Dossier de synthèse

Année 2022

Pour le titre professionnel de Développeur Web et Web Mobile

WORLD TRAVEL PHOTO

WALLIS
Maximillian

Elan FORMATION

Table des matières

Préambule

Remerciements :

Je tiens avant tout à remercier les formateurs, Stéphane SMAIL, Mickael MURMANN, Cindy CAHEN, Philippe BOUGET ainsi que Céline HUGONNOT qui ont su me procurer un enseignement de qualité tout au long de la formation.

Je tiens également à remercier mes camarades de classe avec qui j'ai passé de très bons moments au cours de la formation. Nous nous sommes régulièrement entraidés, c'était un véritable plaisir de partager cette aventure auprès de vous tous.

J'aimerais encore remercier toute l'équipe de l'Eurométropole Strasbourg où j'ai réalisé mon stage et où j'ai pu acquérir de nouvelles connaissances.

1 Introduction :

Je m'appelle Maximillian WALLIS, actuellement âgé de 29 ans, je suis une formation avec ELAN formation afin de devenir développeur.

Je me suis intéressé à la programmation dès mon plus jeune âge. Mon grand-père Jean SIEVERT était ingénieur informaticien à l'hôpital de Mulhouse et de ce fait, j'ai baigné dans un environnement ouvert sur l'informatique pendant de longues années.

Je suis passé par différentes études avant de réellement penser au développement.
Après un baccalauréat scientifique avec une spécialité en physique, j'ai été en formation PPL (private pilote licence), puis inscrit à Kehl, pour devenir kinésithérapeute.
Pour intégrer l'école de kinésithérapie à Kehl, j'ai dû apprendre l'allemand afin d'atteindre le niveau C1. J'ai été accepté sur dossier et passé brièvement la Schnuppertag.

Finalement, je me suis rendu compte que le métier de kinésithérapeute n'était pas fait pour moi.

C'est en passant plusieurs tests d'orientation chez REAGIR à Mulhouse que j'ai découvert que le métier de développeur m'attirait.
Je me suis ainsi inscrit à une préparation en école de codage à ELAN Didenheim.

Durant cette préparation, on m'a expliqué en détail les différentes formations et décrit le métier.
J'ai aussi pu commencer à apprendre et comprendre le langage informatique. J'ai réalisé plusieurs projets dont un pendu en PHP.

Après cette préparation plusieurs options s'offraient à moi.
J'ai été accepté à l'école 42 à Mulhouse après avoir passé les tests psychotechniques (17) et de mémoires (15).
J'ai aussi assisté à une journée Portes ouvertes à Epitech mais un bac+2 en informatique était demandé pour intégrer la formation qui m'intéressait.

Je pouvais aussi continuer avec ELAN à Colmar pour un titre RNCP 5. Option que j'ai choisie étant donné que la préparation de ELAN Didenheim m'avait beaucoup plu.
De plus, plusieurs des formateurs qui m'avaient donné envie de continuer ce métier étaient présents à Colmar.

2 Résumé du projet :

Nous entrons dans une nouvelle ère où la photo prend une grande ampleur sur les réseaux sociaux. La CNIL a mené une enquête et quelques chiffres y sont ressortis montrant l'importance de l'image sur les réseaux sociaux :

- « En moyenne, 300 millions de photos sont partagées chaque jour sur les réseaux sociaux ».
- « Un français sur deux publie des photos sur Internet, 86 % d'entre eux ont entre 18 et 24 ans ».
- « Plus d'un internaute sur deux prend une photo d'abord dans le but de la publier ».

J'ai ainsi décidé de réaliser un site de partage de photos avec pour thème le voyage.

Le développement comprend les fonctionnalités principales suivantes :

- Un utilisateur doit pouvoir créer un compte, se connecter et se déconnecter.
- Un utilisateur doit pouvoir modifier son profil ainsi que l'image de son profil.
- Un utilisateur doit pouvoir ajouter une photo, la supprimer et modifier sa description.
- Un administrateur doit pouvoir bannir les utilisateurs, supprimer une photo, supprimer/modifier la description d'une photo, supprimer/modifier un commentaire.
- Un utilisateur doit pouvoir voir les photos selon les villes dans lesquelles elles ont été prises.
- Un utilisateur doit pouvoir ajouter, modifier et supprimer un commentaire sur une photo.
- Un utilisateur doit pouvoir « aimer » une photo et supprimer son « j'aime ».
- Un utilisateur doit pouvoir « suivre » le contenu d'un autre utilisateur et « arrêter de suivre » le contenu d'un autre utilisateur.

Les langages de programmation que j'ai choisis pour ce projet sont PHP7, HTML5, CSS3, SQL ainsi que Javascript.
Aucun framework n'a été utilisé.

3 Compétences couvertes par le projet

3.1 Compétences Front-end travaillées

- Maquetter une application : création d'un wireframe, d'un prototype et d'une maquette via le logiciel figma.
- Élaborer la représentation des données d'un système d'information (modèle conceptuel de données et modèle logique de données via le logiciel looping).
- Développer une interface utilisateur web statique : réalisation de plusieurs pages à l'aide de HTML5 et CSS3.
- Développer une interface utilisateur web dynamique : intégration de scripts évènementiels avec un langage de script client (JavaScript).

3.2 Compétences Back-end travaillées

- Respecter les normes de sécurité (faille XSS, faille CSRF, faille SQLi, attaque par force brute et par dictionnaire, hacher les mots de passe).
- Création d'une base de données (à l'aide du logiciel heidiSQL).
- Maitrise des quatre fonctions de base du stockage persistant : de lecture, de création, de modification et de suppression (CRUD).
- Manipulation des données à l'aide de PHP et du langage SQL.
- Utilisation de la programmation orientée objet ainsi que du design pattern MVC.

3.3 Compétences transversales travaillées

- Respecter le RGPD (Minifier les données, avoir le consentement de la personne, avertir de la finalité du traitement des données, gestion du Droit à l'oubli, du Droit à la rectification ainsi que du Droit d'accès aux données personnelles).
- Conception d'interface utilisateur (UX et UI, signifiant respectivement User Experience (expérience utilisateur) et User Interface (interface utilisateur.
- Utilisation de l'anglais.
- Capacité à rechercher l'information souhaitée sur Internet.

4 Organisation du projet

4.1 Cahier des charges

4.1.1 Respect du RGPD

- Minifier les données (on demande juste le nécessaire. On ne stocke jamais ! même au cas où).
  L'utilisateur doit renseigner ses informations personnelles : e-mail, mot de passe, pseudonyme et une image de profil s'il le souhaite (une image par défaut s'affichera dans le cas contraire).
- Le consentement de la personne : acceptez-vous les conditions générales d'utilisations ? (La case à cocher ne doit pas être pré cochée).
- Avertir de la finalité du traitement des données (dans les conditions générales d'utilisation).
- Droit à l'oubli : Bouton suppression du profil (On ne garde rien).
- Droit à la rectification : On pourra avoir accès à son profil, modifier/supprimer son profil, modifier son mot de passe, réinitialiser son mot de passe.
- Droit d'accès aux données personnelles : Onglet contact qui permettra de demander ses données personnelles à l'administrateur.
- Il sera possible de se connecter/déconnecter.

4.1.2 L'enregistrement et la création d'un profil

- Enregistrer le profil utilisateur en base de données.
- Lui permettre de se connecter.
- Mettre l'utilisateur en session lorsqu'il se connecte.
- Lui proposer de mémoriser son adresse-mail via des cookies.
- Une fois connecté, proposer à l'utilisateur d'effacer la totalité des cookies liés au site.
- Permettre à l'utilisateur de poster des commentaires et des photos, les modifier ou encore les supprimer.
- L'utilisateur doit pouvoir suivre d'autres utilisateurs afin d'accéder facilement à leur profil.
- Distinguer l'affichage à l'écran entre une personne connectée et non connectée.
- Créer le rôle administrateur, permettre à l'administrateur de modifier les rôles des utilisateurs. Lui permettre de supprimer les messages et photos des autres utilisateurs.

- L'administrateur doit pouvoir bannir un utilisateur. Lors d'un ban, l'utilisateur sanctionné doit impérativement être directement banni. Il ne faudra pas attendre que l'utilisateur modifie les valeurs de sa session mais le faire directement en base de données (l'utilisateur sanctionné ne devra plus pouvoir poster/modifier de photo ni de commentaires).

4.1.3 Sécurité

- Vérification, hachage du mot de passe, messages d'erreurs/succès.
- Une regex pour le mot de passe sur le formulaire d'inscription.
- Un recaptcha pour les formulaires de connexion et d'inscription.
- Assainir les variables pour la faille XSS.
- Préparer puis exécuter les requêtes SQL pour la faille SQL injection.
- Comparer des token pour la faille CSRF.

4.1.4 Création du reste du site

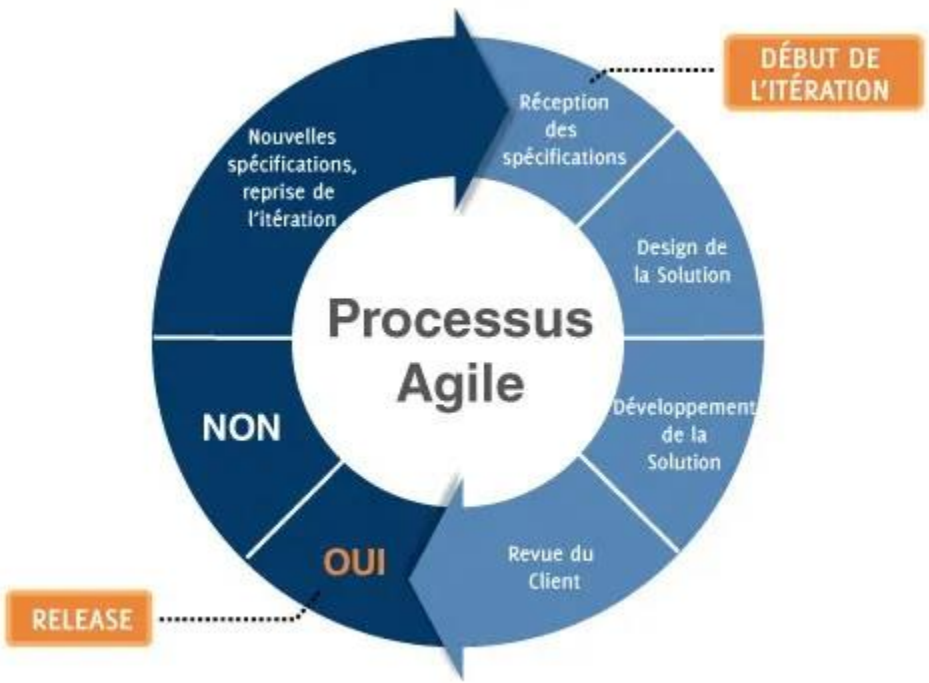
- Les photos des utilisateurs doivent automatiquement adapter leurs dimensions pour un bon affichage sur le navigateur.
- Le contenu d'un utilisateur doit être gardé si celui-ci supprime son profil (image et commentaires).
- Lorsque l'utilisateur supprime une photo, il faut la supprimer de notre dossier et ne rien garder.
- On pensera à mettre le sitemap, le plan du site dans le footer.
- La couleur du site pourra être modifiée. Deux choix seront proposés ainsi qu'une troisième couleur pour les dyslexiques (on choisira la couleur de base ajustée pour les dyslexiques).
- Un logo sera créé pour le site.
- Un footer adaptatif sera créé. Celui-ci sera en bas de page lorsqu'il aura assez de place (comme si l'élément était en bottom : 0) et dans les autres cas il se mettra automatiquement en bas de page. Il ne devra jamais passer devant ou derrière un élément de la page.
- Une barre de navigation adaptative sera créée.
- Le SEO sera respecté.
- Penser aux malvoyants (aria-current, aria-label, aria-expanded etc…).
- Le choix des couleurs sera en UI Design (ici bleu, blanc, noir, gris et éventuellement orange).

– Une recherche filtrée sera créée afin de trouver une ville ou un utilisateur plus rapidement.

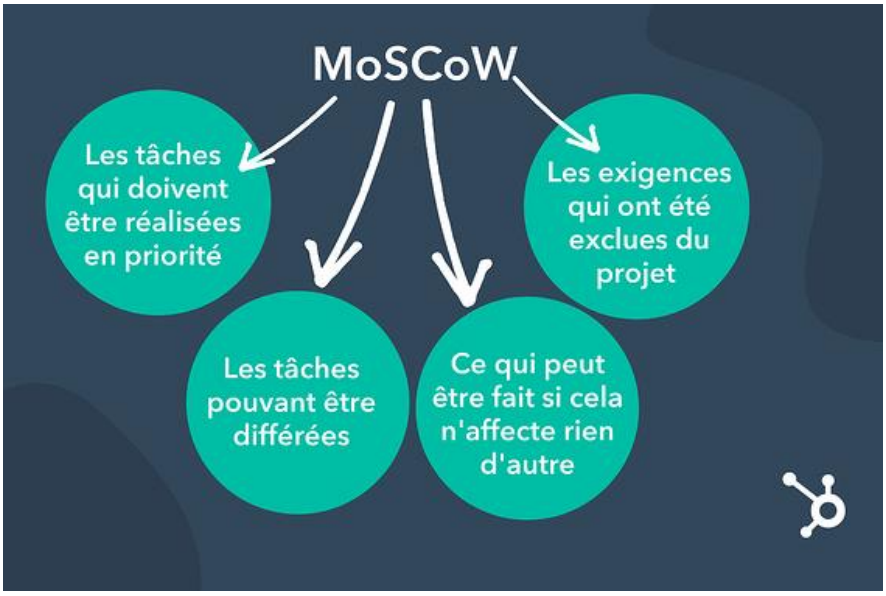
4.2 Méthodologie appliquée

J'ai décidé d'appliquer **la méthodologie agile**, plus particulièrement **la méthode MoSCoW** afin de mieux organiser mon projet.

La méthode agile se base sur **la rapidité de développement** de nouveaux logiciels ou applications par un ou une équipe de développeurs.
Plutôt que de déployer des mises à jour importantes et modifier plusieurs extraits de code en même temps, le développeur **priorisera chaque tâche de la plus importante à la moins essentielle**, de manière à résoudre les problèmes **progressivement**.



La méthode MoSCoW a été inventée par Dai CLEGG. Elle a été utilisée pour la première fois dans le cadre du projet Agile Dynamic Systems Development Method (DSDM).



MoSCoW est l'acronyme de :

M - Must have this : (« Doit avoir ceci » en français).
Ceci désigne les tâches qui doivent être réalisées en premier, les points les plus essentiels du site web ou de l'application.

S - Should have this if at all possible : (« Devrait l'avoir si c'est possible » en français).
Les tâches doivent être traitées dès que possible.

C - Could have this if it does not affect anything else : (« Pourrait l'avoir si cela n'affecte rien d'autre » en français).
Ces tâches sont bonnes à avoir, elles représentent donc un plus pour un projet qui s'est bien déroulé. À l'inverse, elles ont plutôt tendance à faire s'enliser un projet qui accuse déjà de nombreux retards.

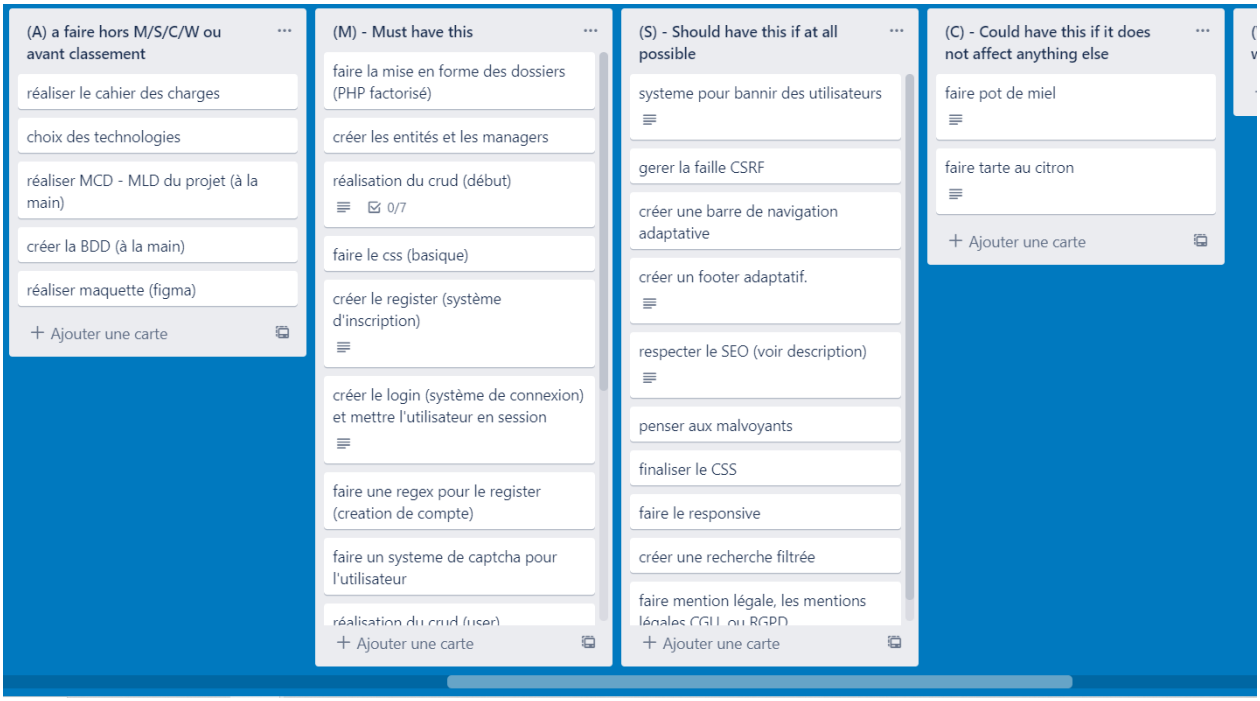
W - Won't have this time but would like in the future : (« Je ne l'aurai pas cette fois-ci, mais j'aimerais l'avoir à l'avenir » en français).
Ces tâches sont exclues du projet, cependant elles pourraient servir à une future mise à jour ou une future application.

Les « o » de la méthode MoSCoW servent à rendre l'acronyme prononçable et faciliter sa mémorisation.

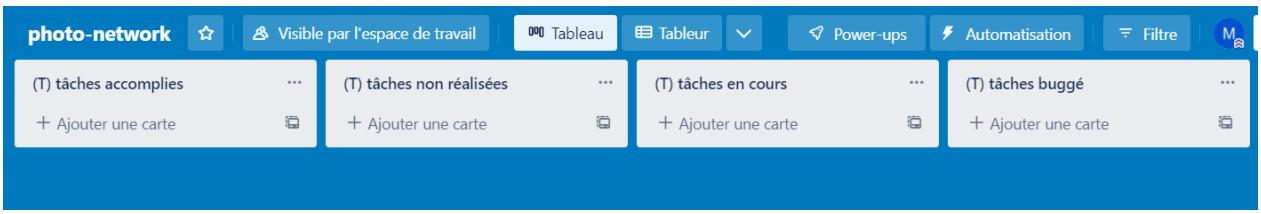
4.3 Gestion du projet

Afin de mieux gérer mon projet j'ai décidé d'utiliser trello, c'est un outil de gestion de projet en ligne.

Je me suis inspiré de la méthode agile et plus particulièrement de la méthode MoSCoW comme en témoigne l'organisation de mon trello.



Une fois une tâche réalisée, il nous suffira de la ranger dans l'une des quatre listes suivantes : (suivant le travail réalisé)



4.4 Référencement

Il est nécessaire d'avoir un bon référencement sur les moteurs de recherche si l'on souhaite avoir un maximum de visiteurs.
Une personne s'arrête en général à la page 1 de google lors d'une recherche. C'est pourquoi il est important de s'intéresser au référencement.

Le SEO (such engin optimisation) est ce que l'on appelle le référencement naturel, c'est-à-dire toutes les techniques qui visent à améliorer le positionnement d'un site internet dans un moteur de recherche tel que Google ou Bing.
Le SEO porte son action sur trois piliers : la technique, le contenu et la popularité. Il est nécessaire d'agir sur ces trois piliers afin d'améliorer la visibilité d'un site sur internet.

Le SEA (such engin advertising) concerne principalement la publicité diffusée sur un moteur de recherche. C'est d'une certaine manière un référencement payant.

SMO (Social media optimization) comprend toutes les activités visant à améliorer la visibilité au travers des médias sociaux.

Les 3 réunis donnent le SEM (such engin marketing).
Pour un bon référencement les 3 techniques sont complémentaires.

4.4.1 Histoire

La première apparition du SEO date de 1945 alors que google n'apparait qu'en 1998, le but recherché du SEO était de réunir des informations.
Le SEO voit sa deuxième évolution arriver en 1990 et devient le premier moteur de recherche archi (à l'époque pour chercher des données, il fallait le faire manuellement).
En 1993 viennent les premiers classements d'informations sur le net et les premiers hackeurs apparaissent.

Google a su mettre en place des algorithmes beaucoup plus efficaces pour juger la pertinence du contenu.
Google commence à mettre en place des pénalités pour les blackhat en 1998.
En 2002, on commence à mesurer l'audience des pages web. C'est Google qui a mis cela en place.

Évidemment, l'entreprise n'est pas dominante partout, on pourrait citer Baidu en Chine ou encore Yandex en Russie.
En 2005 google a perfectionné ses algorithmes et a commencé à prendre en compte la recherche des internautes et la géolocalisation
En 2010 vient l'entrée en matière des réseaux sociaux, à ce moment ce n'est pas la quantité mais la qualité qui fait le référencement (par exemple les avis, la popularité sur les réseaux sociaux, qui parle de vous ? Est-ce une entité avec une notoriété ?)

Les pénalités s'accentuent fortement à partir de 2011 avec la mise en application de PANDA (pénalité concernant les sites avec des mauvais contenu, des mises à jour non effectuées ou des contenus dupliqués) puis en 2012 avec la mise en application de PENGUIN (pénalité concernant la surabondance de lien au sein d'un contenu, lorsque ces liens n'ont aucun rapport avec le contenu proposé).
En 2015 débutent les pénalités pour les sites web non adaptés aux mobiles et la même année l'UI et l'UX apparaissent (user interface et user expérience).

4.4.2 Mise en application

L'optimisation du SEO possède trois axes principaux :

Le SEO on-site regroupe toutes les manières d'optimisation du référencement naturel à propos de la structure même du site internet ou du blog. Le SEO on-site couvre toutes les actions « techniques » qui concernent le développement même du site. Il concerne :

- Le fichier robots.txt (le robot.txt regarde quelles pages doivent être indexées dans les moteurs de recherche).
- Le sitemap.xml ou plan de site pour les robots (pour l'indexation automatique).
- Le plan du site HTML.
- La sécurisation https.
- La minification du code html, css js et autres.
- Le regroupement des fichiers CSS.
- La compression des images.
- La gestion du cache du navigateur.
- Les règles de réécritures des URL (par exemple pas d'espace).
- Le fil d'ariane avec des slugs cliquables (identifiant texte d'un contenu Web).
- La connexion à un compte Google Analytics.
- La connexion à un compte Google Search Console.

- La barre de partage vers les réseaux sociaux.
- Les données structurées.

Le SEO on-page désigne l'optimisation du contenu publié sur chaque page web, il concerne :

- Les mots clés du titre H1 de la page.
- La pertinence du texte principal de la page.
- La longueur des textes :
  - Fiche d'un produit : de 200 à 300 mots.
  - FAQ : 300 mots en moyenne.
  - Page catégorie : de 400 à 500 mots.
  - Article de blog : un minimum 800 morts.
  - Guide d'achat : 2000 mots en moyenne.
  - Livre blanc : entre 3000 et 6000 mots, afin d'avoir un contenu qualitatif et pertinent, un minimum de 5000 mots est nécessaire.
- La hiérarchisation du texte principal à l'aide d'intertitres H2, H3, H4 etc…
- La description des images dans la balise alternative.
- Le contenu des balises title et méta-description.

- Le SEO off-page désigne les améliorations du référencement naturel qu'on réalise en dehors de son site web.

  - Envoyer des newsletters et des mails renvoyant vers notre site
  - Animer des réseaux sociaux qui renvoient vers notre site
  - Apparaître dans un journal local
  - Participer à des événements professionnels
  - Etc…

4.5 Choix technologiques

4.5.1 Front-end

HTML (HyperText Markup Language) est un langage de programmation de balisage conçu pour représenter le contenu et l'apparence des pages Web. Ce langage permet d'écrire de l'hypertexte (d'où son nom), de structurer sémantiquement une page web et de mettre en forme son contenu à l'aide de balises. Nous utiliserons pour ce projet la version 5 de HTML.

CSS (Cascading Style Sheets) est un langage de mise en forme des documents HTML et XML. Nous utiliserons pour ce projet la version 3 de CSS.

JavaScript est un langage de programmation de scripts côté navigateur, il est principalement utilisé pour rendre les pages HTML et CSS interactives (comme créer un système de pagination, déplacer du contenu lors d'une action etc…) Ce langage peut aussi servir à extraire une API.

4.5.2 Back-end

PHP (PHP: Hypertext Preprocessor) un langage de programmation de script côté serveur, il sert principalement à faire des pages dynamiques en passant par un serveur HTTP. PHP est un langage impératif orienté objet. Nous utiliserons pour ce projet la version 7 de PHP.

4.5.3 Base de données

SQL (Structured Query Language) est un langage de programmation permettant de manipuler les données et les systèmes de bases de données relationnelles.

Laragon est un serveur de développement web pour Windows.

Looping est un logiciel de modélisation conceptuelle de données

HeidiSQL est un logiciel fourni avec laragon, c'est un outil de gestion pour les bases de données. Ce logiciel permet aussi de réaliser des requêtes SQL

4.5.4 Autres logiciels utilisés

Figma est un éditeur graphique. Ce logiciel permet de créer des wireframes, des maquettes et des prototypes de projets.

Trello est un logiciel de gestion de projet.

5 Développement du projet

5.1 Modélisation des données

La modélisation des données est l'analyse et la conception de l'information contenue dans le système afin de représenter la structure de ces informations et de structurer le stockage et les traitements informatiques.

La modélisation des données sert à conceptualiser une application en organisant les données.
Le modèle issu de la modélisation sert à prédire la façon dont les données entreront et sortiront de la base de données.

Merise est une méthode d'analyse, de conception et de gestion de projet informatique.

Elle fut introduite pour la première fois en 1980, depuis elle est fortement utilisée en France.
La méthode Merise est issue des travaux menés par René Colletti, Arnold Rochfeld et Hubert Tardieu dans les années 1970. Grace à la demande du ministère de l'Industrie, elle est devenue exploitable au début des années 1980.
Elle a été développée et affinée à tel point qu'elle est aujourd'hui adoptée par la plupart des grandes organisations en France.

Dans la méthodologie Merise, il existe plusieurs couches dédiées au traitement des données. Le Modèle Conceptuel de Données (MCD) et le Modèle Logique des Données (MLD) sont deux des principaux outils.

Le Modèle Conceptuel de Données (MCD) permet d'écrire les données qui seront utilisées par un système d'information. Le but est d'obtenir une représentation de données avant d'écrire les informations en base de données.
On utilisera des entités pour décrire les systèmes d'information, des associations pour les relier entre elles et des cardinalités pour voir quel type de relation ont les entités entre elles.
Une entité est composée d'un identifiant unique et d'un ou plusieurs attributs (aussi appelés propriété identifiante et propriété quelconque).

Le MLD ou Modèle Logique des Données est simplement la représentation textuelle du MCD.

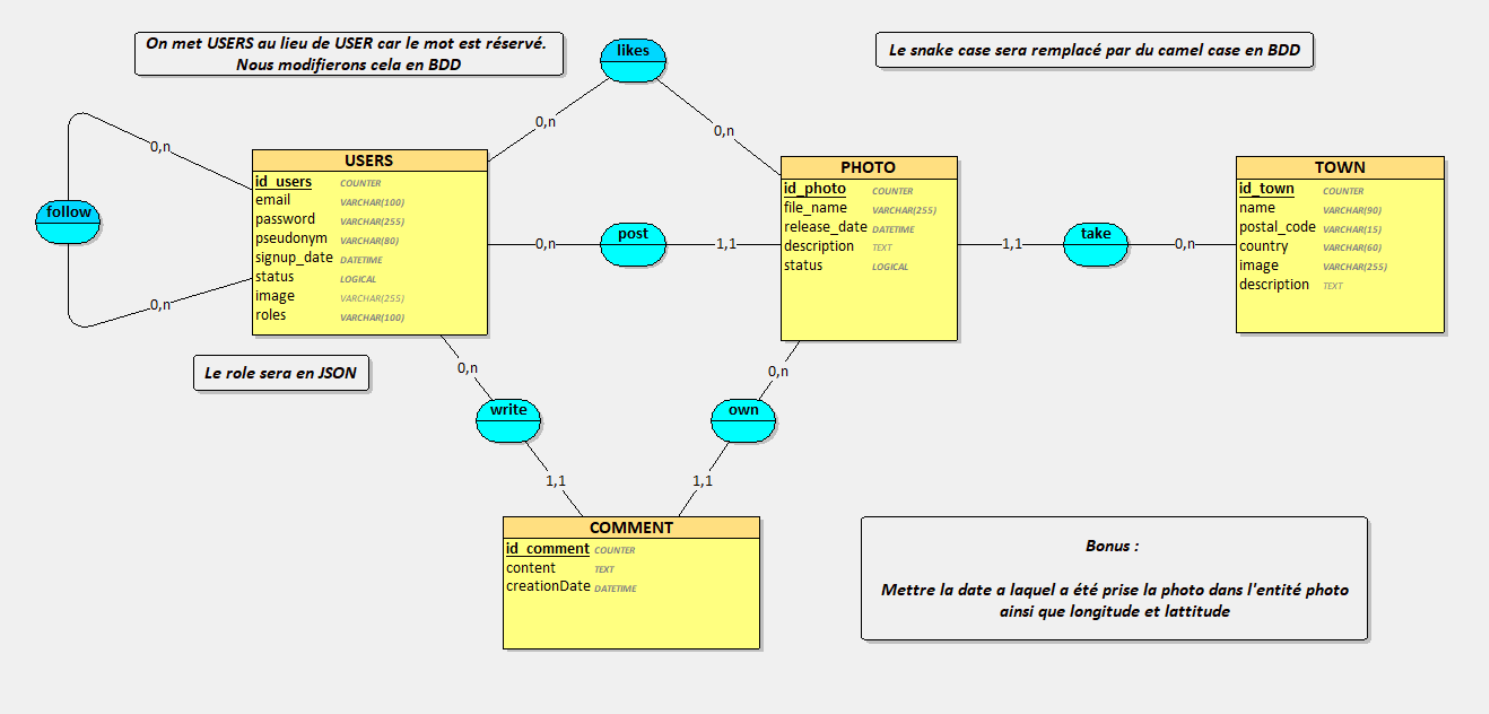
Lorsque l'on effectue le passage du modèle conceptuel de données vers le modèle logique de données, les entités deviennent des tables et les propriétés identifiantes deviennent des clés primaires.

Suivant les cardinalités et les associations du MCD, le MLD va générer une ou plusieurs clés étrangères dans les entités.
C'est avec ces clés étrangères que nous relierons nos futures tables entre elles en base de données.

Une clé primaire sert à garantir **l'unicité d'un enregistrement** dans la base de données.
Les clés étrangères servent à garantir **l'intégrité référentielle entre deux tables**.

A noter qu'une association peut devenir une table associative. Cela se produit lorsque les cardinalités ont pour valeur 0-n ou 1-n de chaque côté de l'association.
Elle sera considérée comme une table en base de données avec pour identifiant deux ou plusieurs clés étrangères.

5.2 Création du modèle conceptuel de données (MCD)



On peut voir ci-dessus la réalisation de notre modèle conceptuel de données.

Nous avons créé 4 entités : users (utilisateur), photo (photo), town (ville) et comment (commentaire).
Un utilisateur renseignera son e-mail, son mot de passe (celui-ci sera haché en base de données), son pseudonyme et une image.
La date d'inscription, le statut et le rôle seront gérés indépendamment. La date d'inscription sera calculée au moment où l'utilisateur créera son profil. Le statut sera automatiquement « non banni » et le rôle sera automatiquement « utilisateur » au moment de la création du compte.

Un commentaire sera composé du contenu du message ainsi que de la date de création de celui-ci. Une date de modification est aussi prévue dans le cas où l'utilisateur modifierait son message, la date de modification a comme valeur par défaut : « null ».

Une photo sera composée du nom du fichier (nous créerons ce nom nous même avant de l'insérer en base de données afin qu'il soit impossible que deux images aient le même nom).

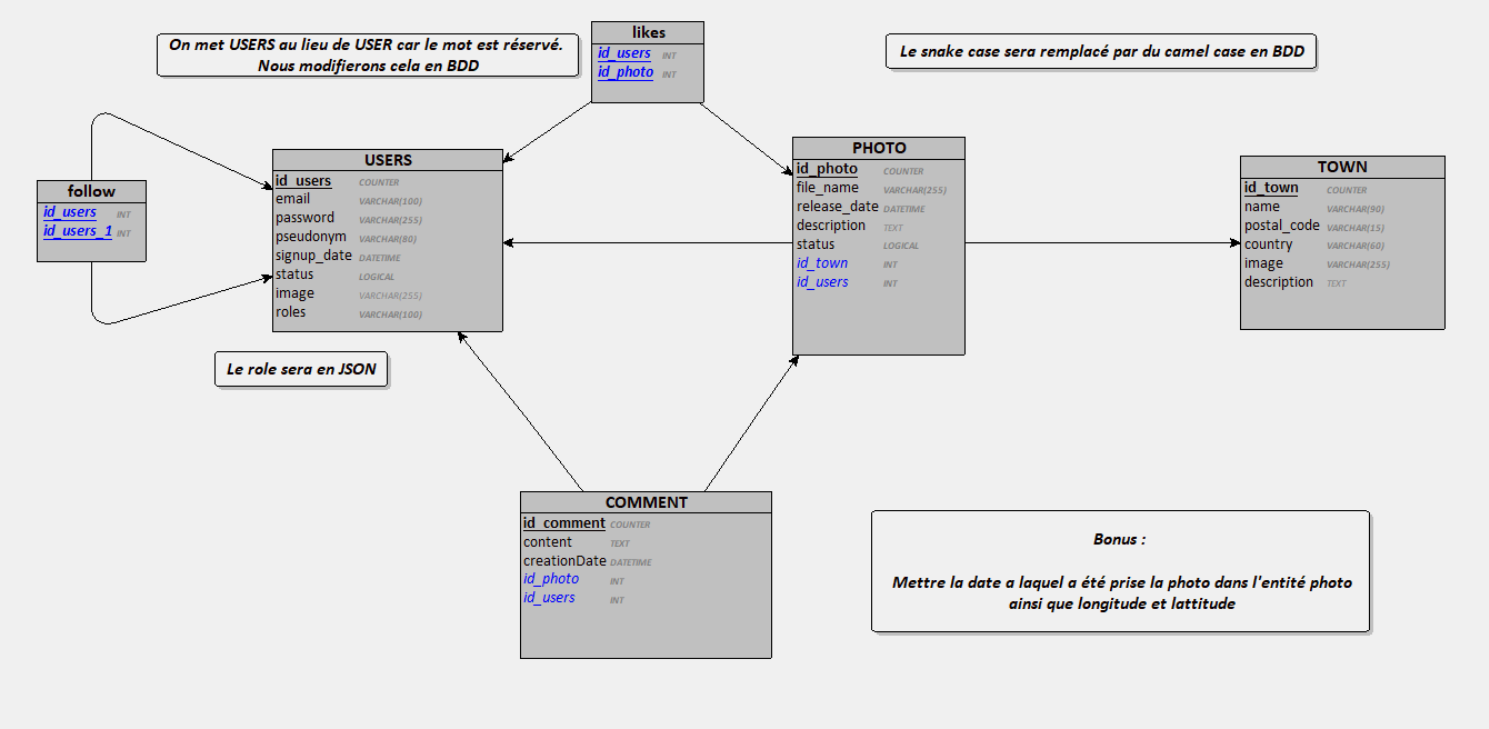
L’image sera stockée dans un dossier et nous afficherons le lien de ce dossier dans le code afin d’afficher l’image.
La date de parution sera créée automatiquement au moment où l’utilisateur ajoutera l’image sur le site.
L’utilisateur pourra, s’il le souhaite, ajouter une description à sa photo.

Les villes seront exclusivement gérées par l’administrateur.

Pour les cardinalités :

- Un utilisateur peut suivre aucun ou plusieurs utilisateurs, un utilisateur peut être suivi par aucun ou plusieurs utilisateurs.
  C’est une association qui débute et finit sur la même entité, aussi appelée association réflexive.
- Un utilisateur peut ajouter aucun ou plusieurs commentaires, de même pour les photos.
- Une photo ou un commentaire ne peuvent être ajoutés que par un seul et unique utilisateur.
- Le reste des cardinalités sont réalisées suivant la même logique.

5.3 Génération du modèle logique de données (MLD)



Le logiciel looping nous permet de générer automatiquement le MLD qui découle du MCD en temps-réel.

On constate que les entités sont devenues des tables, les identifiants sont devenus des clés primaires et des clés étrangères sont apparues dans nos entités, ceci afin de nous permettre de les relier ensemble en base de données. Ces clés étrangères sont générées en fonction des cardinalités que nous avons renseignées dans notre MCD.

On peut aussi voir que les deux associations follow et likes sont devenues des tables associatives. Ce seront des tables en base de données, la clé primaire d'une table associative est l'association des clés étrangères qui la compose.

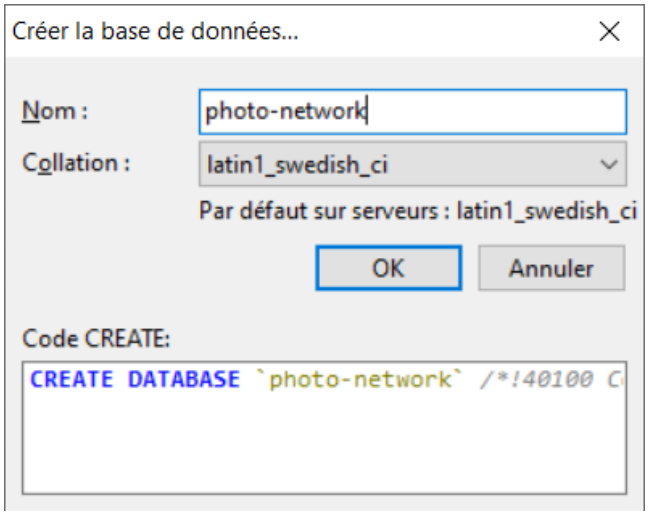
5.3 Base de données

5.3.1 Création de la base de données

A présent que notre modèle logique de données (MLD) est fini, nous pouvons nous en servir comme modèle afin de créer notre base de données.
Pour ce faire nous nous servirons du logiciel HeidiSQL fournit avec Laragon.

Une base de données permet de stocker et de retrouver des données structurées, semi-structurées ou des données brutes ou de l'information, souvent en rapport avec un thème ou une activité ; celles-ci peuvent être de natures différentes et plus ou moins reliées entre elles.

Tout d'abord il faut nommer notre base de données, ici « photo-network », et choisir la manière de représenter le texte Unicode. Ici j'ai choisi latin1_swedish_ci car c'est la valeur proposée par défaut.
Les valeurs par défaut sont généralement choisies parce qu'elles constituent le meilleur choix universel.



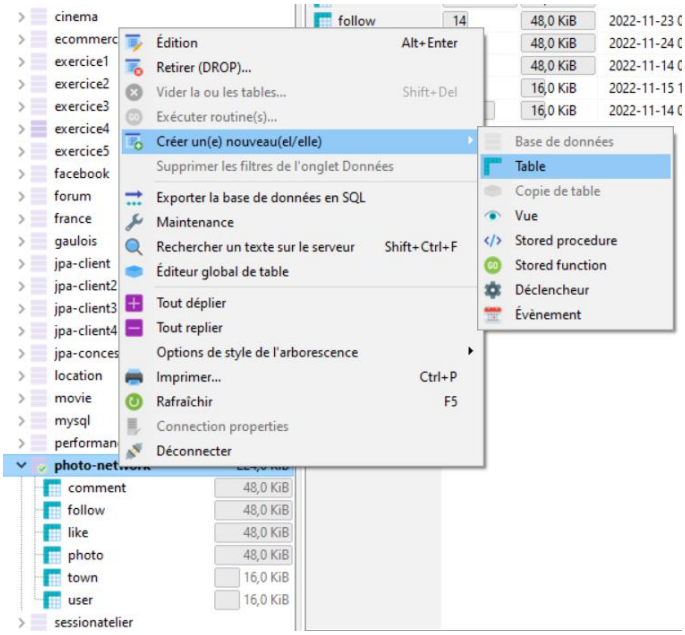
On pourrait s'interroger et se demander pourquoi latin1_swedish est la valeur par défaut pour MySQL car il semblerait que l'UTF-8 soit plus compatible.

latin1 était le jeu de caractères par défaut à l'époque du pré-multioctet et il semblerait que cela ait été maintenu, probablement pour des raisons de compatibilité.

Étant donné que nous ne voulons pas nous risquer à un quelconque problème nous garderons la valeur par défaut « latin1_swedish_ci ».

5.3.2 Création des tables de la base de données

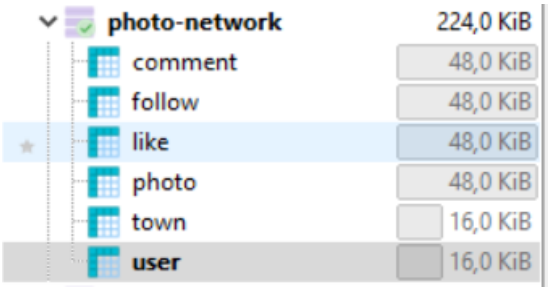
A présent que notre base de données est créée, il nous faut créer les tables qui composent cette base de données.



Une **table** est une catégorie d'information d'une base de données. C'est un tableau composé d'un ensemble de **lignes** et de **colonnes**.
Une ligne représente un enregistrement.
Une colonne va contenir des données d'un même type liées aux enregistrements, comme par exemple, le nom, le type de données, la taille ou encore la valeur par défaut.

A l'aide de notre Modèle logique de données nous connaissons nos tables : follow, user, like, photo, comment ainsi que town.

Nous pouvons donc les créer comme ci-dessous :

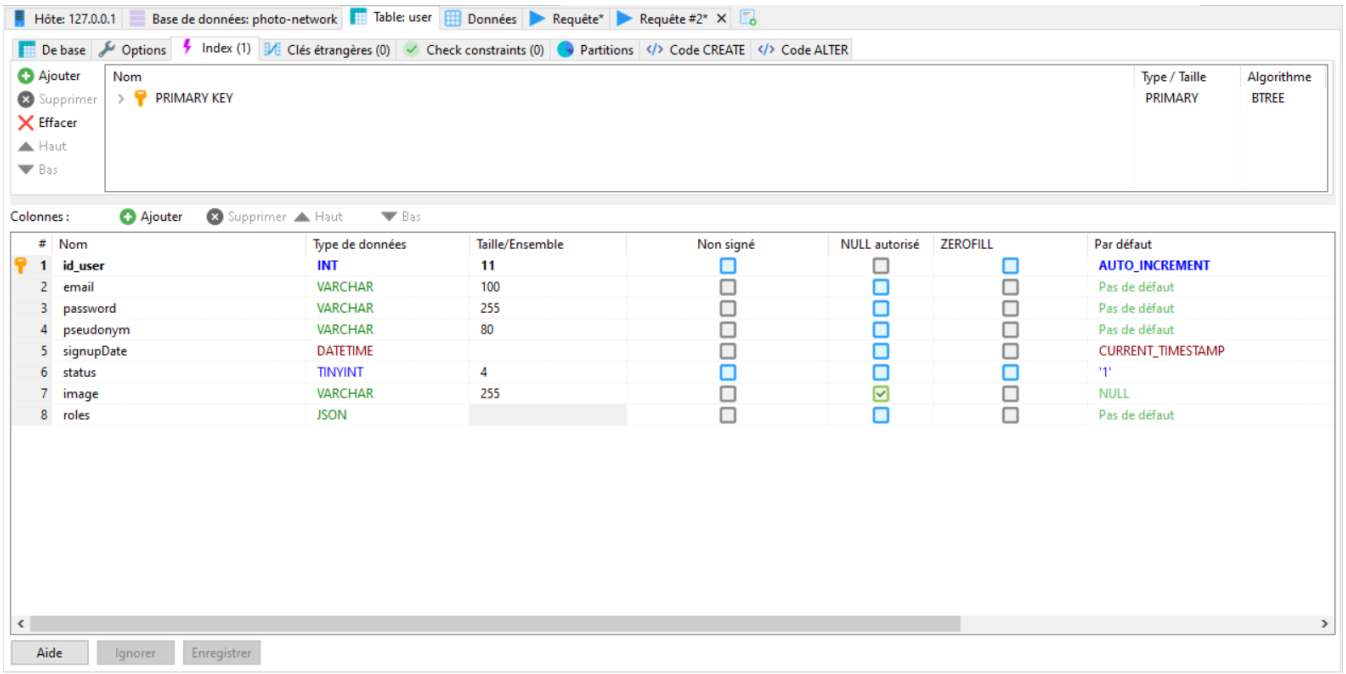


Il nous faut à présent peupler ces tables en renseignant chaque ligne (enregistrement aussi appelé entrée) et chaque colonne (donnée correspondant au type de la colonne).

Il suffit de regarder notre MLD pour connaître nos enregistrements. Par exemple pour l'utilisateur (user) :

- id_user (identifiant de notre utilisateur, cette valeur sera auto-incrémentée) Il est très important d'avoir un identifiant. Cela permet de distinguer les futures données que nous placerons dans cette table. Cet enregistrement sera notre clé primaire.)
- email (l'email de l'utilisateur, ici un varchar (string) de 100 caractères).
- password (le mot de passe de l'utilisateur, celui-ci sera haché en base de données. Nous avons donc besoin du maximum de place disponible soit un varchar de 255 charactères).
- pseudonym (le pseudonyme de l'utilisateur, ici un varchar de 80 charactères).
- signupDate (la date de création du compte de l'utilisateur).
- status (un booléen indiquant si l'utilisateur est banni ou non).
- image (l'image liée au profil de l'utilisateur sera stockée sur notre disque C :, ici cet enregistrement est un varchar de 255 caractères. Lorsque l'utilisateur enverra une image sur le site, le nom de son image sera modifié afin qu'il soit impossible que deux images aient le même nom dans notre base de données. Nous prévoyons donc, avec la taille maximum possible pour un varchar).
- rôles (un rôle renseigné en JSON, soit utilisateur soit administrateur).

Notre MLD nous renseigne sur les types d'informations de nos enregistrements et nous permet de compléter la table user :



Nous avons simplement rentré les informations de notre MLD, à quelques exceptions près :

- Le statut est un booléen. Sa valeur par défaut est 1 .
- L'image peut n'avoir aucune valeur.
- La date d'inscription est automatiquement gérée avec la fonction current_timestamp() en tant que valeur par défaut. Ici heidiSQL nous permet de faire cela. Dans le cas contraire il aurait fallut créer cette fonction dans le construct de la class User :

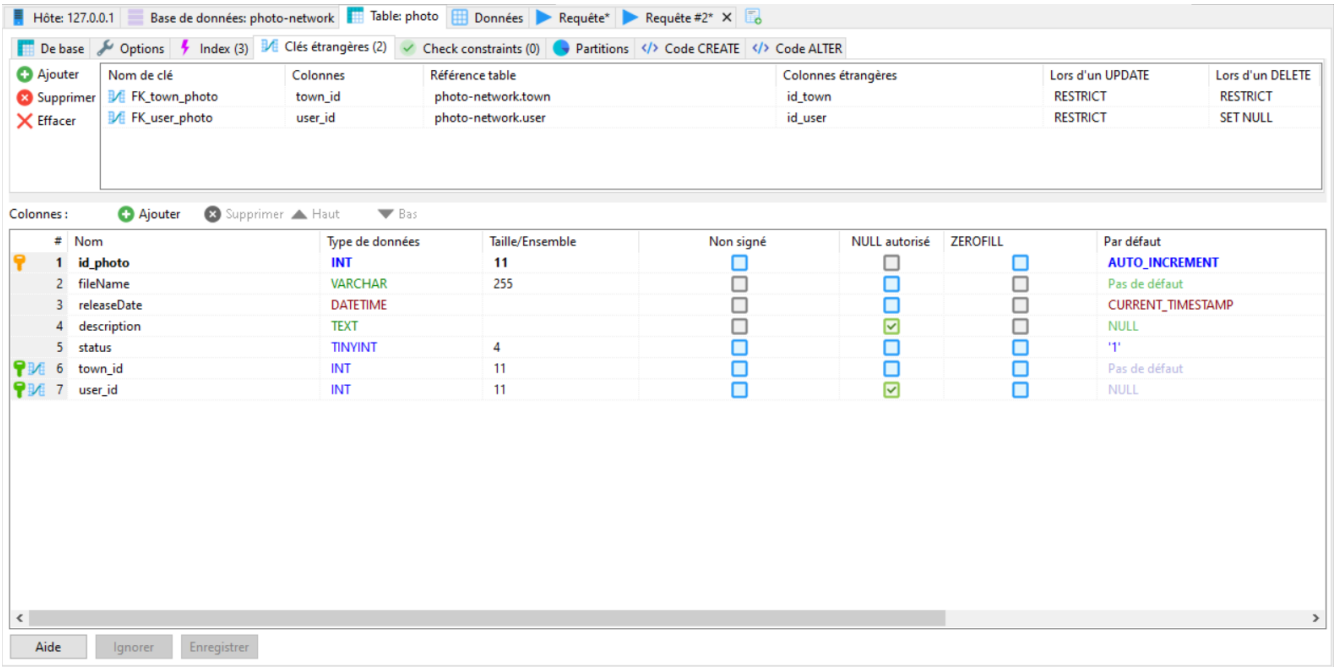
```
$this->_signupDate = new DateTime("now", new DateTimeZone('Europe/Paris'));
```

Une autre manière serait de créer cette fonction dans le controller, à l'endroit où l'on recoit le formulaire d'inscription de l'utilisateur. Il faudrait stocker cette fonction dans une variable puis penser à l'intégrer dans la requête SQL INSERT INTO du user.

En suivant le même modèle nous pouvons peupler la totalité de nos tables.

Nous pouvons voir ci-dessous la réalisation de la table photo, comment ainsi que follow :

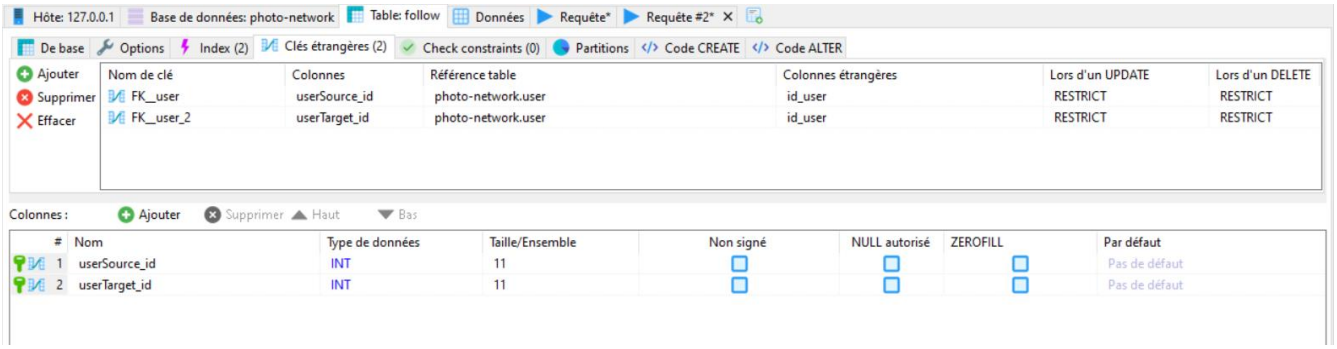
Table photo :



Ici nous avons deux clés étrangères dans cette table, c'est la seule particularité par rapport à la table user.
La clé étrangère town_id correspond à l'identifiant id_town de la table town. Elle est RESTRICT lors d'un DELETE, cela signifie que si l'on supprime une ville qui contient des photos une erreur apparaîtra à l'écran et la requête de suppression ne se fera pas. On aurait pu mettre ce paramètre en CASCADE afin de supprimer automatiquement toutes les photos liées à cette ville mais il est préférable de ne pas le faire dans le cas d'une suppression non voulue par l'administrateur.

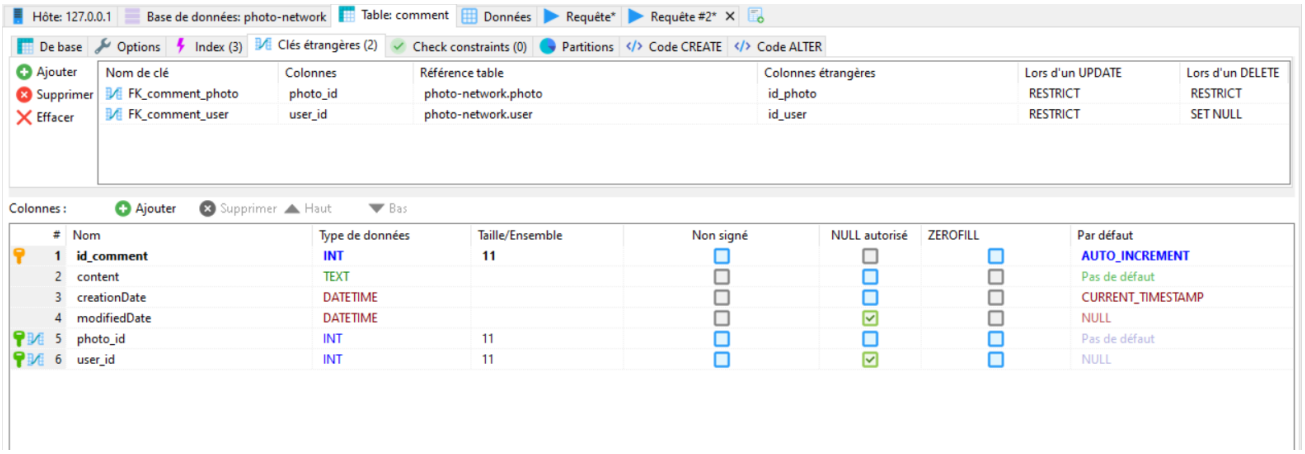
La clé étrangère user_id correspond à l'identifiant id_user de la table user. Elle est SET NULL lors d'un DELETE, cela signifie que si l'utilisateur ayant posté cette photo supprime son compte, la photo continuera d'exister avec NULL comme valeur pour son utilisateur.

Table follow :



Cette table est une table associative, elle a été générée par notre MLD suite à une association Many too many.
La clé primaire de la table follow est l'association des clés étrangères userSource_id et userTarget_id.

Table comment :



Ici rien de particulier, si ce n'est la modifiedDate (date de modification du message) qui a comme valeur par défaut null (pas de valeur attribuée).
Lorsqu'un utilisateur postera un message, on affichera la date de création du message, la date de modification du message n'aura à ce moment aucune valeur attribuée.
C'est seulement si l'utilisateur modifie son message que nous intégrerons une valeur à la date de modification.

On constate que la totalité des identifiants de nos tables se présente sous la forme, id_nom pour une clé primaire et nom_id pour une clé secondaire.
Nous avons fait le choix de mettre des tirets du bas (_) uniquement dans le nom des clés primaires et étrangères afin de réaliser par la suite un système d'hydratation que nous détaillerons plus tard.

5.4 Design Pattern

A présent que notre base de données est peuplée, nous pouvons nous atteler à la plus grosse partie : le Code.

Il serait possible de coder la totalité du projet sur 4-5 fichiers PHP, chacune en One page mais cela est peu recommandable d'un point de vue pratique, de plus, cela ne fait pas partie des normes recommandées aujourd'hui.

C'est pourquoi nous utiliserons un **Design Pattern**, patron de conception en français, plus particulièrement un MVP (Model View Présentateur).

Pourquoi utiliser un Design Pattern et quel est le but recherché ?

Il y a encore quelques dizaines d'années chaque développeur avait sa manière de coder et sa manière d'organiser un projet.
Il était donc difficile de relire et de comprendre le code d'un autre développeur.
C'est pourquoi des modèles de Design Pattern ont émergés et ont fini par être reconnus à partir de 1994.

Un Design Pattern (patron de conception) décrit un **arrangement** récurrent de **rôles** et d'**opérations** exécutés par les modules d'un logiciel.

L'utilisation d'un Design Pattern permet d'identifier les problèmes de conception de logiciels plus rapidement et permet ainsi de calculer le temps et les ressources nécessaires pour les résoudre. C'est pourquoi l'emploi d'un Design Pattern est considéré comme une bonne pratique.
Les Design Pattern sont issus de l'expérience des concepteurs de logiciels.

Utiliser un design Pattern permet aussi d'accélérer le temps de développement d'un projet. Leurs conceptions ont été réalisées de manière à optimiser au maximum le temps de développement d'un projet.

Un autre attrait des designs Pattern est la séparation des responsabilités au sein d'une équipe. Les tâches et les charges seront réparties entre les différents groupes de développeurs, à défaut de laisser toutes les responsabilités à la même personne.

5.4.1 MVC (Model View Controller)

Un MVC (Model View Controller, soit Modèle Vue Contrôleur en français) est une manière d'organiser la structure d'un logiciel. Un MVC est un Design Pattern.

Un **MVC** (Modèle Vue Contrôleur) est une combinaison de plusieurs Design Pattern, on y compte des patrons **observer** (ce patron crée une relation un à plusieurs entre des objets), **strategy** (permet d'encapsuler des familles d'algorithmes de manière à les rendre interchangeables) et **composite** (permet de composer une hiérarchie entre les différents objets).
L'architecture MVC est parmi les plus populaires aujourd'hui au sein des développeurs.

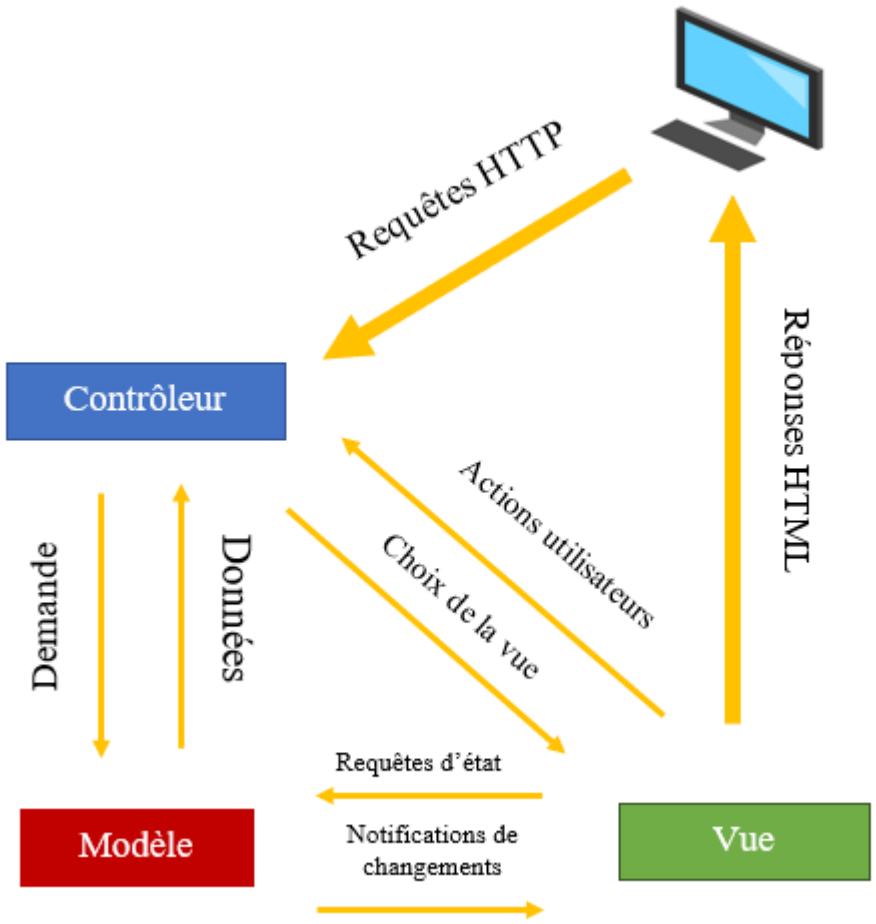
La totalité des frameworks orientés objets comme Symfony, CodeIgniter, CakePHP, RubyOnRails etc… implémentent un pattern MVC ou l'une de ses variantes (MVP, MVVM, MVPC…).

On entend régulièrement parler du MVC mais il s'agit souvent de maladresse pour désigner l'une de ses variantes.

Il est à noter que les variantes du MVC modifient la base de réflexion du pattern.

5.4.2 Fonctionnement du MVC

L'architecture MVC est composée de trois entités : le modèle, la vue et le contrôleur.



Le modèle est la partie relative au traitement des données.

La vue est la partie que l'on affiche à l'utilisateur.
Le contrôleur est la partie qui va contenir les méthodes appelées lors de la requête http et traiter la demande de l'utilisateur.

On pourrait faire l'analogie avec un orchestre. Les partitions de musiques seraient le modèle, les musiciens jouant la musique seraient la vue et le chef d'orchestre serait le contrôleur.

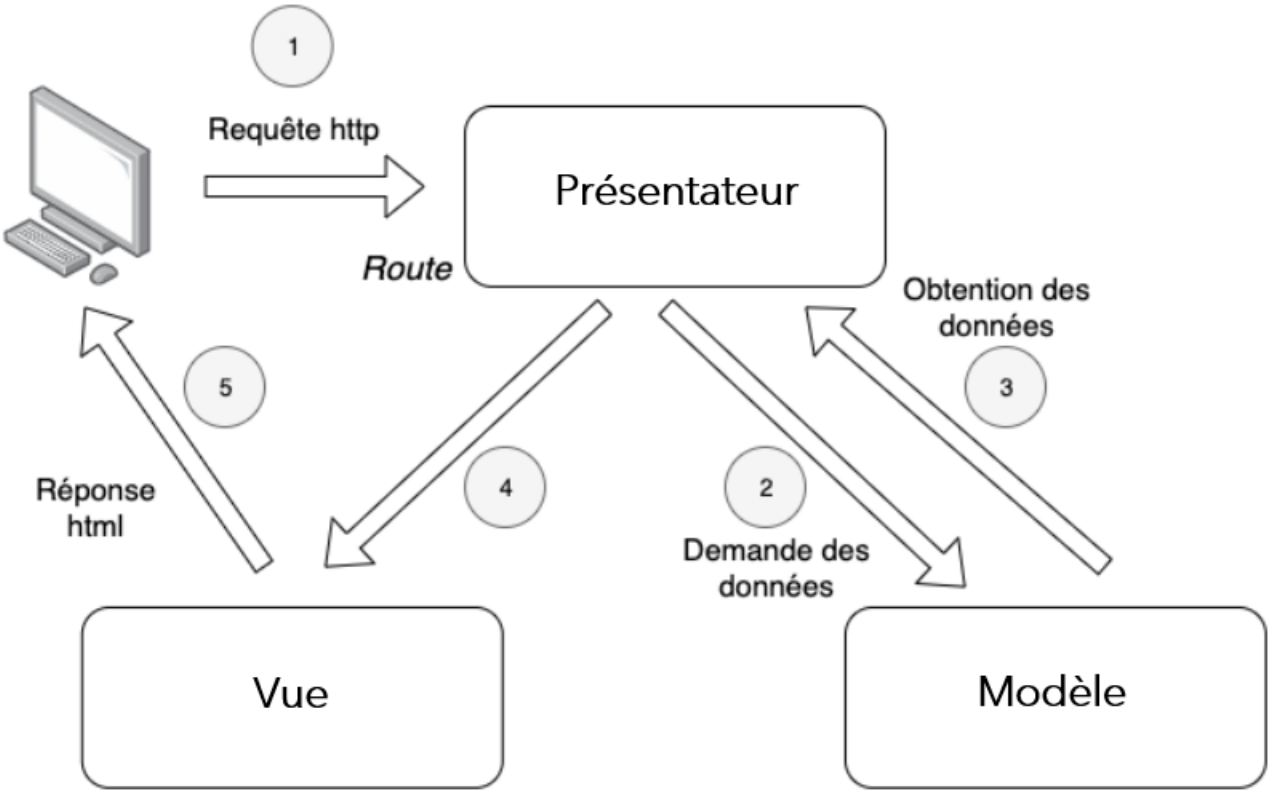
5.4.3 MVP (Model View Presenter)

L'architecture MVP (Model View Presenter, soit Modèle Vue Présentateur en français) est une variante du MVC.

La principale différence est que dans un MVP le modèle et la vue ne communiquent pas. La vue est directement retranscrite par une réponse HTML chez l'utilisateur.

L'architecture MVP est le Design pattern choisi pour réaliser ce projet.

On peut voir ci-dessous les 5 étapes d'un MVP :



Étape 1 :
L'utilisateur envoie une requête HTTP à notre contrôleur.
HTTP signifie **Hypertext Transfer Protocol**, soit **protocole de transfert hypertexte** en français.
Le protocole HTTP réalise la totalité des transferts de données sur le Web. Les requêtes sont généralement initiées par un navigateur Web qui a besoin de ressources récupérées telles que des documents HTML. Il s'agit d'un protocole client-serveur où les demandes sont faites par le destinataire.

Étape 2 :
Notre contrôleur traite la requête. Il va appeler en conséquence les méthodes à réaliser avant le traitement des données (par exemple vérification du Captcha, vérification que l'utilisateur est bien en SESSION etc..) puis notre contrôleur va appeler la couche modèle afin de demander les données demandées par l'utilisateur.

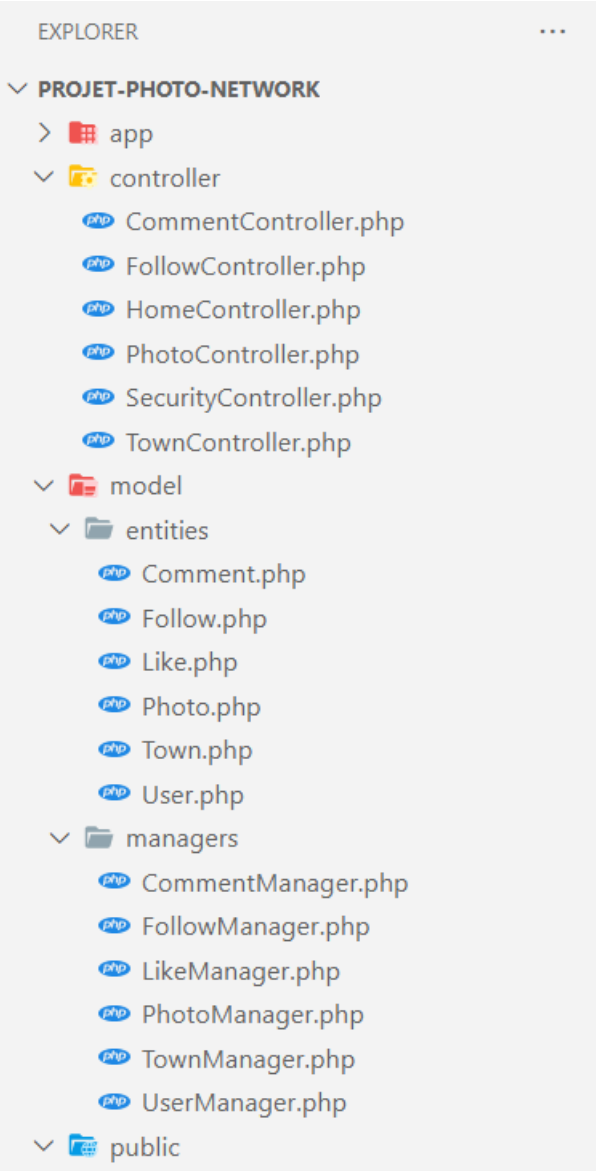
Étape 3 :
Notre modèle va appeler la base de données et va réaliser l'action souhaitée.
Dans notre cas le modèle va se connecter à la base de données via PDO (PHP Data Object) et réaliser l'action souhaitée par le biais d'une requête SQL.
Le modèle envoie ensuite les données au contrôleur.

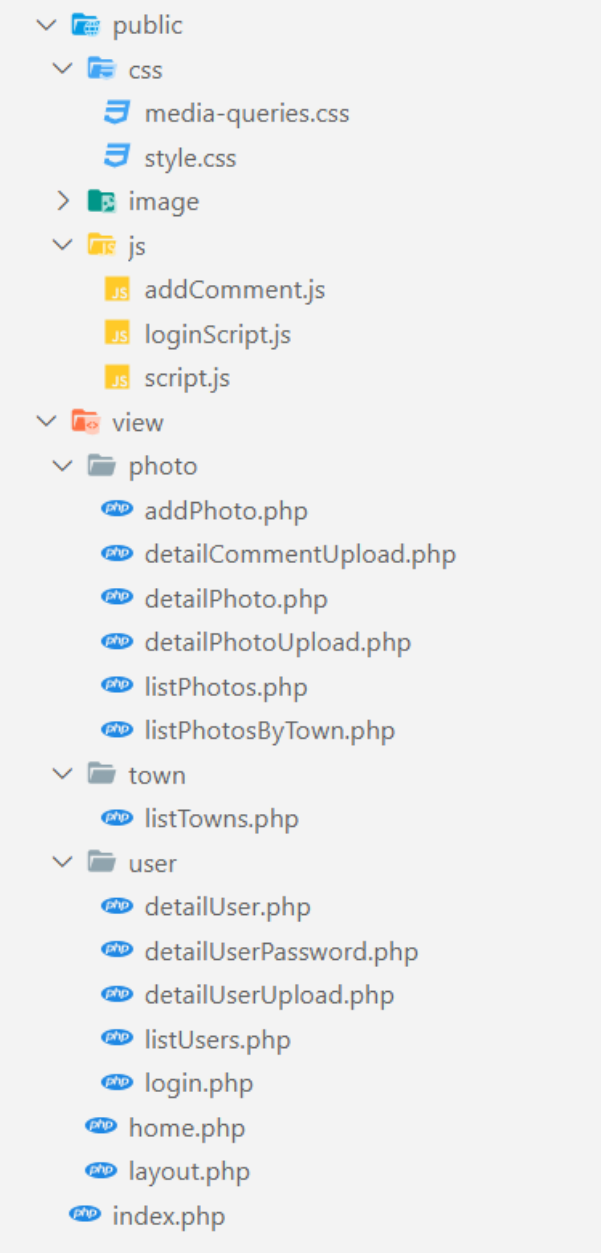
Étape 4 :
Le contrôleur va sélectionner la vue, envoyer les données à cette vue et rediriger l'utilisateur sur cette vue.

Étape 5 :
L'utilisateur reçoit le code de réponse HTTP et si tout s'est bien déroulé accède à la vue souhaitée.

5.4.4 Organisation du projet

On peut voir ci-dessous les fichiers du projet. Le dossier app contient les fichiers nécessaires au fonctionnement du programme. Le dossier public contient tous les éléments pouvant être vus à partir de l'inspecteur des éléments (tout ce qui est public d'où le nom). Ici il contient le CSS, les images, ainsi que le javascript.
On peut aussi voir un index.php à la racine du projet car ce sera le premier fichier lu par notre programme, nous le détaillerons par la suite.
On constate que l'on retrouve bien nos trois couches controller, model et view.





6 Programmation du projet

6.1 Programmation orientée objet

Nous avons choisi d'utiliser PHP orientée objet afin de créer ce projet.

La programmation orientée objet permet de mieux organiser son projet et facilite la maintenance du code. Par-dessus, la POO offre une certaine souplesse pour faire évoluer un programme sans avoir à tout réécrire.

L'idée générale derrière la POO est de regrouper les fonctions similaires au sein d'objets.
Cela permet aux programmeurs de créer des applications représentant des personnes, des objets, des lieux, et d'autres choses dans le monde.

La programmation orientée objet (POO) est un paradigme de programmation informatique, elle est réalisée en utilisant des techniques de programmation basées sur des objets.

Un objet représente un concept, une idée ou toute entité dans le monde physique, comme une maison, un véhicule, un être humain/animal ou même une feuille de papier.

Les objets ont une structure et un comportement internes et peuvent interagir avec leurs pairs. Il s'agira donc de représenter ces objets et leurs relations.

La phase de modélisation est essentielle à la POO. Cela permet de retranscrire des éléments du réel sous une forme virtuelle.

6.2 Fonctionnement du projet

Les actions demandées par l'utilisateur sont réalisées par une méthode GET dans l'URL.
La méthode $_GET est l'une des 9 superglobales de PHP, les superglobales de PHP sont des variables internes qui sont toujours disponibles quel que soit le contexte du script.

La superglobale $_GET est un tableau associatif des valeurs passées au script courant via les paramètres d'URL.

L'URL (Uniform Resource Locator) est une référence à un serveur web. Un URL permet à un navigateur web d'établir une connexion avec ce serveur web.

L'URL et URI (Uniform Resource Identifier) sont les noms que la plupart des utilisateurs associent aux adresses web.

Notre URL est composé en 3 parties principales avec en tout 6 ou 7 parties suivant la page demandée par l'utilisateur :
Prenons par exemple :
http://localhost/projet-photo-network/index.php?ctrl=photo&action=seePhotoById&id=34&page=1

http :	protocole HTTP
localhost/projet-photo-network :	nom du serveur
index.php :	nom du fichier spécifique demandé. Dans notre cas, chaque requête de l'utilisateur passera par l'index car c'est lui qui va appeler les contrôleurs et les méthodes nécessaires à la réalisation des actions demandées.
ctrl :	nom du contrôleur à appeler.
action :	nom de la méthode à utiliser.
id :	Si un Id est nécessaire afin de pouvoir différencier les utilisateurs, photos, commentaires etc…
page :	S'il faut pouvoir différencier les pages en plus d'un Id. (page sera uniquement disponible sur le détail d'une photo pour la section commentaire).

6.3 Index du projet

Comme pour tout projet, c'est toujours par l'index que le code se lance.
Quel que soit le type de page sur lequel on se trouve, le fichier index.php est toujours utilisé et disponible. Chaque URL de notre site contient index.php.

D'une certaine manière l'index est le contrôleur frontal de notre projet.
Le rôle de notre index est d'intercepter les requêtes http.

Notre fichier index.php se présente sous la forme :

index.php > ...

```php
<?php
    namespace App;

    define('DS', DIRECTORY_SEPARATOR); // le caractère séparateur de dossier (/ ou \)
    // meilleure portabilité sur les différents systêmes.
    define('BASE_DIR', dirname(__FILE__).DS); // pour se simplifier la vie
    define('VIEW_DIR', BASE_DIR."view/");     //le chemin où se trouvent les vues
    define('PUBLIC_DIR', "/public");     //le chemin où se trouvent les fichiers publics (CSS, JS, IMG)

    define('DEFAULT_CTRL', 'Home');//nom du contrôleur par défaut
    define('ADMIN_MAIL', "admin@gmail.com");//mail de l'administrateur

    // Recaptcha v3 (https://developers.google.com/recaptcha/docs/faq#id-like-to-run-automated-tests-with-recaptcha.-what-should-i-do)
    // define('SITE_KEY', '6LeIxAcTAAAAAJcZVRqyHh71UMIEGNQ_MXjiZKhI');
    // define('SECRET_KEY', '6LeIxAcTAAAAAGG-vFI1TnRWxMZNFuojJ4WifJWe');

    // Documentation -> https://developers.google.com/recaptcha/docs/v3
    // Installation -> https://www.youtube.com/watch?v=e0EQ6QHcwDU et https://www.devenir-webmaster.com/V2/TUTO/CHAPITRE/OUTILS/23-recaptcha/
    // Comment faire en local ? -> https://stackoverflow.com/questions/3232904/using-recaptcha-on-localhost
    // Clés -> https://www.google.com/recaptcha/admin/site/588199444/setup
    define('SITE_KEY', '6LcUNg8jAAAAAGkdM5zN7esjkQ5eEcCrxhHP_0pX');
    define('SECRET_KEY', '6LcUNg8jAAAAAOCE6yyehxUKgz7W8WIjJF5NGBz_');
```

On a d'abord créé un namespace App.
Cela signifie que le fichier index.php fait partie du namespace App.
En appelant un namespace nous aurons accès aux fonctions, méthodes publiques ou statiques et informations des fichiers faisant partie du namespace.

Pour accéder au namespace, il nous suffira d'écrire use App ; (pour avoir accès à la totalité des fichier de App) ou use App\index ; (pour avoir accès uniquement à ce fichier).

Un namespace fonctionne comme un require mais pour plusieurs éléments, un namespace (espace de nom en français) est un contexte qui permet d'identifier et grouper un ensemble logique d'éléments utilisés par un logiciel.

Le namespace App n'est jamais appelé entièrement dans notre application, on préfèrera cibler uniquement le fichier recherché :

use App\Session;
use App\AbstractController;
use App\ControllerInterface;

Une fois le namespace défini, on crée les constantes ayant besoin d'être disponibles quel que soit le script en cours.
Ici des liens de redirection (des routes) afin de se simplifier la vie, et deux clés pour le Recaptcha (recapatcha est la version de google du captcha).

```php
require("app/Autoloader.php");

Autoloader::register();

//démarre une session ou récupère la session actuelle
session_start();
//et on intègre la classe Session qui prend la main sur les messages en session
use App\Session as Session;

// si pas de token, initialisation d'un token temporaire hors connexion
if (!Session::getTokenCSRF()){
    Session::setTokenCSRF(bin2hex(random_bytes(32)));
}
```

Par la suite il nous faut charger nos classes. C'est l'Autoloader.php qui va s'en charger.
Autoloader.php est un fichier faisant partie du dossier app.
A chaque fois que l'autoloader va appeler un fichier, ce sera un fichier de classe.
L'autoloader va partir de l'emplacement du fully qualified class name par exemple : model/managers/PhotoManager.php
Puis l'autoloader va charger la classe en question, dans notre exemple c'est la classe Photo qui sera chargée.

Une fois nos classes chargées via l'autoloader, il nous faut démarrer une nouvelle session. Ceci est nécessaire car nous utiliserons la superglobale $_SESSION.
Une superglobale est une variable disponible quel que soit le contexte du script.
Session_start() est une fonction qui démarre une nouvelle session ou reprend une session existante.

Si aucun token n'est en session pour gérer la faille CSRF, il nous faut en créer un.
Bin2hex() convertit des données binaires en une représentation hexadécimale.
Random_bytes() génère une chaîne de longueur arbitraire d'octets aléatoires qui conviennent à une utilisation cryptographique, par exemple pour générer des sels, des clés ou des vecteurs d'initialisation.

```php
//---------REQUETE HTTP INTERCEPTEE-----------
    $ctrlname = DEFAULT_CTRL;//on prend le controller par défaut
    //ex : index.php?ctrl=home
    if(isset($_GET['ctrl'])){
        $ctrlname = $_GET['ctrl'];
    }
    //on construit le namespace de la classe Controller à appeller
    $ctrlNS = "controller\\".ucfirst($ctrlname)."Controller";
    //on vérifie que le namespace pointe vers une classe qui existe
    if(!class_exists($ctrlNS)){
        //si c'est pas le cas, on choisit le namespace du controller par défaut
        $ctrlNS = "controller\\".DEFAULT_CTRL."Controller";
    }
    $ctrl = new $ctrlNS();

    $action = "index";//action par défaut de n'importe quel contrôleur
    //si l'action est présente dans l'url ET que la méthode correspondante existe dans le ctrl
    if(isset($_GET['action']) && method_exists($ctrl, $_GET['action'])){
        //la méthode à appeller sera celle de l'url
        $action = $_GET['action'];
    }
    if(isset($_GET['id'])){
        $id = $_GET['id'];
    }
    else $id = null;
    //ex : HomeController->users(null)
    $result = $ctrl->$action($id);
```

La partie qui suit consiste à appeler le bon contrôleur, la bonne méthode du contrôleur en question ainsi qu'un id si besoin.
Le système de pagination est géré indépendamment.

C'est notre système de rooting.

C'est au moment où la requête http est interceptée que l'on va récupérer via la superglobale $_GET les mot clés : ctrl (contrôleur), action (méthode), et id (si défini).
Si aucun contrôleur n'est appelé ou si le contôleur appelé n'existe pas, c'est le contrôleur par défaut qui sera selectionné, ici HomeController.

Il faut pouvoir accéder au contôleur selectionné, pour cela nous allons chercher le cheminement du fichier du contrôleur souhaité. Puis, instancier un objet de la classe de ce contrôleur ($ctrl = new $ctrlNS();) afin d'avoir accès aux méthodes et aux attributs de cet objet. Ici seules les méthodes nous intéressent, nos contrôleurs ne possèdent pas d'attributs.

Chaque objet instancié par la classe d'un contrôleur a, par défaut, une méthode index. C'est cette méthode qui sera appelée dans le cas où une action présente dans l'URL n'aurait pas de méthode correspondante.

Pour l'id, il vaudra simplement null si aucun id n'est présent dans l'URL.
Il reste à placer la requête demandée dans une variable, ici dans $result

```
66  /*--------CHARGEMENT PAGE--------*/
67
68  if($action == "ajax"){//si l'action était ajax
69      echo $result;//on affiche directement le return du contrôleur (car la réponse HTTP sera uniquement celle-ci)
70  }
71  else{
72      ob_start();//démarre un buffer (tampon de sortie)
73      /*la vue s'insère dans le buffer qui devra être vidé au milieu du layout*/
74      include($result['view']);
75      /*je mets cet affichage dans une variable*/
76      $page = ob_get_contents();
77
78      /*j'efface le tampon*/
79      ob_end_clean();
80      /*j'affiche le template principal (layout)*/
81      include VIEW_DIR."layout.php";
82  }
83
84
```

La dernière partie de notre index consiste à mettre en place une temporisation de sortie.
PHP peut bloquer l'envoi des données au navigateur grâce à la fonction ob_start() qui enclenche une temporisation de sortie. Cela signifie que les données ne sont pas directement envoyées mais temporairement mises en mémoire tampon.

Nous avons créé dans notre projet un fichier template (nommé layout.php).
La totalité de nos pages vont toujours inclure notre layout.php.

Chaque fois que l'on va appeler une vue, on va démarrer la temporisation de sortie (via ob_start()), puis mettre en mémoire tampon tout le contenu de la vue, puis utiliser ob_get_content pour récupérer ce qui se trouve en mémoire tampon et le placer dans une variable $page.

Nous utilisons la fonction ob_end_clean pour supprimer ce qu'il y'a en mémoire tampon afin de ne pas stocker du contenu devenu inutile.

La variable $page sera ensuite affichée dans notre layout.php comme ci-dessous :

```
<main id="photo">
    <?= $page ?>
</main>
```

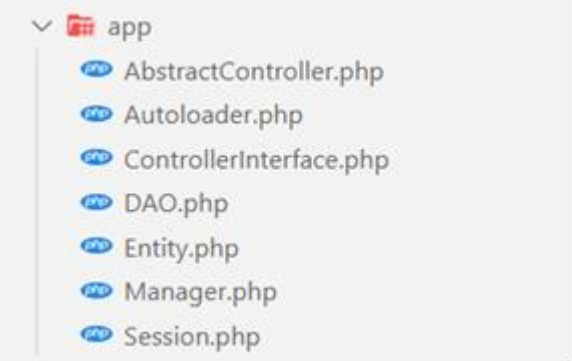
Il reste à inclure (fonction include) notre layout.php afin de l'afficher.

La fonction include inclut et exécute le fichier spécifié en argument. La différence par rapport à la fonction require est que require arrêtera l'exécution du fichier à la ligne à laquelle l'erreur s'est produite. Include continuera à s'exécuter quelles que soient les erreurs rencontrées.

6.4 App

A présent que nous avons détaillé notre système de rooting, expliqué l'URL et présenté l'index, il reste à présenter le dossier app.

Le dossier app se présente sous la forme :



App est un dossier qui comprend tous les fichiers utilisables partout dans l'application. Il contient 7 fichiers : AbstractController.php (utilisé pour les redirections et restrictions), Autoloader.php (utilisé pour charger nos classes), ControllerInterface.php (interface qui si implémentée dans une classe, force à définir une fonction index), DAO.php (connexion à la base de données via PDO et fonctions du CRUD), Entity.php (hydrater nos objets), Manager.php (fonctions du CRUD détaillé), Session.php (fonctions liées à la session).

Le namespace App comprend la totalité des fichiers présents dans app ainsi que l'index.php.

Chaque fichier du dossier app se présente sous la forme de classes.

Les classes sont le cœur de la programmation orientée objet en PHP.
Une classe est composée d'un ensemble d'états de de comportements. Les états sont les propriétés et les méthodes sont le comportement.
D'une certaine façon, une classe est un moule, un mode d'emploi de l'objet.

Afin d'illustrer les descriptions, nous utiliserons des captures d'écrans du projet.

```
final class Photo extends Entity{
```

Un « objet » est une représentation d'une chose matérielle ou immatérielle du réel, à laquelle on associe des propriétés et des actions.
Un objet est toujours créé à partir d'une classe, on dit qu'un objet est instancié d'une classe. C'est pourquoi on pourra parler « d'instance » ou « d'objet »

```
$photoManager = new PhotoManager();
$commentManager = new CommentManager();
```

– Les attributs ou propriétés définissent les caractéristiques d'un objet d'une classe (ou d'une classe lorsque l'objet n'est pas encore instancié).

```
private int $_id;
private string $_fileName;
private DateTime $_releaseDate;
private $_description;
private bool $_status;
private Town $_town;
private $_user;
```

– Les méthodes définissent quant à elles les fonctions propres aux instances d'une classe. Elles vont servir à manipuler l'objet.

```
public function getDescription()
{
    return $this->_description;
}

public function setDescription($description)
{
    $this->_description = $description;
}
```

En POO (programmation orientée objet), l'encapsulation est un mécanisme qui va permettre de rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet.
C'est-à-dire, en empêchant l'accès par un autre moyen que par le service proposé. On parle souvent de visibilité des propriétés et des méthodes.

Public : au sein de la classe et à l'extérieur.
Private : au sein de la classe.
Protected : au sein de la classe et de ses enfants.

```
protected function connect(){
    DAO::connect();
}

public function findAll($order = null){
```

Quelques spécificités supplémentaires des classes :

- Il est possible d'étendre une classe en créant une classe « fille » qui va hériter de toutes les propriétés et méthodes de son parent par défaut et qui va pouvoir les manipuler de la même façon.

```
class Photo extends Entity{
```

- Une méthode ou un attribut peuvent être static, cela signifie qu'ils sont accessibles sans que des objets aient besoin d'être instanciés.
  Static : Modificateur qui permet de partager une propriété entre différents objets d'une même classe.

```
public static function setUser($user){
    $_SESSION["user"] = $user;
}

public static function getUser(){
    return (isset($_SESSION['user'])) ? $_SESSION['user'] : false;
}
```

- Une classe peut être abstraite. Une classe est abstraite si elle contient au moins une méthode abstraite. Elle ne peut pas être instanciée, mais ses sous-classes non abstraites le peuvent.
  Quel est le but d'une class abstraite ? C'est une classe dont les méthodes ont pour but d'être redéfinies dans une classe fille.

```
abstract class DAO{
```

- Il est possible d'accéder aux membres static ou constant, ainsi qu'aux propriétés ou méthodes surchargées d'une classe en appelant l'opérateur de résolution de portée (::).

```
DAO::connect();
```

Regardons à présent plus en détails les fichiers qui composent notre dossier app :

6.4.1 AbstractController.php

```php
<?php
    namespace App;

    abstract class AbstractController{

        public function index(){}

        public function redirectTo($ctrl = null, $action = null, $id = null, $page = null){

            if($ctrl != "home"){
                $url = $ctrl ? "?ctrl=".$ctrl : "";
                $url.= $action ? "&action=".$action : "";
                $url.= $id ? "&id=".$id : "";
                $url.= $page ? "&page=".$page : "";
            }

            else $url = "/";
            header("Location: $url");
            die();

        }

        public function restrictTo($role){

            if(!Session::getUser() || !Session::getUser()->hasRole($role)){
                $this->redirectTo("security", "login");
            }
            return;
        }

    }
```

Ce fichier est une classe abstraite, c'est pour cela qu'elle ne possède pas de constructeur.
La totalité des classes de nos contrôleurs vont hériter (descendre) de la classe AbstractController.
L'utilité est ici de factoriser notre code, nous n'aurons pas à réécrire les fonctions de redirection et de restriction dans chaque contrôleur.

6.4.2 Autoloader.php

```php

    public static function register()
    {
        spl_autoload_register(array(__CLASS__, 'autoload'));
    }


    public static function autoload($class)
    {

        //$class = Model\Managers\VehiculeManager (FullyQualifiedClassName)
        //namespace = Model\Managers, nom de la classe = VehiculeManager

        // on explose notre variable $class par \
        $parts = preg_split('#\\\#', $class);
        //$parts = ['Model', 'Managers', 'VehiculeManager']

        // on extrait le dernier element
        $className = array_pop($parts);
        //$className = VehiculeManager

        // on créé le chemin vers la classe
        // on utilise DS car plus propre et meilleure portabilité entre les différents systèmes (windows/linux)

        $path = strtolower(implode(DS, $parts));
        //$path = 'model/manager'
        $file = $className.'.php';
        //$file = VehiculeManager.php

        $filepath = BASE_DIR.$path.DS.$file;
        //$filepath = model/managers/VehiculeManager.php
        if(file_exists($filepath)){

            require $filepath;
        }
```

Ce fichier sert à charger nos classes à partir du fully qualified class name.
Cela va nous permettre de charger une classe à l'appel du fichier.

6.4.3 ControllerInterface.php

```php
<?php

    namespace App;

    interface ControllerInterface{

        public function index();

    }

```

La totalité des classes de nos contrôleurs vont implémenter l'interface du ControllerInterface. Il leurs faudra donc définir une public fonction index.
Par souci de lisibilité, il est important d'avoir une fonction index dans chacun de nos contrôleurs.

6.4.4 DAO.php

```php
<?php
    namespace App;

    /**
     * Classe d'accès aux données de la BDD, abstraite
     *
     * @property static $bdd l'instance de PDO que la classe stockera lorsque connect() sera appelé
     *
     * @method static connect() connexion à la BDD
     * @method static insert() requètes d'insertion dans la BDD
     * @method static select() requètes de sélection
     */
    abstract class DAO{

        private static $host   = 'mysql:host=127.0.0.1;port=3306';
        private static $dbname = 'photo-network';
        private static $dbuser = 'root';
        private static $dbpass = '';

        private static $bdd;

        /**
         * cette méthode permet de créer l'unique instance de PDO de l'application
         */
        public static function connect(){

            self::$bdd = new \PDO(
                self::$host.';dbname='.self::$dbname,
                self::$dbuser,
                self::$dbpass,
                array(
                    \PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES 'utf8'",
                    \PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION,
                    \PDO::ATTR_DEFAULT_FETCH_MODE => \PDO::FETCH_ASSOC
                )
            );
        }
```

```php
    public static function insert($sql){
        try{
            $stmt = self::$bdd->prepare($sql);
            $stmt->execute();
            //on renvoie l'id de l'enregistrement qui vient d'être ajouté en base,
            //pour s'en servir aussitôt dans le controleur
            return self::$bdd->lastInsertId();

        }
        catch(\Exception $e){
            echo $e->getMessage();
        }
    }

    public static function update($sql, $params){
        try{
            $stmt = self::$bdd->prepare($sql);

            //on renvoie l'état du statement après exécution (true ou false)
            return $stmt->execute($params);

        }
        catch(\Exception $e){

            echo $e->getMessage();
        }
    }

    public static function delete($sql, $params){
        try{
            $stmt = self::$bdd->prepare($sql);

            //on renvoie l'état du statement après exécution (true ou false)
            return $stmt->execute($params);

        }
```

DAO (Data Access Object) est une classe abstraite qui permet de se connecter à la base de données.
Elle est créée dans un souci d'optimisation du code.
DAO.php contient aussi toutes nos fonctions du CRUD (create, read, upload et delete), ceci toujours dans un souci d'optimisation du code afin de ne pas avoir à préparer puis exécuter nos requêtes à chaque fois.

Nous utiliserons à chaque fois PDO pour nous connecter à notre base de données.

PDO (PHP Data object) est une interface qui permet au script PHP d'interroger une base de données via des requêtes SQL. C'est une extension de PHP.
PDO fournit une interface d'abstraction à l'accès des données. C'est-à-dire qu'on ne va pas réécrire le SQL. Mais il ne fournit pas une abstraction de base de données.

6.4.5 Entity.php

```php
<?php
    namespace App;

    abstract class Entity{

        protected function hydrate($data){

            foreach($data as $field => $value){

                //field = marque_id
                //fieldarray = ['marque','id']
                $fieldArray = explode("_", $field);

                if(isset($fieldArray[1]) && $fieldArray[1] == "id"){
                    $manName = ucfirst($fieldArray[0])."Manager";
                    $FQCName = "Model\Managers".DS.$manName;

                    $man = new $FQCName();
                    $value = $man->findOneById($value);
                }
                //fabrication du nom du setter à appeler (ex: setMarque)
                $method = "set".ucfirst($fieldArray[0]);

                if(method_exists($this, $method)){
                    $this->$method($value);
                }

            }
        }

        public function getClass(){
            return get_class($this);
        }
    }
```

Ce fichier est utilisé pour l'hydratation de nos objets, c'est-à-dire à remplir automatiquement les données des propriétés de l'objet à partir de la base de données.
Nous avons mis des tirets du bas (_) uniquement pour les clés primaires et étrangères en base de données.

Pour chaque nom, on regarde si celui-ci est composé d'un tiret du bas. Si c'est le cas, on met dans un tableau la partie avant le tiret du bas en clé 0 du tableau et la partie après le tiret du bas en clé 1 du tableau.
Ainsi, si la clé 1 du tableau vaut « id », nous saurons qu'on est face à une clé étrangère. Nous pourrons donc instancier cet objet et avoir ainsi accès à ses propriétés et méthodes.

6.4.6 Manager.php

```php
<?php
    namespace App;

    abstract class Manager{

        protected function connect(){
            DAO::connect();
        }

        /**
        * get all the records of a table, sorted by optionnal field and order
        *
        * @param array $order an array with field and order option
        * @return Collection a collection of objects hydrated by DAO, which are results of the request sent
        */
        public function findAll($order = null){

            $orderQuery = ($order) ?
                "ORDER BY ".$order[0]. " ".$order[1] :
                "";

            $sql = "SELECT *
                    FROM ".$this->tableName." a
                    ".$orderQuery;

            return $this->getMultipleResults(
                DAO::select($sql),
                $this->className
            );
        }

        public function findOneById($id){

            $sql = "SELECT *
                    FROM ".$this->tableName." a
                    WHERE a.id_".$this->tableName." = :id
                    ";
```

Ce fichier nous sert à factoriser le détail de nos fonctions du CRUD.
Nous appellerons cette classe afin de nous connecter à la base de données.

6.4.7 Session.php

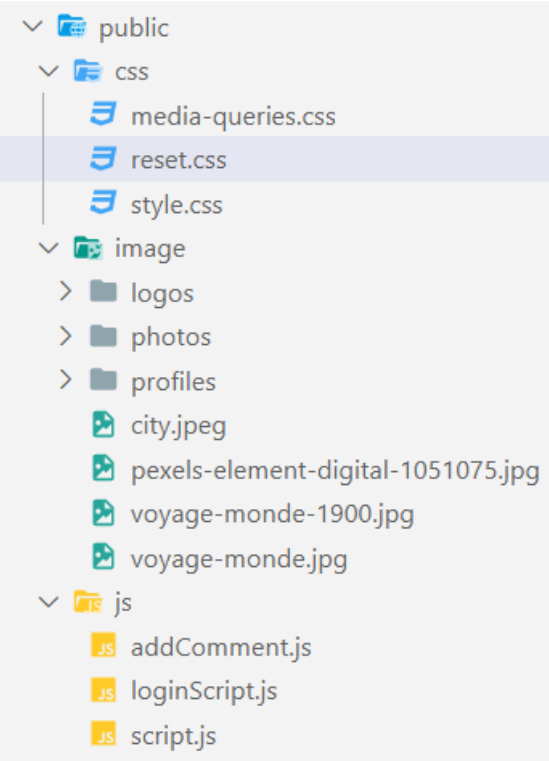
```php
<?php
    namespace App;

    class Session{

        private static $categories = ['error', 'success'];


        //? ajoute un message en session, dans la catégorie $categ

        public static function addFlash($categ, $msg){
            $_SESSION[$categ] = $msg;
        }


        //? renvoie un message de la catégorie $categ, s'il y en a !

        public static function getFlash($categ){

            if(isset($_SESSION[$categ])){
                $msg = $_SESSION[$categ];
                unset($_SESSION[$categ]);
            }
            else $msg = "";

            return $msg;
        }


        //? met un user dans la session (pour le maintenir connecté)

        public static function setUser($user){
            $_SESSION["user"] = $user;
        }
```

Ce fichier est une classe Session. Cette classe contient uniquement des méthode public static.
Cette classe nous sert à factoriser notre code, afin de ne pas avoir à réécrire chaque fois nos fonctions liées à la session.

Il nous reste encore à voir les dossiers controller, model et view que nous détaillerons dans un exemple par la suite.

A noter que notre dossier model est composé de deux parties :
La partie entities qui va stocker nos classes, la partie manager qui va s'occuper du traitement des données et qui va stocker nos requêtes SQL.

Le dossier public contient tous les éléments pouvant être vus à partir de l'inspecteur des éléments (tout ce qui est public). Ici il contient :



- Un dossier image comportant toutes les images de notre site web et trois sous dossiers logos (pour les logos de notre site web), photos (pour les photos que les utilisateurs mettent en ligne sur notre site web) et profiles (pour les images que les utilisateurs mettent en ligne sur leurs profils)

- Un dossier JavaScript pour tous les fichiers Javascript nécessaires à l'interactivité de notre site

- Un dossier CSS composé de :

  - Un fichier reset.css
    Ce fichier permet d'appliquer un style par défaut à toutes les balises HTML. Cela permet d'obtenir le même style de base quel que soit le navigateur utilisé par l'utilisateur

  - Un fichier style.css
    Ce fichier est utilisé afin de styliser notre site web

  - Un fichier media-queries.css
    Ce fichier permet de styliser les versions mobiles (320px jusqu'à 480px),
    les tablettes (481px jusqu'à 768px),
    les petits écrans (769px jusqu'à 1024px),
    les écrans classiques (1025px jusqu'à 1200px)
    et les écrans larges (1021px et plus), de notre site web.
    Nous garderons le style de base et modifierons juste le nécessaire

6.5 Exemple de fonctionnalité

A présent que nous avons détaillé les fondations du projet, nous pouvons voir un exemple de fonctionnalité et comment le réaliser.

Affichons simplement la totalité des photos sur une page de notre site :

Tout d'abord nous devons créer le contrôleur adéquat dans notre dossier controller (si celui-ci n'existe pas déjà). C'est lui qui va gérer la requête lorsqu'elle sera appelée.
Ici nous avons créé le fichier PhotoController.php dans notre dossier controller.

Il nous faut aussi créer la vue sur laquelle nous afficherons la totalité des photos.
Ici nous avons créé le fichier listPhotos.php dans notre dossier view

Ci-dessous notre fichier PhotoController.php :

```php
<?php

    namespace Controller;

    use App\Session;
    use App\AbstractController;
    use App\ControllerInterface;
    use Model\Managers\PhotoManager;
    use Model\Managers\TownManager;
    use Model\Managers\CommentManager;
    use Model\Managers\UserManager;

    class PhotoController extends AbstractController implements ControllerInterface{

        public function index(){

           $photoManager = new PhotoManager();


            return [
                "view" => VIEW_DIR."photo/listPhotos.php",
                "data" => [
                    "photos" => $photoManager->findAll(["releaseDate", "DESC"])
                ]
            ];

        }

```

Une fois notre fichier créé, il nous faut créer la classe PhotoController qui, comme vu précédemment, hérite de la classe AbstractController et implémente l'interface ControllerInterface.

Étant donné que nous implémentons ControllerInterface, nous sommes forcés de redéfinir une fonction index.

La fonction que nous souhaitons développer (afficher toutes les photos) semble propice comme fonction d'index pour notre PhotoController. C'est pourquoi nous la choisirons en tant que fonction index.

Nous commençons par instancier un objet de la classe PhotoManager et le plaçons dans une variable $photoManager.

Nous retournerons par la suite un tableau indexé possédant deux clés : « view » et « data ».

View est la partie vue précédemment dans l'index (`include($result['view']);`).
C'est la route de la vue à appeler. C'est sur cette vue que nous afficherons les informations. C'est dans la vue que sera le code HTML.

Data est un tableau indexé contenant toutes les informations demandées par l'utilisateur. Ici il ne possède qu'une seule valeur mais nous préférons garder le format tableau afin de garder la même syntaxe partout. Il sera facile d'ajouter d'autres informations à la fonction index par-dessus, si besoin.

La classe PhotoManager se présente sous la forme :

```php
<?php
    namespace Model\Managers;

    use App\Manager;
    use App\DAO;

    class PhotoManager extends Manager{

        protected $className = "Model\Entities\Photo";
        protected $tableName = "photo";


        public function __construct(){
            parent::connect();
        }
```

On constate que notre PhotoManager hérite de Manager.
Cela signifie que notre PhotoManager a accès à toutes les méthodes de la classe Manager vues précédemment (du moins les méthodes protected et public).

On définit ici deux attributs : la route de la classe Photo ($className) ainsi que le nom de la table ($tableName).

On peut voir que notre fonction possède un __construct.

Il n'est pas obligé de définir un constructeur dans une classe, mais si l'on souhaite passer des paramètres lors de la construction d'un objet, nous en aurons besoin.

__construct() est le nom de la méthode pour le constructeur.
Dans PHP, __construct() est appelé méthode magique.
Le constructeur est appelé sur un objet après qu'il ait été créé, et c'est le bon endroit pour mettre le code d'initialisation ou faire passer des valeurs aux attributs de notre objet.

En nous servant de l'opérateur de résolution de portée, nous appelons le parent de la classe PhotoManager (donc Manager) et sa méthode connect().
Le manager va par la suite appeler la fonction connect de la classe DAO, encore une fois via l'opérateur de résolution de portée, et se connecter à notre base de données.

Etant donné que Manager est un parent de PhotoManager, nous avons accès à sa méthode findAll(). La fonction findAll permet de trouver tous les enregistrements d'une table.
Dans le cas où la fonction souhaitée ne se trouverait pas dans la classe Manager, il nous faudrait créer une fonction spécifique (une requête SQL) dans notre PhotoManager.

Ci-dessous notre fonction findAll :

```php
public function findAll($order = null){

    $orderQuery = ($order) ?
        "ORDER BY ".$order[0]. " ".$order[1] :
        "";

    $sql = "SELECT *
            FROM ".$this->tableName." a
            ".$orderQuery;

    return $this->getMultipleResults(
        DAO::select($sql),
        $this->className
    );
}
```

La fonction findAll fait appel à deux autres fonctions :
La fonction getMultipleResults de la classe Manager et la fonction select de la classe DAO.

```php
public static function select($sql, $params = null, bool $multiple = true):?array
{
    try{
        $stmt = self::$bdd->prepare($sql);
        $stmt->execute($params);

        $results = ($multiple) ? $stmt->fetchAll() : $stmt->fetch();

        $stmt->closeCursor();
        return ($results == false) ? null : $results;
    }
    catch(\Exception $e){
        echo $e->getMessage();
    }
}

protected function getMultipleResults($rows, $class){

    if(is_iterable($rows)){
        return $this->generate($rows, $class);
    }
    else return null;
}
```

La fonction getMultipleResults fait elle-même appel à une autre fonction generate qui nous permet de créer des itérators que nous allons détailler ci-dessous.

```php
private function generate($rows, $class){
    foreach($rows as $row){
        yield new $class($row);
    }
}
```

Il était demandé de traduire une documentation d'au moins 700 caractères afin de prouver la maîtrise de la langue anglaise.
Nous nous servirons de cette documentation afin d'expliquer ce qu'est un iterator ainsi qu'un generator.
Lien : https://theiconic.tech/why-and-when-to-use-generator-instead-of-array-in-php-fe5c60231746

Why and when to use a generator instead of an array in PHP:

A generator is a VERY powerful concept that provides a host of benefits. I believe they are very under-used and under-appreciated by PHP developers. In this post, I will explain what are the advantages of generators and some of the example usages that will hopefully motivate you to use more and more generators in PHP.

First Things First, What is an Iterator in PHP

A generator is a special type of Iterator that provides an easy way to implement Iterators in PHP for a lot of use-cases. So, we need to understand what is an Iterator before we learn what is a generator.

Arrays are not the only thing over which you can iterate using `foreach` in PHP. Any instance of any class implementing Iterator interface can be iterated using `foreach`.

Here's a simple example of a file iterator. It simply receives a file path as an argument and then you can iterate over the object using `foreach`.

Why Iterator instead of Array

Suppose there's a function that expects a list of image contents to upload it to somewhere like AWS S3 bucket.

```php
<?php

function uploadImages(iterable $images) {
    foreach ($images as $image) {
        // code to upload image to somewhere
    }
}
```

The problem with the above code is that you load all the images to memory even before you need any one of the images.

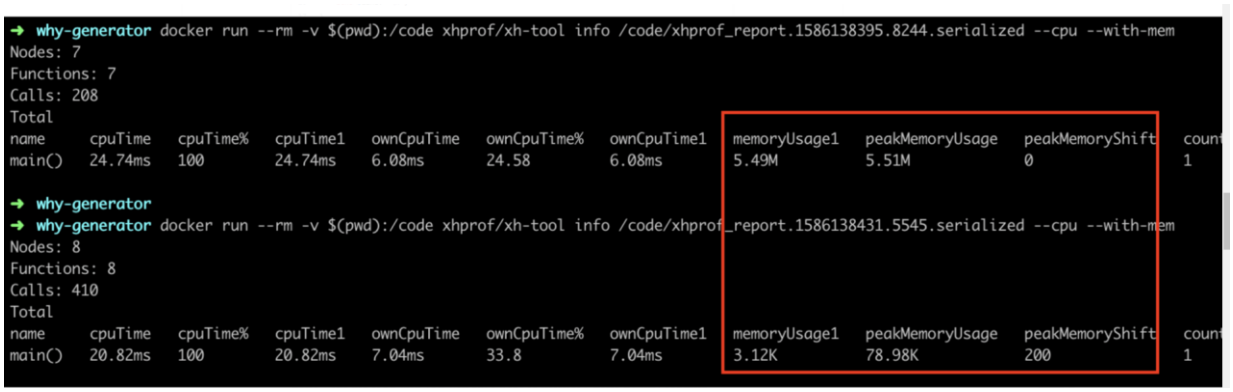
Instead, if you change the above code to support Iterator, the iterator may load images lazily during iteration, which avoids loading all the images to memory at once.

So what is a Generator?

Generator is the easiest way to generate lazy loading iterators. Generators expose a `yield` keyword which allows an easy way to create lazy loading iterators. The first code example of an Iterator can be converted to a generator quite easily, requiring much less code.

Benchmarking

Let's do some profiling of the above code using a generator vs the following equivalent code using an array.



```
➜ why-generator docker run --rm -v $(pwd):/code xhprof/xh-tool info /code/xhprof_report.1586138395.8244.serialized --cpu --with-mem
Nodes: 7
Functions: 7
Calls: 208
Total
name     cpuTime   cpuTime%   cpuTime1   ownCpuTime   ownCpuTime%   ownCpuTime1   memoryUsage1   peakMemoryUsage   peakMemoryShift   count
main()   24.74ms   100        24.74ms    6.08ms       24.58         6.08ms        5.49M          5.51M             0                 1

➜ why-generator
➜ why-generator docker run --rm -v $(pwd):/code xhprof/xh-tool info /code/xhprof_report.1586138431.5545.serialized --cpu --with-mem
Nodes: 8
Functions: 8
Calls: 410
Total
name     cpuTime   cpuTime%   cpuTime1   ownCpuTime   ownCpuTime%   ownCpuTime1   memoryUsage1   peakMemoryUsage   peakMemoryShift   count
main()   20.82ms   100        20.82ms    7.04ms       33.8          7.04ms        3.12K          78.98K            200               1
```

The first one is the report of the above PHP script which uses an array and the second one is the report of the php script which uses a generator. We want to focus on the memory side of the profiling which is marked by the red box and the results are astounding. The script using generator beats the other script using an array by a huge margin.

Early Break

With Generators, you may choose to simply break from the loop after you are done with the loop and you will save a lot of memory done by lazy loading data.

```php
<?php
$lines = getLines('/path-to-file');
foreach ($lines as $line) {
    if (SOME CONDITION) {
      break;
    }
    echo $line. "\n";
}
```

Conclusion

Generators are an under-used concept that provides so many benefits and I think it should be used more often wherever applicable

Pourquoi et quand utiliser un générateur à la place d'un tableau en PHP

Un générateur est un concept TRÈS puissant qui offre une foule d'avantages. Je pense qu'ils sont très sous-utilisés et sous-appréciés par les développeurs PHP. Dans cet article, je vais expliquer quels sont les avantages des générateurs et quelques exemples d'utilisation qui, je l'espère, vous motiveront à utiliser de plus en plus de générateurs en PHP.

Tout d'abord, qu'est-ce qu'un Iterator en PHP ?

Un générateur est un type spécial d'Iterator qui fournit un moyen facile d'implémenter des Iterators en PHP pour de nombreux cas d'utilisation. Donc, nous devons comprendre ce qu'est un Iterator avant d'apprendre ce qu'est un générateur.

Les tableaux ne sont pas les seules choses sur lesquelles vous pouvez itérer en utilisant foreach en PHP. Toute instance d'une classe implémentant l'interface Iterator peut être itérée en utilisant foreach.

Voici un exemple simple d'un itérateur de fichiers. Il reçoit simplement le chemin d'un fichier comme argument et vous pouvez ensuite itérer sur l'objet en utilisant foreach.

Pourquoi un itérateur plutôt qu'un tableau ?

Supposons qu'une fonction attende une liste de contenus d'images pour les télécharger vers un endroit tel qu'un seau AWS S3.

```php
<?php

function uploadImages(iterable $images) {
    foreach ($images as $image) {
        // code to upload image to somewhere
    }
}
```

Le problème avec le code ci-dessus est que vous chargez toutes les images en mémoire avant même d'avoir besoin de l'une d'entre elles.

Au lieu de cela, si vous modifiez le code ci-dessus pour prendre en charge l'itérateur, l'itérateur peut charger les images paresseusement pendant l'itération, ce qui évite de charger toutes les images en mémoire en une seule fois.

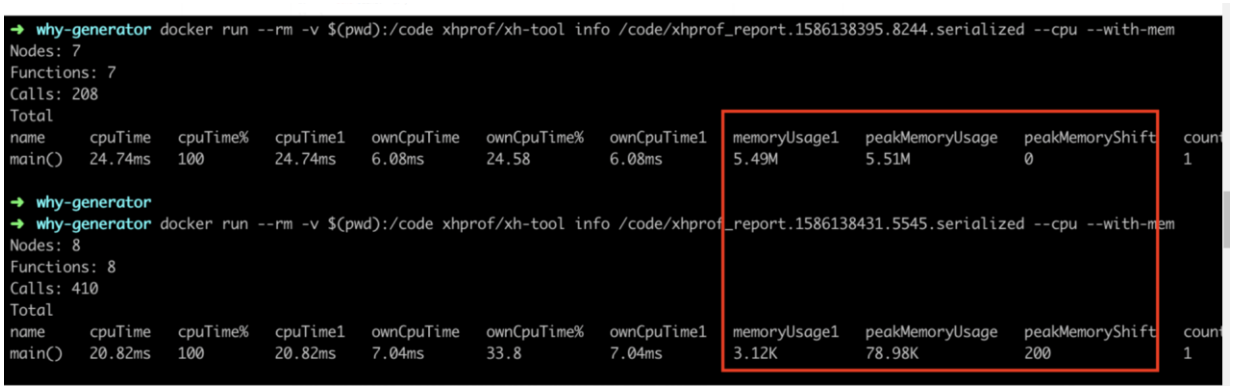
Alors, qu'est-ce qu'un générateur ?

Un générateur est le moyen le plus simple de générer des itérateurs à chargement paresseux. Les générateurs exposent un mot-clé yield qui permet de créer facilement

des itérateurs à chargement paresseux. Le premier exemple de code d'un itérateur peut être converti en un générateur assez facilement, nécessitant beaucoup moins de code.

Analyse comparative

Effectuons un profilage du code ci-dessus utilisant un générateur par rapport au code équivalent suivant utilisant un tableau.



Le premier est le rapport du script PHP ci-dessus qui utilise un tableau et le second est le rapport du script php qui utilise un générateur. Nous voulons nous concentrer sur l'aspect mémoire du profilage qui est marqué par l'encadré rouge et les résultats sont stupéfiants. Le script utilisant un générateur bat l'autre script utilisant un tableau par une marge énorme.

Une pause auparavant

Avec les générateurs, vous pouvez choisir de rompre simplement la boucle après l'avoir terminée et vous économiserez beaucoup de mémoire en chargeant les données paresseusement.

```php
<?php
$lines = getLines('/path-to-file');
foreach ($lines as $line) {
    if (SOME CONDITION) {
      break;
    }
    echo $line. "\n";
}
```

Conclusion :

Les générateurs sont un concept sous utilisé qui offrent de nombreux avantages et je pense qu'ils devraient être utilisés plus souvent lorsqu'ils sont applicables.

Il nous reste à présent à coder notre vue listePhotos.php et à y afficher la totalité des photos.

Tout d'abord, il nous faut récupérer nos données et les mettre dans une variable, ici $photo :

```php
<?php

    $photos = $result["data"]["photos"];

?>

<h3 class="messageErreur"><?= App\Session::getFlash("error") ?></h3>
<h3 class="messageSucces"><?= App\Session::getFlash("success") ?></h3>

<h3 class="titreResponsiveNavigation">liste photos</h3>

```

Puis, étant donné que $photo est un generator, il nous faut faire un foreach afin de passer sur chaque objet. Nous appellerons ensuite les méthodes de cet objet, plus particulièrement les getters afin d'afficher les informations souhaitées.

```php
<div class="content">
    <?php foreach ($photos as $photo) { ?>


        <div class="photoFrame">
            <?php
                if ($photo->getUser() == true) {
                    echo "<p>".$photo->getUser()->getPseudonym()."</p>";

                } else {
                    echo "Compte supprimé";
                }
            ?>

            <figure>
                <a href="index.php?ctrl=photo&action=seePhotoById&id=<?= $photo->getId() ?>&page=1">
                    <img src="public/image/photos/<?= $photo->getFileName() ?>" alt="">
                </a>
                <figcaption><?= $photo->getReleaseDate() ?></figcaption>
            </figure>

            <?php
                if ($photo->getDescription() != null) {
                    echo "<p>".$photo->getDescription()."</p>";

                } else {
                    echo "";
                }

            ?>
        </div>

    <?php }; ?>
</div>
```

En se basant sur cet exemple, nous pouvons afficher la totalité de notre base de données et n'importe quel enregistrement souhaité.
Le fonctionnement est toujours le même.

Le projet possède de nombreuses particularités mais il serait trop long de toutes les détailler. Pour en citer quelques une :

- Il est possible de commenter une photo, ceci est uniquement disponible si un utilisateur est en session, c'est-à-dire connecté. L'auteur pourra commenter la photo, modifier son message jusqu'à 24 heures après l'avoir posté et supprimer son message.
  Si jamais l'utilisateur modifie son message c'est la date de modification qui sera affichée.

- Il est possible de suivre un autre utilisateur afin de voir ses photos postées. Un utilisateur peut aussi nous suivre.
  La liste des utilisateurs que nous suivons est affichée sur notre profil.

- Il est possible de s'inscrire sur le site et de se connecter. Lors de la connexion l'utilisateur est mis en session.
  Si jamais l'utilisateur a cliqué sur la case « mémoriser » lors de sa connexion, son adresse mail sera mémorisée dans le formulaire de connexion.
  Il pourra à tout moment supprimer la totalité des cookies liées au site sur son profil (seuls son id et son adresse mail sont présents dans les cookies).

  Un cookie est un petit fichier qui peut contenir une quantité limitée d'informations. Limitée car c'est un très petit fichier.
  Le but d'un cookie est de faciliter la navigation côté utilisateur afin de pouvoir recevoir des données ciblées.
  Il faut éviter de stocker des données sensibles dans la session ou dans les cookies (comme les mots de passes, le nom/prénom d'utilisateur etc…).
  On se servira de set_cookie pour donner une valeur a un cookie, sinon il est aussi possible d'utiliser la superglobale $_COOKIE.
  Un autre exemple d'utilisation des cookies serait par exemple la Géolocalisation.

- Un utilisateur peut supprimer son profil. Le suivi d'autres utilisateurs sera totalement supprimé mais les commentaires et les photos postées par cet utilisateur continuerons d'apparaître sur le site avec comme utilisateur : « compte supprimé ».
- Si jamais un utilisateur supprime une photo ou modifie son image de profil, nous supprimerons du dossier public les photos.

– Il est évidemment possible d'ajouter une photo (pas plus de 7mo) en sélectionnant la ville dans laquelle a été prise la photo. Il est possible d'ajouter une description, celle-ci est en option et non obligatoire.

7 Sécurité

La sécurité est une partie importante, si ce n'est la partie la plus importante.
Il nous faut gérer plusieurs failles connues à ce jour afin de s'en prémunir.

7.1 La faille XSS

La faille XSS ou XML se nomme : le cross-site scripting. C'est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page. (En général du JavaScript)

Un filter input permet de se prémunir de ce type de faille, cette fonction récupère une variable externe et la filtre.
Une autre manière est d'utiliser la fonction htmlspecialchars, elle convertit les caractères spéciaux en entités HTML.
On peut encore utiliser la fonction htmlentities qui est identique à la fonction htmlspecialchars, excepté que tous les caractères qui ont des équivalents en entités HTML sont aussi traduits.

Nous pouvons voir ci-dessous que nous sommes bien protégés contre la faille XSS,

```
$idTown = filter_input(INPUT_POST, 'ville', FILTER_SANITIZE_NUMBER_INT);
// On pourrait aussi écrire :
// $idTown = htmlspecialchars("$_POST['ville']");
$description = filter_input(INPUT_POST, 'description', FILTER_SANITIZE_FULL_SPECIAL_CHARS);
```

Nous filtrons de la même manière chaque variable que l'utilisateur entre sur notre site.

7.2 La faille SQL Injection

La faille SQLi, abréviation de SQL Injection, soit injection SQL en français, est un groupe de méthodes d'exploitation de faille de sécurité d'une application interagissant avec une base de données.
La solution consiste à utiliser des requêtes préparées : dans ce cas, une compilation de la requête est réalisée avant d'y insérer les paramètres et de l'exécuter, ce qui empêche un éventuel code inséré dans les paramètres d'être interprété.
Il y a trois étapes, étape de préparation, compilation et l'exécution.
Cela permet d'exécuter un très grand nombre de requêtes ou plusieurs fois la même requête avec un meilleur rendement.

Étape de préparation : On va créer un schéma de requête sans y préciser les valeurs réelles (exemple : nom : prénom)

Étape de compilation : quand on va remplacer la préparation par les vraies valeurs. Elle peut optimiser le Template et stocker le résultat sans exécuter la requête.

Étape d'exécution : Lier des valeurs aux marqueurs et la base de données va exécuter la requête.

On peut voir ci-dessous que la totalité de nos requêtes sont préparées, puis compilées et exécutées.

```php
public static function delete($sql, $params){
    try{
        $stmt = self::$bdd->prepare($sql);

        //on renvoie l'état du statement après exécution (true ou false)
        return $stmt->execute($params);

    }
```

Chacune des méthodes du CRUD présentes dans DAO est preparée puis éxécutée.

7.3 La vulnérabilité CSRF

CSRF pour Cross-Site Request Forgery, en français : falsification de requête inter-site.
L'objectif de l'attaque est de forcer un utilisateur authentifié sur un site ou une application web à exécuter des actions spécifiques à son insu.

La solution est d'initialiser un token sur la session de l'utilisateur. Puis, à chaque formulaire, il faudra penser à mettre un deuxièe token dans un champ input hidden du formulaire.
Il faudra par la suite dans notre contrôleur, vérifier si les deux valeurs des token correspondent.

Un token devra posséder une valeur hachée.

```
// si pas de token, initialisation d'un token temporaire hors connexion
if (!Session::getTokenCSRF()){
    Session::setTokenCSRF(bin2hex(random_bytes(32)));
}
```

7.4 Hacher les mots de passe

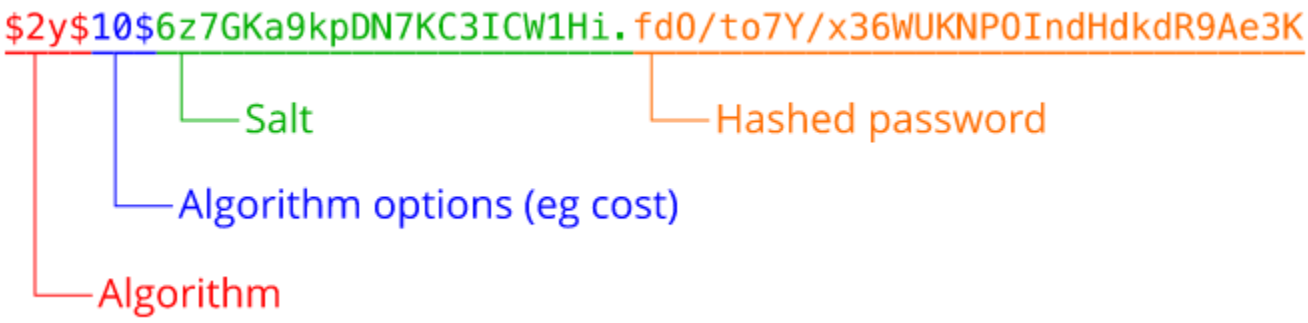
Il est important de hacher les mots de passe, c'est même l'une des premières choses à faire.
Ceci dans le cas où un utilisateur mal intentionné accèderait à notre base de données.
Nous suivons les recommandations de la CNIL et d'owasp pour la sécurisation du mot de passe.
On utilisera ainsi la fonction password_hash qui va demander deux arguments. On utilise PASSWORD_DEFAUT qui prend l'algorithme le plus optimal du moment soit becrypt.

```
$passwordHash = password_hash($password, PASSWORD_DEFAULT);
```

Becrypt est un algorithme fort.
Un algorithme fort crée un salt (grain de sel), un salt est une donnée supplémentaire qui renforce significativement la puissance du hachage de notre mot passe.

C'est très utile car cela rend le mot de passe beaucoup plus difficile à cracker. De plus, cela nous prémunis contre la possibilité d'attaques par dictionnaires (les hachages sont enregistrés dans une longue liste à partir de mots du dictionnaire puis comparés).

Quand on hache avec becrypt on va stocker une emprunte numérique qui se présente sous la forme :



Il faudra encore penser à stocker les mots de passe dans un varchar de 255 caractères (le maximum possible) afin d'avoir assez de place pour stocker en base de données les mots de passe hachés.
Il est à noter qu'un mot de passe ne peut pas être « déhacher », il nous faudra donc hacher avec le même algorithme le mot de passe que l'utilisateur entrera pour se connecter, puis vérifier si le mot de passe donné par l'utilisateur concorde avec le mot de passe en base de données.
Heureusement toutes ces étapes ne sont pas nécessaires, PHP propose déjà une fonction faisant cela : password_verify

```
if(password_verify($password, $userOne->current()->getPassword())){
```

7.5 Attaque par dictionnaire

Comme expliqué plus haut, l'attaque par dictionnaire consiste à prendre tous les mots d'un dictionnaire et les tester en tant que mots de passe.
Pour s'en prémunir, on forcera l'utilisateur à utiliser une expression régulière (Regex) lors de la création de son mot de passe. On fera de même si l'utilisateur souhaite modifier son mot de passe.

Une expression régulière (regex) est une chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles.

Comme nous pouvons le voir ci-dessous nous avons suivi les recommandations de la CNIL en demandant au moins 12 caractères, 1 majuscule, 1 minuscule, 1 chiffre et 1 caractère spécial et maximum 64 caractères.

```
$password = filter_var($passwordSanitize, FILTER_VALIDATE_REGEXP,
    array(
       "options" => array("regexp" => '/^\S*(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=\S*[\W])[a-zA-Z\d]{9,}\S*$/')
    )
);
```

Ou encore,

```
if($pseudo && (preg_match('`^([a-zA-Z0-9-_]{3,36})$`', $pseudo))) {
```

7.6 Attaque par force brute

L'attaque par force brute consiste à tester des millions de combinaisons de mots de passe sur un même utilisateur jusqu'à trouver le bon mot de passe.
Aujourd'hui, les ordinateurs sont de plus en plus puissants et les algorithmes de plus en plus rapides. Il est donc important de se prémunir contre ce type de faille.

Pour cela nous utiliserons un Captcha, plus particulièrement un ReCaptcha qui n'est autre que la version du Captcha de Google.
Un captcha permet de différencier de manière automatisée un utilisateur humain d'un ordinateur.

Nous avons simplement suivi la documentation Google afin de le mettre en place : https://developers.google.com/recaptcha/docs/v3
Ici la V3 est utilisée car plus agréable pour l'utilisateur, étant donné qu'elle ne demande pas d'actions particulières à l'utilisateur.

Nous avons commencé par créer deux clés en localhost. Nous les mettons en tant que constante dans notre index.

```
define('SITE_KEY', '6LcUNg8jAAAAAGkdM5zN7esjkQ5eEcCrxhHP_0pX');
define('SECRET_KEY', '6LcUNg8jAAAAAOCE6yyehxUKgz7W8WIjJF5NGBz_');
```

Il nous faut à présent joindre tous les scripts demandés par google. Nous les plaçons avant la fermeture de notre balise <body>.

Dans un souci de factorisation du code, nous appellerons les scripts du recatpcha uniquement dans les pages qui le requièrent.

```
<?php

    $title = "modifier profil";
    $recaptcha = "formUploadUser";

?>
```

formUploadUser est l'id de notre formulaire (demandé par google).

```php
if(isset($recaptcha)){ ?>
    <script src="https://www.google.com/recaptcha/api.js" async defer></script>
    <script>
        function onSubmit(token) {
            document.getElementById("<?=$recaptcha?>").submit();
        }
    </script>
```

Il nous faut à présent placer le bouton du formulaire demandé par google.

```php
<button class="g-recaptcha boutonForm"
        data-sitekey="<?php echo SITE_KEY ?>"
        data-callback='onSubmit'
        data-action='submit'>
        Modifier votre profil
</button>
```

A partir d'ici, la vérification est faite. Lors de l'envoi d'un formulaire, notre clé de site sera envoyée à google, puis google ira vérifier (via la vitesse des cliques et de nombreux paramètres) si nous étions ou non un robot.

Cependant, il faut encore faire une vérification avec la clé secrète. Il faut vérifier si la réponse donnée est la bonne.

Google nous renvoie un token qu'il faudra concaténer avec notre clé secrète.

Toujours dans un souci d'optimisation du code, nous avons factorisé la fonction checkToken afin de ne pas avoir à la réécrire à chaque fois.

```php
//? recaptcha V3

public static function checkToken($token, $secretKey){

    $verif_url  = "https://www.google.com/recaptcha/api/siteverify?secret=$secretKey&response=$token";
    $curl = curl_init($verif_url);
    curl_setopt($curl, CURLOPT_HEADER, false);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    // curl_setopt($curl, CURLOPT_TIMEOUT, 1);
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);
    $verif_response = curl_exec($curl);
    if ( empty($verif_response) ) return false;
    else {
        $json = json_decode($verif_response);
        return $json->success;
    }
}
```

La fonction checkToken nous renverra un tableau json qu'il suffira de décoder. Ce qui nous intéresse particulièrement ici, est de voir si la réponse est true (si l'utilisateur est bien un humain et non un robot).

Il nous suffit à présent de placer la fonction checkToken à l'endroit souhaité.
Cette fonction nous demandera de renseigner deux paramètres : la réponse de google (le token) et notre clé secrète.
Elle renverra true si Google a déterminé que l'utilisateur était un humain et false dans le cas contraire.

```php
if(Session::checkToken($_POST['g-recaptcha-response'], SECRET_KEY)){
```

Il est difficile de savoir à 100% si cela a bien fonctionné. C'est pourquoi nous avons créé une fonction checkTokenDebug nous permettant de constater les résultats envoyés par Google.

```php
public static function checkTokenDebug($token,$secret_key,$DEBUG=false) {

    if ( $DEBUG ) {
        echo "<b>token =></b> $token<br><br>";
        echo "<b>clé secrète =></b> $secret_key<br><br>";
    }

    if ( empty($token) ) return false;

    $verif_url  = "https://www.google.com/recaptcha/api/siteverify?secret=$secret_key&response=$token";

    if ( function_exists('curl_version') ) {

        if ( $DEBUG ) echo "<b>Curl sur url =></b> ",htmlspecialchars($verif_url),"<br><br>";
        $curl = curl_init($verif_url);
        curl_setopt($curl, CURLOPT_HEADER, false);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_TIMEOUT, 1);
        curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);
        $verif_response = curl_exec($curl);

    } else {

        if ( $DEBUG ) echo "On fait du file_get_content<br><br>";

        $verif_response = file_get_content($verif_url);
    }

    if ( $DEBUG )  echo "<b>Réponse =></b> $verif_response<br><br>";

    if ( empty($verif_response) ) return false;

    else  {
        $json = json_decode($verif_response);
        return $json->success;
    }
}
```

Il nous faut encore placer notre fonction checkTokenDebug dans le contrôleur et créer les conditions.

```php
if(Session::checkTokenDebug($_POST['g-recaptcha-response'], SECRET_KEY, true)){
        echo "oui";
        var_dump($_POST['g-recaptcha-response']);
        die;
    } else {
        echo "non";
        var_dump($_POST['g-recaptcha-response']);
        die;
    }
```

Une fois sur le navigateur, on peut constater que notre ReCaptcha fonctionne correctement.

Ici nous pouvons voir une réponse success positive,
Le score de 0.9 signifie 90% de chance que ce soit un être humain.

Réponse => { "success": true, "challenge_ts": "2022-12-05T08:55:44Z", "hostname": "localhost", "score": 0.9, "action": "submit" }

oui

C:\laragon\www\projet-photo-network\controller\SecurityController.php:73:string '03AEkXODDgjXjhxicELtpIlnA7OViIQgyy1z5WBuUhFUK0ZVYIxc

Ici nous pouvons voir une réponse négative :

Réponse => { "success": false, "error-codes": ["timeout-or-duplicate"] }

non

C:\laragon\www\projet-photo-network\controller\SecurityController.php:77:string '03AEkXODDgjXjh

On peut, à présent, remplacer notre fonction checkTokenDebug par checkToken et mettre cette condition au bon emplacement dans le contrôleur à la réception du formulaire sans avoir à se soucier d'une mal fonction de notre ReCaptcha.

7.7 Faille Upload

De nombreux sites Web permettent aux utilisateurs de télécharger des fichiers tels que des vidéos, des CV, des photos etc...
Jusqu'ici il ne s'agit pas réellement d'une faille de sécurité, c'est le fait de ne pas contrôler ce que le client charge sur le serveur qui constitue une vulnérabilité très dangereuse.

Le principe de l'attaque est très simple. Un pirate essaie de télécharger un fichier contenant du code malveillant ou du code PHP qu'il a créé. S'il y a un défaut, le fichier se retrouvera sur le serveur. Le pirate aura à ce moment juste à appeler son fichier afin de l'exécuter.

Une telle attaque peut avoir de graves conséquences, comme :

- L'espionnage des fichiers et dossiers du site.
- L'accès aux fichiers systèmes et fichiers confidentiels.
- La destruction ou l'altération de données existantes sur le serveur.
- La prise de contrôle du serveur.

Afin de nous prémunir contre la faille upload, il faudra limiter les extensions des fichiers autorisées.

Nous pouvons voir à la prochaine page (page 76) que nous nous sommes bien prémunis contre la faille upload :

```php
if(isset($_FILES['photo'])){
    $tmpName = $_FILES['photo']['tmp_name'];
    $name = $_FILES['photo']['name'];
    $size = $_FILES['photo']['size'];
    $error = $_FILES['photo']['error'];


    $tabExtension = explode('.', $name);
    // explode() -> Scinde une chaîne de caractères en segments
    // Retourne un tableau de chaînes de caractères créées en scindant la chaîne du paramètre string
    // en plusieurs morceaux suivant le paramètre separator.
    $extension = strtolower(end($tabExtension));


    $extensions = ['jpg', 'png', 'jpeg', 'gif'];
    // 7 000 000 octet Soit 7 000 Ko Soit 7 Mo
    $maxSize = 7000000;

    if($size <= $maxSize){

        if(in_array($extension, $extensions) && $error == 0){
            $uniqueName = uniqid('', true);
            // uniqid() : Génère un identifiant unique, préfixé, basé sur la date et heure courante en microsecondes.
            // uniqid() produira une valeur comme : 4b340550242239.64159797.

            $file = $uniqueName.".".$extension;
            // $file = 4b340550242239.64159797.jpg

            move_uploaded_file($tmpName, './public/image/photos/'.$file);

            $photo = $file;

        } else {
            Session::addFlash("error", "Une erreur est survenue lors du téléchargement de l'image <br> Veuillez réessayer")
            $this->redirectTo("photo", "remoteAddPhoto");
        }
```

Il existe encore de nombreux types d'attaques, comme copier le HTML et le CSS de notre site afin de créer un site ressemblant fortement au notre mais avec des intentions malveillantes.
Owasp qui est l'un des sites référence en matière de sécurité propose son top 10 : https://www.httpcs.com/fr/top-10-owasp

8 Mise en forme du projet

8.1 Conception d'interface ergonomique

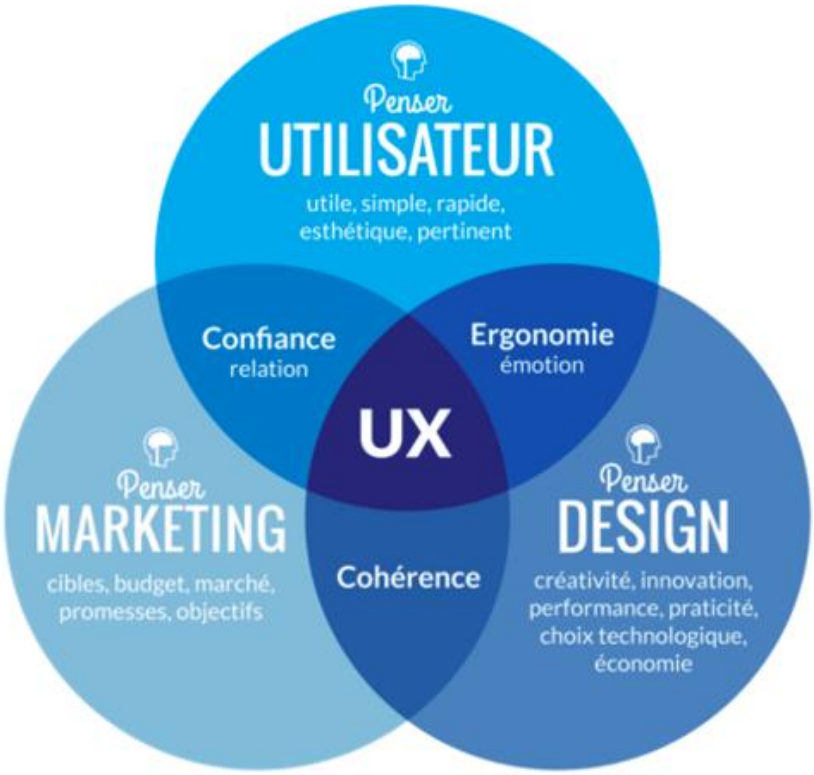
Il est important que la navigation sur notre site soit agréable pour l'utilisateur, il est ainsi nécessaire d'avoir une bonne ergonomie et une interface accessible.

Deux notions deviennent ainsi essentielles : l'UX et l'UI

Les termes UX et UI signifient respectivement User Experience (expérience utilisateur) et User Interface (interface utilisateur), ils ont en commun une approche centrée sur l'utilisateur.

8.1.1 Expérience utilisateur (UX)

L'UX représente la qualité de l'expérience vécue par l'utilisateur lors de sa visite sur notre site web.

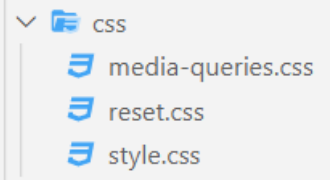


8.1.2 Interface utilisateur (UI)

L'interface utilisateur (UI) concerne l'environnement graphique dans lequel évolue l'utilisateur d'un site web ou d'une application.
L'UI consiste à créer une interface agréable, ergonomique et pratique, facile à prendre en main. Le choix des couleurs, de la typographie fait partie des critères essentiels.

8.2 Mise en forme des vues

Afin de mettre en forme nos vues, nous utiliserons le langage de programmation CSS. Pour cela un dossier css a été créé. Il contient trois fichiers : media-queries.css (pour le responsive), reset.css (pour redéfinir les valeurs par défauts des navigateurs), style.css (pour styliser notre site).



Media-queries.css :
Ce dossier sert à rendre le projet responsive. Cette partie est détaillée dans la section suivante : 8.3 responsive design.

Reset.css :
Avant d'appliquer de nouveaux styles à un site Web, un moyen efficace de recommencer consiste à effectuer une « réinitialisation CSS ». Une réinitialisation CSS efface tous les styles et paramètres CSS d'un site Web. Il renvoie toutes les valeurs à leur état par défaut.

Style.css :
Le fichier style.css ne se lit pas de gauche à droite ni de haut en bas mais d'un seul coup.

La manière la plus idéale d'écrire du code CSS est de sélectionner ses éléments via leurs noms de balises ou si nécessaire leurs donner une classe. On réserve les identifiants pour le JavaScript. Le style JavaScript doit être mis dans des classes actives.
Tout le style doit être dans le même fichier style.css

Le style fonctionne suivant un système de points :
Une balise vaut 1 point.
Un enfant de cette balise vaut 1 point + les points de ses parents.
Une classe (class .) vaut 10 points.
Un identifiant (id #) vaut 100 points.
Du style directement intégré dans une balise HTML vaut 1000 points.
Le sélecteur universel (*) vaut 0 points.
La règle de feuille de style !important vaut 1 000 000 points.

Selector	Specificity Value	Calculation
p	1	1
p.test	11	1 + 10
p#demo	101	1 + 100
<p style="color: pink;">	1000	1000
#demo	100	100
.test	10	10
p.test1.test2	21	1 + 10 + 10
#navbar p#demo	201	100 + 1 + 100
*	0	0 (the universal selector is ignored)

Les blocs ou les éléments que l'on sélectionne avec les balises, classes ou id peuvent s'écrire de plusieurs manières :

Il est possible de sélectionner un élément précisément inclus dans un autre élément Et ceci autant de fois que nécessaire.
C'est très utile lorsque plusieurs balises portent la même classe et que l'on veut en sélectionner juste une seule.

```
.sectionComments .comment .headerComment .modifComment{
    order: 3;
    margin-left: auto;
}
```

Il est possible de sélectionner un élément possédant plusieurs classes. Ceci est utilisé pour les classes actives en JavaScript.

```
.addComment .buttonAddComment.disappear{
    display: none;
}
```

Il est possible d'appliquer le même style à plusieurs éléments, on s'en sert principalement afin d'optimiser le code.

```
.headerPagePrincipal .imagePrincipale h3, .headerPagePrincipal .imagePrincipale span{
    position: relative;
    display: flex;
    justify-content: center;
    top: 104px;
    color: #f0f2f5;
    font-size: 45px;
    text-shadow: 1px 2px 2px #000, 0 0 1em #55595c, 0 0 0.1px #55595c;
    margin: 0;
    letter-spacing: 1px;
    line-height: normal;
    margin-top: 10px;
}
```

8.3 Responsive Design

Un site web responsive design (en français conception réactive) est un site web dont l'affichage s'adapte aux différentes tailles d'écrans.

Ici le responsive a été réalisé via les Media queries car c'est une technique recommandée.
Pas de resize-object ni d'autres technologies. Tout se fait directement dans le CSS.

```
@media only screen and (max-width: 500px){
```

Pour la totalité des boutons et des éléments cliquables, une taille de
38px a été appliquée car Nokia et Microsoft recommandent des boutons d'une taille minimum de 1cm, ce qui donne 37.7953 px, que l'on arrondit à 38px.

Aucune utilisation de hyphens : auto ; ni de word-wrap: break-word ; afin que tout reste très lisible quel que soit la taille de l'affichage.

Le design choisit est celui de base, adapté en une version téléphone comme ce que l'on voit régulièrement sur le net.
La taille minimale d'un mobile semble être 320px.

Tout est basé sur un meta name ='viewport' content= 'width=device-width, initial-scale=1'

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Comme cela, le zoom du responsive se fera parfaitement, suivant la marque du téléphone et suivant les choix du constructeur.

La largeur en « pixels réels » (surface physique) est la résolution de l'écran. Soit le nombre de pixels dans X surface.
Il ne faut pas confondre les « pixels réels » avec les « pixels CSS » (device-width).
Les pixels CSS sont le nombre de pixels virtuels que le terminal pense avoir et sur lequel il fonde son affichage.

Le viewport (fenêtre d'observation en français) utilise les « pixels CSS ».
La taille du viewport ne correspond ni à la taille réelle d'un écran (cm) ni à celle en pixels CSS.

La valeur du viewport est soit déterminée par le navigateur, soit par la valeur donnée dans la balise meta name=viewport.

Les pages Web sont affichées par défaut de sorte que toute la surface tienne sur l'écran. Ce niveau de zoom initial est une division mathématique :

Zoom = pixels CSS (device-width) / viewport

Par exemple :
La résolution de l'iphone 4 est de 640x960px.
La surface en "pixels CSS" de l'iphone 4 est de 320x480px.
La largeur du viewport de Safari mobile est de 980px.

Sans l'utilisation du viewport :
320/980 = 0.3

Le navigateur mobile considère que sa fenêtre a une largeur de 980px.

Avec l'utilisation du viewport :
320/320 = 1
Le navigateur mobile considère que sa fenêtre a une largeur de 320px.

Les tailles des différents supports en pixels CSS sont les suivantes :

- Les téléphones mobiles : 320px jusqu'à 480px
- Les tablettes : 481px jusqu'à 768px
- Les petits écrans : 769px jusqu'à 1024px
- Les écrans classiques : 1025px jusqu'à 1200px
- Les écrans larges : 1021px et plus

9 Axes d'amélioration

Étant donné que la création et l'idée même de ce projet ne datent que du 03 novembre, il y a un nombre conséquent d'améliorations qui pourraient être apportées :

- Finir le système de like.
- Se protéger de la faille CSRF (si possible sans champ hidden afin de ne pas être vu dans l'inspecteur des éléments et sans librairy comme jquery).
- Ajouter les coordonnées des photos et les placer sur une map (en option facultative pour l'utilisateur).
- Améliorer l'administrateur.
- Ajouter un système de messagerie.
- Faire un système afin de vérifier le mail de l'utilisateur.
- Utiliser tarte au citron pour les cookies et aller plus loin avec les cookies.
- Faire la totalité de toutes les tailles responsives du CSS.
- Mettre le site en ligne.
- Améliorer le SEO.