

Projet réalisé dans le cadre de la présentation au
Titre Professionnel Développeur Web et Web Mobile
Présenté et soutenu par Clément CANALE

DOSSIER DE PROJET

Février 2023

Table des matières

1.	RÉSUMÉ DU PROJET	3
2.	CAHIER DES CHARGES	4
	Objectif du site	4
	Charte graphique.....	4
	Arborescence.....	5
	Description fonctionnelle	6
	Fonctionnalités particulières	6
3.	LISTE DES COMPÉTENCES.....	7
	CP.1 Maquetter une application	7
	CP.2 Réaliser une interface utilisateur web statique et adaptable	8
	CP.3 Développer un interface utilisateur web dynamique	8
	CP.5 Créer une base de données.....	8
	CP.6 Développer les composants d'accès aux données.....	9
	CP.7 Développer la partie back-end	10
4.	SPÉCIFICATIONS TECHNIQUES.....	11
	Technologies du frontend	11
	Technologies du backend	11
	Technologies pour le développement.....	12
5.	VEILLE ET SECURITÉ	13
6.	RÉALISATION DU CANDIDAT.....	15
	Fonctionnement de Symfony	15
	Les vues	15
	Les modèles	16
	Les contrôleurs	18
	Fonctionnalités importantes du site	19
	Signalement de contenu indésirable.....	19
	Note des sujets	23
	Expérience utilisateur et référencement	28
7.	JEU D'ESSAIE.....	32
8.	RESSOURCES EN ANGLAIS	35
9.	CONCLUSION	38

1. RÉSUMÉ DU PROJET

Le site Dev In a été conçu pour être une plateforme d'information et d'échange pour les développeurs web sous une forme de réseau social.

Il y a deux sections distinctes dans le site :

- ➔ Une partie avec des articles créés par le fondateur du site sur différents thèmes et technologies autour du développement web
- ➔ Une partie avec des sujets postés par la communauté dans différentes catégories créées par le fondateur

L'utilisateur non inscrit peut lire les différents articles et sujets postés.

Pour interagir, l'utilisateur doit s'inscrire et il pourra dès lors poster des sujets, ajouter des sujets et des articles dans ses favoris, suivre d'autres utilisateurs, poster des commentaires et faire des signalements de contenu indésirable ou bien noter des sujets postés.

Le site a également une partie administrateur pour pouvoir écrire les articles et modérer les utilisateurs, les sujets postés et les commentaires.

Technologies utilisées :

Le site a été développé avec le framework PHP Symfony.

La partie front-end est conçue en TWIG et la partie back-end en PHP.

Le style a été fait en SCSS.

La base de données a été créée avec MySQL.

Dépôts du code sur Github pour le versionner.

Visual Studio Code comme éditeur de code.

Utilisation de Nifty pour la gestion de projet, lister les tâches en cours et à venir.

2. CAHIER DES CHARGES

Objectif du site

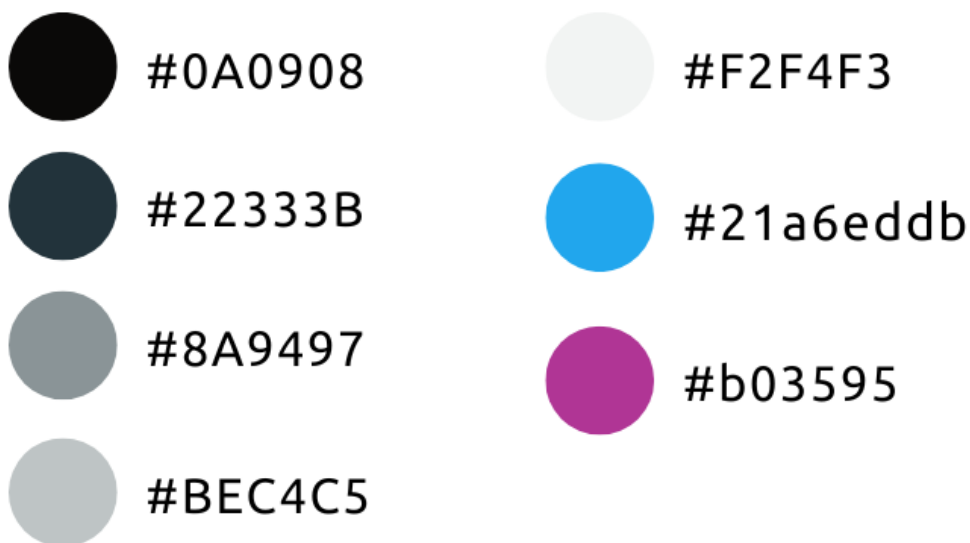
DevIn est un réseau social d'information sur le développement web dans lequel on trouve des articles sur les différents langages et technologie en rapport avec ce métier.

Le site se veut être une bibliothèque pour rendre accessible et au même endroit tout ce qui concerne de développement web Fullstack avec la possibilité de demander de l'aide ou de partager ses astuces avec la communauté.

Il s'adresse aux développeurs web aussi bien débutants que expérimentés et qui veulent rejoindre une communauté d'échange et de partage autour de ce métier.

Charte graphique

Les couleurs du site sont basés sur un dégradé de gris, une couleur primaire bleu et une secondaire le violet.



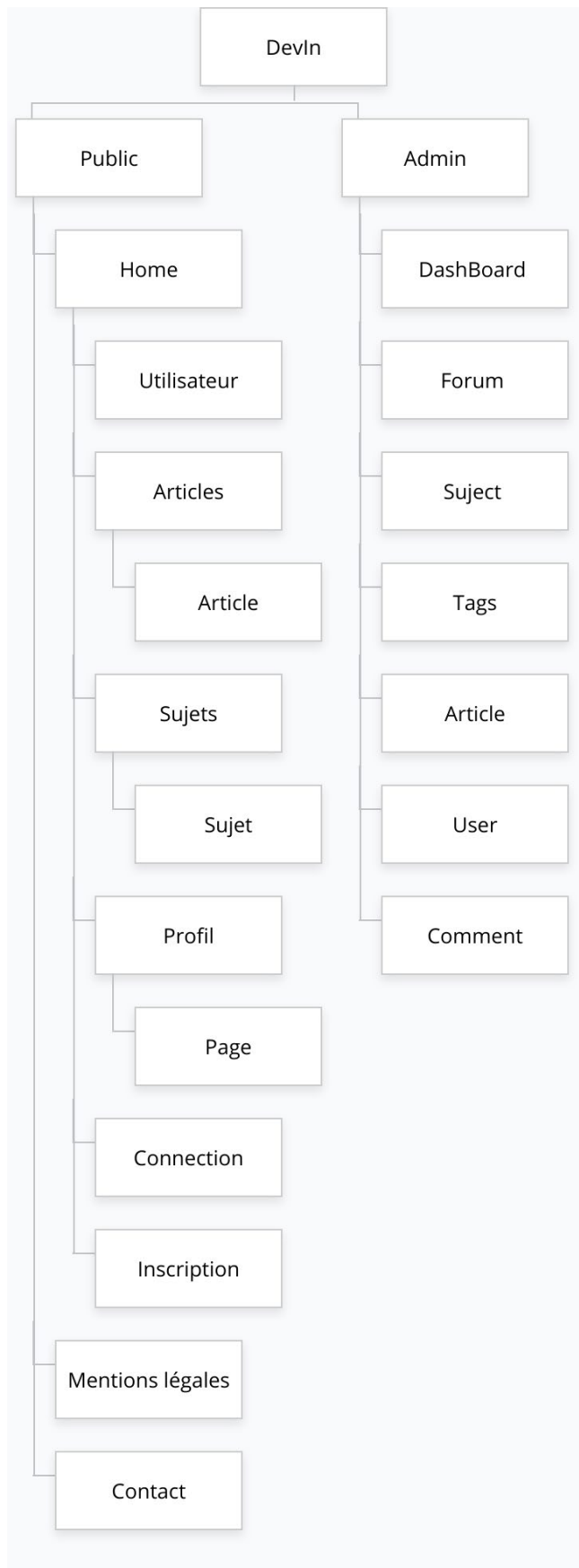
Les polices utilisées viennent de Google Fonts avec pour les titres

Lobster

Et pour les textes

Righteous

Arborescence



Description fonctionnelle

Fonctionnalités du visiteur :

- ➔ Afficher les utilisateurs du site
- ➔ Afficher les articles
- ➔ Afficher les sujets
- ➔ Afficher les commentaires

Fonctionnalités des utilisateurs inscrit :

- ➔ Poster un sujet avec possibilité de le modifier et de le supprimer
- ➔ Poster un commentaire avec possibilité de le supprimer
- ➔ Ajouter un article ou un sujet en favoris
- ➔ Noter un sujet
- ➔ Liker un commentaire
- ➔ Suivre un utilisateur
- ➔ Signaler un sujet ou un commentaire
- ➔ Faire une suggestion sur un article
- ➔ Supprimer ou modifier son compte

Fonctionnalités des administrateurs du site :

- ➔ Créer / modifier / supprimer de nouvelles catégories pour les sujets
- ➔ Créer / modifier / supprimer de nouveaux sujets
- ➔ Lire les signalements de sujets
- ➔ Lire les signalements de commentaires
- ➔ Modifier / supprimer des utilisateurs
- ➔ Créer / modifier / supprimer de nouveaux tags pour les articles
- ➔ Créer / modifier / supprimer de nouveaux articles
- ➔ Modifier / supprimer des utilisateurs

Fonctionnalités particulières

Un lien de mot de passe oublié renvoi un mail à l'utilisateur avec un lien pour pouvoir le changer.

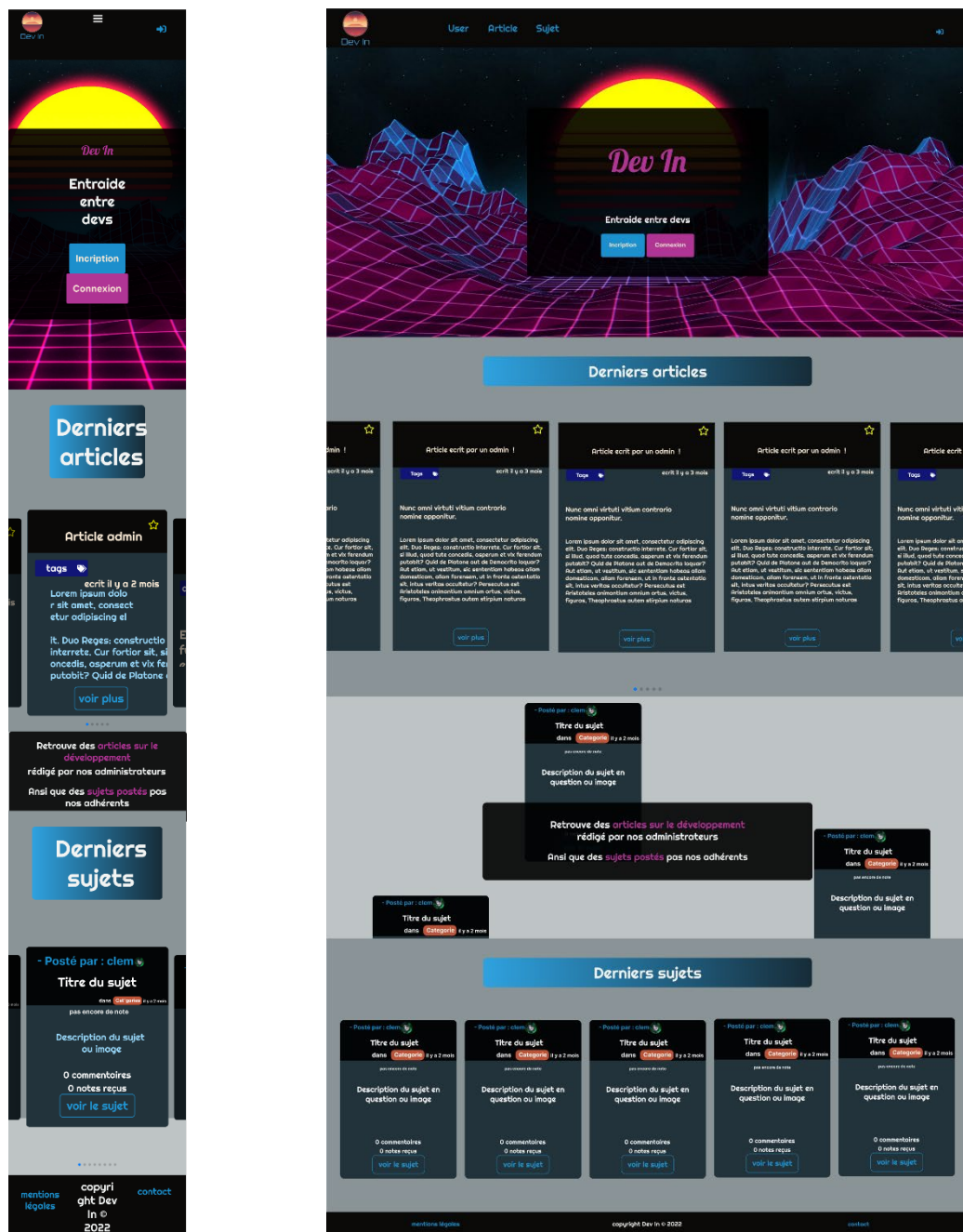
Quand un commentaire ou un sujet est supprimé par un administrateur, l'utilisateur l'ayant écrit perd un point de crédibilité et ceux l'ayant signalé en gagne un. Tout signalement abusif fait perdre également un point de crédibilité. Le système de crédibilité permet de pouvoir modifier les droits des utilisateurs par la suite pour un meilleur fonctionnement du site.

3. LISTE DES COMPÉTENCES

CP.1 Maquetter une application

Avant de développer le front du site DevIn il a fallu maquetter l'application. Pour cela j'ai utilisé Figma en suivant la stratégie mobile first qui vise à privilégier la version mobile qui est maintenant largement plus utilisé que la version desktop.

Voici un exemple avec la page d'accueil du site :



Pour la gestion de projet j'ai utilisé Nifty qui m'a permis de noter toutes mes idées, de pouvoir les classer selon leurs priorités et de ne rien oublier dans mes tâches à effectuer.

CP.2 Réaliser une interface utilisateur web statique et adaptable

Les feuilles de styles ont été conçues en SCSS qui permet l'utilisation de variables et d'imbriquer le CSS le rendant plus proche d'un langage de programmation.

Pour gérer le responsive je suis passé par les media-queries en utilisant :

```
@media all and (max-device-width: 491px)
{
}
```

Entre les accolades se trouvent tous les changements de propriétés pour les classes qui interviennent pour les écrans d'une largeur inférieure à 491px.

CP.3 Développer une interface utilisateur web dynamique

Pour la partie dynamique j'ai bien sûr utilisé Javascript. Il y a deux façons de l'utiliser dans Symfony, la manière la plus classique en écrivant dans nos fichiers .js appelé dans app.js dans le dossier assets mais pour certaines fonctionnalités j'ai dû faire autrement.

Effectivement comme j'utilise une pagination et que j'ai mis en place un scroll infini sur le site, les articles sont créés au fur et à mesure du scroll de l'utilisateur avec des clones.

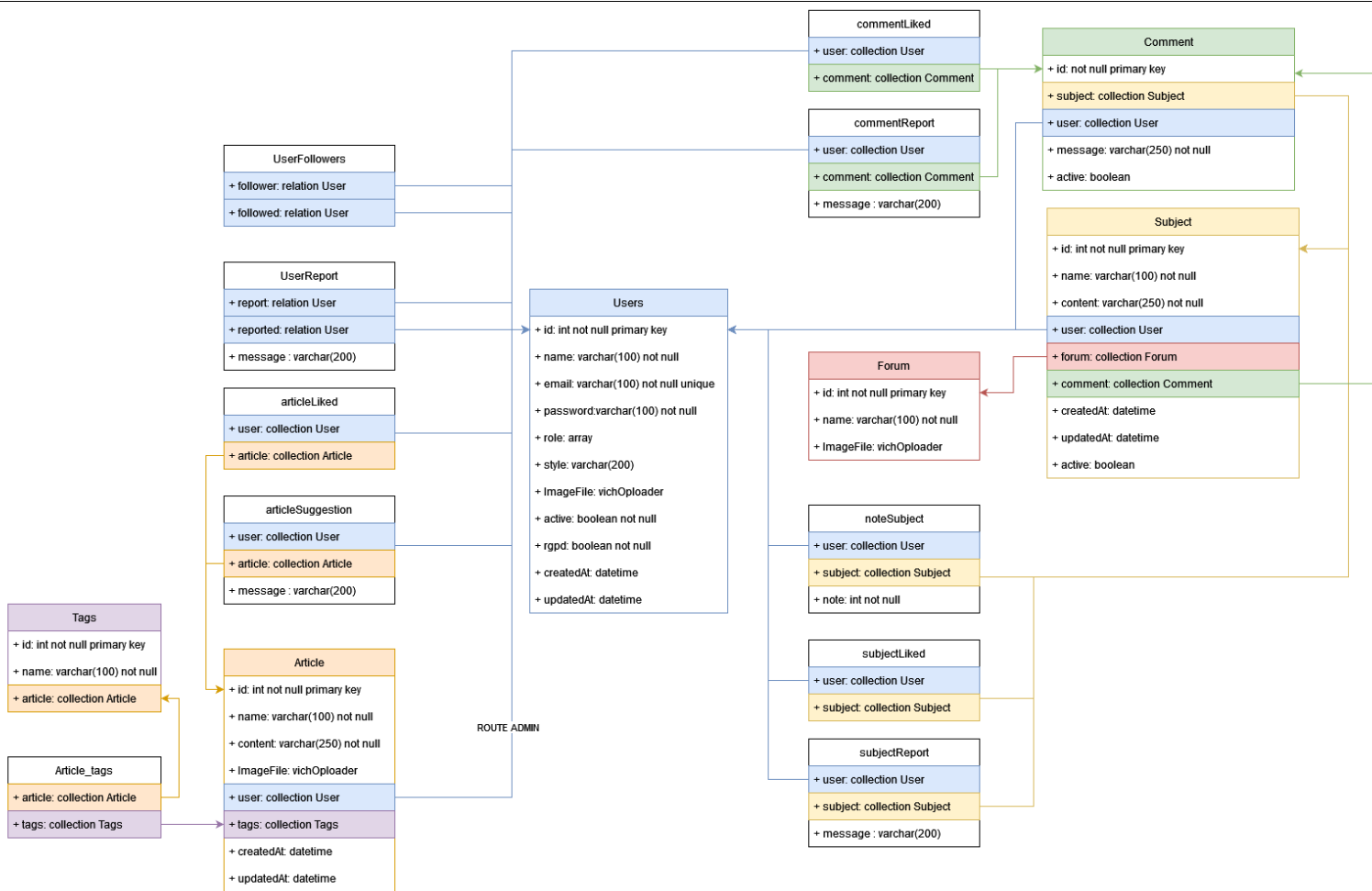
Les fonctions javascript classiques utilisant le querySelector ne fonctionnant pas sur ces clones, je suis passé par les controllers javascript, une solution mise en place par Stimulus et prise en compte par Symfony pour palier à ce problème.

CP.5 Créer une base de données

Pour la conception de la base de données j'ai commencé par faire un schéma des tables qu'il me fallait pour le projet avec draw.io.

Comme DevIn est un réseau social tout tourne principalement autour de l'utilisateur.

Schéma de la base de données :



CP.6 Développer les composants d'accès aux données

Avec Symfony pour nos composants d'accès aux données on utilise Doctrine.

Les requêtes de bases sont déjà implémentées et pour les requêtes custom elles sont définies dans le dossier Repository.

CP.7 Développer la partie back-end

Dans Symfony toute la logique back-end se retrouve dans les Controllers et dans les Repository.

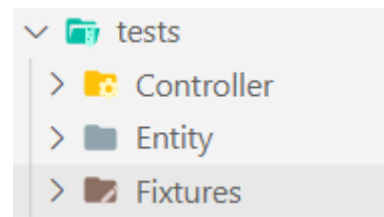
Chaque route correspond à un controller dans lequel on va récupérer les données voulues et les envoyer à notre vue, mais également faire les vérifications de sécurité et établir nos fonctionnalités.

Tout cela va être détaillé dans la suite du dossier.

J'ai également mis en place quelques tests unitaire avec phpunit.

Dans le dossier tests se trouvent les dossiers :

- Controller pour les tests sur nos controller
- Entity pour les tests sur nos Entity
- Fixtures qui vont créer nos données en bases pour effectuer les tests



Exemple d'un test simple pour la page de login :

```
public function testLoginPageResponse()
{
    $this->client->request('GET', '/login');
    $this->assertResponseIsSuccessful();
}
```

En lançant la commande 'php bin/phpunit --testdox' nous avons la réponse :

```
Security Controller (App\Tests\Controller\SecurityController)
✓ Login page response
```

Preuve que le test s'est bien effectué.

Exemple pour tester la connexion à l'espace Admin avec un utilisateur ayant le rôle admin :

```
public function testAdminGoodLoggedIn()
{
    $userAdmin = $this->userRepository->findOneByEmail('clement@test.com');

    $this->client->loginUser($userAdmin);

    $this->client->request('GET', '/admin');
    $this->assertResponseIsSuccessful();
}
```

On récupère notre utilisateur, on le connecte avec 'loginUser()' et on teste la page /admin.

On doit avoir une réponse positive (assertResponseIsSuccessful()). On testera un accès admin avec un utilisateur n'ayant pas le rôle en vérifiant que la réponse est bien forbidden :

```
$this->assertResponseStatusCodeSame(Response::HTTP_FORBIDDEN);
```

4. SPÉCIFICATIONS TECHNIQUES

Pour développer DevIn, j'ai fait le choix du Framework PHP Symfony. C'est un Framework MVC (Modèle-Vue-Contrôleur) qui permet de bien organiser le code en le séparant en ces trois parties distinctes :

- ➔ Les modèles qui sont nos objets (classes) qui correspondent à nos tables en bases de données
- ➔ Les vues qui sont nos pages du site
- ➔ Les contrôleurs qui contiennent la logique concernant les actions à effectuer

Symfony a de nombreux avantages dont le contrôle de la sécurité qui est avancé. Il est facilement modulable avec de nombreux bundles (groupe de produits) qui permettent de coller au mieux aux spécificités du projet. Enfin il est facile d'utilisation permettant une rapidité dans la création avec une bonne maintenabilité. Symfony est dans le top 3 des meilleurs Framework PHP et des plus utilisés.

Technologies du frontend

Le HTML a été écrit en TWIG qui est un moteur de template pour PHP.

Le CSS a été écrit en SCSS à la main.

Le Javascript avec l'utilisation d'axios pour pouvoir faire des requêtes asynchrones.

Technologies du backend

La base de données a été créée avec MySQL.

Avec Symfony j'ai utilisé ces bundles suivants (installé grâce aux gestionnaires de paquets COMPOSER et YARN) :

- ➔ Doctrine\Bundle\DoctrineBundle\DoctrineBundle
(stockage de données et mappage d'objets)
- ➔ Twig\Extra\TwigExtraBundle\TwigExtraBundle
(moteur de template)
- ➔ Symfony\Bundle\SecurityBundle\SecurityBundle
(gestion des utilisateurs)
- ➔ Symfony\Bundle\MakerBundle\MakerBundle
(création de la base de données)

- ➔ Vich\UploaderBundle\VichUploaderBundle
(gestion d'uploade d'image)
- ➔ Knp\Bundle\PaginatorBundle\KnpPaginatorBundle
(gestion de la pagination)
- ➔ EasyCorp\Bundle\EasyAdminBundle\EasyAdminBundle
(gestion administrateur)
- ➔ FOS\CKEditorBundle\FOSCKEditorBundle
(éditeur de texte enrichi)
- ➔ Symfony\UX\Chartjs\ChartjsBundle
(création de graphique)

Technologies pour le développement

Utilisation de Visual Studio Code comme éditeur de code avec notamment les plugins de Symfony extensions pack de Steven DUBOIS comprenant entre autres PHP Intelephense pour vérifier le code PHP.

Utilisation de Github pour versionner mon code.

Utilisation de Photoshop pour le traitement des images.

5. VEILLE ET SECURITÉ

L'avantage d'utiliser Symfony est que le Framework prend en charge beaucoup de spécificités concernant la sécurité.

En faisant des recherches sur les vulnérabilités possibles sur Symfony j'ai découvert des failles par injections de requêtes illégitimes par rebond (CSRF). [Lien de l'article](#)

Il s'avère que cette faille n'affecte que des anciennes versions, la version utilisée pour DevIn est la version 6.1.8 et n'était pas concernée.

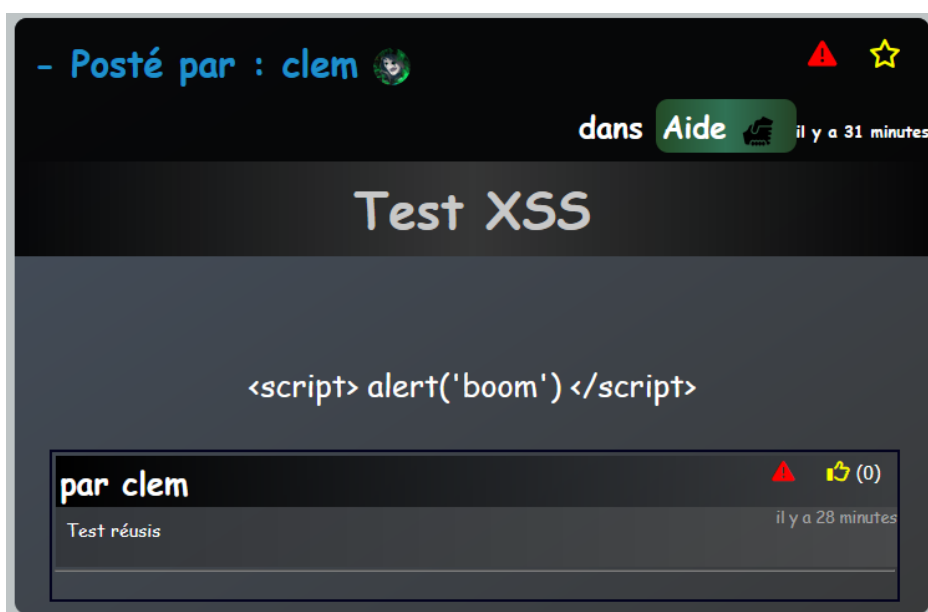
Il est important de faire une veille constante sur le sujet pour éviter toute attaque qui pourrait compromettre le site ainsi que les données personnelles des utilisateurs.

Un autre type de faille est les attaques XSS (injection de code indirect à distance). Je me suis renseigné sur le très renommé site de [developer.mozilla](https://developer.mozilla.org/).

Ces attaques sont généralement commises en entrant du script dans les formulaires.

Comme j'utilise CKeditor pour les sujets postés par les utilisateurs, j'ai désactivé l'option code source qui ne permet pas l'interprétation du code entré dans le navigateur en le chiffrant. Comme le but du site DevIn est le partage d'information sur le code il était essentiel que les utilisateurs puissent le partager sans s'exposer aux failles de sécurité.

Pour vérifier les failles XSS je teste le site en envoyant dans les différents input (entrée de données dans un système informatique) des scripts de type « `<script> alert('faille XSS trouvé') </script>` » et je vérifie qu'aucune fenêtre d'alerte ne s'affiche à l'envoi du formulaire et à l'affichage du post en question.



Concernant l'accès au site, au moment de l'inscription j'ai établi un Regex (contraction de regular expression et pouvant examiner, changer et manier du texte) pour le mot de passe car un mot de passe trop faible serait facilement hacké.

```
new Regex( '/^((?=\S*[A-Z])(?=\S*[a-z])(?=\S*[0-9])).{8,})\S$/' )
```

Ce regex n'accepte que les entrées avec au moins 8 caractères, au moins une majuscule, une minuscule et 1 chiffre sans espace.

Au moment de l'envoi du mot de passe en base de données il est encodé avec la fonction `hashPassword` du composant Symfony `UserPasswordHasherInterface`.

Ceci pour éviter de les exposer en clair dans la base de données qui n'est absolument pas une bonne pratique.

Enfin pour différencier les autorisations entre les utilisateurs et les administrateurs j'ai défini un rôle ADMIN qui protège les routes admin. Pour cela il suffit de configurer l'accès de ces routes dans le fichier `security.yaml` :

```
access_control:
  - { path: ^/admin, roles: ROLE_ADMIN }
```

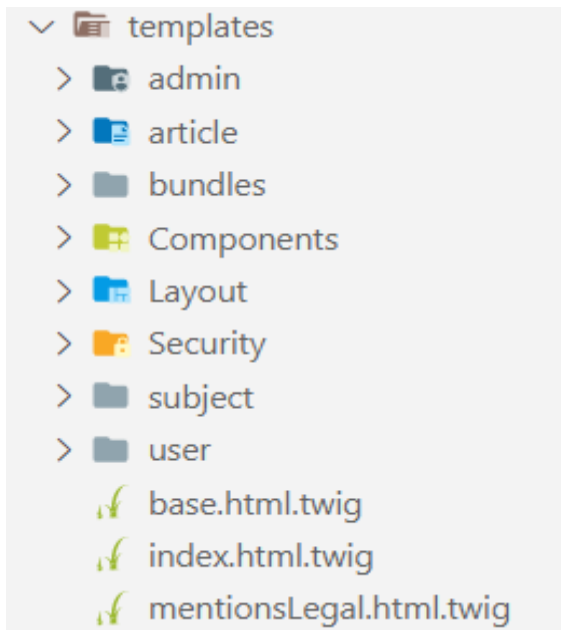
Simplement à l'accès d'une route `/admin` le composant de sécurité vérifiera que l'utilisateur dispose bien du rôle admin, le cas échéant il sera renvoyé à la page de connexion s'il n'est pas connecté ou renverra une page 403.

6. RÉALISATION DU CANDIDAT

Fonctionnement de Symfony

Pour commencer voilà un bref aperçu du fonctionnement du Framework.

Les vues



Toutes nos vues sont dans le dossier templates, elles sont groupées par dossier.

Les composants qui sont utilisés plusieurs fois sont dans le dossier Components et sont appelés à l'utilisation pour éviter la duplication de code et améliorer sa lisibilité.

Dans le dossier bundles se trouvent les pages d'erreur (404, 403...).

Et dans le dossier Layout se trouvent les composants utilisés sur toutes les pages comme le header et le footer.

La page base.html.twig est la branche principale, toutes les autres vont découler de celle-ci.

On va y trouver le doctype Html de base avec le système de block de TWIG.

```
<!DOCTYPE html>
<html>
  <head>
    {% block meta %}{% endblock %}

    {% block title %}{% endblock %}

    {% block stylesheets %}{% endblock %}

    {% block javascripts %}{% endblock %}
  </head>
  <body>
    {% block body %}{% endblock %}
  </body>
</html>
```

Dans la base sont inclus le header, le footer et les messages d'erreur au moyen d'un include :

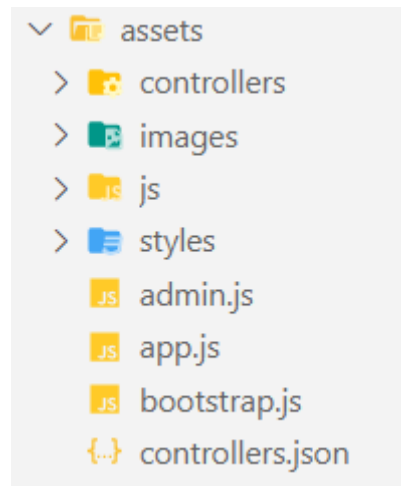
```
{% include "Layout/header.html.twig" %}
```

Nos autres pages devront commencer par un block extends avec le chemin du fichier de la vue à partir de la racine du dossier templates :

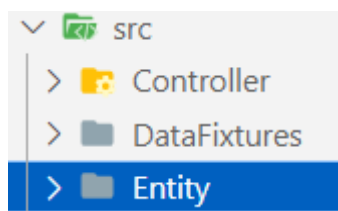
```
{% extends "base.html.twig" %}
```

On pourra ensuite utiliser les blocks issus de la vue parente pour y injecter notre code.

Les feuilles de styles et les scripts sont rangés dans le dossier assets.



Les modèles



C'est dans le dossier src que se trouve le sous dossier Entity où se trouvent nos modèles. On y trouvera nos classes correspondantes aux tables en bases de données.

Dans chaque classe seront listés leurs propriétés qui sont les champs en base de données.

Grace au composant `Symfony\Component\Validator\Constraints` (as Assert) on pourra indiquer dans chaque propriété ces contraintes liées en bases de données pour éviter des erreurs SQL.

Exemples avec l'entité Article.

```
#[Assert\Length(
    max: 100,
    maxMessage: 'Le titre ne doit pas dépasser {{ limit }} caracteres'
)]
#[Assert\NotBlank]
#[ORM\Column(length: 100, unique: true)]
private ?string $name = null;
```


Et avec le composant Doctrine\ORM\Mapping (as ORM) on indique les spécificités du champ associé en base de données.

On indiquera nos relations de table avec également :

```
#[ORM\ManyToOne(targetEntity: Tags::class, mappedBy: 'article')]
private Collection $tags;
```

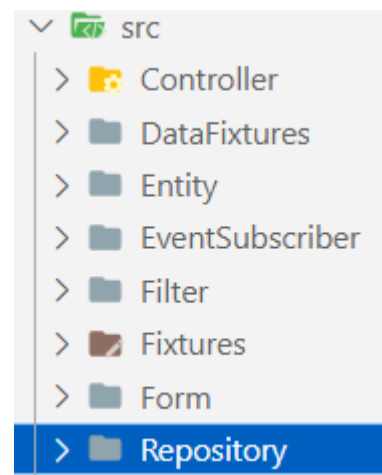
Ici on indique que la propriété `$tags` est en relation ManyToMany avec l'entité `Tags::class`.

Et on indique également le champ associé dans l'entité Tags : `'article'`.

Se trouve aussi dans les classes les Getter et Setter qui permettront de lire les propriétés de nos objets dans les contrôleurs ainsi que pour leur assigner de nouvelles valeurs.

A noter que dans Symfony les fonctions référentes à nos classes sont rangées dans le sous dossier Repository.

On y trouvera les requêtes SQL associée à chacune de nos classes.



Exemple d'une requête personnalisée avec ArticleRepository :

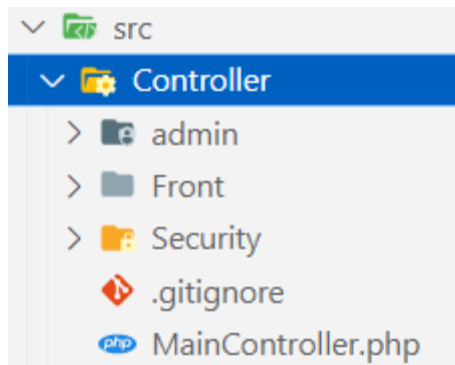
```
public function findArticleWithSameTags($tags): array
{
    return $this->createQueryBuilder('a')
        ->leftJoin('a.tags', 't')
        ->andWhere('t.id IN (:tags)')
        ->setParameter('tags', $tags)
        ->orderBy('a.createdAt', 'DESC')
        ->setMaxResults(5)
        ->getQuery()
        ->getResult();
}
```

On utilise le QueryBuilder,

'a' fait référence à notre table article à laquelle on joint nos tags pour récupérer uniquement les articles avec les tags voulus.

Et on indique une limite maximum de 5 articles demandés.

Les contrôleurs



Enfin c'est dans le sous dossier Controller que se trouve toute la logique ainsi que la sécurité de notre site.

Il est organisé avec une partie admin et une partie front avec quatre contrôleurs :

- ➔ ArticleController
- ➔ CommentController
- ➔ SubjectController
- ➔ UserController

Enfin dans le dossier Security se trouvent les fonctions d'inscription, de connexion ainsi que de réinitialisation du mot de passe.

Exemple avec la fonction show de l'ArticleController.php qui renvoie un article demandé :

```
#[Route('/{id}/{slug}', name: 'article_show', methods: ['GET'])]
public function show(?Article $article, string $slug, ArticleRepository $articleRepository): Response
{
    if(!$article instanceof Article) {
        $this->addFlash('error', 'Nous ne trouvons pas l'article demandé');

        return $this->redirectToRoute('app_article_index', [], Response::HTTP_SEE_OTHER);
    }
    $articles = $articleRepository->findArticleWithSameTags($article->getTags());

    return $this->render('article/show.html.twig', [
        'article' => $article,
        'articles' => $articles,
    ]);
}
```

En haut de la fonction, en commentaire, on indique la route qui renvoie à cette fonction, puis son nom qui nous sert à l'appeler dans nos vues ainsi que la ou les méthodes autorisées.

On vérifie en premier lieu si l'article demandé existe, et le cas échéant, on renvoie directement l'utilisateur sur l'index des articles.

Si l'article existe bien, on utilise le repository pour récupérer quelques articles avec le même tag pour les proposer à l'utilisateur après la lecture de l'article concerné.

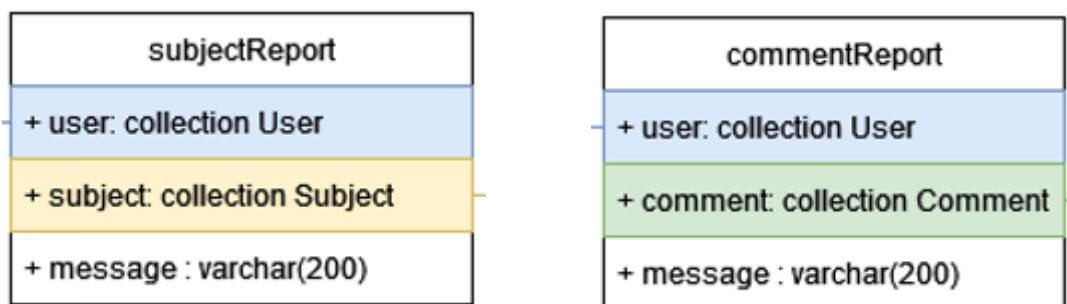
Enfin on envoie nos data à la vue : l'article demandé et quelques articles suggérés pour la suite de la visite sur le site.

Fonctionnalités importantes du site

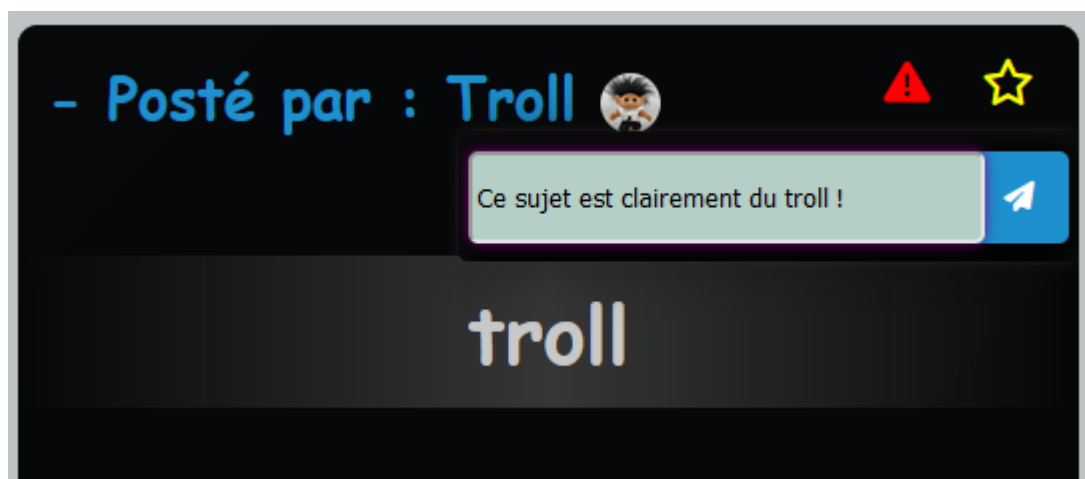
Signalement de contenu indésirable

Comme Dev In est un réseau social, il a été intéressant d'ajouter des éléments de signalisations sur ce que peuvent poster les utilisateurs pour qu'ils puissent remonter des propos indésirables.

J'ai donc les tables CommentReport et SubjectReport dans lesquelles sont stockés l'id de l'utilisateur qui fait le signalement, son message et l'id du commentaire ou du sujet en question.



En cliquant sur l'icône de signalement un input passe en display block pour entrer le message et l'envoyer.



Les signalements sont ensuite affichés dans la partie admin pour pouvoir faire la modération et supprimer les contenus inappropriés.

<input type="checkbox"/>	ID ↓	Message ↕	User ↕	Subject ↕
<input type="checkbox"/>	11	Ce sujet est clairement du troll !	clem	Subject #23
<input type="checkbox"/>	10	troll	clem	Subject #21
<input type="checkbox"/>	9	nul	Linus	Subject #21

C'est à ce moment que j'ai pensé ajouter à la table User un champ crédibilité de type integer pour pouvoir différencier les utilisateurs et donner plus de droits à ceux qui feraient des signalements justifiés.

La crédibilité est fixée à 0 à l'inscription. Ensuite pour chaque signalement utile on rajoute +1, et pour tout signalement inutile ou tout contenu indésirable posté on enlève -1.

Le traitement des points est fait au moment de la suppression d'un commentaire, d'un sujet ou d'un signalement. La partie admin étant faite avec le bundle easyAdmin, il a fallu passer par un EventSubscriber pour pouvoir le faire.

L'EventSubscriber permet de faire des actions à chaque création, modification ou suppression d'une entité en base de données.

Ici on utilisera seulement l'action à mener avant la suppression d'un élément.

```
public static function getSubscribedEvents() : array
{
    return [
        BeforeEntityDeletedEvent::class => ['setCredibility'],
    ];
}
```

On place notre gestionnaire d'évènement dans la fonction `getSusbscribedEvents`. Ici un gestionnaire d'évènement avant la suppression d'une entité dont la fonction associée sera `setCredibility`.

```
public function setCredibility(BeforeEntityDeletedEvent $event)
{
    $entityInstance = $event->getEntityInstance();

    if (
        !$entityInstance instanceof Comment
        && !$entityInstance instanceof CommentReport
        && !$entityInstance instanceof Subject
        && !$entityInstance instanceof SubjectReport
        && !$entityInstance instanceof ArticleSuggestion
    ) {
        return;
    }
}
```

Ensuite on vérifie si l'entité supprimée est bien une instance des entités qui nous intéressent pour la crédibilité et si ce n'est pas le cas on sort de la fonction.

Après cette vérification faite on gère les différents cas suivant les entités concernées à l'aide d'un `switch`.

```
switch(true) {
    case $entityInstance instanceof Comment:
        $entityInstance->getCommentReports();
        foreach ($entityInstance->getCommentReports() as $report) {
            $user = $this->userRepository->find($report->getUser());
            $user -> setCredibility( $user->getCredibility()+1);
            $this->userRepository->add($user,true);
        }
        $user = $entityInstance->getUser();
        $user -> setCredibility( $user->getCredibility()-1);
        break;
}
```

Dans le cas d'une suppression d'un commentaire on récupère les signalements faits, et pour chaque signalement, on récupère son auteur pour lui rajouter un point de crédibilité.

(A noter qu'un utilisateur ne peut faire qu'un seul signalement par commentaire ou sujet pour éviter une fraude au point de crédibilité)

Ensuite on enlève un point à l'auteur du commentaire supprimé.

```

case $entityInstance instanceof CommentReport:
    $user = $entityInstance->getUser();
    $user -> setCredibility( $user->getCredibility()-1);
    $this->userRepository->add($user,true);
    break;

```

Si la suppression concerne un signalement on enlève simplement un point de crédibilité à l'auteur du signalement jugé abusif.

Pour les suppressions d'articles ou de signalements d'articles la logique est la même.

En dernier lieu les utilisateurs peuvent également faire des suggestions sur les articles écrit par l'admin. Dans l'entité ArticleSuggestion est ajouté un champ 'util' qui est un booléen pour différencier l'action à mener à sa suppression.

```

case $entityInstance instanceof ArticleSuggestion:
    $user = $entityInstance->getUser();
    if ($entityInstance->isUtil()){
        $user -> setCredibility( $user->getCredibility()+1);
        $this->userRepository->add($user,true);
    }else{
        $user -> setCredibility( $user->getCredibility()-1);
        $this->userRepository->add($user,true);
    }
    break;

```

Ici on récupère la valeur du champ 'util' de la suggestion et si 'util' est à 'true' on rajoute un point et sinon on en enlève un.

Grâce à cette donnée on peut différencier les signalements, et maintenant, si c'est un utilisateur avec une forte crédibilité qui le fait (strictement plus de 20), on passe directement le contenu visé en non visible pour qu'il ne s'affiche plus sur le site.

```

$newSignal = new SubjectReport();
$newSignal->setUser($user)
    ->setSubject($subject)
    ->setMessage($message);
$subjectReportRepository->add($newSignal, true);
if($user->getCredibility() > 20 ){
    $subject->setActive(0);
    $subjectRepository->add($subject, true);
}

```

On ajoute dans un premier temps le signalement en base de données en lui assignant son auteur \$user, son sujet \$subject et son message.

Ensuite on verifie si sa valeur de crédibilité est au-dessus de 20 et si c'est le cas on passe le sujet en non-actif pour ne plus l'afficher.

Ensuite l'admin aura la possibilité de le supprimer définitivement s'il le trouve justifié ou le remettre en actif le cas échéant.

Note des sujets

Une des principales fonctionnalités données à l'utilisateur est de soumettre une note de 1 à 5 étoiles sur un sujet.

Pour que l'utilisateur connecté puisse noter on affiche en front les étoiles.

```

<div {{ stimulus_controller('subject') }} class="note">
    {% for i in 1..5 %}
        <button
            {{ stimulus_action('subject', 'noter') }}
            class="btn-vierge"
            value="{{i}}-{{subject.id}}"
            type="button"
            title="Notez {{i}} étoile{{i>1?'s'}}">
            <i class="far fa-star stars" ></i>
        </button>
    {% endfor %}
</div>

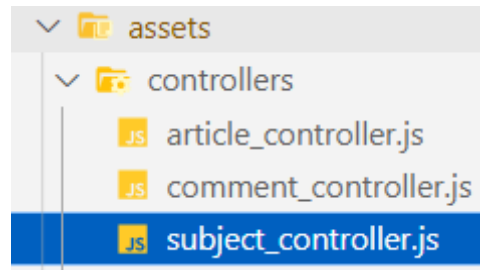
```



On crée à l'aide d'une boucle 5 étoiles avec en valeur le nombre de 1 à 5 et l'id du sujet à noter.

On met un message dans l'attribut title pour avoir un message d'information quand l'utilisateur survole de manière prolongée une étoile.

Le stimulus_controller sert à pointer une écoute d'événement sur l'élément HTML avec son controller JS associé, ici 'subject' pointant vers le fichier subject_controller.js.



Le stimulus_action donne l'action à effectuer au click avec le nom de la fonction associée, ici 'noter'.

```
noter(event){  
    let elem = event.target  
    if (elem.type != "button" ){  
        elem = event.target.parentNode  
    }  
}
```

Notre fonction 'noter' prend un paramètre qui est 'event' et qui nous donne accès aux informations de l'événement qui vient de se passer.

En premier lieu on récupère notre 'event.target' qui pourra être soit notre bouton soit notre icône. Si c'est sur l'icône que le click a été effectué, on ajuste notre variable 'elem' à son parent pour bien pointer sur le bouton, qui a dans sa valeur la note et l'id du sujet à envoyer à notre 'controller' pour l'écrire en base de données.

On récupère ensuite nos valeurs.


```

let value = elem.value.split('-');
let note = value[0];
let subjectId = value[1];
let verifSubjectId = elem.parentNode.parentNode.parentNode.id
verifSubjectId = verifSubjectId.split('t')[1];

```

Avec la fonction JS split on coupe la valeur de notre bouton au '-' et on récupère ainsi la note en position [0] et l'id du sujet en position [1]. (La fonction split renvoie un tableau)

Et on récupère l'id du sujet sur la div du sujet lui-même en remontant avec 'parentNode' pour rajouter une couche de vérification. (En changeant l'id dans la console un utilisateur pourrait falsifier les données).

Avec nos valeurs récupérées et avant d'envoyer notre requête on fait une vérification pour détecter une éventuelle fraude. On va tester si 'subjectId' et 'verifSubjectId' correspondent bien et tester également si la valeur de la note est bien comprise entre 0 et 5.

```

if(verifSubjectId != subjectId || note > 5 || note < 0){
  window.setTimeout(function(){
    elem.parentNode.style.display = "none";
  },700);
  let btn= elem.parentElement.childNodes;
  btn.forEach(elements => {
    if(elements.nodeType == 1 ){
      elements.style.display = "none";
    }
  });
  let divFraude = document.createElement("div");
  let responseFraude = document.createTextNode('Fraude détectée 🚫');
  divFraude.appendChild(responseFraude);
  elem.parentNode.insertBefore(divFraude, elem);
  return ;
}

```


Dans le cas où les valeurs envoyées correspondent à une tentative de fraude on fait disparaître la div parente qui englobe les étoiles avec un delay de 0.7s .

On enlève les étoiles pour noter en passant leur display à « none » et ensuite on crée une div dans laquelle on indique le message de fraude détecté.

Comme ça l'utilisateur ne peut pas venir modifier les valeurs en console et fausser les résultats.

Si les valeurs sont bonnes on les envoie à notre controlleur avec axios.

```
axios.get(`/subject/note/${note}/${subjectId}`)
.then(function (reponse) {
  // enleve les étoiles
  let btn= elem.parentElement.childNodes;
  btn.forEach(elements => {
    if(elements.nodeType == 1 ){
      elements.style.display = "none";
    }
  });

  // fait disparaître la div du message après 0.7s
  window.setTimeout(function(){
    elem.parentNode.style.display = "none";
  },700);
  // crée une div pour afficher la reponse positive
  let newDiv = document.createElement("div");
  let newContent = document.createTextNode('Note envoyée ');
  newDiv.appendChild(newContent);
  elem.parentNode.insertBefore(newDiv, elem);
})
.catch(function (erreur) {
  console.log(erreur.response);
});
```

Le .then dans notre fonction s'exécutera si la requête s'est bien effectué. Dans le cas contraire le .catch s'exécutera.

Du coté de notre controlleur la fonction prends en paramètre :

- ➔ \$note qui est un integer
- ➔ \$subjectId qui doit être une instance de notre entité Subject
- ➔ Les SubjectRepository et NoteRepository que nous utiliserons dans la fonction
- ➔ Et Security que nous utiliserons pour récupérer l'utilisateur connecté.

```
#[Route('/note/{note}/{subjectId}', name: 'app_subject_note', methods: ['GET'])]
public function noteSubject(
    int $note,
    Subject $subjectId,
    SubjectRepository $subjectRepository,
    NoteSubjectRepository $noteRepo,
    Security $security)
{
    |
```

```

if( !$subjectId instanceof Subject){
    return new Response('Ce sujet n\'existe pas', 404);
}

$user = $security->getUser();

if (!$user) {
    return new Response('utilisateur non connecté', 403);
}

```

On vérifie ensuite que '\$subjectId' est bien une instance de 'Subject' sinon on renvoie une réponse d'erreur directement.

Avec le composant Security on récupère notre utilisateur connecté et s'il n'y en a pas on renvoie une réponse d'erreur également.

On va ensuite refaire une vérification si toute fois un utilisateur essaierais de noter un sujet à lui ou un sujet qu'il aurait déjà noté.

```

$dejaNoter = $noteRepo->findOneBy(['user' => $user, 'subject' => $subjectId->getId()]);
$isUserSubject = $subjectRepository->findOneBy(['user' => $user, 'id' => $subjectId->getId()]);

if ($dejaNoter || $isUserSubject ) {
    return new Response('fraude suspecté 🚫', 403);
}

```

On cherche dans la table 'NoteSubject' s'il y a une entrée avec notre user et l'id du sujet et on cherche dans la table 'Subject' si il y a une entrée avec notre user et l'id de sujet sélectionné.

Si '\$dejaNoter' ou '\$isUserSubject' est à true alors on renvoie une réponse d'erreur.

Passé ces vérifications on va pouvoir maintenant envoyer notre note en base de données.

```

$newNote = new NoteSubject();
$newNote->setUser($user)
    ->setSubject($subjectId)
    ->setNote($note);

$noteRepo->add($newNote, true);

return new Response('note envoyer', 201);

```

On crée une nouvelle instance de la classe 'NoteSubject' dans laquelle on rentre l'utilisateur \$user, le sujet '\$subjectId' et la note '\$note'.

On utilise la fonction 'add' pour envoyer la note en base et on peut sortir avec une réponse en code 201.

Expérience utilisateur et référencement

Pendant la conception du site j'ai fait attention à l'expérience donnée à l'utilisateur, ce qui est primordiale, le site étant fait pour lui.

Pour ce faire j'ai passé beaucoup de requête classique en asynchrone avec axios. Les requêtes asynchrones permettent de traiter la requête à coté tout en continuant l'exécution de la tâche en cours. Ici on l'utilise pour éviter les rechargements de page qui n'offre pas une expérience optimale.

Comme on l'a vu plus haut pour l'envoi de notes, on fait appel en javascript et avec axios en indiquant la méthode utilisée (ici get) suivie de l'url à exécuter.

```
axios.get(`/article/liked/${id}`)
```

Ensuite on utilisera le .then pour la logique à exécuter en cas de succès et le .catch pour celle en cas d'échec.

```
.then(function (reponse) { ...  
  })  
.catch(function (erreur) { ...  
  });
```

Il a été donné également la possibilité de personnaliser sa bannière de carte de profil pour se différencier des autres. De plus en plus les utilisateurs aiment avoir l'occasion de particulariser leur compte.

J'ai développé ce point-ci dans le dossier professionnel. Il s'agit simplement de fonctions javascript qui récupèrent au clique sur une icône de couleur ses propriétés RGB (rouge, vert, bleu) pour créer un dégradé de couleur.

Un soin particulier a été apporté également au fait de pouvoir signaler tout contenu indésirable pour ne pas heurter l'utilisateur durant sa navigation. Et bien évidemment de pouvoir modifier et/ou supprimer tout ce qu'un utilisateur peut créer sur le site, y compris son compte.

Concernant l'accessibilité et le référencement j'ai découvert pendant mes recherches le site [opquast](#) qui référence 240 règles pour permettre d'améliorer les sites et de mieux prendre en compte nos utilisateurs.

Voici quelques-unes des règles que j'ai pu mettre en pratique :

- Règle n°3 : Le code source de chaque page contient une métadonnée qui en décrit le contenu.

Pour ce faire j'ai utilisé dans mes page un block meta.

D'abord placé dans la page de base :

```
{% block meta %}{% endblock %}
```

Tous ce qui sera maintenant écrit dans un block meta sur les pages qui s'étendent depuis la page de base viendra s'inscrire ici.

```
{% block meta %}
  <meta name="description"
  |   content="Le site d'entraide entre developpeur, dans une idée de partage et de progression en toute solidarité ."
  />
  <meta name="robots"
  |   content="index, follow"
  />
{% endblock %}
```

Ici pour la page d'accueil dans une balise meta « description » on indique un bref résumé du contenu qui pourra être vu lors d'une impression par un moteur de recherche.

La balise meta « robots » sert à indiquer au robot d'indexation des moteurs de recherche si la page concernée doit être indexée ou non.

Pour les articles il a été convenu d'utiliser le titre de l'article dans la description :

```
<meta name="description"
|   content="Article ' {{article.name}} ' posté sur Dev In"
/>
```

- Règle n°11 : Les espaces publics proposent au moins un moyen de signaler un abus.
- Règle n°77 : En cas de rejet des données saisies dans un formulaire, les champs contenant les données rejetées sont indiqués à l'utilisateur :

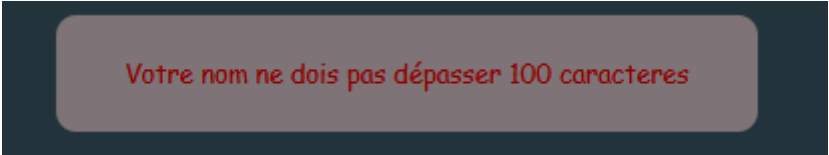
Pour ce faire j'ai utilisé la fonction form_errors implémenté dans symfony :

```
{{ form_errors(form.description) }}
```

De plus j'ai utilisé du css pour afficher l'encart d'alerte en rouge :

```
.alert-danger{
  color: ■ rgb(143, 0, 0);
  background-color: ■ rgba(168, 145, 145, 0.685);
}
```

Ce qui nous donne en cas d'erreur :



Votre nom ne dois pas dépasser 100 caracteres

Le message affiché est celui que l'on a indiqué au préalable dans l'Entity User au moyen du composant validator constraint de symfony :

```
#[Assert\Length(
    max: 100,
    maxMessage: 'Votre nom ne dois pas dépasser {{ limit }} caracteres'
)]
```

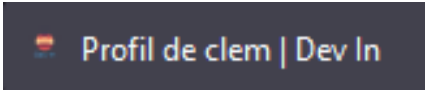
- Règle n°97 : Le titre de chaque page permet d'identifier le site

Profil de {{user.name}} | {{ parent() }}

Dans le block title de chaque page on appelle la fonction parent() qui permet de récupérer le titre écrit dans la page parente.

```
<title>{% block title %}Dev In{% endblock %}</title>
```

Ce qui nous donne :



Profil de clem | Dev In

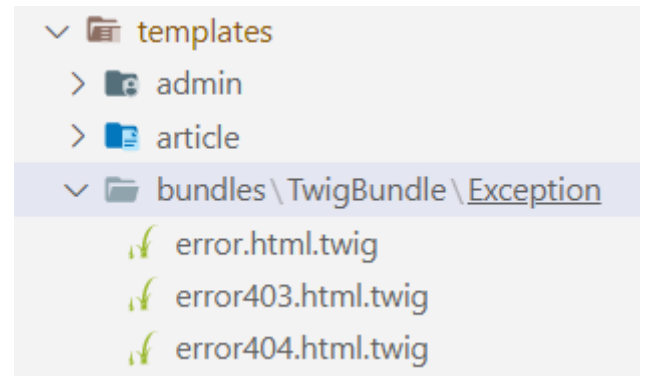
- Règle n°212 : La racine du site contient des instructions pour les robots d'indexation.

Pour cela il y a deux fichiers à la racine du projet robots.txt et sitemap.xml. Le premier indique juste que le site est accessible à tous à l'exception des routes ayant le préfixe admin. Il y a également l'url de la sitemap :

```
1 User-agent: *
2 Disallow: /admin
3 Sitemap: http://www.devIn.com/sitemap.xml
```

- Règle n°216 : Le serveur envoie une page d'erreur 404 personnalisée.

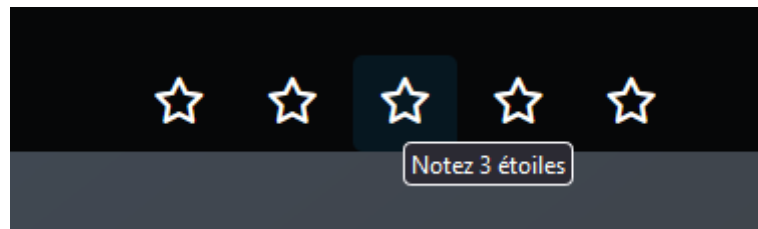
Dans le dossier
template/bundles/TwigBundle/Exception sont rangés
nos différentes pages d'erreur. Il n'y a pour le
moment que les page 404 et 403 de traitées
spécifiquement, les autres auront le mêmes
template.



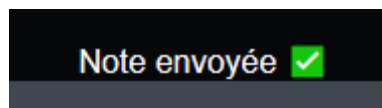
7. JEU D'ESSAI

Nous allons maintenant vérifier que la fonctionnalité de note s'effectue correctement.

Pour ce faire nous allons envoyer une note de trois étoiles sur le sujet ayant l'ID 19 avec l'utilisateur ayant l'ID 13.



Une fois le clique effectué les étoiles disparaissent et s'affiche alors :



Allons vérifier en base de données la dernière entrée dans la table note_subject :

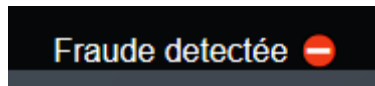
	id	user_id	subject_id	note
	16	10	19	3
	17	8	20	5
	18	11	20	4
	19	11	14	3
	20	11	18	5
	21	1	22	4
	22	1	19	4
	23	1	20	5
	24	1	18	4
	25	1	17	4
	26	6	20	4
	27	6	22	1
	28	13	20	3
	29	13	19	3
	NULL	NULL	NULL	NULL

On a bien une note de 3 faite par l'utilisateur 13 sur le sujet 19, tout est conforme aux attentes.

Maintenant essayons avec une tentative de fraude. On va entrer dans la console du navigateur une note de 8 pour augmenter la moyenne du sujet.

```
<button class="btn-vierge" data-action="subject#noter" value="8-18" type="button" title="Notez 3 étoiles">event
```

Maintenant en appuyant sur la troisième étoile une note de 8 sera envoyé.



Le message de fraude vient remplacer les étoiles et en base de données la note n'a pas été inscrite.

	id	user_id	subject_id	note
	16	10	19	3
	17	8	20	5
	18	11	20	4
	19	11	14	3
	20	11	18	5
	21	1	22	4
	22	1	19	4
	23	1	20	5
	24	1	18	4
	25	1	17	4
	26	6	20	4
	27	6	22	1
	28	13	20	3
▶	29	13	19	3
✱	NULL	NULL	NULL	NULL

Essayons maintenant de noter un sujet que l'on a posté nous-même.

Avec l'utilisateur ayant l'ID 13 on crée d'abord un sujet ayant l'ID 24.

	id	user_id	forum_id	nom
	22	11	1	Comment centrer une...
	23	10	1	troll
	24	13	3	Commencer une liste ...
✱	NULL	NULL	NULL	NULL

Maintenant nous allons sur un autre sujet pour avoir les notes qui apparaissent et changeons en console l'ID du sujet à noter.

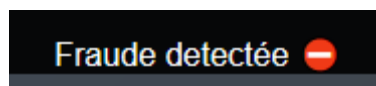
On va aller sur le sujet 18 et changer l'ID sur la cinquième étoile et sur la div de la carte sujet pour contourner la petite vérification de fraude faite en javascript comme vu précédemment dans le chapitre réalisation du candidat.

```

▼ <div id="subject24" class="card-subject">
  ▼ <div class="black">
    ▶ <h2 class="toptitre left">...</h2>
    ▶ <h2 class="toptitre2 right">...</h2>
    ▶ <div class="iconSubject" data-controller="subject">...</div>
  ▼ <div>
    ▶ <h1 class="title">...</h1>
  </div>
  ▼ <div class="note" data-controller="subject">
    ▶ <button class="btn-vierge" data-action="subject#noter" value="1-18" type="button"
      title="Notez 1 étoile">...</button> (event)
    (espaces)
    ▶ <button class="btn-vierge" data-action="subject#noter" value="2-18" type="button"
      title="Notez 2 étoiles">...</button> (event)
    (espaces)
    ▼ <button class="btn-vierge" data-action="subject#noter" value="3-18" type="button"
      title="Notez 3 étoiles"> (event)
      ▶ <i class="far fa-star stars" aria-hidden="true">...</i>
    </button>
    (espaces)
    ▶ <button class="btn-vierge" data-action="subject#noter" value="4-18" type="button"
      title="Notez 4 étoiles">...</button> (event)
    (espaces)
    ▶ <button class="btn-vierge" data-action="subject#noter" value="5-24" type="button"
      title="Notez 5 étoiles">...</button> (event)
  </div>

```

Une fois l'étoile cliquée nous recevons également ce message :



L'envoi de la note est stoppé et sa valeur frauduleuse n'est pas entrée en base de données.

8. RESSOURCES EN ANGLAIS

Durant le développement du site j'ai fait face à un problème dans mes fonctions javascript qui ne fonctionnaient après avoir implémenté la pagination.

En effet, les boutons sur mes articles présents à l'ouverture de la page fonctionnaient comme attendu mais il ne se passait rien sur les articles chargés à la suite en asynchrone.

En recherchant d'où pouvait venir ce problème j'ai trouvé un poste sur un forum renvoyant vers la documentation de Symfony à propos du [Stimulus](#).

La documentation en anglais :

Stimulus & Symfony UX

As simple as the above example is, instead of building your application inside of `app.js`, we recommend Stimulus: a small JavaScript framework that makes it easy to attach behavior to HTML. It's powerful, and you will love it! Symfony even provides packages to add more features to Stimulus. These are called the Symfony UX Packages.

If you followed the setup instructions, you should already have Stimulus installed and ready to go! In fact, that's the purpose of the `assets/bootstrap.js` file: to initialize Stimulus and automatically load any "controllers" from the `assets/controllers/` directory.

Let's look at a simple Stimulus example. In a Twig template, suppose you have:

```
<div {{ stimulus_controller('say-hello') }}>
  <input type="text" {{ stimulus_target('say-hello', 'name') }}>

  <button {{ stimulus_action('say-hello', 'greet') }}>
    Greet
  </button>

  <div {{ stimulus_target('say-hello', 'output') }}></div>
</div>
```

The `stimulus_controller('say-hello')` renders a `data-controller="say-hello"` attribute. Whenever this element appears on the page, Stimulus will automatically look for and initialize a controller called `say-hello-controller.js`. Create that in your `assets/controllers/` directory:

```
// assets/controllers/say-hello-controller.js
import { Controller } from '@hotwired/stimulus';

export default class extends Controller {
  static targets = ['name', 'output']

  greet() {
    this.outputTarget.textContent = `Hello, ${this.nameTarget.value}!`
  }
}
```

The result? When you click the "Greet" button, it prints your name! And if more `{{ stimulus_controller('say-hello') }}` elements are added to the page - like via Ajax - those will instantly work: no need to reinitialize anything.

La traduction que j'en ai faite :

Stimulus & Symfony UX

Aussi simple que l'exemple ci-après, au lieu de créer votre application dans `app.js`, nous recommandons Stimulus : un petit framework Javascript qui rend facile l'attribution de comportement sur des éléments HTML.

C'est puissant, et vous allez l'adorer ! Symfony fournit aussi des packages pour ajouter plus de fonctionnalités à Stimulus.

Ils sont appelés Symfony UX Package.

Si vous avez suivis les instructions d'installation, vous avez Stimulus installé et prêt pour être utilisé.

En fait, c'est l'intérêt du fichier `assets/bootstrap.js` : initialiser Stimulus et charger automatiquement tous les contrôleurs dans le dossier `assets/controllers/`.

Regardez ce simple exemple de Stimulus. Dans un template Twig, supposons que vous ayez :

```
<div {{ stimulus_controller('say-hello') }}>
  <input type="text" {{ stimulus_target('say-hello', 'name') }}>

  <button {{ stimulus_action('say-hello', 'greet') }}>
    Greet
  </button>

  <div {{ stimulus_target('say-hello', 'output') }}></div>
</div>
```

Le `stimulus_controller('say-hello')` rend un attribut `data-controller="say-hello"`.

A chaque fois que cet élément apparaît sur une page, Stimulus va automatiquement rechercher et initialiser le contrôleur nommé `say-hello-controller.js`. Créez dans le dossier `assets/controller/` :

```
// assets/controllers/say-hello-controller.js
import { Controller } from '@hotwired/stimulus';

export default class extends Controller {
  static targets = ['name', 'output']

  greet() {
    this.outputTarget.textContent = `Hello, ${this.nameTarget.value}!`
  }
}
```

Le résultat ? Quand vous cliquez sur le bouton 'Greet', cela va écrire votre prénom. Et si il y a plus d'éléments `{{ stimulus_controller('say_hello') }}` dans votre page ajoutée – comme en Ajax – ils fonctionneront instantanément : pas besoin de réinitialiser quoi que ce soit.

Grâce à ça j'ai pu passer les fonctionnalités Javascript qui ne marchaient plus sur les éléments ajoutés à la page via Ajax et tout mes éléments ont retrouvé leurs fonctionnalités.

9. CONCLUSIONS

J'ai pris plaisir à développer ce projet. En commençant la formation par du procédurale je ne me voyais pas utiliser de framework, j'avais l'impression que de tout coder soi-même offrait beaucoup plus de liberté. En voyant la réalité du marché actuel et le fait que l'immense majorité des entreprises passent maintenant par des framework je me suis donc fait à cette idée.

Finalement j'ai trouvé l'utilisation de Symfony vraiment très efficace. L'architecture donne une clarté dans l'organisation des dossiers, il est beaucoup plus facile de s'y retrouver au fur à mesure que le projet s'étoffe et l'ajout de bundle offre également de multiples possibilités en gagnant également du temps de développement ce qui permet de pouvoir aller plus loin.

Au fur et à mesure de la conception du site j'ai pu entrevoir ce qu'est la gestion de projet, entre l'idée principal et les problèmes qu'on décèle au fur et à mesure du développement. J'ai passé beaucoup de temps à tester les fonctionnalités pour déceler les bugs et le responsive pour offrir une meilleure expérience utilisateur possible.

Les points fort du site sont, à mon avis, tout d'abord le coté crédibilité et signalement, qui combinés, offre un bon rempart contre les contenus indésirables qui pourraient affecter le site.

Ensuite le fait de pouvoir s'appuyer sur la communauté pour enrichir le contenu du site est également intéressant. Et pour finir la possibilité de personnaliser son profil est également un atout, les utilisateurs sont de plus en plus en demande de cela.

Il serait intéressant d'ajouter l'utilisation d'une API externe pour la modération du contenu pour vérifier avant l'envoi en base de données les textes soumis par les utilisateurs.

J'aimerais aussi faire un chat pour que les utilisateurs connectés puissent directement discuter en direct.

Un dernier point qui me semble important et que je n'ai pas eu le temps de mettre en place c'est la vérification d'adresse mail à l'inscription en envoyant un mail avec un lien de validation. Cela permettra d'éviter la création de faux compte.

Pour finir j'aimerais remercier la CCI Formation ainsi que tous les formateurs qui nous ont transmis leur savoir et qui ont fait le maximum pour nous. Après un an de formation je me sens prêt et j'ai hâte de travailler en tant que développeur junior pour enrichir mon expérience.

En commençant la programmation, ce qui m'a immédiatement plu, c'est cette sensation que tout devient possible, et en avançant dans l'apprentissage, cette sensation s'est confirmée.