

CATÉGORIE TECHNIQUE
Parc des Marêts - Rue Peetermans, 80 - 4100 Seraing

SITE WEB PROMOTIONNEL D'UN BACHELIER EN INFOGRAPHIE



Par Jimmy Letecheur

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



<http://ecoleinfographie.be>

REMERCIEMENTS

Je remercie Laura, ma compagne, pour le soutien infaillible dont elle a fait preuve à mon égard et pour l'aide qu'elle m'a apportée.

Je remercie ma famille pour leur soutien et leurs encouragements.

Je remercie l'équipe de WhiteCube et Toon Van den Bos pour les conseils avisés qu'ils ont pu me donner.

Je remercie mes professeurs pour la formation apportée durant ces 3 années.

Je remercie Dominique Vilain, mon superviseur, pour avoir toujours pris le temps de répondre à mes questions et pour ses nombreux conseils.

Introduction	5
Partie 1 : cahier des charges	8
1. Introduction.....	9
2. Définition du produit	9
3. Objectifs	9
3.1. Principaux.....	9
3.2. Secondaires.....	10
4. Techniques utilisées	10
Partie 2 : Réalisation du design	12
1. Introduction.....	13
2. Création de la structure	13
3. Recherche d'inspirations	14
4. Wireframes	16
5. Création de la maquette graphique.....	18
5.1. Typographie.....	18
5.2. Grille	19
5.3. Création du design.....	19
5.4. Outils utilisés	23
Partie 3 : Développement	24
1. Développement HTML et accessibilité	25
1.1.Un menu clair, sémantique et accessible	25
1.2.Baliser correctement un fil d'ariane	30
1.3.des microdatas pour tout.....	33
2. Développement CSS.....	34
2.1.Technologies utilisées	35
2.2.Architecture	35
2.3. Print CSS.....	36
3. Développement JavaScript	37
3.1. Technologies utilisées	37
3.2. Structure des fichiers.....	38
3.3. Choisir de bons plug-ins.....	41

4. Développement côté serveur	42
4.1. Avant de commencer.....	42
4.2. Schématisation de la base de données	43
4.3. Présentation de Laravel Backpack.....	47
4.4. Crédit d'un premier CRUD	49
4.5. Réalisations des étudiants	64
4.6. Crédit d'un blog et autre fonctionnalités	65
Conclusion	84
Bibliographie	86

INTRODUCTION

J'ai longtemps hésité sur le choix de ma formation. Bien avant d'arriver en infographie, je me demandais ce que j'allais faire, je me suis orienté vers différents domaines, la communication, le marketing, l'e-business, la vidéo et tout cela en l'espace de 2 ans. J'étais perdu et je ne trouvais pas ma voie. C'est alors que j'ai décidé de tenter une dernière expérience, celle de l'infographie. Je ne savais pas vraiment dans quoi je m'engageais, j'avais toujours eu une attirance particulière pour les belles choses et la communication visuelle.

Avant mon inscription en infographie, j'ai fait beaucoup de recherches pour avoir des informations, j'ai commencé à chercher sur Google via différents mots clés, comme **école d'infographie, école de web, formation en web, bachelier en web**, etc. Et je me souviens encore des sites que Google m'a proposés.

Le premier était le site de la Haute École Albert Jacquard. Le deuxième était celui de la Haute École de la Province de Liège sur la page « Bachelier en Techniques graphiques ».

J'ai épluché de fond en comble le site de la H.E.A.J., je savais que je voulais faire du web, même si je ne fermais pas la porte aux autres options. Mais je voulais vraiment avoir une bonne formation en web. J'ai continué ma visite, j'ai lu, j'en ai appris un peu, mais pas grand-chose. Je savais que j'allais faire du web, j'avais une liste de cours, mais je n'avais pas vraiment d'explications. Globalement, j'avais eu une bonne impression sur le web, mais en n'ayant rien appris de plus. J'étais plus inquiet de me tromper une nouvelle fois, alors que j'aurais dû être persuadé par ce que je venais de lire.

Ensuite, j'ai parcouru la page du site de la H.E.P.L, et j'ai vraiment été déçu.

Un petit texte, qui tient sur 400 mots, et qui nous explique tout sauf ce que l'on va pouvoir devenir. Pourtant, c'est quand même la base d'une formation : exercer un métier.

On n'y parlait pas vraiment de web, mais plus de multimédias (et la création de CD-ROM).

En lisant ça, j'étais vraiment inquiet. Je me souviens encore de ma réflexion « Faire des CD-ROM, en 2014 ? », je n'étais vraiment pas partant.

J'ai quand même consulté la grille des cours, mais elle n'était pas très claire, et la fiche d'un cours était assez complexe et n'allait pas à l'essentiel. Je suis sorti de cette visite, plus perdu que jamais.

Je me voyais m'inscrire à la H.E.A.J. plutôt qu'à la H.E.P.L. au vu des informations recueillies. Il faut dire qu'une formation en infographie qui prétend nous apprendre des règles pour réaliser de belles choses et qui comporte une formation en web, mais qui n'a aucune communication (qu'elle soit visuelle ou basée sur l'information), ça ne prête pas envie de tenter l'expérience.

Cependant connaissant déjà la H.E.P.L, je me doutais que c'était une communication unie pour toutes les options et qui ne laissait pas place à la personnalisation. J'ai eu cette réflexion, mais beaucoup ne l'auraient peut-être pas eue.

Dès lors, j'ai à nouveau demandé à Google de me fournir des résultats sur base d'autres mots clés, tels que : « **avis haute école liège infographie, web hepl ou heaj, meilleure formation hepl ou heaj**, etc. ».

Je suis tombé sur des forums de discussions où des élèves exprimaient leur ressenti. Et il en ressortait nettement que la formation en web à la H.E.P.L était plus poussée, et de meilleure qualité, qu'à la H.E.A.J. J'ai lu plusieurs sujets comme celui-là et au fur et à mesure que j'avancais dans ma lecture, mon avis changeait. J'ai pris le temps de la réflexion pendant plusieurs semaines et j'ai finalement décidé de m'inscrire à la H.E.P.L. Et certainement pas à cause de leur site web, mais parce que les avis de personnes ayant vécu l'expérience de la formation étaient très positifs. Si je n'avais pas vu cela, j'aurais certainement continué mes études à Namur.

Je parle de tout cela, car c'est ce qui m'a poussé à réaliser mon travail de fin d'études. Un site web sert à communiquer, et une école d'infographie qui ne peut pas communiquer, c'est un peu comme pécher sans hameçon, on veut de nouveaux élèves, mais on ne sait pas les attirer.

Présenter des travaux, les cours de manière concrète, c'est tout de même un gros plus. Et le fait de ne pas avoir de site web montre aussi un manque de vie à travers l'établissement, qui est caché derrière la grande enseigne qu'est la Haute École de la Province de Liège. Tout semble froid et impersonnel, et en infographie, on n'aime pas ça.

J'ai proposé comme travail de fin d'études de réaliser une première version du site web de la formation en infographie. Par envie, car j'apprécie réaliser des sites web, mais aussi parce que je juge vraiment nécessaire le fait de promouvoir les différentes formations que l'on propose.

À travers ce document, vous allez pouvoir analyser les différentes parties de sa conception et les différentes réflexions qui m'ont amené à faire certains choix. Je n'ai pas la prétention à avoir réalisé le produit parfait, loin de là, il est encore perfectible sur beaucoup de points, car un site web n'est jamais parfait. Mais, j'y ai mis toute mon envie, mes compétences et mon temps pour tenter de délivrer un produit qui respecte un cahier des charges complet afin de donner satisfaction à mon « client » et d'apporter une première pierre à l'édifice du tant attendu, site web de l'infographie à Liège.

PARTIE 1 : CAHIER DES CHARGES

1. INTRODUCTION

Avant tout projet, il est essentiel de déterminer plusieurs objectifs afin d'avancer dans la bonne direction. C'est pourquoi la bonne démarche est de réaliser un cahier des charges complet et structuré pour établir le produit.

2. DÉFINITION DU PRODUIT

Le produit consiste en la réalisation d'un site web qui va promouvoir la formation supérieure en infographie dispensée à Seraing. Toute fois, il faut arriver à se distinguer de la Haute École de la Province de Liège dans la communication visuelle et textuelle, car il n'a pas pour but de communiquer de manière officielle, mais d'informer plus précisément sur la formation et sur ce qu'il se passe à l'école.

3. OBJECTIFS

J'ai défini différents objectifs, des principaux et des secondaires. Les objectifs principaux (macro utilités) sont ceux qui représentent le site web et sa nature d'être. Tandis que les objectifs secondaires (micro utilités) sont ceux qui vont approfondir l'expérience par des contenus supplémentaires.

3.1. PRINCIPAUX

- **Présenter les différents métiers** : expliquer et faire découvrir ce qu'il est possible de devenir en accomplissant l'une des formations dispensées.
- **Présenter les formations** : présenter concrètement les différentes formations qu'il est possible de réaliser en présentant le programme et les cours de manière détaillée et personnelle en utilisant des termes adaptés, motivants et convaincants ainsi que de lier au cours, les professeurs.
- **Présenter les réalisations des étudiants** : en infographie, les étudiants sont amenés à réaliser différents travaux au cours de leur cursus. Afin de mieux se rendre compte des compétences qu'il est possible d'acquérir, il est nécessaire de promouvoir les travaux réalisés au cours de la formation.
- **Informations sur l'établissement** : Le site étant créé pour présenter la formation, il faut pouvoir y retrouver différentes informations, comme celles de contact ou une page proposant l'inscription. Il est aussi important de pouvoir suivre l'actualité de l'établissement afin de donner plus de

personnalités et de donner une impression de vie active ainsi que de pouvoir annoncer les différents évènements.

3.2. SECONDAIRES

- **Mettre à l'honneur les différents diplômés** : tenir une archive, comme un livre d'or, des différentes promotions afin d'avoir une trace dans le temps de tous ceux qui ont été diplômés.
- **Être un outil de veille** : Réalisation d'un blog de veille sur l'infographie afin de pouvoir écrire des articles qui permettront aux élèves et aux membres externes de se référer au site pour apprendre de nouvelles choses. Le blog pourrait aussi être utilisé pour écrire des cours accessibles en ligne.
- **Proposer des offres de stage** : le site aura un espace dédié aux entreprises qui désirent poster une offre de stage en ligne afin qu'elle puisse être transmise aux élèves.

4. TECHNIQUES UTILISÉES

Pour ce projet, j'ai décidé d'utiliser différentes technologies, dont certaines que je détaillerais un peu plus loin. Voici ce qui a été utilisé pour la réalisation de ce projet :

- **HTML/CSS** : J'ai utilisé les incontournables du web, à savoir le langage **HTML** pour structurer sémantiquement mes pages et le **CSS** pour le mettre en forme visuellement. J'ai accordé une attention particulière à l'utilisation des microdatas, que je détaillerais dans le chapitre approprié. J'ai utilisé des propriétés CSS compatibles avec Internet Explorer 9, afin de le rendre compatible avec un grand nombre de navigateurs. Et, à chaque fois que j'utilisais des propriétés non supportées, je prévoyais un fallback afin que le rendu soit correct.
- **JavaScript** : La modernité oblige, le site contiendra du **JavaScript** afin de tenter d'améliorer l'expérience utilisateur et visuelle. Néanmoins, il fonctionnera entièrement sans, ce qui permettra aux différents types d'utilisateurs de l'utiliser.
- **Laravel et Backpack** : J'ai décidé d'utiliser le framework **Laravel** pour intégrer la logique serveur de mon projet. Comme je désirais quelque chose de souple, je ne pouvais pas m'orienter vers un CMS classique. Cependant, j'ai décidé d'utiliser en parallèle un package (ou plutôt, un groupe de packages) qui s'appelle **Backpack**,

qui permet de gérer l'administration simplement et de créer son propre **CMS** en **Laravel**. Je détaillerais plus loin cette technologie.

PARTIE 2 : RÉALISATION DU DESIGN

1. INTRODUCTION

Une fois, les objectifs du site web défini, j'ai dû commencer la phase de la création graphique. Pour ce faire, je suis passé par différentes phases. J'ai d'abord commencé par chercher des sources d'inspirations, j'ai ensuite créé des maquettes fonctionnelles appelées wireframes, afin de structurer la façon schématique dont les informations vont apparaître sur les pages. Après cela, j'ai commencé la phase de conception, où, sur base de mes wireframes, j'ai réalisé les maquettes graphiques qui détermineront l'aspect visuel du site web.

2. CRÉATION DE LA STRUCTURE

Avant de commencer à réaliser mon site web et à entrer dans la phase suivante, la première étape consiste à imaginer la structure (les différentes pages) que comportera mon site. Voici ce que j'ai imaginé :

- **Page d'accueil**
 - Introduction
 - Dernière actualité
 - Derniers articles de blog
 - Mise en avant de 3 réalisations d'étudiants
 - Présentation de trois professionnels qui racontent leur parcours
- **Les métiers du web/de la 3D et audiovisuel/du design graphique**
 - Présentation des différents métiers qu'il est possible de devenir en choisissant une orientation
 - La formation : présentation la formation et des cours d'option
 - Le programme : la liste des cours
 - Possibilité de cliquer sur un cours pour accéder à sa fiche
 - Le parcours des anciens étudiants
 - Accès à l'interview d'un ancien étudiant
- **L'école**
 - L'actualité : liste des articles et vue d'un article
 - Les commodités : les facilités qu'offre l'école (restaurant, aux alentours, etc.)
 - Le matériel et les bâtiments : présentation des équipements, etc.

- Contactez-nous : une page pour contacter l'établissement et obtenir diverses informations de localisation.
- **Réalisations** : liste des travaux d'étudiants et page vue d'une réalisation.
- **Nos diplômés** : liste des diplômés, s'il existe un portfolio de disponible ou une interview, possibilité de la retrouver en cliquant sur un lien.
- **Le blog** : réalisation d'un blog pour pouvoir y poster des articles de veille, de cours, etc.
- **Inscription** : Une page qui donne les modalités pour s'inscrire.
- **La liste des professeurs** : possibilité de cliquer sur un professeur et d'afficher sa fiche.
- **Proposer une offre de stage** : deux formulaires permettant de proposer un stage aux étudiants

3. RECHERCHE D'INSPIRATIONS

La recherche d'inspiration est l'une des phases essentielles d'un bon design.

Techniquement, on peut construire un design de site web sans prendre d'inspiration, mais s'inspirer de ce qui a déjà été créé et de ce qui fonctionne est non négligeable. D'autant que ça donne des idées, et que le syndrome de la page blanche n'existe pas que chez les écrivains. Je décide toujours de prendre des inspirations avant de réaliser mes wireframes. Beaucoup font l'inverse, moi j'estime que même si les wireframes sont là pour la structure, il est important d'avoir une base minimum présentable au client. Et pour avoir une base présentable, il faut au moins voir ce qu'il se fait dans le domaine et chez la concurrence afin de pouvoir faire quelque chose de bien.

Pour rechercher de l'inspiration, j'ai utilisé différentes ressources que j'énumère ci-dessous :

- **Pinterest** : C'est mon site favori pour rechercher des inspirations. Il est très populaire et beaucoup de designers postent leur travail sur cette plateforme.
- **Behance** : C'est un autre site qui propose aux designers de partager leurs créations. Il est créé par Adobe.
- **line25.com** : C'est un site qui propose régulièrement des regroupements selon un thème de site web intégré.
- **Google** : Vous connaissez certainement, j'utilise le moteur de recherche pour chercher après des concurrents et analyser ce qu'ils font de bien (et de mal).

Je ne vais pas mettre une liste d'images des inspirations, d'une part parce que ça prendrait beaucoup de place (j'en ai énormément) et d'une autre, parce qu'il y en a beaucoup que je n'ai pas utilisé. Mais je vais vous montrer quelques composants qui ont déterminé certaines parties de mon site web.

- [**stripe.com**](#) : Stripe a été une très bonne source d'inspiration pour mon projet. La qualité de ce site est incroyable, et j'ai notamment essayé de faire un menu dropdown dans leur style, je ne voulais pas d'une simple liste déroulante et utiliser des blocs en pente.

The screenshot shows the Stripe homepage with a blue header and footer. The main content area is white with a grid of six cards:

- PAIEMENTS**: Includes a blue camera icon and a description: "Une boîte à outil complète pour le commerce, construite pour les développeurs."
- ABONNEMENTS**: Includes a green circular icon and a description: "Le moteur des paiements récurrents."
- CONNECT NOUVEAU**: Includes a grey square icon and a description: "Tout ce dont les places de marché ont besoin pour payer les vendeurs."
- RELAY**: Includes an orange circular icon and a description: "Vendez vos produits au sein d'autres applications mobiles."
- ATLAS**: Includes a yellow triangle icon and a description: "Une nouvelle manière de créer une entreprise sur Internet."
- RADAR**: Includes a purple circular icon and a description: "Des outils modernes de prévention de la fraude, parfaitement intégrés à vos paiements."

At the bottom left is a blue icon with a book and the text **WORKS WITH STRIPE**, and at the bottom right is a small "POURQUOI" link.

EXEMPLE D'UN MENU DROPDOWN DISPONIBLE SUR STRIPE

- [msichicago.org](#) : je trouve le site du musée de Chicago très beau, j'ai voulu reprendre la façon dont ils utilisent de grands titres. Je me suis aussi inspiré de leur footer pour créer le mien.

The screenshot shows the footer area of the MSICHICAGO.ORG website. At the top, there are two buttons: "Subscribe to our newsletter" and "Register". To the right are four social media icons: Facebook, Twitter, Instagram, and YouTube. Below these are four main navigation categories: "Support Us", "Visit", "Explore", and "Location + Hours".

Support Us	Visit	Explore	Location + Hours
Help transform lives through the power of science and science education. › Become a member › Donate now	Plan a Visit Ticket Prices Map Groups + Field Trips Host an Event Museum Store	What's Here Education Experiment About Us Careers Press	5700 S Lake Shore Drive Chicago, IL 60637 Open today from 11:00 a.m. to 5:30 p.m. Getting Here

FOOTER DU SITE MSICHICAGO.ORG

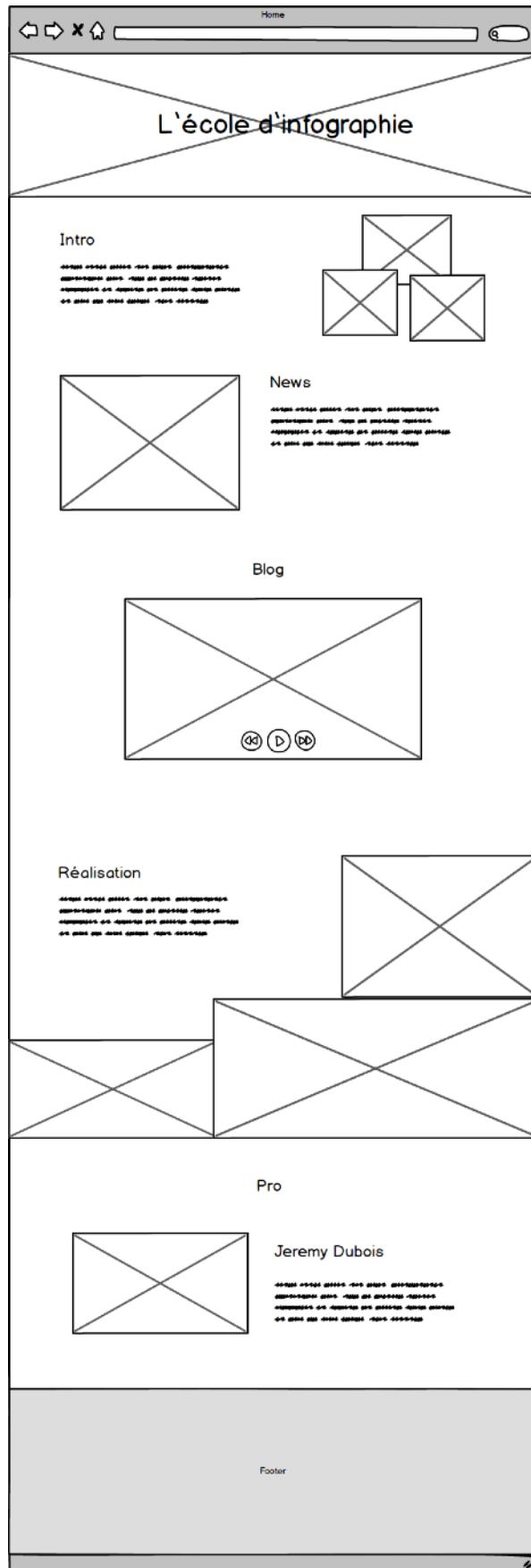
J'ai également été sur les sites des autres établissements scolaires. Je n'ai pas trouvé grand-chose d'intéressant. La plupart étant mal réalisés et les autres ne correspondant pas à ce que je recherchais.

Après la collecte de toutes ces inspirations, j'ai pu commencer la phase suivante, les wireframes.

4. WIREFRAMES

J'ai réalisé des wireframes à l'aide de Balsamiq Mockup, qui est un logiciel qui permet de créer simplement et rapidement des maquettes graphiques. Cependant, après avoir pris du recul avec le stage que j'ai réalisé, je ne pense pas que j'utiliserais ce logiciel pour les présenter au client. Il est bien pour aller vite, mais ce que l'on délivre est peut-être un peu trop schématique et peut sembler amateur.

Les wireframes m'ont permis d'imaginer plus précisément la structure de mon site et des différentes relations que j'aurais entre les pages. Voici par exemple, le wireframe que j'ai réalisé pour la page d'accueil.



Comme on peut le voir, c'est très schématique. En comparant mon design actuel, on y retrouve les grandes idées, même si beaucoup de choses ont changé. Les wireframes sont une première phase de conception, pour imaginer les différentes zones. Ce n'est que par après que l'on commence à réaliser concrètement nos idées. Et c'est la phase suivante.

5. CRÉATION DE LA MAQUETTE GRAPHIQUE

C'est la partie la plus compliquée. Pour créer mon design, j'ai dû prendre en compte tout le travail réalisé avant. J'ai d'abord tenté de réaliser certaines choses. Je suis passé par plusieurs phases, bonnes et moins bonnes.

5.1. TYPOGRAPHIE

J'ai commencé par chercher la bonne typographie. Il en existe énormément. De très belles et de très moches. Après de nombreux tests, j'ai opté pour Catamaran. C'est d'ailleurs avec elle que j'écris ce rapport. Elle dispose de tous les styles possibles. D'extra-léger à noir. Elle présente l'avantage d'être open source, donc gratuit.

Catamaran ExtraBold

Je n'utilise que cette police-là sur tout le site. Elle dispose de tellement de graisses différentes qu'il y avait moyen de réaliser plein de choses. Son défaut, c'est qu'elle ne dispose pas d'italique.

J'ai ensuite décidé d'utiliser un scale pour mes titres, afin d'avoir une font -size cohérente à l'œil. J'ai utilisé le site modularscale.com. Cependant, j'ai vite remarqué qu'il ne fallait pas s'y fier à 100 %. C'est ce que l'on m'a fait remarqué lors de mon évaluation au jury du cours de projet. J'ai donc ajouté quelques variations, afin d'avoir un rendu plus en harmonie avec ce que je voulais présenter.

51.88px
41.5px
33.2px
26.56px
21.25px
18px
17px
13px

5.2. GRILLE

Après avoir défini mes tailles de polices, j'ai décidé de créer une grille. En voici les paramètres :

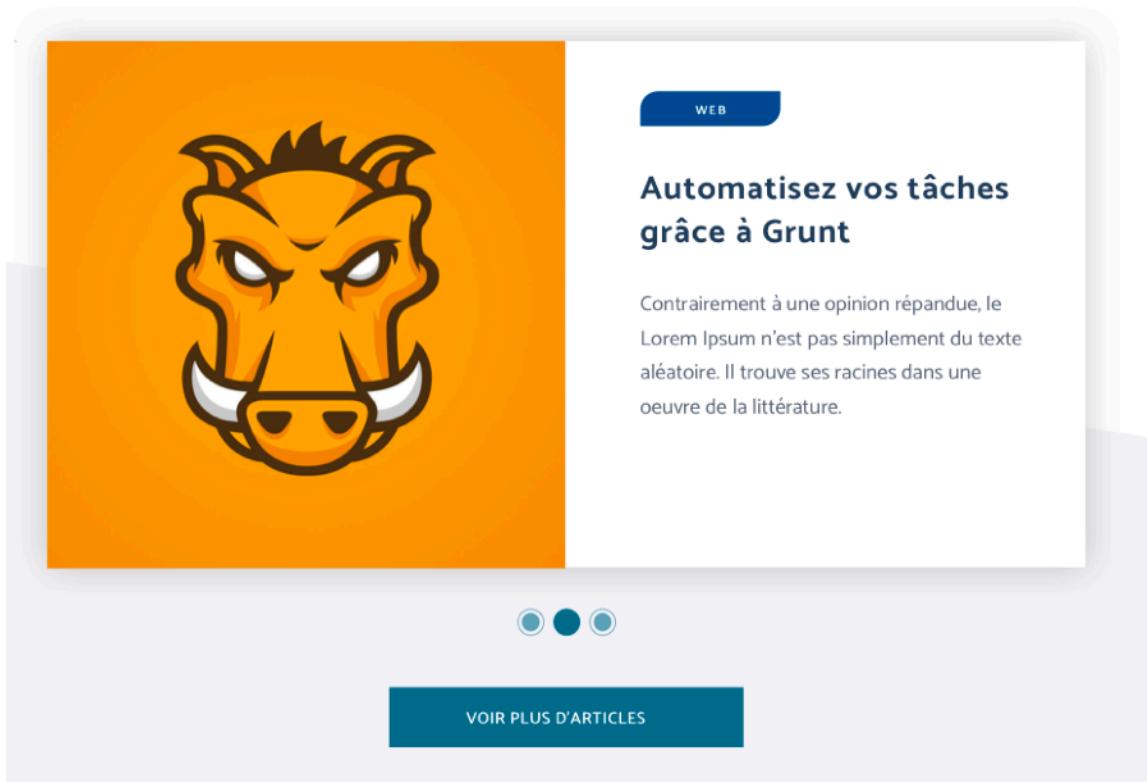
Largeur totale	Nombre de colonnes	Largeur gouttières	Largeur colonnes
1260px	12	20px	85px

J'ai choisi ces paramètres, car le design que je voulais réaliser s'y prêtait. Je souhaite quelque chose de moderne et de large. La plupart des écrans de bureaux rendent au moins 1280px. Dès lors, j'ai voulu m'accorder le maximum d'espace disponible afin de pouvoir afficher un maximum d'informations de manière claire et lisible.

5.3. CRÉATION DU DESIGN

Une fois que j'avais tous mes paramètres, j'ai commencé à intégrer mes idées. Ce fut une partie très longue et qui a reçu de nombreux changements. J'ai fait plusieurs tests, certains qui plaisaient, d'autres moins. L'une des pages les plus compliquées fut la page d'accueil. Il y avait beaucoup d'informations à y intégrer, venant de différents endroits du site web. Il devait y avoir une place pour les actualités, pour les articles de blog, pour les réalisations et pour une sélection de parcours d'anciens étudiants. J'ai dû imaginer différents styles. J'ai également commis des erreurs et j'ai dû revenir sur mon design à plusieurs reprises. Par exemple, j'avais créé un slider pour les articles de blog, mais les personnes à qui j'ai fait tester mon site m'ont dit que ça ne leur plaisait pas, qu'ils auraient aimé

avoir un aperçu de tous les articles d'un coup, plus tôt que de devoir slider entre les différents éléments.

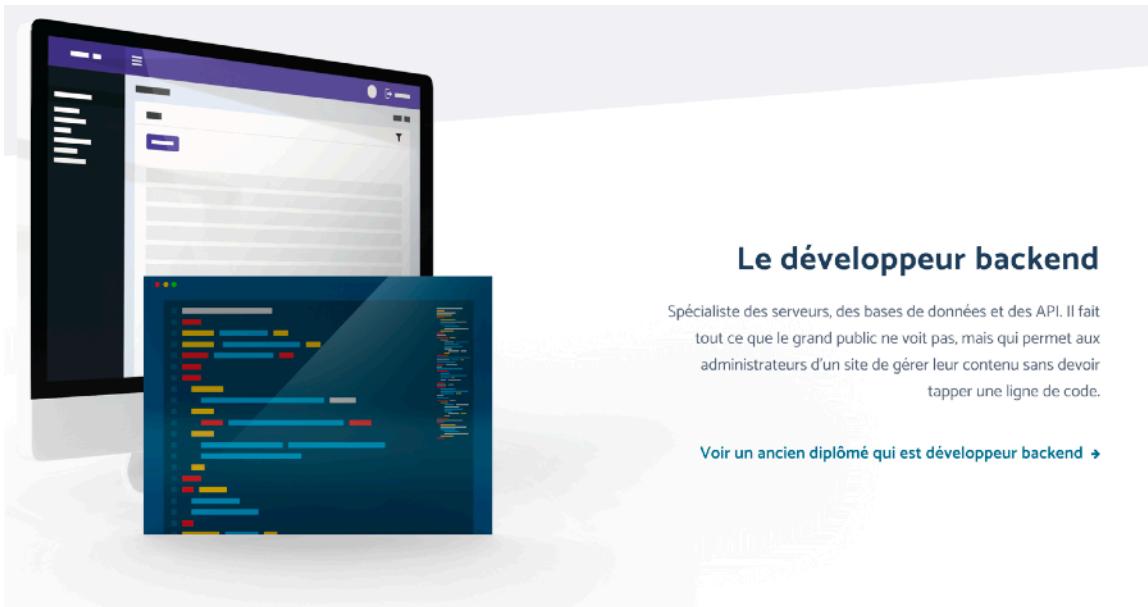


ANCIEN SLIDER DES ARTICLES SUR LA PAGE D'ACCUEIL

Même si je le trouvais plutôt bien fait, j'ai dû écouter mes utilisateurs et j'ai finalement pris une solution plus simple, j'y ai simplement apposé les cartes des articles que l'on retrouve sur mon blog.

L'une des autres pages compliquées a été celle où je présente les différents métiers du web. C'est une page très personnelle et unique. Elle doit représenter la section et peut varier dans le style. J'ai imaginé la présentation des différents métiers avec des illustrations afin de rendre le tout plus harmonieux. Pour cette section spéciale (et toutes les autres qui sont des parties importantes du site ou des points d'entrée vers d'autres pages), j'ai utilisé une version élargie du header, ce qui permet d'être plus visuel.

Voici quelques illustrations que j'ai réalisées sur la page des métiers du web :

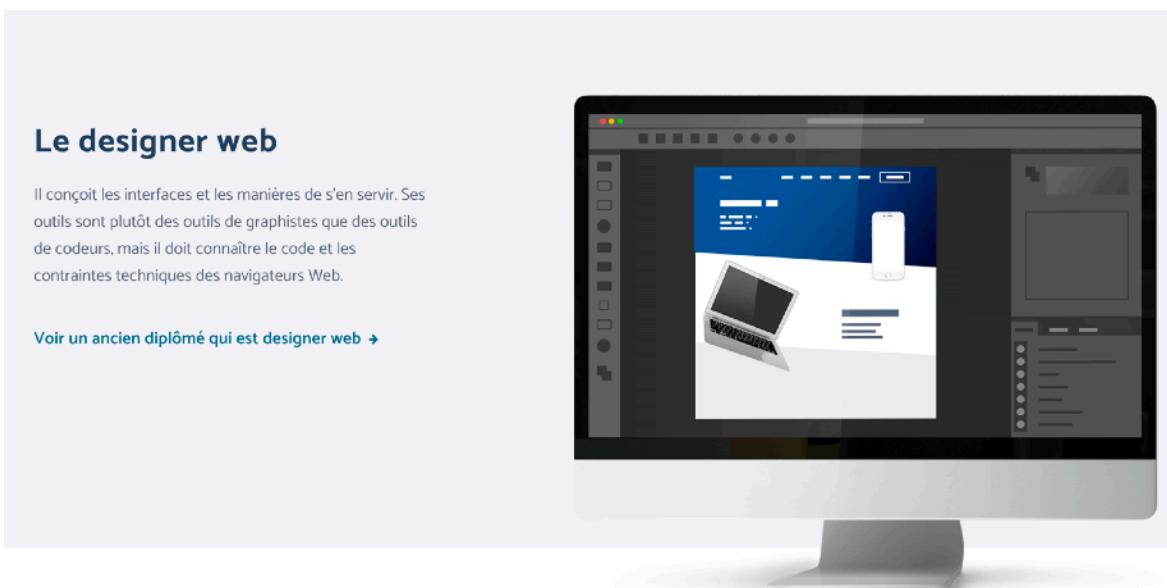


Le développeur backend

Spécialiste des serveurs, des bases de données et des API. Il fait tout ce que le grand public ne voit pas, mais qui permet aux administrateurs d'un site de gérer leur contenu sans devoir taper une ligne de code.

[Voir un ancien diplômé qui est développeur backend →](#)

ILLUSTRATION RÉALISÉE POUR LA PRÉSENTATION DU DÉVELOPPEUR BACKEND



Le designer web

Il conçoit les interfaces et les manières de s'en servir. Ses outils sont plutôt des outils de graphistes que des outils de codeurs, mais il doit connaître le code et les contraintes techniques des navigateurs Web.

[Voir un ancien diplômé qui est designer web →](#)

ILLUSTRATION RÉALISÉE POUR LA PRÉSENTATION DU DESIGNER WEB

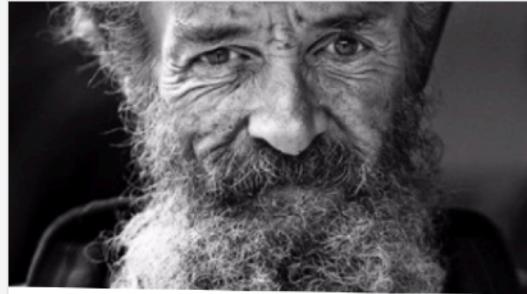
Le site comportant beaucoup d'informations et beaucoup de profils différents (professeurs, élèves, anciens étudiants, auteur, etc.), j'ai imaginé des cartes pour toutes les personnes qui seront représentées sur le site. Ces cartes sont réutilisées partout, pour inviter l'utilisateur à cliquer et à accéder au profil mis en avant. En voici un exemple :



Dylan
Schirino

Développeur front-end

[VOIR SON PARCOURS](#)



Toon
Van den Bos

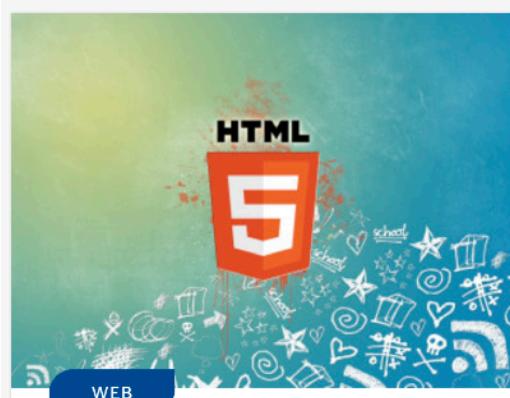
Creative developper

[VOIR SON PARCOURS](#)

CARTES DE PRÉSENTATION DES PERSONNES

Pour la page des réalisations, j'ai imaginé une sorte de grilles en « waterfall » à la façon Pinterest. Les images auraient toutes la même largeur, mais seraient de hauteur différente.

Pour le blog, j'ai imaginé des cartes d'articles, qui auraient une image et un label, le nom de l'auteur, le nombre de commentaires sur l'article ainsi que sa catégorie. En voici un exemple :



95% des intégrateurs écrivent mal leurs titres

 Par Dominique Vilain

4 ↗



Sint enim veniam sapiente dolorem distinctio eius...

 Par Bernadette Green

3 ↗

CARTES DU BLOG DE L'INFOGRAPHIE

Le design a été long à réaliser et il comporte certainement encore beaucoup d'imperfections. Il y a beaucoup de choses que je ferais différemment aujourd'hui, avec l'expérience que j'ai acquise en stage et lors de mes différentes explorations depuis que j'ai commencé. Néanmoins, j'ai essayé de garder une certaine cohérence entre tous les éléments et jusqu'à maintenant, on ne m'a pas encore fait remarquer qu'il y avait des problèmes à ce niveau-là.

5.4. OUTILS UTILISÉS

Pour réaliser ce design, j'ai utilisé différents outils. Donc principalement **Sketch**, une application pour le système d'exploitation Mac. Elle permet de se substituer à Adobe. Elle fournit une interface claire et basée pour le design d'application web et mobile, elle est très rapide et fonctionne en vectoriel, à l'inverse de Photoshop qui est beaucoup trop souvent utilisé pour réaliser des maquettes graphiques par les designers.

En addition à **Sketch**, j'ai utilisé une application nommée **PaintCode** qui permet de réaliser des SVG au code très propre. **Sketch** permet de créer des SVG, mais le code qu'il sort est vraiment mauvais. On a la possibilité de le minifier avec des outils comme le célèbre compresseur [svgomg](#). Seulement, ça compresse et ça ne rend pas le code plus lisible. Pour les SVG que l'on n'a pas besoin de manipuler, c'est parfait, mais pour ceux pour lesquels on voudrait réaliser une animation un peu plus poussée, **PaintCode** permet de rendre la tâche plus facile. Évidemment, PaintCode offre d'autres avantages, comme l'export des SVG pour les intégrer à un canvas JavaScript, ou dessiner des formes pour Android ou iOS et pouvoir en exporter le code. Il permet aussi de créer des animations.

J'ai également utilisé l'application Affinity Designer, que j'apprécie tout particulièrement. C'est un Illustrator en moins compliqué et plus ergonomique. Elle est rapide et très bien réalisée.

PARTIE 3 : DÉVELOPPEMENT

1. DÉVELOPPEMENT HTML ET ACCESSIBILITÉ

J'ai commencé par développer mon site web avec le premier langage du web, le **HTML**. J'ai directement commencé à développer dans le framework Laravel, pour avoir la souplesse du moteur **blade**. Il permet, à l'instar d'autre moteur comme **PUG**, d'utiliser des variables et des partials. Je ne vais pas parler de ce qu'est **le HTML**, mais je vais parler de l'importance qu'ont eue les microdatas et les rôles arias sur l'accessibilité dans ma structure.

1.1. UN MENU CLAIR, SÉMANTIQUE ET ACCESSIBLE

Comme ma structure et mon design le supposent, j'ai été amené à penser la structure du menu de navigation. Le site de mon projet comportant beaucoup de pages, il a été important de faire un tri des éléments à afficher dans le menu. J'ai commencé par lister toutes les pages puis je les ai classées par ordre d'importance.

Pour définir l'importance d'une page (et donc, d'un lien), je les ai comparés l'une à l'autre en me demandant à chaque fois « Est-ce que cette page contient des contenus indispensables pour la cible de mon site ? » Le produit étant le site de l'école, il fallait que je mette en valeur les contenus s'y rapportant le mieux pour mettre en évidence l'établissement et ses formations, ainsi que ceux se rapportant aux objectifs de mes utilisateurs. Tout de suite, plusieurs éléments logiques sont sortis du lot. Mais il y en avait beaucoup trop pour rester dans une barre de navigation standard.

Dans ma structure, j'ai pensé diviser les formations en différentes pages de métiers qui comportent chacune d'elles différente sous pages. J'ai décidé de les regrouper en un dropdown avec l'intitulé « Les métiers ». Ces pages font sans doute partie des plus importantes du site.

Le deuxième élément de menu, qui est aussi un menu déroulant, parle de l'école et contient ses sous-pages, comme la page de contact.

Le troisième élément présente la galerie des travaux que les étudiants ont réalisés lors de leur cursus. C'est aussi, selon moi, une des parties les plus importantes du site, qui est le mieux à même de mettre en valeur la formation reçue.

Le quatrième élément liste les diplômés de l'école. J'ai décidé de le mettre dans le menu pour montrer que chaque année des étudiants sortent avec leur diplôme et mettre en valeur cet élément montre que l'école est active.

Le cinquième élément est le blog tenu par les professeurs et/ou externe. Le site étant aussi une possibilité de maintenir sa veille à jour, il est important d'avoir un lien direct et visible sur chaque page.

Le dernier élément est un bouton qui a plus d'importance, dans le but d'inciter au clic, c'est celui qui propose à l'étudiant de s'inscrire.

Pour rappel, voici la structure définitive du menu :

- **Les métiers (qui n'est pas un lien)**
 - Les métiers du web
 - La formation
 - Le programme
 - Les parcours des anciens
 - **Les métiers du design graphique**
 - La formation
 - Le programme
 - Le parcours des anciens
 - **Les métiers de la 3D et de l'audiovisuel**
 - La formation
 - Le programme
 - Le parcours des anciens
 - **L'école (qui n'est pas un lien)**
 - L'actualité
 - Les commodités
 - Le matériel et les bâtiments

- Contactez-nous
- **Réalisations**
- **Nos diplômés**
- **Blog**
- **Je veux m'inscrire**

Certains liens de moins grande importance pour la cible principale sont placés dans le footer, comme à l'accoutumée en web, comme proposer une offre de stage ou avoir la liste de tous les professeurs.

Comme on peut le voir dans la structure proposée plus haut, je vais devoir créer un menu déroulant, ce qui pose des problèmes d'accessibilité. En effet, la plupart des menus dropdowns ne respectent pas les principes élémentaires de l'utilisation d'un site web par les « handicapés du web », comme la navigation au clavier par exemple.

Les avantages du menu déroulant sont entre autres :

- Structurer les nombreux choix possibles en créant des zones, et sous-zones.
- Illustrer ces choix avec des visuels pour renforcer leur compréhension.
- Rendre accessible un grand nombre de pages tout en limitant la quantité d'éléments de premier niveau qui se trouveront dans le menu.
- Quand il est bien fait, le dropdown permet d'améliorer l'expérience utilisateur, car il permet à l'usager de trouver facilement ce qu'il souhaite.

L'une des premières questions que je me suis posées en pensant mon menu déroulant est : **est-ce que l'élément de premier niveau qui contient les items du menu déroulant doit être un lien qui pointe vers une page ?** Dans la pratique, c'est un lien qui pointerait vers une page contenant une redirection vers des pages de son domaine. Cette page, sauf si on lui trouve un intérêt autre que de **répéter** le contenu qui se trouvera sur d'autres pages, n'a pas d'intérêt. Ça a été longtemps (et ça l'est toujours) une solution de contournement pour rendre accessible un menu déroulant. Si l'utilisateur n'arrive pas à accéder au menu déroulant, sans lien sur l'item principal, il ne peut pas terminer son action. Cela

peut arriver par exemple, si l'utilisateur n'a pas JavaScript. C'est une manière de rendre accessible le contenu dans n'importe quelle situation, mais on crée dans ce cas une page au contenu peu utile sauf si elle est bien exploitée.

Dans mon cas, j'ai décidé de ne pas créer de page « alternative », mais de proposer une solution fonctionnelle sans JavaScript.

Ensuite, je me suis demandé, lors de la construction **HTML** du menu, si son accessibilité serait bonne, **doit-il être utilisable au clavier ?** La réponse à cette question est un grand oui. Selon anysurfer.be, le site doit être entièrement navigable au clavier et je les rejoins. Il existe un organisme, le DSGWG (DHTML Style Guide Working Group) qui régit la façon dont des éléments d'interface web doivent se comporter avec les habitudes des utilisateurs.

Dans leurs règles, ils disent, par exemple :

- Le menu non déroulé doit pouvoir être utilisé avec la touche de tabulation et les flèches de gauche à droite.
- Sur un item de premier niveau, les liens de sous niveaux doivent être accessible lorsque l'on appuie sur la touche **enter** ou **flèches du bas**.
- On doit pouvoir quitter le menu déroulant grâce à la touche **Esc**
- Parcourir le menu dans toutes les directions à l'aide des touches fléchées.
- ...

En continuant mes recherches, notamment sur les **ROLES-ARIA** je suis tombé sur les travaux qu'a réalisés l'équipe d'Adobe.

Grâce aux rôles ARIA, nous pouvons communiquer la relation entre l'élément de menu de niveau supérieur et son sous-menu, ainsi que l'état actuel du sous-menu. À cette fin, le menu d'Adobe inclut le balisage **ARIA** suivant, qui est ajouté à l'aide de JavaScript (car sans JavaScript, notre menu déroulant ne serait qu'une liste imbriquée non interactive, aucun de ces états et propriétés ne serait pertinent) :

Les ARIAS ajoutés sur les liens de premier niveau :

- **ID**
- **aria-haspopup="true"** (permet la simulation du pointage d'une souris sur un écran tactile)
- **aria-owns=" id_of_subnav"** (identifie un élément (ou des éléments) afin de définir une relation parent/enfant visuelle, fonctionnelle ou contextuelle entre des éléments DOM où la hiérarchie DOM ne peut pas être utilisée pour représenter la relation)
- **aria-controls=" id_of_subnav"** (identifie l'élément (ou les éléments) dont le contenu ou la présence sont contrôlés par l'élément courant)
- **aria-expanded=" false"** (indique si l'élément ou un autre élément de regroupement qu'il contrôle est actuellement déroulé ou non. Sa valeur change en fonction de l'état)

Les ARIAS ajoutés sur les liens de seconds niveaux :

- **ID**
- **rôle=" group"** (identifie un groupe d'éléments)
- **aria-expanded=" false"**
- **aria-labelledby=" id_of_top_level_link"** (permet de lié le sous menu à son élément qui le déroule)

Tous ces attributs devraient être ajoutés lors de la réalisation d'un menu déroulant qui utilise JavaScript. Adobe a sorti en open source un plug-in qu'ils ont réalisé pour la création d'un menu dropdown accessible qui réponde à tous les points énoncés plus haut. J'ai donc décidé de l'utiliser dans mon projet, bien que j'ai dû modifier le code source original pour pouvoir ajouter un titre à mon menu (le plug-in prend le premier élément après la balise **nav**, mais comme la balise est un élément sectionnant, il faut y ajouter un titre pour diverses raisons de compatibilité).

Pour améliorer l'accessibilité de mon menu, j'ai en outre décidé d'ajouter **un lien d'évitement** (un lien avant le menu) qui permet à un utilisateur qui utilise une

interface de lecteur d'écran de pouvoir passer directement au contenu lorsqu'il est sur une page afin d'éviter la relecture systématique du contenu du menu. I

1.2. BALISER CORRECTEMENT UN FIL D'ARIANE

J'ai décidé d'utiliser un fil d'ariane pour mon site. Il s'agit, en ergonomie web, d'une aide à la navigation sous forme de signalisation de la localisation du lecteur dans une page web. Cela permet de donner à l'utilisateur un moyen de garder une trace de son emplacement à l'intérieur de l'environnement.

Lors de la construction de ce fil d'ariane, je me suis posé la question, « Mais, comment faut-il le baliser ? ». C'est un détail, nous sommes d'accord, mais existe-t-il une manière sémantique de le réaliser ?

On pourrait par exemple le créer dans un **** qui contiendrait des **** eux-mêmes contenant des liens. C'est la solution rapide. On a une liste de lien. Mais sémantiquement, elle ne semble pas correcte. Il s'agit d'une liste non ordonnée, ce qui donne l'impression que tous les éléments sont sur le même niveau, ce qui n'est pas le cas. Il faudrait que l'on puisse distinguer les différents niveaux des liens. On pourrait imbriquer des **** dans des ****, mais ce serait lourd à utiliser. Et pas très lisible.

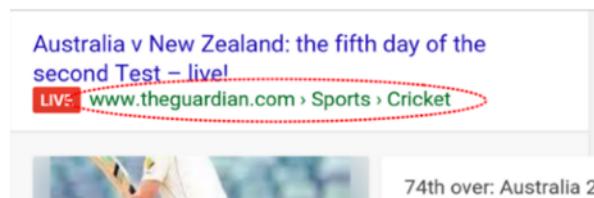
Un attribut HTML avait été créé pour pallier à ce problème des liens imbriqués pour un fil d'ariane mais il a été abandonné, car la technique mise en œuvre n'était pas la meilleure. Grâce à l'attribut **rel** on pouvait lui passer en argument le **up**. On avait par exemple :

```
<p>
  <a href="/" rel="index up up up">Main</a>
  <a href="/products/" rel="up up">Products</a>
  <a href="/products/dishwashers/" rel="up">Dishwashers</a>
  <a>Second hand</a>
</p>
```

C'est très peu lisible, imbriquer plusieurs **up** n'était pas la solution. Ça a été abandonné, heureusement.

Il y a une autre solution, un peu plus sémantique. C'est d'utiliser des listes ordonnées, un **** suivi de **** ce qui offre une différentiation de niveau entre un élément et un autre.

Le tout est aussi de savoir si le fil d'ariane est une navigation, ou non. Dans ce cas, il serait pertinent de la mettre dans la balise **<nav>**. Je pense pouvoir dire que oui, le fil d'ariane, même si c'est un élément de repérage, est une zone qui permet de naviguer entre plusieurs pages (au minimum 2). Comme le dit le site w3schools, une **<nav>** définit un ensemble de liens de navigation. Tous les liens d'un document ne doivent pas faire partie d'un élément **<nav>**. Ce dernier est fait pour créer des blocs conséquents contenant des liens de navigations au sein du document. On peut penser à la navigation principale, ou une sous-section. J'hésite toujours à l'inclure dans un **<nav>** j'ai donc continué mes recherches qui m'ont amené vers le non moins important site des développeurs Google. Google utilise les fils d'ariane pour l'afficher sur sa page de recherche. J'ai donc analysé comment ils faisaient pour marquer leurs données.



La solution retenue par Google est d'utiliser les microdatas de <http://schema.org/BreadcrumbList>. GoogleSearch demande trois propriétés obligatoires :

- **Item** : Un élément individuel dans le fil d'ariane. Il contient une URL unique et la propriété **name**
- **name**: le titre du fil d'ariane affiché pour l'utilisateur.
- **Position** : La position dans le fil d'ariane. La position 1 signifie qu'il est l'élément premier (par exemple, Accueil).

À noter que Google n'utilise pas le tag **<nav>**, mais simplement une liste non ordonnée ****. J'ai analysé les exemples qu'ils donnent sur cette page (<https://>

developers.google.com/search/docs/data-types/breadcrumbs#guidelines) et je l'ai reproduit pour mon fil d'ariane. En voici une copie :

- Le tag **** est balisé avec le type qui le définit comme étant un fil d'ariane **breadcrumbList**.
- Chaque item **** est défini comme **itemprop** en étant un élément de la liste d'élément par **itemListElement** et ont un nouveau type en étant balisé comme élément de liste par **listItem**.
- Chaque lien à obtient l'itemtype **thing** pour pouvoir donner un nom à son titre et il a aussi une propriété **item** propre au fil d'ariane.

La ou ça devient intéressant, c'est que l'on utilise une balise **<meta>** à la fin de chaque ****, avec comme attribut **itemprop position** et comme **content** sa place dans la hiérarchie.

```
<ol itemscope itemtype="http://schema.org/BreadcrumbList">
    <li itemprop="itemListElement" itemscope itemtype="http://
schema.org/ListItem">
        <a href="#" itemscope itemtype="http://schema.org/Thing"
itemprop="item">
            <span itemprop="name">Page d'accueil</span>
        </a>
        <meta itemprop="position" content="1" />
    </li>
    <li itemprop="itemListElement" itemscope itemtype="http://
schema.org/ListItem">
        <a href="#" itemscope itemtype="http://schema.org/Thing"
itemprop="item">
            <span itemprop="name">Les métiers du web</span>
        </a>
        <meta itemprop="position" content="2" />
    </li>
</ol>
```

C'est la technique préconisée par Google. Comme quoi, un fil d'ariane n'est pas si simple à construire sémantiquement parlant. Il est très important, car grâce à ça, Google est capable de montrer à l'utilisateur, en fonction de ses termes de

recherche, à quel endroit dans le site web il va apparaître précisément, avant le clic.

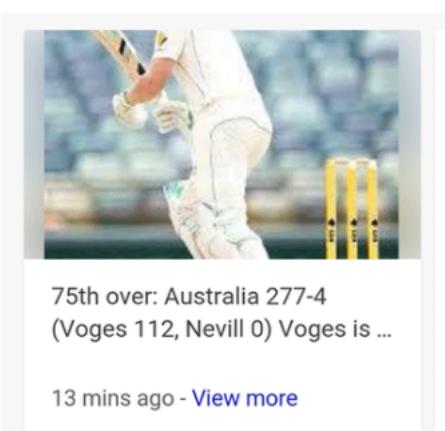
1.3. DES MICRODATAS POUR TOUT

Pour mon projet, j'ai tenté d'utiliser un maximum de fois les microdatas là où elles étaient pertinentes. J'en ai par exemple ajouté sur la page de contact et dans le footer (où on y retrouve aussi des informations de contact). Mais surtout, j'ai voulu les utiliser pour des composants spécifiques. L'un de ces composants est les **articles du blog**.

Les microdatas sont vraiment puissantes et utilisées le plus possible par Google. C'est ce que j'ai appris via mes recherches. Le HTML permet de construire un code sémantiquement correct, mais les microdatas donnent de la précision et de la force au contenu. Pour baliser mon article, j'ai utilisé les attributs suivant :

- Sur la balise article, j'ai utilisé l'itemtype **BlogPosting**.
- J'ai mentionné à Google qu'il s'agissait d'une page web entière grâce à l'itemprop **mainEntityOfPage**.
- Sur le titre, l'auteur et la description j'y ai ajouté respectivement les itemprop **headline**, **author** et **description**.
- J'ai également ajouté deux balises meta, qui sont obligatoire (ou du moins l'information demandée) qui indiquent la date de publication et la date de modification grâce aux itemprop **datePublished** et **dateModified**.

Grâce à cette balisation spécifique, Google sera en mesure d'afficher les articles pertinents de la façon suivante dans les résultats de recherche :



Dans mes recherches sur [schema.org](#), je me suis rendu compte qu'il était possible de baliser des cours pour que Google les référence pour les étudiants potentiels. Pour ce faire, j'ai utilisé :

- Sur le conteneur, j'ai utilisé l'itemtype **Course**.
- J'ai mentionné le titre du cours grâce à l'itemprop **Name**.
- J'ai également mentionné une description du cours grâce à l'itemprop **description**.
- J'ai aussi indiqué l'organisation qui dispensait ces cours sur chaque page afin que Google puisse les associer.

Avec ces informations, Google devrait être capable de rendre les informations suivantes :

The screenshot shows a search results page for 'Learning English Courses | Coursera'. It displays three course cards:

- English for Journalism**: Welcome to English for Journalism, a course created by the ... University of Pennsylvania
- Speak English Professionally: In Person, Online & O...**: Do you want to speak better English? This course will help yo... Georgia Institute of Technology
- Grammar and Punctuation**: Course 1: Grammar and Punctuation Do you need to review ... University of California, Irvine

Pour résumer, j'ai vraiment tenté d'aller plus loin avec l'**HTML**, et de chercher toutes les façons possibles pour l'améliorer. Car c'est la base de tout site web et le négliger désert tout le travail réaliser. Si Google n'a pas d'informations pertinentes à afficher, il n'en affichera pas.

2. DÉVELOPPEMENT CSS

Le **CSS**, pour Cascading Styles Sheet, est le langage qui met en couleur le web. Grâce à différentes propriétés, on peut intégrer à peu près tous les designs dans un navigateur. Je ne vais pas parler du **CSS** en lui-même, car c'est une technologie largement utilisée, mais je vais parler de la façon dont j'ai arrangé mon code et dont je l'ai maintenu et des erreurs que j'ai commises au début.

2.1.TECHNOLOGIES UTILISÉES

Cela peut paraître étrange de parler des technologies utilisées alors que je suis en train de parler de la technologie **CSS**. J'ai décidé d'utiliser un préprocesseur, un langage de programmation qui permet de gérer plus efficacement ses feuilles de styles **CSS**. Je n'ai pas utilisé SASS, celui que j'ai découvert lors de ma formation. J'ai préféré aller vers **Stylus**, qui est plus minimaliste et, à mon sens, plus clair.

Stylus permet de se substituer de beaucoup de choses que je trouvais assez mal faites dans **Sass** et le **CSS** standard. Par exemple, grâce à **Stylus**, je peux éviter de mettre des accolades à la fin de chaque sélecteur. Je peux aussi éviter de mettre les deux points après une propriété ainsi que le point virgule.

Stylus permet aussi d'organiser son code visuellement, il faut être rigoureux, car l'indentation déterminée si une propriété est imbriquée dans un sélecteur. Grâce à ça, le code se lit très facilement.

Pour ce projet, j'ai également utilisé une librairie appelée **WhiteSass**, créé par Toon Van den Bos, un professeur invité et ancien étudiant de la H.E.P.L. seulement, **WhiteSass** a été codé à l'origine... pour **Sass**. Voulant absolument utiliser certaines fonctions, j'ai décidé de convertir cette librairie pour **Stylus**. Et après des milliers de lignes de code, je n'y ai pas trouvé de bug. Le système de grille fonctionne, les différentes fonctions de conversions aussi. Il y a fort à parier que je l'utiliserais encore à l'avenir.

2.2.ARCHITECTURE

Pour organiser mes feuilles de styles, j'ai utilisé l'architecture que l'on a apprise lors des cours, qui s'appelle **ITCSS**. Elle divise l'organisation des feuilles de styles en différentes parties.

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

- Trumps

J'ai un peu interprété cette structure pour en faire quelque chose qui me convenait plus. J'ai gardé le fichier base pour y implémenter tous mes styles génériques. On peut y retrouver par exemple les styles généraux pour la taille de police standard, le font-smoothing et une organisation des fichiers permettant de définir les styles pour mes titres, pour mes boutons, mes liens, etc.

Ensuite, dans le dossier generic, j'y ai apposé tous les reset que j'utilise (le reset de Meyer et celui pour le flexbox).

Dans les settings, on y retrouve les fichiers qui gèrent les différents chargements de police, mes variables de couleurs, la liste de mes icônes, des fonctions de easings, etc.

La couche components est la couche que j'ai utilisée majoritairement. C'est là que j'ai stylé tous les composants et les pages de mon projet. Quand il y avait plusieurs fichiers pour une même page (par exemple, pour le blog, où je pouvais avoir un fichier pour l'aside, pour l'index, le post, les commentaires et les cartes), j'ai décidé de les regrouper dans un sous dossier.

À l'inverse de ce que recommande **ITCSS**, je n'ai pas vraiment utilisé la couche object. J'ai préféré mettre tous mes styles dans components, je trouve cette approche plus rapide. Seulement, j'ai essayé de prendre pour habitude de mettre tous les éléments de positionnement en premier dans mes feuilles de styles et dans mes sélecteurs.

2.3. PRINT CSS

Comme j'ai prévu un blog, et que ce blog pourrait contenir des cours ou des articles de veille, il serait agréable de pouvoir imprimer ces pages pour ceux qui en auraient le besoin. J'ai décidé de prendre le temps de faire une petite feuille de style basique pour les articles du blog afin qu'il soit possible de les imprimer sans trop d'incohérence visuelle. Mon seul regret est que je n'ai pas eu le temps de le faire pour toutes les pages du site.

3. DÉVELOPPEMENT JAVASCRIPT

Mon site se voulant dynamique, je l'ai agrémenté de code JavaScript. On y retrouve différents composants comme des sliders, des formulaires de recherche, du lazy-loading, un système de grille waterfall, des Google maps, etc.

3.1. TECHNOLOGIES UTILISÉES

Au départ, je voulais réaliser ce site en **JavaScript vanilla**. Seulement, après avoir discuté avec différents professionnels lors de mon stage, **jQuery** m'a été présenté comme une véritable ressource pour gagner du temps. Et comme au final, j'en ai manqué, il m'a été d'une grande utilité. **jQuery** à l'avantage que l'on peut écrire du code plus rapidement et qu'il existe différentes fonctions et plug-ins qu'il est possible d'insérer facilement dans la plupart des projets. De plus, la communauté est très active et la plupart des réponses sont données en jQuery.

Attention, j'ai utilisé **jQuery** parce que le projet sur lequel je devais travailler nécessitait beaucoup de code, il va de soi que si ça n'avait pas été le cas, je n'aurais pas opté pour cette option.

Avant de me lancer dans **jQuery**, j'ai quand même analysé le framework pour voir s'il n'était pas trop lourd, comme certain s'amusent à le dire. J'ai préféré vérifier par moi-même. La version non minifiée de **jQuery** pèse... 238 ko. C'est lourd et ça a de quoi refroidir. Une fois minifié, on tombe à moins de 87 ko. C'est toujours lourd, mais plus acceptable. Ensuite, gzipé, on arrive à une librairie qui pèse moins de 32 ko. Autant dire que ce n'est pas grand-chose pour un site de cette envergure. Et au vu du temps que je gagnerais à l'utiliser, je prends les 32 ko avec sourire. C'est moins lourd qu'une image. Et des images dans mon projet, il va y en avoir beaucoup.

Au moment où j'écris ces lignes (le projet est presque terminé), le JavaScript de mon site, minifier, fait 53 ko. Si on y ajoute **jQuery**, on arrive à 85 ko. Encore une fois, c'est moins qu'une belle image. Au niveau performance, je trouve cela très correct. Je ne regrette pas mon choix d'avoir utilisé **jQuery**, surtout qu'il permet de manipuler **ajax** très simplement.

Le seul regret que j'aie par rapport au JavaScript que j'ai utilisé est que je n'ai pas utilisé les nouvelles versions d'ES 2016. J'y ai pensé après et il était trop tard pour l'implémenter. Par souci de temps, j'ai préféré ne pas revenir sur ce point.

3.2. STRUCTURE DES FICHIERS

Au départ, je n'avais pas vraiment de structure. Je codais tout dans un fichier, mais j'ai très (très) rapidement vu que pour la maintenabilité, ce ne serait pas rentable. J'ai alors commencé à diviser mes scripts en plusieurs fichiers que je nommais 001_menuAccessible, 002_menuResponsive, 003_sliderHome, etc. Mais j'y ai aussi vu toutes les limites que cela impliquait (notamment si je devais insérer un script entre, je devais modifier tous les numéros). Je n'avais pas trop réfléchi puis j'ai fait des recherches et j'ai finalement utilisé une technique que j'ai apprécié mettre en œuvre.

J'ai incorporé mon fichier de ressource nommé **js** trois dossiers, nommés **vendor**, **pages** et **class**. Dans le fichier **vendor**, j'y place tous mes plug-ins et dépendances. Par exemple, j'ai utilisé le plug-in **chosen** pour styler un select. C'est là que je l'y ai mis. Et tous les autres plug-ins que j'ai utilisés, je les ai placés dans ce dossier. Lors de la compilation des fichiers, je dis dans ma configuration de charger d'abord tous les scripts de vendor.

Dans le dossier **pages**, j'y ai placé un fichier par page utilisant du JavaScript dans mon site. Je définis chaque page par une classe, ce qui permettra de l'appeler facilement dans la suite. Dans ces fichiers, j'y place une fonction init qui me sert à lancer les différentes fonctions que j'ai codées. Dans l'exemple ci-dessous, on voit que je charge un fichier script externe appelé **sliderHome** et un petit appel à une fonction d'un de mes fichiers placer dans **vendor** qui me permet d'afficher aléatoirement les articles sur la page d'accueil.

```

w.home = {
    oConf: {}, 

    init: function () {
        w.sliderProsHome.init();
        this.randomizeBlog();
    },
    randomizeBlog: function () {
        $("#blog-home").randomize("article.blog-card");
    }
}

```

CLASSE HOME, QUI PERMET D'INITIALISER DES SCRIPTS SPÉCIFIQUEMENT SUR LA PAGE D'ACCUEIL.

Ensuite, dans le dossier **class**, j'y place tous les bouts de code JavaScript que j'ai développé, soit ce sont des fichiers de configuration pour un plug-in, soit ce sont des bouts de code. Par exemple, dans mon site, j'ai choisi d'utiliser pour tous les formulaires des labels flottants (au clic sur un input, le label qui est placé comme un placeholder reprend sa place de label). Voici à quoi ressemble le code d'une classe :

```

w.floatLabel = {
    init: function () {
        this.floatLabel();
    },
    floatLabel: function () {
        (function($){
            function floatLabel(inputType){
                $(inputType).each(function(){
                    var $this = $(this);
                    $this.focus(function(){
                        $this.prev().addClass("active");
                    });
                    if ($this.attr('value') !== ''){
                        $this.prev().addClass("active");
                    }
                    $this.blur(function(){
                        if($this.val() === '' || $this.val()
                        === 'blank') {
                            $this.prev().removeClass('active');
                        }
                    });
                });
                floatLabel(".floatLabel");
            })(jQuery)
        },
    }
}

```

Ce code sera utilisé sur toutes les pages qui contiennent des formulaires. Et uniquement sur ces pages-là. Car rappelez-vous, nous avons un fichier **JavaScript** pour chaque page nécessitant du code, il suffit d'appeler cette fonction dans la fonction **init** de la page en question.

Ensuite, j'ai créé un fichier à la racine du dossier **js**, que j'ai nommé **front.js**. Sur ce fichier, je peux initialiser toutes les classes que je veux et les rendre disponibles sur toutes les pages. Et c'est dans ce fichier aussi que j'initialise les scripts pour chaque page (comme ça, un script n'est utilisé que sur la page où on l'a demandé). Pour ce faire, je compare la classe qui est sur le body, et si c'est la même, retourne le bon script. Voici à quoi ressemble ce fichier :

```
var w = {
    // setup
    init: function () {
        w.getElements();
        w.sCurrent = w.getCurrentPage();
        w.loadPage();
        w.isLoaded = true;
    },
    loadPage: function () {
        if (w.sCurrent) w[w.sCurrent].init();
    },
    // functions
    getElements: function () {
        w.$body = $('body');
    },
    getCurrentPage: function () {
        if (w.$body.hasClass('home')) return 'home';
        if (w.$body.hasClass('blog')) return 'blog';
        if (w.$body.hasClass('postBlog')) return 'postBlog';
        if (w.$body.hasClass('graduate')) return 'graduated';
        if (w.$body.hasClass('realisations')) return 'realisations';
        if (w.$body.hasClass('internship')) return 'internship';
        if (w.$body.hasClass('registration')) return 'registration';
        if (w.$body.hasClass('contact-us')) return 'contact';
        if (w.$body.hasClass('postNews')) return 'postNews';

        return false;
    }
};

$(window).on('load', w.init);
```

Au final, tout ça me semble bien plus structuré et surtout, les scripts se lancent au bon endroit, au bon moment et pas pour rien. Ce n'est pas la méthode idéale, je pense, car il y a beaucoup de fichiers différents, mais elle est efficace.

3.3. CHOISIR DE BONS PLUG-INS

Personnellement, je ne suis pas pour coder des choses qui ont déjà été faites une dizaine de fois juste pour le plaisir. C'est pourquoi j'ai utilisé une série de plug-ins jQuery (et non-jquery) pour m'aider à accomplir des tâches.

Je n'ai pas fait le choix de ces plug-ins par hasard, je n'ai pris les premiers venus. J'ai fait plusieurs recherches, je les comparais et si au final, ça me convenait, que le code était propre et que c'était flexible, je l'utilisais. Dans l'autre cas, je codais le fichier moi-même. L'avantage d'un plug-in connu est qu'il a été testé par toute une communauté et qu'il a été enrichi sur différent aspect. Néanmoins, il faut toujours se méfier, car il existe beaucoup d'usines à gaz.

Voici quelques plug-ins que j'ai décidé d'utiliser pour ce projet :

- **JQuery Accessible Mega Menu** : comme expliquer plus haut dans ce rapport, j'ai décidé d'utiliser le script réalisé par Adobe pour rendre accessible mon menu. Je l'ai trouvé un peu lourd pour un code de menu dropdown (**12 ko** minifiés). Mais après réflexion, il s'occupe de tellement d'aspects différents (navigation au clavier, rôle aria, accessibilité...) qu'il en vaut vraiment la peine. Et surtout, le code est très clair, ce qui permet au besoin de se l'approprier et d'en faire les modifications que l'on juge utiles.
- **stupid jquery table** : dans le site, on peut être amené à trier un tableau par ordre croissant ou décroissant. J'étais parti pour réaliser le code moi-même et quand je cherchais des pistes pour savoir par où commencer, je suis tombé sur ce petit script qui faisait exactement ce que je n'attendais, ni plus ni moins. Minifier, il fait **2 ko**.
- **Modernizr** : très connu, il permet de mettre en place des classes fallback lorsqu'un élément n'est pas supporté. Je l'ai notamment utilisé pour détecter si JavaScript est activé (même si pour faire cela, deux lignes de code suffisent),

mais surtout, pour avoir une comptabilité lorsque j'utilisais flexbox, je pouvais déterminer un fallback s'il n'était pas supporté par le navigateur.

- **Waterfall** : J'ai décidé de présenter certains éléments en grille à la façon Pinterest et après moult tentatives de réaliser l'opération en **CSS** (et que ce soit compatible avec ie9), j'ai décidé de me tourner vers **JavaScript**. Je savais qu'il existait des plug-ins pour gérer les grilles comme Masonry, mais généralement ce sont des usines à gaz et ils font beaucoup plus que ce que je souhaite. J'étais parti pour coder la grille moi-même, mais encore une fois, je suis tombé sur un script très léger, **1 ko** minifier qui ne se contente que de positionner les images. Le reste, c'est moi qui gère.
- **Chosen** : il sert à styler des select. Je n'aime pas trop ce plug-in, car on est dépendant de plein de choses, comme sa feuille de styles CSS longue et incompréhensible. Mais en se creusant la tête, on arrive à l'utiliser plus ou moins comme on le veut. Il existe des alternatives, mais pas meilleure (ni plus légère). Il fait **29 ko** minifier, lourd pour ce que c'est. Mais c'est le genre de plug-in qui offre plein de possibilités, et on est obligé de tout prendre.
- **JQuery viewport checker** : un plug-in qui sert à détecter la position dans la page lorsque l'on scroll. Je l'utilise sur beaucoup de projets, il est léger et simple d'utilisation. Grâce à lui, ça me permet d'ajouter des effets d'images au survol. Il fait **2 ko** minifier.
- ...

4. DÉVELOPPEMENT CÔTÉ SERVEUR

Nous voici à la partie la plus importante. Non pas que les autres le soient moins, car pour faire un site web, on a besoin de toutes les technologies citées précédemment. Mais je veux dire par là que c'est la partie qui m'a demandé le plus de travail. Les autres technologies, je les connaissais déjà, je les avais apprises au cours et j'avais pu les expérimenter dans différents projets. Je vais dès lors être plus précis que pour les autres parties, avec plus d'exemples de code et plus de détails.

4.1. AVANT DE COMMENCER

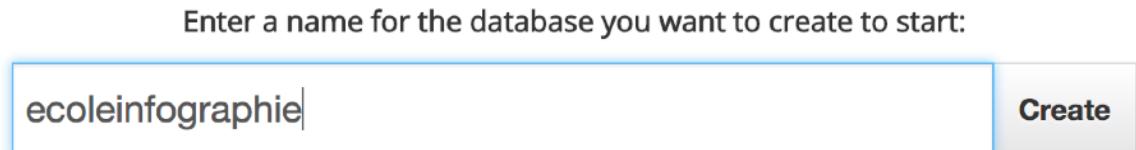
Avant de débuter à coder la partie serveur, j'ai pris quelques jours pour revoir les bases du **PHP** et faire quelques tutoriels sur le framework **Laravel**. J'ai notamment utilisé des sites comme openclassroom, laravel.sillo.org et j'ai regardé quelques

vidéos sur laracast.com. Ces quelques jours ont été essentiels, car ça m'a permis de me lancer dans l'aventure plus posément, en ayant quelques notions.

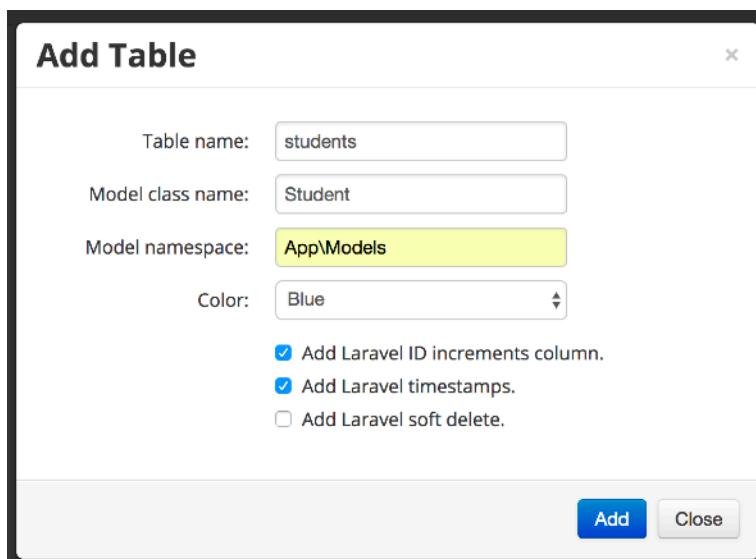
4.2. SCHÉMATISATION DE LA BASE DE DONNÉES

Toute application qui gère des données le fait en relation avec une base de données. Avec Laravel, on peut utiliser les bases de données mysql. Avant de commencer le développement, je voulais avoir un schéma clair et précis de ce que je devais créer comme tables. Pour ce faire, j'ai utilisé un outil en ligne appelé **Laravel Schema Designer**. On le trouve à l'adresse laravelsd.com.

Pour créer un schéma de base de données, il suffit d'y inscrire le nom que l'on veut lui donner dans l'encart suivant :



Une fois la base de données créée, on arrive sur un écran vide comportant un menu. La première chose à faire est de créer une table en cliquant sur « Add table ». Ensuite, une popup apparait :



Comme on le voit, on nous demande plusieurs informations. On doit lui donner le nom de table, le nom du fichier modèle et le namespace. On peut lui assigner une couleur et demander que la table soit créée avec les champs automatiques tel

qu'un ID auto-incrémenté et les timestamps (created_at et updated_at). Une fois la table créée, on obtient ceci :

The screenshot shows a table named 'students' with two columns: 'id increments' and 'timestamps'. Each column has an edit icon and a delete icon. Below the table, there is a button labeled 'Add new column' with a plus sign icon.

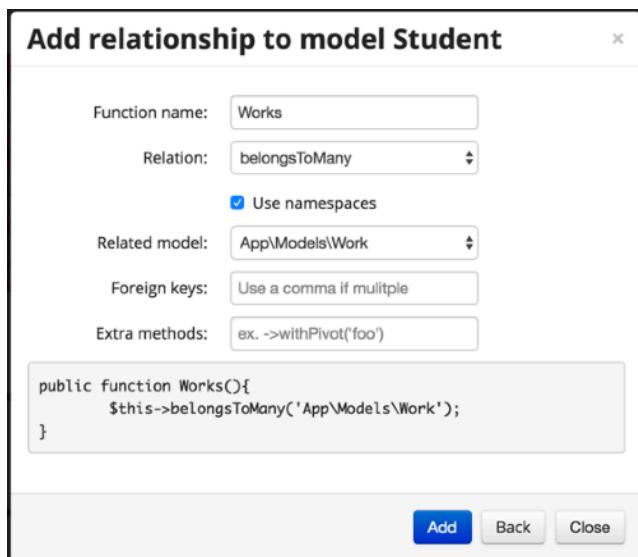
On a la base de notre table. Maintenant, on doit lui ajouter des champs. Pour ce faire, on peut cliquer sur le bouton « plus ». Une popup s'ouvre à nouveau :

The dialog box is titled 'Add column to table students'. It contains fields for Name (firstname), Type (STRING (VARCHAR)), Length (255), Default value (Default), and Enum value (enum1, enum2). Below these are several checkboxes: Auto Incremental, Unsigned Integer, Primary Key, Index, Nullable, Unique Index, Fillable (which is checked), Guarded, Visible, Hidden, and Foreign key. At the bottom are 'Add', 'Add Another', and 'Close' buttons.

À nouveau, on nous demande de remplir de nouveaux champs. Il s'agit de la colonne qui sera ajoutée à la table. Pour l'exemple, j'ai créé une chaîne de caractère, avec maximum de 255. Je lui ai donné un nom et j'ai coché la case « fillable », pour dire qu'elle peut être remplie. Une fois cliqué sur « Add » on peut voir qu'elle s'est bien ajoutée à la table. Après avoir ajouté quelques champs et d'autres tables, on obtient rapidement un schéma de base de données visuel.

Laravel schéma designer permet aussi d'assigner des relations à la Laravel. Reprenons notre table students. Maintenant qu'elle est remplie, je veux lui ajouter une relation **belongsToMany** avec la table **Work**, car un étudiant peut avoir plusieurs travaux. Pour ce faire, rien de plus simple, je vais cliquer sur le bouton de création de relation (celui à droite du nom de la table). Une popup s'ouvre et il

suffit de cliquer sur « Add » pour ajouter une nouvelle relation. Voici ce que l'on obtient après avoir rempli tous les champs.



Comme on le voit, ça respecte vraiment le type de relation de Laravel. Et on a un aperçu de la fonction qui sera créée en dessous. Une fois la relation créée, on peut la voir schématiquement représentée par des lignes qui relient les deux tables.

Vous pouvez accéder au schéma de ma base de données via l'URL suivante : <https://goo.gl/i6DSii>. Toutes les relations ne sont pas établies (notamment pour le blog), mais elle donne une bonne indication de ce que j'ai dû réaliser pour mon application.

Maintenant, les avantages de Laravel Schema Designer ne s'arrêtent pas là. On peut générer des fichiers grâce au bouton « Export ALL ». Exportons par exemple les schémas et allons voir ce qu'il se passe dans ce dossier. On peut remarquer qu'il nous a créé toute une série de fichiers de migrations (qui en Laravel, servent à créer des tables dans la base de données). Ouvrons le fichier concernant les students.

```

<?php

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

class CreateStudentsTable extends Migration {

    public function up()
    {
        Schema::create('students', function(Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->string('slug', 255);
            $table->string('firstname', 255);
            $table->string('lastname', 255);
            $table->string('image', 255)->nullable();
            $table->string('profession', 255);
            $table->integer('year');
            $table->string('orientation', 1000);
            $table->boolean('is_freelance');
            $table->string('company', 255)->nullable();
            $table->string('social', 5000)->nullable();
            $table->boolean('has_interview');
            $table->text('interview')->nullable();
        });
    }

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

```

Comme on peut le voir, ça nous crée complètement le fichier de migration. Il ne reste plus qu'à copier le fichier généré dans Laravel. On peut aussi générer d'autres fichiers, comme le fichier de model ou de contrôleur. Voici l'export du fichier modèle, toujours basé sur la table student.

```

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Student extends Model
{

    protected $table = 'Students';
    public $timestamps = true;
    protected $fillable = array('slug', 'firstname', 'lastname', 'image', 'profession',
'year', 'orientation', 'is_freelance', 'company', 'social', 'has_interview', 'interview');

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

}

```

Comme on peut le voir, le fichier généré contient tout ce qu'un fichier modèle de base doit contenir. Il a même créé la relation pour nous.

Laravel Schema Designer n'est donc pas simplement un outil de schématisation de base de données, mais un réel outil permettant d'intégrer une application Laravel rapidement.

4.3.PRÉSENTATION DE LARAVEL BACKPACK

Avant de commencer à parler de ce que j'ai mis en place, j'aimerais expliquer ce qu'est Backpack et pourquoi je l'ai utilisé.

Concrètement, Backpack se présente comme « The most popular admin panel software for Laravel ». Ils disent qu'avec leur produit, il est possible de créer une page d'administration dix fois plus rapidement qu'avant.

Au départ, j'étais parti pour développer moi-même le design de l'administration et son code, mais j'ai très vite remarqué que je n'aurais pas le temps de tout réaliser. Où en tout cas, réaliser comme j'aimerais le faire ? Je ne voulais pas utiliser de

CMS, car c'est s'enfermer dans une boite, même si certains sont mieux que d'autres. Ce que je voulais, c'était une application souple, qui puissent évoluer facilement dans le temps, avec une architecture claire et un contrôle total sur le backend. Et par-dessus tout, je voulais livrer un site administrables. Je voulais que tout le monde, même sans connaissances, puisse ajouter des cours, des articles, etc. Ce n'aurait pas été une tâche simple si j'avais dû réaliser toute l'administration moi-même. Je me suis alors orienté vers un CMS en Laravel, en me disant que c'était sans doute la bonne solution. J'ai voulu utiliser **Asgard**. Mais je suis rapidement tombé sur Backpack et j'ai tout de suite compris que c'est ce qu'il me fallait. Backpack allait pouvoir me permettre de créer une interface d'administration simplement et permettre à mon client de modifier les données de son site rapidement, simplement et sans se prendre la tête.

Backpack contient toute une série de packages. Le premier à installer est le package appelé « Base ». Il s'agit en fait de l'intégration du thème très connu « Admin LTE » dans Laravel. Seul, il ne permet pas de faire grand-chose, si ce n'est de commencer à développer une interface d'administration avec les caractéristiques de ce thème, mais rien de plus.

Pour installer le package « Base », il faut d'abord avoir une installation de Laravel fonctionnelle. Une fois cela fait, il suffit de suivre la documentation que l'on peut trouver ici <https://laravel-backpack.readme.io/docs/install-on-laravel-54>.

Le deuxième package à installer est le « CRUD », cela signifie **C**reate, **R**ead, **U**pdate, **D**elete. Il s'agit du cœur de Backpack. Il va permettre de réaliser une interface d'ajout/mise à jour et de suppression très rapidement. Parmi ses fonctionnalités, ce package offre plus de 44 types de champs différents, un gestionnaire de média, différents types de colonnes, etc.

Backpack propose aussi différent CRUD prêt à l'emploi, comme un package pour gérer les pages (que j'ai utilisé au début, mais que j'ai abandonné, n'étant pas encore multilingue), un packpage pour gérer les options du site, pour créer un menu, pour gérer les permissions, etc.

Ce que j'ai particulièrement apprécié dans Backpack, c'est sa communauté active. À chaque fois que j'avais une question, j'ai toujours reçu une réponse, que ce soit sur le Github, sur le chat gitter, etc.

Backpack prend de l'ampleur. D'après leur site, 5 % de toutes les nouvelles installations de Laravel utilisent Backpack. Son seul point noir est qu'il n'est pas gratuit. On doit s'acquitter d'une licence de 39 € si on veut l'utiliser pour un projet rémunérant. Mais ça tombe bien, je ne suis pas payé pour réaliser mon projet de fin d'études.

4.4.CRÉATION D'UN PREMIER CRUD

La première chose que j'ai faite une fois Backpack installé, c'est de créer mon premier CRUD. Avant de commencer, j'ai voulu installer une barre d'outils, ce qui allait me permettre de tester plusieurs choses, dont les requêtes SQL que j'allais recevoir, les vues chargées, le nom des routes, etc. Je suis parti sur la barre de Barryvdh, laravel-debugbar. Pour l'installer, rien de plus simple, il suffit de faire un **composer require barryvdh/laravel-debugbar** et de l'ajouter dans les services provider.

Une fois cela fait, il était temps de commencer à faire mes premières lignes de code pour le backend de mon application. Faisons ensemble le premier CRUD que j'ai réalisé, celui pour l'ajout des professeurs.

GÉNÉRER LES FICHIERS DE BASE

Pour créer notre CRUD pour les professeurs, Backpack a besoin de 3 fichiers de base.

- Un fichier model, qui va interagir avec la base de données, en lui indiquant quels champs sont autorisés à être remplis, les différentes relations, etc.
- Un fichier qui sera placé dans App/Http/Controllers/Admin qui s'occupera de la création de la vue dans l'administration et de la gestion CRUD.
- Un fichier Request, qui va interagir avec les différents types de champs présents dans l'administration afin d'y ajouter des contraintes pour que l'utilisateur ne puisse pas remplir n'importe quoi.

Et c'est tout. Backpack n'a besoin de rien d'autre. Il existe un package pour générer des fichiers rapidement, comme le propose **artisan**. Son nom est Backpack Generators. Pour l'installer, il suffit de faire la commande **composer require backpack/generators -- dev** et de suivre les instructions indiquées à l'adresse suivante : <https://github.com/Laravel-Backpack/Generators>.

Maintenant que le package pour générer des fichiers est installé, il est très simple de créer la liste des fichiers. On a plusieurs possibilités. Soit on les crée un par un, par exemple, pour un modèle : **php artisan backpack:model Teacher**. Le générateur va nous créer un nouveau fichier model, prérempli et avec les dépendances dont Backpack a besoin. Où l'on peut utiliser la commande générale qui va créer tous les fichiers dont on a besoin (model, request et CRUDcontroller). Pour ce faire, c'est aussi simple, il suffit de taper la commande **php artisan backpack:crud Teacher** et les trois fichiers sont générés.

LE FICHIER MODEL TEACHER

Maintenant, analysons le fichier model **Teacher** placé dans App\Models

```
<?php

namespace App\Models;

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

class Teacher extends Model
{
    use CrudTrait;

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

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

}
```

Comme on le voit, le fichier est prérempli avec le namespace, le trait CrudTrait, qui va permettre de lier le modèle à la fonctionnalité CRUD de Backpack. Ainsi que des variables standard pour gérer les éléments de la base de données.

Maintenant, ce que je veux, c'est ajouter à ce modèle les fonctions dont j'aurais besoin pour continuer à avancer dans la réalisation de mon CRUD. La première chose à faire est de remplir la variable **\$fillable** avec les différents champs remplissables qui sont dans ma base de données.

```
protected $fillable = [
    'slug',
    'lastname',
    'firstname',
    'role',
    'description',
    'picture',
    'email',
    'social',
    'status',
    'extras'
];
```

Comme on le voit, j'ai un champ slug. Ce slug sera utilisé pour définir une URL lisible. Pour cela, je vais utiliser une dépendance appelée **eloquent-sluggable**. Elle va me permettre de réaliser, sur base des éléments que je lui fournirais, un slug qui sera utilisable dans une URL. Pour cela, je dis à mon fichier modèle d'utiliser l'espace de nom **Cviebrock\EloquentSluggable\Sluggable**. Et de l'utiliser dans ma class Teacher. Je vais créer une méthode **sluggable()** afin de définir quel sera mon slug. Et en parallèle, je vais créer un **accessor** pour que mon **slug** se base sur le champ **'firstname'** et **'lastname'**.

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

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

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

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

Maintenant, quand le formulaire de création d'un professeur sera envoyé, il créera automatiquement un slug sur la base du nom et du prénom.

RENDRE LE MODEL MULTILANGUE

Pour rendre le modèle compatible multilangue, il faut installer un package chez Spatie (et qui a été remanié par Backpack pour fonctionner avec son système de CRUD). Il s'agit de **spatie/laravel-translatable**. Pour rendre le modèle traduisible, il suffit de créer une variable **\$translatable** et de lui passer dans un tableau les champs que l'on veut rendre multilangue, comme ceci :

```
protected $translatable = ['role', 'description'];
```

Le package va alors stocker sous forme d'un tableau dans la base de données les traductions, avec comme clé, la locale. Et en fonction de la locale définie, il va utiliser la bonne clé. Rien de plus simple.

UTILISATION DE L'ATTRIBUTE CASTING

Certains types de champs (notamment les repeaters) seront stockés dans la base de données sous forme de json. Seulement, on veut pouvoir les utiliser comme des éléments d'un tableau ou comme un objet. Pour se faire, utiliser un attribute casting permet de changer le type de donnée pour pouvoir l'utiliser comme on le souhaite. Avec Laravel, c'est très simple à réaliser. Pour la colonne **sociale** où il y aura un repeater, voici ce qu'il faut faire pour l'utiliser comme un tableau.

```
protected $casts = [
    'social' => 'array'
];
```

Ensuite, il serait louable de créer d'autre accès. Par exemple, pour le nom et le prénom. Je veux m'assurer que tous les noms et prénoms commenceront par une majuscule. Et j'aimerais aussi pouvoir récupérer le nom complet sans devoir créer une nouvelle colonne dans ma table. Il est possible de réaliser cela en Laravel.

Voici les méthodes créées :

```

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

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

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

```

Grâce à ces accès, je pourrais récupérer facilement le nom formaté ou le nom complet. Je pourrais maintenant, par exemple, appeler le nom complet comme ceci : `$teacher->fullname` et l'accès se contentera de me retourner la bonne valeur.

GÉNÉRATION DES IMAGES

Maintenant, que reste-t-il à faire dans mon fichier model ? Et bien, on peut remarquer que chaque professeur aura une photo. Il serait bien de pouvoir gérer cette image, car elle sera utilisée à plusieurs endroits différents dans différentes tailles. Et puis, on ne voudrait pas qu'un professeur puisse mettre sa photo de lui en vacances de 10mo, ça plomberait les performances de la page. Avec Backpack, quand on utilise le champ image, il faut créer un mutator (ce qui va se charger de modifier les données reçues) afin de stockées l'image en base de données, car de base, Backpack va essayer de stockée l'image en base64. La fonction de base ressemble à ça :

```

public function setPictureAttribute($value)
{
    $attribute_name = "picture";
    $disk = "public_folder";
    $path = "uploads/teachers";

    // if a base64 was sent, store it in the db
    if (starts_with($value, 'data:image'))
    {
        // 0. Make the original image size and other sizes
        $image = \Image::make($value);
        // 1. Generate a filename.
        $filename = md5($value.time());
        // 2. Store the image original on disk and new sizes.
        \Storage::disk($disk)->put($path.'/'.$filename.'.jpg', $image->stream());

        // 3. Save the path of original image to the database
        $this->attributes[$attribute_name] = $path.'/'.$filename.'.jpg';
    }
}

```

Comme on peut le voir, l'image est stockée dans un dossier défini. Un nom unique lui est attribué. Et le package intervention image sert à créer et à stocker l'image reçue. Du coup, vu que l'on a accès à intervention image, il est très simple de créer différentes tailles d'image différentes. Voici le code avec des tailles différentes :

Je crée différentes instances de l'image basée sur l'originale, que je redimensionne à des tailles définies grâce à la fonction `->fit()`. Et une fois générée, je lui ajoute un suffixe. En base de données, je stocke le nom original. Pour pouvoir accéder à l'image, j'ai besoin de créer une autre fonction, qui se chargera d'appeler l'image avec le bon suffixe. Elle se trouve aussi dans mon modèle. En voici une copie :

Grâce à cette fonction, je pourrais appeler à n'importe quel moment l'image avec le suffixe désiré. Par exemple, dans une vue, je pourrais faire `$teacher->getImageProfile('_120x120.jpg')` et ça me retournera le chemin de l'image avec le suffixe _120x120.jpg.

Et s'il n'y a pas d'images, que se passe-t-il ? Si aucune image n'est envoyée, ou que le professeur n'a tout simplement pas envie de mettre sa photo, une image sera générée aléatoirement sur base de l'API <http://avatars.adorable.io>.

Pour ce faire, je vais créer une méthode presque identique à celle qui va stocker les images en base de données. En voici une copie :

```
|if(strpos($value, 'no-avatar.jpg') !== false || $value == null)
{
    $fakeAvatar = 'https://api.adorable.io/avatars/600/' . Request::get('email') .
'.png';
    $image = \Image::make($fakeAvatar);
    $imageProfile = \Image::make($fakeAvatar)->fit(295,281);
    $imageCards = \Image::make($fakeAvatar)->fit(313,180);
    $fakeAvatar30x30 = \Image::make($fakeAvatar)->fit(30,30);
    $fakeAvatar60x60 = \Image::make($fakeAvatar)->fit(60,60);
    $fakeAvatar120x120 = \Image::make($fakeAvatar)->fit(120,120);

    $filename = md5($fakeAvatar.time());

    \Storage::disk($disk)->put($path.'/'.$filename.'.jpg', $image->stream());
    Utils::storeNewSize($path, $filename, '_profile', $imageProfile);
    Utils::storeNewSize($path, $filename, '_cards', $imageCards);
    Utils::storeNewSize($path, $filename, '_120x120', $fakeAvatar120x120);
    Utils::storeNewSize($path, $filename, '_60x60', $fakeAvatar60x60);
    Utils::storeNewSize($path, $filename, '_30x30', $fakeAvatar30x30);

    $this->attributes['picture'] = $path.'/'.$filename.'.jpg';
}
```

```

if (starts_with($value, 'data:image'))
{
    // 0. Make the original image size and other sizes
    $image = \Image::make($value);
    $imageProfile = \Image::make($value)->fit(295,281);
    $imageCards = \Image::make($value)->fit(313,180);
    $image120x120 = \Image::make($value)->fit(120,120);
    $image60x60 = \Image::make($value)->fit(60,60);
    $image30x30 = \Image::make($value)->fit(30,30);
    // 1. Generate a filename.
    $filename = md5($value.time());
    // 2. Store the image original on disk and new sizes.
    \Storage::disk($disk)->put($path.'/'.$filename.'.jpg', $image->stream());
    Utils::storeNewSize($path, $filename, '_profile', $imageProfile);
    Utils::storeNewSize($path, $filename, '_cards', $imageCards);
    Utils::storeNewSize($path, $filename, '_120x120', $image120x120);
    Utils::storeNewSize($path, $filename, '_60x60', $image60x60);
    Utils::storeNewSize($path, $filename, '_30x30', $image30x30);

    // 3. Save the path of original image to the database
    $this->attributes[$attribute_name] = $path.'/'.$filename.'.jpg';
}

```

Comme vous pouvez le voir, je génère un avatar sur base de l'adresse email du professeur. Je stocke ensuite cet avatar dans la base de données et je le manipule afin de l'avoir en différentes tailles.

L'une des dernières étapes est de créer les différentes relations qu'il y aura. Les

```

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

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

```

professeurs pourront être liés à des cours et des cours à des professeurs. Et il peut y avoir plusieurs cours pour un même professeur. Pour ce faire, j'ai besoin d'une relation **belongsToMany**. J'ai aussi besoin d'une relation avec les articles, car un

professeur pour être l'auteur d'un article. Et comme il peut en avoir plusieurs, mais qu'un article ne peut avoir qu'un seul auteur, j'ai besoin de créer une relation **hasMany**.

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

public function articles()
{
    return $this->hasMany('App\Models\Article');
}
```

La dernière chose à faire sur mon fichier modèle est de changer la clé qui sera définie pour la route. En effet, on a créé un slug pour que l'URL ne soit pas un ID, mais quelque chose de lisible. Par défaut, les routes de Laravel utilisent les ID, ici, on va lui dire qu'il faut utiliser le slug.

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

Voilà, notre premier modèle est maintenant créé et tous les autres ressembleront fortement à celui-là (c'est ce qu'il y a de bien avec Laravel). Bien sûr, en fonction des besoins, ça va changer, mais je ne vais pas tous les énumérer ici, sinon, mon rapport fera 400 pages. Maintenant, regardons de plus près comment créer une page d'administration avec Backpack. Vous pouvez retrouver la dernière version de ce code en ligne sur mon Github à l'adresse suivante : <https://goo.gl/gft2F6>

CRÉATION DE NOTRE PREMIÈRE PAGE D'ADMINISTRATION

Allons maintenant voir ce qu'il se passe du côté du controller de l'administration. Avec Backpack, ils sont tous placés dans App/Http/Controller/Admin et ouvrons le fichier TeacherCrudController. Voici ce que l'on peut trouver à l'intérieur de la classe :

```
public function setUp() {

    /*
    |--------------------------------------------------------------------------
    | BASIC CRUD INFORMATION
    |--------------------------------------------------------------------------
    */

    $this->crud->setModel("App\Models\Teacher");
    $this->crud->setRoute("admin/teacher");
    $this->crud->setEntityNameStrings('teacher', 'teachers');

    /*
    |--------------------------------------------------------------------------
    | BASIC CRUD INFORMATION
    |--------------------------------------------------------------------------
    */

    $this->crud->setFromDb();

}

public function store(StoreRequest $request)
{
    return parent::storeCrud();
}

public function update(UpdateRequest $request)
{
    return parent::updateCrud();
}
```

On remarque que dans la méthode **setUp()** on donne quelques informations au contrôleur, qui utilise le CRUD de Backpack. On lui passe le fichier model que l'on veut utiliser grâce à la fonction **setModel()**. Ensuite, on lui donne la route que l'on veut utiliser pour afficher la page d'administration des professeurs. Et enfin, on peut modifier les chaînes de caractères qui seront utilisés pour, par exemple, le bouton d'ajouter d'un professeur.

En dessous et de base, avec la fonction **setFromDb()**, on dit à Backpack de créer des colonnes (pour la page vue de l'administration) de toutes les entrées dans la

base de données. On va voir comment modifier ça plus tard. Ensuite, il y a une fonction store et update, qui s'occupe de stocker les informations dans la base de données et de les mettre à jour. Elles font appel au CRUD de Backpack aussi, mais sont facilement modifiables. La première chose à faire pour que notre page s'affiche correctement, c'est de créer la route. Comme Backpack le propose par défaut, on va créer une route avec le chemin **admin/teacher**.

Rendez-vous dans le fichier routes/web.php. Pour mon projet, j'ai décidé de créer un groupe de route, car toutes les pages d'administration auront le même préfixe (admin). Voici comment procéder pour créer une route pour la page d'administration de Backpack :

```
Route::group(['prefix' => 'admin', 'middleware' => ['web', 'auth'], 'namespace' => 'Admin'], function()
{
    CRUD::resource('teacher', 'TeacherCrudController');
});
```

Comme on le voit, c'est spécifique à Backpack, mais c'est très simple.

Maintenant que notre route est opérationnelle, on peut y accéder grâce à l'URL **admin/teacher**. On peut aussi ajouter un élément au menu, pour que ce soit plus utilisable (ressources/views/vendor/backpack/base/inc/sidebar.blade.php).

Après ça, nous pouvons accéder à notre page d'administration. Pour le moment, elle est vierge remplie avec toutes les colonnes disponibles en base de données. Retirons la fonction **setFromDb()** et ajoutons-lui quelques champs. Comment faire ? Et bien, Backpack propose plus de 44 types de champs différents. Il n'y a plus qu'à choisir. Voici comment ajouter un simple champ de type text.

```
$this->crud->addField
([
    'name' => 'firstname',
    'label' => 'Prénom',
    'tab' => 'Contenu'
]);
```

Comme on le voit, on appelle la fonction **addField()** du CRUD. On lui passe un **name**, qui est le nom de la colonne en DB, un **label** qui sera affiché au-dessus du champ, et on peut éventuellement lui passer un **tab**, qui servira à séparer le contenu de l'administration en différents onglets.

Voici ce que l'on obtient dans la vue d'édition :

Prénom

Rosamond

Après, il existe différents types de champs, en voici une brève sélection (que l'on peut retrouver à l'adresse suivante : <https://goo.gl/tL8sTa>

address	hidden	simplemde
browse	icon_picker	summernote
base64_image	image	table
checkbox	month	text
checklist	number	textarea
checklist_dependency	page_or_link	time
ckeditor	password	tinymce
color	radio	upload
color_picker	range	upload_multiple
custom_html	select (1-n relationship)	url
date	select2 (1-n relationship)	video
date_picker	select_multiple (n-n relationship)	view
date_range	select2_multiple (n-n relationship)	week
datetime	select_from_array	wysiwyg
datetime_picker	select2_from_array	
email	select2_from_ajax	
enum	select2_from_ajax_multiple	

Comme on peut le voir, il en existe pas mal. Du simple champ URL au Wysiwyg, en passant par des champs qui prennent en compte les différents types de relations.

Bien sûr, un jour, on pourrait avoir besoin de créer son propre champ et c'est très simple à réaliser.

Voici une copie de certains champs qui sont présents dans mon CRUD controller de mes professeurs (vous pouvez retrouver la version complète à l'adresse suivante <https://goo.gl/xNXTZI>).

```
$this->crud->addField([
    'name' => 'description',
    'label' => 'Description',
    'type' => 'textarea',
    'hint' => 'Description du professeur qui apparaîtra sur son profile',
    'attributes' => [
        'style' => 'min-height: 150px'
    ],
    'tab' => 'Contenu'
]);
$this->crud->addField
([
    'name' => 'email',
    'label' => 'L'adresse email du professeur',
    'type' => 'email',
    'tab' => 'Contenu'
]);
$this->crud->addField([
    'name' => 'picture',
    'label' => 'Photo du professeur',
    'type' => 'image',
    'upload' => true,
    'crop' => true,
    'default' => 'img/no-avatar.jpg',
    'tab' => 'Contenu',
]);
$this->crud->addField
([
    'label' => 'Sélectionnez les cours du prof',
    'type' => 'select2_multiple',
    'name' => 'courses',
    'entity' => 'courses',
    'attribute' => 'title',
    'model' => "App\Models\Course",
    'pivot' => true,
    'tab' => 'Contenu'
]);
```

Analysons le dernier champ sur l'image. Il s'agit d'un champ select multiple. Il va servir à, sur la page d'un professeur, pouvoir lui ajouter différents cours. Avec ce champ, on peut interagir avec d'autres fichiers model. Dans **name** et **entity**, on lui

passe le nom de la relation que l'on a définie plus haut dans le fichier model Teacher. Ensuite, on lui donne un attribut, c'est l'élément qui sera affiché dans le select (nous voulons afficher le titre du cours). Et enfin, on lui renseigne le model que l'on veut cibler, ici, c'est le modèle des cours en indiquant qu'il y a une table pivot.

On obtient un champ comme ceci :

Omnis itaque quibusdam est.

Debitis vero ducimus qui.

Deserunt est minus iusto.

In et ipsum neque ipsa et.

Quia voluptas aut magni sed.

Corporis minima porro quos.

Ut totam quidem quo est.

Etiam invenimus.

* Qui rerum animi qui aut.

On peut voir que l'on obtient la liste de tous les cours et que l'on peut en sélectionner plusieurs. Et lorsque l'on enregistrera, tout s'enregistrera en DB dans la table pivot. Simple comme bonjour.

GESTION DES ERREURS

Maintenant que notre page d'administration est créée, il serait bien de pouvoir gérer les erreurs sur ces différents types de champs. C'est très facile avec Laravel et le générateur de fichiers de Backpack crées pour nous un fichier dans App/Http/Requests appelé TeacherRequest. Après, il suffit de faire comme quand on déclarerait des règles avec Laravel, c'est de lui les passer dans un tableau. Voici ce que ça donne une fois rempli, pour le CRUD des professeurs.

```
public function rules()
{
    return [
        'slug'      => 'unique:teachers,slug,' . \Request::get('id'),
        'lastname'  => 'required|min:2|max:255',
        'firstname' => 'required|min:2|max:255',
        'role'      => 'required',
        // 'picture'  => 'required',
        'description' => 'required|min:2',
        'email'     => 'email',
        'status'    => 'required',
    ];
}
```

Une fois que l'on enverra notre formulaire de création ou d'édition sans respecter l'une des règles déclarées, la page nous retournera des erreurs sur les champs. Vous remarquez que je n'ai pas rendu requise la colonne 'picture'. Si vous vous souvenez, nous générerons un avatar pour l'utilisateur s'il n'a pas d'images.

CRÉATION DU CONTRÔLEUR

On vient de créer notre première page d'administration et tout s'enregistre bien en base de données. Maintenant, il ne reste plus qu'à créer du code pour que les données soient rendues dans une vue. Pour cela, on doit créer un contrôleur grâce à la commande **php artisan make:controller Teacher**. Par exemple, voici à quoi ressemble ma méthode **show()**, qui va se charger d'afficher les données d'un professeur.

```
protected function show(Teacher $teacher)
{
    SEO::setTitle($teacher->fullname);
    SEO::setDescription('La page de ' . $teacher->fullname . ' ' . strtolower($teacher->role) . ' à la Haute École de la Province de Liège en infographie');

    return view('posts.postTeacher', [
        'teacher' => $teacher
    ]);
}
```

Dans ma méthode, je lui passe comme argument le model Teacher auquel j'assigne une variable.

Ensuite, comme on peut le voir, j'utilise une classe **SEO**. Il s'agit d'un petit package que j'ai installé pour gérer simplement et efficacement tout le S.E.O. sur mon site. Grâce à elle, je peux lui définir un title, une meta-description, etc., et ça va se charger de générer toutes les meta-og pour les réseaux sociaux, pour Google, etc. C'est propre et simple à utiliser. Pour en savoir plus sur ce package, qui se nomme seotools, vous pouvez voir ses fonctions sur cette page Github <https://goo.gl/2xleM5>.

Ensuite, je retourne simplement la variable **teacher**, afin que je puisse accéder aux données via ma vue blade **postTeacher**. Pour voir le fichier complet, rendez-vous à cette adresse : <https://goo.gl/SiAewS>.

4.5. RÉALISATIONS DES ÉTUDIANTS

Pour les réalisations des étudiants, j'ai dû réaliser quelque chose d'un peu spécifique. En effet, dans la vue d'une réalisation, j'ai prévu que l'on puisse utiliser un slider. Ce slider permettrait d'avoir un nombre indéfini d'images à présenter.

La première chose que j'ai voulu faire, c'est de pouvoir y ajouter des vidéos en provenance de YouTube et des modélisations 3D en provenance de Sketchfab. Pour les vidéos, j'ai utilisé le type de champ video de Backpack. Voici comment on fait pour l'initialiser dans la vue de création/édition.

```
$this->crud->addField( [
    'name' => 'video',
    'label' => 'Link to video file on Youtube or Vimeo',
    'type' => 'video',
])
```

Et comme le champ est bien fait, il stocke en base de données toutes les informations relatives à la vidéo, comme son image de Prévisualisation, son titre, etc.

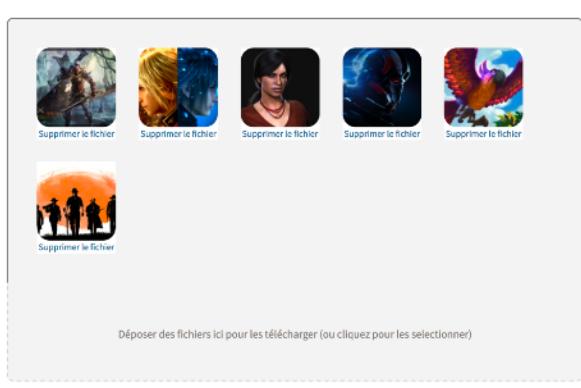
```
{
    id: 234324,
    title: 'Pirates de Caraïbes',
    image: 'https://provider.com/image.jpg',
    url: 'http://provider.com/video',
    provider: 'youtube'
}
```

Pour le champ sketchfab, j'ai utilisé un simple champ URL. Il suffit de passer l'URL de la modélisation dans ce champ. Et comme l'API fournit du contenu embarqué (**oEmbed**), je n'ai qu'à récupérer l'URL embarquée, traiter les informations reçues (du **JSON**) et retourner la miniature dans la vue.

```
$oEmbedUrl = 'https://sketchfab.com/oembed?url=' . $work->view3d;
$obj = json_decode(file_get_contents($oEmbedUrl), true);
```

Ensuite, j'ai du réaliser un type de champ galerie. Comme il n'est pas disponible de base, j'ai dû jouer avec plusieurs choses. Il y a déjà eu pas mal de discussions sur le github de Backpack à propos de ce genre de type de champ.

Il s'agit d'un champ dropzone qui ressemble à cela :



Grâce à ce champ, l'utilisateur peut ajouter des images par glisser/déposer. Il est encore imparfait, pour le moment on sait ajouter des images, les supprimer de la base de données, mais je n'ai pas été plus loin (donc pour le moment, les images qui se stockent sur le serveur restent même si on les supprime). En plus du champ que j'ai dû créer, j'ai aussi du faire une route pour que ça fonctionne, car le dropdown utilise de l'AJAX. Je ne vais pas détailler tout le code, car c'était assez fastidieux. Mais maintenant, il peut être utilisé sur tout le site, il suffit juste de modifier le chemin de la route. Voici les paramètres qu'il prend :

```
$this->crud->addField([
    'name' => 'images',
    'label' => 'Photos',
    'type' => 'dropzone',
    'prefix' => 'public_folder',
    'upload-url' => '/' . config('backpack.base.route_prefix') . '/media-dropzone',
    'tab' => $media
]);
```

Ce n'est pas la solution parfaite, loin de là, mais c'est fonctionnel et c'est ce qui compte pour le moment.

4.6. CRÉATION DU BLOG ET AUTRE FONCTIONNALITÉS

Le blog a été l'une des plus grosses parties du travail backend. J'ai commencé par générer les fichiers pour l'ajout d'un article puis par remplir le fichier modèle Article comme on a pu le voir pour le model Teacher. J'ai fait en sorte que l'on puisse avoir des articles **publié** ou **brouillon**. Ça permettrait de ne pas devoir écrire un article en une seule fois, ce qui est un plus pour l'utilisateur. Pour que ça fonctionne, je devais faire une requête **SQL** qui ne devrait afficher que les articles avec le statut **PUBLIÉS** et une date inférieure à la date indiquée dans l'article (car

j'ai prévu un champ date, qui permet de donner une date de parution à l'article. Je ne voulais pas me baser sur le traditionnel « created_at », car ça posait plusieurs problèmes. Notamment si un article est un brouillon, le « created_at » serait celui de la création du brouillon et non de la parution de l'article. Et ensuite, il est quand même agréable de pouvoir donner une date de parution à un article et ainsi, pouvoir créer plusieurs articles qui seront mis en ligne à intervalle voulue.

Comme il va s'agir d'une requête que je vais réutiliser à plusieurs reprises (vérifier si l'article est publié et que la date n'est pas dans le futur) j'ai créé ce que l'on appelle un scope. Grâce à cette méthode dans mon modèle, je pourrais l'utiliser partout dans mes contrôleurs simplement afin de vérifier si l'article doit être affiché. Elle ressemble à ceci :

```
public function scopePublished($query)
{
    return $query->where('status', 'PUBLIÉ')
        ->where('date', '<=', date('Y-m-d'))
        ->orderBy('date', 'DESC');
}
```

Je n'aurais plus qu'à faire, par exemple `$article->published()` pour avoir un article publié. Comme on peut le voir dans cette méthode, j'ordonne aussi les articles du plus récent au plus ancien.

J'ai créé d'autres méthodes, par exemple une pour récupérer une date lisible par les humains (j'ai utilisé Carbon, qui est embarqué dans Laravel). Cette méthode va me permettre de pouvoir afficher la date comme « 28 mai 2017 ».

```
public function getDateForHuman()
{
    if (App::getLocale() == 'fr') {
        setlocale(LC_TIME, 'fr_FR');

        return Carbon::parse($this->date)->formatLocalized('%d %B %Y');
    } else {
        return Carbon::parse($this->date)->formatLocalized('%d %B %Y');
    }
}
```

Je vérifie si la locale est le Français, sinon, je retourne la fonction en Anglais.

Une des autres fonctions qui fait partie du fichier modèle, c'est une qui va me servir pour préremplir les champs du formulaire pour poster des commentaires. J'ai prévu que l'une fois que l'on a posté un commentaire, un cookie s'enregistre (on verra ça plus bas), mais ce que j'aimerais, lors de la validation, c'est si un des champs est faux, le formulaire préremplisse les champs déjà remplis et corrects (pour ne pas perdre tout son commentaire, par exemple). Le modèle ne s'occupe pas de ça, donc je ne vais pas m'étendre sur le sujet de comment réaliser cette opération. Mais seulement, comme j'ai des cookies, ça crée des doublons dans les champs de formulaire. Prenons un exemple. J'écris un commentaire. Je mets un commentaire qui contient seulement une lettre. Ça va me renvoyer une erreur de validation, car j'ai mentionné qu'il en fallait au moins 50. Si des cookies existe déjà (parce que l'utilisateur a déjà écrit un commentaire autre part sur le site), il va me préremplir les champs avec les datas enregistrées dans les cookies. Seulement, le système de formulaire de Laravel va aussi préremplir les champs corrects (Nom et adresse email), ça va créer des doublons. C'est ce problème que la méthode ci-dessous résout.

```
public function setValueCommentForm($data)
{
    if (Request::old($data) && Cookie::get($data) == null)
    {
        echo Request::old($data);
    } elseif (Cookie::get($data) !== null){
        echo Request::cookie($data);
    }
}
```

J'ai aussi créé des relations pour interagir avec les auteurs, les commentaires, les tags, etc. Vous pouvez voir le fichier complet à l'adresse suivante : <https://goo.gl/6F74oG>.

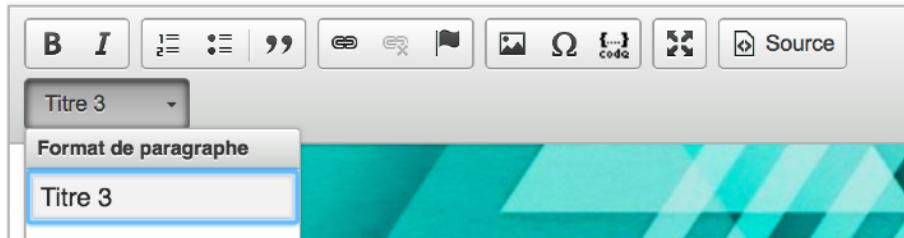
J'ai évidemment créé d'autres fichiers model pour ce blog. Un pour les catégories, un autre pour les tags, etc.

L'ADMINISTRATION

La deuxième grosse partie a été de créer l'administration. Je voulais un blog qui convient aux trois options. Pour moi, on devait pouvoir ajouter des images

légendées ou non, mettre en forme son texte avec différents styles (gras, italique, souligné...), des blocs de citations...

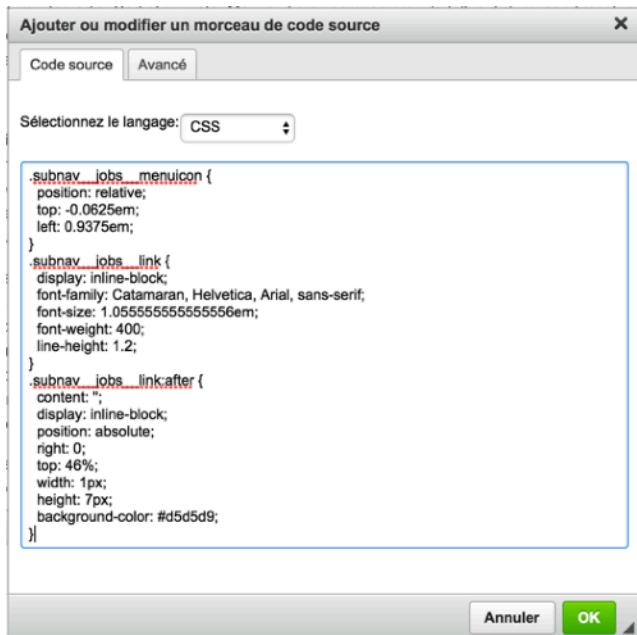
Backpack fournit différents Wysiwyg. Dont un qui s'appelle **ckeditor**. J'ai donc pris en mains ce Wysiwyg en parcourant son API et j'ai réussi à faire un peu tout ce que je voulais. Il existait différents plug-ins que je me suis amusé à tester. J'ai parcouru les fichiers de configuration pour avoir une barre de choix simple et efficace.



Je n'ai retenu que ces quelques options, même si ckeditor en proposait beaucoup plus. Dans chaque article, on peut ajouter des sous-titres. Ce seront automatiquement des titres de niveau 3 (pour respecter le plan de mon document). J'ai gardé le style (que j'ai stylé en CSS pour mon article) bold, italique, les listes, les blocs de citations, les liens, j'ai utilisé un plug-in pour pouvoir ajouter des images légendées avec des figures (à la base, le ckeditor est livré sans).

Lorsque l'on upload une image, ça ouvre le média manager. On peut ensuite y déposer son fichier et le sélectionner pour l'intégrer dans le champ. Les images sont automatiquement redimensionnées à la taille d'affichage dans la page.

J'ai prévu aussi une autre option. J'ai utilisé un SyntaxHighlighter pour les blocs de codes. Cette fonctionnalité était pour moi indispensable, un blog de veille sur le web sans pouvoir y insérer des morceaux de codes, c'est contre-productif. J'ai donc utilisé l'API SyntaxHighlighter (<https://goo.gl/ooScqn>). Pour ajouter du code dans un article, on clique sur le bouton code. Une popup apparaît nous demandant de choisir son langage et notre morceau de code. Il suffit de simplement le coller dans le textarea. Voici ce que ça donne pour du CSS :



Comme on le voit, il n'y a pas de coloration syntaxique. C'est normal, le code JavaScript pour la reconnaissance ne fonctionne qu'au niveau front. Maintenant que le code est ajouté, voyons ce que cela donne dans la vue.

```

vitae vel eros. Quisque semper sapien odio, nec gravida libero dignissim lacinia. Nam
venenatis nulla rutrum dolor laoreet ultricies.

1  .subnav__jobs__menuicon {
2      position: relative;
3      top: -0.0625em;
4      left: 0.9375em;
5  }
6  .subnav__jobs__link {
7      display: inline-block;
8      font-family: Catamaran, Helvetica, Arial, sans-serif;
9      font-size: 1.05555555555556em;
10     font-weight: 400;
11     line-height: 1.2;
12 }
13 .subnav__jobs__link:after {
14     content: '';
15     display: inline-block;
16     position: absolute;
17     right: 0;
18     top: 46%;
19     width: 1px;
20     height: 7px;
21     background-color: #d5d5d9;
22 }

```

Etiam ultricies blandit est, ut convallis dolor fermentum vitae. Duis porta egestas ex at
gravida. Aenean in cursus quam, at finibus metus. Etiam sollicitudin metus sed erat tristique

Comme on le voit, on a un beau morceau de code (que j'ai stylé, s'il vous plaît !). Il existe plusieurs options, on peut par exemple mettre des lignes en évidence grâce à l'onglet « Avancé ».

```

4   left: 0.9375em;
5 }
6 .subnav__jobs__link {
7     display: inline-block;
8     font-family: Catamaran, Helvetica, Arial, sans-serif;
9     font-size: 1.05555555555556em;
10    font-weight: 400;
11    line-height: 1.2;
12 }

```

Et on a aussi d'autres options, comme masquer les numéros de lignes... La plupart des langages du web sont pris en charge. Le PHP, l'HTML, le SQL, le JavaScript, etc. Les préprocesseurs ne sont pas pris réellement en charge, mais la coloration syntaxique pour Sass, par exemple, est la même que pour le CSS, donc il suffit de sélectionner le CSS comme langage voulu. Pour modifier un code existant, il suffit de le sélectionner dans le Wysiwyg et de cliquer à nouveau sur le bouton code.

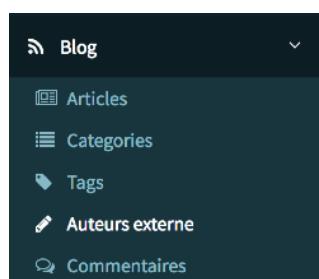
DIFFÉRENTS AUTEURS

À la base, je n'avais prévu qu'un seul type d'auteur. Les professeurs. Quand on ajoute un professeur dans l'administration, on peut ensuite le marquer comme auteur d'un article, grâce au champ adéquat dans l'administration d'un article.

Auteur de l'article (si professeur)

Dominique Vilain
Ettie Thiel
Mafalda Hansen
Kendall Schneider
Reilly Cremin
Freeman Gerhold
Oceane Pacocha
Dominique Vilain

Vous remarquerez que l'on ne peut sélectionner qu'un seul auteur d'article. Cependant, cette approche ou seuls les professeurs pouvaient écrire des articles n'étaient pas forcément la plus ouverte. J'aurais aimé, par exemple, en tant qu'élève, avoir le droit d'écrire un article à la demande de mes professeurs. Pour cela, j'ai prévu un système d'auteur externe. J'ai créé un nouveau CRUD que j'ai lié au model article, ce qui permet dans l'administration de pouvoir ajouter des auteurs qui ne seront pas des professeurs.



Et dans la vue d'un article, on peut ajouter un auteur externe de la même manière que l'on aurait ajouté un professeur.

TAGS ET CATÉGORIES

Sur tous les blogs, il y a des catégories et des tags (mots-clés). J'ai voulu faire de même pour celui que j'ai créé pour l'école, à l'inverse que j'ai voulu scinder les catégories et les sous-catégories. Une catégorie de base c'est une option. Dans un article, on nous demande de sélectionner l'option auquel appartient cet article (web, 2D ou 3D). Il s'agira de la catégorie principale. Il sera simple d'en ajouter une 4e ou une 5e en ajouter simplement une entrée dans le type de champ select. Mais à priori, ce ne sont pas des catégories qui vont souvent bouger.

Ensuite viennent les sous-catégories. Pour ceux-là, j'ai créé un CRUD et on peut facilement ajouter une liste de catégories. J'en ai créé quelques une pour tester.

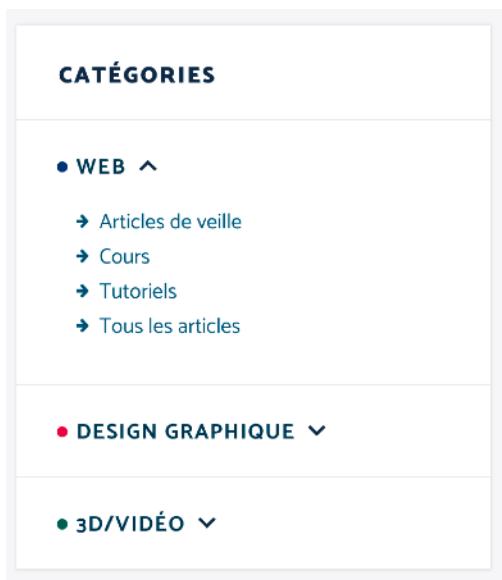
Categories		
		ACTIONS
Actions		ACTIONS
		Ajouter category Réordonner categories
25	enregistrement par page	
Name	Parent	Actions
Articles de veille		Modifier Supprimer
Tutoriels		Modifier Supprimer
Cours		Modifier Supprimer
Concours		Modifier Supprimer
Toto		Modifier Supprimer
Name	Parent	Actions
Affichage des éléments 1 à 5 sur 5		

Et dans un article, on peut sélectionner la sous-catégorie auquel appartient l'article (il en aura toujours une).

Sous-catégorie

- Articles de veille
-
- Articles de veille
- Tutoriels
- Cours
- Concours
- Toto

Dans le blog, on peut choisir de trier les articles par sous-catégories et par catégories.



Si on crée une nouvelle catégorie et que l'on marque un article d'une section faisant partie de cette sous-catégorie, l'entrée sera automatiquement ajoutée à cette liste.

J'ai également créé un CRUD pour les tags. Pour ajouter, c'est le même système que pour les catégories. Et à la fin de chaque article, on a une liste des mots-clé qui lui appartiennent. Et cliquer sur un de ces liens affiche tous les articles qui ont ce mot clé.

LES COMMENTAIRES

Comme j'en ai déjà brièvement parlé plus haut, j'ai offert la possibilité aux visiteurs de laisser des commentaires. Je ne souhaitais pas que chaque visiteur doive s'inscrire pour en poster un. Je demande simplement un nom d'utilisateur et une adresse email.

Pour pouvoir poster mes commentaires, j'ai du créer contrôleur avec une méthode **store ()**. En voici une retranscription :

```

public function store(Request $request, $article_id)
{
    $validator = Validator::make($request->all(), [
        'user_name' => 'required|min:2|max:255',
        'email'      => 'required|email|max:255',
        'content'    => 'required|min:10|max:4000'
    ]);

    $article = Article::find($article_id);

    $comment = new Comment();
    $comment->user_name = $request->input('user_name');
    $comment->email = $request->input('email');
    $comment->content = $request->input('content');
    $comment->article()=>associate($article);

    if ($validator->fails())
    {
        return redirect()->to(route('article.single', [$article->slug]) . '#comment')
            ->withInput()
            ->withErrors($validator);
    }

    $comment->save();

    \Session::flash('success', 'Votre message a été posté. Merci !');

    Cookie::queue(Cookie::forever('user_name', $comment->user_name));
    Cookie::queue(Cookie::forever('email', $comment->email));

    return redirect()->to(route('article.single', [$article->slug]) . '#comment');
}

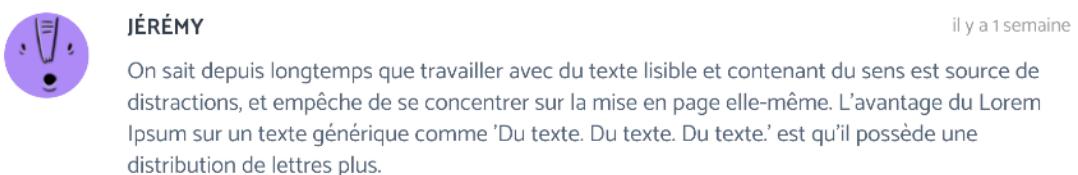
```

Dans cette méthode, je fais plusieurs choses. J'utilise le validator de Laravel, qui permet de faire comme avec les fichiers Request dont j'ai parlé plus haut, à savoir, gérer les erreurs des formulaires.

Je crée un nouvel objet **Comment()**, et je lui passe toutes les infos reçues des inputs du formulaire de commentaires. S'il y a des erreurs, je lui demande de me retourner la page avec les erreurs et la valeur des inputs correctement remplis. S'il n'y en a pas, je sauvegarde le commentaire en base de données et j'affiche un message de succès.

Ensuite, une fois que le commentaire a été stocké (et que toutes les étapes de validations ont été réussies), je crée deux nouveaux Cookies. Un pour le nom d'utilisateur et un pour l'adresse email. Grâce à ça, le visiteur aura toujours ses identifiants sauvegardés. L'adresse email reçue n'est jamais publiée.

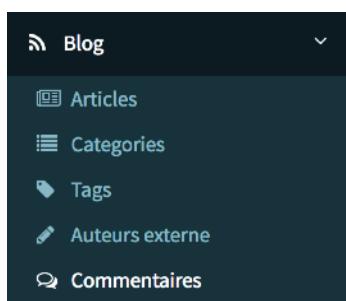
J'ai prévu un avatar pour chaque personne qui poste un commentaire. Comme elle n'a pas de compte utilisateur, j'en génère un aléatoirement avec l'API Adorable Avatar présentée plus haut. L'avatar est généré sur base de l'adresse email. Un commentaire ressemble à ceci :



Évidemment, je ne passe pas réellement l'adresse email dans l'URL de l'API d'Adorable Avatar. Avant de faire cela, je l'encode en md5.

```
user_name }}">
```

Maintenant, que se passe-t-il si un commentaire est jugé indésirable par les administrateurs ? Il serait bien de pouvoir les supprimer simplement sans devoir accéder à la base de données. J'ai donc créé un CRUD simple pour les commentaires, ou ça liste tous les commentaires du blog.



Nom/Pseudo	Adresse e-mail	Sur l'article	Commentaire	Actions
Jimmy Letecheur	letecheur.j@gmail.com	Automatiser son workflow avec Grunt	Quisque tempus sapien risus, in congue risus rhoncus nec. Quisque non mattis nib[...]	Supprimer
Dominique Vilain	domi.vilain@gmail.com	Automatiser son workflow avec Grunt	Curabitur ornare eleifend turpis a faucibus. Mauris aliquet pretium congue. Null[...]	Supprimer
Dylan Schirino	dylan@schirino.be	Automatiser son workflow avec Grunt	Contrairement à une opinion répandue, le Lorem Ipsum n'est pas simplement du tex[...]	Supprimer

On peut facilement grâce à l'interface d'administration supprimer un commentaire existant. J'aurais pu prévoir une option de validation de commentaires, mais je n'ai pas jugé cela utile. Ce sont des articles techniques, il n'y a aucun contenu sensible. Mais j'ai tout de même voulu apporter une gestion des commentaires via l'interface d'administration. À la base, on pouvait aussi modifier le commentaire directement via l'interface d'administration, mais j'ai désactivé l'option.

LA RECHERCHE

Dans tous les blogs, on doit savoir rechercher des articles simplement, que ce soit par mots clé, par thème, par catégories, etc. La première étape a été de créer un formulaire de recherche. Mais je le voulais plus poussé que sur la plupart des blogs. J'ai fait en sorte que l'on ait une autocomplétion en AJAX sur la recherche d'article. Lorsque l'on écrit différents mots clés, on reçoit une liste de titre d'articles.



Pour ce faire, j'ai utilisé AJAX. Je ne vais pas parler du code JavaScript, mais je vais vous montrer ce que j'ai fait dans mon contrôleur.

En voici une copie :

```

public function autocomplete(Request $request)
{
    $term = $request->get('term');
    $keywords = explode(" ", $term);

    $filtered = ["avec", "an", "the"];
    $filteredKeywords = array_diff($keywords, $filtered);

    $article = Article::query();
    foreach($filteredKeywords as $word){
        $article->orWhere('title', 'LIKE', '%'.$word.'%');
    }
    $queries = $article->published()->get();

    foreach ($queries as $query) {
        $results[] = ['slug' => $query->slug, 'value' => $query->title];
    }

    return \Response::json($results);
}

```

Comme on peut le voir sur ce bout de code, j'ai créé une méthode autocomplete. Je récupère les mots clés passés dans l'input de recherche et je sépare les mots. J'ai décidé d'ajouter un tableau pour filtrer les différentes prépositions, comme cela, un "je" ou un "le" ne sera pas pris en compte dans la requête. Je tiens à indiquer que de toute manière, dans le code JavaScript, la requête ajax commencent à renvoyer des données à partir de la troisième lettre. J'ajoute aussi que si le premier mot est "avec" il sera quand même pris en compte tant qu'il n'y a pas d'autre terme qui le suive.

Dans la suite de mon code, je crée un tableau qui va filtrer tous ces termes, et je prépare mes requêtes SQL dans un foreach (car il y en aura plusieurs, une pour chaque terme) (en réalité, réellement une seule, mais actualités à chaque fois).

Si j'écris les mots clés **workflow avec grunt** on peut voir que dans la requête SQL qui est retournée, on a bien le mot workflow et le mot grunt. « Avec » est ignoré, car il est passé dans le tableau des termes à ignorer.

A screenshot of a search interface. At the top is a search bar with the placeholder "Je recherche..." and the query "workflow avec grunt". To the right of the search bar is a blue search button with a white magnifying glass icon. Below the search bar is a search result card. The card has a title "Automatiser son workflow avec Grunt" and a small thumbnail image.

```
select * from `articles` where (`title` LIKE '%workflow%' or `title` LIKE '%grunt%') and  
`status` = 'PUBLIÉ' and `date` <= '2017-05-29' order by `date` desc
```

Évidemment, le formulaire de recherche n'est pas la seule façon de rechercher un article dans le blog. On peut par exemple, vouloir afficher les articles d'une orientation. Sur une carte de blog, je peux par exemple cliquer sur sa catégorie.

En cliquant dessus, on est dirigé vers une page qui liste tous les articles de la catégorie web.

A screenshot of a blog category page titled "LES ARTICLES DE LA CATÉGORIE WEB". The page lists two articles:

- HTML 5**: An article with a green and blue background featuring the HTML5 logo and various icons. It has a "WEB" tag at the bottom. The text "95% des intégrateurs écrivent mal leurs titres" is visible.
- Automatiser son workflow avec Grunt**: An article with an orange background featuring a cartoon bull's head. It has a "WEB" tag at the bottom.

Both articles have author information: "Par Dominique Vilain" and a small profile picture. The first article has 4 likes and the second has 22 likes.

Si on analyse l'url, on peut lire ceci :

« <http://ecoleinfographie.app/blog?category=web#anchor> »

Je passe un paramètre dans l'URL pour que je puisse le récupérer et afficher les bons articles sur la page grâce à une requête SQL. Voici la méthode :

```
public function searchCategory($request)
{
    $search = $request->get('category');

    $articles = Article::published()
        ->where('orientation', 'LIKE', '%' . $search . '%')
        ->paginate(8);

    return view('pages.blog', $this->getViewData($articles));
}
```

Je récupère le paramètre GET category que j'ai passé à l'URL et j'effectue une requête où je lui demande de ne m'afficher que les articles avec l'orientation qui se trouve dans le paramètre. Ici, c'est web. Je lui demande aussi de me créer une pagination pour chaque 8 éléments. Voici la requête qui est retournée :

```
select * from `articles` where `status` = 'PUBLIÉ' and `date` <= '2017-05-29' and
`orientation` LIKE '%web%' order by `date` desc limit 8 offset 0
```

Ce n'est pas terminé. Dans mon contrôleur, il me reste retourner la bonne valeur en fonction du paramètre qui est passé. J'ai procédé comme suit :

```
if ($request->has('category') && ! $request->has('subcategory')) {
    return $this->searchCategory($request);
}
```

Je dis à ma méthode de ne charger la méthode **searchCategory()** uniquement si le paramètre de l'URL contient une catégorie et ne contient pas de sous-catégorie (car il peut y avoir une catégorie et une sous-catégorie, mais elle n'est pas gérée dans cette méthode-ci). Il existe sans doute d'autre façon de procéder, mais c'est celle qui m'a paru la plus simple à réaliser.

Tout mon projet fonctionne de la même façon lorsqu'il s'agit de trier des données. Vous pouvez accéder au contrôleur complet des articles ici : <https://goo.gl/EcgkMH>.

LE LAZY-LOADING

J'ai décidé d'utiliser du lazy-loading à quelques endroits sur mon site web, pour éviter de devoir utiliser la traditionnelle pagination qui recharge la page à chaque fois. Je trouve ça plus intuitif et plus rapide. J'ai implémenté cette fonctionnalité, par exemple, sur la page des réalisations.

Comme on a une liste de travaux (et d'images), il me semblait opportun de proposer le lazy-loading comme solution. Seulement, avant de commencer, je me suis posé une question « **Est-ce que je fais apparaître les nouvelles images automatiquement au scroll ou est-ce que j'utilise un bouton ?** » J'ai opté pour utiliser un bouton "charger plus". Pour la simple et bonne raison que si l'on a 100 articles, on ne saura pas accéder au footer, de nouveaux articles se chargeant automatiquement.

Voici les différents statuts du bouton (un statut qui attend d'être cliqué, un qui charge, un autre qui dit qu'il n'y a plus de données à charger).



Pour réaliser le lazy-loading, j'ai aussi utilisé de l'AJAX. Et pour garder une comptabilité sans JavaScript, je l'ai implémenté avec la pagination de Laravel. Voici une partie de la méthode qui gère le code du lazy-loading.

```
$works = $query->paginate(11);

if ($request->ajax()) {
    return [
        'works'      => view('partials.realisations.item-realisations',
        [
            'works'      => $works,
            'orientations' => $this->getOrientation()
        ])->render(),
        'next_page' => $works->nextPageUrl() . $this->getLoadMoreLink($request),
    ];
}
```

Ici, je retourne une vue (la vue d'une carte de réalisation) si une requête AJAX existe. Je retourne aussi différentes données, comme l'url de la page suivante afin de l'intégrer au bouton load-more.

D'AUTRES MODULES

Pour ce projet et pour l'administration, j'ai aussi utilisé d'autres modules que Backpack propose. Par exemple, un module pour gérer facilement les options du site. C'est à dire, les textes qui seront utilisés à plusieurs endroits dans le site. Par exemple, les adresses email, les liens de réseaux sociaux, etc.

Comme il s'agit d'un système qui doit être géré par le développeur (vu que les données sont ajoutées dans les vues), elles s'ajoutent via la base de données, mais sont modifiables via l'interface d'administration. Grâce à ce module, qu'il faut préalablement installé, on peut dans n'importe quelle vue appelée appeler la donnée comme ceci : `Config ::get('settings.contact_email')`.

Grâce à ce code, on récupère l'adresse email de contact. Et avec ce module, on peut aussi récupérer des données dans les contrôleurs. Par exemple, j'ai fait un formulaire pour l'envoi d'offre de stage. Et dans ce formulaire, on peut sélectionner les options pour lesquels on veut que l'email soit envoyé.

Je remplis un formulaire détaillé

J'envoie une offre au format PDF

Votre nom

Nom de l'entreprise concernée

Quel est le sujet de votre demande ?

Quel est votre adresse e-mail ?

À quelle option s'adresse votre offre ?

3D ET AUDIOVISUEL

DESIGN GRAPHIQUE

WEB

En cliquant sur une (ou plusieurs) checkbox, on envoie l'offre aux contacts concernés, qui sont les responsables des sections. Dans les options du site, elles sont définies comme cela :

Adresse email du responsable de la catégorie WEB	dominique.vilain@hepl.be
Adresse email du responsable de la catégorie DESIGN GRAPHIQUE	veronique.etienne@hepl.be
Adresse email du responsable de la catégorie 3D	jean.dupont@hepl.be

Et dans mon contrôleur, pour envoyer l'email, je fais ceci :

```
$input3d = $request->input('checkbox1-pdf');
$input2d = $request->input('checkbox2-pdf');
$inputWeb = $request->input('checkbox3-pdf');

$mailWeb = Config::get('settings.email_web');
$mail2d = Config::get('settings.email_2d');
$mail3d = Config::get('settings.email_3d');

$emails = [];

if ($input3d == '3D-pdf') {
    array_push($emails, $mail3d);
}
if ($input2d == '2D-pdf') {
    array_push($emails, $mail2d);
}
if ($inputWeb == 'WEB-pdf') {
    array_push($emails, $mailWeb);
}

Mail::to($emails)
->send(new InternshipMailFile($request));
```

On voit que je peux passer les informations du module d'options directement dedans. Ensuite, je crée un array vide auquel j'injecte les données (l'adresse email) si la case est cochée. À la fin, j'envoie l'email au (x) destinataire (s).

GESTION DES UTILISATEURS ET DES RÔLES

Un site comme j'ai construis va sans doute avoir différents profiles d'utilisateur. Pour bien faire, il faudrait pouvoir gérer leurs droits. Backpack offre un CRUD qui permet d'ajouter des utilisateurs, des rôles et de leur assigner des rôles.

Ce CRUD utilise le très connu spatie/laravel-permission, qui permet d'assigner des rôles et de les gérer simplement. Mais avec Backpack, c'est encore plus simple.

Pour le moment, j'ai créé trois types de rôle. Un rôle pour les administrateurs, un rôle pour les professeurs et un autre pour les invités (ceux qui seraient amené, par exemple, à écrire un article de blog). On peut facilement ajouter des utilisateurs manuellement grâce à ce package.

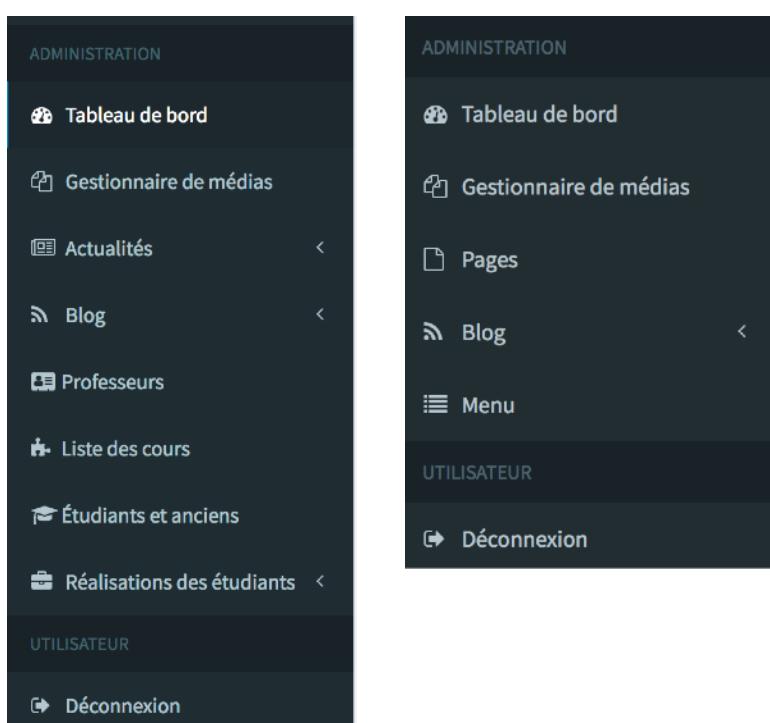
Pour tester, j'ai décidé d'afficher les éléments du menu en fonction du rôle de l'utilisateur. On peut soit utiliser une directive blade :

```
@role('administrateur')  
    // My elements  
@endrole
```

Ou bien d'utiliser des if et des elses en appelant la classe User.

```
@if(auth()>user()>hasRole('administrateur') || auth()>user()>hasRole('professeur'))  
    // My elements  
@endif
```

Voici, par exemple, les éléments que j'ai décidé d'afficher pour un professeur (à gauche) et pour un auteur externe (à droite), il va de soi qu'un administrateur peut accéder à tous les éléments.



Je n'ai pas investigué plus loin avec les rôles utilisateurs car je n'ai pas eu le temps. C'est à améliorer, car un utilisateur connaissant la route pourrait avoir accès au CRUD. Ici, je n'ai fait que masquer les différents éléments du menu. J'aurais aimé en faire plus.

CONCLUSION

Je pourrais continuer à parler de tout ce que j'ai mis en place pendant encore plusieurs pages, mais malheureusement, le temps me manque (et ça aura été une vérité pour toute la durée de la réalisation de mon projet).

Je retiens énormément de choses positives sur ce travail. Je pense avoir réussi à atteindre mes objectifs et à vaincre mes craintes.

Le problème lorsque l'on apprend des choses lors d'une formation, c'est que l'on n'a pas l'habitude d'apprendre par soi-même. Les professeurs sont là et dès que l'on a une question, il suffit de leur poser et la réponse tombe tout aussi vite.

Je me suis lancé dans un framework que je ne connaissais pas, ça n'a pas toujours été facile, mais j'ai réussi à faire tout ce dont j'avais imaginé. À de nombreuses reprises, j'imaginais des solutions et je me disais « **Je veux faire ça, mais je ne sais pas du tout par où commencer** ». L'inconnu fait toujours peur. Mais à force de persévérance et d'envie, on réussit toujours à sortir quelque chose. C'est pourquoi je suis globalement satisfait. Parce que j'ai réussi à apprendre tout seul, à prendre le temps de lire des documentations, à regarder des exemples et à faire de bonne recherche pour avoir les réponses dont j'avais besoin. Je finis ce projet avec l'impression que maintenant, je peux tout apprendre.

Néanmoins, il y a certaines choses pour lesquelles je suis moins satisfait.

La première concerne mon design, qui, à certains endroits, je le trouve très correct et à d'autre un peu moins. Je n'ai pas toujours été minutieux et soucieux du moindre détail. De plus, avec l'expérience que j'ai acquise en stage, il y a beaucoup de choses que je ferais différemment. J'espère seulement que le design en l'état plaira. J'ai pris beaucoup d'expérience là dedans qui me servira pour la suite.

Je regrette aussi le manque de temps. J'ai appris à travailler jour et nuit avec ce projet, ça n'a pas toujours été facile, surtout dans les derniers instants. J'ai dû faire des choix et malheureusement faire l'impasse sur certaines fonctionnalités, comme le fait de pouvoir aimer un article. J'ai aussi du faire l'impasse sur deux pages statiques qui présentent l'école et les installations. Et à mon grand regret, même si j'ai fait attention à beaucoup de détails, je n'ai pas pu faire de tests utilisateurs approfondis.

Mais globalement, je suis heureux d'avoir pu réaliser ce projet. J'ai longtemps imaginé le site idéal pour une école, et je ne dis pas que le mien l'est, mais dans tous les cas, j'ai essayé d'apporter ma vision de ce que ça devrait être. Un site où tout est présenté, avec des actualités, un blog et la possibilité d'avoir des cours en ligne, la présentation des professeurs, des cours (de façon claire et ordonnée), une véritable information sur les différents métiers que l'on peut être à même de réaliser, etc. J'ai donné tout mon temps pour essayer de faire quelque chose de bien et surtout d'utilisable.

Au final, j'ai peut-être été un peu plus loin que ce que je m'étais fixé initialement. Je partais sur la création d'un site en Laravel, mais j'ai fini par essayer de construire un petit CMS je voulais que toutes les actions puissent être faites dans l'administration.

Par facilité, j'aurais pu me dire, comme mon professeur superviseur est développeur et que c'est lui qui sera en charge du projet par la suite, si il voit le jour, « Je ne vais rien faire, il pourra modifier les données dans la base », mais j'ai vraiment voulu lui fournir un produit qu'il pourrait simplement maintenir, que ce soit au niveau du code (Laravel) ou au niveau de l'administration (Backpack).

J'espère que ce rapport n'a pas été trop ennuyeux à lire et qu'il a permis de découvrir certaines fonctionnalités que j'ai mises en place. Je n'ai pas parlé de tout, il y a encore beaucoup de choses à découvrir, mais comme je l'ai déjà répété, le temps a joué contre moi. Et c'est frustrant, car étant de nature perfectionniste, je n'aurais pas eu de trop de six mois supplémentaires.

Et enfin, pour conclure, je pense avoir réalisé mon objectif, fournir à mon client un produit utilisable et facile à maintenir tout en correspondant au cahier des charges. Ce fut un travail très long, stressant, mais terriblement instructif. J'ai appris énormément et je suis prêt à entrer dans le monde professionnel sans les craintes que j'avais avant de commencer.

BIBLIOGRAPHIE

SITES INTERNET

<https://laravel.com/>, le site officiel du framework Laravel.

Consulté régulièrement

<https://backpackforlaravel.com/>, le site officiel de Laravel-Backpack.

Consulté régulièrement

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

Consulté régulièrement

stackoverflow.com, communauté de développeurs.

Consulté régulièrement

laravel.sillo.org/laravel-5, un tutoriel sur Laravel 5.3 en français

Consulté régulièrement

[https://laracasts.com](http://laracasts.com), une série de vidéo sur Laravel

Consulté régulièrement

[https://css-tricks.com](http://css-tricks.com), un site regroupant plusieurs astuces CSS.

Consulté régulièrement

pinterest.com, pour l'inspiration graphique

Consulté régulièrement

behance.com, pour l'inspiration graphique

Consulté régulièrement

caniuse.com, pour le support des propriétés CSS sur les différents navigateurs

Consulté régulièrement

LIVRES

Ergonomie web, 3e édition, Amélie Boucher, Eyrolles, Paris, 2013, 355p.

Design d'expérience utilisateur principes et méthodes UX, Sylvie Daumal, Eyrolles, Paris, 2012, 192p.

CSS 3 Flexbox, plongez dans les CSS modernes, Raphaël Goetter, Paris, 2016, 134p.