

LAMY
Jean Christophe
jctlamy@gmail.com

GRETA MTE93
Noisy le Grand
93160

GRETAGRAM

2020/2021

GRETAGRAM
SHARE



Scannez-moi !

Table des matières

Remerciements.....	3
Présentation du tuteur, du stagiaire et du projet.....	4
A. LE PROJET.....	6
Cahier des charges.....	6
Présentation du projet.....	6
Github.....	6
Suivi de projet.....	6
Contenu du projet.....	7
Livraison du projet.....	8
Rapport de stage.....	8
Les outils :.....	9
Organisation du travail:	12
* Readme github du projet & Markdown * (Github/Taishi66).....	13
Maquettage du projet.....	14
Diagramme de cas d'utilisation UML.....	14
Maquettes.....	15
Base de données (méthode Merise).....	17
Modèle Conceptuel de Données :.....	17
Modèle Logique de Données :.....	18
B. LE DÉVELOPPEMENT.....	20
Architecture MVC.....	20
Organisation des répertoires :.....	22
Refactorisation.....	23
Principes.....	23
Routeur.....	23
. Routes.....	26
. Render.....	28
Patrons de conception.....	31
. Façade :.....	31
. Singleton :.....	31
. Helper.....	32
Intégration de l'API instagram.....	34
Changements avec Composer.....	35
PHP-CS-FIXER.....	35
DOTENV ou .ENV.....	36
AUTOLOAD.....	36
Htaccess (URL rewriting).....	37
Sécurité mise en place au sein du projet.....	38
Sécurité Côté client:.....	38
Test des ressources tierces :.....	38
Sécurité Base de données:.....	38
Concernant les formulaires :.....	39
Concernant l'inscription et la protection de mot de passe :.....	40
Vérification des données récupérées en GET ou POST :.....	41
C. Aperçu du site	43
Conclusion.....	46

Remerciements

La réalisation de ce rapport a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Je voudrais tout d'abord adresser toute ma reconnaissance à mon tuteur de stage, Guillaume Argiles, pour sa patience, sa disponibilité et surtout ses judicieux conseils avant et après le stage, qui ont contribué à alimenter ma réflexion, et développer mes compétences.

Je désire aussi remercier les formateurs du GMTE93, qui m'ont fourni les outils nécessaires à la réussite de ma formation au titre professionnel de Développeur Web et Web mobile.

Je tiens à remercier spécialement Marc Chenebaux et Esat Bylykbashi, qui ont su, en peu de temps et au travers d'une pédagogie éclairée, nous transmettre les connaissances nécessaires pour entrer avec succès dans le monde du développement.

Je voudrais exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur soutien moral et intellectuel tout au long de ma démarche.

Enfin, je tiens à témoigner toute ma gratitude à mon frère pour m'avoir fait découvrir l'informatique et à mon épouse pour sa patience et sa compréhension ainsi qu'à leur soutien inestimable.

Présentation du tuteur, du stagiaire et du projet.



ARGILES Guillaume / ga-creation.fr (Perpignan)

*Développeur Back-end chez PrestaShop
Freelance spécialisé e-commerce*

Développeur fullstack depuis 8 ans, il s'est spécialisé dans le e-commerce ces 5 dernières années notamment en travaillant chez Octipas et leurs solution de digitalisation de points de ventes .

Il a géré, développé et maintenu les solutions de grands comptes comme Aubade, Nature & Découverte, The Kooples et Carré blanc pour atteindre leur objectifs (augmentation du CA, stocks unifiés, queue boosting etc...)

Il propose également des modules sur mesure qui s'adaptent à vos besoin et à votre architecture (liaison avec votre ERP CRM) et répond aux questions que vous pouvez vous poser.



LAMY Jean Christophe

*Eleve au GRETA MTE 93 de Noisy le Grand |
Formation DWM*

Ex sportif de haut niveau judo de 2005 à 2012, j'ai entamé une carrière de 5 ans dans la restauration, serveur au début, puis barman, brasseur de bière artisanal et enfin Adjoint et Responsable de point de vente.

Je suis actuellement en reconversion professionnelle, afin de pouvoir m'épanouir dans un milieu que j'apprécie tout particulièrement,

guidé par mon frère lui-même développeur depuis 15 ans.



Le projet de départ était d'utiliser l'API instagram afin d'afficher l'ensemble des données (post / commentaires / likes / follow, etc...) d'un compte sur notre interface retravaillée pour ce projet.

Le fait est que Facebook est très restrictif sur l'utilisation des données d'instagram au niveau de leur api, on a été confronté à une non validation des autorisations de données.

Facebook demande pour chaque type de données souhaitée le résultat de son utilisation, avec urls et contenus, ce qui n'était pas envisageable pour ce type de projet.

Bien que nous ayons créés une page Facebook entreprise et une application Facebook reliée à un compte instagram professionnel, nous n'avons obtenu que le niveau de base des autorisations, que j'ai quand même implémenté.

Le fait d'attendre en permanence des retours de Facebook ayant des délais de réponses assez longs, nous a poussé à trouver une solution alternative pour ce projet de stage de fin de formation. Et c'est ainsi que s'est créé Gretagram. Réseau social de partage de photo, où l'on peut s'inscrire, créer un compte, partager ses photos sous forme de post.

Encore en construction, les fonctionnalités d'abonnement, de hashtag et de messagerie interne doivent être ajoutées dans le futur.

A. LE PROJET



Cahier des charges

Présentation du projet

Github

Nous aurons un répertoire github public (ou privé) dans lequel tu vas travailler
Dans ce dépôt nous aurons une branche master qui sera la branche de production

Chaque sprint sera une pull request sur ce dépôt, que je validerai

Chaque commit et pull request suivra la règle de conventional commit pour avoir une structure du git plus claire

Suivi de projet

Nous allons travailler en kanban de 5 jours

Pour suivre le projet, nous allons utiliser JIRA sur lequel je vais créer des sprints et les tâches

Nous aurons un point journalier sur lequel on va travailler sur ton avancée et tes points de blocage

Ainsi qu'un gros point tous les 10 jours (entre les sprints) pour voir comment se passe le projet de ton côté, ce que tu veux améliorer, ce que tu ne veux plus faire, etc... pour avoir un stage le plus productif pour toi

Contenu du projet

Une base de données mysql contenant

- Les utilisateurs qui se connectent
- Les comptes gretagram associés aux utilisateurs
- Les posts associés aux comptes gretagram
- Les commentaires des posts

Une gestion des urls gérée avec le routeur de symfony installé avec composer pour te permettre de comprendre composer et d'éviter le bordel qu'est un routeur.(cf ressources)

Un login utilisateur qui se connectera soit via email, (soit via son compte facebook ?!)

Une architecture orientée objet de ton choix

Un gestionnaire de requête sql fait maison, pour les select / update

Une classe gérant la connectique de l'api instagram

L'affichage avec bootstrap (ou tailwind) des posts récupérés

La sauvegarde des posts en base de données

La mise à jour pour récupérer les posts depuis l'api instagram

En terme d'écran on aura :

Création d'un compte sur la plateforme

Login d'un compte

Connexion à gretagram si aucun compte n'est lié

Affichage des posts gretagram

Affichage de la grille avec tous les posts / nombre de commentaires / nombre de like

Affichage d'un détail d'un post avec commentaires / likes (et les stats peut être ?!)

Pour aller plus loin :

Liker des commentaires

Liker des photos

Répondre aux commentaires

Poster une photo

Liste des follow et follow back

Déconnecter un utilisateur

Mot de passe oublié

Déconnecter le compte gretagram

Avoir plusieurs comptes gretagram

Follow back unfollow

connecter un utilisateur via facebook

Livraison du projet

Entre chaque sprint, je mettrai le travail sur un environnement personnel pour que tu puisses le tester et faire ta démo

Rapport de stage

le vendredi après midi est consacré à ton rapport de stage, tu pourras le faire au fur et à mesure afin d'avoir ton travail en tête pour le rédiger et éviter de te prendre 2 semaines de rush à la fin du projet

Les outils :



Bootstrap est un framework CSS,

En d'autres termes, et pour le dire très simplement, Bootstrap est un ensemble de fichiers CSS et JavaScript fonctionnant ensemble et qu'on va pouvoir utiliser pour créer des design complexes de manière relativement simple.

Le framework Bootstrap est donc un ensemble de fichiers CSS et JavaScript qui contiennent des règles prédéfinies et qui définissent des composants. Ces ensembles de règles sont enfermés dans des classes et nous n'aurons donc qu'à utiliser les classes qui nous intéressent afin d'appliquer un ensemble de styles à tel ou tel élément HTML.

De plus, Bootstrap utilise également des fichiers JavaScript et notamment des bibliothèques JavaScript externes comme jQuery pour définir des composants entiers comme des barres de navigation, des fenêtres modales, etc. qu'on va pouvoir également directement implémenter.

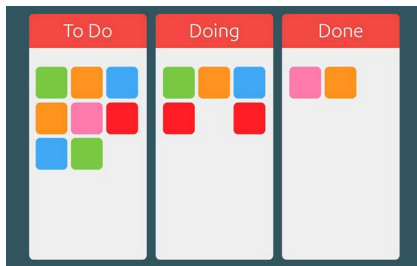


JIRA est un outil de gestion de projet permet de créer et de répartir toutes sortes de tickets (que cela soit une tâche à réaliser

ou un bug à corriger) qui s'afficheront

sur un tableau de bord. Les tâches à réaliser, celles en cours ainsi que les tâches à achever sont facilement identifiables par l'utilisateur. Chacun peut apporter sa contribution en ajoutant des annotations, ce qui favorise la transmission d'informations entre les équipes et évite d'avoir recours à d'autres canaux de communication susceptibles d'être chronophages tels que les emails ou la tenue de réunions.

Outre la création de rapports, Jira offre à ses utilisateurs une certaine liberté pour personifier l'outil à leur convenance. Ainsi, un administrateur Jira peut ajouter un grand nombre d'extensions à la plateforme de façon à avoir accès à des fonctionnalités plus poussées que cela soit pour automatiser des tâches ou modifier l'apparence de l'outil de gestion de projet.



La méthode **Kanban** fait partie des méthodes agiles. Il s'agit d'une méthode de gestion des stocks dont le but principal est de mettre en place une organisation du juste-à-temps, c'est-à-dire de ne produire que ce dont on a besoin à un instant T afin d'éviter les stocks qui coûtent de l'argent.

La méthode Kanban s'inspire de **l'approche Lean**, une méthode de gestion de la production fondée sur l'amélioration continue et qui vise à éliminer les gaspillages afin de rendre l'entreprise plus efficace et performante.



Postman permet de construire et d'exécuter des requêtes HTTP, de les stocker dans un historique afin de pouvoir les rejouer, mais surtout de les organiser en **Collections**. Cette classification permet notamment de regrouper des requêtes de façon « fonctionnelle » (par

exemple enchaînement d'ajout d'item au panier, ou bien un processus d'identification).

Postman assure également la gestion des **Environnements**, qui permet de contextualiser des variables et d'exécuter des requêtes ou des séries de requêtes dans différentes configurations (typiquement : dev, recette, prod). Idéal pour tester une API et ce qu'elle renvoie.



Grâce à une gestion d'erreur centralisée et aux informations de l'utilisateur et de leur problème rencontré

Idéal pour des environnements de productions lorsque la reproduction

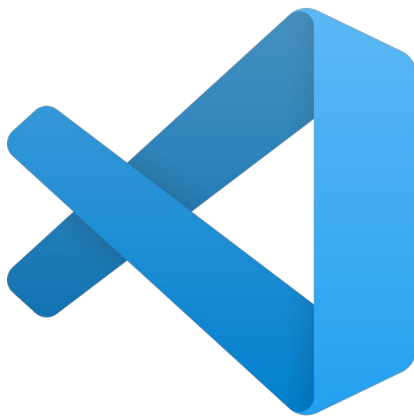
d'erreurs est compliquée.

Des fonctionnalités avancées peuvent également être mises en place afin d'ajouter un ticket Jira à chaque bug pour un meilleur suivi des incidents et de leurs corrections.



COMPOSER est un logiciel gestionnaire de dépendances libres écrit en PHP.

Il permet de gérer les projets facilement et réutiliser des codes existants d'autres développeur en ayant également une gestion de mise à jour pour avoir les derniers fix de sécurité par exemple.



VISUAL STUDIO CODE qui est mon éditeur de code actuel.

Coupler à certains plugins, il permet un développement plus efficace de autocompletion php, liveshare pour le collaboratif etc...



Git est un logiciel de gestion de contrôle de version,

qui peut-être utilisé en respectant le conventionnal commit (par exemple, pour ajouter une fonctionnalité nous écrirons dans le message commit : "feat(addUser):ajouter un nouvel utilisateur" au lieu de mettre « ajout fonction ajouter utilisateur »).

Nous l'utiliserons avec l'interface Github. **(Github/Taishi66)**

Il permet de structurer les commits et les pull requests pour voir en un coup d'œil le développement et extraire facilement le % de fix / feature / refactor que l'on fait.

Organisation du travail:

Réunion de pré-projet en présentiel afin de définir les objectifs de ces prochaines semaines et la fréquence des réunions que nous allons avoir.

Nous travaillons par fréquences de 4H de réunions par semaine réparties le mardi et le vendredi. La réunion se faisait en vision (Discord) avec un liveshare ouvert pour le partage du code.

Dans un premier temps je montrais mon travail, mes avancées et on mettait à jour JIRA.

Ensuite nous regardions ensemble mes points de blocages et nous débloquions les situations.

Enfin nous définissions des axes de refactorisation à traiter ainsi que les prochaines tâches à effectuer selon la durée de cette refactorisation

Le vendredi après midi était consacré à la rédaction de récapitulatifs du travail de la semaine en vue de la rédaction de mon rapport

* Readme github du projet & Markdown * (Github/Taishi66)

☰ README.md



Bienvenue \$visiteur[pseudo] , tu viens d'atterrir sur le site Gretagram 😊!

🔥 Il s'agit en fait d'un clone du véritable instagram que j'ai produit durant mon stage en entreprise, pendant ma formation de développeur web, au GMTE93 🔥



🌸 Peux tu nous parler plus en détail du code?

Alors le site est codé suivant une architecture **MVC**, à 90% en **PHP** pur avec une touche de **Javascript** pour le rendre plus dynamique. Le design est quant à lui réalisé avec **Bootstrap**, le tout avec l'éditeur de code **VSCODE**

👉 Ok et les fonctionnalités dans tout ça?

Bon, pour être honnête, le site est encore en construction, alors il manque pas mal de fonctionnalités... 😞, mais chaque chose en son temps non ?

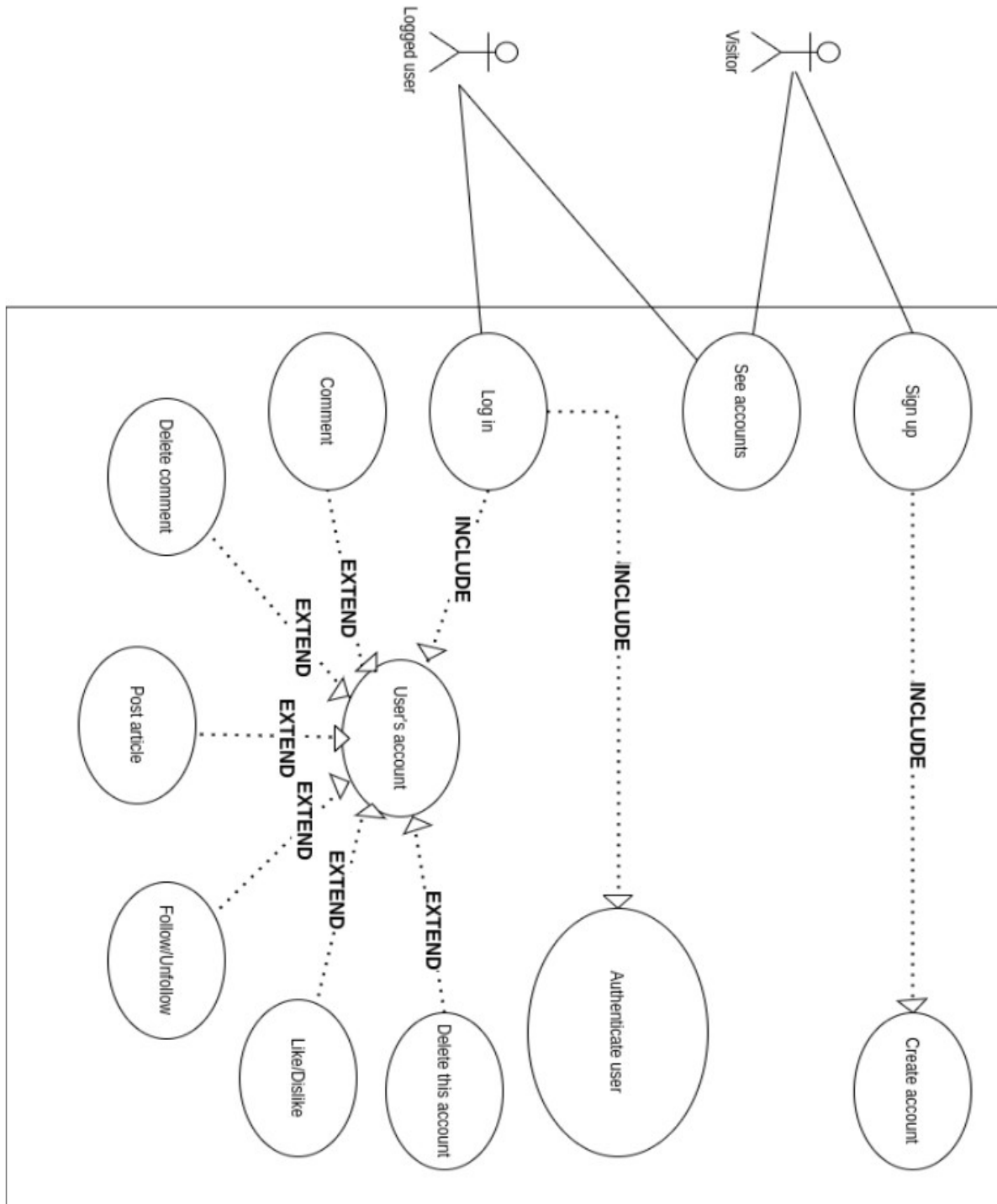
- ☒ Se créer un compte et se connecter
- ☒ Se déconnecter
- ☒ Partager ses photos sous forme de post
- ☒ Liker et commenter ses/les posts
- ☒ Retrouver un profil en particulier avec la barre de recherche
- ☒ Afficher tout les posts sous forme de grille avec le nombre de like et de commentaire
- ☒ Connecter son compte à Instagram grâce à leur API et récupérer ses posts! (mais que pour mon compte hélas 😞)
- ☐ Système de Hashtag
- ☐ Enregistrer les posts qui nous intéressent
- ☐ (un)Follow et (un)follow back
- ☐ Avoir plusieurs comptes **oui** mais interchanger sans passer par le login **non**...



Error: Too many functionalities missing... Go back to work

Maquettage du projet

Diagramme de cas d'utilisation UML



Comme présenté précédemment, ce diagramme représente les fonctionnalités nécessaires pour répondre aux besoins de l'utilisateur.

Il existe principalement deux types de relation :

- Les dépendances stéréotypées, qui sont explicitées par un stéréotype (les plus souvent utilisés sont l'inclusion et l'extension)

Par exemple, la création de compte est inclus avec l'inscription, la connexion exécute par extension une vérification du mot de passe.

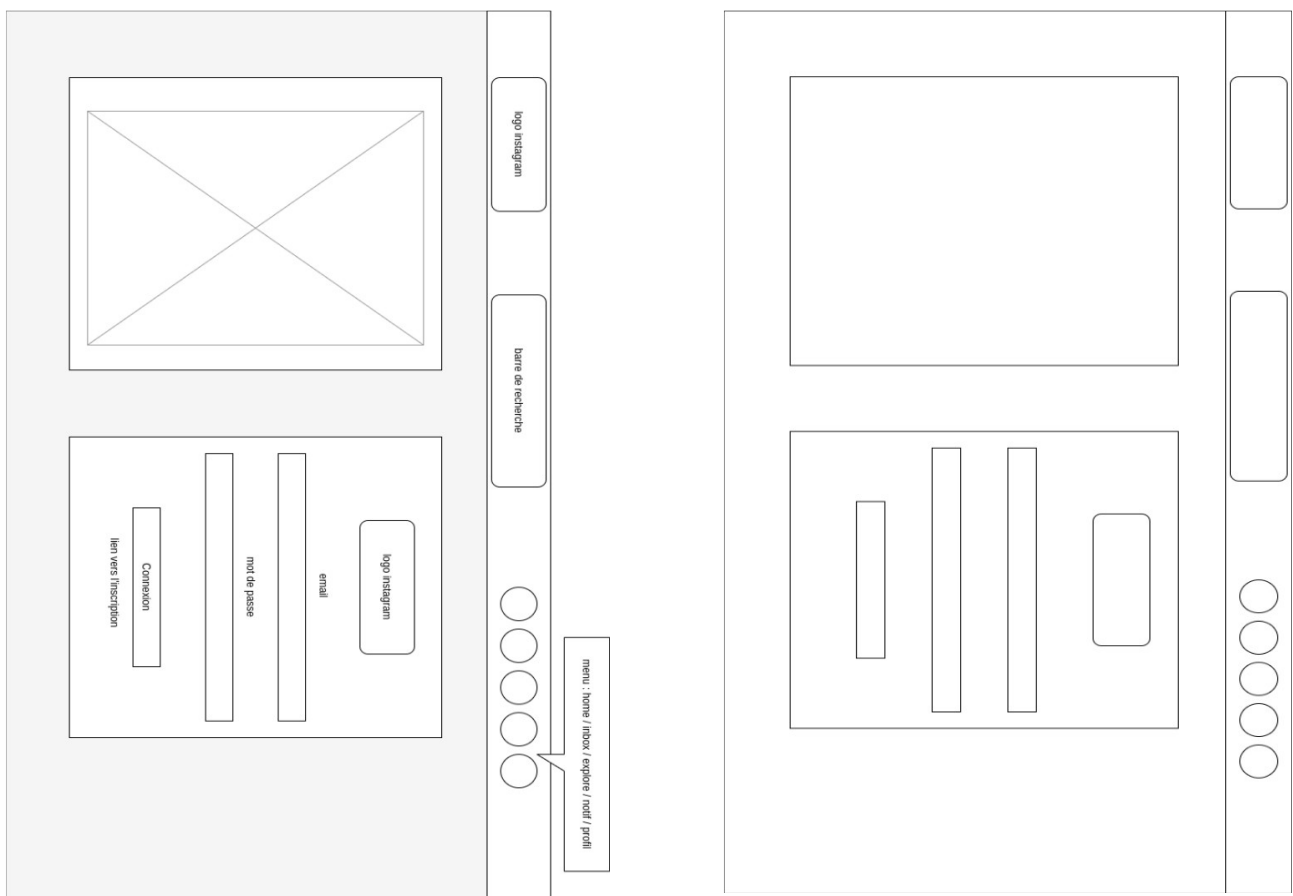
- La généralisation et la spécialisation

Une dépendance se représente par une flèche avec un trait pointillé. Si le cas A étend ou inclut le cas B alors la flèche est dirigée de A vers B.

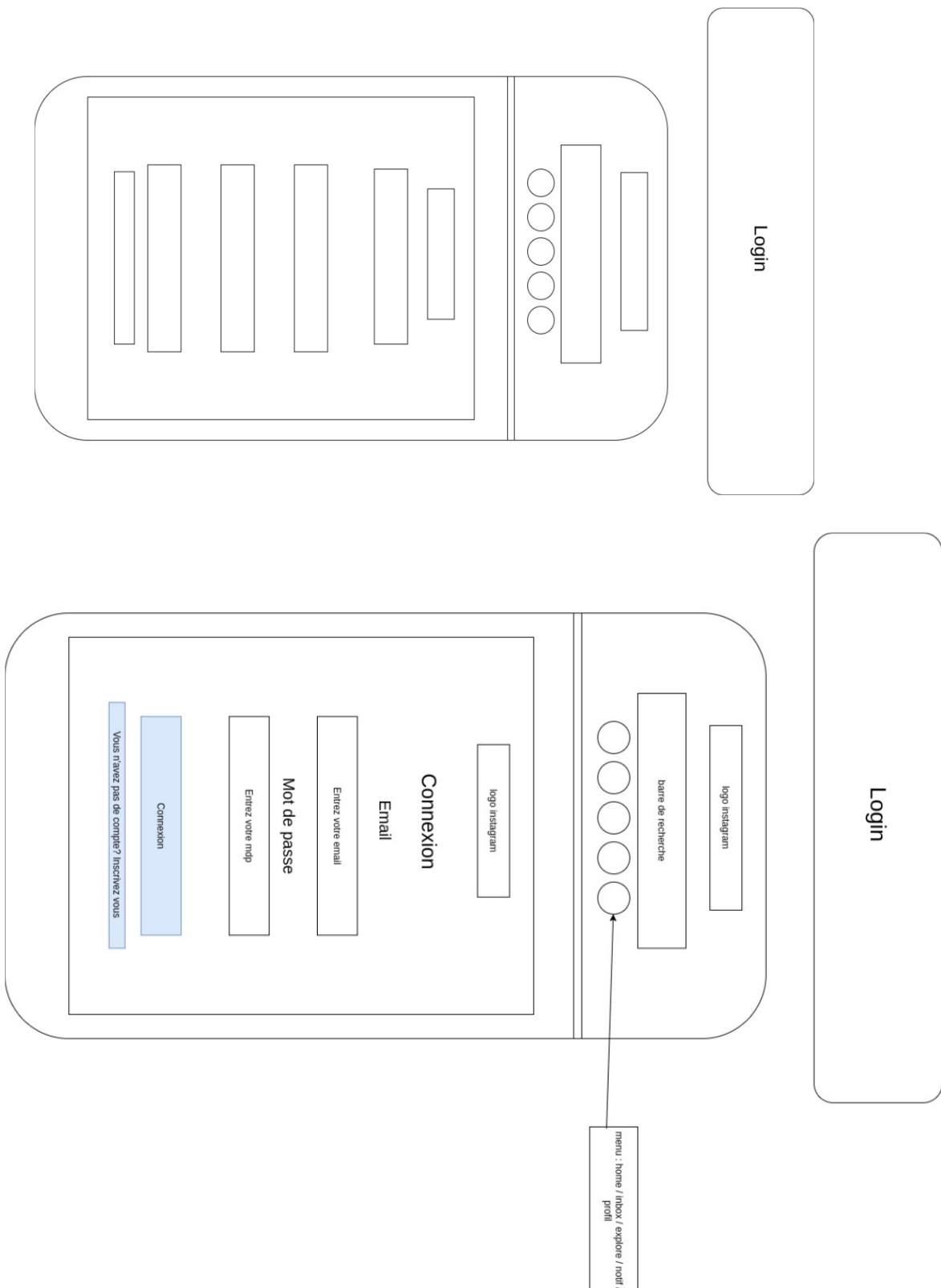
Le symbole utilisé pour la généralisation est une flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.

Maquettes

Exemple de Zoning / Wireframe version Desktop pour la page login

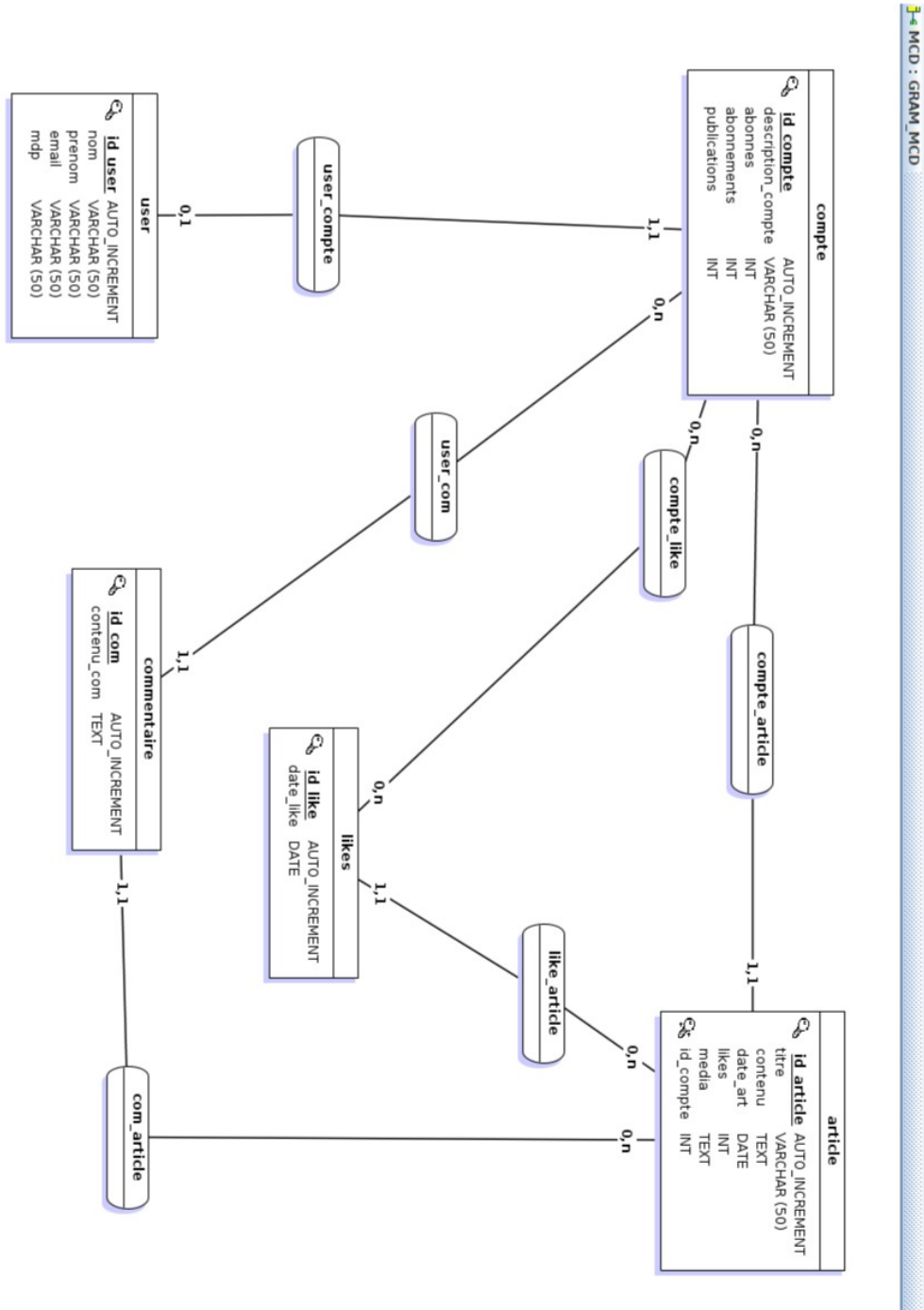


Exemple de Zoning / Wireframe version Mobile pour la page login.php(accueil)

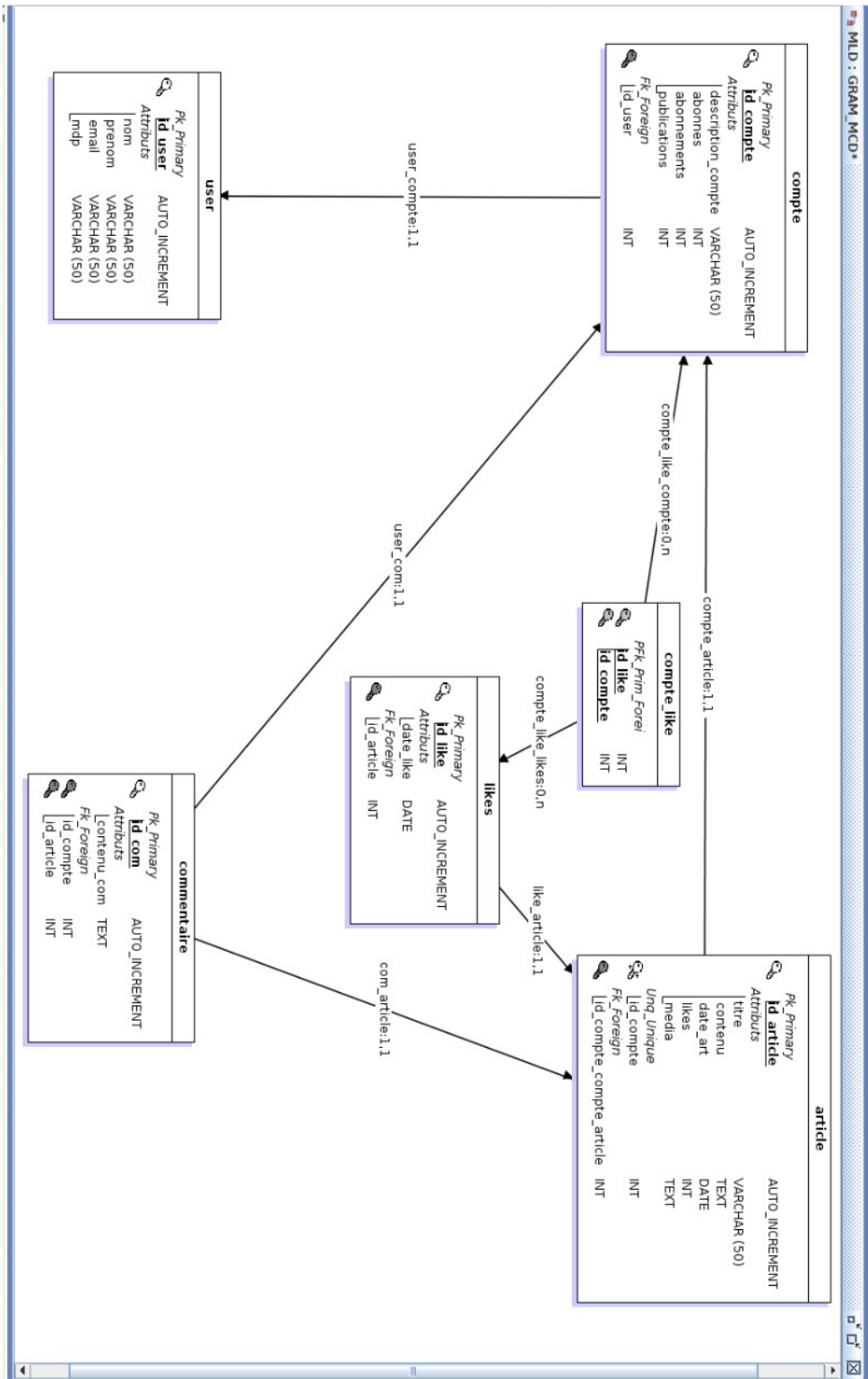


Base de données (méthode Merise)

Modèle Conceptuel de Données :



Modèle Logique de Données :



Script SQL importé en BDD - Exemple des Tables user et compte :

#-----

Table: user

#-----

```
CREATE TABLE user(  
    id_user Int Auto_increment NOT NULL ,  
    nom    Varchar (50) NOT NULL ,  
    prenom Varchar (50) NOT NULL ,  
    email  Varchar (50) NOT NULL ,  
    mdp    Varchar (50) NOT NULL  
    ,CONSTRAINT user_PK PRIMARY KEY (id_user)  
)ENGINE=InnoDB;
```

#-----

Table: compte

#-----

```
CREATE TABLE compte(  
    id_compte      Int Auto_increment NOT NULL ,  
    description_compte Varchar (50) NOT NULL ,  
    abonnées       Int NOT NULL ,  
    abonnements    Int NOT NULL ,  
    publications    Int NOT NULL ,  
    id_user         Int NOT NULL  
    ,CONSTRAINT compte_PK PRIMARY KEY (id_compte)  
  
    ,CONSTRAINT compte_user_FK FOREIGN KEY (id_user) REFERENCES  
user(id_user)  
    ,CONSTRAINT compte_user_AK UNIQUE (id_user)  
)ENGINE=InnoDB;
```

Moteur de stockage INNODB, pour avoir une meilleure gestion des relations (clefs étrangères) entre les tables. (mysql versions 8.0.25)

B. LE DÉVELOPPEMENT

Architecture MVC

Le site est construit selon le modèle d'architecture Modèle Vue Contrôleur, afin d'obtenir une structure professionnelle, c'est à dire :

- Facile à comprendre grâce au rôle unique de chaque fichier
- Une facilité à prendre en main même pour un novice
- Permet une maintenance globale du code plus simple également

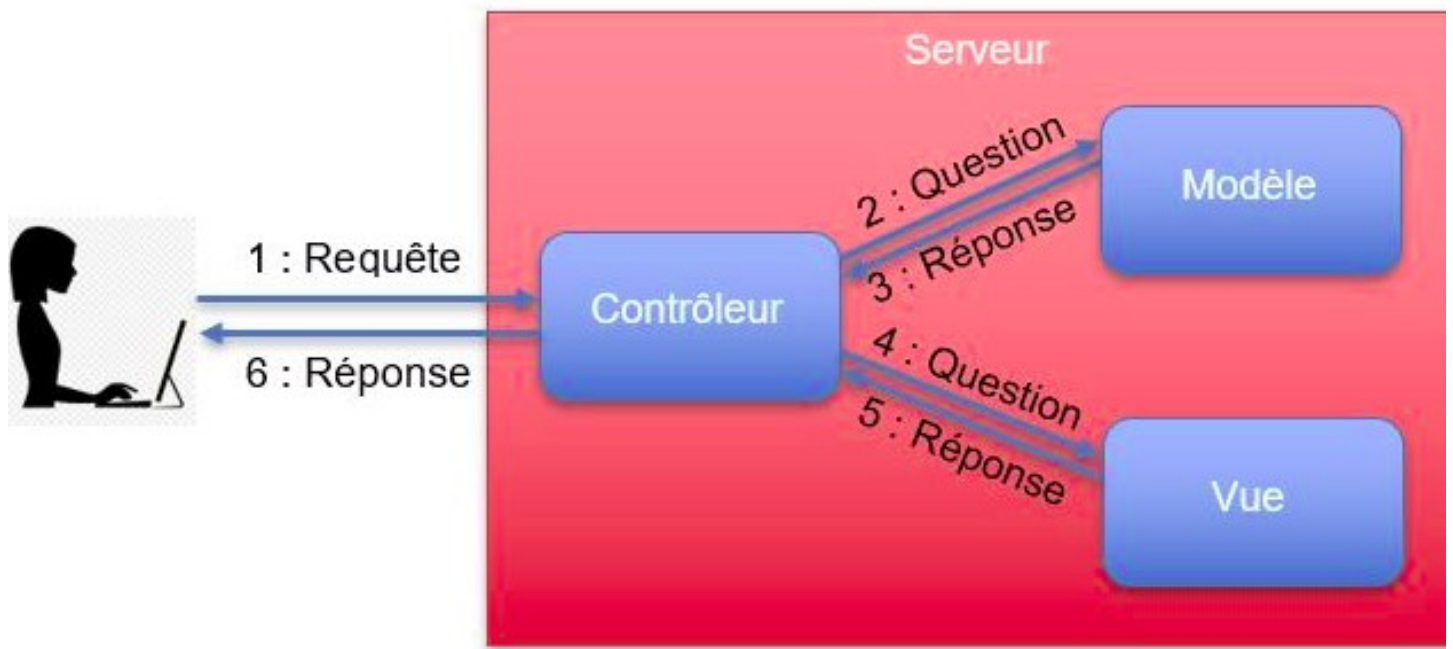
La logique du code est pour cela découpée en 3 parties qui sont :

Modèle : Cette partie gère les données de votre site, son rôle est d'aller récupérer les informations brutes dans la base de données, de les organiser et de les assembler afin qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc principalement les requêtes SQL .

Vue : Cette partie se concentre sur l'affichage. Elle gère l'affichage et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouvera donc essentiellement du code HTML mais aussi quelques boucles ou conditions PHP très simples pour afficher par exemple dans notre cas une liste de posts.

Contrôleur : Cette partie gère la logique du code qui prend des décisions.

C'est en quelques sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Il contient exclusivement du PHP et c'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès.)

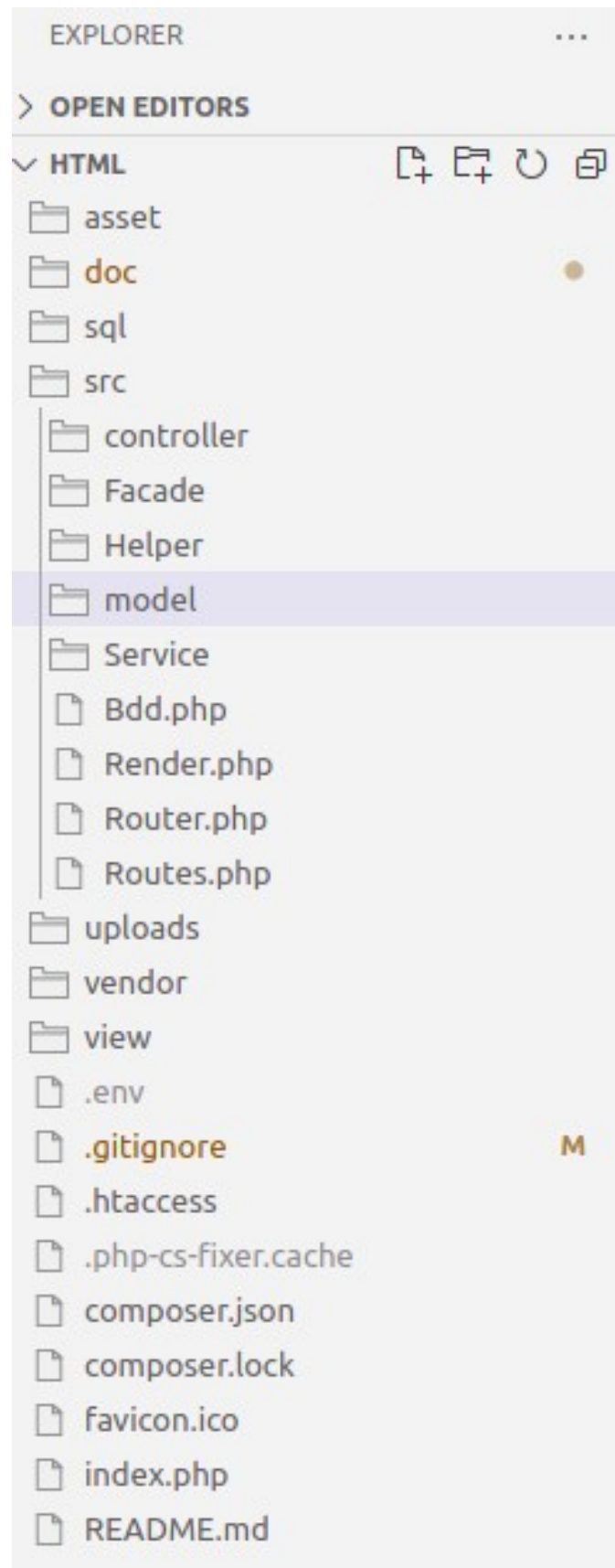


Comme on peut le voir ci-dessus, l'USER envoie sa requête au contrôleur, celui-ci traite les données en utilisant les méthodes du modèle afin d'obtenir les bonnes datas de la BDD.

Une fois les datas récupérées, le contrôleur renvoie donc la vue correspondante accompagnée des datas correspondantes.

(Source : <https://iparnet.fr/le-modele-mvc-nexiste-pas/>)

Organisation des répertoires :



Refactorisation

Principes

60% de mon stage a été consacré à de la refactorisation et de l'optimisation de code.

80% du temps, un développeur lit du code au lieu d'en écrire et cette refactorisation en fait partie. Le principe de refactoriser un code existant permet; en plus de voir l'évolution des compétences du développeur; de repasser sur des morceaux de codes devenus obsolètes et de comprendre pourquoi une solution est mieux qu'une autre.

C'est ainsi que les patrons de conceptions ont été d'une grande efficacité.

Parmi les plus grosse refactorisations, il y a eut le découpage du routeur que je vous détaille ci dessous :

Routeur

Mise en place d'un routeur type framework :

(source: <https://grafikart.fr>)

L'index servira de chef d'orchestre :

```
//Initialise le router
$router = new Router($_GET['url']);

//include routes
$router->includeRoutes();

//lance le routeur pour trouver la route
$router->run();
```

A chaque URL envoyée, il initialisera un nouvel objet Router, ajoutera l'ensemble des routes disponibles avec **includeRoutes()** et lance la recherche de correspondance des routes avec **run()**.

La classe Router se trouve dans le fichier éponyme et permet de :

Récupérer les données **GET** ou **POST** et exécute la méthode correspondante à un des **callable** écrit dans **incudeRoutes()**.

```

public function get($path, $callable, $name = null)
{
    //Récupère les données en GET
    return $this->add($path, $callable, $name, 'GET');
}

//Envoie les données en POST
public function post($path, $callable, $name = null)
{
    return $this->add($path, $callable, $name, 'POST');
}

```

Ces méthodes **GET** et **POST** permettent d'exécuter la méthode correspondante grâce à **add()** qui instancie un nouvel objet de type Route.

```

private function add($path, $callable, $name, $method)
{
    $route = new Route($path, $callable);
    $this->routes[$method][] = $route;
    if (is_string($callable) && $name === null) {
        $name = $callable;
    }
    if ($name) {
        $this->name = $route;
    }
    return $route;
}

```

Une condition vérifie que le **callable** est bien une string et différente(type, nom, valeur) de **null**.

Si oui alors la variable \$name prendra la valeur du callable, et si cette variable \$name existe alors elle deviendra la nouvelle route, que l'on renvoie.

La méthode run() vient elle aussi de la classe Router :

```

public function run()
{
    if (!isset($this->routes[$_SERVER['REQUEST_METHOD']])) {
        error_log('REQUEST_METHOD does not exist');
    }
    foreach ($this->routes[$_SERVER['REQUEST_METHOD']] as $route) {
        if ($route->match($this->url)) {

```



```

        return $route->call();
    }
}
error_log('No matching routes for ' . $this->url);
$this->render->renderErrorNotFound();
}

```

Elle vérifie que la route n'existe pas dans la superglobale \$_SERVER, auquel cas la condition renverra un message d'erreur à la console.

Dans le cas contraire, on parcourra l'array \$_SERVER et on cherchera la correspondance (méthode **match()**) avec les routes présente dans **includeRoutes()**. Si correspondance il y a, alors on appellera la route trouvée avec **call()**.

Dans le cas où la route envoyée n'a pas de correspondance alors **run()** renverra vers une page 404 avec une méthode de la classe **Render**.

```

public function includeRoutes()
{
    //Route login pour se connecter
    $this->get('/', 'User#seConnecter');
    $this->post('/', 'User#seConnecter');
    //Si user non connecté alors retour à la page d'inscription
    $this->get('Inscription', 'User#nouvelleInscription');
    $this->post('Inscription', 'User#nouvelleInscription');
    //Route pour se créer un compte
    $this->get('NoAccount', 'Compte#nouveauCompte');
    $this->post('NoAccount', 'Compte#nouveauCompte');
    //Route accessible si connecté
    if (!empty(SessionFacade::getUserSession())) {
        $this->get('Instagram', 'Instagram#afficherMesMedias');
        $this->get('Profil', 'User#afficherMonprofil');
        $this->post('Profil', 'User#afficherMonprofil');
        //Route pour voir un article
        $this->get('Article', 'Article#afficheArticleController');
        $this->post('Article', 'Article#afficheArticleController');
        //Route pour effacer un commentaire
        $this->get('Delete_com', 'Commentaire#supprimerCommentaire');
        $this->post('Delete_com', 'Commentaire#supprimerCommentaire');
        //Route pour se déconnecter
        $this->get('Deconnexion', 'User#deconnexion');
        //Route pour voir le feed des derniers posts
        $this->get('Home', 'Article#AfficherDerniersArticles');
        $this->post('Home', 'Article#AfficherDerniersArticles');
        //Route pour voir l'ensemble des comptes
        $this->get('Explore', 'Compte#afficherToutLesComptes');
    }
}

```

```

        //Route pour rediriger vers la visite d'un compte
        $this->get('Compte/:id_compte', 'Compte#afficheProfil');
        $this->get('Compte', 'Compte#afficheProfil');
        //Route pour rechercher un profil
        $this->get('Recherche', 'Recherche#recherche');
        //Se déconnecter
        $this->get('Deconnexion', 'User#deconnexion');
    } else {
        $this->get('/', 'User#seConnecter');
        $this->post('/', 'User#seConnecter');
        //Si user non connecté alors retour à la page d'inscription
        $this->get('Inscription', 'User#nouvelleInscription');
        $this->post('Inscription', 'User#nouvelleInscription');
        //Route pour se créer un compte
        $this->get('NoAccount', 'Compte#nouveauCompte');
        $this->post('NoAccount', 'Compte#nouveauCompte');
    }
}

```

Afin d'alléger le code par fichier, la classe Router est allégée par la classe Routes qui contient les méthodes de correspondance et d'appel de fonction.

. Routes

```

public function match($url)
{
    $url = trim($url, '/');
    $path = preg_replace_callback('#:([\w]+)#', [$this, 'paramMatch'], $this->path);
    $regex = "#^$path$#i";
    if (!preg_match($regex, $url, $matches)) {
        return false;
    }
    array_shift($matches); //dégage le premier élément d'un tableau
    $this->matches = $matches; //stock les routes qui matchent dans la variable
    $matches
    return true;
}

```

Cette méthode permet de réécrire l'url envoyée avec **trim()** qui supprimera tous les slash, puis de remplacer la correspondance par une méthode plutôt que par une string avec la méthode **preg_replace_callback()**. Les paramètres sont dans l'ordre :

- une regex

- la méthode paramMatch()

```
private function paramMatch($match)
{
    if (isset($this->params[$match[1]])) {
        return '(' . $this->params[$match[1]] . ')';
    }
    return '([^/]+)';
}
```

Si j'ai un paramètre qui correspond à la variable \$match alors je retourne ce paramètre, sinon je renvoie une regex qui accepte tout sauf un slash.

- **\$this->path** qui est le paramètre passé à la méthode **paramMatch()**.

Je crée une regex qui va contenir toute la chaîne envoyée, « i » me permettant de prendre en compte les minuscules/majuscules.

preg_match() me permettra de voir s'il y a correspondance de la **regex** avec le 2nd param qui est **\$url**, donc **!preg_match()** vérifie l'inverse et renvoie directement un **return false**.

Si correspondance il y a alors **array_shifts()** sortira le premier élément du tableau des routes trouvées et le retournera.(true)

La route trouvée sera appelé et la méthode voulu sera exécutée avec la fonction **call()**

```
public function call()
{
    if (is_string($this->callable)) {
        $params = explode('#', $this->callable);
        $controller = $params[0] . "Controller";
        $controller = new $controller();
        return call_user_func_array([$controller, $params[1]],
            $this->matches);
    } else {
        return call_user_func_array($this->callable, $this->matches);
    }
}
```

Elle récupère le **callable** qu'elle sépare en deux strings en utilisant comme séparateur le « # » définit en paramètre de la fonction **explode()**.

\$params stock la première chaîne de caractère est y concatène la string « Controller », ce qui donne par exemple :

- UserseConnecter devient : UserController'#seConnecter

De cette manière, la nouvelle string **UserController** correspond au **controller** du même nom, on peut donc instancier un nouvel objet avec le mot clef **new**. **call_user_func_array()** va appeler une fonction avec les paramètres passés en **array**, il cherchera d'abord une correspondance dans les méthodes de l'objet rassemblée dans l'array **matches**, avec le deuxième paramètre dont la chaîne de caractère est une string égale au nom de la méthode voulu.

Si le callable n'est pas une string alors il cherchera directement une correspondance du callable dans le tableau matches.

Toutes ces classes servent donc à trouver la bonne route, récupérer les données de la bonne manière en **GET** ou **POST**, puis à exécuter les méthodes nécessaires à la requête utilisateur.

Il nous manque la **VUE**, renvoyé par les **Controllers**, mais traité en amont par le **Render**.

. Render

Le Render s'occupe donc de renvoyer la vue requise, mais aussi d'organiser l'ordre d'apparence de ces dernières.

```
public function renderContent($template, $datas)
{
    ob_start();
    include('view/header.php');
    include('view/' . $template);
    include('view/footer.php');
    $this->content = ob_get_contents();
    ob_end_clean();

    $this->showPage();
}
```

ob_start met les données en tampon à avoir ici dans l'ordre les templates header/ vue renvoyée par le controller / footer.

ob_get_content() affiche toute ces données en tampon, qui sont ici stocké dans la variable \$content.

ob_end_clean vide le tampon.

On exécute la méthode showPage()

```
public function showPage()
{
    echo $this->content;
    exit;
}
```

Méthode qui se contente de faire apparaître sur le navigateur le contenu de la variable \$content et par conséquent tout ce qui était en tampon dans ob_get_content().

```
public function renderErrorNotFound()
{
    ob_start();
    include('view/404.php');
    $this->content = ob_get_contents();
    ob_end_clean();
    $this->showPage();
}
```

Sert uniquement à renvoyer le template d'erreur404.

Voici en comparaison le code de l'ancien Routeur qui était un simple switch/case contenant les objet contrôleurs, la méthode nécessaire pour accéder à la bonne page et parfois le template en include().

* @param PDO \$bdd Objet de connexion à la BDD.

*/

/*function getPage()

```
{
    switch ($this->page) {
        case '':
            $user = new UserController();
            return $user->seConnecter();
            // break;
        case 'login':
            $user = new UserController();
            return $user->seConnecter();
            // break;
        case 'inscription':
            $user = new UserController();
            return $user->nouvelleInscription();
            //break;
        case 'noAccount':
```

```

$user = new UserController();
return $user->nouveauCompte(SessionFacade::getUserId());
// break;
case 'article':
    $article = new ArticleController();
    return $article->afficheArticleController();
//break;
case 'delete_com':
    $commentaire = new CommentaireController();
    return $commentaire->supprimerCommentaire();
//break;
case 'profil':
    $user = new UserController();
    return $user->afficherMonprofil(SessionFacade::getUserId(),
    CompteFacade::getCompteld());
//break;
case 'compte':
    $compte = new compteController();
    return $compte->afficheProfil($_GET['id_compte']);
//break;
case 'home':
    $articles = new ArticleController();
    return $articles->showLastArticles();
//break;
case 'inbox':
    include('view/inbox.php');
    break;
case 'deconnexion':
    // vide la session et donc je me deconnecte
    $_SESSION = array();
    header('Location:');
    break;
case 'recherche':
    $result = new RechercheController();
    return $result->recherche(@$_GET['q']);
//break;
case 'explore':
    $compte = new CompteController();
    return $compte->afficherToutLesComptes();
//break;
default:
    header('location:/home');
    break;
    }
}*/

```

Patrons de conception

(Source : <https://refactoring.guru.>)

. Façade :

Une façade est une classe de setter/getter statiques qui procure une interface simple masquant la complexité du code des classes antérieures.

Dans le cas de **SessionFaçade**, celle ci me permet de récupérer les données de l'utilisateur sans instancier d'objet et en ayant directement accès à la source.

La façade prend tout son sens avec le **CompteFaçade**, qui utilisera des méthodes de **CompteService** qui lui-même utilisera le **compteModel**.
Par exemple :

```
/**
 * @return int $id_compte
 */
public static function getCompteld()
{
    $compteService = new CompteService();
    return $compteService->getCompteFromUser(SessionFacade::getUserId())['id_compte'];
}
```

Ici on instancie un nouvel objet **CompteService** pour utiliser sa méthode **getCompteFromUser()** qui a besoin en paramètre de l'id de l'utilisateur connecté. Du tableau on ne prend que l'**id_compte**.

. Singleton :

Le Singleton permet de n'avoir qu'une seule et unique instance d'une même classe au sein d'un programme, comme par exemple, dans le cas de la connexion à la base de données.

Un singleton possède 3 caractéristiques :

- Un attribut privé et statique qui conservera l'instance unique de la classe.
- Un constructeur privé afin d'empêcher la création d'objet depuis l'extérieur de la classe.
- Une méthode statique qui permet soit d'instancier la classe soit de retourner l'unique instance créée.

```

class Bdd
{
    private static $bdd = null;

    // methode static
    public static function Connexion()
    { //Si n'est pas connecté alors il se connectera
        if (self::$bdd === null) {
            try {
                $bdd = new PDO("mysql:host=" . $_ENV['DB_HOST'] . ";dbname=" .
                    $_ENV['DB_NAME'], $_ENV['DB_DATABASE'], $_ENV['DB_PASSWORD'],
                    array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"));
                self::$bdd = $bdd;
            } catch (Exception $e) {
                die('erreur de connexion à la bdd <br> $e');
            }
        }
        //Si déjà connecté alors il reprendra cette connexion
        return self::$bdd;
    }
}

```

. Helper

Un helper est une classe qui contient des fonctions nécessaires dans de nombreuses parties du code et qui doivent donc être d'accès facile. Il s'agira principalement dans notre cas de la récupération de donnée en get et post avec les formulaires, de la fonction Upload() ou encore du debugger. (debug possède également une façade, comme il est statique il est plus facile à utiliser puisqu'il n'a pas besoin d'être instancié)

Le validatorHelper possède aussi les méthodes de vérifications d'email, de nom et de prénom, quant à sa méthode getValue, elle permet de se protéger face aux injections SQL. (voir chapitre 10 sur la Sécurité)

ValidatorHelper

```
class ValidatorHelper
{
    public function __construct()
    {
    }

    /**
     * Method getValue (sécurise en vérifiant à la fois la valeur, avec une valeur par
     * défaut, et le type //Contre les injections SQL)
     *
     * @param $key
     * @param $default_
     * @param $typeOfValue
     *
     * @return $value
     */
    public function getValue($key, $default_value = null, $typeOfValue = '')
    {
        if (empty($key) || !is_string($key)) {
            return false;
        }

        $value = false;
        if (isset($_POST[$key]) || isset($_GET[$key])) {
            $value = isset($_POST[$key]) ? $_POST[$key] : $_GET[$key];
        }

        if (!isset($value)) {
            $value = $default_value;
        }

        if ($typeOfValue == 'integer') {
            return (int) $value;
        }

        return $value;
    }
}
```

Intégration de l'API instagram

L'intégration de l'api se fera via le logiciel CURL, et l'url testée avec POSTMAN.

(source : <https://php.net>)

```
class InstagramService
```

```
{
```

```
    private $url = 'https://graph.instagram.com/';
```

```
    public function getMedias()
```

```
    {
```

```
        $params = 'me/media?'
```

```
        fields=id,caption,media_type,media_url,username,timestamp'; //posts du compte
```

```
        $access_token = '&access_token=' . $_ENV['T_INSTAGRAM']; //
```

```
        $url = $this->url . $params . $access_token;
```

```
        $curl = curl_init(); //Initialise la session CURL
```

```
        curl_setopt_array($curl, array( //Définit les options de transfert
```

```
            CURLOPT_URL => $url,
```

```
            CURLOPT_RETURNTRANSFER => true, //retourne contenu de l'url sous chaîne  
            de caractère
```

```
            CURLOPT_MAXREDIRS => 10, //permet de rediriger le transfert en cas de  
            changement d'adresse d'API
```

```
            CURLOPT_TIMEOUT => 0, //temps maximale d'execution 0 = pas de timeout
```

```
            CURLOPT_FOLLOWLOCATION => true, //autorise la redirection/nombre max  
            de redirections http à suivre, à utiliser avec followlocation. 20 par défaut, -1  
            pour infinie et 0 pour refuser
```

```
            CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
```

```
            CURLOPT_CUSTOMREQUEST => 'GET' //type de requête
```

```
        ));
```

```
        $response = curl_exec($curl); //Execute la session
```

```
        curl_close($curl); //Met fin à la session
```

```
        return json_decode($response, true); //true pour renvoyer sous forme de tableau  
        associatif
```

```
    }
```

```
}
```

On peut voir que l'URL à interroger est ici découpé en 3 partie, à savoir l'adresse principale <https://graph.instagram.com> , ainsi l'URL paramètre concaténée correspond aux posts du compte et enfin le token d'identification donnée par Facebook valable 2 mois. Access token conservé dans une variable d'environnement du fichier .env pour une meilleure sécurité.

Changements avec Composer

Composer me permettra de grandement optimiser mon code pour une meilleure organisation, et également plus de sécurité.

(source : <https://github> & <https://packagist.org>)

Lors de son installation (**composer init**) un fichier **composer.json** sera créé à la racine du projet. Il s'agit d'un fichier de donnée JSON dans lequel il faudra spécifier les paquets dont le projet dépend (dans la clef 'require'), lors de nouvelles installations de dépendances, en suivant la syntaxe d'un array JSON. Au même moment un dossier /vendor/ apparaîtra, le 'vendeur ' contenant lui les dépendances des paquets installés.

Le nom des paquets vont dans le tableau « require ».

Certain paquets comme l'autoload nécessitent des spécifications (comme ici l'emplacement des fichiers **MODEL** et **CONTROLLER** nécessaires au fonctionnement de l'autoload)

Exemple : {

```
    "require": {
        "friendsofphp/php-cs-fixer": "^3.0",
        "sentry/sdk": "^3.1",
        "vlucas/phpdotenv": "^5.3"
    },
    "autoload": {
        "classmap": ["src"]
    }
}
```

PHP-CS-FIXER

Il me permettra de vérifier la bonne indentation de mon code. Il y a possibilité de préciser le dossier à analyser pour ne pas perdre de temps à repasser sur des fichiers déjà correctement présentés. (utiliser : vendor/bin/php-cs-fixer fix src {nom du répertoire})

Sa configuration peut être plus poussée en fonction des besoins.

DOTENV ou .ENV

C'est un paquet permettant de masquer dans le code les données sensibles tel que les identifiants de connexion à la base de donnée, ou encore les tokens d'authentification comme pour mon compte SENTRY ou INSTAGRAM.

Ces données seront stockées dans des variables d'environnement `$_ENV`, qui sont manuellement assignées dans le fichier `.env` qui lui-même est dans le fichier `.gitignore` afin qu'il ne soit pas inclus dans les commits.

AUTOLOAD

Certainement le paquet le plus pratique puisqu'il s'occupe du chargement automatique des classes dans le code, ce qui nous permet de nous débarrasser de tous les `include()` qui se baladaient au sein du code, notamment dans `l'index.php`.

Le fichier important à retenir est le `classmap`, comme précisé dans le `composer.json`, puisqu'il initialise les chemins des fichiers contenant les classes à partir du dossier indiqué dans `composer.json`. Comme on peut le voir plus haut, le dossier indiqué est `/SRC/`.

Htaccess (URL rewriting)

(source : <https://grafikart.fr>)

.htaccess est un fichier de configuration pour le serveur Apache. Il permet de définir des règles spéciales qui indiquent au serveur comment fonctionner, il se trouve dans le dossier racine.

Dans mon cas je m'en servirai pour réécrire l'url afin qu'elle soit plus user friendly grâce à une regex qui est la suivante :

a) RewriteCond %{REQUEST_FILENAME} !-f

b) RewriteRule ^(.*)\$ index.php?url=\$1 [QSA,L]

a) Va tester si le fichier du même nom n'existe pas (**!-f**) dans le navigateur ou dans le dossier courant, s'il n'existe pas alors il exécutera la règle de réécriture suivante qui signifie :

Le nom de la route envoyée redirigera vers index.php?url=

Le drapeau **[QSA]** transmet à la redirection les paramètres d'URLs. (derrière le point d'interrogation ?truc=machin&bidule=chouette).

« **L** » signifie qu'une fois cette règle exécutée, il s'arrête et n'exécutera pas d'autre règle de réécriture.

Sécurité mise en place au sein du projet

Sécurité Côté client:

Il n'y a à ce jour pas encore de sécurité front mise en place dans le code, c'est une feature encore en cours de développement, mais j'aurai pu :

Implémenter un plugin jquery de type : <https://www.pierrefay.fr/blog/jquery-validate-formulaire-validation-tutoriel.html>

Et également vérifier les champs avec le html 5 de ce type :

<https://pageclip.co/blog/2018-02-20-you-should-use-html5-form-validation.html#html-validation-attributes>

Test des ressources tierces :

Il s'agit là principalement de veille technologique à savoir vérifier la viabilité des ressources externes tel que les APIS qui sont utilisées dans le site (API chuknorris, Pokemon, Instagram).

Sécurité Base de données:

La sécurité du code sera ici beaucoup plus développée, le back-end étant la partie la plus important à protéger.

A commencer par les requêtes SQL des models, qui ne sont pas exécutées en simple **query()** mais en **prepare()** puis **execute()**.

Prepare() : La donnée entrée est d'abord traitée par le serveur (au lieu d'être directement envoyé dans la bdd) qui va l'examiner puis comme son nom l'indique, la préparer avant de l'exécuter. Utile pour se protéger des injections SQL mais aussi lorsqu'on a besoin d'effectuer une requête plusieurs fois, celle ci sera préparée dans sa totalité une unique fois avant de pouvoir être exécutée le nombre de fois voulu.(économie de ressource)

Execute() : Exécute la requête SQL préparé en envoyant dans le serveur la variable (« **\$id_user** ») correspondant au marqueur de la requête (ici : « **:id_user** »).

Exemple :

```
public function getUser($id_user)
{
    $profil = $this->bdd->prepare('SELECT * FROM compte
    INNER JOIN user ON compte.id_user = user.id_user
    INNER JOIN article ON compte.id_compte = article.id_compte
    WHERE compte.id_user=:id_user');
    $profil->execute([":id_user" => $id_user]);
    return $profil->fetch(PDO::FETCH_ASSOC);
}
```

Concernant les formulaires :

Les regex permettent de vérifier ce qui est tapé dans les champs input afin d'éviter toute attaque comme l'envoi de requête SQL pirate dans un formulaire de connexion.

Exemple de regex pour l'input **mail** :

```
public function verfEmail($valeur)
{
    if (preg_match("#^[a-z0-9._-]{2,50}+@[a-z0-9._-]{2,50}\.[a-z]{2,5}$#", $valeur)) {
        return $valeur; // true
    } else {
        return false;
    }
}
```

Regex de protection face aux attaques sur un upload de fichier :

Ici je nettoie le nom du fichier en refusant tout caractères spéciaux comme par exemple les «», ' ', [], () ou bien encore \$.

```
function sanitize_file_name($string, $force_lowercase = true, $anal = false)
{
    $strip = array(
        "~", "`", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "_", "=", "+", "[", "{", "]",
        "}", "\\", "|", ";", ":", "\'", "\"", "&#8216;", "&#8217;", "&#8220;", "&#8221;",
        "&#8211;", "&#8212;",
        "â€", "â€", " ", "<", ".", ">", "/", "?"
    );
}
```

```

    );
    $clean = trim(str_replace($strip, "", strip_tags($string)));
    $clean = preg_replace('/\s+/', "-", $clean);
    $clean = ($anal) ? preg_replace("/[^\a-zA-Z0-9]/", "", $clean) : $clean;
    return ($force_lowercase) ?
    (function_exists('mb_strtolower')) ?
    mb_strtolower($clean, 'UTF-8') :
    strtolower($clean) :
    $clean;
}

```

*J'ai repris ce code de github :

(Source : <https://github.com/vito/chyrp/blob/35c646dda657300b345a233ab10eaca7ccd4ec10/includes/helpers.php#L515>)

Concernant l'upload il est aussi important de tester le **type/mime** du fichier envoyé afin de ne pas se retrouver avec un script caché dans un fichier **.png**

```

public function typeContenuFichier($fichier)
{
    $aTypeMyme = array('image/png', 'image/jpeg', 'image/gif', 'image/jpg');

    /**
     * mime_content_type($fichier) retourne le type de contenu dans le fichier
     * $fichier
     */
    $mime = mime_content_type($fichier);
    return in_array($mime, $aTypeMyme); // "true" ou "false"
}

```

Si la condition retourne **false** alors le traitement sera stoppé et l'image ne sera pas uploadée. L'extension doit aussi être vérifiée.

Concernant l'inscription et la protection de mot de passe :

Le mot de passe envoyé en base de données doit obligatoirement être **haché** selon les règles de la **RGPD**, le cas inverse peut-être passible de **prison**.

Exemple :

```

if (isset($_POST['inscription'])) {
    $mdp = password_hash($this->validatorHelper->getValue("mdp"),
    PASSWORD_DEFAULT);
    //hachage du mdp pour sécuriser en BDD
}

```


Le mot de passe est haché selon un algorithme de hachage fort et irréversible avec la méthode `password_hash()` qui prend en paramètre le mot de passe envoyé et en second paramètre la méthode de hash choisi (donc celle par défaut) qui est la méthode `bcrypt`.

A noter que cette méthode peut évoluer avec les différentes versions de php vers des méthodes plus performantes.

L'avantage de `password_hash()` est qu'il applique le `salt()`, c'est à dire qu'il ajoute une chaine de caractères afin que le mot de passe dans sa totalité atteigne les 60 caractères (important d'avoir une colonne mot de passe dans sa base de donnée en `VARCHAR 255`), ce qui rend le décryptage beaucoup plus difficile.

Plus le mot de passe est long, plus il est dur de le hacker.

Il est important de noter que chaque mot de passe doit avoir un `salt()` différent afin de pouvoir optimiser la protection de ces derniers.

Vérification des données récupérées en GET ou POST :

Le `ValidatorHelper` permet également de gérer de manière centralisé la récupération des données, validation et initialisation des paramètres envoyés.

```
public function getValue($key, $default_value = null, $typeOfValue = '') //protection
injection sql
{
    if (empty($key) || !is_string($key)) {
        return false;
    }

    $value = false;
    if (isset($_POST[$key]) || isset($_GET[$key])) {
        $value = isset($_POST[$key]) ? $_POST[$key] : $_GET[$key];
    }

    if (!isset($value)) {
        $value = $default_value;
    }

    if (is_integer($typeOfValue)) {
        return (int) $value;
    }

    return $value;
}
```

Les paramètres correspondent à :

\$key => donnée \$_GET[''] ou \$_POST[''] envoyée par l'utilisateur

\$default_value => On peut attribuer une valeur par défaut si jamais l'action effectuée par l'user n'aboutit pas. Ce paramètre est idéale pour l'inbox(pas encore créé sur le site)

Par exemple : Si l'utilisateur veut envoyer un message privé à un autre utilisateur, mais que le compte ciblée a été effacé ou simplement s'il y a eu une erreur serveur, alors le paramètre par défaut sera l'adresse mail de l'utilisateur courant avec un message précisant que son dernier message posté n'est pas arrivé à destination pour telle ou telle raison et qu'il lui est donc renvoyé.

\$typeOfValue => précise le type de valeur, notamment sur les integers lorsqu'on veut récupérer une id.

Par exemple :

Si le hacker tape dans l'url :

profil/20/article/5'DROP DATABASE users;'

La donnée récupéré en utilisant \$_GET sera :

\$_GET['5'DROP DATABASE ;']

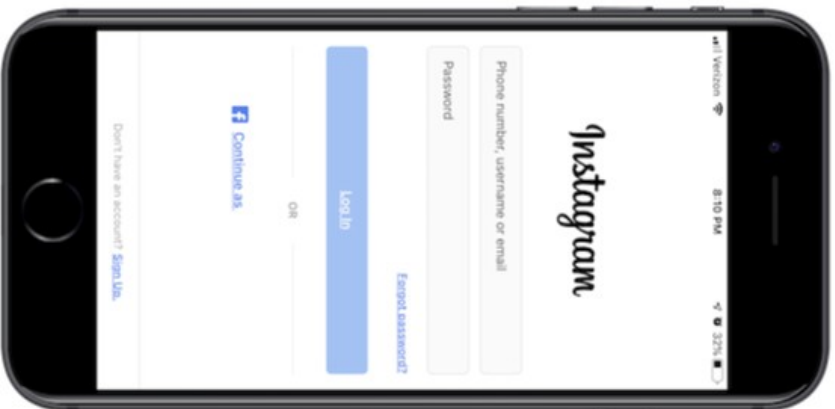
Ce qui est une injection SQL.

Avec le getValue(), la donnée entrée ne sera acceptée que si la variable contient un int et ce grâce à la vérification :

```
if (is_integer($typeOfValue)) {  
    return (int) $value;  
}
```

Si la string est un int alors il renvoie le chiffre en question.

On aurait dû également préfixer les tables de la base de données en donnant des noms moins générique, tel que 'grt_users' ou 'grm_compte', de tel que sorte que s'il y a tentative d'injection, la table 'users' ou 'compte' n'existent pas.



Connexion

E-mail

Entrez votre email

Mot de passe

Entrez votre mdp

Connexion

Vous n'avez pas de compte? [Inscrivez-vous.](#)

C. Aperçu du site






Page de connexion


Page utilisateur connecté

G

GRETAGRAMP
SHARE


Search profil





JC

COMPTÉ



LAMY JC


description du profil popopopopop

Modifier profil

Nouveau Post

Instagram




Déconnexion





14 Publications

Abonnées

Abonnements











GRETAGRAM
SHARE

G


Search profil




When Chuck Norris break the build, you can't fix it, because there is not a single line of code left.




lululul




BienTot...




Les stories




seront




disponibles!




iiiiiiiiiiii



azz






JC COMPTE
LAMY JC


Suggestions pour vous

Voir Tout




edede

Voir le profil



COMPTE TEST JC
21??

Voir le profil



bobbibibi

Voir le profil

À propos Aide Presse API Emplois
Confidentialité Conditions Lieux Comptes
principaux Hashtags Langue Français

© 2021 INSTAGRAM PAR FACEBOOK

45

Conclusion

Ma mission lors de ce stage chez GA-CREATION était de mettre en application les compétences acquises lors de ma formation au GMTE93, de continuer à les développer et d'élargir l'éventail de mes connaissances afin d'être prêt à intégrer le monde professionnel informatique au plus vite.

N'ayant jamais travaillé dans ce milieu auparavant, l'organisation du projet et la rigueur imposée par mon tuteur m'ont permis d'avoir un avant-goût de ce qui m'attends, et surtout de découvrir en temps réel mes points forts et ceux à travailler.

J'ai bien évidemment conscience que le télétravail au vu des récents événements, et la marge d'erreur permise par mon tuteur ne reflète pas tout à fait la réalité, et qu'il faut continuer à se former de manière régulière et autonome, maintenant que les fondations sont posées.

Au cours de ces deux derniers mois, mon stage s'est découpé en 3 axes principaux : la création du site web, la refactorisation/optimisation du code, et enfin la documentation, chacune aussi passionnante les unes que les autres.

La régularité et la curiosité informatique sont été les clefs qui m'ont permis d'aller au bout de chacune de ces étapes, et surtout de capitaliser sur les connaissances obtenus lors des mois précédents.

Le site est temporairement disponible à l'adresse suivante :

<http://gretagram.ga-creation.fr>

Il n'est pas encore terminé et reste en phase de développement, un de mes objectifs étant de réaliser toute les fonctionnalités nécessaires et même d'aller au-delà, afin que ce site reflète au mieux mes compétences aux yeux des recruteurs.