**PAGE DE GARDE**

SOMMAIRE

1. **Liste des compétences** du référentiel qui sont couvertes par le projet  
2. **Résumé du projet en français d’une longueur d’environ 20 lignes** soit 200 à 250 mots, ou environ 1200 caractères espaces non compris  
3. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet  
4. Spécifications **techniques** du projet, élaborées par le candidat, **y compris pour la sécurité et le web mobile**  
5. Réalisations du candidat comportant les **extraits de code les plus significatifs et en les argumentant**, y compris pour la sécurité et le web mobile  
6. Présentation du **jeu d’essai** élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)  
7. Description de la **veille, effectuée par le candidat durant le projet**, sur les vulnérabilités de sécurité  
8. Description d’une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, **à partir de site anglophone**  
9. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la **traduction en français** effectuée par le candidat sans traducteur automatique (environ 750 signes).

Typo

Titre1 Calibri

Titre2 Calibri Light

Espacement entre 8 & 12 pts

**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.

**I. 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.

**II. Organisation et expression des besoins**

**1. Spécifications fonctionnelles**

a. 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 fonctionnalitées que j’ai trié par le statut de la personne sur le site :

*Anonyme/Tous*

* 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

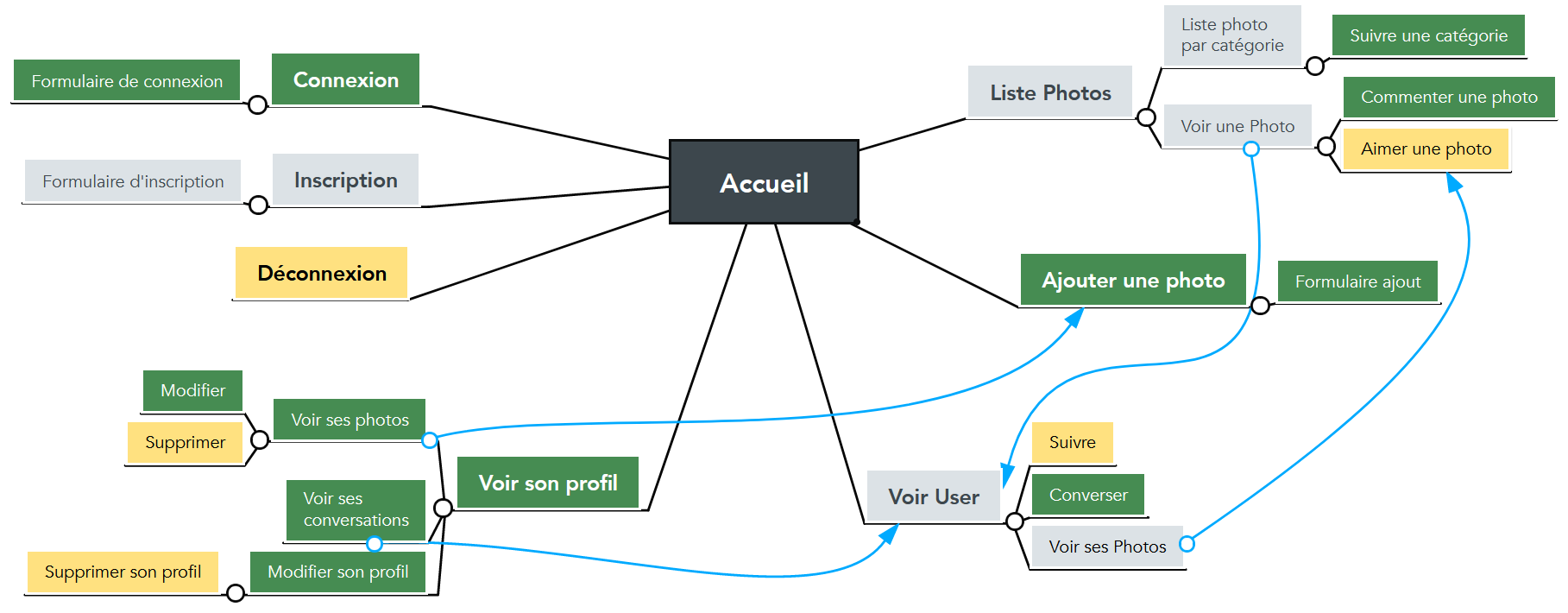
*Admin*

* 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.*

2. **Structuration et gestion de projet.**

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

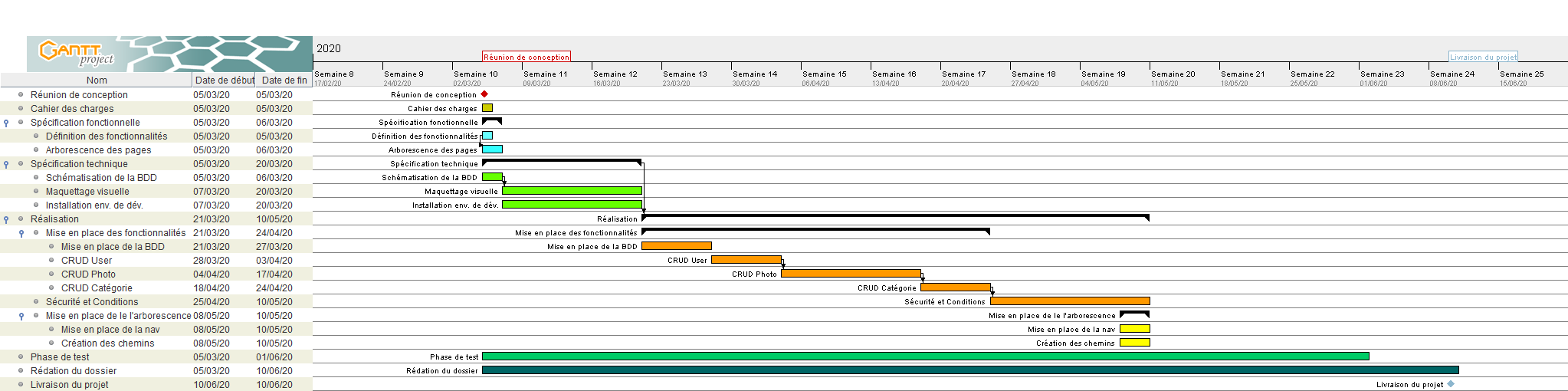
L’écriture et la réalisation du projet allait se passer en parallèle du stage en entreprise, il allait donc falloir me projeter et organiser mon temps, tout en gardant une certaine flexibilité.

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

a. 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.

La diagramme de Gantt est un outil de gestion de projet permettant de visualiser dans le temps les diverses tâche 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.

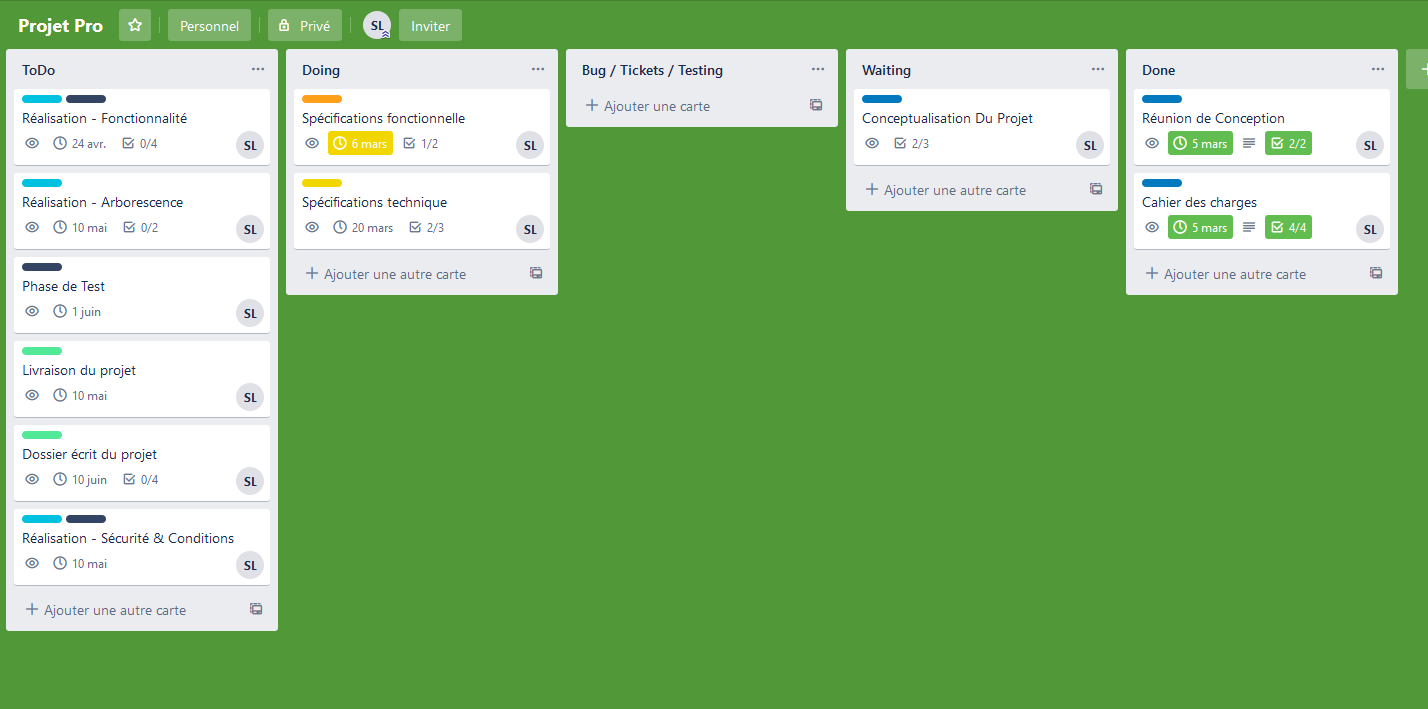
b. Organisation dans les tâches : Trello

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.

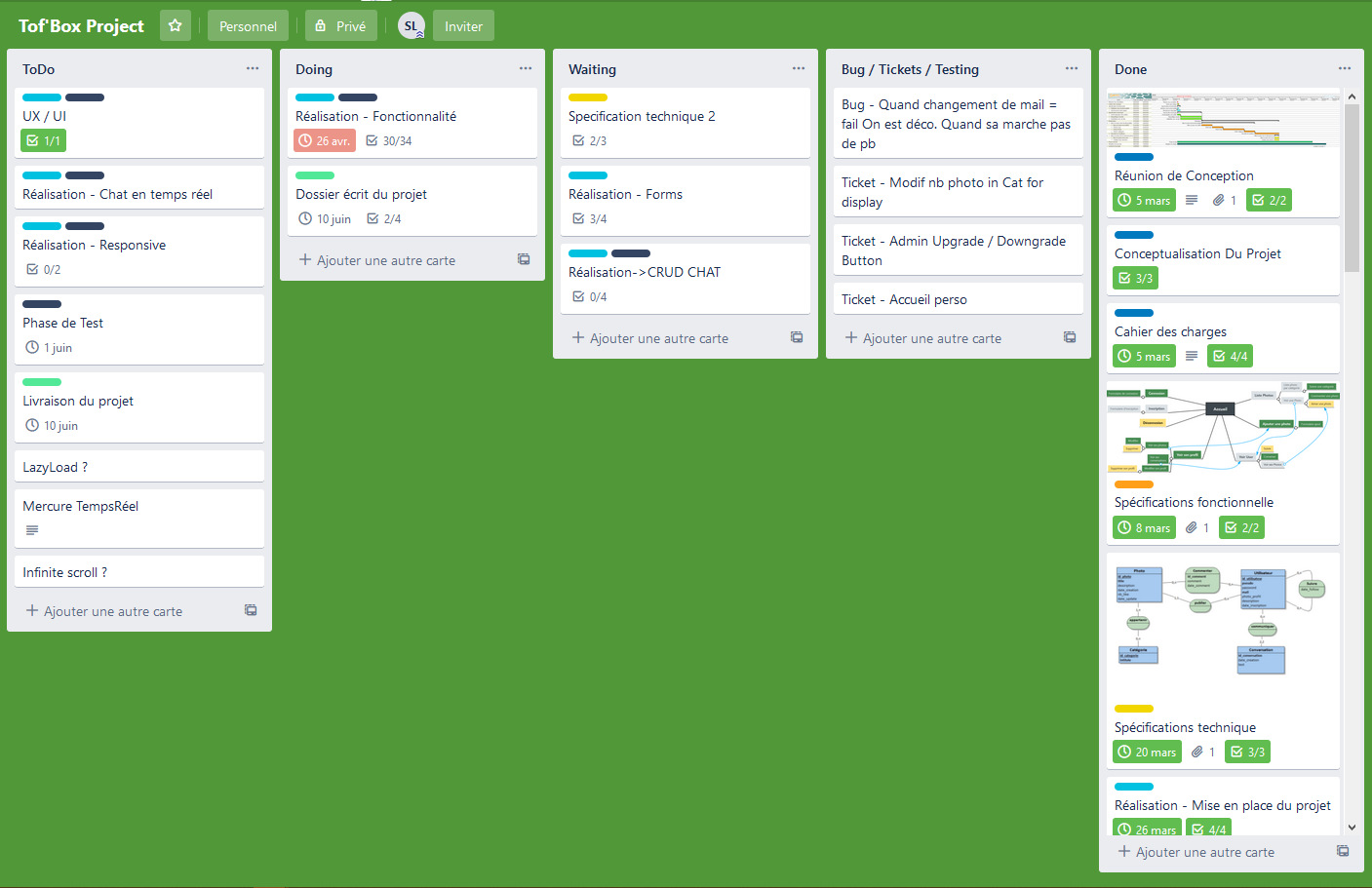
Trello est un outil que j’ai découvert lors d’un stage en agence web.

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 prévues n’ont pas été réalisé à la date prévu comme la fonctionnalité Chat.
* Des éléments non anticipé se sont rajouté comme la pagination par exemple.

3. **Spécifications techniques**

**y compris pour la sécurité et le web mobile**

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

*a. 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.

*b. Outils de conceptualisation*

*** Looping***

Looping est un logiciel français permettant de concevoir des modèles conceptuel 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***

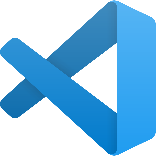
Pencil est un logiciel gratuit permettant de réaliser des maquettages visuels.

*c. Outils de versionning*

** Git/Github**

 Git est un logiciel de gestion de version. Git permet en effet de gérer les différentes version 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).

*d. Outils liés au développement*

*** Visual Studio Code***

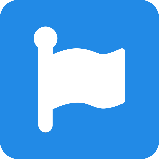
Pour le développement de mon projet, j’ai souhaité travailler avec Visual Studio Code car il permet l’ajout de nombreux plugins utiles au développement.



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



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

 **Fontawesome**

Fontawesome est une librairie d’icônes vectorielles.

Tous les icônes présent dans le projet vienne de cette librairie.

e. *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 plus facilement réaliser le responsive du site, j’ai décidé d’utiliser la bibliothèque CSS KnaCSS.

** JavaScript et la bibliothèque JQuery**

JavaScript est un langage asynchrone de programmation événementiel.

 A la base employé 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 modals, 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.

** SQL**

Structured Query Language (SQL) est un langage permettant de communiquer avec une base de données.

* 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 choisis de travailler avec le framework Symfony.

Ecrit en PHP, ce framework est basé sur un design pattern MVC.

4. **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.

*a. 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 a *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.

b. Le design pattern MVC

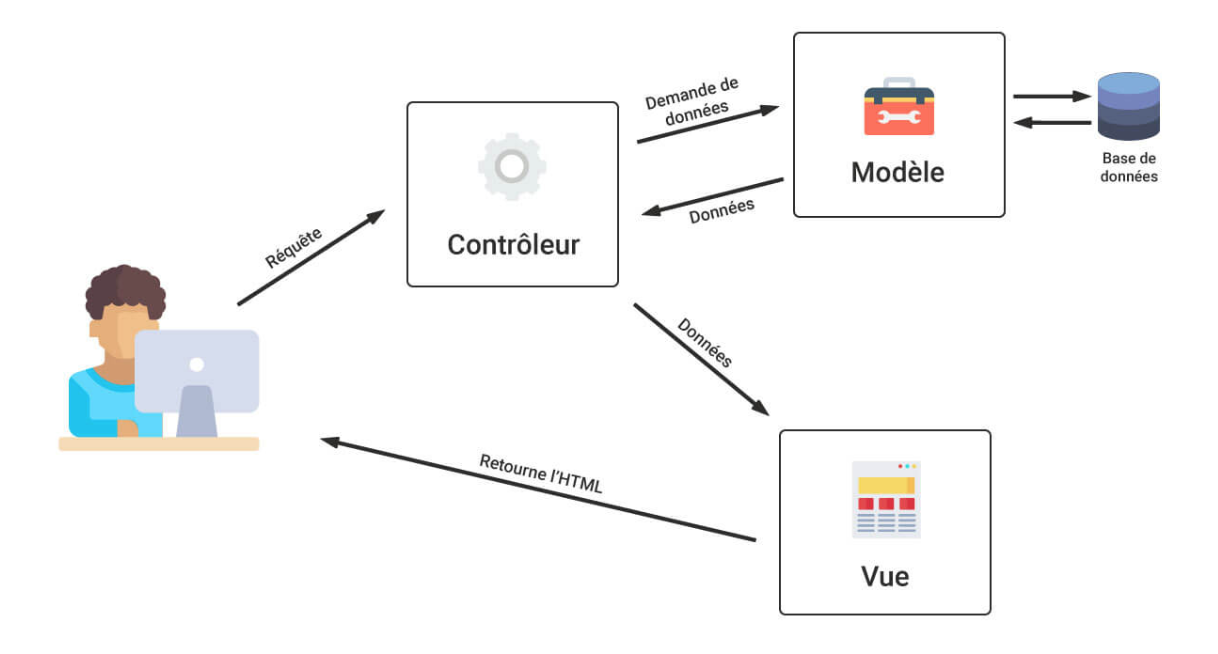
Un design pattern est un modèle de conception, une organisation caractéristique reconnue comme bonne pratique. Il existe de nombreux designs pattern.  
Symfony repose sur le design pattern Model-View-Controller (MVC)

* C’est quoi le MVC ?

Le design pattern MVC 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 un 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 MVC adapté au web*

* Le MVC 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 *setters* ). 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 un extrait de l’entité Photo :

|  |
| --- |
| <?php  namespace App\Entity;  use Doctrine\Common\Collections\ArrayCollection;  use Doctrine\Common\Collections\Collection;  use Doctrine\ORM\Mapping as ORM;  /\*\*   \* @ORM\Entity(repositoryClass="App\Repository\PhotoRepository")   \*/  class Photo  {      /\*\*       \* @ORM\Id()       \* @ORM\GeneratedValue()       \* @ORM\Column(type="integer")       \*/      private $id;      /\*\*       \* @ORM\Column(type="string", length=255)       \*/      private $title;  ...      public function getId(): ?int      {          return $this->id;      }      public function getTitle(): ?string      {          return $this->title;      }      public function setTitle(string $title): self      {          $this->title = $title;          return $this;      }  ...  } |

**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 :

|  |
| --- |
| {% extends 'base.html.twig' %}  {% block title %}Tof'Box{% endblock %}  {% block body %}      <h1>Liste des photos</h1>      <div class="photo-wrap">          {% for photo in photos %}          <figure>              <img class="photo\_view" src="{{ asset('img/' ~ photo.path ) }}" alt="{{ photo.title }}">              <figcaption><div>                              <a  href="{{ path('profil', {'id':photo.user.id}) }}">                                      <img class="profil\_photo"  src="{{ asset('img/' ~ photo.user.photoProfil) }}"  alt="{{photo.user.nickname}}">                                      &nbsp;{{ photo.user.nickname }}                              </a>                          </div>                          <div>                              {% if app.user %}                                  <a href="{{ path('add\_follow',  {'id':app.user.id, 'id\_follow':photo.user.id}) }}">Follow</a>                              {% else %}                                  <a href="{{ path('app\_login') }}">Follow</a>                              {% endif %}                          </div>              </figcaption>          </figure>          {% endfor %}      </div>  {% endblock %} |

**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.

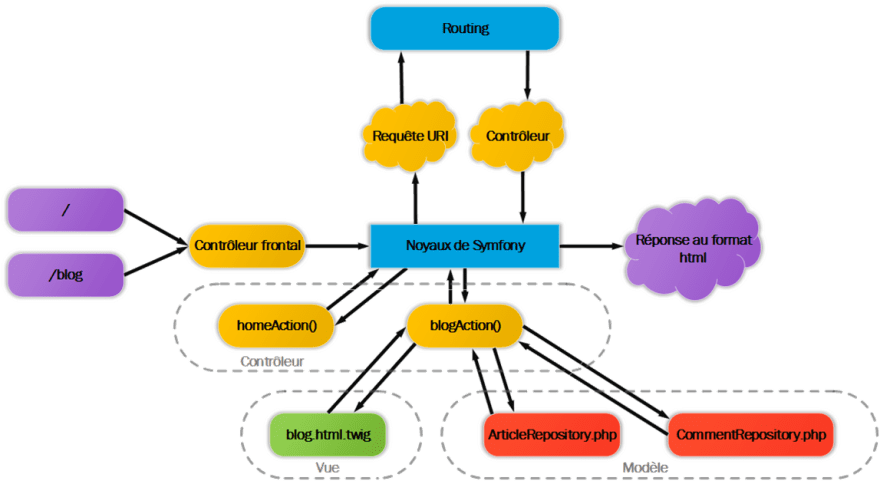
Son second rôle est de traiter la requête. Selon la méthode appelée, le *Controller* va devoir vérifier la validité de la requête ( s’il s’agit d’un formulaire que les données soit valide par exemple ). Si besoin il fera appel à Doctrine pour interagir avec la base de données, que ce soit pour en créer,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

|  |
| --- |
| <?php  namespace App\Controller;  use App\Entity\Photo;  use App\Entity\Comment;  use App\Form\PhotoType;  use App\Form\PhotoEditType;  use App\Form\AddCommentType;  use App\Repository\PhotoRepository;  use App\Repository\FollowRepository;  use Doctrine\ORM\EntityManagerInterface;  use Symfony\Component\Filesystem\Filesystem;  use Symfony\Component\HttpFoundation\Request;  use Symfony\Component\HttpFoundation\Response;  use Symfony\Component\Routing\Annotation\Route;  use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  use Symfony\Component\HttpFoundation\File\Exception\FileException;  /\*\*   \* @Route("/photo")   \*/  class PhotoController extends AbstractController  {  ...      /\*\*       \* @Route("/delete/{id}", name="delete\_photo")       \* @IsGranted("ROLE\_USER")       \*/      public function deletePhoto(Photo $photo, EntityManagerInterface $em )      {          $fileSystem = new Filesystem();          if ( $this->isGranted('ROLE\_ADMIN') || $this->getUser() == $photo->getUser()){              try {                  $em->remove($photo);                  $fileSystem->remove($this->getParameter('img\_directory').$photo->getPath());                  $em->flush();                  $this->addFlash("success", "Photo supprimée avec succès !");              } catch (FileException $e){                  $this->addFlash("error", "Un problème est survenu lors de la suppression");              }            } else {              $this->addFlash("error", "La photo ne vous appartient pas !");          }          return $this->redirectToRoute("profil", array('id' => $photo->getUser()->getId()));      }  ...  } |

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



***Le MVC au sein de Symfony***

***c. 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 crée 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.

**III.Réalisation**

Avant de commencer a taper des lignes de codes, j’ai du me poser quelques question : à quoi va ressembler le site ? Que doit contenir la base de donnée ?

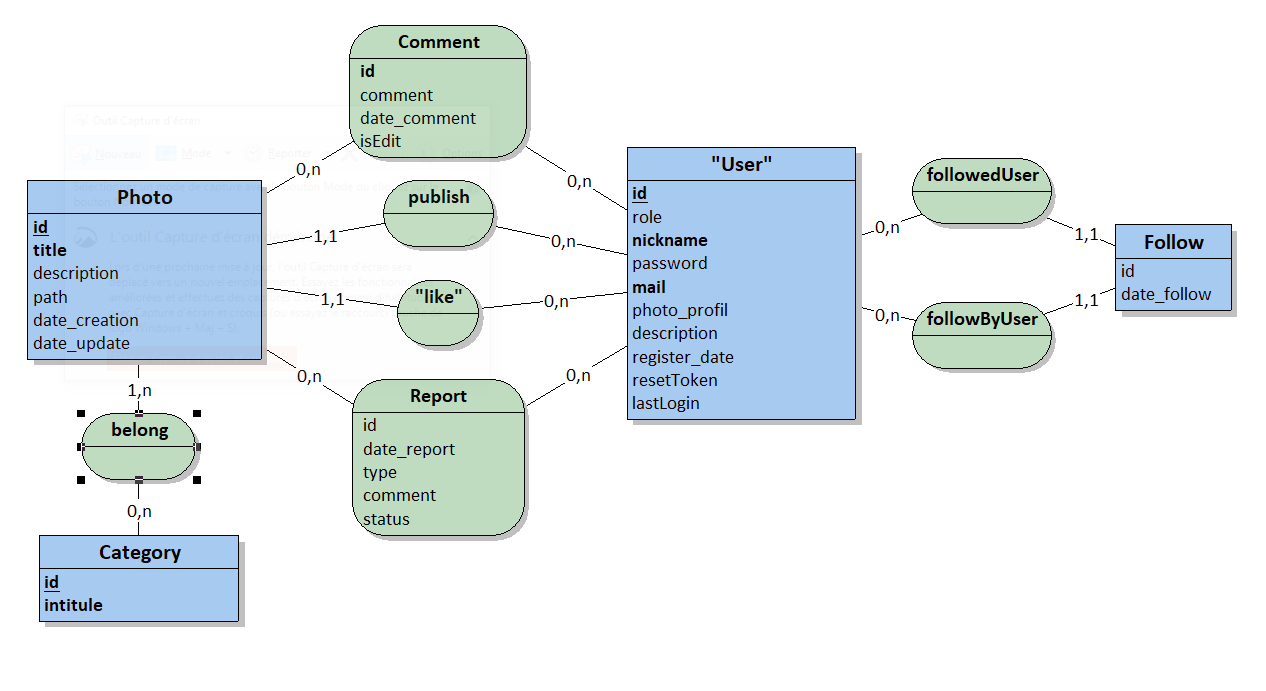
Ainsi la réalisation du projet commence par s*a conception*.

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

**1. Conception**

**a. Schématisation de la base de donnée**

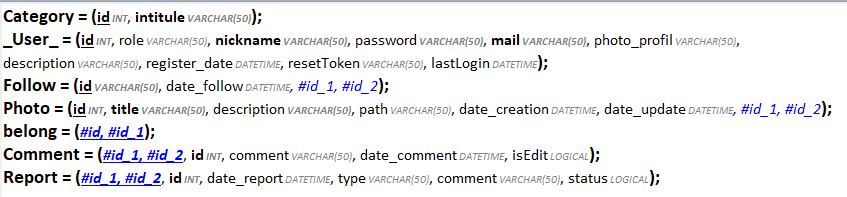
**MCD :** Le modèle de conception de donnée a pour but schématiser la base de données et les relations entre elles.



*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 relationnels, ainsi que les clés primaires et étrangères.

Looping génère le MLD dans un format textuel.



MLD du projet Tof’Box

La base de donnée 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 centrale de mon projet : un réseau sociale de partage de photographie.

**b. Maquettage et rendu visuel**

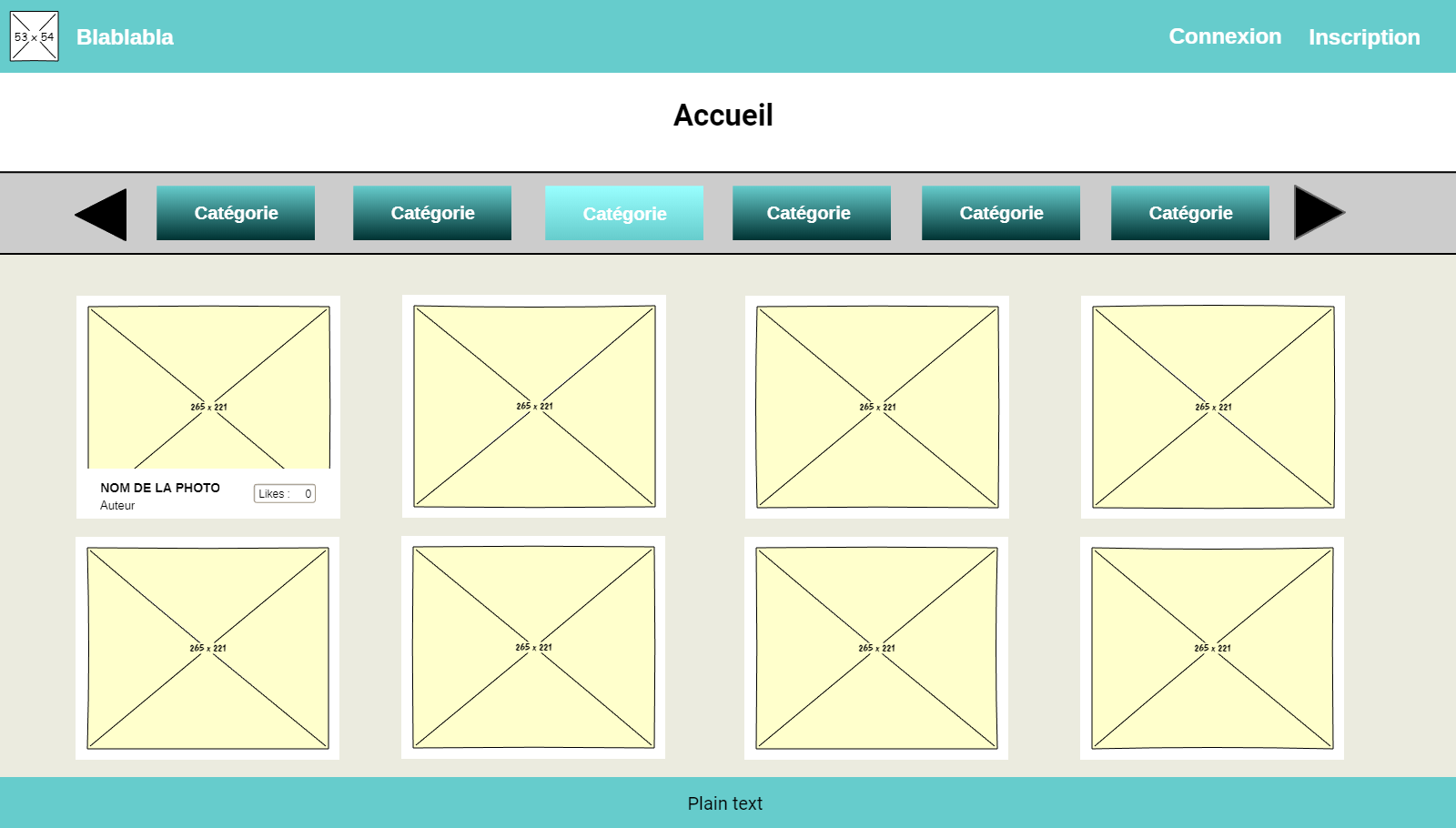
Afin de savoir dans quelle direction partir, il fallait donner un squelette à au rendu visuelle de ce projet, faire des choix de design UX qui découlent des objectifs du projet, des fonctionnalités à mettre en avant, au responsive design...

**Maquettage :**

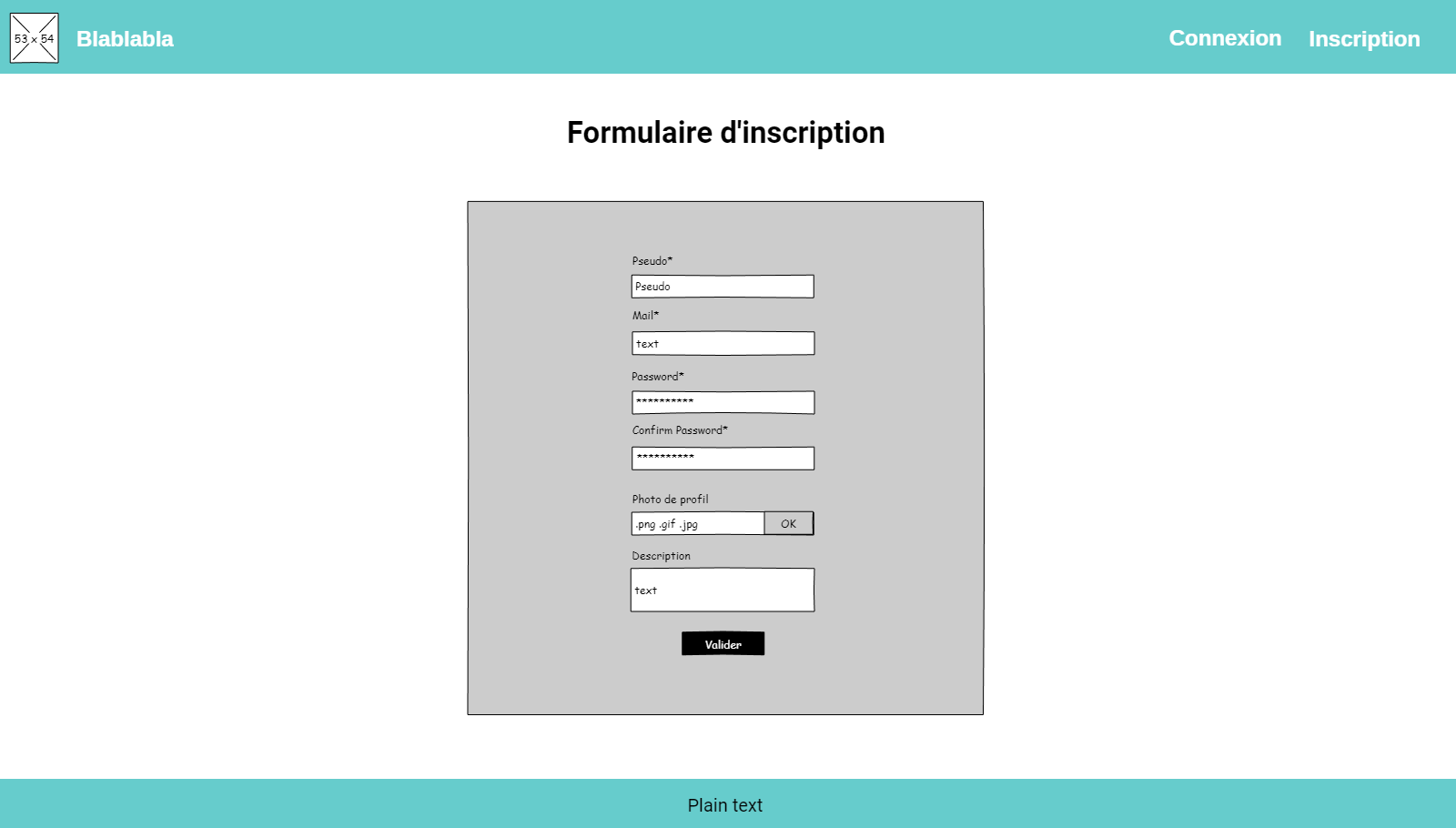
Le maquettage permet donc de réaliser ce squelette, de donner un premier rendu visuelle au projet.

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*

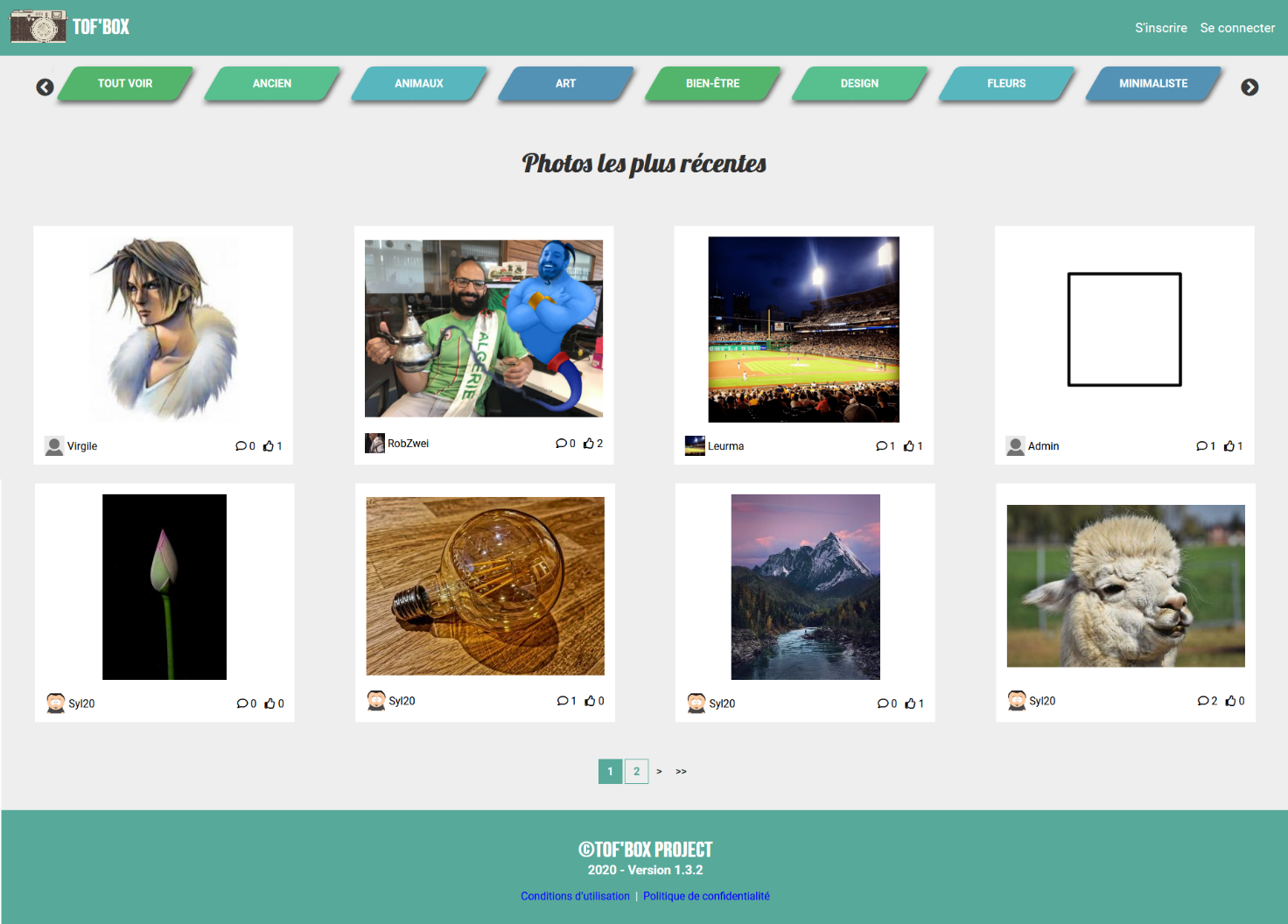


*Maquette d’une page de fomulaire type*



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

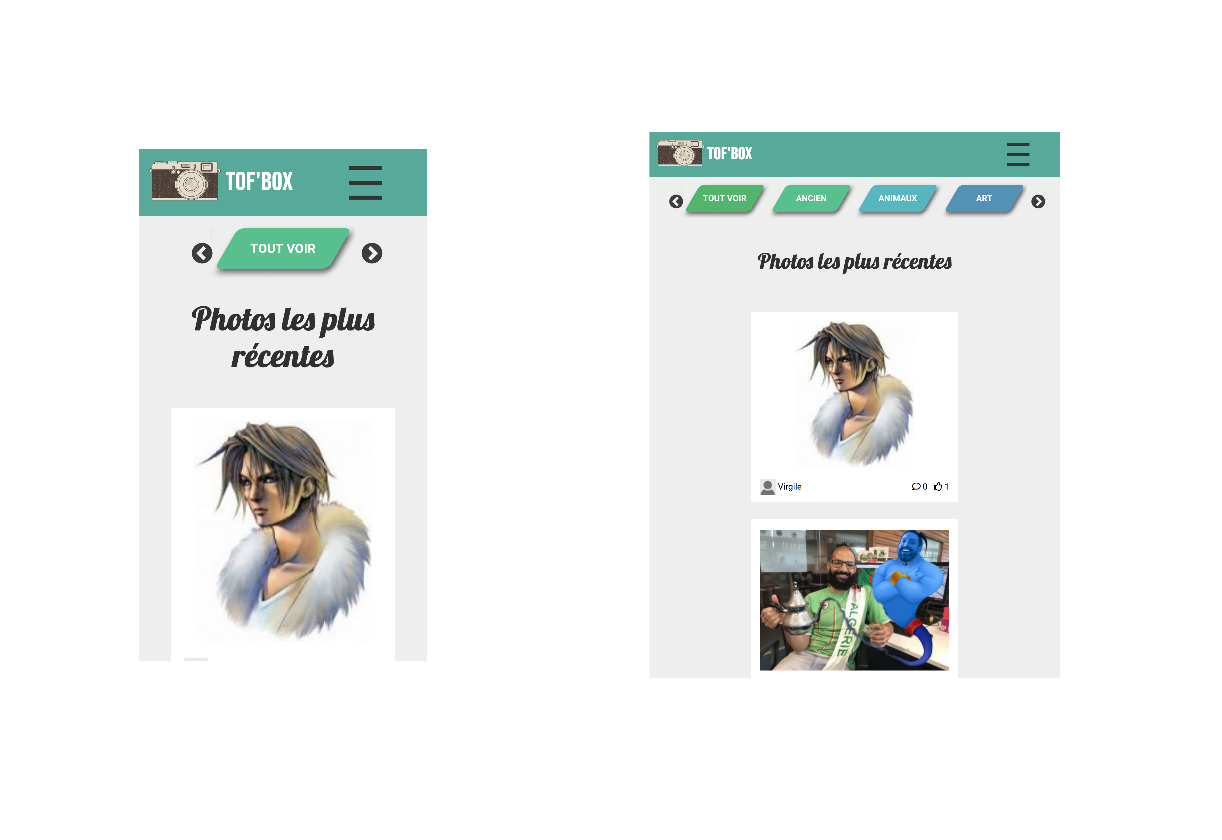
**Rendu visuel effectif**



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.

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

Une fois la conception réalisé, la mise en œuvre du projet a pu commencé.

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 nécessaire 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 donnée, puis mettre en place les controleurs.

**a. Mise en place des Entités**

En reprennant 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 nécessaire 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’execution de cette commande va crée aussi, pour chaque entité, un *Repository.*

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

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

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

Ensuite, Symfony ne gère pas les tables relationnelle, 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.*

**b. Création de la base de donnée et migration**

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

Ici, trois commandes seront nécessaire :

La première me permet de crée la base de donnée en question, avec le nom que je lui ai donnée.

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

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

**php bin/console make:migration**

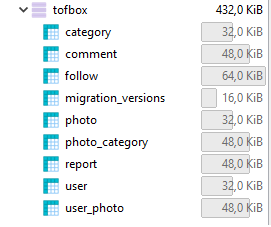
La dernière permet d’effectuer la migration. Elle va donc créer les table, les champs, les clés primaire & étrangères nécessaire, conformément à ce qui a été mis en place lors de la création des entités.

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

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

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

Voici un aperçu de la base de donnée 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.

**c. Mise en place des Contrôleurs**

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

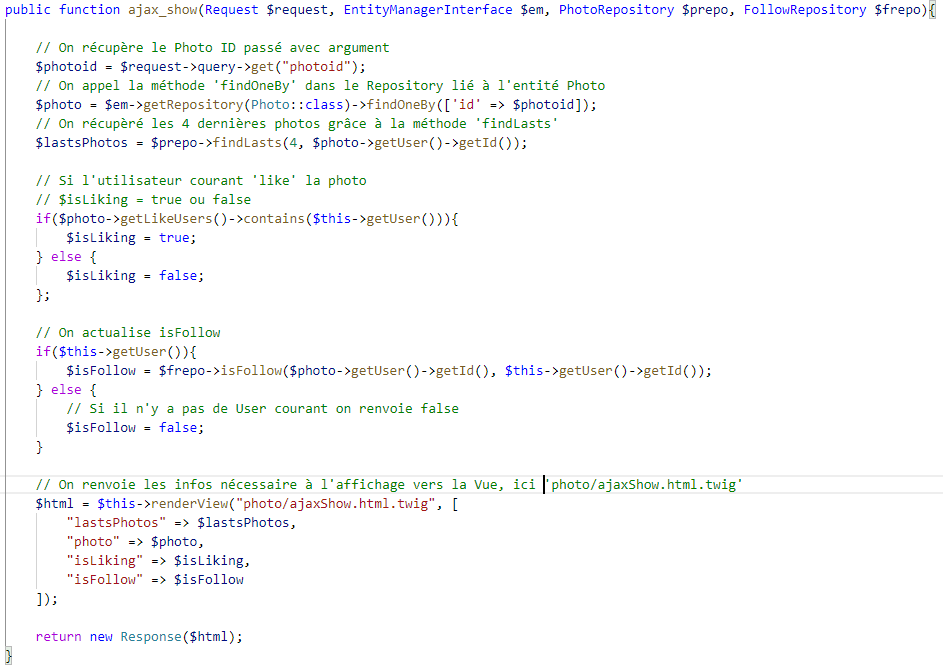
Là aussi une commande permet de les mettre en place facilement :

**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.

**3. Cheminement d’une fonctionnalité**



1. **Traduction de la page en anglais**
2. **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 security.interactive\_login event. You do so by adding the following piece to config/services.yaml:

App\EventListener\LoginListener:

tags:

- { name: 'kernel.event\_listener', event: 'security.interactive\_login' }

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

// src/EventListener/LoginListener.php

namespace App\EventListener;

use Doctrine\ORM\EntityManagerInterface;

use Symfony\Component\Security\Http\Event\InteractiveLoginEvent;

use App\Entity\User;

class LoginListener

{

private $em;

public function \_\_construct(EntityManagerInterface $em)

{

$this->em = $em;

}

public function onSecurityInteractiveLogin(InteractiveLoginEvent $event)

{

// Get the User entity.

$user = $event->getAuthenticationToken()->getUser();

// Update your field here.

$user->setLastLogin(new \DateTime());

// Persist the data to database.

$this->em->persist($user);

$this->em->flush();

}

}

**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 être déjà 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.

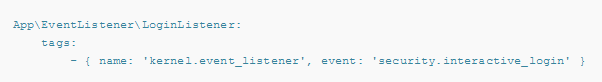
J’ai donc pensé à documenter ce qui semblait être une fonction très commune, au cas où quelqu’un d’autre serait bloqué et chercherais une réponse.

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

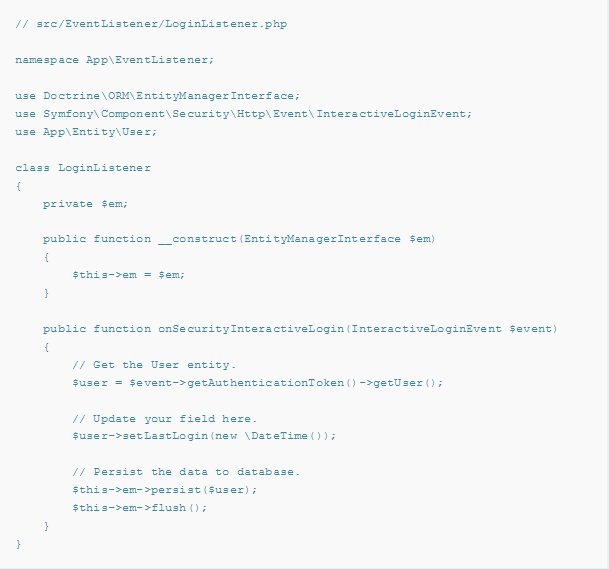
**Ecoute d’évènements**

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

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



Maintenant que nouss avons l’ « écouteur d’évènement » configuré, allons écrire le code a exécuté :

****

**En résumé**

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

Certains diront 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 comment les petites « boites à outil » de code que j’ai a écrire pour faire fonctionner les choses.

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

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

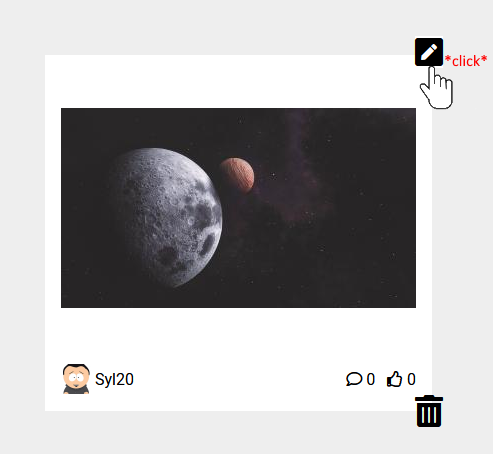
Pour cela nous allons prendre l’exemple d’un utilisateur connecté ( Syl20 ) qui souhaiterais éditer une photo qu’il a publié.

**a. Requête**

La requête est l’action de l’utilisateur. Il peut s’agir d’une URL, ou d’un lien sur lequel il a cliqué dans son moteur de recherche.

Ici, il s’agit du bouton d’édition d’une photo.

Ce bouton n’est accessible que si l’utilisateur a publié cette photo ou si il s’agit d’un administrateur du site.



**b. Controller si besoin entité ou repository**

En cliquant sur ce bouton, la requête est envoyé au PhotoController.

La methode *editPhoto()* va être appelé deux fois.

Lors du premier passage, le clic sur le bouton édition, le contrôleur va générer le formulaire d’édition *PhotoEditType* et rediriger vers la View *photo/form.html.twig*.

Une fois le formulaire validé par le clic de l’utilisateur, la requête va à nouveau appeler la méthode *editPhoto()*

Cette fois, la méthode va récupérer et traité le contenu de la requête grâce a *$form→handleRequest($request)*.

Après avoir vérifier si le formulaire a bien été soumis et sa validités, les données peuvent être modifié en base de donnée grâce à l’EntityManagerInterface via le *flush()*

Il n’y aura plus qu’a redirigé vers la liste des photos de l’utilisateur avec un message de confirmation.

|  |
| --- |
| /\*\*       \* @Route("/edit/{id}", name="edit\_photo")       \* @IsGranted("ROLE\_USER")       \*       \* Check if User can edit this photo       \* Display edit photo form       \* Save informations       \*/      public function editPhoto(Photo $photo, Request $request, EntityManagerInterface $em)      {          // 1. L'utilisateur est-il administrateur OU est-il l'utilisateur qui a posté la photo ?          if( $this->isGranted('ROLE\_ADMIN') || $this->getUser() == $photo->getUser() ){              // 2. On génère le formulaire d'édition, avec comme object l’objet Photo courant.              $form = $this->createForm(PhotoEditType::class, $photo);              // 3.              $form->handleRequest($request);                  // 4. On vérifie que le formulaire a été soumis et sa validité                  if($form->isSubmitted() && $form->isValid()) {                      // 5. On modifie la date de mise à jours                      $photo->setDateUpdate(new \DateTime('now', new \DateTimeZone('Europe/Paris') ));                      // 6. On met à jours la base de donnée                      $em->flush();                      // 7. On renvoie vers la page de affichant les photos de l'utilisateur  avec un message indiquant le succès de la requête.                      $this->addFlash("success", "La photo a bien été modifié !");                      return $this->redirectToRoute('user\_photos', array('id' => $photo->getUser()->getId()));                  }              // 4. On renvoie vers la vue avec le formulaire et les informations de la  photo concerné              return $this->render('photo/form.html.twig', [                  "form" => $form->createView(),                  "photo" => $photo              ]);          } else {              // 2. On renvoie vers la page d'accueil en indiquant que le User n'est pas  autorisé à faire cela.                  $this->addFlash("error", "La photo ne vous appartient pas !");                  return $this->redirectToRoute('home');              }      } |

**c. Réponse**

**4. Contextualisation, extrait et traduction d’une recherche sur un site anglophone.**

**a. Contextualisation**

**b. Extrait en anglais**

**c. Traduction**

6. Présentation du **jeu d’essai** élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)

Chemin d’une fonctionnalité à définir ( soit une simple « ajouter une photo » ) soit une plus complexe. Ajax ou pas ?

Requête → Controller → View

7. Description de la **veille, effectuée par le candidat durant le projet**, sur les vulnérabilités de sécurité

**Tests effectués :**

* Faille XSS - <scipt>alert('testXSS')</script>
* Injection SQL
* Faille CSRF = Chemins sécurisé !
* Vol de session => token
* Faille CRLF - [test@test.fr](mailto:test@test.fr);[pirate@pirate.fr](mailto:pirate@pirate.fr) - [victime@service.com](mailto:victime@service.com)[%0Apirate@service.com](mailto:%0Apirate@service.com) - "" ''

Se renseigner sur comment Symfony gère ça tout seul !

8. Description d’une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, **à partir de site anglophone**

**A choisir !**

9. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la **traduction en français** effectuée par le candidat sans traducteur automatique (environ 750 signes).

**A choisir !**

* **Source**
* **Traduction**

**Remerciements**

Merci Micka, Virgile, Nicolas et Stéphane !

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