université de TOURS | Faculté des sciences & techniques

Dossier synthèse :
Programmation WEB.

Réalisation d'un site portfolio pour l'artiste Eugénie PICASSOUILLE.

Groupe 4
Inès FEUGIER
Maria Grace MBOMY MBALA
Sophie NUNG
Léo PERROUAULT
Laurie PETRIKOVA

Table de matière :

I. Projet

1. Contexte :

Eugénie Picassouille est une artiste spécialisée dans la réalisation d'illustrations, de logos et d'affiches sur commande. Elle partage déjà ses créations sur différents sites, tels qu'Instagram, Facebook et Twitter, elle possède donc déjà une présence en ligne.

Seulement, cette présence en ligne est limitée, et cela se remarque sur le faible nombre de vues sur ses publications et l'absence de nouveaux clients dans son carnet de commandes.
La plupart de ses réalisations actuelles sont des œuvres spontanées, qui n'ont pas été commandées, dont le nombre d'acheteurs ne lui permettent pas d'effectuer autant de bénéfices que des œuvres sur-mesure.
De plus, Eugénie ne possède pas de plateforme afin de gérer sa seconde activité, l'animation d'ateliers artistiques. Elle doit gérer les inscriptions elle-même par mail et refuser des participants manuellement comme les places sont limitées.

L'objectif de ce projet est de créer un site web incarnant le cœur de l'activité d'Eugénie Picassouille ainsi que son identité artistique.
Toutes ses activités doivent être rassemblées sur une seule plateforme, pour ainsi centraliser ses œuvres et actualités, gérer automatiquement les inscriptions aux ateliers et faciliter la prise de commande des clients.
La clientèle ciblée :
Avec son site, la clientèle visée par l'artiste Eugénie Picassouille est assez diverse et variée.
Comme elle réalise aussi bien des œuvres de manière spontanée que sur-mesure, ce qui fait qu'elle compte dans son carnet de commandes des professionnels, notamment de l'événementiel ou du secteur textile, ou bien des particuliers.

2. Cahier des charges :

Le site comprendra à la fois une partie publique accessible à tous et une partie administrateur, accessible uniquement à Eugénie grâce à un identifiant et un mot de passe.

La partie publique reflète son style et permettra aux potentiels clients d'accéder plus facilement à toutes les informations la concernant grâce à :

- Une présentation de l'artiste, soit son identité et son activité.

- Le portfolio, dossier numérique contenant des exemples de ses réalisations.
  Comme ce portfolio permet d'attirer les futurs acheteurs, les œuvres les plus récentes doivent être visibles en premier, de manière à ce qu'ils aient accès aux œuvres les plus représentatives du style graphique d'Eugénie. Pour faciliter les demandes lors des prises de commandes, les œuvres pourront aussi être triés par type (Logo, Illustration et Affiche), mais aussi avec des mots-clés (par exemple définissant le format, les couleurs utilisées ou le contenu de l'œuvre) ou par date, afin de faciliter la visualisation de l'archive des œuvres.
  Cliquer sur une des œuvres de son portfolio permettra d'accéder au détail de cette œuvre : image agrandie, titre, dimensions (en pixel) et commentaire de l'artiste.

- Les actualités de l'artiste, triées par date, les articles les plus récents se trouvant au sommet de la page. Un titre et une image sont associés à chaque actualité. La date doit aussi être mentionnée. Cliquer sur une actualité permet d'accéder au contenu détaillé de cet article. Les différents articles peuvent être partagés sur les réseaux sociaux sur lesquels sont présents l'artiste, soit Facebook, Twitter et Instagram.

- Les ateliers proposés par l'artiste ont un titre, une date de début et de fin et un nombre de places défini. Cliquer sur un atelier permet d'accéder au détail de celui-ci et à une description détaillée, qui contiendra les détails permettant d'accéder à cet atelier. Les visiteurs du site peuvent s'inscrire à un atelier s'il reste encore des places. Ils doivent fournir leurs informations (nom, prénom) ainsi que leur email et leur téléphone pour être contactés en cas de problèmes concernant l'atelier. Les participants peuvent aussi laisser un message lors de leur inscription afin de prévenir d'un éventuel désistement possible ou d'autres requêtes (besoin d'un aménagement suite à un handicap, etc.) destinés à l'artiste.

- Une page de contact, permettant aux potentiels acheteurs de pouvoir contacter l'artiste, afin de demander des informations sur les œuvres déjà présentes, par exemple pour en acheter des droits d'utilisation commerciale, ou afin de commander une œuvre originale et sur-mesure. Ils devront fournir leur nom, prénom, adresse mail à laquelle ils souhaitent être contactés ainsi qu'un sujet et leur message.
  La réponse de l'artiste et les éventuelles conversations suivant cette demande de contact se feront hors du site, sur un service de messagerie.

La partie administrative permettra à Eugénie Picassouille de pouvoir gérer les différentes fonctionnalités de son site afin de mieux gérer son activité. Elle contiendra :

- Un espace pour gérer son identité : modification de ses coordonnées, adresse, de sa biographie.

- Une page permettant de gérer ses œuvres mises en ligne : elles permettent l'ajout, la modification et la suppression des œuvres dans sa base de données.
  Lors de l'ajout d'une œuvre, on enregistre l'image, le titre, le type d'œuvre, une liste de tags permettant de trier l'œuvre par caractéristiques et les dimensions de cette œuvre.
  La modification d'une œuvre permet de modifier ces mêmes paramètres.

- Une page permettant de gérer ses actualités, qui permet d'ajouter, modifier et supprimer ces données.
  Lors de l'ajout d'une actualité, on enregistre une image, le titre, une liste de tags permettant de trier la news par caractéristiques et le contenu de cette actualité.
  La modification d'une actualité permet de modifier ces mêmes paramètres.

- Une page permettant de gérer ses ateliers, qui permet d'ajouter, modifier et supprimer ces ateliers.
  Lors de l'ajout d'un atelier, on enregistre le titre, une description détaillant l'atelier, une date de début, une durée et un nombre de places.
  La modification d'un atelier permet de modifier ces mêmes paramètres, mais de consulter ou supprimer des participants, afin d'éventuellement modifier le nombre de places.

- Une page permettant de consulter les demandes de contact. Bien que la suite de la conversation se déroule hors du site, on souhaite tout de même savoir quelles demandes ont eu une réponse ou non.

II. Environnement de développement

1. Présentation du framework Symfony et Bootstrap :

Symfony est un ensemble de composants PHP ainsi qu'un framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer le développement d'un site web.

Il a été principalement utilisé afin de créer ce projet. Doté d'une documentation complète, Symfony compte beaucoup d'adeptes à son utilisation. De plus, elle compte une grande communauté de développeurs qui n'hésite pas à ajouter plus d'indications ou de nouvelles fonctionnalités pour le framework. Symfony ne cesse de faire de nouvelles mises à jour de leur framework, la plus récente étant la version 6 sortie en novembre 2021. Pour ce projet, nous avons utilisé la version 5 de Symfony.
Symfony est compatible avec plusieurs systèmes d'exploitation (Windows, Linux, MacOS), un serveur web avec PHP 8 minimum est requis.
Symfony permet de développer des solutions en PHP en fournissant de multiples fonctionnalités afin de faciliter le développement d'un site web avec son architecture de MVC (Modèle Vue Contrôleur). Plusieurs outils sont mis à disposition pour les développeurs, simplifiant les documentations, Symfony permet aussi de déboguer et tester les projets. Le framework génère automatiquement plusieurs options, et gère de lui-même les configurations nécessaires.

En pair avec Symfony, Bootstrap peut être rapidement et aisément intégré dans les projets auxquels les développeurs travaillent. Pour la création du site web pour la cliente, l'utilisation de ce framework fut adoptée.
Bootstrap étant un framework CSS, il regroupe un bon nombre de librairies CSS et JavaScript, ayant en avantage un gain de temps considérable pour le développeur, une architecture propre pour le code, et contenant aussi beaucoup de fonctionnalités.
Mobile-first, il permet aussi de rendre un site web responsive, c'est-à-dire que le soit adapté à tous les écrans sans avoir la nécessité de tout coder un par un en CSS.

2. Description des différentes technologies utilisées :

Le framework Symfony est développé en PHP et utilise un moteur de template Twig pour gérer tout ce qui est affichage HTML. Symfony grâce à son code open source nous permet d'utiliser plusieurs types de librairies que l'on nomme bundle qui se comporte comme des extensions qui vont complémenter le framework.

Tout d'abord, nous avons installé les bundles Webpack et Bootstrap qui nous permettent une meilleure gestion de tout ce qui est visuel et conception graphique.

Webpack nous permet de transformer des modules avec dépendances en des fichiers statique que l'on nomme Asset, ainsi, c'est plus facile de mettre en place tout ce qui est CSS et image, car au lieu de transmettre directement le chemin du dossier de l'image ou du CSS, on transmet directement le nom.

Quant à Bootstrap, il s'agit d'un bundle qui simplifie la conception graphique en effet, il s'agit d' une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. On l'utilise pour simplifier différentes tâches, mais aussi pour apporter une touche de modernité et faire en sorte que notre site soit responsive.

Nous avons utilisé différents IDE pour le développement en fonction du choix de la personne.
IntelliJ pour Sophie NUNG et Laurie PETRIKOVA
PhpStorm pour Maria Grace MBOMY MBALA et Léo PERROUAULT
NetBeans pour Inès FEUGIER

Pour faire fonctionner notre site Web nous avons utilisé un serveur Local sur Windows Xampp 8 et sur Linux Lampp. Le serveur tourne en PHP 8 avec MySQL comme base de données puis Symfony et Composer.

Pour finir, nous avons utilisé un Gestionnaire de code source décentralisé Git par l'intermédiaire de Github qui permet de nous répartir les différentes tâches de travail, mais aussi de séparer le code ainsi nous garantissons une sécurité sur le développement en n'impactant pas sur une tâche qu'un tel est entrain de faire.

3. Procédure d'installation :

Pour installer Symfony, il est nécessaire d'avoir quelques pré-requis.
Comme dit précédemment, il faut être muni d'une plateforme permettant de coder en PHP (Wamp, XAMPP ou bien Lamp) ayant comme version PHP 8 au minimum. L'installation de Composer (qui sert à installer les packages de PHP) et de Node JS (qui sert à utiliser le JavaScript).

- Sous Windows, avec XAMPP et une fois que la dernière version de Composer (sur leur site officiel) est téléchargée, il suffit de lancer l'assistant d'installation de Composer et de suivre le processus d'installation.
- Sous Linux et MacOS, cela est un peu différent, il faut rentrer quelques lignes de commandes, les commandes à rentrer se trouve sur le site de Composer : https://getcomposer.org/download/

Pour mettre à jour le Composer (ou être sûr de bien installer tous ces composants), il suffit de taper dans un terminal les commandes suivantes :

- composer -install
- composer -update

Pour Node JS :

- Sous Windows, après que la version de Node JS pour notre propre système d'exploitation soit installée (sur leur site officiel : https://nodejs.org/fr/download/), une nouvelle fois, il faudra simplement suivre les étapes d'installations du programme.
- Pour le système d'exploitation Linux de type Debian, afin d'être certain d'avoir la dernière version de Node.js, il faut installer le dépôt Node Source avec la commande suivante :
  (Pour être sûr d'avoir la commande curl, il est mieux d'effectuer : sudo apt-get install curl; curl permettant d'accéder aux url).
  - curl -sL https://deb.nodesource.com/setup_??_.x | sudo -E bash-

  Remplacer les ?? par la version voulue du nodeSource
  Le dépôt est installé, il n'y a plus qu'à installer Node JS avec la commande :
  - sudo apt-get install nodejs

- Et sous MacOS, comme pour le système d'exploitation sous Windows, après l'installation de Node JS, il suffit de suivre la procédure d'installation.

Après c'est pré-requis implémenter, le framework Symfony se télécharge sur leur site officiel : https://symfony.com/download.
Grâce à la documentation complète fournie par le framework, toutes les étapes se trouvent sur leur site. Et voici le framework Symfony prêt à l'utilisation de création de projet avec quelques lignes de commandes dans un terminal.

Pour importer le projet du site web, il est donc nécessaire d'avoir Symfony et PHP 8. Pour importer le projet correctement, il faut le lien du GitHub du projet :
https://github.com/LPerrouault/Picassouille
Importer la base de données (qui se trouve dans le GitHub) dans votre localhost (MySQL, MariaDB, etc).
Dans un IDE permettant de coder du PHP (nous avons utilisé IntelliJ, PhpStorm, NetBeans), il faut cloner le projet avec Git.

Dans votre terminal ayant pour chemin le projet, il faut rentrer les commandes pour être sûr d'avoir la bonne version du composer, il faut taper les commandes suivantes :

- composer -i
- composer -u

Ensuite, il faut aller dans le fichier .env dans le projet et le modifier pour mettre les informations de la base de données du projet. Il faut décommenter la ligne en rapport avec mysql et la modifier comme ce qui suit (si ce n'est pas déjà fait quand le projet a été cloné) :
DATABASE_URL="mysql://root@127.0.0.1:3306/l3_projet_picasouille"
Pour compiler correctement le CSS et le JS, il est fortement recommandé de faire la commande suivante dans votre terminal :

- npm run dev

Avant de lancer son serveur symfony, il faut se placer sur la branche main si cela n'est pas déjà fait. Une fois toutes les étapes réalisées, il faut lancer le serveur avec la commande suivante toujours dans votre terminal :

- symfone serve

ou bien

- symfony serve:start -d

Puis aller sur un navigateur et suivre ce que dit le terminal (aller sur : 127.0.0.1:8000), pour aller sur le projet, il faut mettre bonne route, et pour afficher l'accueil par exemple, il suffit simplement de rajouter juste derrière : /accueil

III. Architecture et conception

1. Modèle de données :

Pour l'élaboration de notre base de données nous sommes partie de notre cahier des charges ainsi le site est un portfolio d'une artiste solo qui prend des commissions par devis par l'intermédiaire de contact.

Ses œuvres se décomposent en trois catégories : illustrations, affiches pour les évènements, logos.

Ses différentes œuvres sont toutes caractérisées dans une table "oeuvre" par :
un id, un titre, une taille (longueur et hauteur), un type d'œuvre, le nom du fichier, une description, et des tags. Pour ce qui est des types, ils seront séparés dans une autre table qui sera appelée dans œuvre par l'id du type.

Pour les mails reçus par le contact, on aura une table MailContact qui contiendra :
un id, le nom et prénom du client qui voudra contacter l'artiste, le mail, le sujet et le contenu du message.

Une table Article qui sera composée :
d'un id, d'un titre, du corps de l'article, de la date, puis id du tag de l'article.

La table Atelier quant à elle possède une table qui constituera :
d'un id, d'un titre, d'une description, d'une date de début puis d'un nombre de places.

Une table Inscription qui permet de s'inscrire à un atelier qui sera composée :
d'un id, du nom et prénom de la personne, de son email et téléphone, du message qui peut être null, puis l'id de l'atelier auquel le client voudra s'inscrire.

Une table Tag qui est composée d'un id et d'un libellé.

Puis deux table ListTagArticle et ListeTagOeuvre qui fait le lien entre
- Article et Tag pour les Articles
- Oeuvre et Tag pour les Oeuvres

Et pour finir une table Utilisateur qui nous servira à nous connecter à la partie Admin qui est composée d'un id, un login, du rôle, du mot de passe, de son nom et prénom.

a. On établit la forme normale :

Oeuvres (id, titre, largeur, hauteur, description, nomFichierImage, #idType)
Clé primaire: id
Clé étrangères: idType en référence à id de la table type

Type (id, libelle)
Clé primaire: id

Tag (id, libelle)
Clé primaire: id

ListeTagOeuvre (#idOeuvre, #idTag)
Clé primaire: idOeuvre, idTag
Clé étrangère: idOeuvre en référence à id de la table oeuvre
idTag en référence à id de la table tagl

ListeTagArticle (#idArticle, #idTag)
Clé primaire: idArticle, idTag
Clé étrangère: idArticle en référence à id de la table article
idTag en référence à id de la table tag

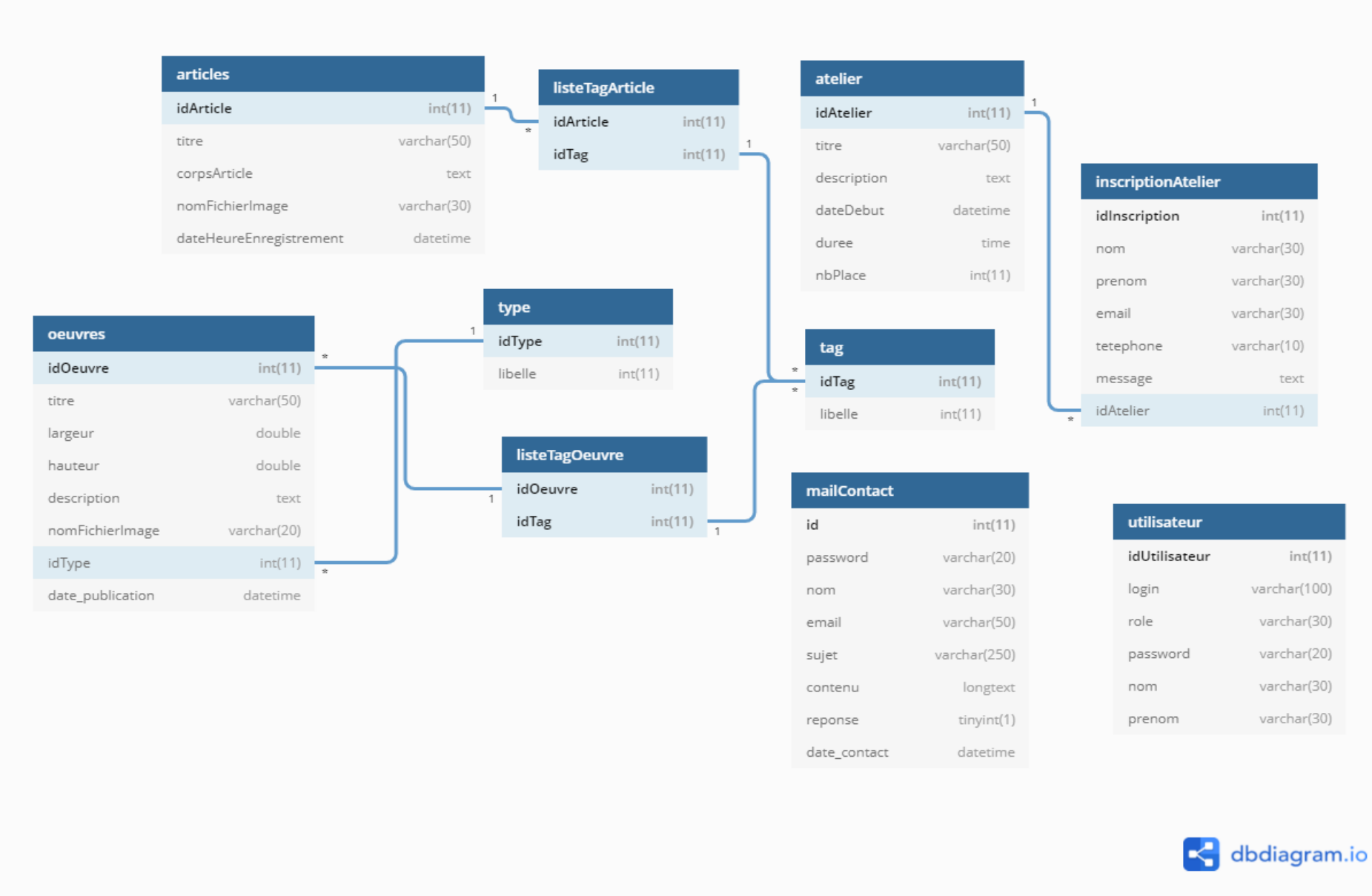
Atelier (id, titre, description, dateDebut, duree, nbPlace)
Clé primaire: id

Inscription (id, nom, prenom, email, telephone, message, #idAtelier)
Clé primaire : id
Clé étrangère : idAtelier

Articles (id, titre, corpsArticle, nomFichierImage, dateHeureEnregistrement)
Clé primaire: id

Utilisateur (login, nom, prenom, mdp, rôle)
Clé primaire: login

b. Ainsi nous avons pue concevoir le diagramme MCD suivant :


articles
idArticle int(11)
titre varchar(50)
corpsArticle text
nomFichierImage varchar(30)
dateHeureEnregistrement datetime
listeTagArticle
idArticle int(11)
idTag int(11)
atelier
idAtelier int(11)
titre varchar(50)
description text
dateDebut datetime
duree time
nbPlace int(11)
inscriptionAtelier
idInscription int(11)
nom varchar(30)
prenom varchar(30)
email varchar(30)
tetephone varchar(10)
message text
idAtelier int(11)
type
idType int(11)
libelle int(11)
oeuvres
idOeuvre int(11)
titre varchar(50)
largeur double
hauteur double
description text
nomFichierImage varchar(20)
idType int(11)
date_publication datetime
tag
idTag int(11)
libelle int(11)
listeTagOeuvre
idOeuvre int(11)
idTag int(11)
mailContact
id int(11)
password varchar(20)
nom varchar(30)
email varchar(50)
sujet varchar(250)
contenu longtext
reponse tinyint(1)
date_contact datetime
utilisateur
idUtilisateur int(11)
login varchar(100)
role varchar(30)
password varchar(20)
nom varchar(30)
prenom varchar(30)
dbdiagram.io

2. Découpage des controllers :

Pour ce qui en est de la répartition des controllers, chacune des pages de notre site (Accueil, Œuvres, News, Ateliers, Contact) a en réalité deux controllers : un pour la partie admin, un pour la partie publique. Ceux-ci ont été créés à l'aide de la commande make:controller de Symfony, ce qui les a automatiquement placés dans le dossier Controller dans le source de notre projet.
Les controllers de la partie publique sont nommés de la même façon que la barre de navigation, ceux de la partie admin sont nommés de la même façon, suivis par 'Admin'. Les noms des controllers sont en capitales.
Les controllers ont chacun au moins une route distincte : une url qui permet de les faire fonctionner et permet l'affichage de leur template.
Par la suite, les différentes pages du site ont été implémentées en HTML dans les templates.

3. Présentation du templating :

Grâce à la commande make:controller de Symfony, les templates correspondantes sont automatiquement créées et placées dans un dossier portant le même nom que le controller, tous les dossier sont rangés dans le dossier qui se nomme template dans le projet.

Ces templates ainsi créées implémentent le fichier base.html.twig où se situent les premières lignes de code nécessaires à toute page HTML ("!Doctype" etc.), ainsi que les éléments que l'on retrouvera dans toutes les pages : le header, le footer et le bouton ramenant au sommet de la page. De plus, on retrouvera dans le "style" du fichier base les liens permettant d'accéder aux librairies Bootstrap et, tout en bas du fichier base, les liens pour le Javascript, afin que toutes nos templates y aient accès et puissent les implémenter.

Dans chaque dossier de template qui renvoie au controller, nous avons un index.html.twig qui correspond tout simplement au body qui se rajoute sur le fichier, base.html.twig. Dans ce dossier, nous pouvons aussi créer d'autre ".html.twig" qui peuvent exécuter différente chose en fonction du besoin, par exemple pour la page Actualité grâce au dossier template news nous avons deux extension "_card.html .twig et _filter.html.twig " qui permet l'affichage des différentes actualité par une boucle et le filtre qui permet de sélectionner le tag de l'article, vont s'ajouter à index.html.twig. En effectuant cette méthode, cela permet de mieux distinguer le code, mais aussi simplifié de nombreuses fonctions comme les boucles.

4. Gestion de la sécurité, des rôles utilisateurs et partie admin :

Afin que la cliente Eugénie P. puisse gérer son site web, il est nécessaire d'avoir une partie administrateur. De ce fait, la partie administrateur n'est accessible que par Eugénie P. et les administrateurs qu'elle aura choisie.

Le site en mode admin est donc réservé aux admins et non aux clients. Pour y accéder il faut posséder un compte et se connecter avec l'URL de connexion. Même si les clients écrivent l'URL d'une certaine page dédiée aux administrateurs, l'accès ne sera pas possible. Il y a donc un système de connexion afin de sécuriser cette partie privée.

Dans la partie publique, il n'y a aucun système d'inscription ou de connexion, les clients potentiels de l'artiste n'ont pas besoin d'avoir un compte pour passer commande ou pour contacter Eugénie P. Dans la page Contact, les clients peuvent envoyer un message à l'artiste directement sans qu'ils n'aient besoin de s'inscrire. C'est pour cela qu'il n'y a pas d'utilisateurs clients avec un compte sur le site.

La gestion de la messagerie pour Eugénie P. se fait ainsi dans la partie admin, elle peut voir les messages envoyés par les clients et leur répondre étant donnée qu'elle possédera leur contact après un message de leur part. Mettre ses différents mails sous différents statuts, s'ils ont été répondu ou non.

Ainsi, sur les pages admin, l'artiste Eugénie P. peut modifier tout ce qu'elle souhaite, ajouter ou bien supprimer des articles ou des œuvres. Et comme la partie client, elle peut voir les différentes œuvres et les différents articles en fonction des tags ou du type ainsi elle pourra mieux s'organiser sur les relations avec les clients.

5. Solution d'upload des fichiers :

Pour l'upload des fichiers nous avons utilisé le bundle communautaire vichuplaoderbundle qui comporte une documentation riche et facile à comprendre.
Nous avons utilisé ce bundle dans la partie admin pour la page Œuvres et Actualités (News) pour la création d'article ou l'ajout d'une œuvre, mais aussi pour effectuer leur modification.

Ici ne sera traitée que la partie Actualité le système d'upload est le même, il s'agit seulement du formulaire de création ou de modification qui change ainsi que le chemin du fichier.

a. Partie Formulaire

Tout d'abord, nous avons créé un formulaire "ArticleFormType" qui comporte les champs suivants :

- titre de la forme TextType
- listTag, de type choicetype d'entity
- nomFichier, de type FileType aev constraint Image
- corpArticle de type CKEditorType
- puis un bouton submit et reset

Lorsque que l'on rentre les informations dans le formulaire, c'est le champ nom Fichier Image qui récupérera l'image par cette partie de code dans le formulaire

```
->add('nomFichierImage', FileType::class,[
    'label' => false,
    'mapped' => false,
    'attr'=>['class'=>'form-control-file'],
    'required' => true,
    'constraints' => [
        new Image()
    ]
])
```

En effet la contrainte Image fait en sorte que l'on ne puisse pas importer des fichiers autres que des images ainsi, on garantit une sécurité sur le type de fichier que l'on importe.

b. Partie Controller

Dans notre controller "NewsAdminController" nous avons deux méthodes qui permettent l'upload de fichier :
La première est "addNews" qui permet la création d'un article.
La deuxième est "updateNews" qui permet de modifier un article.

Dans chacune des méthodes on attend que le formulaire est submit sinon on ne peut tout simplement pas effectuer l'upload, une fois que ceci est fait on a deux possibilités :

La première est s'il s'agit d'un ajout d'un article alors on récupère le fichier du formulaire par une variable de type UploadedFile.

```php
/** @var UploadedFile $brochureFile */
$brochureFile = $form->get('nomFichierImage')->getData();
```

Cette variable est obligatoire, car il s'agit du fichier en lui-même et il n'est pas nécessaire de vérifier qu'il existe ainsi, on fait la condition suivante pour vérifier s'il existe :

```php
if ($form->isSubmitted() && $form->isValid()){
    /** @var UploadedFile $brochureFile */
    $brochureFile = $form->get('nomFichierImage')->getData();
   // Cette condition est nécessaire parce que le champ "brochure" n'est pas obligatoire.
   // le fichier  doit donc être traité uniquement lorsqu'un fichier est upload.
    if ($brochureFile) {
        $originalFilename = pathinfo($brochureFile->getClientOriginalName(), PATHINFO_FILENAME);
        // this is needed to safely include the file name as part of the URL
        $safeFilename = $slugger->slug($originalFilename);
        $newFilename = $safeFilename .'.' . $brochureFile->guessExtension();

        $path = '/public/image/article/'.$newFilename;

        //verification si le fichier existe false on upload, true on fait rien
        if (!file_exists($path)){
            try {
                $brochureFile->move(
                    $this->getParameter('brochures_directory'),
                    $newFilename
                );
            } catch (FileException $e) { $e = null; }
        }
```

Tout d'abord, on enregistre le chemin système du fichier dans une variable, cette variable sera ensuite transformée sous forme de string par le fonction slug et enregistrée dans une autre variable "safeFilename" qui nous permet de sauvegarder de manière temporaire et sécurisé l'URL du fichier et notamment son nom propre.

Ensuite, nous créons une autre variable "newFilename" qui nous servira à enregistrer le nouveau nom du fichier sur notre serveur.
Cette variable est composée de plusieurs concaténations.

- la variable "safeFilename" qui possède l'url et nom du fichier
- un string '.' pour faire la séparation
- $brochureFile->guessExtension() qui renvoie l'extension du fichier soit (png,jpeg, etc.)

Après avoir effectué toutes ces étapes, nous pouvons upload votre fichier, pour cela, nous avons tout d’abord créer une condition qui vérifie si le fichier existe déjà par son nom dans le dossier d'importation, s'il existe on ne l'emportera pas sinon nous réutilisons notre variable de type UploadedFile $brochureFile que nous associons avec la fonction move qui permet de déplacer le fichier dans un dossier défini avec le nom que l’on lui a défini, dans notre cas nous avons uploadé dans le dossier image de publique.

6. Configuration des librairies communautaires :

a. Configuration de l'uploader (vichuplaoderbundle)

Sur Symfony on ne peut pas directement upload des image par le chemin de destination direct du dossier, on doit configurer et créer un paramètre dans config/service.yaml pour que le controller trouve le chemin.

Dans l'exemple de 5. pour article et œuvre, ont doit configurer le paramètre sous cette forme:

```
parameters:
    brochures_directory: '%kernel.project_dir%/public/image/article'
    oeuvre_directory: '%kernel.project_dir%/public/image/oeuvre'
```

Le paramètre que nous avons configuré nous renvoie vers le dossier asset de Webpack des images pour que celle-ci soit ensuite mise en avant sur la partie client.

Quand on exécutera la fonction move, notre fichier est uploadé dans le dossier défini par notre paramètre, en cas de problème nous avons mis une exception pour avertir l'utilisateur.

b. Configuration de Webpack et Bootstrap

Nous avons dû configurer plusieurs choses pour que Webpack et Bootstrap fonctionnent mutuellement.

Tout d'abord, nous avons installé Webpack Encore, avec les commandes suivante :

- composer require symfony/webpack-encore-bundle

Si l'utilisateur utilise le package Yarn

- yarn install

Et s'il utilise le package npm

- npm install

Puis dans un second temps, il était nécessaire de renommer le fichier /assets/styles/app.css en app.scss, ensuite modifier dans le fichier /assets/app.js : import './styles/app.scss';
Dans le fichier webpack.config.js, il faut décommenter : .enableSassLoader().
Alors, afin d'utiliser Sass, il faut l'installer avec la ligne suivante à rentrer dans un terminal :

- npm install sass-loader@^12.0.0 sass --save-dev
- npm run watch

Dans le fichier /templates/base.html.twig, il faut rajouter les "block" qui suivent :

{% block stylesheets %}{{ encore_entry_link_tags('app') }}{% endblock %}
{% block javascripts %}{{ encore_entry_script_tags('app') }}{% endblock %}

Une fois Webpack installé et configuré, nous pouvons dorénavant installer Bootstrap :

- npm install bootstrap --save-dev

Bootstrap est maintenant intégré dans Symfony, nous pouvons maintenant l'importer depuis n'importe quel Sass ou JavaScript. Par exemple, en modifiant le fichier /assets/styles/global.scss : import '~bootstrap/scss/bootstrap'; (ici pour le Sass).

7. Structure des fichiers de style :

Le projet contient deux sources différentes de fichiers de style. Ces deux sources sont incluses dans le fichier base.html.twig, qui contient la base des pages web du projet.

La première source est le CSS de Bootstrap, qui est ajouté au projet grâce à un lien vers son CDN. La version 4.3.1 est utilisée.
La seconde source est le CSS propre au site, ajouté au projet grâce à Webpack Encore. Dans le fichier base.html.twig, notre source de CSS doit être placée après celle de Bootstrap de manière à ce que nos modifications graphiques ne soient pas écrasées.

Le CSS propre au projet se trouve dans le fichier /assets/styles/app.scss
Les codes couleurs HTML du projet sont commentés en haut du fichier.
Toutes les classes créées sont entièrement écrites en minuscules, doivent être signalées avec le préfixe ‘pi’ (qui représente Picassouille) et les espaces sont signalés par un tiret ‘-’.

Le fichier est découpé en plusieurs parties, indiquées par des commentaires entourés de ‘//’:

- Les éléments en commun: c’est-à-dire le code permettant de mettre en place la couleur de fond et du texte de la page, le header, le footer, les éléments HTML h1 à h4 et br et le bouton de retour vers le haut de la page. Il permet aussi de positionner le footer et le bouton retour vers le haut de la page en bas de la page.

- Les classes communes à plusieurs pages, dont le nom respecte les règles de nommages citées ci-dessus. Elles permettent notamment l’ajout de boutons et de différents types de textes. Certains éléments de Bootstrap (par exemple .card) peuvent aussi être modifiés dans cette partie s’ils sont utilisés dans plusieurs pages.

- Les classes spécifiques à certaines pages, chacunes séparées par des commentaires signalant la page à laquelle les règles CSS correspondent.

8. Structure des fichiers JS :

Tout comme les fichiers de style, le projet contient plusieurs sources de fichiers JS. Ces sources sont incluses à la fin de la balise body du fichier base.html.twig.

Les trois premières sources de JavaScript sont les fichiers JS nécessaires à Bootstrap: JQuery (version 3.3.1), Popper.js (version 1.14.7) et finalement le JavaScript de Bootstrap (version 4.3.1).

La quatrième et dernière source est le JavaScript propre au site, ajouté au projet grâce à Webpack Encore. Tout comme le fichier de style du site, la source doit être placée après celles nécessaires au fonctionnement de Bootstrap.

Le JavaScript propre au projet se trouve dans le fichier /assets/app.js
Les fonctions créées sont entièrement écrites en minuscules, doivent être signalées avec le préfixe ‘pi’ (qui représente Picassouille) et les espaces sont signalés par un tiret ‘_’.

Le fichier n’a pas de structure particulière, il contient les fonctions permettant l’affichage et le fonction du bouton retour vers le haut de la page et le fonctionnement des boutons associés aux types d’une œuvre dans la liste des œuvres.