, j'ai utilisé un compresseur qui est assez connu : svgomg. Comme je ne comptais pas faire des animations sur mes SVG, cela me suffisait. Pour finir, j'ai utilisé **Illustrator** pour la réalisation d'icônes personnalisées. Cet outil est je trouve à mon sens parfait pour faire du vectoriel.

PARTIE 3 : DÉVELOPPEMENT

1. DÉVELOPPEMENT CSS

Le **CSS** (Cascading **S**tyle **S**heet) est le langage incontournable qui permet de mettre en forme du contenu visuellement. Ce langage est aujourd'hui très riche en propriétés qui permettent d'intégrer un peu près n'importe quoi très rapidement. Si je parle du **CSS**, c'est surtout pour expliquer la façon dont j'ai structuré mon code et qui m'a permis au fil de mes erreurs d'améliorer cette structure.

1.1 TECHNOLOGIES UTILISÉES

Pour les technologies utilisées, j'ai décidé d'utiliser un préprocesseur, un langage de programmation qui permet de gérer plus efficacement ses feuilles de styles **CSS**.

Lors de ma formation, on a appris à utiliser **Sass,** mais j'ai préféré m'orienter vers **Stylus** qui est beaucoup plus simple que **Sass** déjà rien que dans la syntaxe. Si je n'ai pas envie de mettre des accolades, des points virgules ou des deux-points, cela ne m'empêchera pas de faire compiler mon code. Rien que pour cela, je préfère utiliser **Stylus**. Le seul défaut avec **Stylus**, c'est qu'il faut être attentif avec l'indentation du code.

1.2 ARCHITECTURE

Pour organiser mes feuilles de styles, j'ai utilisé l'architecture **ITCSS** que j'ai apprise lors de ma formation. Elle permet de diviser l'organisation des feuilles de styles en différentes parties.

- Base
- Components
- Object
- Generic
- Settings
- Tools
- Trumps

Le but d'ITCSS est aussi de pouvoir organiser son code comme on le souhaite. J'ai d'abord mis tous mes styles de bases dans le dossier **Base**. En dehors des dossiers, je crée souvent des fichiers pour y stocker mes styles pour mes titres, mes buttons de façon à ce qu'il soit réutilisable sur tout le site.

Dans le dossier **Generic**, j'y ai intégré tous les resets que j'utilise, c'est-à-dire, le reset de **Meyer** et de **Flexbox**. Dans le dossier **Settings,** j'ai géré mes différents chargements de polices, d'icônes, mes variables pour les couleurs et mes fonctions easing.

En ce qui concerne le dossier **Objects**, je n'ai jamais vraiment compris son utilité. Je m'embrouillais plus l'esprit qu'autre chose à disperser mon code entre le dossier **Objects** et le dossier **Components**. C'est d'ailleurs dans ce dossier que j'y code la plupart de mes styles. Généralement, je crée plusieurs dossiers à l'intérieur reprenant une page spécifique auquel je divise encore en plusieurs fichiers à l'intérieur. Cela m'aide à trouver directement ce que je cherche et m'évite de perdre du temps à chercher dans de longs fichiers.

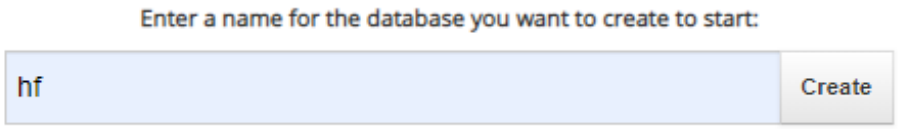
2. DÉVELOPPEMENT CÔTÉ SERVEUR

On arrive à la partie la plus importante du site, la partie qui a demandé le plus de travail, car je n'ai utilisé **Laravel** que très brièvement. J'ai donc dû suivre quelques formations pour me remettre à jour. Je vais tâcher d'expliquer au mieux la réalisation de mon application, étape par étape.

2.1 SCHÉMATISATION DE LA BASE DE DONNÉES

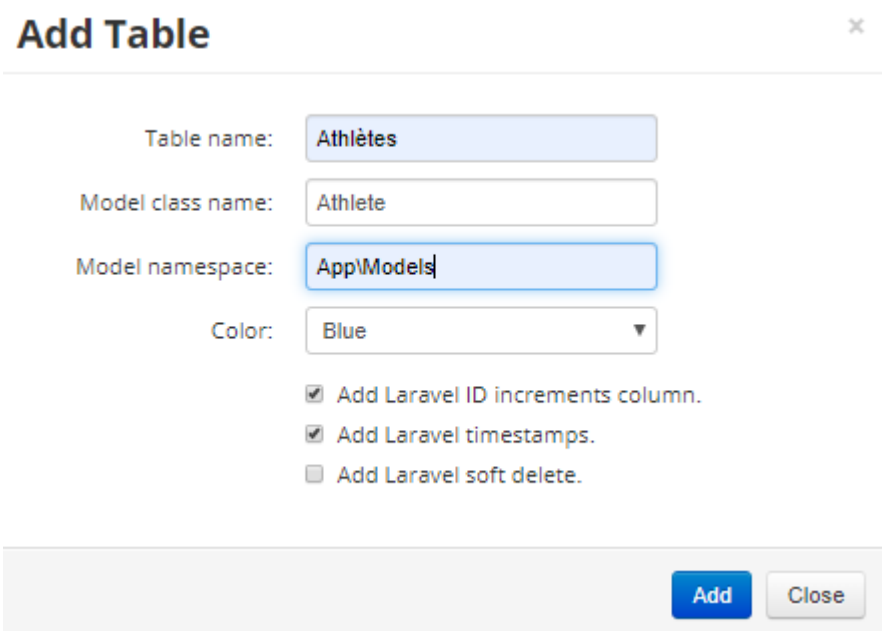
Pour réaliser ma base de données, j'ai d'abord voulu la schématiser pour avoir une vue d'ensemble claire et précise. Pour ce faire, j'ai utilisé un outil en ligne qui s'appelle **Laravel Schema Designer** (laravelsd.com).

Pour créer un schéma, il suffit de m'inscrire et d'y entrer le nom que je veux lui donner.

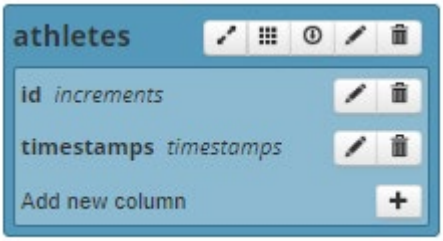


Une fois, la base de données crée, je peux commencer à créer les tables dont j'aurais besoin en appuyant sur le bouton « Add table ». Une fenêtre pop-up apparaît :

On me demande de remplir plusieurs informations. Je peux donner un nom à ma table et mon namespace, ainsi que le nom pour mon fichier Model. Je peux également lui assigner une couleur pour ne pas que les tables soient toutes identiques visuellement.

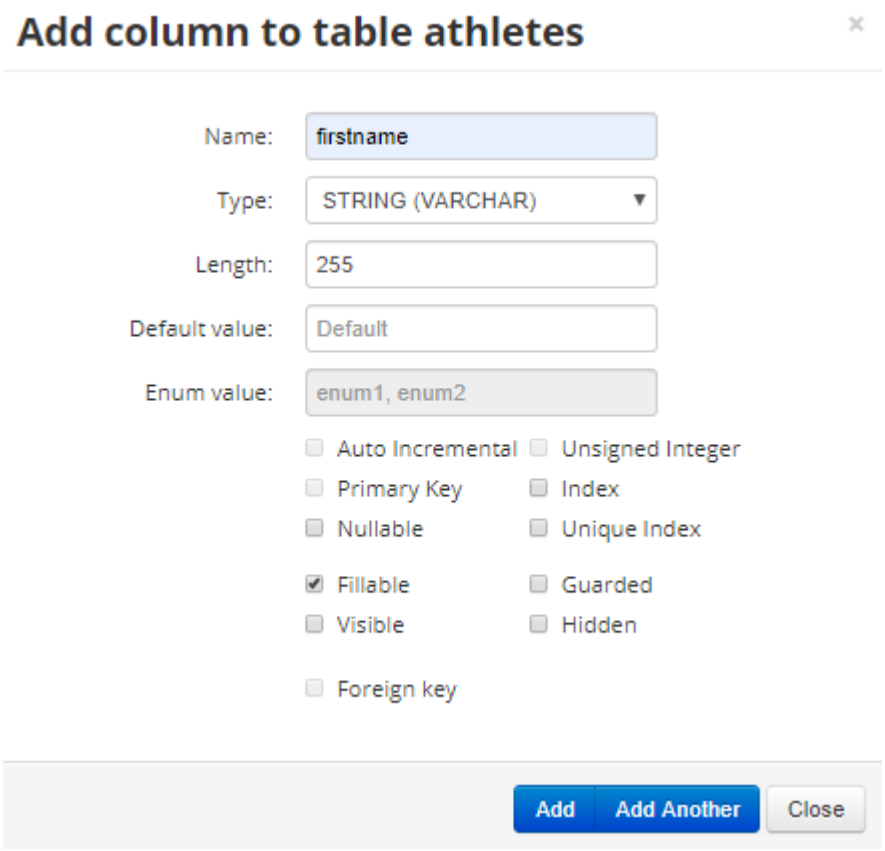


Après avoir remplis tous les champs, voici ce que j'obtiens après avoir cliqué sur le bouton « Add » :

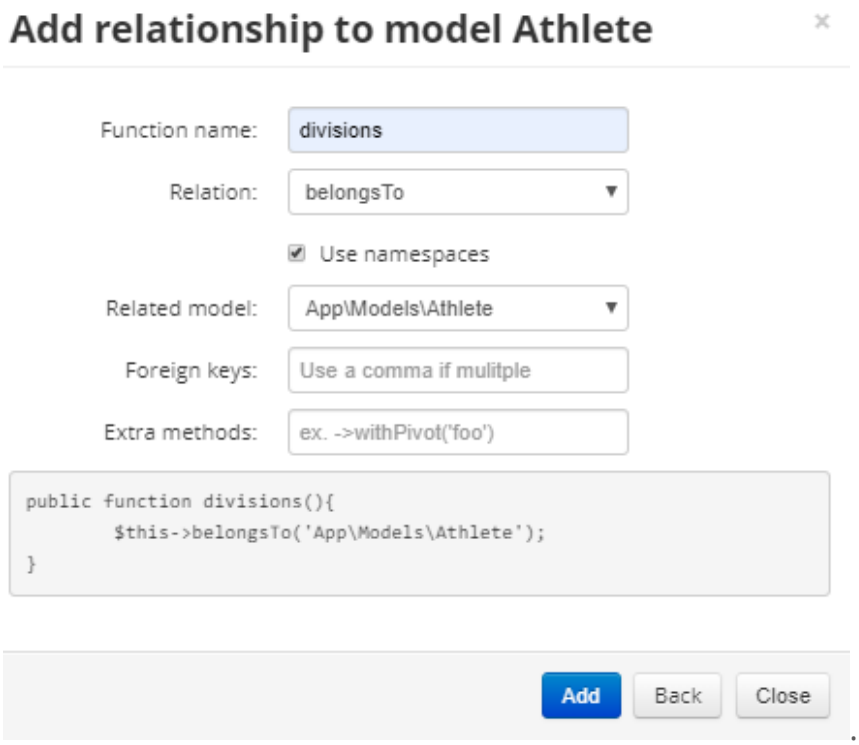


Je n'ai que les champs de base qui sont créés, je n'ai plus qu'à ajouter ceux qui me manquent. Pour ce faire, je clique sur le petit bouton « + ». Une nouvelle fenêtre pop-up apparaît :

On me demande ici de remplir toutes les données de la colonne. Je crée ici une colonne qui reprendra le nom complet de l'animal, je précise que c'est un VARCHAR avec une limite de caractères de 255 maximum. Pour les cases à cocher, ici je précise que le champ peut être nul et que le champ peut être rempli grâce à la case « Fillable ». Ensuite, je peux choisir d'appuyer sur « Add » pour voir le résultat ou « Add another » pour continuer directement a ajouté d'autres champs.

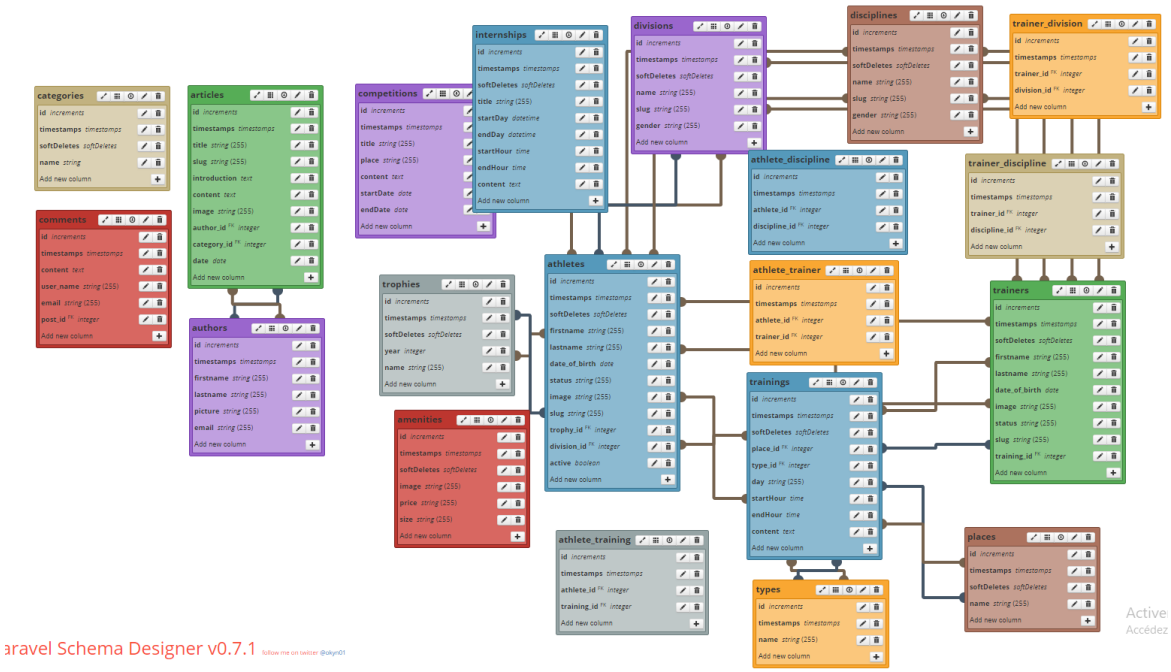


La prochaine étape est de créer des relations entre chaque table au besoin. Ici, j'ai besoin de créer une relation avec la table **Divisions**. Donc, je dis qu'un athlète ne peut avoir qu'une seule division, dans ce cas-là, j'aurais besoin d'une relation **BelongsTo**. Et dans l'autre sens, une division peut avoir un ou plusieurs athlètes, j'aurais donc besoin d'une relation **hasMany**. Pour créer une relation entre des tables, je clique sur les petites flèches en haut à gauche de la table, une nouvelle fenêtre pop-up apparaît de nouveau :



Comme on peut le voir en dessous des différents champs, cela respecte assez bien la création d'une fonction pour faire une relation.

En fin de compte, cela aura été un challenge de réaliser cette base de données, car je devais également penser plus loin que ce que j'allais présenter. J'ai donc déjà mis en place les différentes tables dont j'aurais besoin dans un futur proche. Voici un aperçu de ma base donnée, pour vous donner une idée de la taille de l'application.



Cependant, les avantages de cet outil ne s'arrêtent pas là. Je peux appuyer sur le bouton « Export All ». Exportons par exemple les schémas. Si je vais dans mon dossier que l'outil m'a exporté et que je choisis par exemple la table pour la création d'un athlète.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;

class CreateAthletesTable extends Migration {

    public function up()
    {
        Schema::create('athletes', function(Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->softDeletes();
            $table->string('firstname', 255);
            $table->string('lastname', 255);
            $table->date('date_of_birth');
            $table->string('status', 255);
            $table->string('image', 255)->nullable();
            $table->string('slug', 255);
            $table->integer('trophy_id')->unsigned();
            $table->integer('division_id')->unsigned();
            $table->boolean('active');
        });
    }

    public function down()
    {
        Schema::drop('athletes');
    }
```

Laravel Schéma Designer nous a donc créé la migration complète de la table. Maintenant voyons voir ce que cela donne si j'exporte le fichier de cette même table.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Athlete extends Model
{

    protected $table = 'athletes';
    public $timestamps = true;

    use SoftDeletes;

    protected $dates = ['deleted_at'];
    protected $fillable = array('firstname', 'lastname', 'date_of_birth', 'status', 'image', 'slug', 'trophy_id', 'division_id', 'active');

    public function trainers()
    {
        return $this->belongsToMany('App\Models\Trainer');
    }

    public function disciplines()
    {
        return $this->belongsToMany('App\Models\Discipline');
    }

    public function divisions()
    {
        return $this->belongsTo('App\Models\Division');
    }

    public function trophies()
    {
        return $this->hasMany('App\Models\Trophy');
    }

    public function trainings()
    {
        return $this->belongsToMany('App\Models\Training');
    }
```

Comme on peut le voir, le fichier contient bien tous les éléments que contient un fichier modèle. Si l'on regarde plus, on peut même voir qu'il nous a créé les relations pour nous.

Cet outil est vraiment un gain de temps énorme en ce qui concerne la schématisation d'une base de données. Simple et rapide à prendre en main.

2.2 PRÉSENTATION DE LARAVEL BACKPACK

Avant de commencer à réaliser mon projet, je ne savais pas trop avec quelle technologie, j'allais pouvoir développer mon site web. Puis, j'ai repensé à **Backpack**. Je me suis dit que cela serait une bonne opportunité pour apprendre l'outil.

D'après leur site, Backpack se présente comme étant « The most popular admin panel software for Laravel » et qu'avec leur produit on peut une page d'administration super rapidement. Pourquoi est-ce que j'ai choisi Backpack alors que maintenant il existe un autre framework qui utilise Vue en plus ? Premièrement, car Backpack est gratuit et deuxièmement, car je voulais d'abord apprendre à maîtriser Backpack qui m'a plu au premier regard.

Backpack contient plusieurs packages utiles à installer. Le plus important et le premier à installer sont **« Base »**. C'est l'intégration du thème « Admin TLE » dans Laravel. Tout seul, ce package ne sert pas à grand-chose à part si vous voulez développer l'administration avec ce thème. Pour l'installer, il faut bien évidemment avoir une installation Laravel.Ensuite, vous n'avez plus qu'à suivre la documentation officielle juste ici
https://backpackforlaravel.com/docs/3.4/installation

Le deuxième package à installer afin de pouvoir commencer votre application est le « **CRUD** » qui signifie **C**reate **R**ead **U**pdate **D**elete. Ses éléments centraux de Backpack, pour faire simple il permet de créer une interface pour l'ajout/mise à jour et la suppression efficacement et rapidement. Ce package dispose d'une multitude de types de champs différents qui vous permettent de gérer un peu près tout.

Backpack propose également ensuite d'autres packages comme **PageManager** qui permet de gérer les différentes pages, **PermissionManager** pour donnée des droits d'accès…

2.3 CRÉATION DU CRUD D'UN ATHLÈTE

La première chose que j'ai faite après avoir installé Backpack, c'est de créer mon premier CRUD. Je vais montrer pas à pas la réalisation d'un CRUD pour athlète.

2.3.1 GÉNÉRER LES FICHIERS DE BASE

Pour créer un CRUD, nous avons besoin de 3 fichiers de base, afin de pouvoir commencer :

- Un fichier **Model** pour interagir avec la base de données, lui dire quels champs peuvent être remplis, les différentes relations…
- Un fichier **CRUD** Controller qui sera placé dans un dossier Admin dans les controllers. C'est ce fichier qui sert à la création d'une vue dans l'administration et a géré ses CRUD.
- Un fichier **Request** qui va ajouter des contraintes aux différents types de champs dans l'administration.

Pour générer ses 3 fichiers de base, j'ai utilisé la commande suivante :

- **PHP artisan backpack : crud athlète**

Grâce à cette commande, vous pouvez donc créer les fichiers de base nécessaire à la création d'un CRUD dans Backpack. Regardons maintenant de plus les 3 fichiers qui ont été générés.

2.3.2 LE FICHIER MODÈLE ATHLÈTE

Analysons ce fichier Model **Athlète** qui a été placé dans App\Models :

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Backpack\CRUD\CrudTrait;

class Athlete extends Model
{
    use CrudTrait;

    /*
    |--------------------------------------------------------------------------
    | GLOBAL VARIABLES
    |--------------------------------------------------------------------------
    */

    protected $table = 'athletes';
    // protected $primaryKey = 'id';
    // public $timestamps = false;
    // protected $guarded = ['id'];
    protected $fillable = [];
    // protected $hidden = [];
    // protected $dates = [];
```

On peut déjà donc constater que le fichier est déjà bien configuré. Le fichier est prérempli avec le **namespace**, le **CrudTrait** (va permettre de lier le Model à la fonctionnalité de CRUD de Backpack), ainsi que quelques variables pour gérer les éléments dans la base de données.

Dans le fichier de base, la plupart des variables sont mises en commentaires, car nous n'avons pas besoin de toutes les variables pour le moment. Les seules dont nous avons besoin pour l'instant sont : **primaryKey, timestamps** et **fillable.**

La première chose à faire après avoir dé commenter les variables nécessaires, c'est de remplir la variable **$ fillable** avec les différents champs qui sont présents dans ma table.

```
protected $fillable = array(
    'firstname',
    'lastname',
    'image',
    'date_of_birth',
    'status',
    'slug',
    'division_id',
    'active',
    'profession',
    'records',
    'evolution'
);
```

Comme on peut le voir, j'ai un champ qui se nomme « **slug** ». Il est très important, car c'est grâce à lui que je vais pouvoir définir des URL à mes athlètes. Pour pouvoir faire cela, je vais devoir installer une dépendance qui se nomme **eloquent-sluggable.** Cette dépendance va permettre sur base des éléments que je vais lui donner de créer un slug qui sera utilisable sous forme d'URL.

Pour que cela fonctionne, je dois préciser à mon Model d'utiliser cette dépendance que je viens d'installer. Je dis donc à mon fichier d'utiliser l'espace de nom **Cviebrock\EloquentSluggable\Sluggable**. Et de l'utiliser dans ma classe **Athlète**.

Je vais ensuite créer ma méthode **sluggable()** afin de définir mon slug.

```
public function sluggable()
{
    return [
        'slug' => [
            'source' => 'slug_or_title',
        ],
    ];
}
```

Et ensuite, créer un **accessor** pour mon slug se base sur le **firstname** et le **lastname** de mon athlète.

```
public function getSlugOrTitleAttribute()
{
    if ($this->slug != '') {
        return $this->slug;
    }

    $lastname = $this->lastname;
    $firstname = $this->firstname;

    return $firstname . '-' . $lastname;
}
```

Cependant, lors de la création d'un athlète, mon slug ne renvoyait pas ce que je désirais, il ne renvoyait que l'id de l'objet. Pour pallier à ce problème, j'ai dû préciser que mon slug devait retourner le champ **slug** et non l'id de l'objet en question.

```
public function getRouteKeyName()
{
    return 'slug';
}
```

Maintenant, si je crée un athlète grâce à mon formulaire, un slug sera automatiquement généré sur base des éléments que je lui ai fournis.

2.3.3 UTILISATION DE L'ATTRIBUT CASTING

Certains types de champs devront être stockés dans la base de données sous format json (généralement des tableaux, répéteurs…), à la différence que l'on pourra utiliser ses champs comme des objets ou des tableaux. Pour utiliser un attribut casting, c'est très simple. Par exemple, pour ma colonne **record** ou j'aurais besoin d'un répéteur, voici la marche à suivre :

```
protected $casts = array(
    'array' => 'records',
    'array' => 'evolution'
);
```

Par après, j'ai dû créer plusieurs accessors par nécessité. Par exemple, si un utilisateur rentre son prénom et son nom, et qu'il oublie de mettre une majuscule, j'aimerais faire en sorte que le prénom et le nom commencent par une majuscule. J'aimerais également créer un nouveau champ qui n'est pas présent dans ma table pour récupérer directement le nom complet de mon athlète. Voici donc les accessors que j'ai du créées :

```php
public function getFirstNameAttribute($value)
{
    return ucfirst($value);
}

public function getLastNameAttribute($value)
{
    return ucfirst($value);
}

public function getStatusAttribute($value)
{
    return ucfirst($value);
}

public function getProfessionAttribute($value)
{
    return ucfirst($value);
}

public function getFullNameAttribute()
{
    return ucfirst($this->firstname) . ' ' . ucfirst($this->lastname);
}
```

Grâce à l'accessor **getFullNameAttribute**, je pourrais maintenant récupérer le nom complet de mon athlète comme ceci : **{{$ athlète->fullname}}**.

2.3.4 GÉNÉRATION DES IMAGES

La dernière chose qu'il me reste à faire dans mon fichier Model est de générer des images, car chaque athlète aura besoin d'une photo de présentation. Avec Backpack, pour la création d'une image, il faut suivre toute une série d'étapes.

Backpack possède dans ses champs, un champ **image.** Lorsqu'on utilise ce champ, il faut au préalable créer un **mutator** (qui va permettre de charger les données qui seront reçues) afin de stocker les images en base de données. La fonction de départ ressemble à ceci :

```php
public function setImageAttribute($value)
{
    $attribute_name = "image";
    $disk = "public_folder";
    $destination_path = "/athletes";

    // if the image was erased
    if ($value==null) {
        // delete the image from disk
        \Storage::disk($disk)->delete($this->{$attribute_name});

        // set null in the database column
        $this->attributes[$attribute_name] = null;
    }

    // if a base64 was sent, store it in the db
    if (starts_with($value, 'data:image'))
    {
        // 0. Make the image
        $image = \Image::make($value)->encode('jpg', 90);
        // 1. Generate a filename.
        $filename = md5($value.time()).'.jpg';
        // 2. Store the image on disk.
        \Storage::disk($disk)->put($destination_path.'/'.$filename, $image->stream());
        // 3. Save the path to the database
        $this->attributes[$attribute_name] = $destination_path.'/'.$filename;
    }
}
```

L'image va être stockée dans un dossier défini avec un nom unique qui lui sera attribué. Grâce au package **Image,** l'image sera stockée et pourra être utilisée.

Je peux ensuite définir les différentes tailles dont j'aurais besoin, voici le code réalisé pour définir mes tailles d'images. Je vais donc créer différentes tailles de l'image qui seront basées sur l'originale, que je vais redimensionner à des tailles définies grâce à la fonction **fit ()**. Une fois, l'image générée, je lui ajoute un suffixe.

Pour accéder à l'image, j'ai besoin de créer une autre fonction qui aura comme rôle de charger l'image avec le bon suffixe. Voici un exemple :

```
public function getImageProfile($suffix)
{
    $basePath = 'uploads/athletes/';
    $fullname = pathinfo($this->image, PATHINFO_FILENAME);
    $imageProfile = $basePath . $fullname . $suffix;

    if (file_exists($imageProfile)) {
        return URL('/') . '/' . $imageProfile;
    } else {
        return $this->image;
    }
}
```

Grâce à ma fonction créée, je pourrais donc appeler n'importe quelle image avec le suffixe choisi. Dans ma vue d'un athlète, je pourrais faire : **{{$ athlete->getImageProfile (« _profile .jpg »)}}** et cela me retournera mon image avec le bon suffixe.

L'une des dernières étapes est de créer les relations que l'athlète aura. Les athlètes auront des entraîneurs, des disciplines, d'une division, des trophées et des entrainements. J'aurais besoin d'utiliser les relations **belongsToMany** et **BelongsTo**.

```php
public function trainers()
{
    return $this->belongsToMany('App\Models\Trainer');
}

public function disciplines()
{
    return $this->belongsToMany('App\Models\Discipline');
}

public function division()
{
    return $this->belongsTo('App\Models\Division');
}

public function trophies()
{
    return $this->belongsToMany('App\Models\Trophie');
}

public function trainings()
{
    return $this->belongsToMany('App\Models\Training');
}
```

2.3.5 CRÉATION DE LA PAGE D'ADMINISTRATION D'UN ATHLÈTE

Maintenant que le Model est terminé, passons à la réalisation de lu controller d'administration. Dans Backpack, ils sont tous placés dans **App/Http/Controllers/Admin**. Ouvrons maintenant le fichier et regardons ce qui se trouve à l'intérieur.

```php
<?php

namespace App\Http\Controllers\Admin;

use Backpack\CRUD\app\Http\Controllers\CrudController;

// VALIDATION: change the requests to match your own file names if you need form validation
use App\Http\Requests\AthleteRequest as StoreRequest;
use App\Http\Requests\AthleteRequest as UpdateRequest;
use Backpack\CRUD\CrudPanel;

/**
 * Class AthleteCrudController
 * @package App\Http\Controllers\Admin
 * @property-read CrudPanel $crud
 */
class AthleteCrudController extends CrudController
{
    public function setup()
    {
        /*
        |--------------------------------------------------------------------------
        | CrudPanel Basic Information
        |--------------------------------------------------------------------------
        */
        $this->crud->setModel('App\Models\Athlete');
        $this->crud->setRoute(config('backpack.base.route_prefix') . '/athlete');
        $this->crud->setEntityNameStrings('athlete', 'athletes');

        /*
        |--------------------------------------------------------------------------
        | CrudPanel Configuration
        |--------------------------------------------------------------------------
        */

        // TODO: remove setFromDb() and manually define Fields and Columns
        $this->crud->setFromDb();

        // add asterisk for fields that are required in AthleteRequest
        $this->crud->setRequiredFields(StoreRequest::class, 'create');
        $this->crud->setRequiredFields(UpdateRequest::class, 'edit');
    }

    public function store(StoreRequest $request)
    {
        // your additional operations before save here
        $redirect_location = parent::storeCrud($request);
        // your additional operations after save here
        // use $this->data['entry'] or $this->crud->entry
        return $redirect_location;
    }
```

La fonction **setup ()** nous renseigne déjà quelques informations sur le contrôleur, qui utilise le CRUD de Backpack. Premièrement, on lui passe le fichier Model dont il a besoin grâce à la fonction **setModel()**. Ensuite, on lui donne la route que l'on veut afficher pour la page d'administration. Et enfin, on peut modifier les chaînes de caractères qui seront utilisés, pour afficher le titre de la page ou visible dans un bouton pour ajouter un athlète.

Ensuite, nous avons de base une fonction **setFromDb()** qui dit à Backpack de créer des colonnes dans la vue d'administration de toutes les entrées présentes dans la table en base de données, ce qui n'est pas bien, car on aimerait pouvoir afficher les colonnes que l'on souhaite. On verra un peu plus tard comment modifier cela. Pour terminer, nous avons une fonction **store ()** et **update** () qui permet de stocker et de mettre à jour le contenu.

La première chose à faire pour que notre page s'affiche correctement dans notre administration est de créer une route. Artisan va nous aider sur ce coup, car il existe une commande qui va directement aller inscrire le nom de la route dans le fichier **custom.php** qui est le fichier qui stocke toutes les routes de l'administration, elles sont donc séparées des routes plus classiques qui se trouvent dans le fichier **web.php**. Rendez-vous dans le fichier qui se trouve dans **routes/backpack/custom.php**.

```php
<?php

// --------------------------
// Custom Backpack Routes
// --------------------------
// This route file is loaded automatically by Backpack\Base.
// Routes you generate using Backpack\Generators will be placed here.

Route::group([
    'prefix' => config('backpack.base.route_prefix', 'admin'),
    'middleware' => ['web', config('backpack.base.middleware_key', 'admin')],
    'namespace' => 'App\Http\Controllers\Admin',
], function () { // custom admin routes
    CRUD::resource('athlete', 'AthleteCrudController');
    CRUD::resource('trainer','TrainerCrudController');
    CRUD::resource('discipline','DisciplineCrudController');
    CRUD::resource('division','DivisionCrudController');
    CRUD::resource('place','PlaceCrudController');
    CRUD::resource('type','TypeCrudController');
    CRUD::resource('training','TrainingCrudController');
    CRUD::resource('trophie','TrophieCrudController');
    CRUD::resource('internship','InternshipCrudController');
    CRUD::resource('competition','CompetitionCrudController');
    CRUD::resource('article','ArticleCrudController');
    CRUD::resource('author','AuthorCrudController');
    CRUD::resource('amenitie','AmenitieCrudController');
    CRUD::resource('size','SizeCrudController');
}); // this should be the absolute last line of this file
```

Backpack a donc créé un groupe de routes pour que toutes les pages d'administration aient le même préfixe (admin). La route est maintenant opérationnelle, on peut donc y accéder grâce à l'URL **admin/athlete**. On peut également l'ajouter dans le menu de l'administration pour pouvoir y accéder plus facilement (ressources/views/vendor/backpack/base/inc/sidebar.blade.php).

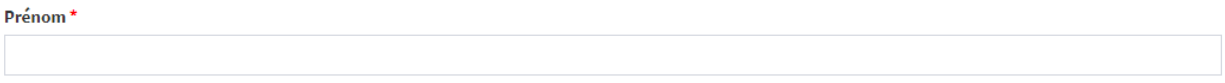
Nous pouvons maintenant accéder à notre page. Elle est pour l'instant vide et remplie avec toutes les colonnes de notre table. Retirons la fonction **setFromDb()** dont j'ai parlé plus haut et ajoutons les champs dont nous avons réellement besoin.

Backpack propose actuellement 44 types de champs différents. Par exemple pour ajouter un champ de type TEXT :

```
// Firstname Field
$this->crud->addField(
    [
        'name' => 'firstname',
        'type' => 'text',
        'label' => 'Prénom',
        'tab' => 'Informations générales'
    ]
);
```

Comme on peut le voir, pour créer un champ on appelle la fonction **addField()** de CRUD, on va lui passer un **nome** qui sera le nom de la colonne dans votre base de données, un **type** pour le type de champ, un **label** pour l'intitulé du champ au-dessus, et éventuellement d'autres champs comme : **tab**, **hint**...

Voici le résultat de ces lignes de code :



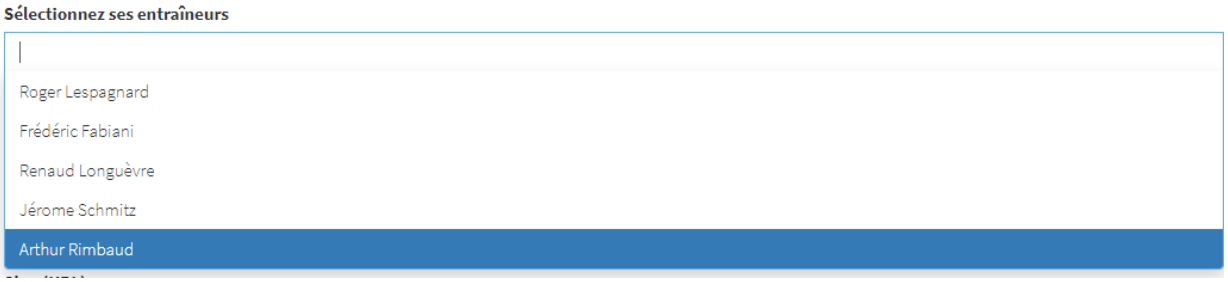
C'est donc très simple de créer un champ comme on peut le constater.

Voici maintenant un aperçu des champs utilisés pour la création d'un athlète que nous allons analyser maintenant :

```
// Active Field
$this->crud->addField(
    [
        'name' => 'active',
        'type' => 'select_from_array',
        'options' => ['actif' => 'Actif', 'inactif' => 'Inactif'],
        'allows_null' => false,
        'default' => 'actif',
        'label' => 'Activité de l\'athlète',
        'tab' => 'Informations générales'
    ]
);

// Trainer field
$this->crud->addField
([
    'label' => 'Sélectionnez ses entraîneurs',
    'type' => 'select2_multiple',
    'name' => 'trainers',
    'entity' => 'trainers',
    'attribute' => 'fullname',
    'model' => "App\Models\Trainer",
    'pivot' => true,
    'tab' => 'Informations générales'
]);
```

Regardons le champ « Trainer », il s'agit d'un champ **select2_multiple**. Ce champ va surtout être utile pour associer un entraîneur à un athlète. Je vais donc lui passer un **nome** qui sera le nom de la relation entre les deux tables, un **attribut** qui sera le nom d'une colonne présente dans la sa table pour afficher l'information voulue pour ce Model. Dans mon cas, j'ai créé un attribut **fullname** afin d'afficher directement le **firstname** et le **lastname** d'un athlète dans l'administration. On lui passe également un **Model**, en l'occurrence ici, celui d'un entraîneur et on précise s'il y a une table pivot entre les deux, afin de savoir si l'athlète peut choisir plusieurs entraîneurs. Ce qui est mon cas ici. Voici donc à quoi ressemble le champ, une fois terminé :



2.3.6 GESTION DES ERREURS

Notre page d'administration est créée, mais il serait bien de pouvoir gérer les erreurs de ses champs. Lors de la génération du CRUD, un fichier **AthleteRequest** a été généré, c'est donc là que nous allons gérer nos erreurs. On lui déclare les règles de validation dans un tableau et voilà.

```php
public function rules()
{
    return [
        'firstname' => 'required|min:2|max:255',
        'lastname' => 'required|min:2|max:255',
        'image' => 'required',
        'status' => 'required|min:2|max:50',
        'profession' => 'required|min:2|max:50',
        'active' => 'required|min:2|max:50',
        'date_of_birth' => 'required|date',
        'slug' => 'unique:athletes,slug,' . \Request::get('id'),
    ];
}
```

2.3.7 CRÉATION D'UN CONTRÔLEUR

Nous arrivons à la dernière étape de la création d'un athlète. Maintenant, il ne nous reste plus qu'à générer notre contrôleur pour pouvoir afficher les données dans la vue.

Pour créer mon contrôleur, j'ai utilisé la commande artisan suivante :

- **PHP artisan make:controller Athlete**

Une fois le contrôleur généré, je remplis ma fonction **index ()** afin de pouvoir afficher la liste de mes athlètes.

```
public function index()
{
    $page = Page::where('template', 'athletes_index')->firstOrFail();
    $this->data['athletes'] = Athlete::orderBy('lastname', 'ASC')->get();

    $this->data['title'] = $page->title;
    $this->data['page'] = $page->withFakes();

    return view('pages.athletes.' . $page->template, $this->data);
}
```

J'ai utilisé **PageManager** pour afficher mes pages comme vous pouvez le voir. Ce n'était pas une obligation de l'utiliser, mais j'ai dû en avoir recours pour des pages par exemple la page A Propos, ou je devais créer des champs **fake fields** afin que l'utilisateur puisse modifier à sa guise le contenu de la page.

Je crée donc une variable **$ page** ou je vais chercher le nom du template que j'ai créé pour cette page. Je récupère ensuite les données des athlètes que je trie avec le nom par ordre croissant et je retourne finalement la vue qui va servir à afficher mes athlètes.

Passons maintenant à la fonction **show ()** qui gérer l'affichage d'un seul athlète.

```
public function show(Athlete $athlete)
{
    return view('pages.athletes.athletes_show', [
        'athlete' => $athlete
    ]);
}
```

Je passe comme argument à ma fonction la Model **Athlète** auquel j'assigne une variable. Je retourne ensuite simplement la variable **$ athlete,** afin de pouvoir accéder aux données de l'athlète.

2.4 CRÉATION D'UN SYSTÈME DE RECHERCHE

Je voulais que mes athlètes, entraîneurs ou autres puissent bénéficier d'un système de recherche afin de faciliter la recherche d'un athlète par exemple. J'ai donc opté pour un système de recherche avec des filtres.

2.4.1 LA FONCTION FILTER

La première chose que j'ai faite a été de créer une fonction **filter ()** afin de pouvoir récupérer les éléments utiles au triage des athlètes.

```
public function filter(Request $request)
{

    $query = Athlete::query();
    if ($request->has('discipline')) {
        $query->whereHas('disciplines', function ($query) use ($request) {
            $query->where('slug', $request->get('discipline'));
        });
    }
    if ($request->has('division')) {
        $query->whereHas('division', function ($query) use ($request) {
            $query->where('slug', $request->get('division'));
        });
    }

    if ($request->has('status')) {
        $query->where('status', $request->get('status')
        );
    }


    $athletes = $query->paginate(3);

    if ($request->ajax()) {
        return [
            'athletes' => view('partials.single.athlete.item_athlete',
                [
                    'athletes' => $athletes
                ])->render(),
            'next_page' => $athletes->nextPageUrl()
        ];
    }

    return view('pages.athletes.athletes-filter',
        [
            'athletes' => $athletes
        ]);
}
```

Donc, je crée une variable **$ query** pour récupérer la query d'un athlète. Je souhaite donc pouvoir filtrer mes athlètes par leurs disciplines, leur division et leur statut.

Je démarre une condition, ou je dis que s'il y a une requête qui contient une discipline, je vais chercher les query avec toutes les disciplines et je récupère le slug de la discipline.

Je crée ensuite une variable **$ athlètes** pour définir le nombre d'athlètes à afficher par page. Si la requête est en ajax, je retourne la vue avec le **foreach** qui boucle tous les athlètes et je fais le rendu. Une fois la requête ajax vérifié, je renvoie vers la vue filtrée des athlètes.

2.4.2 RÉALISATION DU SELECT

La fonction **filter ()** terminée, j'ai besoin de plusieurs select afin de lister les disciplines, divisions pour pouvoir filtrer les athlètes.

```
<form action="{{ Request::url() }}" method="get" class="switcher__form">
    <div class="switcher__container">
        <div class="switcher__bloc">
            <label for="discipline" class="switcher__label">Discipline(s)</label>
            <select name="discipline" id="discipline" class="switcher__select">
                <option value="all">Toutes</option>
                @foreach($disciplines as $discipline)
                    <option <?php echo (Request::get('discipline') == $discipline->slug) ? 'selected' : '' ;?> value="{{ $discipline->slug
                @endforeach
            </select>
        </div>
        <div class="switcher__bloc">
            <label for="division" class="switcher__label">Catégorie(s)</label>
            <select name="division" id="division" class="switcher__select">
                    <option value="all">Toutes</option>
                    @foreach($division as $division)
                        <option <?php echo (Request::get('division') == $division->slug) ? 'selected' : '' ;?> value="{{ $division->slug }}
                    @endforeach
            </select>
        </div>
        <div class="switcher__bloc">
            <label for="status" class="switcher__label">Statut</label>
                <select name="status" id="status" class="switcher__select">
                <option <?php echo (Request::get('status') == '') ? 'selected' : '' ;?> value="all">Tous</option>
                <option <?php echo (Request::get('status') == 'novice') ? 'selected' : '' ;?> value="novice">Novice</option>
                <option <?php echo (Request::get('status') == 'intermediaire') ? 'selected' : '' ;?> value="intermediaire">Intermédiaire</o
                <option <?php echo (Request::get('status') == 'haut-niveau') ? 'selected' : '' ;?>  value="haut-niveau">Haut Niveau</option
            </select>
        </div>
    </div>
```

Analysons par exemple, le filtre pour les **disciplines** qui est une relation à un athlète. Pour lister les disciplines rien de plus simple, je boucle les disciplines en disant qui la discipline correspond au slug de la discipline, je sélectionne la discipline sinon, la discipline n'est pas sélectionnée.

Pour le statut d'un athlète, c'est un peu différent comme il s'agit de donner statiques stockées dans un tableau. Je suis obligé de lister directement tous les statuts en disant que si le statut est par exemple « novice » que novice soit sélectionné.

J'ai également besoin de définir une route pour afficher la vue filtrée, une fois la recherche lancée.

```
Route::get('athletes/filter', 'AthleteController@filter')->name('athletes-filter');
```

2.4.3 LA PAGINATION

Pour la pagination, j'ai opté pour un « load-more ». J'ai choisi d'afficher un nombre défini d'athlètes par défaut et de rajouter un nombre défini d'athlètes lorsque je décide de charger plus d'athlètes.

Pour ce faire, j'ai créé une fonction **getLoadMoreLink()** pour définir une URL pour la query.

```php
public function getLoadMoreLink(Request $request) {

    $querystring = '';

    if ($request->has('discipline') ) {
        $querystring .= '&discipline=' . $request->get('discipline');
    }

    if ($request->has('division') ) {
        $querystring .= '&division=' . $request->get('division');
    }

    if ($request->has('status') ) {
        $querystring .= '&status=' . $request->get('status');
    }

    return $querystring;
}
```

Je crée une variable vide **$ querystring** qui contiendra l'URL. Ensuite, je dis que si la requête contient une discipline, je concatène à la variable **$ querystring** la chaîne « & discipline » concaténée par le nom de la discipline choisie et je fais de même pour les 2 autres. Je retourne donc la variable qui me retournera la requête complète.

J'avais besoin de créer cette fonction car lorsque je cliquerais sur « Charger plus », il faudra que la recherche soit prise en compte et que je puisse concaténer la requête sur la prochaine page et ainsi de suite.

CONCLUSION

Je pourrais parler de tout ce que j'ai mis en place pendant encore un moment, mais le temps me manque.

Durant ce projet, j'ai appris énormément de choses. Je pense avoir rempli l'objectif que je m'étais fixé pour le projet avec le temps que j'avais. Si la matière que j'avais dû apprendre ici, je le connaissais déjà avant de commencer le projet, je pense que mon projet serait encore beaucoup plus abouti. Mais, j'ai voulu me lancer dans quelque chose que je ne connaissais pas beaucoup. Quel est l'intérêt de faire un site sur une technologie que l'on sait déjà très bien manié. C'est là qu'a été mon challenge d'apprendre à utiliser ces nouveaux outils.

Je dois l'avouer, le back -end n'a jamais vraiment été mon fort, j'ai toujours été plus attiré par de l'intégration ou de la réalisation de maquettes graphiques. Mais, ce projet m'a fait changer de point de vue. La plupart du temps, lorsque je voulais commencer quelque chose, je n'avais pas la moindre idée de comment m'y prendre. Le fait d'apprendre à chercher par moi-même dans la documentation m'a vraiment aidé. La meilleure façon d'apprendre c'est comme je dirais « d'apprendre sur le tas ».

Je regrette de ne pas avoir eu plus de temps, j'aurais tellement aimé rajouter tout un tas de fonctionnalités, car les idées ne me manquent pas. Mais le temps manque malheureusement. Je compte bien continuer ce projet dans le futur et essayer de faire quelque chose avec cette application.

J'espère cependant vous avoir plongé un peu plus dans mon site web et que je n'aurai pas trop été monotone. Je n'ai jamais été très doué pour expliquer les choses, c'est un de mes défauts, mais j'ai tâché de faire de mon mieux.

Pour conclure, je pense avoir rempli mon objectif et avoir fourni un travail propre et utilisable facilement. J'aurais stressé, travailler comme un possédé, mais je ne regrette pas l'expérience que ce projet aura pu m'apporter et je pense que j'ai à présent suffisamment de bagages sur moi pour commencer ma vie dans le monde professionnel.

BIBLIOGRAPHIE

https://laravel.com/, le site officiel du **Framework Laravel**.

https://backpackforlaravel.com/, le site officiel de **Laravel Backpack**.

laravelsd.com, pour créer un schéma de sa base de données.

https://mjml.io, le site officiel du Framework **Mjml**.

https://stackoverflow.com/, un site dédié à une communauté de développeur.

https://www.pinterest.fr/, pour l'inspiration graphique.

https://www.behance.net/, pour l'inspiration graphique.