

Dossier de projet



Pour la soutenance DWWM

- Titre de niveau 5 -

Melvin SEDDIK
Année 2021



Sommaire

Introduction.....	1
1 - Cahier des charges.....	2
a. Contexte – Analyse du besoin.....	2
b. Présentation du projet.....	2
c. Description fonctionnelle.....	4
d. Règlement Général sur la Protection des Données (RGPD)	8
e. Planning et suivi	11
2 – Spécifications techniques.....	12
a. Méthode MERISE et modèles de données.....	12
b. Charte graphique et typographie.....	18
c. Maquettes (cf. complément en annexe).....	18
d. Arborescence de l'application.....	19
e. Ressources logicielles.....	20
f. Architecture logicielle.....	20
3 - Réalisation.....	21
a. Réalisation de la couche modèle.....	21
b. Réalisation de la base de données	23
c. L'accès aux données	24
d. Réalisation des contrôleurs	28
e. Réalisation des vues.....	30
f. Le responsive design : l'adaptation des vues aux mobiles et aux tablettes	32
4 – Présentation d'une fonctionnalité : l'échange.....	34
5 – Sécurité	41
a. Présentation.....	41
b. Sécurisation de l'authentification	41
c. Sécurisation des routes.....	42
d. Protection contre les injections SQL.....	42
e. Protection contre la faille XSS (Cross Site Scripting).....	43
f. Protection contre les attaques CSRF (Cross-Site Request Forgery).....	44
6 – Recherche anglophone.....	44
Conclusion	48

Remerciements

Je remercie l'équipe des formateurs d'ELAN, tout particulièrement Stéphane SMAIL et Gilles MUESS pour la qualité de leur enseignement et leur disponibilité.

Je tiens également à remercier mes camarades de promo avec qui j'ai passé de très bons moments au cours de la formation.

Je remercie aussi ma famille pour son soutien sans faille.

Introduction

Je m'appelle Melvin SEDDIK, j'ai 24 ans et depuis janvier 2021, je suis une formation à ELAN pour devenir Développeur Web et Web Mobile (DWWM). Je me suis intéressé de près à la programmation en autodidacte il y a deux ans, pendant lesquels j'ai acquis quelques bases dans les langages web. Fier d'avoir pu accomplir des petits projets (page web d'informations sportives, applications utilitaires ...) et m'apercevant de la demande grandissante du marché, j'ai décidé de me professionnaliser dans cette voie. Ainsi, durant ces dix derniers mois de formation, j'ai appris de façon plus structurée et approfondie à développer des applications web qui s'adaptent à tous types d'appareils (ordinateurs, tablettes et mobiles).

Ce rapport vous présente *Troceland*, le projet web que j'ai conçu et développé à titre personnel. Il s'inspire du besoin actuel de la population de changer de mode de consommation, de revenir à des valeurs simples, dans le respect de l'humanité et de la planète.

Les exigences du référentiel de compétences du titre couvertes par le projet sont :

Développer la partie back-end d'une application web en intégrant les recommandations de sécurité	Développer la partie front-end d'une application web en intégrant les recommandations de sécurité
<ul style="list-style-type: none">- Créer une base de données- Développer les composants d'accès aux données (DAO)- Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce	<ul style="list-style-type: none">- Maquetter une application- Réaliser une interface utilisateur web statique et adaptable- Développer une interface utilisateur web dynamique- Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

1 - Cahier des charges

a. Contexte – Analyse du besoin

Il va sans dire que l'environnement s'impose parmi les principales préoccupations des Français. De plus en plus engagés personnellement à travers des gestes quotidiens, toute initiative agissant en faveur de la planète a le vent en poupe et est perçue positivement. Le CRÉDOC (Centre de Recherche pour l'Étude et l'Observation des Conditions de vie) a observé cette tendance de la population à offrir une seconde vie aux biens de consommation courante, qu'ils soient revendus, loués, empruntés ou donnés. Plus économe, le consommateur devient plus écoresponsable. C'est ainsi que certains modes de consommation comme le troc se développent. Le marché de l'occasion représente aujourd'hui 42% de la consommation totale des Français, motivés par le rejet du modèle de l'hyper consommation et une économie plus collaborative et responsable.

b. Présentation du projet

TrocLand se présente comme une application web permettant de faciliter l'échange de biens en France par le troc avec pour objectif de soutenir et favoriser la conscience collective contre le phénomène de surconsommation et à développer la solidarité et le partage.

Cibles : TrocLand offre ainsi une solution concrète à tous ceux qui souhaitent s'inscrire dans un mode de consommation éthique. Elle cible dans un premier temps l'ensemble de la population adulte en France.

Principe : Un utilisateur du site publie une annonce du bien qu'il souhaite troquer et reçoit les propositions d'échange d'autres utilisateurs. Il en accepte une seule avant de convenir d'un lieu et d'une date de rendez-vous par le biais du site. Après la rencontre, les deux troqueurs peuvent laisser une évaluation.

Afin d'être au plus près des besoins de ses cibles, l'application doit répondre à certains objectifs et respecter certains critères.

Objectif 1 : Favoriser les échanges de proximité

L'objectif principal du site est de favoriser les échanges de proximité. Ainsi, l'utilisateur connecté verra sa page d'accueil personnalisée et afficher les annonces des 15 articles les plus proches de son domicile.

L'auteur d'une annonce doit indiquer la ville où il souhaite réaliser son échange et éventuellement un périmètre acceptable autour de celle-ci (en km). L'adresse de l'échange peut donc être différente de celle du domicile. Le site s'intéresse à la distance en kilomètres entre les deux pour étudier le mode de consommation des internautes et mieux s'adapter à leur besoin.

Objectif 2 : Répondre au besoin de consommer mieux, proposer un commerce éthique et écologique

Le principe est de favoriser la circulation des biens entre particuliers, sans engendrer la moindre production. À ce titre, le site ne prévoit pas la circulation de monnaie, qu'elle soit réelle ou virtuelle.

Objectif 3 : Sécuriser les échanges et instaurer un climat de confiance

Pour protéger les utilisateurs et sécuriser les échanges, autrement dit s'assurer qu'ils vont bien recevoir l'un et l'autre ce qu'ils se sont promis, le site permet de gérer uniquement l'échange de biens et non de services. Le but est ici de réaliser un échange à un moment unique, or les services sont difficilement rendus en même temps et peuvent intégrer une notion de fréquence.

Le lieu et la date convenus pour l'échange sont tracés, donc stockés en base de données. En cas de litige, on peut aisément retrouver ces éléments ainsi que les personnes et les biens concernés et les messages échangés via le site.

Les envois postaux ne sont pas prévus pour garantir aux utilisateurs la sécurité des échanges et limiter le risque de tromperie. Dans le cadre d'un échange d'objets, l'envoi

postal est souvent source de litige. Le site ne saurait prouver et s'assurer de la bonne réception des biens échangés. C'est pourquoi, les échanges ne peuvent être réalisés que par le biais d'une rencontre réelle.

Enfin, un système d'évaluation des utilisateurs est mis en place pour favoriser la bonne conduite des utilisateurs et permettre la détection de troqueurs malveillants.

c. Description fonctionnelle

Vue d'ensemble des fonctionnalités et des utilisateurs

J'ai constitué un diagramme de cas d'utilisation (use case) permettant de visualiser les grandes fonctionnalités de l'application disponibles aux différents profils d'utilisateurs.

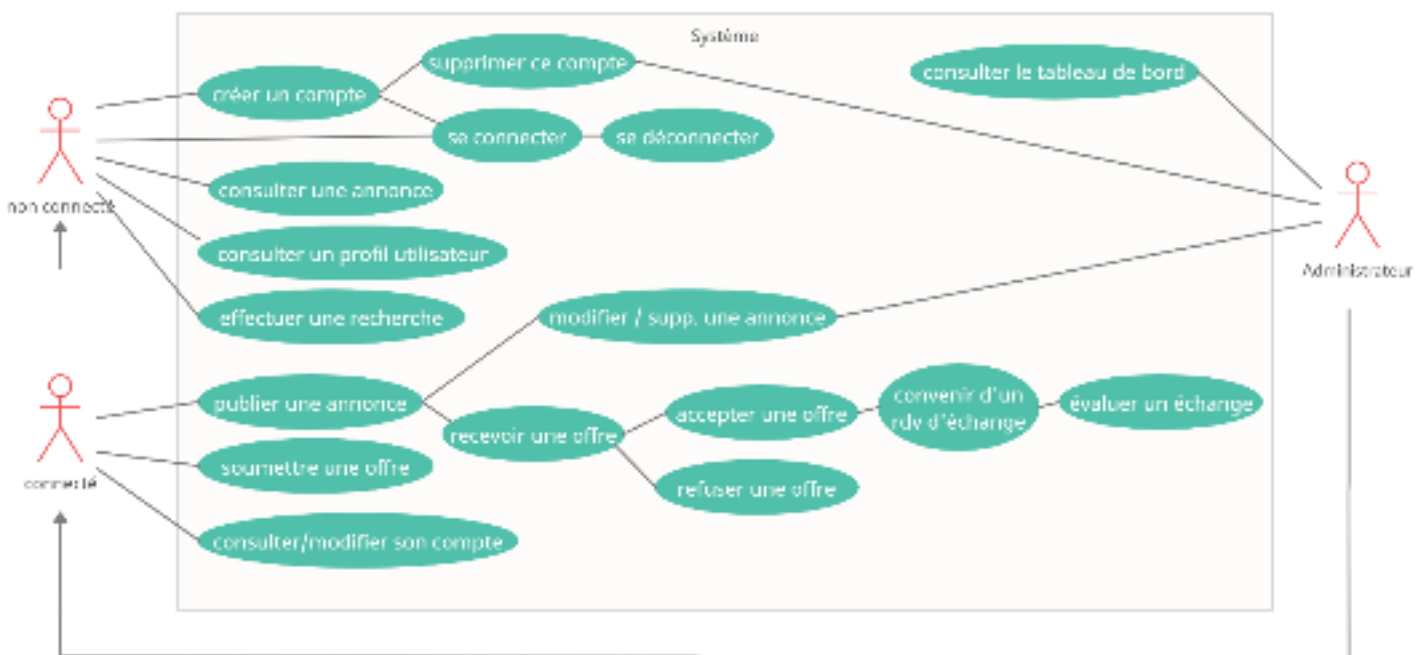


Diagramme de use case (vue d'ensemble des fonctionnalités)

On observe d'ores et déjà que l'application s'articule autour de quatre grands objets : l'utilisateur, l'annonce, l'offre et l'échange. De même, on identifie l'utilisateur du système par ses droits d'accès aux fonctionnalités. Ainsi, on distingue l'utilisateur connecté de celui qui ne l'est pas puis l'utilisateur qui possède un droit d'administration du site une fois authentifié en tant que tel.

Décrivons de manière plus détaillée les grandes fonctionnalités articulées autour de ces entités ainsi que leur accessibilité au sein de l'application.

PAGE D'ACCUEIL :

La page d'accueil est la page principale de l'application. Elle permet la visualisation des annonces les plus récentes et en mode connecté, les plus proches de l'adresse de l'utilisateur. Elle contient une barre de recherche qui permet de retrouver des annonces par mots-clés. Elle comporte les liens d'accès aux pages suivantes :

- Mon compte
- Résultats de recherche
- Inscription (si non connecté)
- Connexion (si non connecté)
- Rédiger une annonce

INSCRIPTION :

Un utilisateur peut s'inscrire et avoir un compte utilisateur. L'utilisateur doit obligatoirement renseigner à travers un formulaire son nom, son prénom, son email, son pseudo, sa date de naissance, son adresse (rue, ville, code postal, pays), son numéro de téléphone, son mot de passe et doit accepter les conditions générales d'utilisation du site.

CONNEXION :

L'utilisateur inscrit se connecte grâce à la combinaison email et mot de passe. L'authentification est nécessaire à l'accès aux pages "mon compte" et "rédiger une annonce".

MON COMPTE :

L'utilisateur connecté peut gérer son profil (nom, prénom, pseudo, email, téléphone, adresse, avatar, mot de passe).

Il peut également visualiser sa note générale sur 5 ainsi que les évaluations laissées par les troqueurs avec qui il a effectué des échanges (notes et commentaires).

Il comporte les liens d'accès aux pages suivantes :

- Mes annonces
- Mes offres
- Mes échanges

RECHERCHE :

La fonctionnalité de recherche d'un produit présente les résultats sous forme d'annonces cliquables qui permettent d'accéder à leur détail. Il est possible aussi d'effectuer de nouvelles recherches avec plus de filtres (localisation, pseudo du troqueur, date de parution).

RÉDIGER UNE ANNONCE :

L'utilisateur connecté peut publier une annonce grâce à un formulaire. Il doit renseigner obligatoirement le nom du bien, la description, la catégorie et la ville qu'il souhaite comme lieu d'échange. Il peut additionnellement renseigner un périmètre accepté autour de cette ville en km ainsi que des photos du bien.

MES ANNONCES :

Un utilisateur connecté peut consulter la liste des annonces qu'il a publiées ainsi que les offres associées. Il peut accepter ou refuser une offre qu'il a reçue. Les intitulés des annonces et offres sont les liens d'accès vers leur détail.

DETAIL D'UNE ANNONCE :

Un utilisateur du site (connecté ou non) peut visualiser le détail d'une annonce ainsi que quelques informations concernant le profil de son auteur (pseudo, note globale) avec un lien permettant d'accéder à plus de détails le concernant.

Si l'utilisateur est l'auteur de cette annonce, un lien lui permet d'accéder à la page "modifier l'annonce" et un autre lui permet de la supprimer.

En revanche, si l'utilisateur n'est pas l'auteur de l'annonce, un lien lui permettant d'accéder à la page "faire une offre" est disponible.

MODIFIER UNE ANNONCE :

L'auteur d'une annonce peut en modifier les éléments et ainsi la mettre à jour.

PROFIL D'UN UTILISATEUR :

Il est accessible à tous les visiteurs, connectés ou non. Il est destiné à visualiser quelques informations sur le profil d'un autre utilisateur, notamment sa note globale sur 5, les avis déposés (notes et commentaires), ainsi que la liste de ses annonces en cours et leur lien permettant d'accéder au détail.

MES OFFRES :

L'utilisateur connecté peut consulter la liste des offres qu'il a envoyées ainsi que les annonces associées. Il peut visualiser le statut de chaque offre (acceptée, refusée, sans réponse) et éventuellement se désister. En cas d'offre acceptée, un lien permet à l'utilisateur d'accéder à l'espace d'échange.

ECRIRE UNE OFFRE :

L'utilisateur connecté peut faire une offre sur une annonce publiée sur le site et ne faisant pas encore l'objet d'un échange. Cette page est semblable à "écrire une annonce", bien que plus minimaliste dans les renseignements à fournir avec uniquement le titre de l'offre, une description, la catégorie et d'éventuelles photos. Une fonctionnalité permettra à l'utilisateur d'importer les informations d'une offre ou d'une annonce qu'il a déjà effectuée.

DETAIL D'UNE OFFRE :

Seuls l'annonceur et l'offreur peuvent consulter le détail de l'offre. L'annonceur y verra un lien lui permettant d'accepter l'offre et d'ouvrir un espace d'échange et un autre lui permettant de la refuser. L'offreur, lui, y verra un lien lui permettant de se rétracter.

ESPACE D'ÉCHANGE :

Cet espace commun à l'annonceur et à l'offreur, comporte :

- une zone de messagerie instantanée permettant aux deux utilisateurs de s'accorder sur les conditions de leur rencontre
- une zone où l'annonceur renseigne une proposition d'adresse complète (avec nom de rue) et un créneau que l'offreur doit valider.

MES ÉCHANGES :

L'utilisateur connecté peut consulter la liste de ses échanges. Un échange comporte les intitulés des deux objets troqués (liens menant à leur détail), le statut (échange en discussion, conclu, terminé ou annulé) et les informations principales liées à cet échange (date, heure et adresse).

Si l'échange est en discussion, l'utilisateur pourra accéder à l'espace d'échange ou se rétracter.

En cas d'échange conclu, l'utilisateur voit le lieu et la date de celui-ci et a la possibilité d'annuler. On lui demandera alors d'indiquer le motif de son abandon parmi la liste suivante :

- Je ne suis plus intéressé par le produit
- Je ne souhaite plus échanger mon produit
- Le créneau ne me convient plus
- Autre

Si l'échange est terminé, l'utilisateur peut laisser une évaluation.

Si l'échange est annulé, le motif figure parmi les informations visibles.

d. Règlement Général sur la Protection des Données (RGPD)

En Europe, le traitement des données personnelles fait l'objet d'une réglementation, le Règlement Général sur la Protection des Données (RGPD) qui renforce le contrôle de l'utilisation des données des utilisateurs et leur permet d'en avoir la pleine maîtrise. En France, c'est la Commission Nationale de l'Informatique et des Libertés (CNIL) qui veille à son respect. En publiant un site Internet, j'ai ainsi l'obligation de :

- citer avec transparence quelles sont les données récoltées concernant les utilisateurs, leur but, leur utilisation, le lieu et la durée de leur conservation
- demander le consentement explicite des utilisateurs (en leur donnant le libre choix d'accepter ou de refuser).
- leur donner les moyens d'exercer leurs droits de consultation, rectification et suppression des données.

Pour le premier point, mon action a été de rédiger et d'intégrer une page spécifique nommée "Politique de confidentialité" dont voici un extrait :

Donnée personnelle	But et utilisation
Nom, prénom, numéro de téléphone	Données utilisées en cas de litige
Pseudo et mot de passe crypté	Authentification sur le site (sécurisation du compte)
Date de naissance	Confirmer la majorité de l'utilisateur. + Etude statistique du profil utilisateur
Email	Authentification sur le site + Communication avec l'utilisateur
Adresse postale	Mise en avant des produits de proximité
Objets troqués et acquis	Données utilisées en cas de litige
Adresse IP	Localiser l'utilisateur et lui présenter la version internationale du site (en anglais) s'il n'est pas situé en France
Distance entre le lieu de l'échange et celui du domicile	Etude statistique du profil utilisateur
Durée de conservation des données : 2 ans	
Lieu de stockage des données : serveur européen	
Acteurs ayant accès aux données : TrocLand	
Adresse mail pour infos et recours	

Cette page est accessible par un lien présent à trois endroits de l'application : sur le bandeau cookie s'affichant dès l'entrée de l'utilisateur sur le site, au niveau du formulaire d'inscription et dans le pied de page. Les deux premiers endroits sont également utilisés pour recueillir le consentement de l'utilisateur et le troisième est présent sur toute page du site.

Dès que l'internaute arrive sur l'application, je l'informe par une bannière "pop-up" que celle-ci peut stocker des cookies sur son navigateur (fichiers de données récoltées lors de sa navigation). Certains dits fonctionnels sont indispensables à l'utilisation du site (comme le cookie de session) et d'autres, optionnels, peuvent servir à personnaliser l'utilisation du site comme celui stockant l'adresse IP et sa localisation dans le but de proposer en page d'accueil les annonces les plus proches de l'utilisateur. Concernant les cookies, je cite les données récoltées, leur but et demande pour chacun l'accord de l'utilisateur.

Pour recueillir le consentement de l'utilisateur quant aux données stockées en base (côté serveur), j'ai disposé au niveau du formulaire d'inscription une mention "J'accepte les conditions générales d'utilisation du site et sa politique de confidentialité" comportant les liens vers les pages correspondantes. Initialement décochée, elle doit être confirmée par l'utilisateur pour pouvoir s'inscrire et utiliser les services du site.

Enfin, les utilisateurs peuvent demander à exercer un droit de regard sur les données récoltées les concernant en envoyant cette demande à l'adresse e-mail indiquée dans la page politique de confidentialité. Ils pourront également faire une demande de rectification ou de suppression par le même biais, ou pour certaines actions directement sur la page "Mon compte" (suppression de compte, modification des coordonnées, du pseudo, etc.)

e. Planning et suivi

Tâches à réaliser	Durée estimée (jours)	Durée réelle (jours)
ANALYSE ET CONCEPTION		
Rédaction du cahier des charges	7	7
Conception de diagrammes - MCD		
Maquettage de l'application	2	3
DEVELOPPEMENT		
Création des entités et des relations	2	2
Création de la base de données		
Sécurisation des propriétés		
Implémentation de données (fixtures)	2	2
Inscription et connexion	1	1
Écriture des fonctions d'accès aux données	5	4
Navigation (contrôleurs et vues)	1	1
Écriture des contrôleurs	10	14
Écriture des vues - Agencement et design	25	21
Gestion des rôles et permissions	1	1
TEST		
Utilisation fonctionnelle	3	1
DEPLOIEMENT		
Hébergement du site	1	X
	60	57

2 – Spécifications techniques

a. Méthode MERISE et modèles de données

La conception d'un projet nécessite de réfléchir à l'ensemble de l'organisation de son système d'informatisation. Je me suis appuyé sur la méthode MERISE (Méthode d'Étude et de Réalisation Informatique par les Sous-Ensembles ou pour les Systèmes d'Entreprise) pour mettre en place le modèle de mon application, le décrire et le représenter de manière formelle. Après étude du besoin, j'ai appliqué le principe de séparation des données et organisé celles-ci en plusieurs entités calquées sur les objets réels manipulés par l'application. Ces entités sont définies par des attributs et établissent des relations entre elles. L'étude de leur organisation aboutit à la construction de représentations graphiques, simples et accessibles, permettant à chacun de comprendre la structure de l'application : le MCD (Modèle Conceptuel des Données) puis le MLD (Modèle Logique des Données).

Description des entités et de leurs attributs :

Utilisateur

Chaque utilisateur possède un numéro d'identification, un nom, un prénom, un pseudo, un email, un mot de passe, une date de naissance, un numéro de téléphone et un avatar.

Annonce

Chaque annonce possède un numéro d'identification, une date de création, un titre, une description, un périmètre (distance acceptée entre la localisation de l'annonce et le lieu d'échange) et un statut (actif ou non).

Offre (proposition)

Chaque offre possède un numéro d'identification, un titre, une description, une date de création et un statut (actif ou non).

Adresse

Chaque adresse possède un numéro d'identification, un numéro et un nom de rue, un code postal, une ville, un département, une région, un pays, une latitude et une longitude.

Photo

Chaque photo possède un numéro d'identification et un nom.

Catégorie

Chaque catégorie possède un numéro d'identification et un nom.

Échange

Chaque échange possède un numéro d'identification, une date de création et une date fixée pour la rencontre de deux troqueurs.

Statut (de l'échange)

Chaque statut possède un numéro d'identification et un nom.

Message

Chaque message possède un numéro d'identification et un nom, un contenu, une date de création et un statut (lu/ non lu).

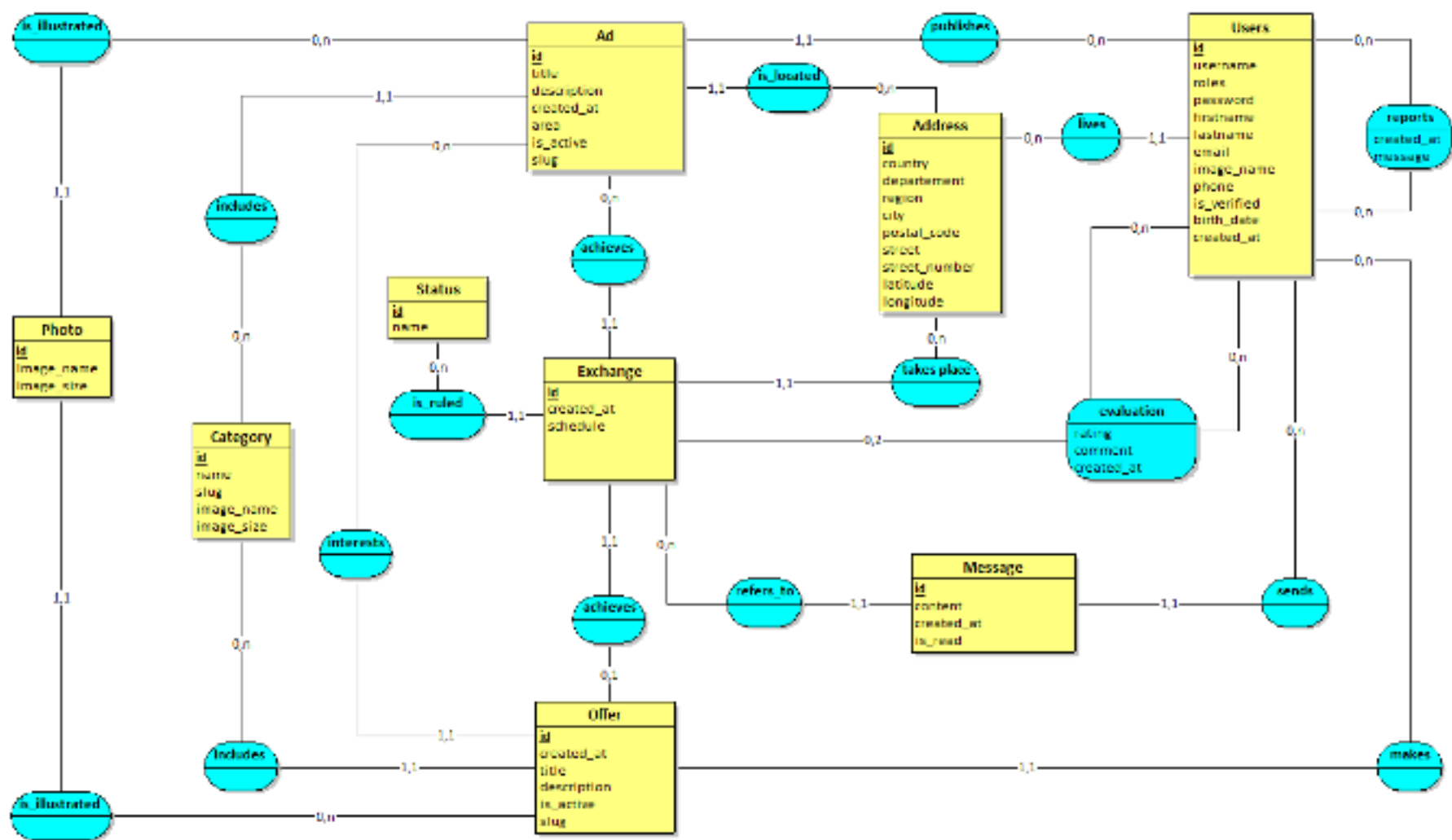


Schéma MCD réalisé avec le logiciel Looping

Description des relations et des cardinalités :

Un utilisateur peut publier des annonces et une annonce est publiée par un seul utilisateur (relation 1-N).

Un utilisateur peut faire plusieurs offres et une offre est faite par un seul utilisateur (relation 1-N).

Un utilisateur peut évaluer des échanges et un échange peut être évalué par deux utilisateurs (relation 2-N).

Un utilisateur possède une adresse et une adresse peut être celle de plusieurs personnes (relation N-1)

Un utilisateur peut envoyer des messages sur un espace dédié dès lors qu'il entre en négociation pour un échange et un message est envoyé par un seul utilisateur (relation 1-N).

Un utilisateur peut signaler plusieurs utilisateurs et un utilisateur peut être signalé plusieurs fois (relation N-N).

Une annonce est classée dans une seule catégorie et une catégorie peut regrouper plusieurs annonces (relation N-1).

Une annonce peut posséder des photos et une photo appartient à une seule annonce (relation 1-N).

Une annonce peut recevoir des offres et une offre est faite pour une seule annonce (relation 1-N).

Une annonce a une adresse et une adresse peut être reliée à plusieurs annonces (relation N-1).

Une offre est classée dans une seule catégorie et une catégorie peut regrouper plusieurs offres (relation N-1).

Une offre peut posséder des photos et une photo appartient à une seule offre (relation 1-N).

Un échange met en relation une annonce et une offre. Une annonce peut faire l'objet d'un seul échange, de même que l'offre (relation 1-1).

Un échange est le support de plusieurs messages et un message apparaît sur un seul espace d'échange (relation 1-N).

Un échange possède un statut et un même statut peut caractériser plusieurs échanges (relation N-1).

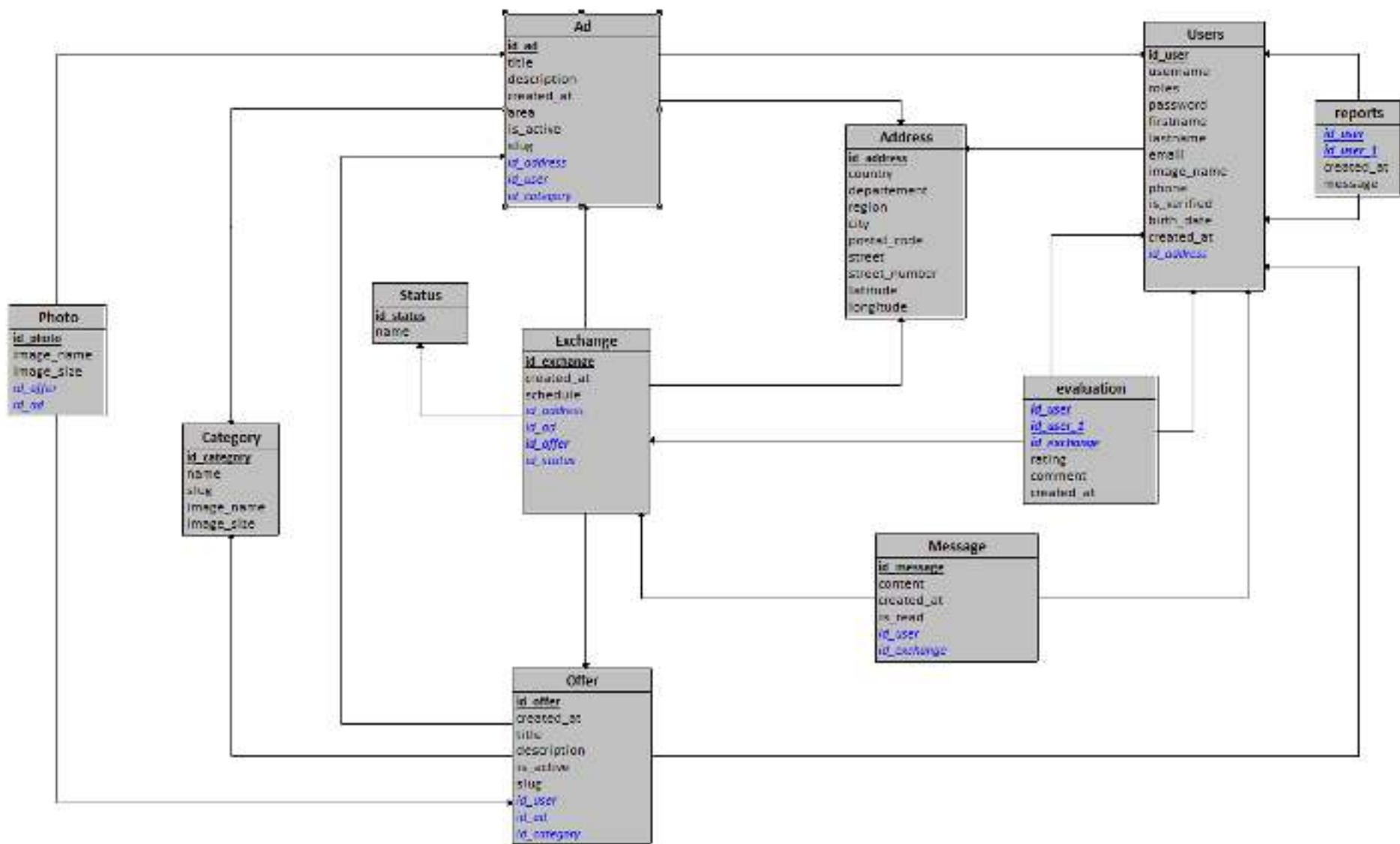





Schéma MLD réalisé avec le logiciel Looping

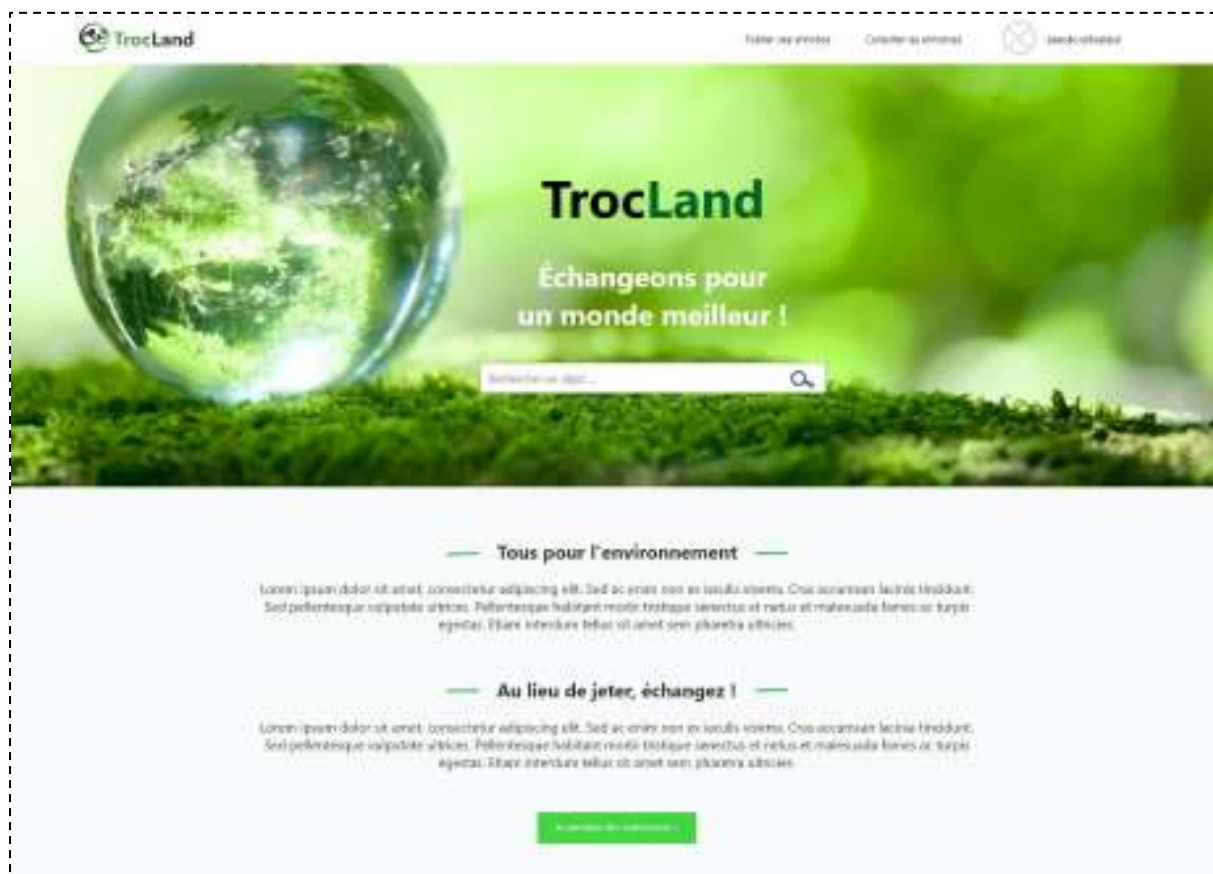
b. Charte graphique et typographie

La police d'écriture choisie pour l'ensemble du site est « Molengo », obtenue à partir du service Google Fonts. Concernant les couleurs, le choix s'est porté sur une palette de tons naturels, en accord avec l'éthique écologique du site web. L'identité de chaque couleur est représentée ici par son code hexadécimal.

Couleur	Aperçu
#F9FAFB	
#1D3C34	
#09AA2C	

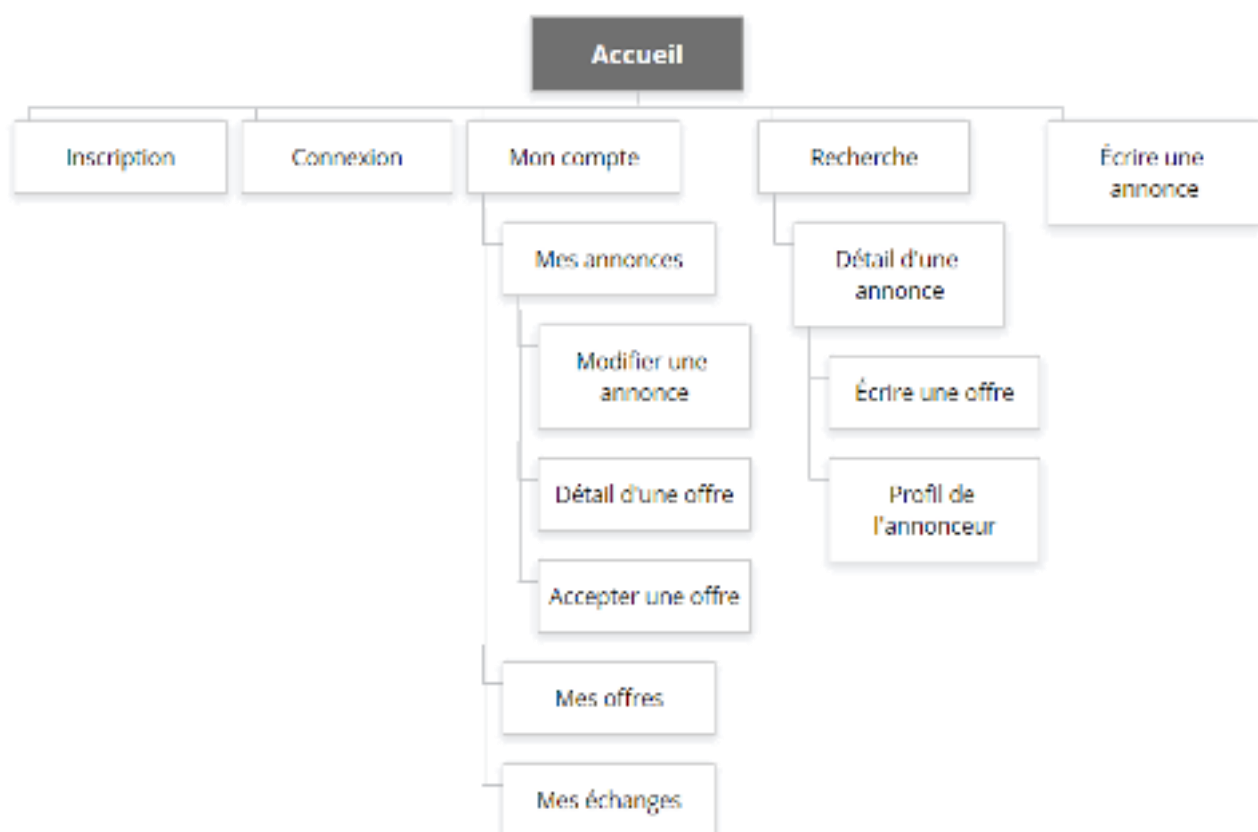
c. Maquettes (cf. complément en annexe)

Le maquettage est une phase importante du projet qui m'a permis de définir l'ambiance du site et l'agencement, l'accessibilité des différentes fonctionnalités selon leur degré de priorité. La maquette m'a aussi servi d'objectif à atteindre lors de la réalisation.



d. Arborescence de l'application

L'arborescence d'un site web représente l'organisation et la structure des informations et des contenus. Elle se présente sous forme d'un schéma qui caractérise les différents niveaux de navigation et permet de visualiser le parcours que pourrait avoir un visiteur lorsqu'il consulte notre site. Elle joue un rôle important pour avoir une expérience utilisateur claire et ergonomique, car en effet, un internaute aura plus de facilité à se repérer et à trouver ce dont il a besoin dans un site bien structuré qui possède des liens intuitifs et logiques entre chaque page. Les moteurs de recherche auront également une meilleure accessibilité à nos pages via leur processus d'indexation ce qui est donc bénéfique pour le référencement naturel

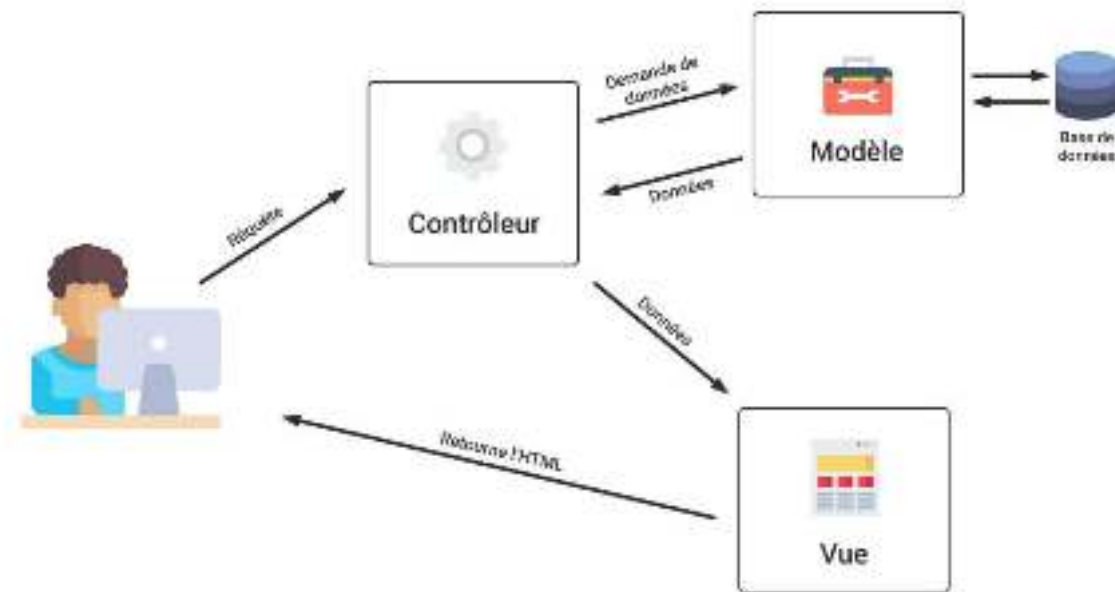


e. Ressources logicielles

	Visual Studio Code : éditeur de code open-source (sans licence donc libre d'utilisation) développé par Microsoft, il supporte de nombreux langages, est gratuit et multi-plateforme (Windows, Mac et Linux). Les nombreuses extensions disponibles font de cet éditeur un outil de choix pour les développeurs.
	PHP (version 7.4) (Hypertext Preprocessor) : c'est un langage de script populaire et open source, permettant l'utilisation du paradigme de la programmation orientée objet. Son exécution côté serveur permet de réaliser des pages web dynamiques dont le contenu varie en fonction d'informations (heure, nom de l'utilisateur, formulaire rempli par l'utilisateur, etc.)
	Symfony (version 5.2.9) : cadre applicatif (framework) basé sur le langage PHP, il donne une structure de base au projet et une méthode de travail intéressante. L'utilisation de Symfony se prête bien aux projets d'architecture MVC, d'autant plus que la documentation, les ressources et tutoriels sont nombreux...
	MySQL (version 5.7) : c'est un Système de Gestion de Base de Données Relationnelle (SGBDR). Une base de données organise les données persistantes d'une application dans des tables constituées de colonnes (champs) et de lignes (enregistrements). Ces tables font l'objet de relations logiques entre elles.
	Twig : moteur moderne de mise en page pour PHP. Sa syntaxe est concise et lisible. Il permet de nombreuses fonctionnalités : héritage multiple, blocs réutilisables, formatage des données de sortie ainsi qu'un échappement de sortie automatique et bien plus encore.
	Doctrine : ORM (Objet Relationnel Mapping) ou couche indépendante qui se base sur le modèle objet de l'application pour établir la communication entre celle-ci et la base de données.

f. Architecture logicielle

Pour cette application, j'ai fait le choix d'une architecture MVC (Modèle Vue Contrôleur) qui propose de séparer le code en plusieurs couches afin de gagner en maintenabilité et en flexibilité. Ainsi, trois parties se distinguent : le modèle gère les données des objets, la vue se concentre sur leur affichage tandis que le contrôleur, intermédiaire entre les deux, est le canal de contrôle de la transmission des données et de la navigation.



3 - Réalisation

a. Réalisation de la couche modèle

La couche modèle est l'ensemble des classes PHP des objets (entités) manipulés par l'application : Utilisateur, Annonce, Offre, Echange, ... Elles constituent la base même du projet. Elles doivent être créées en premier car elles sont le support utilisé par Doctrine pour créer et configurer la base de données. Je constitue le modèle objet de mon application en trois étapes.

Etape 1 : Je construis une entité via la commande du terminal :

```
php bin/console make:entity nom_entite
```

Etape 2 : Je définis les attributs propres à l'entité un à un en suivant les étapes de l'assistant de Symfony. Par exemple, pour ajouter un attribut « titre » à l'entité Offre, j'indique son type (string - chaîne de caractères), sa taille maximale (100) et sa nullabilité (non). Il est à noter que l'attribut Id (identifiant) est ajouté d'office à la création d'une entité.

```
New property name (press <return> to stop adding fields):
title
Field type (enter ? to see all types) [string]:
string
Field length [255]:
100
Can this field be null in the database (nullable) (yes/no) [no]:
no
```

Ainsi, Symfony crée une classe objet PHP du nom de l'entité. Celle-ci comporte dans cet ordre les attributs privés, le constructeur puis les getters/setters, méthodes publiques permettant d'accéder aux valeurs des attributs privés ou de leur en affecter, respectant ainsi le principe d'encapsulation.

Etape 3 : Une fois toutes les entités de l'application créées, j'établis les relations logiques entre elles en y ajoutant des attributs de type objet.

Exemple d'application : je souhaite établir une relation N-1 ou ManyToOne entre les entités Annonce et Catégorie. En effet, chaque annonce est classée dans une seule catégorie et une catégorie peut être affectée à plusieurs annonces. Par ailleurs, je souhaite qu'une annonce dispose de sa catégorie et qu'une catégorie dispose de ses annonces.

Il en résulte une modification au niveau du modèle :

- Dans la classe Annonce, je définis un attribut \$categorie de type Catégorie
- Du fait de la bidirectionnalité de la relation, dans la classe Catégorie, je définis un attribut \$annonces pouvant contenir une collection d'objets Annonce.

b. Réalisation de la base de données

Les données persistantes d'une application nécessitent d'être stockées en base. L'organisation de cette base de données est calquée sur le modèle objet. Ainsi, pour chaque entité, nous avons une table qui possède autant de colonnes que d'attributs et autant de lignes (enregistrements) que d'objets persistés.

On communique avec une base pour établir sa structure ou gérer ses données via des requêtes écrites en SQL (Structured Query Language – Langage de Requête Structuré). Dans un projet Symfony, c'est Doctrine, l'ORM intégré par défaut qui endosse ce rôle. L'intérêt de son utilisation est qu'il propose des méthodes permettant de générer automatiquement ces requêtes, qu'il les optimise selon le SGBD choisi et qu'il assure une sécurité contre les failles qui atteignent le plus souvent l'accès aux bases. Voici les étapes réalisées dans la construction de la base de données :

Etape 1 : Je configure Doctrine en renseignant l'URL d'accès à la base de données dans le fichier .env grâce à la variable d'environnement DATABASE_URL

```
DATABASE_URL="mysql://root@127.0.0.1:3306/trocland?serverVersion=5.7"
```

Etape 2 : Je transpose mon modèle objet à la base de données avec la commande

```
php bin/console doctrine:schema:update --force
```

A l'exécution de cette commande, Doctrine compare l'état actuel de la base de données avec l'état du modèle et transpose ce dernier. On utilisera cette commande également pour toute mise à jour du modèle qu'on voudrait appliquer à la base de données.

Comment Doctrine effectue cette transposition ? Pour explication, lorsque j'ai construit précédemment le modèle objet à l'aide du terminal, Symfony a généré dans les entités des annotations @ORM au-dessus de chaque classe et attribut. Reconnues par

Doctrine, elles lui sont indispensables dans l'établissement de l'architecture de la base de données et son paramétrage. En effet, c'est sur la base de ces annotations que Doctrine transpose le modèle objet de l'application à la base de données.

```
/**
 * @ORM\Entity(repositoryClass=OfferRepository::class)
 */
class Offer
{
    /**
     * @ORM\Column(type="string", length=100)
     */
    private $title;
```

Les entités et attributs ainsi annotés sont dits « mappés », autrement dit ils servent de patron à la création de la base de données. Les entités deviennent des tables tandis que les attributs deviennent des colonnes. Dans l'exemple-ci dessus, nous avons l'entité Offre et l'un de ses attributs \$titre tous deux annotés spécifiquement. Doctrine commande à la base de données la création d'une table « Offre » contenant une colonne « titre » de type VARCHAR(100), autrement dit dont les valeurs sont des chaînes de caractères à longueur variable n'excédant pas 100 caractères.

Les annotations peuvent très bien être écrites manuellement mais l'utilisation de l'assistant permet au développeur de se préserver de l'erreur et de gagner en rapidité. Aussi, il est tout à fait possible de créer des entités et des attributs non mappés si l'on ne souhaite pas leur persistance en base de données.

c. L'accès aux données

Les composants d'accès aux données représentent une couche de l'application permettant d'interagir avec la base de données. Ainsi, ils facilitent la modification, la suppression, la sélection d'objets existants mais également l'insertion de nouveaux.

Exemple d'insertion : Situation : L'utilisateur soumet un formulaire dans le but de déposer une nouvelle annonce. Je souhaite l'enregistrer en base de données.

Pour ce faire, je fais appel au gestionnaire d'entités de Doctrine ou EntityManager disponible à partir du contrôleur. Il est capable d'insérer en base de données un objet hydraté, autrement dit dont les attributs sont renseignés quand bien même ces derniers sont reliés à des tables différentes.

Dans l'exemple ci-après, je crée l'objet Annonce vide puis la fonction `handleRequest` du formulaire se charge d'hydrater l'annonce avec les données de la requête correspondant aux attributs de l'annonce. Je complète cette hydratation en renseignant l'auteur de l'annonce (l'utilisateur connecté) puis je le soumetts au Manager qui s'occupe de sa persistance en base de données en générant les requêtes appropriées.

```
//Je crée un objet de type Ad (Annonce)
$ad = new Ad();
$form = $this->createForm(AdType::class, $ad);
/*Le formulaire se sert des données de la requête pour hydrater
l'objet $ad, autrement dit affecter les valeurs soumises aux
différents attributs*/
$form->handleRequest($request);

$category = $ad->getCategory();
if ($form->isSubmitted() && $form->isValid())
{
    $ad->setUser($this->getUser());

    $slug = str_replace(" ", "-", $ad->getTitle());
    $ad->setSlug($slug);

    $em = $this->getDoctrine()->getManager();
    //Inscrit l'objet $ad à la liste des objets à enregistrer
    $em->persist($ad);
    //Enregistrement en base de données
    $em->flush();

    $this->addFlash("success", "Votre annonce vient d'être publiée.");
    return $this->redirectToRoute("my_ads");
}
```

Pour modifier une annonce, le code est sensiblement le même, à la différence qu'on récupère l'annonce en question en base de données au lieu d'en créer une nouvelle.

La sélection d'objets en base :

Avec Symfony/Doctrine, on distingue deux manières principales de récupérer des objets en base de données :

- Par les méthodes du Repository

Un Repository est une classe PHP générée par Symfony au moment de créer une entité, avec pour nom `EntiteRepository`. Ainsi, il existe autant de Repositories que d'entités. Ils héritent de la classe `Doctrine\ORM\EntityRepository`, qui propose des méthodes couramment utilisées pour récupérer des entités. En voici quelques-unes :

- **findOneBy**(array \$criteres) ou **findOneByX**(\$x) permettent de récupérer un objet en fonction d'un ou plusieurs critères. Si aucun objet ne correspond, la fonction retourne null.

Ex : Récupérer une annonce grâce à son id :

```
$ad = $adRepository->findOneById($id);
```

- **findAll**() permet de récupérer tous les objets d'une entité.

Ex : Récupérer toutes les catégories

```
$categories = $categoryRepository->findAll();
```

- **findBy**(array \$criteres) ou **findByX**(\$x) permettent de récupérer tous les objets d'une entité répondant au(x) critère(s), avec des options possibles.

Ex : Récupérer toutes les annonces de l'utilisateur en cours, triées par date décroissante.

```
$ads = $adRepository
->findBy(["user"=> $user->getId()], ["dateAd"=> "DESC"]);
```

Certaines requêtes ont nécessité que je code des fonctions personnalisées, comme celle ci-après permettant de récupérer une liste limitée d'annonces d'une certaine catégorie. J'utilise pour ce faire l'objet QueryBuilder, un générateur de requête en langage DQL (Doctrine Query Language) similaire au SQL, mais spécifique à Doctrine.

```
public function getLimitedByCategory($category, $limit)
{
    $query = $this->createQueryBuilder("a")
        ->innerJoin("a.category", "c")
        ->where("a.is_active = 1")
        ->andWhere("c.name = :category")
        ->orderBy("a.dateAd", "DESC")
        ->setMaxResults($limit)
        //On remplace le paramètre de la requête par sa valeur
        //Afin de se prémunir de l'injection SQL
        ->setParameter("category", $category)
        ;

    return $query->getQuery()->getResult();
}
```

- Par les getters (accesseurs)

Un getter ou accesseur est une méthode publique définie dans une classe entité permettant d'accéder à la valeur d'un attribut privé de cette entité. Les getters sont mis en place pour respecter le principe d'encapsulation. Ainsi, la valeur réelle d'un attribut sera lue selon le filtre du getter. Par exemple, celui-ci pourra retourner la valeur en lettres majuscules alors que la valeur réelle de l'attribut est en minuscules.

Dans l'exemple suivant, grâce à l'utilisation du getter, je récupère le nom de la catégorie d'une annonce. Lorsqu'il s'agit de récupérer un attribut de type objet, Doctrine se base sur le mode de récupération des données. Par défaut, il est LAZY (paresseux), autrement dit l'objet annonce ne disposera au niveau de son attribut

catégorie que l'id, grâce auquel Doctrine pourra effectuer une requête pour récupérer l'objet catégorie avec tous ses attributs (dont le nom). Si la configuration de cette récupération est en mode EAGER (avide), Doctrine se dispense d'effectuer une requête puisque l'attribut catégorie sera renseigné entièrement au niveau de l'objet annonce.

```
$category = $ad->getCategory()->getName();
```

d. Réalisation des contrôleurs

Le contrôleur représente la couche intermédiaire entre le modèle et la vue. Son rôle est de recevoir les requêtes HTTP en provenance du navigateur (client) et de lui transmettre une réponse appropriée. Support de la logique applicative, il permet de sécuriser la navigation et la circulation des données entre l'application et la base.

Je crée un contrôleur Symfony via la commande :

```
php bin/console make:controller AdController
```

Une méthode du contrôleur est appelée par à une ou plusieurs routes via l'annotation **@Route** de Symfony. Cette annotation définit la route à l'origine de la requête HTTP et lui affecte un nom de chemin (path).

Symfony permet d'intégrer les paramètres de requête à la route. Dans ce cas, ils y seront définis entre accolades et intégrés à la signature de la méthode appelée.

```
/**
 * @Route("/product/remove/{slug}", name="remove_ad")
 */
public function removeAd(AdRepository $adRepository, $slug): Response
{
```


Un contrôleur Symfony est une classe PHP héritant de la classe `AbstractController` qui définit les méthodes capables de retourner une réponse. Parmi elles, j'ai utilisé :

- **`redirectToRoute`** (string `$route`, array `$parameters` = [], int `$status` = 302) : retourne une réponse de redirection vers la méthode d'un contrôleur annotée par la route entrée en paramètre.

```
return $this->redirectToRoute("profile");
```

- **`redirect`**(string `$url`, int `$status` = 302) : retourne une réponse de redirection vers la vue dont l'adresse est entrée en paramètre.

```
return $this->render("user/report_success.html.twig", [  
    "user" => $user,  
    "form" => $form->createView()  
]);
```

- **`json`**(`$data`, int `$status` = 200, array `$headers` = [], array `$context` = []) : retourne une réponse au format JSON en s'occupant de l'encodage et de la sérialisation des données.

Pour retourner une réponse de manière plus générique, j'ai aussi utilisé l'objet `Response` :

```
return new Response("Accès refusé", 403);
```

e. Réalisation des vues

Les vues sont les fichiers permettant l'affichage des données à l'utilisateur et représentent l'interface graphique à travers laquelle il communique avec l'application. Les vues sont structurées grâce au langage HTML permettant l'affichage d'éléments tels que du texte, des boutons, des champs de formulaires, des images, ... mis en forme et stylisés grâce au langage CSS. Mais sans autre langage à l'appui, ces pages offrent du contenu statique, invariable.

J'utilise alors la technologie TWIG qui permet de rendre un affichage dynamique, autrement dit capable de s'adapter à plusieurs paramètres. Ainsi, on peut conditionner l'affichage de données, les filtrer via de nombreuses fonctions ou encore les afficher en boucle. Dans l'exemple ci-après, j'utilise une condition (if) pour l'affichage des photos d'une annonce dans la page de détails : si l'annonce a des photos, j'affiche la première photo de la collection en grand format, puis j'utilise une boucle for pour l'affichage en boucle des autres photos de la galerie en petit format sous l'image principale. Si l'annonce n'a pas de photos (else), l'image par défaut s'affiche.

```
{% if ad.photos|length > 0 %}
<figure class="preview-gallery-item border-light nb-4 mx-auto">
  
</figure>

<div class="flex mx-auto">
  {% for photo in ad.photos %}
    <figure class="gallery-item {{ (loop.first) ? "border-2-main-color" : "" }}">
      
    </figure>
  {% endfor %}
</div>

{% else %}
<figure class="preview-gallery-item border-light nb-4 mx-auto">
  
</figure>
{% endif %}
```



J'ai ajouté du dynamisme aux vues en codant quelques fonctionnalités en langage JavaScript :

- Réalisation de carousels affichant les annonces mises en avant sur la page d'accueil s'adaptant à la résolution de l'écran.
- Gestion des CollectionType : formulaire imbriqué pour l'ajout de photos lors de la création d'une annonce ou d'une offre avec gestion d'un nombre limite de photos à ajouter
- Création de modales
- Utilisation d'une librairie pour la gestion des cookies
- Animations de translation et de zoom sur la page d'accueil
- Gestion de la galerie photo sur les pages de détail d'annonce et d'offre.

f. Le responsive design : l'adaptation des vues aux mobiles et aux tablettes

Aujourd'hui, plus de la moitié de la population accède à Internet sur un appareil mobile. Seulement, un site conçu de manière traditionnelle pour desktop (écran d'ordinateur) n'offrira pas la même expérience d'utilisation et de navigation sur un smartphone ou une tablette. Selon Impact BND, 61% des utilisateurs mobiles ne reviendront jamais sur un site Web s'il n'est pas adapté aux mobiles. Il m'a donc paru essentiel en tant que développeur de considérer cette réalité et d'assurer l'adaptation automatique de mon application aux différentes résolutions d'écran en pratiquant ce qu'on appelle le Responsive Web Design (RWD). Ainsi, le site est ergonomique, sans aucune manipulation à faire au niveau du zoom et de la barre de défilement horizontal. Les blocs de contenus se redimensionnent automatiquement et voient leur disposition se réorganiser.



Maquettes pour l'adaptation mobile

Pour réaliser ceci, j'ai utilisé en CSS les Media Queries : règles permettant d'appliquer des propriétés CSS spécifiques en fonction d'une résolution d'écran. Dans l'exemple ci-après, si l'écran fait moins de 1050px, les éléments passeront d'une disposition en ligne à une disposition en colonne.

```
@media screen and (max-width: 1050px) {  
  flex-direction: column;  
  padding: 1rem;  
}
```

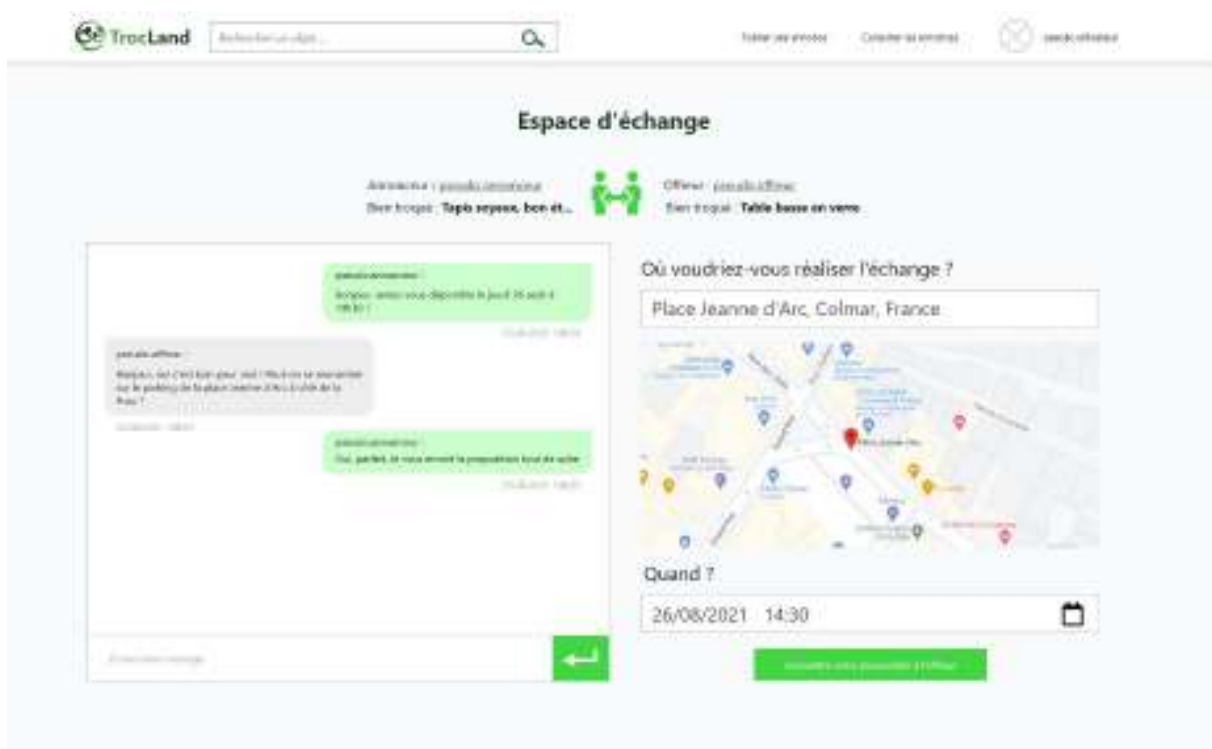
J'ai également utilisé la fonction `matchMedia` en JavaScript, notamment pour la gestion de l'affichage des carousels de la page d'accueil.

```
if (window.matchMedia("(max-width: 600px)").matches)  
{  
  numberOfCards = 1;  
}  
else if (window.matchMedia("(max-width: 800px)").matches)  
{  
  numberOfCards = 2;  
}  
else if (window.matchMedia("(max-width: 980px)").matches)  
{  
  numberOfCards = 3;  
}  
else if (window.matchMedia("(max-width: 1200px)").matches)  
{  
  numberOfCards = 4;  
}
```

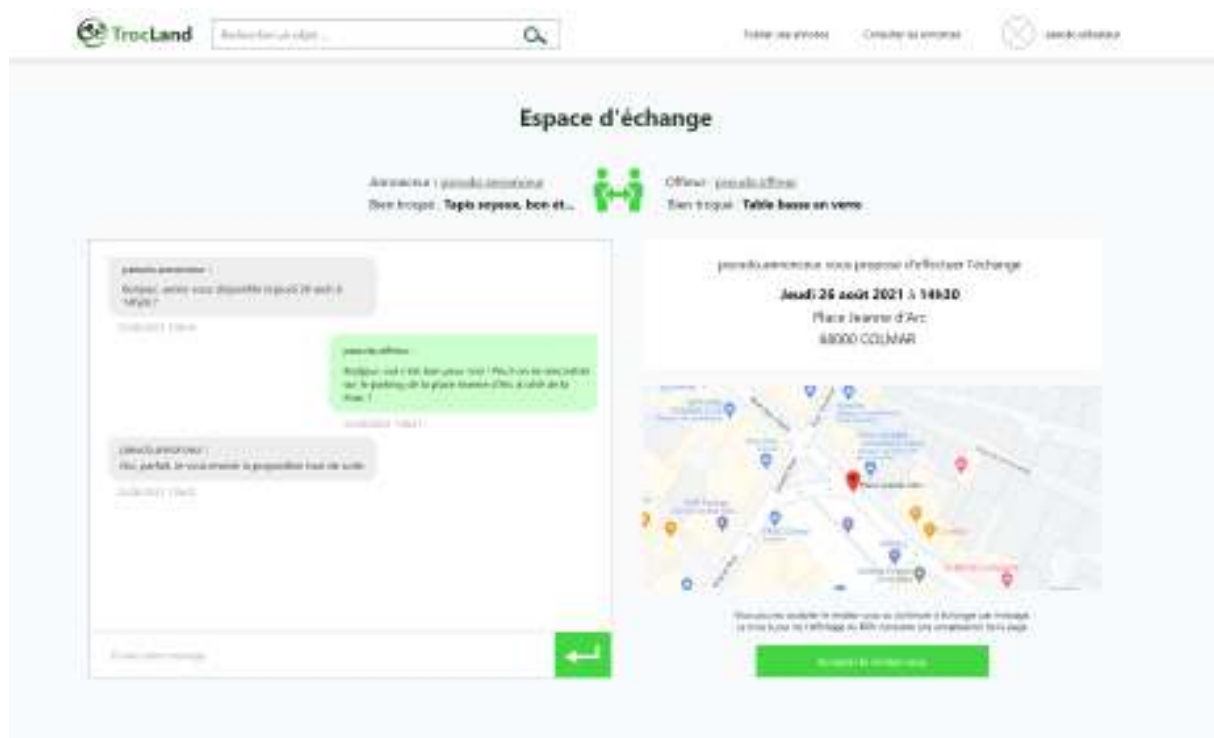
4 – Présentation d'une fonctionnalité : l'échange

Après avoir publié une annonce sur le site, l'utilisateur peut recevoir des offres. Au moment où il en accepte une, il accède à un espace d'échange unique auquel seuls lui et l'offreur peuvent accéder. Une fois cet espace ouvert, l'annonceur ne peut plus accepter d'autres offres sur cette annonce. Cet espace est dédié à l'échange de messages pour convenir d'une date et d'un lieu de rendez-vous. Des éléments sont plus ou moins présents dans cet espace selon que l'utilisateur est annonceur ou offreur :

- L'annonceur dispose d'une messagerie instantanée et d'un formulaire permettant de proposer à l'offreur un lieu et une date de rendez-vous.
- L'offreur dispose de la même messagerie et d'une section l'informant d'une éventuelle proposition de l'annonceur qu'il a la possibilité d'accepter. Si cette proposition ne le satisfaisait pas, il aurait la possibilité de négocier via la messagerie un autre rendez-vous avec l'annonceur qui devra soumettre une nouvelle proposition via le formulaire.



Espace d'échange (côté annonceur)



Espace d'échange (côté offreur)

La messagerie

L'aspect instantané de la messagerie vient de l'utilisation du langage JavaScript et plus particulièrement de la requête AJAX (Asynchronous JavaScript And XML) dite asynchrone. C'est une requête lancée au serveur qui, à la différence des autres types de requêtes, peut échanger des données avec la base sans recharger la page de l'utilisateur, ceci pour améliorer son confort d'utilisation.

Au moment où un utilisateur soumet un message dans le champ du formulaire dédié, je transmets via la méthode fetch le contenu du message et l'identifiant de l'échange au format JSON au contrôleur PHP (ExchangeController). Le format JSON (JavaScript Object Notation) est un format de données permettant de stocker des informations sous forme de couple clé/valeur. Inspiré du JavaScript, beaucoup de langages de programmation peuvent générer et lire ce format.

Ainsi du côté du contrôleur PHP, les données sont récupérées dans ce format, décodées puis stockées dans des variables. Je crée un objet Message dont je renseigne les attributs suivants : l'auteur (l'utilisateur connecté), le contenu et l'échange. La date du message et le statut « lu » (à false) sont hydratés par le constructeur au moment

de la création de l'objet. Je persiste l'objet Message entièrement hydraté en base de données puis j'envoie une réponse contenant cet objet converti au format JSON à la requête AJAX appelante qui va pouvoir dès lors injecter un élément HTML à la messagerie, sans rafraichissement de la page.

Partie JavaScript

```
fetch("{ (path("send_message")) }", {
  method: "POST",
  headers: {
    'Accept': 'application/json, text/plain, */*',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    messageContent,
    exchangeId
  })
})
.then((response) => response.json())
.then(data => {
  if(data.status == "OK")
  {
    let messageContainer = document.querySelector(".message-container");

    let messageClass = "message-box-1";

    messageContainer.innerHTML += `
      <div class="${messageClass}">
        <div class="message-content">
          <span>${data.user} :</span>
          <p class="m-0">${data.messageContent}</p>
        </div>

        <span class="date-message">${data.dateMessage}</span>
      </div>
    `;

    messageContainer.scrollTop(0, messageContainer.scrollHeight);
  }
})
.catch(e => console.log(e))
```



```

//Récupère les données au format JSON
$json = file_get_contents("php://input");

//Decode le JSON afin de pouvoir s'en servir
$data = json_decode($json);

if(isset($data))
{
    //Stockage des données
    $exchangeId = $data->exchangeId;
    $messageContent = $data->messageContent;

    $exchange = $exchangeRepository->findOneById($exchangeId);

    //Création du message à partir des données reçues
    $message = new Message();
    $message->setUser($this->getUser());
    $message->setContent($messageContent);
    $message->setExchange($exchange);
    //Ajout du message en base de données
    $em = $this->getDoctrine()->getManager();
    $em->persist($message);
    $em->flush();

    //Envoi de la réponse au format JSON
    return $this->json([
        "messageContent"=> $messageContent,
        "user"=> $this->getUser()->getUsername(),
        "dateMessage"=> $message->getCreatedAt()->format("d.m.Y à H\\h1"),
        "status"=> "OK",
    ]);
}

```

L'auto-complétion du champ de formulaire adresse par l'API Google Maps

Le formulaire côté annonceur demande à l'utilisateur de renseigner un lieu d'échange. J'ai fait le choix d'utiliser le service Places Autocomplete de l'API Google Maps pour assister l'utilisateur dans cette tâche et garantir l'authenticité de l'adresse renseignée.

Pour installer ce service, j'intègre à la vue une balise HTML `<script>` faisant référence au fichier JavaScript de l'API. Ses données me permettent de créer un objet Autocomplete que je relie au champ d'adresse. Ainsi, dès que l'utilisateur saisit du texte dans ce champ, une liste de lieux existants lui sont proposés par auto-complétion.

```

// Objet autocomplete
let autocomplete;

function initAutocomplete()
{
    if(addressInput)
    {
        autocomplete = new google.maps.places.Autocomplete(addressInput, {
            types: ["address"]
        })
        autocomplete.addListener("place_changed", onPlaceChanged)
    }
}

```

À la sélection de l'un d'entre eux, nombre de données de localisation sont récoltées à partir de l'objet Autocomplete (rue, ville, code postal, département, pays) et je renseigne grâce à elles les différents éléments du formulaire (<input>) cachés de l'utilisateur. La latitude et la longitude également disponibles me servent à initialiser un objet Map qui affiche une carte géographique focalisé sur l'emplacement choisi.

```

//Variable pour l'objet map
let map;

//Fonction permettant d'initialiser une carte à partir de coordonnées
function initMap(latitude, longitude)
{
    let mapPosition = { lat: latitude, lng: longitude }

    map = new google.maps.Map(document.getElementById("map"), {
        center: mapPosition, // Position initiale de la carte
        zoom: 15, // Niveau de zoom
    });

    //Place un marqueur sur la carte
    const marker = new google.maps.Marker({
        position: mapPosition, // Emplacement du marker
        map: map, // Objet map
    });
}

```

```

// lorsque la valeur du champ d'adresse change
function onPlaceChanged()
{
    // Récupère les informations du lieu
    let place = autoComplete.getPlace();

    // Si l'endroit récupéré n'est pas un lieu existant dans la base
    // de données de Google
    if(!place.geometry)
    {
        alert("pas de lieu reconnu");
    }
    // Le lieu saisi existe en base de données est donc correct
    else
    {
        // On stocke les résultats
        placeResults = place;

        // Obtention des coordonnées latitude longitude
        latitude = place.geometry.location.lat();
        longitude = place.geometry.location.lng();

        document.querySelector('.latitude').value = latitude;
        document.querySelector('.longitude').value = longitude;

        initMap(latitude, longitude);

        // Obtention du pays, de la ville, de la région, département,
        // code postal, rue et numéro de rue (si possible)
        for (let i = 0; i < place.address_components.length; i++)
        {
            let addressType = place.address_components[i].types[0];
            let val = place.address_components[i].long_name;
            let targetNode = document.querySelector("." + addressType);

            if(targetNode)
            {
                targetNode.value = val;
            }
        }

        if(addressInput)
        {
            // On enlève les erreurs car tout est bon
            addressInput.classList.remove("input-error");
            addressErrorNode.innerHTML = "";

            // On remplit le champ avec la valeur sélectionnée
            addressInput.innerHTML = place.name;
        }
    }
}

```

À la soumission du formulaire, je vérifie que le lieu entré appartient bien aux lieux reconnus par l'API et signale à l'utilisateur le cas échéant que l'adresse fournie n'est pas conforme et qu'il doit choisir obligatoirement un des lieux qui lui sont proposés.

Le choix d'une date et heure de rendez-vous (champ <input type= "datetime-local">)

Le contrôle des données entrées est effectué grâce à la balise @Assert\Range de Symfony à laquelle j'indique que le rendez-vous ne peut être pris qu'à partir du lendemain à 5h du matin.

```
/**
 * @ORM\Column(type="datetime", nullable=true)
 * @Assert\Range(
 *     min = "tomorrow + 5 hours",
 *     notInRangeMessage="Le rendez-vous doit avoir lieu après le {{ min }}."
 */
private $schedule;
```

Une fois les contrôles effectués et la proposition de rendez-vous enregistrée en base de données, l'offreur peut la visualiser et l'accepter tant que la date du jour reste inférieure à la date de la proposition. Au-delà, l'annonceur devra renouveler sa proposition avec une date différente s'ils veulent poursuivre l'échange.

```
$currentDate = new \DateTime("now");
$isExpired = ($exchange->getSchedule() < $currentDate) ? true : false;

//Si l'offre est expirée alors on redirige :
if($isExpired)
{
    $this->addFlash(
        "error",
        "La date de la proposition à expiré, merci de convenir d'une autre date.");

    return $this->redirectToRoute("create_exchange", [
        "offerId" => $exchange->getOffer()->getId(),
    ]);
}
```

5 – Sécurité

a. Présentation

Une application web fait transiter des données de la base de données vers l'utilisateur et inversement. Ainsi, un utilisateur mal intentionné pourrait transmettre des données mettant en péril le bon fonctionnement de l'application, sa structure et sa protection. Je vais exposer dans cette partie quelques failles de sécurité importantes et ce que j'ai mis en place pour les contrer. Il m'a été nécessaire de lister toutes les entrées utilisateurs : champs de formulaire, paramètres d'URL, headers de requêtes http et cookies... ainsi que la sécurité mise en place pour chacune.

b. Sécurisation de l'authentification

Sécuriser l'authentification est ce qu'il y a d'essentiel pour confirmer l'identité des utilisateurs et protéger leurs données personnelles mais également celles de l'application toute entière. Une attaque dite par force brute aura pour objectif de "cracker" un compte en soumettant de nombreuses combinaisons de connexion jusqu'à arriver à ses fins, profitant de la faiblesse des mots de passe choisis par les utilisateurs. Par exemple, un mot de passe de 6 caractères contenant lettres minuscules, majuscules et chiffres peut être cracké en 3 minutes seulement alors que s'il en avait contenu 4 de plus, il l'aurait été en 1 an. L'OWASP (Open Web Application Security Project) cite cette faille parmi les 10 plus courantes. Concernant les comptes de l'application, j'ai opté pour la validation suivante : 8 caractères contenant au minimum une minuscule, une majuscule, un chiffre et un caractère spécial, ce qui confère une protection satisfaisante, proportionnée aux privilèges accordés à l'utilisateur sur l'application.

J'ai également configuré le fichier `security.yaml` de Symfony au niveau de l'attribut `login_throttling` du firewall pour limiter le nombre de connexions possibles par minute (par défaut, ce nombre est à 5).

J'ai mis en place également un système de CAPTCHA sur les formulaires stratégiques de l'application : l'inscription et l'ajout d'une annonce. Il s'agit d'un test simple visant à vérifier que c'est bien un humain qui a soumis le formulaire et non un ordinateur.

c. Sécurisation des routes

Il est important de déterminer les droits d'accès à chaque page du site. Certaines sont accessibles à tout utilisateur, comme la page d'accueil et d'autres sont réservées aux utilisateurs connectés comme la page « mon compte ». Symfony permet la gestion de ces accès grâce à la configuration du fichier `security.yaml`. Ce système se base sur la notion de rôles attribués aux utilisateurs. Ainsi, par défaut, Symfony affecte le mot-clé `ROLE_USER` à tout utilisateur authentifié. Nous avons la liberté de lui accorder d'autres rôles en les faisant débiter par « `ROLE_` ». Le principe sera alors de déterminer quels sont les modèles d'URL dont on voudra restreindre l'accès. Par exemple, on peut décider que tous les URL contenant `/admin` seront réservés aux utilisateurs dotés du `ROLE_ADMIN`.

d. Protection contre les injections SQL

Le SQL est un langage utilisé pour communiquer avec une base de données. Des commandes sont destinées à modifier le contenu ou la structure d'une base de données comme `CREATE` pour la création de tables, `INSERT` pour l'ajout d'enregistrements (lignes) dans ces tables, `UPDATE` pour leur mise à jour et `DELETE` pour leur suppression. Il existe également une commande `SELECT` destinée à récupérer des informations à partir de la base de données et à les faire remonter à l'utilisateur.

Ces commandes sont transmises par l'application à la base de données à la suite d'une action de l'utilisateur, par exemple lors de la soumission d'un formulaire. Il est à savoir que ces commandes sont des chaînes de caractères composées en partie avec les données que l'utilisateur a lui-même soumises. Or, ces données peuvent être

malveillantes et contenir des instructions visant à compromettre la base de données. Ce type d'attaque s'appelle l'injection SQL.

La première protection a été de filtrer les données entrantes. Au sein d'une application Symfony, il est possible de mettre en place des contraintes de validation sur les attributs des entités, par l'annotation `@Assert`. Ainsi, on pourra par exemple vérifier qu'une valeur respecte des critères précis ou correspond à une certaine expression régulière (regex) qui définira quels caractères peuvent ou doivent être inclus et lesquels sont exclus.

Une protection supplémentaire a été de préparer les requêtes faites à la base de données, en mettant des paramètres à la place des données entrées par les utilisateurs. L'ORM Doctrine comporte une protection de base pour les requêtes classiques. Les requêtes personnalisées en DQL (Doctrine Query Language) et intégrant les données entrées par les utilisateurs sont paramétrées avec la méthode `setParameter()`.

e. Protection contre la faille XSS (Cross Site Scripting)

La faille XSS permet à un pirate d'injecter du code HTML et/ou Javascript dans une variable ou une base de données du site dans le but de le relier à un site distant (cross-site) qui pourra exécuter des requêtes malveillantes.

Pour protection, je filtre les données grâce aux contraintes de validation citées précédemment pour m'assurer dans un premier temps que les données entrées sont conformes au format attendu. Il est également nécessaire d'échapper les données sortantes et pour cela, le moteur de template (gabarit) TWIG fait un bon travail dans ce domaine car il protège et échappe toute valeur par défaut.

Les entêtes de sécurité Content Security Policy (CSP), directement implémentés dans Symfony avec `HttpFoundation` indiquent aux navigateurs des utilisateurs quelles sources sont autorisées à charger du contenu sur le site concerné, ce qui empêche l'intrusion de sources inconnues.

f. Protection contre les attaques CSRF (Cross-Site Request Forgery)

L'attaque CSRF consiste pour le hacker à faire en sorte qu'un utilisateur soumette à son insu les données d'un formulaire d'un site frauduleux vers un site auquel il est connecté pour y déclencher une action malicieuse.

Symfony apporte une protection contre cette faille par l'utilisation du jeton CSRF (CsrfToken), une chaîne de caractère hachée qui est stockée à la fois en session et intégrée à chaque formulaire du site dans un champ caché. Ainsi, lors de la soumission d'un formulaire, les deux jetons sont comparés et leur similitude prouvera que les données soumises proviennent bien d'un formulaire du site auquel l'utilisateur est connecté et non un autre.

6 – Recherche anglophone

Documentation de l'OWASP sur le contrôle d'accès défaillant (Broken Access Contrôle)

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Common access control vulnerabilities include:

- Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.
- Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
- Accessing API with missing access controls for POST, PUT and DELETE.
- Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.

- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.
- CORS misconfiguration allows API access from unauthorized/untrusted origins.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

How to Prevent

Access control is only effective in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- Except for public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).
- Stateful session identifiers should be invalidated on the server after logout. Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.

Developers and QA staff should include functional access control unit and integration tests.

Traduction :

Le contrôle d'accès veille à ce que les utilisateurs ne puissent pas agir en dehors des autorisations prévues. Les failles dans ce domaine entraînent la divulgation d'information non autorisée, la modification, la destruction de toutes les données ou l'exercice d'une fonctionnalité qui sort du cadre autorisé. Les vulnérabilités courantes du contrôle d'accès comprennent :

- La violation du principe du moindre privilège ou du refus d'accès par défaut. Ainsi, l'accès est accessible à tous alors qu'il ne devrait être accordé qu'à des rôles ou des utilisateurs particuliers.
- Le contournement des vérifications d'accès se fait en modifiant l'URL (falsification de paramètres, navigation forcée), ou en modifiant l'état interne de l'application voire la page HTML, ou en utilisant un outil d'attaque modifiant les requêtes API.
- Le fait d'autoriser la consultation ou la modification du compte de quelqu'un d'autre, en fournissant son identifiant unique
- L'accès à l'API avec des contrôles d'accès manquants pour POST, PUT et DELETE.
- Le dépassement de privilège, comme le fait de pouvoir agir en tant qu'utilisateur sans être connecté ou agir en tant qu'administrateur lorsqu'on est connecté en tant qu'utilisateur.
- La manipulation de métadonnées, telle que la relecture ou la falsification d'un jeton de contrôle d'accès JSON Web Token (JWT), ou la manipulation d'un cookie ou d'un champ caché pour augmenter les privilèges ou abuser de l'invalidation JWT.
- La mauvaise configuration de CORS permet l'accès à l'API à partir d'origines non autorisées/non approuvées.
- Le fait de pouvoir forcer la navigation vers des pages soumises à authentification en tant qu'utilisateur non authentifié ou vers des pages privilégiées en tant qu'utilisateur standard.

Comment se prémunir

Le contrôle d'accès trouve son efficacité dans le code mis en place côté serveur ou les API server-less, privant les attaquants de modifier les vérifications de contrôle d'accès ou métadonnées.

- Tout refuser par défaut, à l'exception des ressources publiques
- Implémenter des mécanismes de contrôle d'accès une seule fois et les réutiliser dans toute l'application, notamment en minimisant l'utilisation du partage de ressources cross-origin (CORS).
- Les contrôles d'accès aux modèles doivent imposer la propriété des enregistrements plutôt que d'accepter que l'utilisateur puisse créer, lire, mettre à jour ou supprimer n'importe quel enregistrement.
- Désactiver la liste des répertoires du serveur Web et s'assurer que les métadonnées de fichier (par exemple, .git) et les fichiers de sauvegarde ne sont pas présents dans les racines Web.
- Consigner les échecs de contrôle d'accès, alerter les administrateurs le cas échéant (par exemple pour les échecs répétés).
- Les identifiants de session stateful doivent être invalides sur le serveur après la déconnexion. Les jetons JWT stateless devraient plutôt être de courte durée afin de minimiser la fenêtre d'opportunité pour un attaquant. Pour les JWT de longue durée, il est fortement recommandé de suivre les normes OAuth pour révoquer l'accès.

Les développeurs et le personnel d'assurance qualité doivent inclure une unité de contrôle d'accès fonctionnel et des tests d'intégration.

Conclusion

Je suis heureux d'avoir mené le projet TrocLand qui représente mon humble geste pour la planète. J'espère que des millions de troqueurs l'adopteront bientôt et me permettront de le faire évoluer.

Professionnellement, j'ai appris l'importance d'une bonne analyse des besoins, d'une conception et d'une gestion solides dans le développement d'un projet. La partie réalisation m'a apporté une meilleure connaissance des technologies Symfony, Doctrine et Twig, une nette progression dans plusieurs langages - PHP, SQL, HTML, CSS, Javascript - et une expérience enrichie dans la mise en place d'une logique applicative.

J'espère poursuivre en développant d'autres projets et m'initier à la création d'applications mobiles qui va bientôt devenir un indispensable pour le développeur.

BIBLIOGRAPHIE

Sites web

Documentation Symfony : symfony.com, symfonycasts.com

Documentation PHP : www.php.net

Documentation Doctrine : www.doctrine-project.org

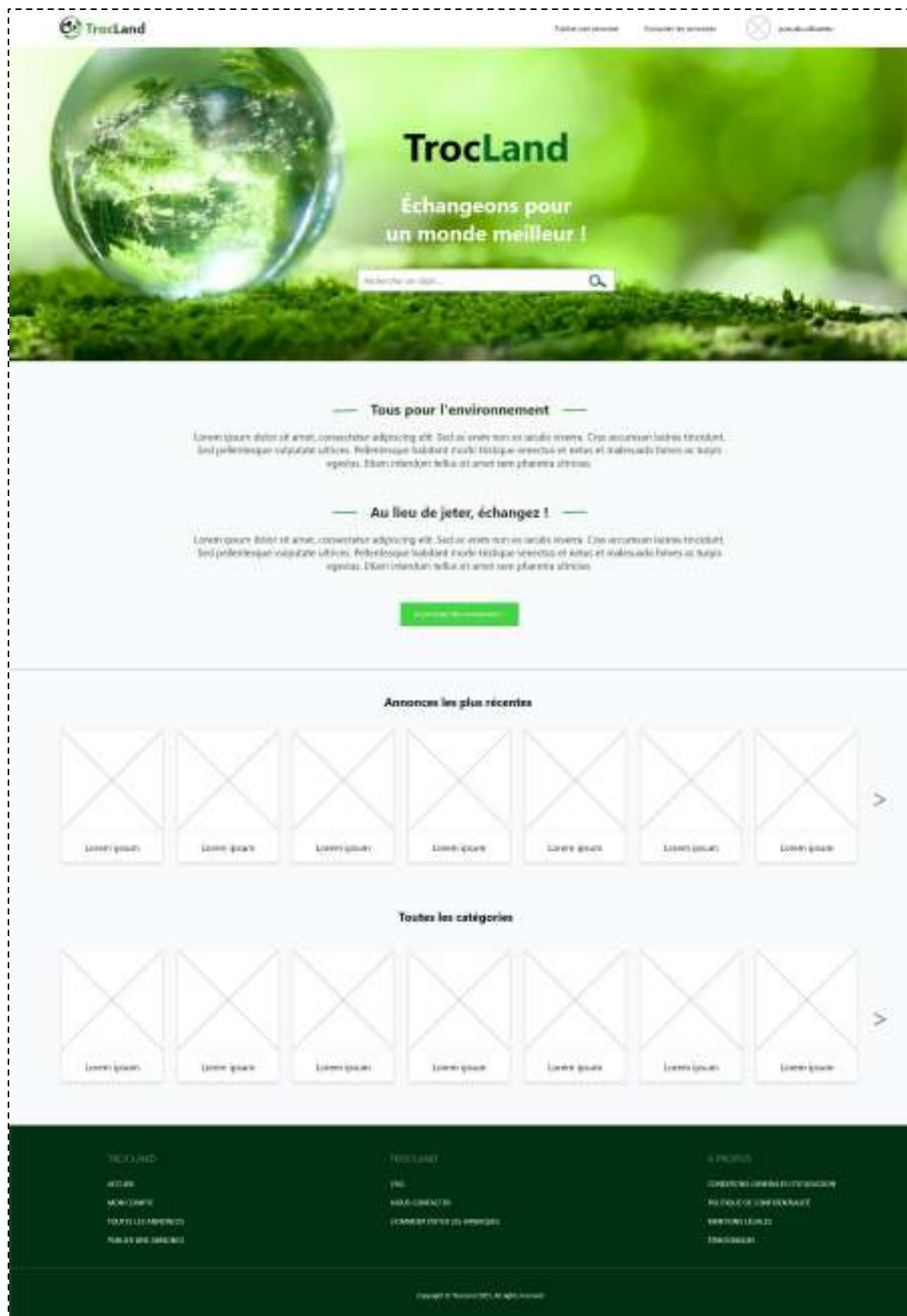
Documentation TWIG : twig.symfony.com

Documentation HTML, CSS, JavaScript : www.w3schools.com, developer.mozilla.org

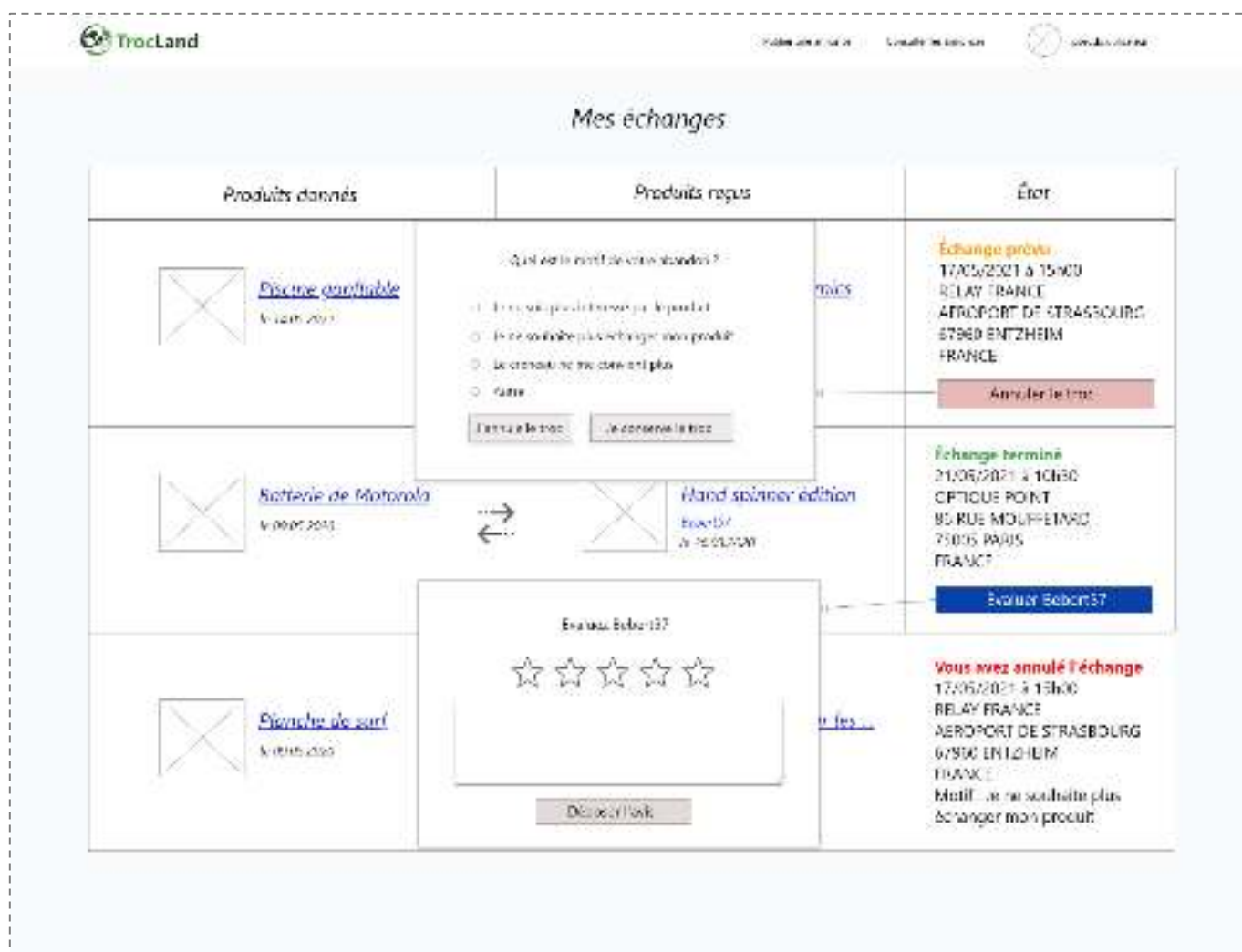
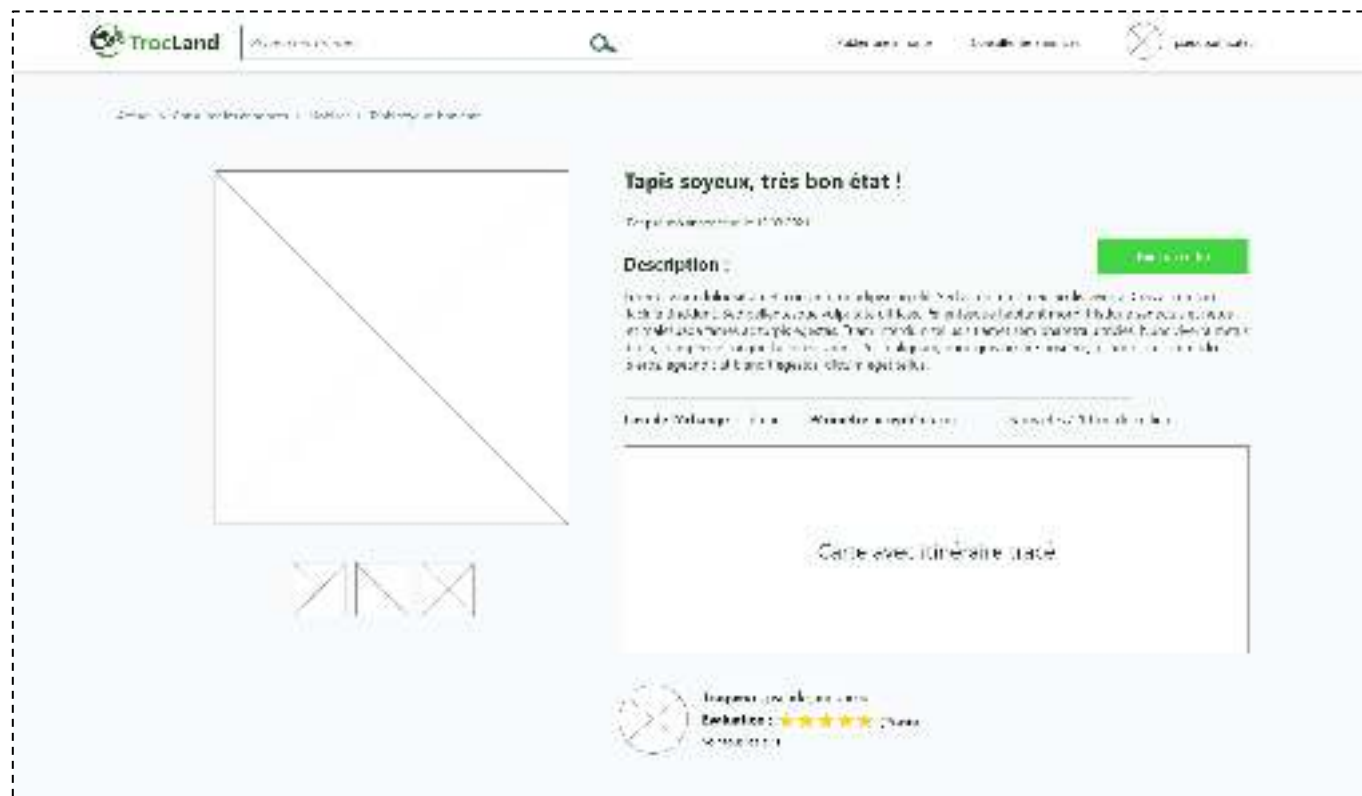
Plateforme d'entraide pour développeurs : stackoverflow.com, developpez.net

Fournisseur d'icônes : fontawesome.com

ANNEXES



Maquette de la page d'accueil



Maquettes "Détail d'une annonce" et "Mes échanges"

Cahier des charges

Importance	Catégorie	Nom	Description
2000	Gestion des utilisateurs	Se connecter	En tant qu'utilisateur, je peux me connecter sur la plateforme avec un login (adresse mail) et un mot de passe.
1950	Gestion des utilisateurs	S'inscrire	En tant qu'utilisateur, je peux m'inscrire sur la plateforme. Je peux aussi que l'inscrit devienne plus unique sur toute la plateforme. Je peux accepter que mes informations administratives. Je peux ne pas être pas désactivé de mon compte. Si la création du profil est validée, l'utilisateur est obligé avec la page d'accueil en mode connecté.
1900	Navigation	Page d'accueil	La page d'accueil du site doit être chargée automatiquement et aucune ressource n'est indiquée dans l'uri. Elle présente : - les nouvelles articles avec le flux d'écriture web le plus proche du domaine de l'utilisateur (la) - chaque catégorie avec quelques articles mis en avant - les pages d'affichage du site
2000	Gestion des livres	Lire une ressource	En tant qu'utilisateur je peux consulter une ressource parmi les ressources avec plusieurs offres. Avec description : - catégorie - localisation - période de disponibilité - le nombre d'ajout en cours ou le nombre de la requête
1800	Navigation	Menu	Un site web le logo de la rubrique l'utilisateur sur la page d'accueil Je peux afficher une ressource par nom d'article Je peux filtrer mes connexions / désconnexions Si je suis connecté je pourrai également : - accéder à mon profil - accéder à mes annonces - offre (à la demande) -> notification lors d'une nouvelle offre disponible
1750	Gestion des utilisateurs	Se déconnecter	En tant qu'utilisateur connecté, je peux me déconnecter, le site alors renvoie vers la page d'accueil en mode déconnecté.
1700	Gestion des utilisateurs	Afficher le profil d'un requête	En tant qu'utilisateur, je peux afficher le profil d'un autre utilisateur : (seuls, moyens, le nombre d'objets, liste des objets, commentaires, dates des annonces en cours (liste par date de démarrage)
1650	Gestion des utilisateurs	Consulter et modifier mon profil	En tant qu'utilisateur, je peux consulter mes informations de profil : prénoms, nom, adresse, e-mail, numéro de téléphone, mot de passe, ville, pays de provenance inspiration (le nombre d'objets, liste de mes objets publiés, commentaires, notes avec possibilité de modifier les publications, les publications)
1600	Gestion des utilisateurs	Suspension non complète	En tant qu'utilisateur, je peux suspendre mon compte. Dans ce cas, je suis déconnecté et redirigé sur la page d'accueil.

1500	Consultation des offres	Ajouter une annonce	En tant qu'utilisateur, je peux mettre en ligne une annonce de type : pour info, je demande les informations suivantes : titre, description, catégorie, lieu de l'annonce (ville), téléphone, e-mail, photos
1450	Consultation des offres	Modifier une annonce	En tant qu'utilisateur, je peux modifier une de mes annonces en cours (ou des infos) tant qu'aucune offre n'est faite dessus
1400	Consultation des offres	Supprimer une annonce	En tant qu'utilisateur, je peux supprimer une de mes annonces en cours. Si elle possède des offres, les offres reçoivent une notification d'info.
1300	Consultation des offres	Envoyer une offre	En tant qu'utilisateur, je peux faire une offre sur un article Pour cela, je propose un article en échange avec les informations suivantes : titre, description, catégorie, photos
1250	Consultation des offres	Recevoir des offres	En tant qu'utilisateur, je peux recevoir des offres pour une de mes annonces. L'icone "Mes annonces" indique la réception d'une nouvelle offre (indicateur numérique : 1, 2, ...)
1200	Consultation des offres	Consulter une offre	En tant qu'utilisateur, je peux consulter la liste des offres reçues en cliquant sur "mes annonces" et accéder au détail de chaque offre. En tant qu'offreur, je peux consulter la liste de mes offres en cliquant sur "mes offres" et accéder aux détails
1150	Consultation des offres	Accepter une offre	En tant qu'utilisateur, je peux accepter une offre. Dans ce cas, je peux accéder à l'espace d'échange et proposer un rendez-vous à l'offreur.
1100	Consultation des offres	Réfuser une offre	En tant qu'utilisateur, je peux refuser une offre. Dans ce cas, l'offre disparaît de la liste et l'offreur reçoit une notification qui l'en informe.
1050	Consultation des offres	Confirmer/Refuser mon offre	En tant qu'offreur, je peux accepter la proposition de RDV de l'utilisateur OU je decline mon offre (et je peux indiquer un motif)
950	Consultation des offres	Ajouter le détail d'une annonce	En tant qu'utilisateur, je peux afficher le détail d'une annonce : titre, description, lieu de mise en ligne, ville, téléphone. Je peux accéder à "Toutes les annonces", aux annonces de la rubrique concernée et aux annonces de la ville/région affichée. En tant que lecteur, je peux accéder à "modifier mon annonce" et je peux supprimer mon annonce. En tant qu'offreur, je peux accéder à "faire une offre"
850	Administration	Bannir ou supprimer un utilisateur	En tant qu'administrateur, je peux supprimer ou suspendre un compte utilisateur. Tous les textes/offres proposés par cet utilisateur sont alors annulés. L'utilisateur ne peut plus se connecter avec ses anciens identifiants
700	Administration	Modifier ou supprimer une annonce	En tant qu'administrateur, je peux modifier une annonce (pas de classement de catégorie) ou la supprimer

600	Navigation	Pageation	Sur la page permettant de faire les tests suivant des critères de recherche, j'affiche en nombre d'entrées maximum (15) et j'accède aux autres pages résultant sur l'interactivité de liens mémorisés.
500	Interpreting User Design	Variation mobile	Les fonctionnalités sont accessibles depuis un seul appareil de type smartphones connecté au web.
400	Sécurité	Sessions utilisateur de 30 min	L'utilisateur doit être déconnecté automatiquement après 30 minutes d'inactivité.
300	Gestion des utilisateurs	Se connecter de mot	En tant qu'utilisateur, je peux choisir d'entrer soit mon login et mon mot de passe sur mon ordinateur pour ne pas avoir à le saisir à la connexion suivante.
200	Gestion des utilisateurs	Mot du passe oublié	En tant qu'utilisateur, je peux faire une demande de réinitialisation de mot de passe. La plateforme dirige vers un écran de capture du nouveau mot de passe.
150	Gestion des mots	Modifier un mot de passe	En tant qu'utilisateur, je peux modifier un autre utilisateur après avoir effectué un déverrouillage.
100	Gestion des mots	Impression une annonce	En tant qu'utilisateur, je peux faire une offre en important les données d'une de mes annonces.
50	Administration	Accéder aux statistiques	En tant qu'administrateur, je peux modifier accéder aux statistiques du site : <ul style="list-style-type: none"> nombre de mots effectués (période, région) nombre d'utilisateurs inscrits nombre d'utilisateurs connectés cartographie des adresses distance moyenne de déplacement pour effectuer un échange taux de réservations