www.isl.be

INSTITUT SAINT-LAURENT
enseignement de promotion sociale

rue Saint-Laurent 33 - 4000 LIEGE
04 223 11 31

PicBak

Liège, le 21/01/2021

Travail de fin d'études présenté par

Talbot Eric

En vue de l'obtention du brevet de l'enseignement supérieur de

WebDeveloper

Enseignement supérieur économique de promotion sociale et de type court

Table des matières

1. Les outils utilisés

Avant de débuter mon projet, je devais m'assurer de la qualité de mon environnement de travail car les outils de développement évoluent sans cesse et les plus adaptés facilitent grandement la vitesse de compréhension, de progression et d'adaptation. Pour se faire, les outils utilisés ont été les suivants :

1.1. Le terminal :

L'utilisation de cette commande a été primordiale pour moi. Pour exécuter des commandes, télécharger des dossiers, des fichiers, etc. Connaître les commandes de base de Linux était impératif.

1.2. Composer

Gérer les dépendances de Composer est important pour un projet. Je me suis donc assuré d'installer la dernière version de Composer et vérifier si les dépendances étaient bien compatibles avec ma version php.

1.3. Bundle Psysh :

« bin/console psysh » qui a pour but de créer / d'appeler des noms de variables, des fonctions, des classes, des méthodes, des propriétés et même les fichiers ; très utile pour créer les premières données.

1.4. GitHub :

J'ai utilisé cet outil en permanence car il me semblait le plus adapté tant pour la sauvegarde de mon projet que pour sa modification.

1.5. PhpStorm :

Phpstorm est un IDE de qualité car il regroupe tous les éléments que j'ai cités ci-dessus, ce qui m'a permis d'avancer plus rapidement sur mon projet.

2. L'application du projet

Le projet proposé s'articule comme suit :

« PicBak » se veut être une application mélangeant les concepts de réseautage social et de partage de photographies. Les options principales proposant une recherche avancée adaptée aux intérêts des utilisateurs ainsi qu'un système de « like » dynamique leur permettant de manifester leurs appréciations.

2.1. Modélisation des données :

Avant de me concentrer sur les codes, j'ai commencé par faire des analyses entre les entités pour concevoir la modélisation des données :

- Une image peut avoir une catégorie.
- Un utilisateur peut avoir plusieurs images.
- Un utilisateur peut avoir plusieurs «like» ou non.

3. Première difficulté rencontrée :

Le système de « like » a été un peu complexe car je n'avais pas une idée précise de la relation qu'il devait y avoir entre l'entité User et Image. Dans l'intention de m'assurer qu'un lien entre les deux existait bel et bien, je me suis rendu sur le moteur de recherche Google et j'ai regardé des exemples de « data modeling » avec les mots de recherches « data modeling like doesn't like ». Les réponses que j'y ai trouvées m'ont permises d'arriver à l'analyse suivante :

Une Image peut avoir plusieurs « like » et un Utilisateur peut avoir plusieurs «like».

A partir de ce constat, j'ai su qu'il fallait une entité « like » qui devait lier l'utilisateur et l'image pour savoir quel utilisateur avait aimé quelle image.

3.1. Créations des premières données

Dans un premier temps, j'ai créé mon projet Symfony «picbak_dev».

J'ai créé ma base de données et créer mes entités en utilisant la commande «make:entity»

Avant de créer mes entités, j'ai commencé par gérer le cycle de vie de ces dernières pour qu'elles puissent avoir une date lors de leurs créations et de leurs modifications. A partir de cela, j'ai créé ma base de données et mes entités en utilisant la commande «make:entity» ; l'ensemble me permettant de réaliser mon projet Symfony «picbak_dev»

J'ai donc inclus une méthode qui contient les annotations suivantes :

@ORM\Presist

@ORM\PreUpdate

J'appelle cette méthode et celle-ci va appeler

le setCreatedAt et setUpdatedAt avec une classe DateTimeImmutable

Lors de la création d'une image, deux dates se créent.

Pour la modification de l'image, j'établis une condition pour ne pas modifier la date «setCreatedAt». Pour se faire, j'ai récupéré le «getCreatedAt» et je vérifie dans la méthode si le «getCreatedAt» est « null » ou non.

Ensuite, je crée un fichier «Timestamable.php» et j'englobe la méthode dans ce fichier pour hériter toutes les entités que je vais créer par la suite.

Ma première entité a été l' image que j'ai défini Picture contenant un titre, une description et une image. J'ai ensuite créé mon premier contrôleur « PictureController » pour créer une liste d'images.

N'ayant aucune donnée au départ, j'ai téléchargé un Bundle « Psysh ».

Au lieu de créer des « fixtures » ou de passer par le controleur, j'ai utilisé cette commande pour avoir des fonctionnalités supplémentaires.

Avec cette commande, j'ai pu faire appel à l'entité manager pour faire des « persist » et «flush » manuellement sur mes nouvelles données.

Et j'ai commencé par l'image «Picture»

J'ai réitéré ce mécanisme pour l'entité «User» (utilisateur). J'ai ensuite établi une relation OneToMany (un utilisateur «User» peut avoir plusieurs images «Picture») pour que l'image obtienne un id_user et j'ai réutilisé la commande psysh pour faire un «persist» et un «flush.»

Pour avoir une confirmation et une visualisation de ces étapes, j'ai utilisé la commande mysql pour accéder à ma base de données via le terminal me permettant de m'améliorer au niveau des requêtes SQL.

3.2. Le controleur d'images

Ma première opération a été la gestion des images que j'ai définies en tant que homepage.

Elle reprend également plusieurs fonctions que j'ai créés :

. Un tableau avec toute la liste des images en homepage reprenant une pagination

. Le moteur de recherche avancé qui permet de rechercher le titre d'une image et filtrer les catégories d'images

. La création de la vue d'une image et d'un commentaire

. La suppression d'une image

. Un système de « like » sur les images

3.3. Concernant le moteur de recherche

j'ai créé une «class» SearchData qui permet de représenter les données liées à la recherche.

Puis, j'ai placé cette « class » dans un nouveau « namespace » qui est « app\Data ».

Ensuite, j'ai précisé dans cette class ce dont j'avais besoin, à savoir, un système permettant de rentrer un mot clé. Autrement dit, une propriété publique qui sera une chaine de caractère vide par défaut et une propriété publique qui contiendra un tableau reprenant les catégories sélectionnées dans la recherche.

Ensuite l'objet « SearchData » va pouvoir être utilisé au niveau d'un formulaire de recherche que je vais devoir créer.

Les champs de mon formulaire vont devoir reprendre les champs créés dans le SearchData. Ensuite, j'afficherai mon formulaire de recherche dans la page d'accueil.

J'ai donc dû créer mon formulaire dans mon Controleur d'images « PictureController ».

4. Deuxième difficulté rencontrée :

A ce stade de mon projet, je ne savais plus de quelle manière évoluer. Je savais que je devais créer une fonction dans le Repository des images mais j'ignorais de quelle façon m'y prendre au niveau de la liaison des tables et des conditions. Je me suis donc orienté vers des vidéos tutorielles et de la documentation.

La documentation de Symfony 5 de Fabien Potencier m'a été très précieuse et cela m'a permis de constater l'évolution des techniques et des outils utilisés.

J'ai également rejoint des communautés Symfony sur Discord et Facebook pour obtenir des réponses. Au fur et à mesure, j'ai trouvé une solution en déterminant les liaisons entre les tables (les images et catégories) et les conditions quand les champs étaient non vides.

4.1. Les like dynamiques et les conditions

C’est la première fois que j’expérimente l’emploi des « like » et je ne savais pas comment maîtriser les « like » dynamiques. En effectuant des recherches, j’ai trouvé une solution sur Internet qui expliquait la manière de procéder étape par étape et pour laquelle Ajax et Axios étaient utilisés.

J'ai eu donc la chance d'utiliser Ajax et Axios.

Ajax permet de modifier partiellement la page affichée et Axios est une bibliothèque qui m’a permis de faire des requêtes qui applique différentes méthodes get, put, post ,delete etc.

L'objectif est de déclencher une requête Ajax venant du bouton clic parte depuis Javascript jusqu à Symfony pour prévenir les instructions qui ont été exécutées sans que la page ne se recharge. C'est là que javascript et Axios interviennent pour faire les modifications suivantes :

Je veux sélectionner tous les éléments qui correspondent au bouton « like » et les modifier.

Pour se faire, j'utilise Axios pour récupérer la réponse de Symfony. Et quand Axios aura la réponse, je renvoie le résultat dans une fonction et je fais les modifications de cette réponse.

La réponse va renvoyer un tableau qui contiendra le nombre de « like » sur l'image.

Pour éviter que l'utilisateur incrémente les «like» en boucle, je récupère le nombre de « like » des utilisateurs et j’applique une condition dans le contrôleur d'image ainsi que le Twig.

5. La troisième difficulté rencontrée : Stripe

Pour faire le système d'abonnement j'ai utilisé l'API Stripe.

J'ai passé plus d'une semaine à comprendre le fonctionnement de Stripe et n'ayant toujours pas une maîtrise des évènements, la construction du développement s’en est vu retardée.

La documentation de Stripe est facile à comprendre une fois la récupération des données maîtrisée.

Le principe de l'application sur le plan du Contrôleur est simple.

Je définis un bouton qui va déclencher une session d'abonnement créée par Stripe pour l'utilisateur.

La session est déclenchée dans une route que j'ai nommée « create-checkout-session ».

Cette route va reprendre la création de l'abonnement pour l'utilisateur.

Tous les éléments de l'abonnement sont repris dans un tableau :

- Le prix de l'abonnement
- Le type d'abonnement
- La durée la création de l'abonnement
- Le statut
- Url_success
- Url_cancel

Stripe nous propose dans ce tableau de définir deux routes qui renverront l'utilisateur sur une page de succès ou annulation de l'abonnement.

Pour enregistrer l'ID de la session de l'utilisateur une fois abonné, j'ai créé un attribut dans l'entité User pour le récupérer.

6. La quatrième difficulté rencontrée

Quand le client revenait sur la page pour s'abonner une nouvelle fois, celle-ci ne s'affichait plus. Je me suis donc posé les questions suivantes : « A quel moment enregistrer l'ID du client? » et « Quand la supprimer ? ».

La récupération des données du client m'a aussi posé problème : « Comment récupérer l'abonnement du client pour accéder à ses données ? »

J'ai donc consulté la documentation de Stripe pour savoir de quelle façon m'y prendre. J'ai finalement compris que Stripe avait plusieurs fonctions pour retrouver l'abonnement, le modifier et le supprimer.

Le mécanisme est le même pour les factures, les prix etc.

Après plusieurs analyses et essais de mon code, j'ai finalement trouvé des solutions en créant des conditions permettant de savoir si l'abonnement est en date d'expiration , et si le statut de l'abonnement est valide ; la question la plus importante étant « Où placer mon code pour éviter des répétitions ? »

Et j'ai trouvé une solution en créant une méthode dans l'entité «User» pour vérifier le statut de l'abonnement.

Une autre technique que j'ai énormément utilisée a été la commande DD() pour analyser ce que Stripe me renvoyait à chaque fois que j'avais un tableau.

Je les décortiquais pour savoir où retrouver les données que je cherchais.

Ces analyses m'ont permis de progresser et de mieux comprendre la manière dont fonctionne Stripe.

6.1. Système d'authentification Github

Ce système a été également une première pour moi . J'en ai déduit qu'il y avait un rapport avec l'authentification et donc, encore une fois, j'ai fait plusieurs recherches pour confirmer mon raisonnement en tapant certains mots clés tels que « AP I»,« auth gihhub » , « register » et « vidéo ».

J’ai alors découvert KnpUOAuth2ClientBundle qui reprend tous les moyens de se connecter avec des réseaux sociaux ; le réseau social que j’ai sélectionné étant github.

J'ai installé le bundle et conclusion a été la suivante :

l'objectif était de trouver un moyen de me/se connecter sur l’application mais plutôt que de se connecter via un identifiant et un mot de passe, c'était d'essayer de trouver un moyen d'activer un bouton qui allait ensuite rediriger sur le réseau social en question et trouver un moyen d'accepter les permissions pour être redirigé sur l'application et se connecter automatiquement.

Selon la documentation que j’ai consultée, j'ai d'abord été sur github pour configurer les applications proposées . J'ai utilisé Aouth Apps qui permet de demander l'accès à certaines informations de l'utilisateur pour qu'il puisse se connecter.

J'ai donc enregistré une nouvelle application qui, pour ma part, sera « PicBak » et j’ai rempli les conditions demandées :

L'url de ma homepage

L'url pour les autorisations « callback URL»

Lorsque l'utilisateur va accepter les permissions, je vais récupérer son email depuis github je vais le rediriger vers mon site.

L'application va fournir également deux informations importantes qui sont le client ID et le client secret ; un moyen de sécurité pour faire fonctionner correctement mon application.

Pour éviter un maximum de risque, j'ai mis ces clés de côté dans un fichier .env.local durant la phase de «_dev» .

Ensuite, j'ai créé un système d'authentification avec la commande make:auth et j'ai créé un champ : « gitHugId » que j'ai ajouté dans l'entité User pour savoir si l'utilisateur s'est connecté via Github.

L'information importante que j'ai retirée était de mettre le champ « nul l» car les utilisateurs ne vont pas nécessairement se connecter via Github.

Ensuite, j'ai dû créer une nouvelle route pour se connecter via le réseau social qui prendra une fonction « connect() » suivit d'un paramètre « ClientRegistry ». Ce paramètre récupèrera toutes les informations de l'utilisateur.

Après une recherche dans la documentation du « Bundle Oauth2-github », j'ai dû créer une méthode en utilisant des « scopes » pour les récupérer.

Les « scopes » vont me permettre d'indiquer ce que je veux récupérer comme données chez l'utilisateur.

- «read:user» : les informations de base de l'utilisateur
- «user:email»: l'email de l'utilisateur

Je fais un premier test pour me connecter à mon application en utilisant le bouton de connexion Github.

Et je constate que j'ai bien une redirection vers Github qui me propose d'autoriser mes informations personnelles pour accéder à mon application.

En appuyant sur le bouton d'autorisation, une page "erreur" est apparue, me renvoyant dans l'url un «github=code» devant être changé contre un accès token pour accéder à l'utilisateur.

7. La cinquième difficulté rencontrée

Je savais qu'il fallait utiliser l'authentification et j'ai d'abord essayé de manière autodidacte mais je n'avais aucun résultat. J'ai donc consulté de nouveau la documentation du Bundle et j'ai réalisé qu'il fallait créer une deuxième authentification « GitHubAuthentificator » .

Dans le « GitHubAuthentificator », j'ai vérifié si la route que j'avais spécifiée pour la connexion Github était correcte et j'ai converti le code que j'ai reçu dans la page d'erreur en « token ». Ensuite, j'ai récupéré les informations via le « getUser($Credentials) ».

Je refais un second test pour me connecter et j'ai rencontré un deuxième problème m'expliquant qu'il y avait un conflit entre l' « Authentification » de base et « GitHubAuthentificator ».

La solution a été de définir l'un des deux en tant que principale authentification. Pour se faire, j'ai dû aller dans le « security.yaml » pour définir dans le « Guard: » un « entry_point: » et introduire la principale authentification.

7.1. l'interface administrateur

Pour l'administration, j'ai installé « Easycorp/easyadmin-bundle » afin d'avoir une interface sans devoir la créer.

Pour se faire j'ai exécuté deux commandes :

« make:admin:crud » pour la suppression, la modification et la création.

« make:admin:dashboard» pour l'interface.

J 'ai utilisé « Font Awesome » pour personnaliser des icons du menu reprenant ;

la liste des images, des catégories, des utilisateurs, des commentaires et les factures.

J'ai eu des soucis de conflit avec « VichUploader » ; un Bundle que j'ai utilisé pour afficher mes images mais j'ignorais à quel niveau cela se trouvait. Mais je ne savais pas à quel niveau c'était.

Au niveau de la visualisation des pages détails de l'image dans l'interface, il indiquait des champs « null » alors que les images étaient bien présentes au niveau de la page de modification et pour être certain, j'ai vérifié dans la base de données si elles étaient bien présentes.

J'ai vérifié dans la documentation de « Easycorp/easyadmin-bundle» (qui ma redirigé vers Symfony) et j'ai regardé s' il y avait une technique à employer pour les images; à part utiliser « ImageField::class » je ne voyais aucune autre solution.

J'ai donc eu deux hypothèses :

soit ce sont des problèmes de redirection dans l'interface admin ou des problèmes de compatibilité au niveau des «classFields» et des images. J'ai choisi de tester certaines class pour constater le résultat obtenu dans la page détail image . J'ai constaté que la « class ImageField:: » fonctionnait uniquement pour la page modifications et que la class que la « class TextareaField:: » affichait l'image dans la page détails. Mais si les deux class restaient active sur les 2 pages, j'obtenais une erreur ou l'image était «null». Pour remédier à ce problème, j'ai créé une variable Z qui est égale à un tableau pour mettre les class de l'image que je n'utilisais pas (la catégorie, la description et le titre), et j'ai créé une variables X qui est égale à TextareaField:: et Y pour ImageField:: J'avais 3 variables. j'ai établis une conditions en disant que si on était sur la page détails, on affichait Y avec [Z] sinon on affiche X avec [Z]

8. Sixième et dernière difficulté rencontrée

8.1. La production

Pour la production j'ai choisi d'héberger mon application sur « Heroku ».

un site internet qui permet de déployer des applications très facilement. Elle permet aussi de proposer des addons et des buildpacks pour les applications.

J'ai constaté qu'il y avait une documentation spécialisée pour Symfony. J'ai donc suivi la procédure pour mettre mon application en production.

Après avoir suivi toutes les étapes de la documentation, j'ai exécuté la commande via github pour héberger mon projet sur Heroku. En pleine installation, je constate une erreur au niveau de Yarn et Node.

Je suis donc retourné dans la documentation d'Heroku. Après plusieurs recherches, j'ai réalisé qu'Heroku avait des «Buildpacks» pour «node.js» et «php».

Redoutant des modifications indésirables, j'ai donc par précaution, fait une copie de mon projet. J'ai installé les «Buildpacks». Je réitère l'hébergement qui à présent fonctionne.

Je consulte le lien de mon application et je vois ma page d'accueil. Je commence à faire des tests au niveau des redirections mais rien ne marche. J'ai des erreurs 404.

Je retourne sur Heroku pour aller sur le tableau de bord de mon application pour savoir ce que je peux faire comme analyse. Et je constate que je peux vérifier les « View logs » pour voir les erreurs.

Et je réalise qu'il ne trouvait pas ma base de données ; cela m'a pris beaucoup de temps pour comprendre qu'il y avait un problème avec un Addons «Heroku Postgres» que j'ai utilisé pour héberger ma base de données.

Durant mon développement j'ai utilisé Mysql. J'ai donc comparé les deux systèmes et j'ai réalisé que les requêtes étaient 'presque' identiques.

J'ai compris que l'erreur venait de la Base de données. J'ai cherché un moyen de convertir ma base de données Mysql en Postgres et j'ai trouvé des solutions. Mais c'était trop complexe pour moi. Alors j'ai regardé la liste des addons qu'Heroku proposait. Je teste avec un autre addons JawsBD Mysql et je refais un hébergement sur Heroku.

Je vérifie les fonctionnalités de mon application et tout fonctionne mais je ne voyais aucune image sur mon site.

J'ai regardé si ce n'était pas dû aux urls. J'ai donc repris l'erreur de l'url que mon application renvoyait et j'ai fait des tests en la modifiant pour trouver le vrai chemin de mon domaine à mon image. Toutefois, ça ne résolvait pas mon problème pour autant.

Dès lors, j'ai effectué des recherches sur des forums afin de récolter des témoignages d'utilisateurs ayant rencontré un souci identique. Malgré un lien fait avec «VichUploader» et Heroku, je n'ai trouvé aucune réponse.

Néanmoins j'ai eu un indice avec cette recherche ; il y avait un autre Bundle qui était lié aux images : «LiipImagineBundle»: qui permet de filtrer la résolution de l'image. J'ai donc pu de cette manière approfondir ma recherche.

Et j'ai découvert qu'il fallait modifier la «GD» de LiipImagineBundle. J'ignorais de quel problème il s'agissait.

Du coup j'ai été voir la définition de «GD».

Pour résumer en PHP, il existe une bibliothèque qui se charge de manipuler dynamiquement les images. Il s'agit de la librairie «GD».

J'ai donc fait une recherche en tapant les mots clés suivant: «GD» , «Heroku».

Et j'ai découvert que Heroku utilisait "ext-gd". une autre extension de «GD» pour traiter les images.

Pour avoir cette extension j'ai du le mettre dans le composer.json pour mettre cette extension dans les exigences «"require": { "ext-gd": "*" }».

J'ai ensuite fait un « composer update » pour qu'il prenne en compte l'extension pour ensuite héberger mon application vers Heroku une nouvelle fois.

Au bout de toutes ces opérations, mon site devient fonctionnel.

9. Conclusion

Ma plus grande erreur aura été de ne pas être passé plus tôt en production pendant mon développement ; cela m'aurait évité certaines fautes et, surtout, cela m'aurait fait gagner du temps. Concernant Stripe, j'ai mis un certain temps à me familiariser avec cette application car je rencontrais des difficultés quant à la gestion et à la récupération des éléments. Toutefois, je n'ai eu besoin que de la documentation la concernant, sans chercher d'autres supports, pour concevoir l'abonnement de mon application, démontrant ainsi mon autonomie. La gestion des "like" m'a, elle, permis d'approfondir mes connaissances quant aux codes dynamiques.

Dès lors, toutes les difficultés rencontrées me permettent d'affirmer d'avoir appris de mes erreurs ; moyen le plus efficace pour continuer de progresser.

10. SOURCES

Vidéo:

Teacher du net

Grafikart

Lior Chamla

Documentation

Stripe:

https://stripe.com/docs/api/subscriptions

Image :

https://github.com/liip/LiipImagineBundle

https://github.com/dustin10/VichUploaderBundle

github authentification:

https://github.com/knpuniversity/oauth2-client-bundle

https://github.com/thephpleague/oauth2-github

https://docs.github.com/en/developers/apps/authorizing-oauth-apps#scopes

EasyAdmin:

https://github.com/EasyCorp/EasyAdminBundle

https://symfony.com/doc/current/bundles/EasyAdminBundle/crud.html

https://symfony.com/doc/current/bundles/EasyAdminBundle/fields.html#field-types

Pagination:

https://github.com/KnpLabs/KnpPaginatorBundle

Symfony:

https://symfony.com/book

Symfony 5: The Fast Track FR

Bootstrap Sass:

https://getbootstrap.com/docs/4.1/getting-started/introduction/

Heroku:

https://elements.heroku.com/addons

https://devcenter.heroku.com/articles/deploying-symfony4

https://stackoverflow.com/questions/39184760/how-to-install-gd-on-heroku

https://waytolearnx.com/2018/11/difference-entre-mysql-et-postgresql.html

https://elements.heroku.com/buildpacks