ELAN Formation

Tof’Box

Réseau social de partage de photographie

Sylvain ALLAIN

Developpeur web & web MOBILE

NOVEMBRE 2020 – Juin 2020

SOMMAIRE

[Introduction 3](#_Toc42269570)

[*I.* Résumé du projet 4](#_Toc42269571)

[*II.* Organisation et expression des besoins 0](#_Toc42269572)

[1. Spécifications fonctionnelles 0](#_Toc42269573)

[a. Fonctionnalités 0](#_Toc42269574)

[b. Arborescence 0](#_Toc42269575)

[2. Structuration et gestion de projet 1](#_Toc42269576)

[a. Organisation dans le temps : Diagramme de Gantt. 1](#_Toc42269577)

[b. Organisation dans les tâches : Trello 2](#_Toc42269578)

[3. Spécifications techniques 3](#_Toc42269579)

[a. Outils de gestion de projet 3](#_Toc42269580)

[b. Outils de conceptualisation 4](#_Toc42269581)

[c. Outils de versionning 4](#_Toc42269582)

[d. Outils liés au développement 4](#_Toc42269583)

[e. Langages, bibliothèques et environnement de développements web 5](#_Toc42269584)

[4. Pourquoi Symfony ? 6](#_Toc42269585)

[a. Son utilisation 7](#_Toc42269586)

[b. Le design pattern MVP 7](#_Toc42269587)

[c. La sécurité 12](#_Toc42269588)

[*III.* Réalisation 14](#_Toc42269589)

[1. Conception 14](#_Toc42269590)

[a. Schématisation de la base de données 14](#_Toc42269591)

[b. Maquettage et rendu visuel 15](#_Toc42269592)

[2. Mise en œuvre et cheminement d’une fonctionnalité 18](#_Toc42269593)

[a. Mise en place des Entités 19](#_Toc42269594)

[b. Création de la base de données et migration 19](#_Toc42269595)

[c. Mise en place des Contrôleurs 20](#_Toc42269596)

[3. Cheminement d’une fonctionnalité 21](#_Toc42269597)

[a. La requête Ajax 21](#_Toc42269598)

[b. Le Controller 22](#_Toc42269599)

[c. La réponse 23](#_Toc42269600)

[d. Affichage de la vue 24](#_Toc42269601)

[e. Extension Twig 25](#_Toc42269602)

[4. Traduction de la page en anglais 26](#_Toc42269603)

[a. Extrait de la page en anglais 27](#_Toc42269604)

[b. Traduction de la page 28](#_Toc42269605)

[Axe d’amélioration 31](#_Toc42269606)

[Remerciements 32](#_Toc42269607)

[Conclusion 33](#_Toc42269608)

Introduction

ELAN Formation

Compétences couvertes

* Front End

Cette application n’utilisant pas de CMS, les compétences Front-End du référentiel de formation utilisées sont :

* Le Maquettage d’une application,
* La création d’une interface web statique et adaptable,
* Le développement d’une interface utilisateur dynamique.
* Back End

Cette application n’utilisant pas de CMS, les compétences Back-End du référentiel de formation utilisées sont :

* Création d’une base de données,
* Développer les composants d’accès aux données,
* Développer la partie Back-End d’une application web ou web mobile.

1. Résumé du projet

Mon premier projet en développement web a été de créer un site afin de répondre à la demande d’une personne de mon entourage.

Celle-ci avait un site pour exposer ses photographies en ligne mais certaines fonctionnalités manquaient et l’interface visuelle ne convenait pas à ses envies.

En autoformation dans le développement web (avant la formation actuelle), j’ai proposé mes services car c’était pour moi l’occasion de mettre en pratique mes connaissances avec un cas concret.  
J’ai donc opté pour un CMS (WordPress) et choisis de créer mon propre thème afin de gérer moi-même l’aspect Front-End du site et ainsi répondre aux besoins et envies de la personne.

Pour que celle-ci puisse être autonome par la suite, j’ai utilisé une extension qui permet de rendre du contenu personnalisable (titre, photo d’accueil, actualités …).

Au moment de choisir mon projet de fin de formation, j’ai en premier lieu pensé à reprendre ce projet. L’idée était de récupérer le concept mais en me détachant du CMS, afin de gérer le Back-End, de mettre en place la BDD et une interface administrateur qui puisse permettre tout autant de personnalisation.

Et puis j’ai eu envie de rajouter d’autres fonctionnalités, d’ouvrir ce projet à un plus grand nombre, comme une association de photographes ou un réseau social plus ouvert, c’est comme cela qu’a émergé le projet actuel, que j’ai nommé « Tof’Box ».

Le but du projet Tof’Box est de faire un réseau social de partage de photographies, permettant de voir, partager et interagir avec les photos et avec les autres utilisateurs.

1. Organisation et expression des besoins
   1. Spécifications fonctionnelles
      1. Fonctionnalités

En premier lieu, lors de la réunion de conception, il fallait énumérer les fonctionnalités souhaitées dans le projet. Après avoir définis son cadre, j’ai listé un ensemble de fonctionnalités que j’ai trié par le statut de la personne sur le site :

**Non connecté :**

* Voir des photos en page d'accueil.
* Voir une photo, ses caractéristiques, ses commentaires.
* Voir des photos par catégories.
* Voir le profil d'un user & ses photos.

**Connecté :**

* Accueil personnalisé (photo des utilisateurs que l’on suit).
* Ajouter/Modifier/Supprimer ses photos
* Ajouter/Modifier/Supprimer des commentaires sous une photo
* Pouvoir envoyer un message privé à un autre utilisateur
* Mettre un « j’aime » sur une photo
* Suivre un autre utilisateur
* Voir et pouvoir modifier son profil
* Signaler une photo
* Gestion de l’oublie de mot de passe avec envoie de mail
* Changer de mot de passe / de mail
* Supprimer son compte

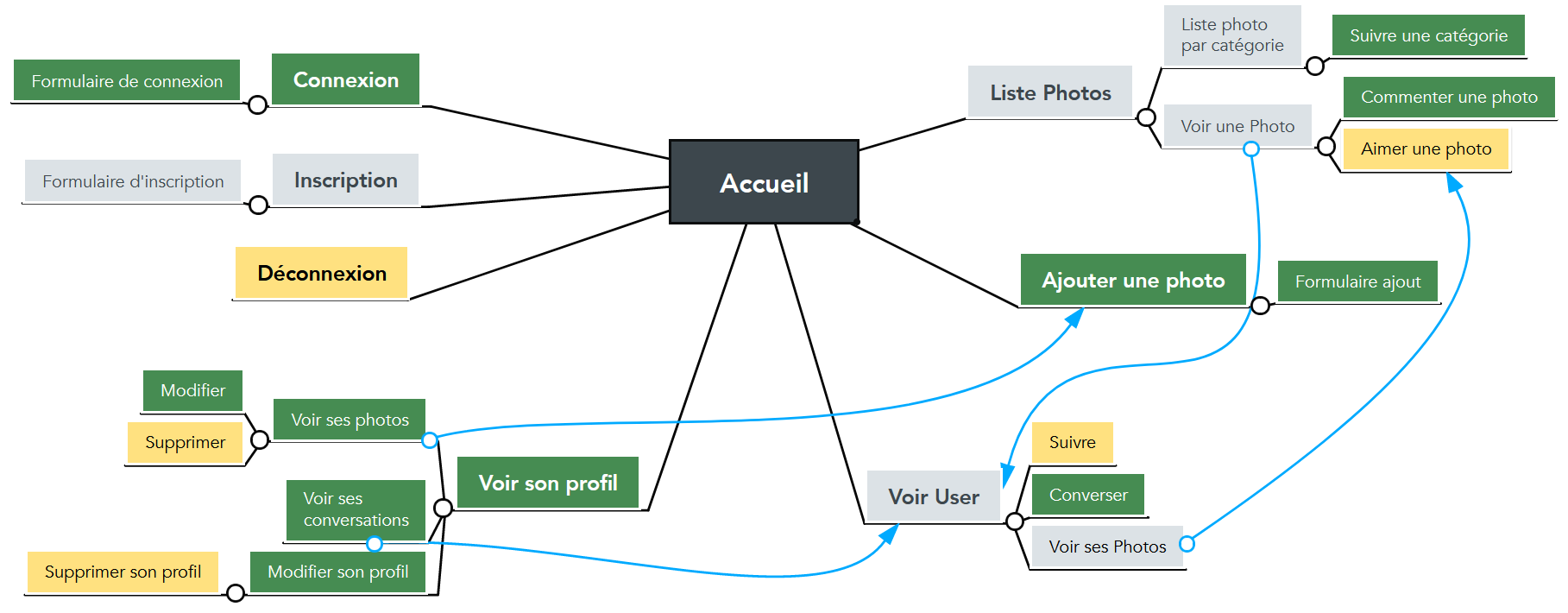
**Admin :**

* Peut faire tout ce que peut faire un utilisateur connecté (sauf changement de mot de passe et d’adresse mail)
* Accès à un espace administrateur avec une vision globale du site.
* Gestion des utilisateurs, des catégories, des signalements.

b. Arborescence

De ces fonctionnalités s’est organisée une arborescence.

L’arborescence représente la manière dont un utilisateur va naviguer entre les pages, les actions.



*Arborescence du projet Tof’Box (voir Annexe 1)*

L’arborescence tourne autour de la page d’accueil.

* *En gris, les pages accessibles à tous.*
* *En vert, les pages accessibles à tous.*
* *En jaune, les actions réalisables depuis ces pages.*
  1. Structuration et gestion de projet

Une fois le projet pensé, il a fallu organiser tout cela dans le temps et dans les tâches.

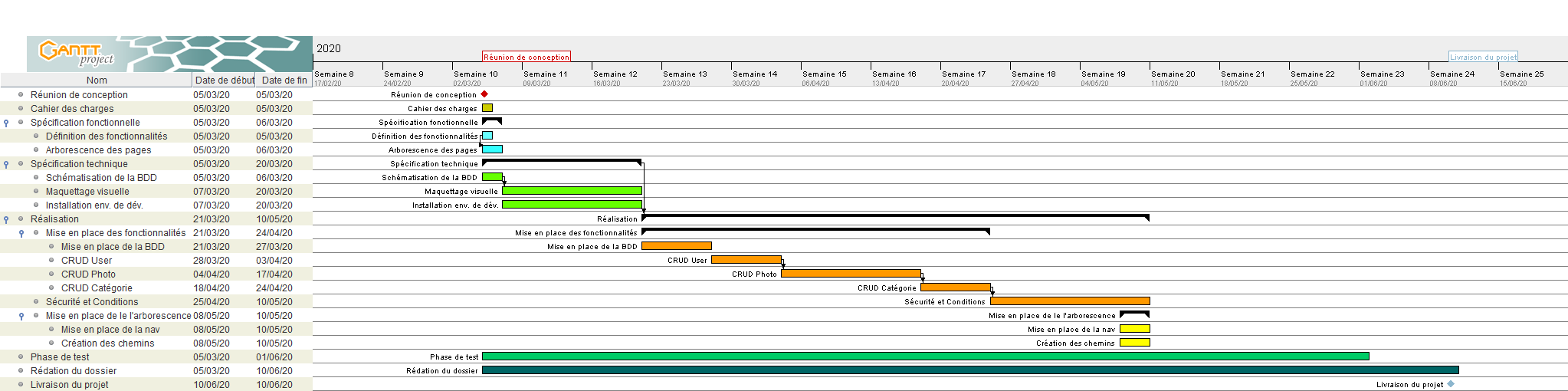
L’écriture et la réalisation du projet allait se passer en parallèle du stage en entreprise, il allait donc falloir me projeter et organiser mon temps, tout en gardant une certaine flexibilité.

Deux outils de gestion de projet m’ont permis d’y parvenir : le diagramme de Gantt et Trello.

* + 1. Organisation dans le temps : Diagramme de Gantt.

Lors de la réunion de conception, un diagramme de Gantt a donc été réalisé à l’aide du logiciel GanttProject.

Le diagramme de Gantt est un outil de gestion de projet permettant de visualiser dans le temps les diverses tâches composant un projet.



*Diagramme de Gantt au jour 1 (voir Annexe 2)*

Ainsi, cet outil m’a permis de structurer les tâches et de me fixer des objectifs.

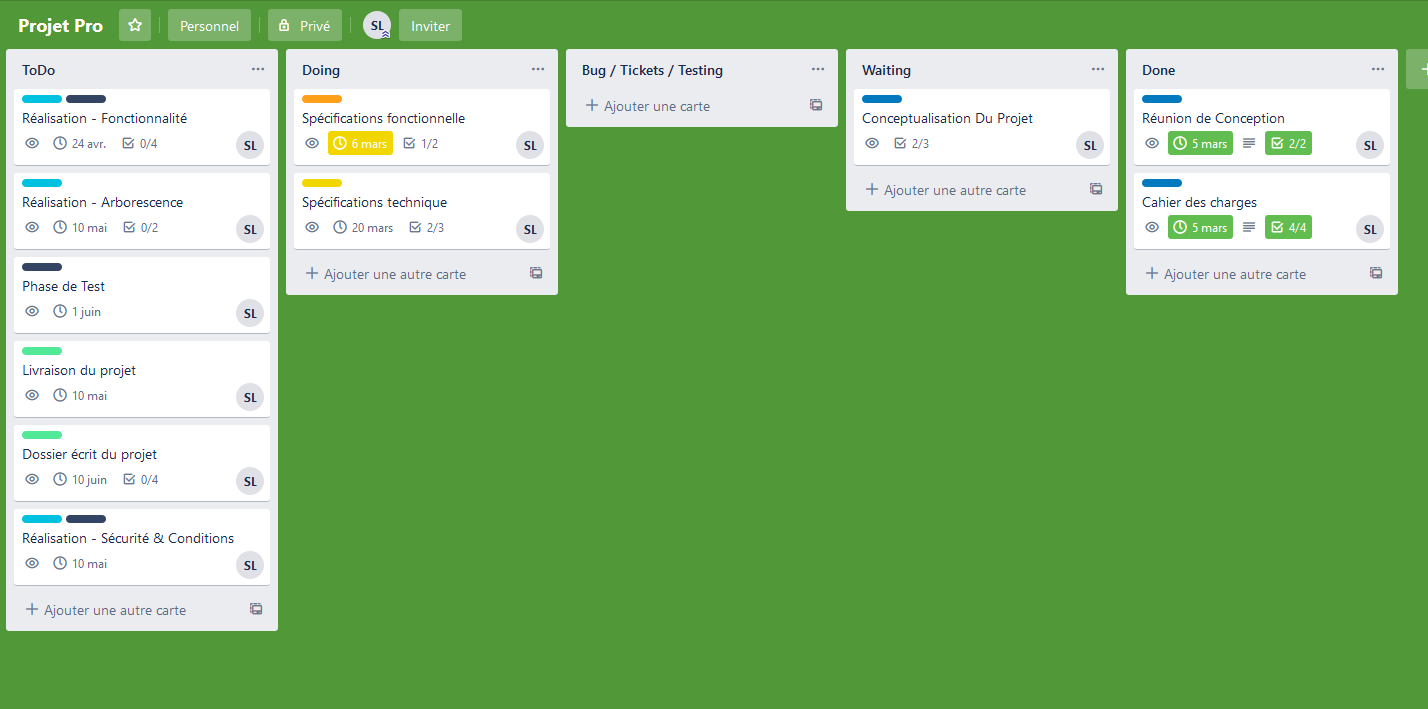
* + 1. Organisation dans les tâches : Trello

Afin de ne pas rester figé sur une vision globale du projet comme le présente le diagramme de Gant, il était important d’avoir un outil plus flexible, où les tâches pouvaient être découpées en sous-tâches et de pouvoir avoir une vision en temps réel de l’avancement du projet.

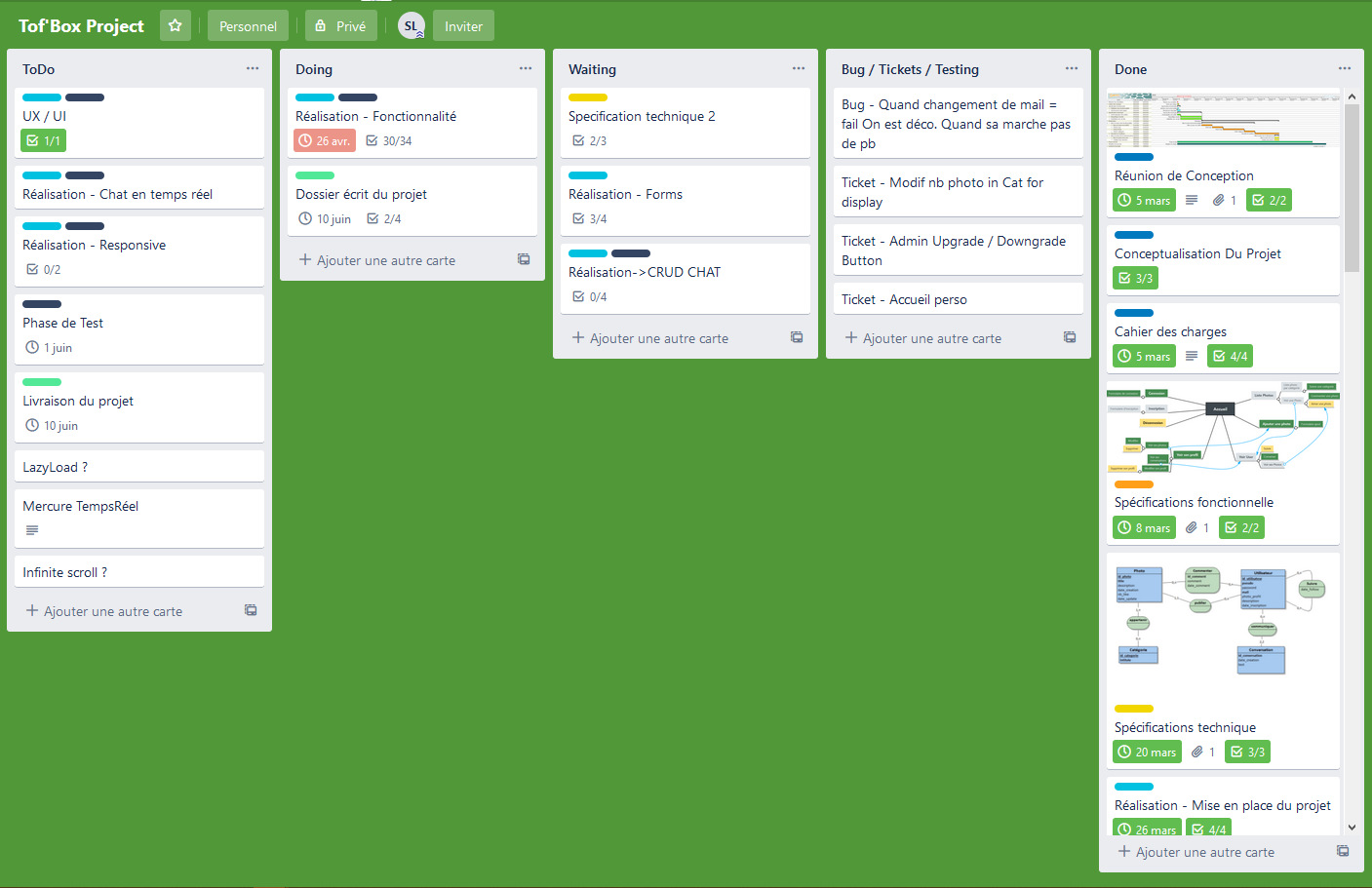
Trello est un outil que j’ai découvert lors d’un stage en agence web.

Inspiré de la méthode Kanban de Toyota, cet outil repose sur une organisation de projet où les tâches sont représentées par des cartes, chaque carte pouvant contenir des check-lists, des commentaires, des dates limite pour réaliser la tâche en question, par exemple.

L’avantage de cet outil est sa flexibilité : organiser entre 4 colonnes (ToDo, Doing, Waiting, Done), chaque carte se déplace et évolue en fonction de son avancée. Si dans la pratique on se rend compte qu’une tâche est trop importante ou englobe trop de sous-tâche, celle-ci peut se transformer en carte à part entière. Ceci permet de ne pas rester figé sur une vision trop globale et de prendre le projet point par point, selon son avancement.



*Trello au jour 1 (voir annexe 3)*

*Trello au jour 60 (voir annexe 4)*

Il y a eu des ajustements à faire :

* Des éléments prévus n’ont pas été réalisé à la date prévue comme la fonctionnalité Chat.
* Des éléments non anticipés se sont rajouté comme la pagination par exemple.
  1. Spécifications techniques

Dans les différentes étapes de réalisation du projet, plusieurs outils, technologies et langages ont été nécessaires pour sa réalisation.

* + 1. Outils de gestion de projet

**Trello**

Présenté dans le chapitre précédent, Trello est un outil en ligne d’aide à la gestion de projet, avec un système de carte.

***GanttProject***

GanttProject est un outil d’aide à la gestion de projet permettant de créer des diagrammes de Gantt.

* + 1. Outils de conceptualisation

***Looping***

Looping est un logiciel français permettant de concevoir des modèles conceptuels de données (MCD / MLD). Le MCD est une représentation schématique de la base de données et des relations entre les différentes tables.

***Mindmeister***

Mindmeister est un outil web gratuit permettant de réaliser des cartographies mentales. Cet outil m’a permis de réaliser l’arborescence du site.

***Pencil***

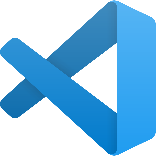
Pencil est un logiciel gratuit permettant de réaliser des maquettages visuels.

* + 1. Outils de versionning

**Git/Github**

Git est un logiciel de gestion de version. Git permet en effet de gérer les différentes versions du site, ses mises à jours. C’est un outil très utilisé en entreprise et dans le travail en équipe.  
GitHub a été pour moi le moyen de stocker les différentes versions de mon projet, d’avoir ainsi des sauvegardes, de pouvoir récupérer mon projet sur différents postes (entre mon domicile et le centre de formation par exemple).

* + 1. Outils liés au développement

***Visual Studio Code***

Pour le développement de mon projet, j’ai souhaité travailler avec Visual Studio Code car il permet l’ajout de nombreux plugins utiles au développement.

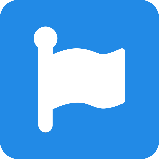
**Laragon**

Pour gérer l’environnement de développement en local j’ai travaillé avec Laragon, qui comprends un serveur Apache, MySQL et interprète PHP.

****

**HeidiSQL**

HeidiSQL est un outil d’administration de base de données, livré avec Laragon.

**Fontawesome**

Fontawesome est une librairie d’icônes vectorielles.

Toutes les icônes présentes dans le projet viennent de cette librairie.

* + 1. Langages, bibliothèques et environnement de développements web
* Les langages côté client :

**HTML 5**

Le HyperText Markup Language (HTML) est un langage permettant de rédiger de l’hypertexte.

Son système de balisage et d’attribut permet de définir le sens du contenu et son interprétation par les navigateurs (représenté dans le Document Object Manager (DOM)).

La sémantique est une notion importante en HTML car chaque balise a son utilité et les moteurs de recherche se basent aujourd’hui sur ces balises pour définir des mots-clés et le référencement.

**CSS et la bibliothèque KnaCSS**

Le Cascading Style Sheets (CSS) est un langage de feuille de style utilisé pour illustrer le langage HTML.

En se basant du les balises et attributs définit en HTML, le CSS permet de mettre du style au texte : définir la taille, la couleur, la position…

Le CSS définit aussi l’aspect responsive (le fait d’adapter le visuel au support) notamment avec les medias queries. Dans ce but a été développé le principe des flexbox et plus récemment les grids.

Pour m’aider dans la réalisation du CSS, j’ai utilisé la bibliothèque CSS KnaCSS pour la réalisation du bouton du menu pour le responsive ainsi que pour styliser les checkbox.

**JavaScript et la bibliothèque JQuery**

JavaScript est un langage asynchrone de programmation événementiel.

A la base employée pour créer des pages web interactives, donc pour le front end, JavaScript est de plus en plus utilisé côté serveur notamment avec Node.js.

JavaScript permet donc de rendre les pages web plus interactives, en interagissant directement avec le DOM, une interface de programmation où les balises et attribut HTML et CSS de chaque page sont représentées.

JavaScript intègre également AJAX (Asynchronous JavaScript And XML), qui est utilisé pour traiter des requêtes http, de manière asynchrone et permettant ainsi de récupérer des données et de les afficher, sans rafraîchir la page internet.

Dans mon projet, j’ai utilisé JQuery, une bibliothèque JavaScript qui simplifie sa syntaxe notamment avec son principe de sélecteur.

JavaScript & JQuery m’ont été très utiles dans l’élaboration de mon projet, notamment pour déclencher les fenêtres modales, et Ajax m’a permis de gérer beaucoup d’événements, par exemple le changement d’état du bouton « J’aime ».

* Les langages côté serveur :

**PHP**

PHP Hypertext Preporcessor (PHP) est un langage côté serveur principalement utilisé pour produire des pages web dynamiques via un serveur HTTP. Il a comme fonction le traitement d’une requête et l’envoi d’une réponse : le résultat de la requête. PHP est un langage orienté objet.

* Le Framework

Un framework représente littéralement un « cadre de travail ».

Le but est d’organiser, de structurer le code par des conventions de nommage, une organisation de fichiers permettant notamment au projet d’être lu et modifié par quelqu’un d’autre.

****

**Symfony**

Pour ce projet, j’ai choisi de travailler avec le framework Symfony.

Ecrit en PHP, ce framework est basé sur un design pattern MVP (Model - View - Presenter).

* 1. Pourquoi Symfony ?

Utilisé en formation, je me suis tourné vers Symfony pour réaliser ce projet pour trois raisons.  
La première est sa facilité de mise en place et son utilisation.  
La seconde concerne son design pattern.  
La troisième est sa sécurité.

* + 1. Son utilisation
* Composer

Composer est un gestionnaire de dépendance. Il permet d’installer l’environnement, de le mettre à jour et de gérer des bundles.

Composer est nécessaire a l’installation de symfony qui, une fois installé, permet de créer un projet ou de récupérer les dépendances nécessaires à un projet grâce à deux commandes :

**symfony new <ProjectName> --full**

**composer install**

* Doctrine

Doctrine est un ORM (Object Relationale Mapping), un moteur qui prend en charge la relation entre la base de données et l’application.

L’ORM fait le lien ou *mapping* entre les objets, appelés *entités,* et les éléments de la base de données.

Doctrine utilise le Doctrine Query Language (DQL), un langage de requête orienté objet. Il est utilisé à la place du langage SQL pour créer des requêtes et manipuler la base de données.

* La console

L’utilisation de la console a été quotidienne, notamment pour utiliser les commandes liées à *Composer* mais aussi celles liées à *php bin* et à *Git.*

**php bin/console make:entity**

**symfony server:start**

**git push**

* Twig

Twig est le générateur de Template utilisé par Symfony.

Toute les *vues* présentes dans le projet sont passées par Twig, qui permet une gestion des modèles de pages dont le contenu s’adapte à l’utilisateur.

* + 1. Le design pattern MVP

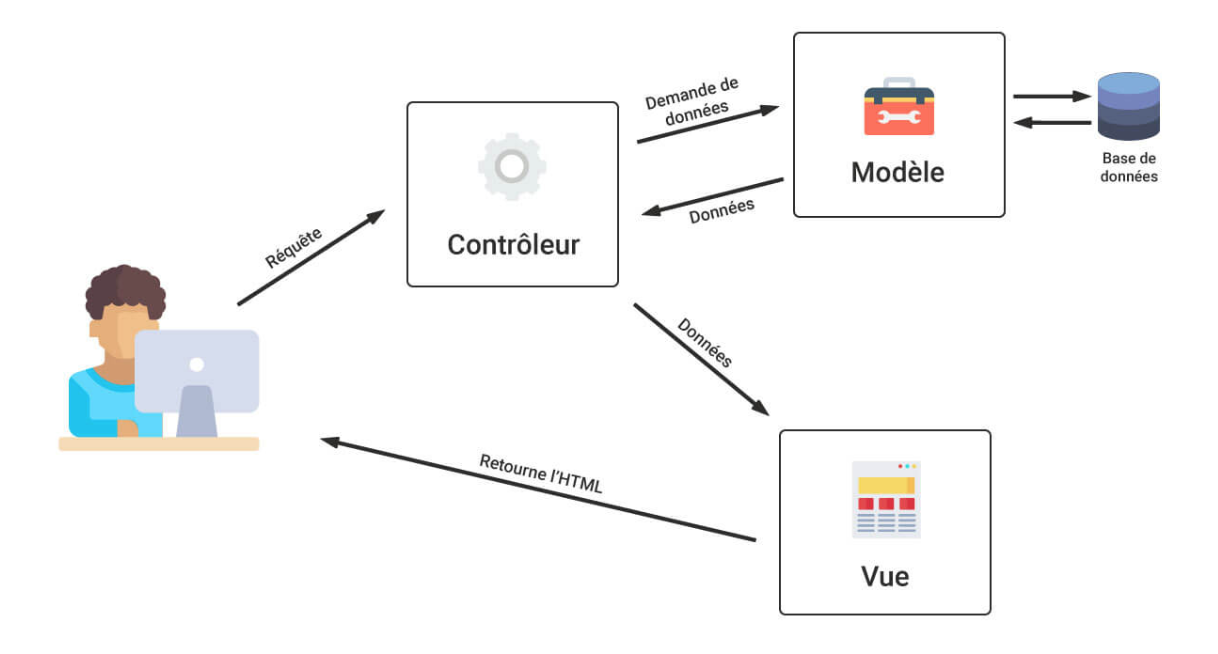
Un design pattern est un modèle de conception, une organisation caractéristique reconnue comme bonne pratique. Il existe de nombreux designs pattern.  
Symfony repose sur le design pattern Model-View-Presenter (MVP)

* C’est quoi le MVP ?

Le design pattern MV consiste à organiser le code selon son affectation (afficher, sécuriser, communiquer avec la base de données). L’idée étant qu’un code organisé selon un modèle reconnu permette de structurer le code, de créer des repères et de travailler en équipe par exemple.

Lorsqu’un utilisateur fait une requête (un clic sur un bouton par exemple), la requête est envoyée au *Controller* qui va vérifier sa validité.  
Si la requête le nécessite, le Controller demande des données au *Model* qui communique avec la base de données. Le *Model* renvoie les données au *Controller* qui organise et appel la *View.*

La *View* prépare la page et l’affiche à l’utilisateur.



*Le MVP adapté au web*

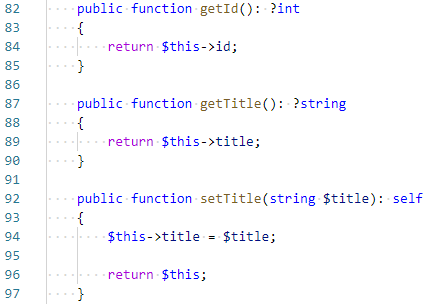
* Le MVP dans Symfony
* **Model**

Le Model est appelé Entité dans Symfony.  
C’est une classe constituée de variables et de fonctions permettant de récupérer les données (les *getters*) ou de les définir (les *setter*s). Chaque *entité* représente une table existante en base de données et les variables représentent les champs de cette table.

C’est aussi ici qu’intervient Doctrine, l’ORM de Symfony en faisant le lien entre le Model et la base de données.

Par exemple deux extraits de l’entité Photo :





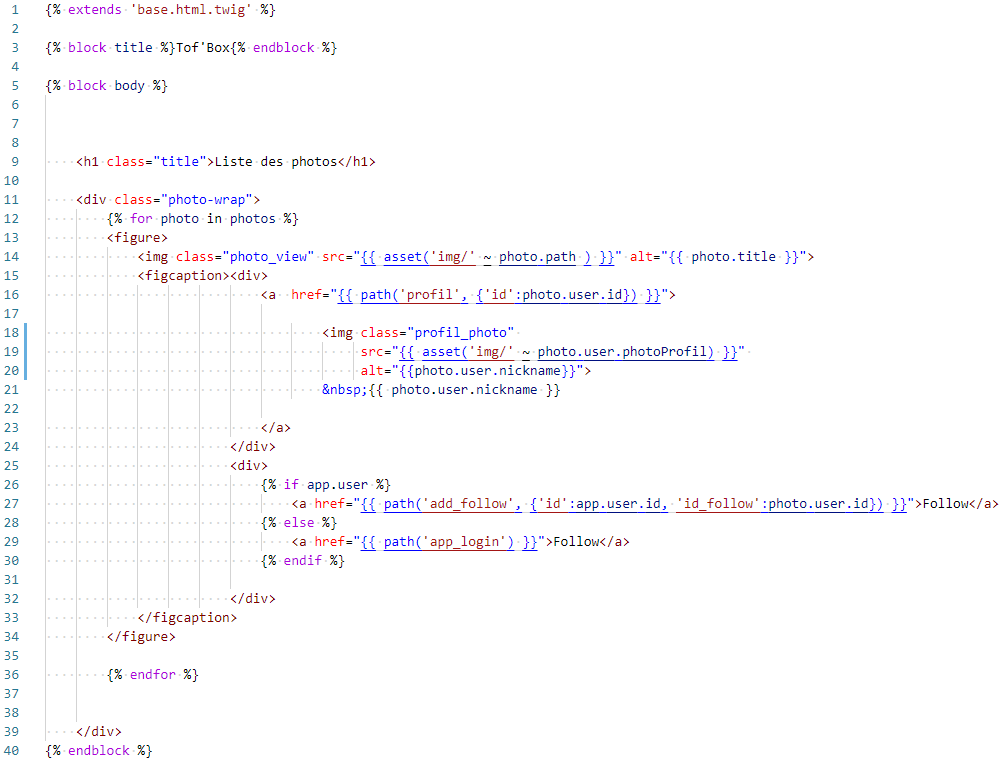
* **View**

La Vue représente le contenu affiché à l’utilisateur, c’est la partie visible de l’application.

Symfony utilise donc le moteur de Template Twig pour générer le rendu HTML.

Twig permet de mettre en place des *blocks* pour séparer le contenu, de mettre des conditions ou des boucles et d’adapter le contenu en fonction de l’utilisateur.

Voici par exemple la vue qui sert à afficher les photos :



* **Controller**

Le Controller fait en quelque sorte l’intermédiaire entre la *View* et le *Model*.

Son premier rôle est de vérifier si le chemin de la requête existe, de vérifier si les paramètre requis sont présents et de l’envoyer vers la méthode appelée.

Son second rôle est de traiter la requête. Selon la méthode appelée, le *Controller* va devoir vérifier la validité de la requête (s’il s’agit d’un formulaire que les données soit valide par exemple ). Si besoin il fera appel à Doctrine pour interagir avec la base de données, que ce soit pour en créer, en modifier, en supprimer ou en récupérer.

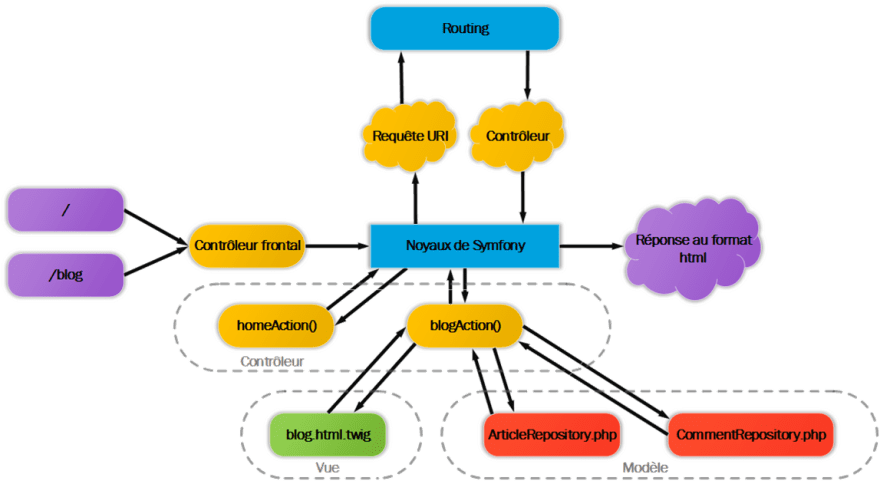
Enfin, son troisième rôle est de préparer la réponse. En général, le *Controller* va renvoyer vers une *View* avec les données mais peut aussi renvoyer vers une autre route.

Voici un extrait du PhotoController présent dans mon projet :





Pour illustrer ce fonctionnement, voici une illustration du design pattern MVP présent dans le fonctionnement de Symfony.



***Le MVP au sein de Symfony***

* + 1. La sécurité

Symfony est aussi très utile pour une raison qui aujourd’hui est incontournable : la sécurité.  
En effet Symfony permet dans son fonctionnement de se protéger contre plusieurs failles de sécurité connues.

* Les failles XSS

Les attaques de type **Cross-Site Scripting** (appelées XSS) représente l’injection de code HTML ou JavaScript dans des variables. Cela passe dans les formulaires, qui par défaut ne sont pas protégés.

Dans Symfony, c’est grâce au moteur de Template Twig que cette faille est sécurisée, car il échappe par défaut les données entrées par l’utilisateur.

* L’injection SQL

Une injection SQL est la modification d’une requête envoyée à la base de données, pour en détourner l’utilisation.

Cette fois c’est l’ORM Doctrine et notamment l’utilisation du DQL dans le *repository* qui n’interprétera pas l’injection SQL.

* La faille CSRF

Le faille **Cross Site Request Forgery** (CSRF) représente le « vol de session ». Cette attaque va forcer un utilisateur à exécuter des actions à son insu.

Symfony utilise un système de « jetons » appelés *token.* Un token est un jeton unique qui sera vérifié à chaque fois qu’un formulaire est remplis, par exemple.

Ainsi, ce token est créé dans tout formulaire généré par Symfony et obligatoire pour toute action ayant un impact sur un utilisateur (changement de mot de passe, d’adresse email…) car jugé sensible.

1. Réalisation

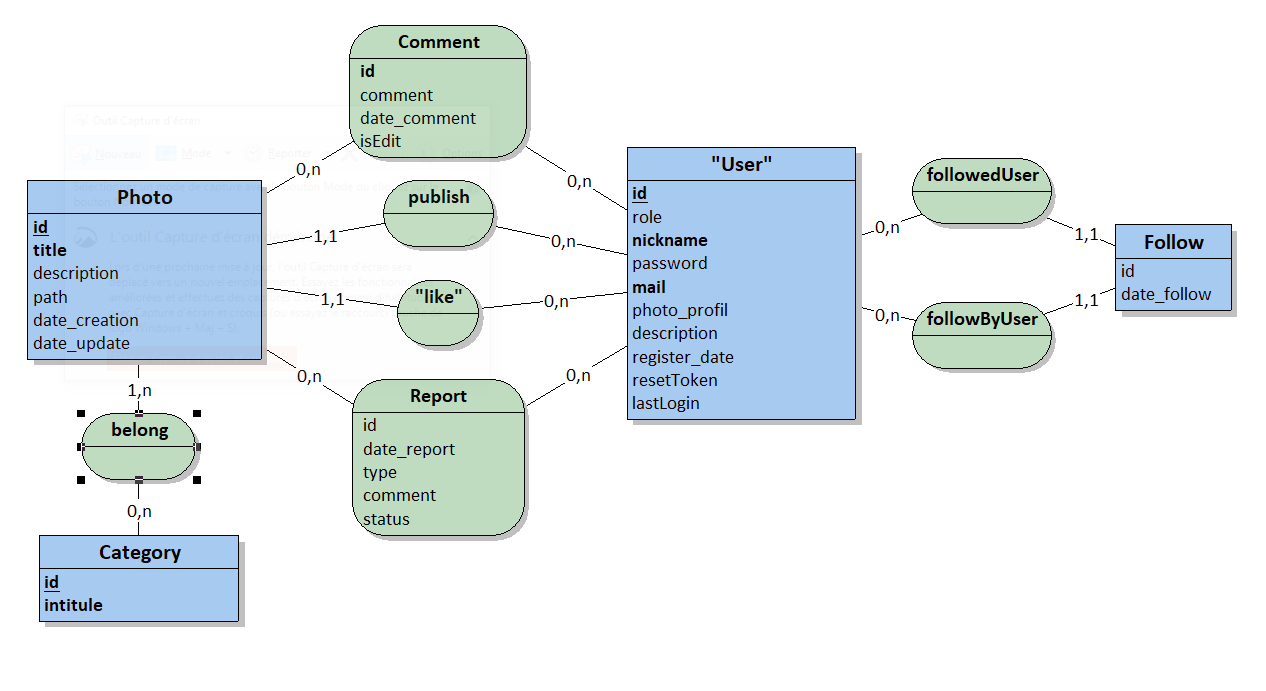
Avant de commencer à taper des lignes de codes, j’ai dû me poser quelques questions : à quoi va ressembler le site ? Que doit contenir la base de données ?

Ainsi la réalisation du projet commence par s*a conception*.

Une fois cette partie définis, la *mise en œuvre* du projet a été entame, avec la création de l’architecture du projet et cette fois à base de lignes de codes.

* 1. Conception
     1. Schématisation de la base de données

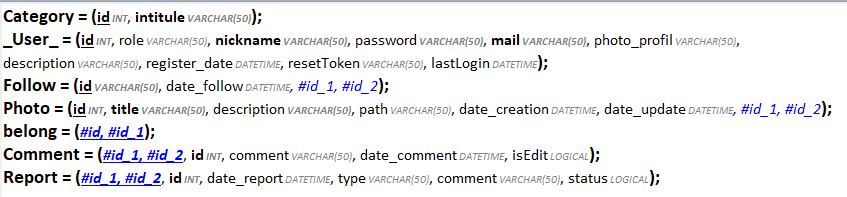
**MCD :** Le modèle de conception de donnée a pour but schématiser la base de données et les relations entre elles.



*MCD du projet Tof’Box (voir Annexe 5)*

**MLD :** Le modèle logique de données est la modélisation des données qui tient compte du niveau d’organisation des données. Fait à partir du MCD, le MLD affiche les tables relationnelles, ainsi que les clés primaires et étrangères.

Looping génère le MLD dans un format textuel.



MLD du projet Tof’Box

La base de données aura donc :

* 4 tables (Photo, Category, User & Follow)
* 3 tables relationnels (Comment, Report et belong) de type *ManyToMany*
* 4 associations de type *OneToMany* (publish, like, followedUser & followByUser)

Mon projet s’articule en grande partie entre l’utilisateur et la photo, qui sont les éléments centraux de mon projet : un réseau social de partage de photographie.

* + 1. Maquettage et rendu visuel

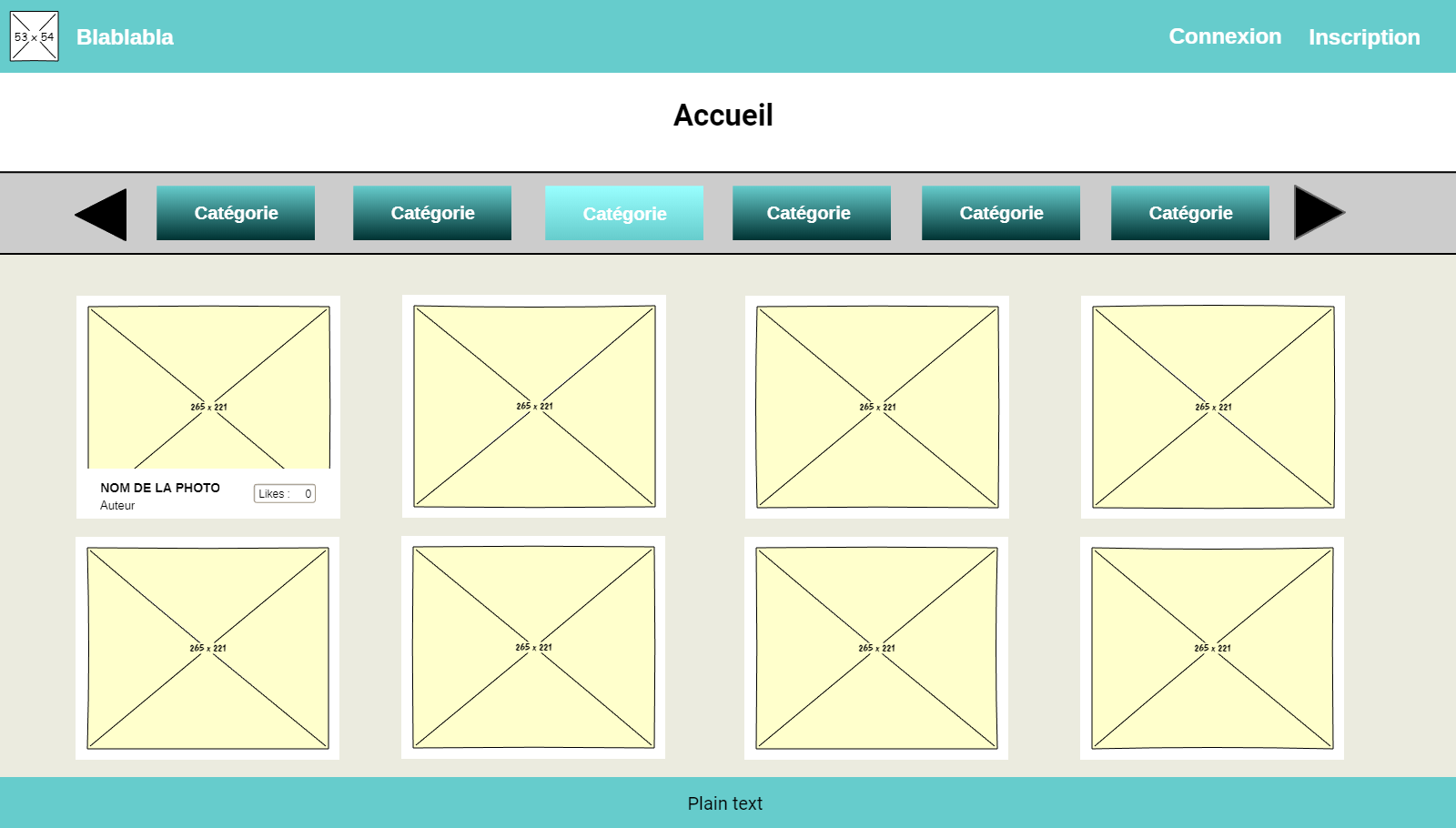
Afin de savoir dans quelle direction partir, il fallait donner un squelette à au rendu visuelle de ce projet, faire des choix de design UX qui découlent des objectifs du projet, des fonctionnalités à mettre en avant, au responsive design...

**Maquettage :**

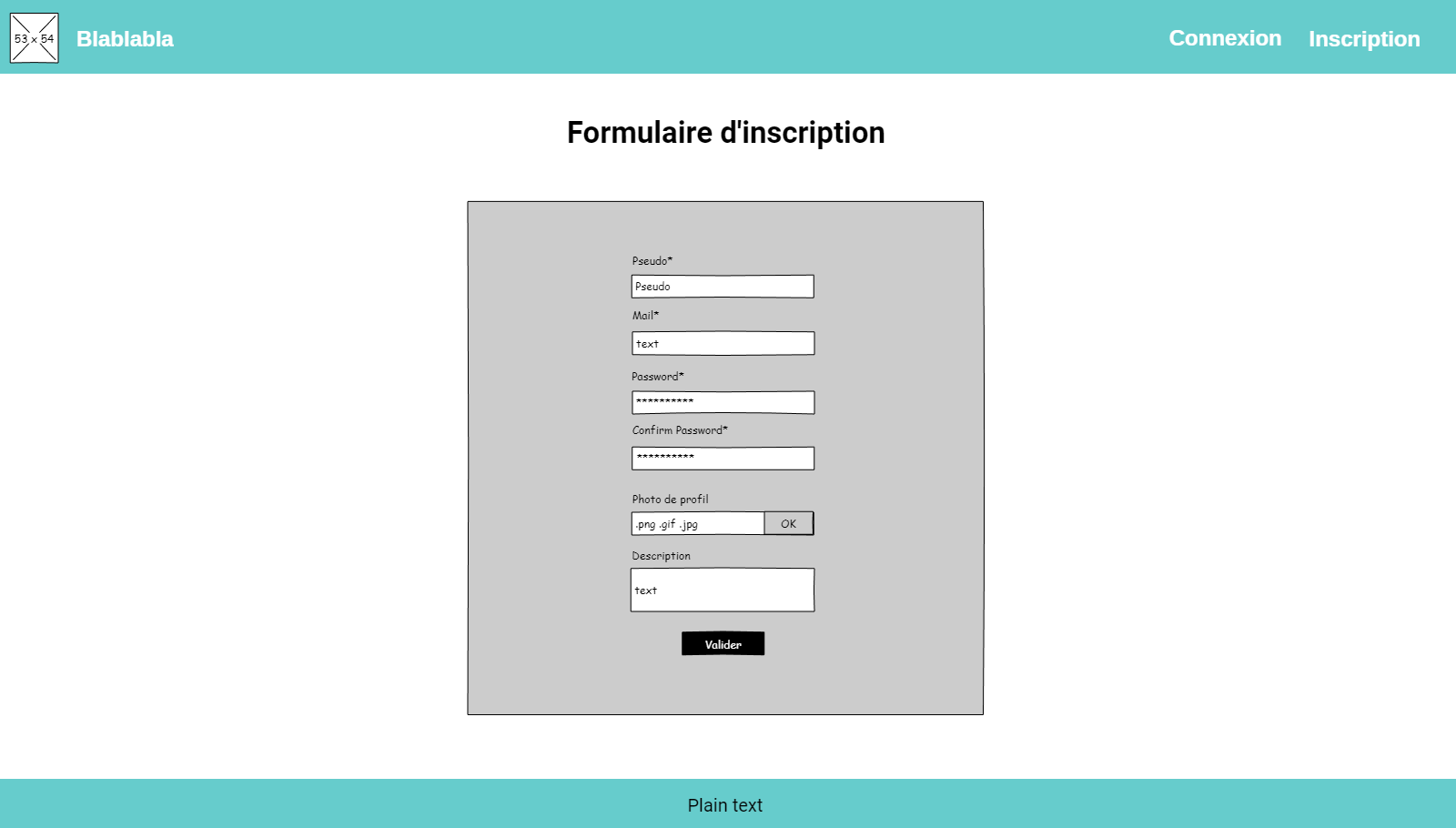
Le maquettage permet donc de réaliser ce squelette, de donner un premier rendu visuel au projet.

J’ai choisi d’illustrer ici 3 images représentant l’essence visuelle du projet : la page d’accueil, un formulaire type ainsi que le responsive de la page d’accueil sur mobile et tablette.

A noter qu’à ce moment de la conception le projet n’avais pas encore de nom et les choix des couleurs était provisoire.  
Le maquettage a été réalisé avec le logiciel Pencil.

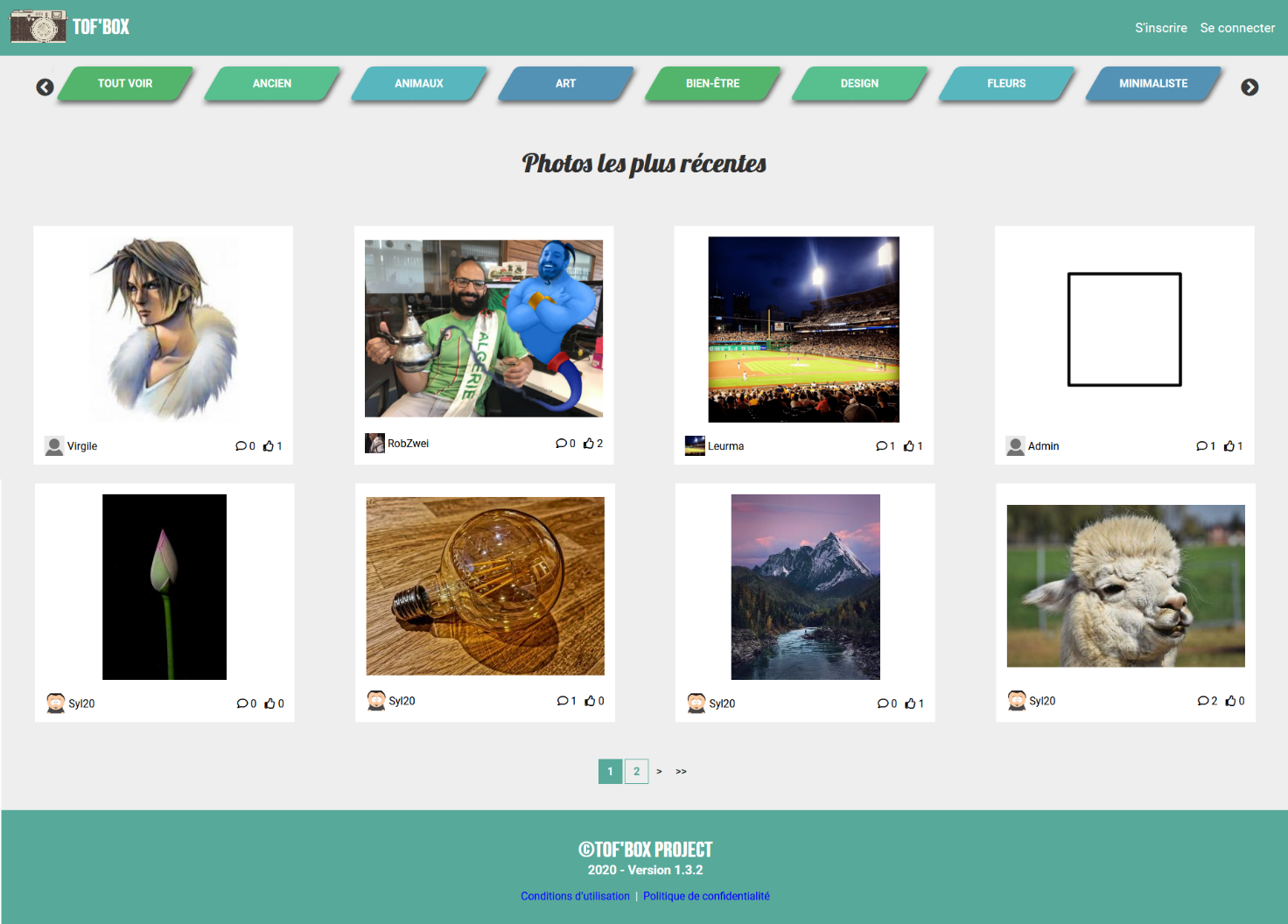


*Maquette de la page d’accueil*



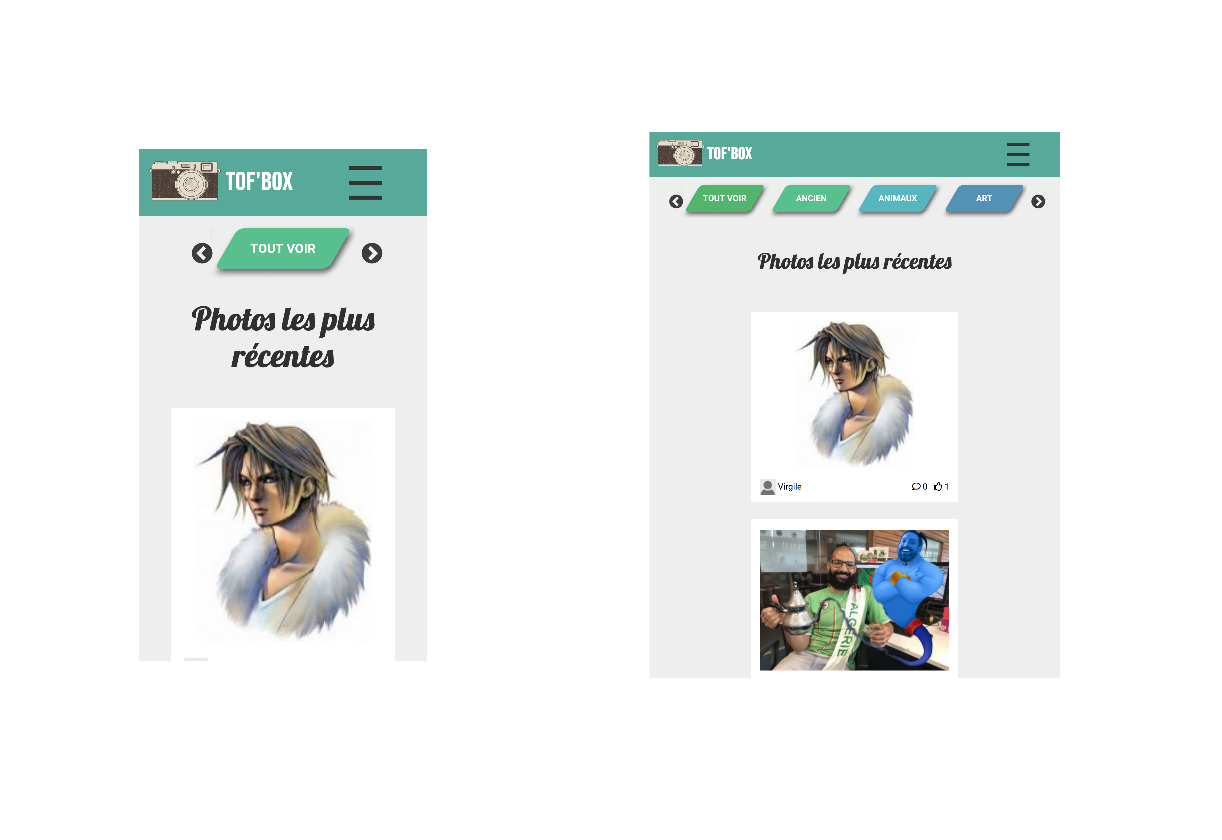
*Maquette d’une page de fomulaire type*

*Maquette de la page d’accueil responsive, tu format tablette et mobile*

**Rendu visuel effectif**

*Rendu visuelle de la page d’accueil*

*Rendu visuelle de la page du formulaire de connexion*



*Rendu visuelle sur mobile et tablette*

On peut constater que l’esprit générale du site a évolué par rapport à la maquette mais que l’ensemble est plutôt respecté.

* 1. Mise en œuvre et cheminement d’une fonctionnalité

Une fois la conception réalisée, la mise en œuvre du projet a pu commencer.

Composer et Symfony sont installés sur l’environnement de travail.

Symfony met à la disposition du développeur un ensemble de commande lui permettant de crée et mettre en place les différents éléments nécessaires à la réalisation du projet.

Tout d’abords, saisir la commande pour créer le projet :

**symfony new TofBoxProject --full**

Ensuite, mettre en place les entités, créer et migrer vers la base de données, puis mettre en place les contrôleurs.

* + 1. Mise en place des Entités

En reprenant le MCD et le MLD, nous allons pouvoir créer les entités, qui représenterons donc les tables de la base de données.

Une commande nous soutiendra dans cette démarche :

**php bin/console make:entity**

Symfony nous guide alors sur les différents champs nécessaires à remplir afin de créer l’entité voulu.

De fait, l’entité contiendra le champ sous forme de variables, ainsi que les méthodes permettant d’obtenir l’information et de la modifier (appelé *Getters & Setters*).

L’exécution de cette commande va créer aussi, pour chaque entité, un *Repository.*

Les repository servirons à créer des méthodes pour des requêtes spécifiques.

Quelques nuances sont a apporté au niveau des *Many relations.*

En effet en cas de relation *ManyToOne* ou *ManyToMany* il faudra avoir créé les entités concernées en amont.

Ensuite, Symfony ne gère pas les tables relationnelles, donc les ManyToMany où la table relationnelle nécessite un champ comme c’est le cas ici pour *Comment* et *Report* (voir MCD). Il faudra donc créer deux relations ManyToOne afin de parvenir à l’effet voulu.

Par exemple pour *Comment :*

1. Je crée l’entité *Photo.*   
   Je crée l’entité *User.*
2. Je crée l’entité *Comment*. Dans cette entité je rajoute un champ *Photo* auquel j’attribue le type *« OneToMany »* avec l’entité *Photo.*
3. Je répète cette dernière étape avec l’entité *User.*
   * 1. Création de la base de données et migration

Une fois toutes les entités créent, il va falloir migrer vers la base de données. En amont, j’ai renseigné les informations de ma base de données dans le fichier *.env* du projet avec le nom que devra avoir la base de données.

Ici, trois commandes seront nécessaires :  
La première me permet de crée la base de données en question, avec le nom que je lui ai donnée.

**php bin/console doctrine:database:create**

La seconde prépare la migration, qui va créer un fichier de version de migration dans *App/src/Migration*

**php bin/console make:migration**

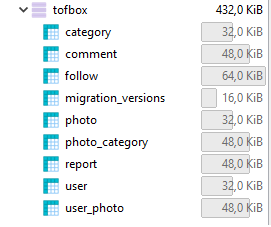
La dernière permet d’effectuer la migration. Elle va donc créer les table, les champs, les clés primaire & étrangères nécessaire, conformément à ce qui a été mis en place lors de la création des entités.

**php bin/console doctrine:migrations:migrate**

Une fois ces trois commandes exécutées, les tables sont créées en base de connées.  
Si le projet nécessite un ajout ou une modification des entités, une dernière commande nous permettra de mettre à jours les schémas de la base de données :

**php bin/console doctrine:schema:update**

Voici un aperçu de la base de données à ce stade :



*Base de données après migration*

Les entités et la base de données étant en place, il est temps maintenant de mettre en place les contrôleurs.

* + 1. Mise en place des Contrôleurs

Enfin, avant de pouvoir mettre en place les fonctionnalités, il faut mettre en place les C*ontrôleurs*.

Là aussi une commande permet de les mettre en place facilement :

**php bin/console make:controller**

Lors de son exécution, il faudra relier le contrôleur à une entité (si nécessaire), qui facilitera la mise en place des dépendances. Le Controller sera alors créé dans *App/src/Controller* ainsi que sa vue principale dans *App/templates ?*

Dans ces contrôleurs seront présent les méthodes CRUD (Create, Read, Update, Delete) mais aussi toute méthode nécessaire à la mise en place des fonctionnalités.

* 1. Cheminement d’une fonctionnalité

Notre environnement étant maintenant en place, nous allons pouvoir suivre le chemin d’une fonctionnalité de la requête jusqu’à la réponse.

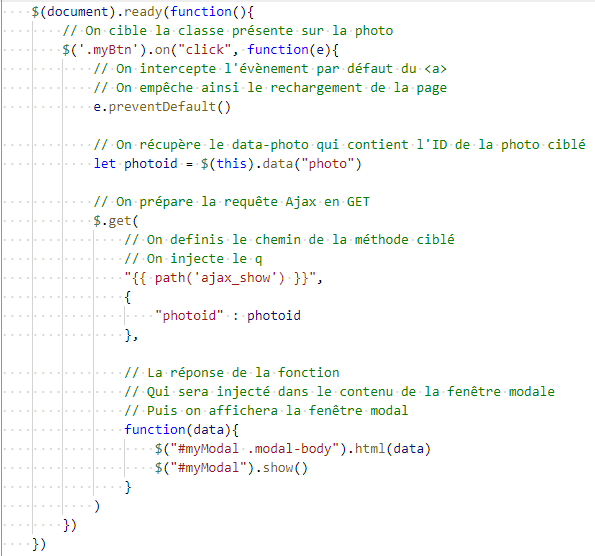
Comme un certain nombre de requêtes dans mon application ont été faite avec Ajax, j’ai choisi une fonctionnalité qui l’utilise car elle présente une particularité par rapport à une fonctionnalité plus classique.

* + 1. La requête Ajax

Dans cet exemple, nous nous situons sur la page d’accueil et l’utilisateur souhaite voire une photo de plus près en cliquant dessus.

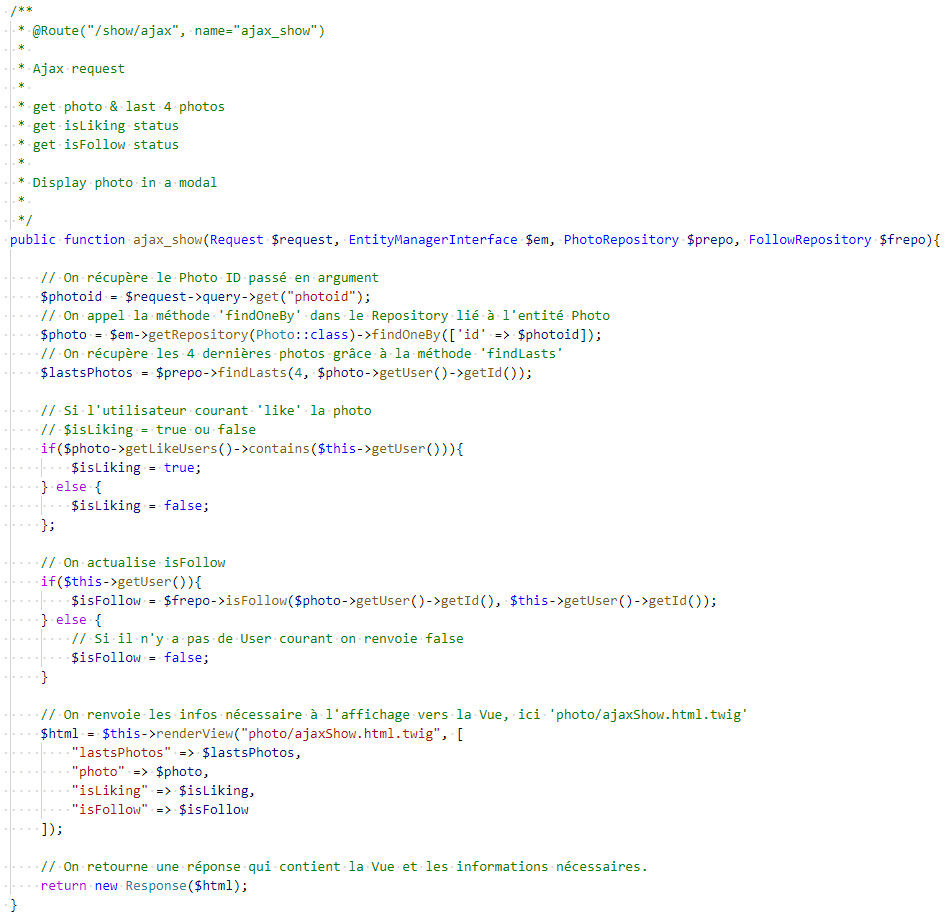
****

Cette action va être intercepté par la requête en Ajax, écrite avec JQuery, qui va récupérer les éléments dont il a besoin (ici l’ID de la photo sur laquelle l’utilisateur à cliquer) et l’envoyer vers le contrôleur concerné.



* + 1. Le Controller

Le PhotoController récupère la requête et va effectuer un ensemble d’instructions afin de récupérer les données nécessaires à l’affichage de la fenêtre modal

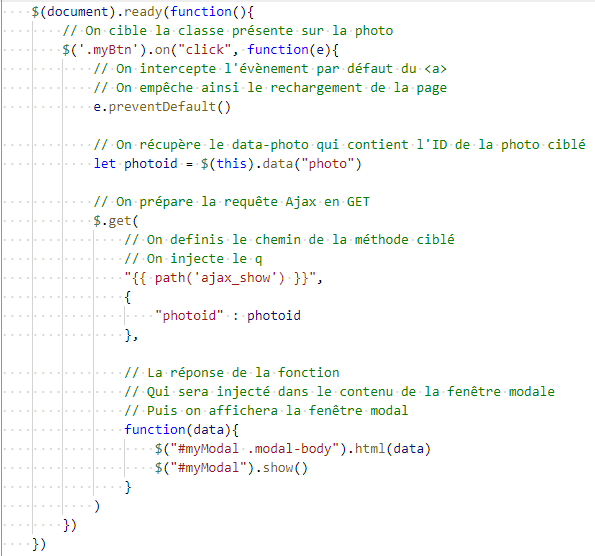
****

Puis retourne la Vue et les données vers la requête en Ajax qui va pouvoir préparer et injecter la réponse

* + 1. La réponse

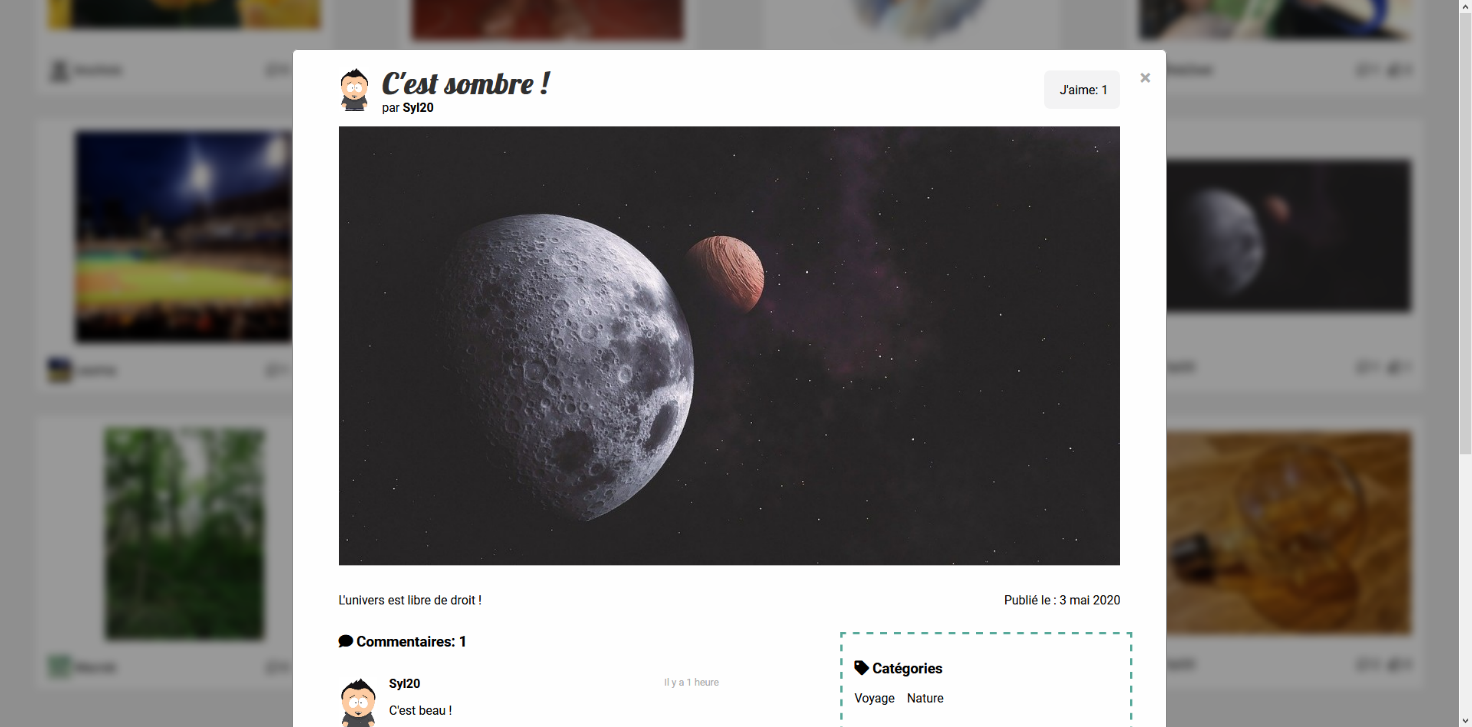
De retour dans la requête Ajax, cette fois avec la réponse renvoyée par le contrôleur.

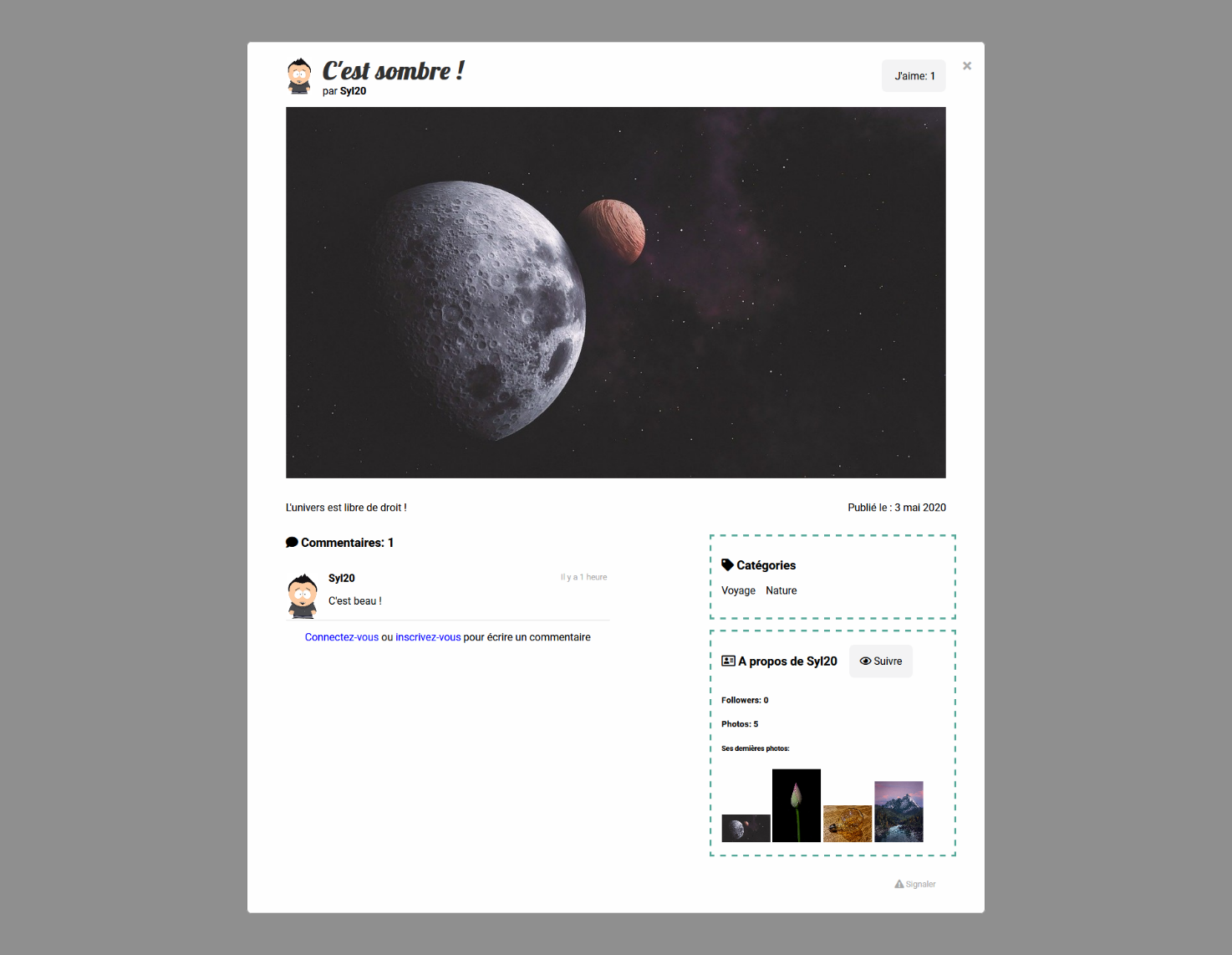
La fonction de réponse n’a plus qu’a injecté les données de la Vue dans le corps de la fenêtre modale et afficher à l’utilisateur la fenêtre



* + 1. Affichage de la vue

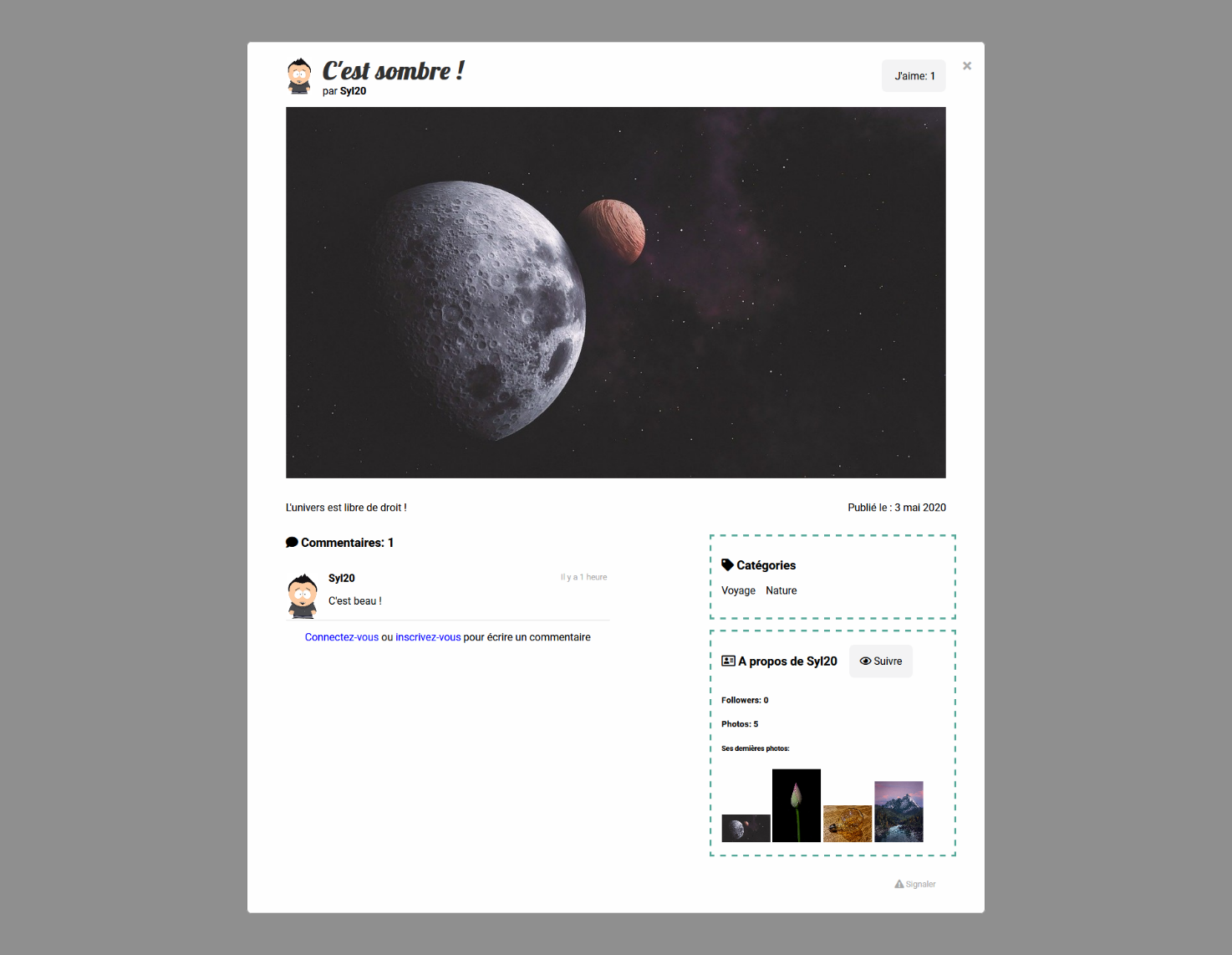
La Vue s’affiche alors à l’utilisateur, sans rafraîchissement de la page, et permet ainsi à l’utilisateur d’interagir de manière plus fluide avec l’application.





* + 1. Extension Twig

Dans mon application j’ai été amené à créer une extension Twig afin d’avoir un filtre me permettant de générer le rendu d’une date en *DateTime* au format « Il y a X temps ».



Pour cela, j’ai dû le configurer dans *App/Config/service.yaml* en rajoutant ceci :



Puis j’ai configuré le filtrer ainsi que son fonctionnement dans *App/src/Twig/AppExtension* :



Cette fonction me permet ainsi d’utiliser le filtre Twig ‘timeago’ afin de formater le rendu visuel et ainsi d’afficher le rendu souhaité.

****

* 1. Traduction de la page en anglais

Dans le cadre de mon application j’ai réalisé une fonctionnalité me permettant de mettre à jours le champ « LastLogin » dans la base de données afin de pouvoir disposer de l’information de la dernière connexion d’un utilisateur.

* + 1. Extrait de la page en anglais

**Background information**

This might be documented somewhere already, and probably also blogged about already. But at the time when I was implementing this myself my Google-fu was failing me.

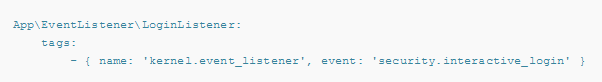
So I thought I'd document this seemingly very common feature, in case someone else is stuck and is searching for an answer.

Keep in mind this is for Symfony 4. At least that's the version I wrote and tested this for.

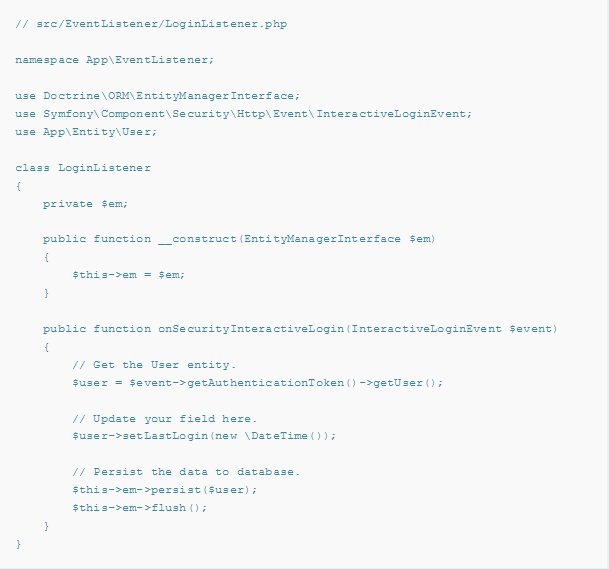
**Event Listeners**

While your application is being executed Symfony triggers a lot of event notifications, and your application can listen to these events and respond to them as well.

First you need to create an event listener that listens to  event. You do so by adding the following piece to :



Now that we have the event listener configured, let's write the code to be executed :

****

**In Summary**

It's that simple, and I'm amazed by the amount of thought and work has gone into Symfony to make things easier and quicker to develop.

Some might argue that having too much “magic” in a framework might not be good.

But from my experience so far with Symfony 4 I'm constantly amazed at how much more efficient I'm with this framework and how little “boilerplate” code I have to write to get things going.

Instead, I can concentrate on creating my application.

* + 1. Traduction de la page

**Contexte**

Cela doit déjà être documenté quelque part, et il y a probablement aussi un sujet de blog qui parle de cela. Mais au moment où je l’ai implémenté moi-même mon Google-Fu m’a fait défaut.

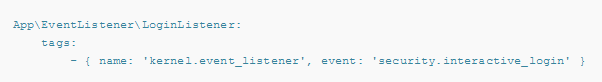
J’ai donc pensé à documenter ce qui semblait être une fonction très commune, au cas où quelqu’un d’autre serait bloqué et chercherais une réponse.

Gardez à l’esprit que cela concerne Symfony 4. Du moins c’est sur cette version que je l’ai écrite et testé.

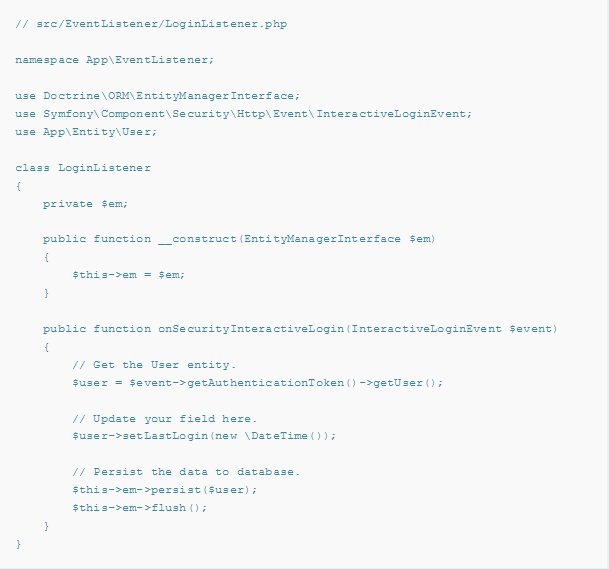
**Ecoute d’évènements**

Durant l’exécution de votre application, Symfony déclenche un certain nombre d’évènements, et votre application peut écouter ces évènements et peut très bien aussi leur répondre.

Tout d’abords vous avez besoin de créer un « écouteur d’évènement » qui écoutera l’évènement . Faites cela en ajoutant l’extrait suivant à  :



Maintenant que nous avons « l’écouteur d’évènement » configuré, nous allons écrire le code à exécuter :

****

**En résumé**

C’est tellement simple, et je suis abasourdis par la réflexion et le travail qui a été fait dans Symfony pour rendre les choses plus facile et plus rapide à développer.

Certains débattrons que d’avoir trop de « magie » dans un « cadriciel » n’est pas une bonne chose.

De mon expérience jusqu’à présent avec Symfony 4, Je suis constamment surpris de l’efficience dont je fais preuve avec ce « cadriciel » et les « standards » de code que j’ai eu à écrire pour faire fonctionner les choses.

Pendant ce temps, je peux me concentrer sur la création de mon application.

Axe d’amélioration

* Fonctionnalité Chat en temps réel avec le bundle Mercure
* Plus d’interaction avec le principe des Follow
* Possibilité de créer des albums
* Repenser l’application avec un framework JS pour plus de dynamisme et de temps réel

Remerciements

Merci Micka, Virgile, Nicolas et Stéphane !

Conclusion

Ce projet a été passionnant et très formateur !

Ce projet est la conclusion de ma formation, il m’a permis de mettre en pratique ce que j’ai appris en formation, en stage et dans ma pratique personnelle.

Ce projet vient aussi ouvrir la porte sur une nouvelle vie professionnelle dans laquelle je vais pouvoir pratiquer, expérimenter et continuer de me former à la pratique du développement.

Annexe 1 : Arborescence du projet

Annexe 2 : Diagramme de Gantt jour 1

Annexe 3 : Trello jour 1

Annexe 4 : Trello jour 67

Annexe 5 : MCD du projet Tof’Box