

Dossier Projet - DWWM

Matthieu NOWAK



SITE GAMETEST
Site de tests de jeux vidéo

Sommaire

- Introduction

- Qui suis-je ?
- Présentation du projet
- Liste des compétences du référentiel
- Cahier des charges de l'application

- Maquettage de l'application

- Wireframe
- Moodboard
- Maquette de l'application

- Mise en place du projet

- Les différents outils utilisés
- Les Commandes symfony

- Changement des thèmes

- La base de données

- Schéma relationnel
- Schéma MCD
- Création des entités

- L'utilisateur

- Inscription
- Les REGEX
- Envoi de mail
- La photo de profil
- Connexion
- Réinitialisation du mot de passe

- Les articles

- Création de l'entité et du CRUD
- Gestion des images et des vidéos
- Système des notes utilisateurs
- Recherche d'articles

- Formulaire de contact

- Conclusion

Introduction

Qui suis-je ?

Avant de commencer à parler du projet je tiens à me présenter. Je m'appelle Matthieu Nowak, j'ai 21 ans et je suis un grand passionné de jeux vidéos, et cela depuis mon plus jeune âge. En plus des jeux vidéos je me suis passionné pour la programmation et le développement informatique. J'ai commencé avec la création de petits jeux sur différents logiciels comme GameMaker Studio et Roblox, et depuis peu je me suis trouvé un domaine de prédilection, le développement web et mobile. J'ai donc après avoir faits différents stages et contrats en tout genre, intégré Pop School, et après une année où j'ai pu apprendre ce qu'est réellement le développement web et quelques-uns de ses langages comme par exemple HTML, CSS, Javascripts, PHP, Symfony, MySQL et React-Native, je me présente à vous pour l'obtention de mon titre professionnel afin de pouvoir continuer mes études dans ce domaine.

Présentation du projet

Le site GameTest est un site de lecture de tests sur des jeux vidéo auxquels j'ai pu jouer. Un modérateur ou moi-même rédigeons un test sur un jeu en toute subjectivité et l'utilisateur peut donc lire le test, et donner une note à la fin de sa rédaction. Les notes des utilisateurs font une moyenne et il y a donc l'avis rédacteur et les avis des utilisateurs qui sont affichés sur le test.

J'ai eu l'idée de ce projet car comme dit précédemment ce média des jeux vidéos me passionne. De plus j'ai toujours aimé avoir un esprit critique et rédigé de moi même des petits tests à note sur des documents excel ou même sur des cahiers. Je me suis même depuis peu lancé sur les réseaux sociaux avec un contenu où je parle de jeux, je réalise des critiques subjectives et toujours dans un format très court avec une note à la fin. Et j'aime aussi les vidéastes ou autres contenus qui exposent leurs avis avec un argumentaire sur un jeu. Je trouve aussi à titre personnel, que les sites actuels manquent réellement de vrais tests car beaucoup de jeux sont surnotés car sponsorisés.

Liste des compétences du référentiel

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	1	Maquetter une application
		2	Réaliser une interface utilisateur web statique et adaptable
		3	Développer une interface utilisateur web dynamique
		4	Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce
2	Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	5	Créer une base de données
		6	Développer les composants d'accès aux données
		7	Développer la partie back-end d'une application web ou web mobile
		8	Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

Cahier des charges de l'application

Le site GameTest est conçu sous symfony, voici les fonctionnalités comprises:

L'utilisateur :

- ❖ Il peut se connecter à son compte ou s'inscrire si il ne possède pas de compte.
- ❖ Lire les articles, donner sa note comprise entre 1 et 20.
- ❖ Contacter la modération via un formulaire de contact.
- ❖ Modifier son profil.
- ❖ Réinitialiser son mot de passe
- ❖ L'utilisateur peut aussi changer de thèmes en fonction de ses préférences.

L'Administrateur :

- ❖ Peut Supprimer, modifier créer des entités, tels que les utilisateurs ou les articles ou encore même les catégories.

L'article :

- ❖ Comprends une note, une image de couverture, un trailer youtube, plusieurs titres, plusieurs paragraphes et plusieurs images.
- ❖ Est relié à l'Entité Catégories pour pouvoir afficher les catégories des jeux.
- ❖ Est relié à l'entité Note pour pouvoir afficher les notes des utilisateurs

Maquettage de l'application

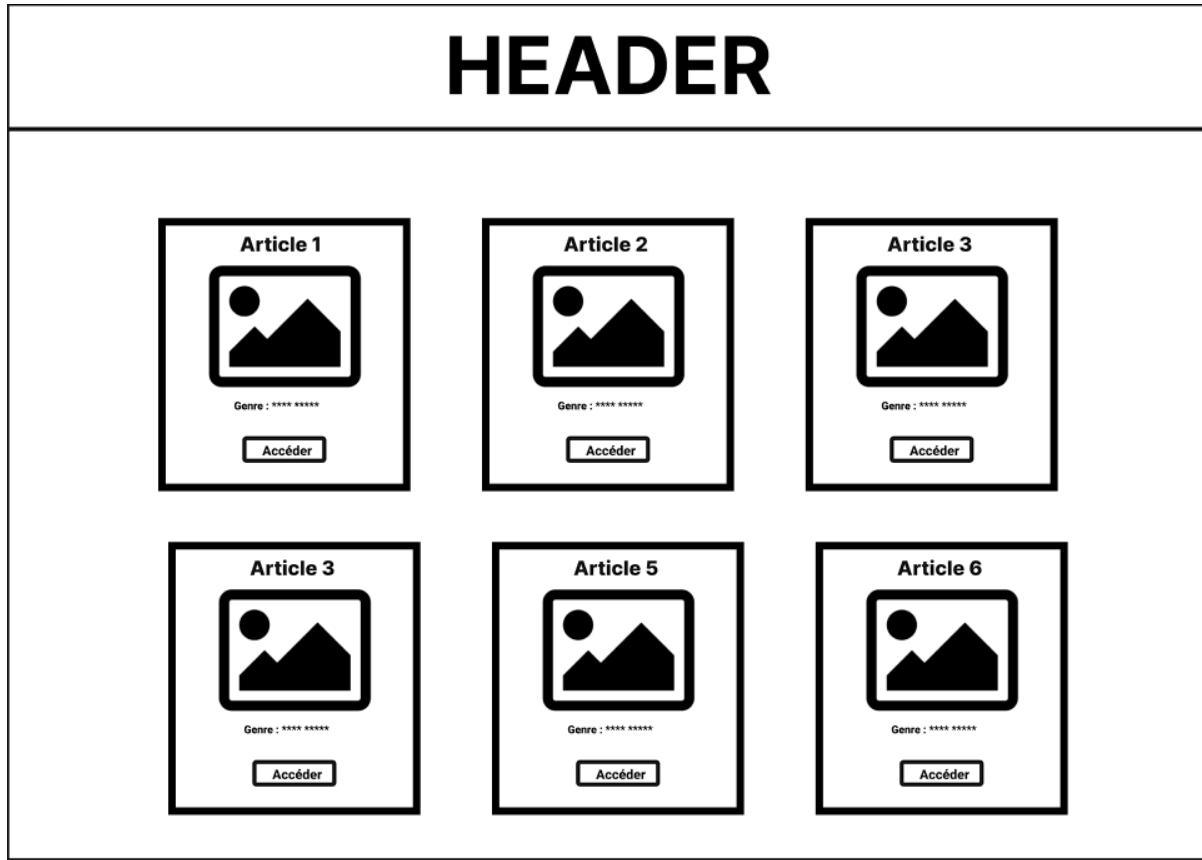
Wireframe

Avant de présenter la maquette graphique et le moodboard il est important de vous montrer les wireframes des pages principales : La connexion, L'article et enfin L'accueil.

Voici en premier temps le wireframe de la connexion, il sera réutilisé plus tard pour d'autres fonctionnalités comme l'inscription et la réinitialisation du mot de passe.

The wireframe consists of two main sections. The top section is labeled "HEADER". The bottom section is labeled "CONNEXION". Inside the "CONNEXION" box, there are two input fields: one for "Email" and one for "Mot de passe". Below these fields is a button labeled "Connexion". At the bottom of the "CONNEXION" box, there is a link labeled "> Inscription <".

Maintenant voici l'accueil qui regroupe les articles par cards:



Et enfin voici le Wireframe de la page Article pour donner une idée à quoi pourrait ressembler l'article :

A detailed wireframe of an article page. The header includes sections for "Trailer" (with a thumbnail image), "Titre" (Title) with a "Couv" (Cover) thumbnail, and "Genre : **** ***** Date publication". The main content area starts with a short text snippet: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Praesent tempus fringilla euismod ultricies. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum." Below this is a large "Image" section containing a larger version of the mountain and sun icon. The bottom section is titled "Conclusion" with a similar short text snippet. At the very bottom are two "Note" fields: "Note 18" and "Note utilisateur 17" under a "20" scale, with a horizontal line separating them.

Moodboard

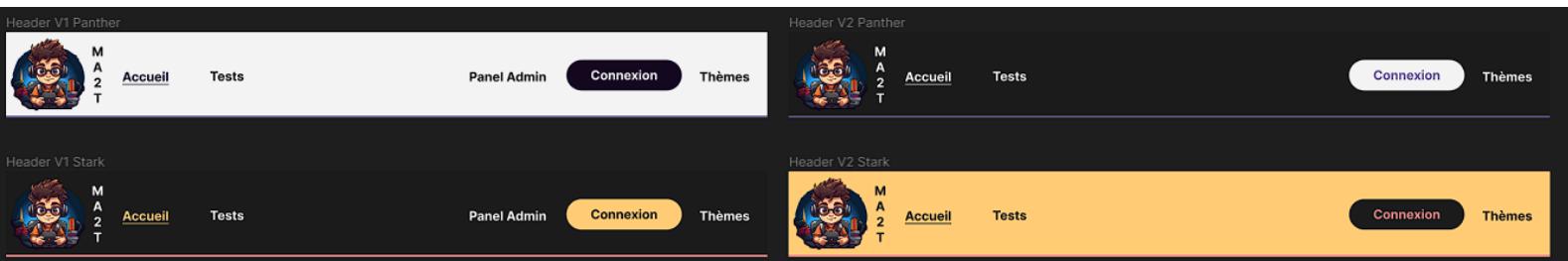
Pour le moodboard il a été conçu avec une de mes autres passions, Marvel. J'ai donc décidé de choisir certains personnages que j'affectionne en choisissant leur couleurs et en modifiant légèrement pour que le contraste soit meilleur, j'ai donc utilisé l'outils contrast checker sur coolors.co pour avoir les meilleurs résultats possible en fonction des couleurs choisies.



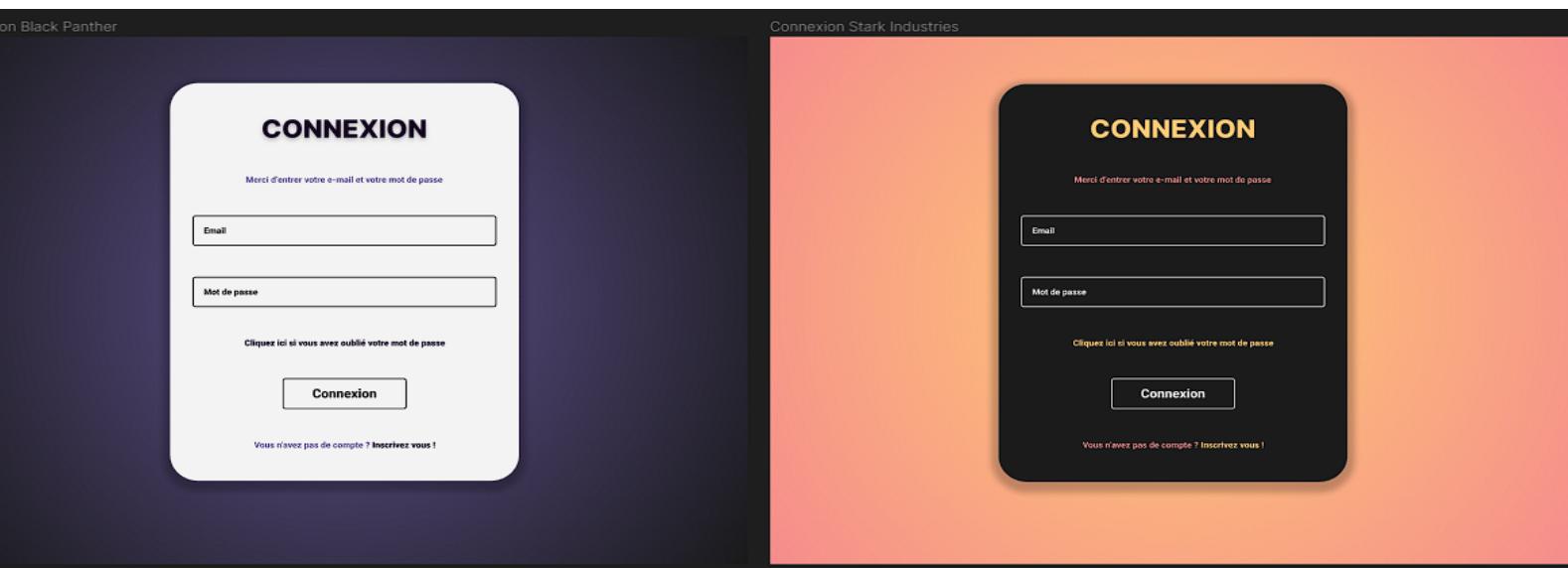
On a donc un thème clair avec celui de Black Panther, et un thème sombre avec celui de Iron Man.

Maquette

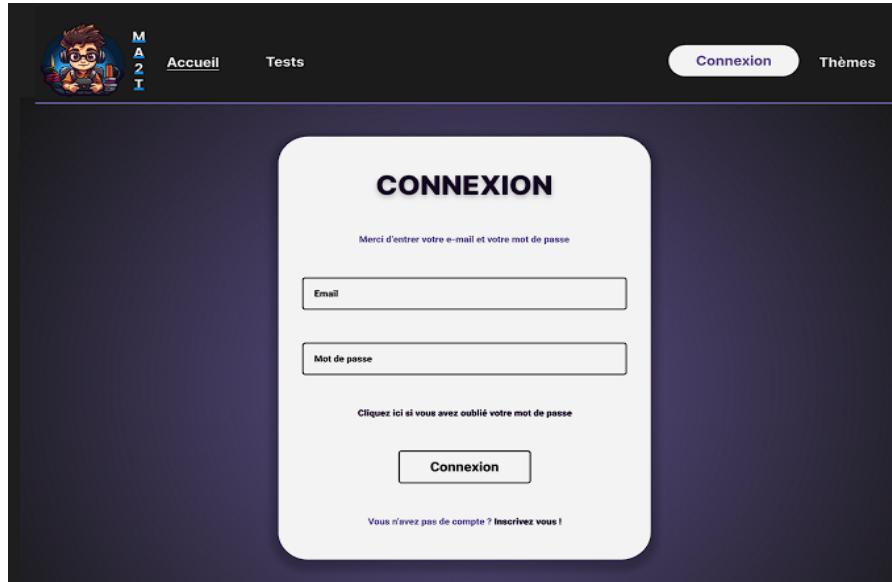
Pour maquetter le site web j'ai utilisé donc Figma, et Coolors.co. J'ai aussi réalisé plusieurs versions de certains éléments, par exemple le header :



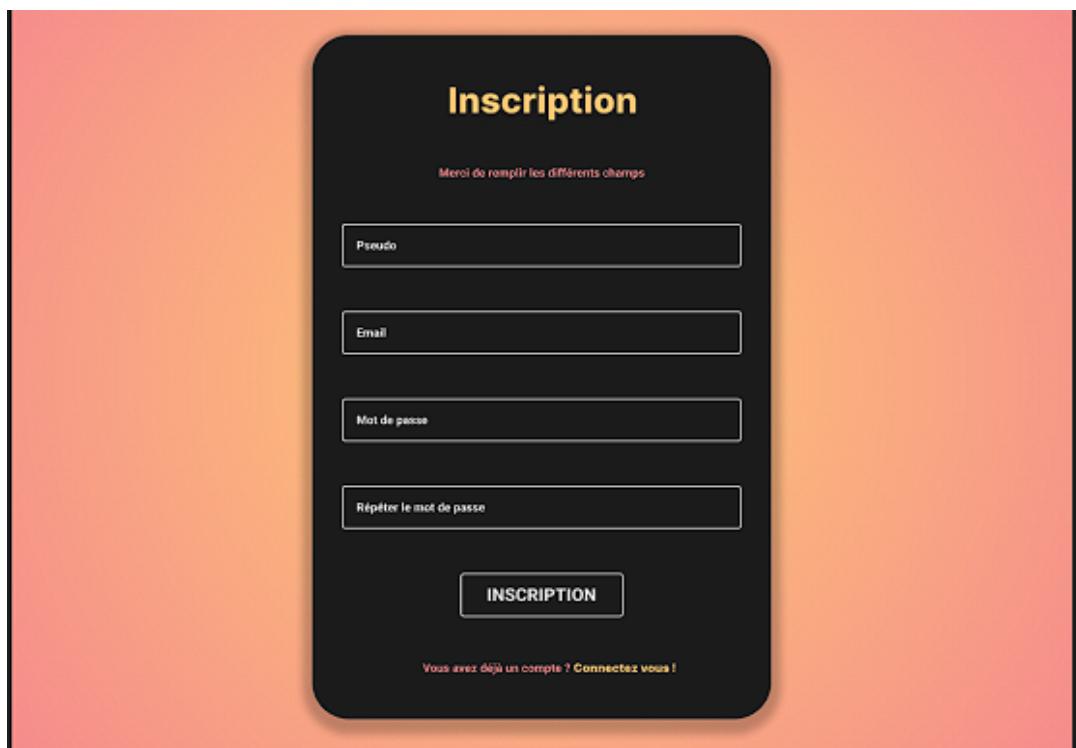
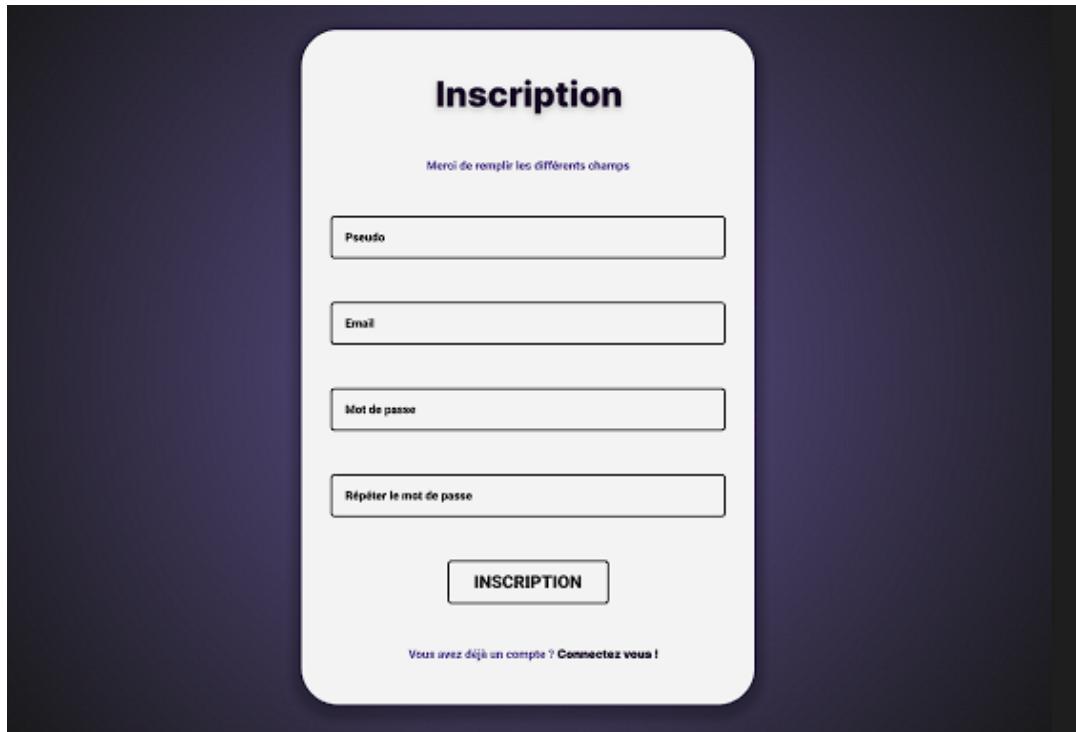
J'ai aussi maquetté la connexion avec un dégradé aussi fait sur coolors.co :



En reliant les deux éléments nous sommes donc sur cette version finale :



Je vais réutiliser le design de la connexion pour l'inscription, la réinitialisation du mot de passe et le formulaire de contact.



Mise en place du projet

Les différents outils utilisés

Pour la mise en place du projet j'ai dû utiliser plusieurs outils. En premier temps j'ai utilisé GitLab, c'est là où mon projet est stocké et grâce aux différentes branches et aux différents commits je peux me repérer plus facilement et l'organisation est meilleure. De plus pour m'assigner les tâches en fonction des journées j'ai utilisé Trello qui est un outil de gestion de projet.



Ensuite pour la partie design, le wireframe et la maquette ont été fait avec Figma et pour la correspondance des couleurs en fonction du contraste j'ai utilisé coolors.co.



Enfin pour le développement du projet j'ai utilisé Visual Studio Code Xampp pour mettre en place le serveur local et enfin Bootstrap pour la partie front de l'application.



Visual Studio Code



XAMPP



Bootstrap

Les commandes symfony

Avant de commencer à développer le projet il faut bien sûr l'initialiser. Pour ce faire nous devons d'abord entrer la ligne de commande suivante :

`symfony new my_project_directory --webapp` : nous devons remplacer `my_project_directory` avec le nom souhaité, cette ligne de commande va permettre de créer l'application symfony.

Ensuite nous devons créer les assets/js etc... : composer require `symfony/webpack-encore-bundle` suivi de `npm install`, `npm run build` et `composer install`.

Seulement après cela nous pouvons installer bootstrap : `npm install sass-loader node-sass --save-dev` ensuite on enlève le commentaire ligne 57 et 60 du `webpack.config.js` :

```
// enables Sass/SCSS support
.enableSassLoader()

// uncomment if you use TypeScript
.enableTypeScriptLoader()
```

Et ensuite faire la commande : `npm install bootstrap`. Et enfin dans le `app.js` et `app.scss` importer Bootstrap :

```
1  @import "~bootstrap/scss/bootstrap";

// any CSS you import will output into a single css file (app.css in this case)
import 'bootstrap';
import './styles/app.scss';
```

Et après tout cela pour bien relier le projet à notre base de données, ligne 28 nous devons l'importer :

```
26  # DATABASE_URL="sqlite:///kernel.project_dir/var/data.db"
27  # DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8.0.32&charset=utf8mb4"
28  DATABASE_URL="mysql://root:@127.0.0.1:3306/site_jv?serverVersion=mariadb-10.4.25"
```

Après toutes ces étapes le projet est prêt à être lancé et les controllers/entités sont prêtes à être créés.

Changement des thèmes

Avant de parler du plus important, faisons un tour dans une des fonctionnalités du site : Le changement des thèmes. Comme vu précédemment dans le moodboard les thèmes sont assigné et inspiré des personnages Marvel.

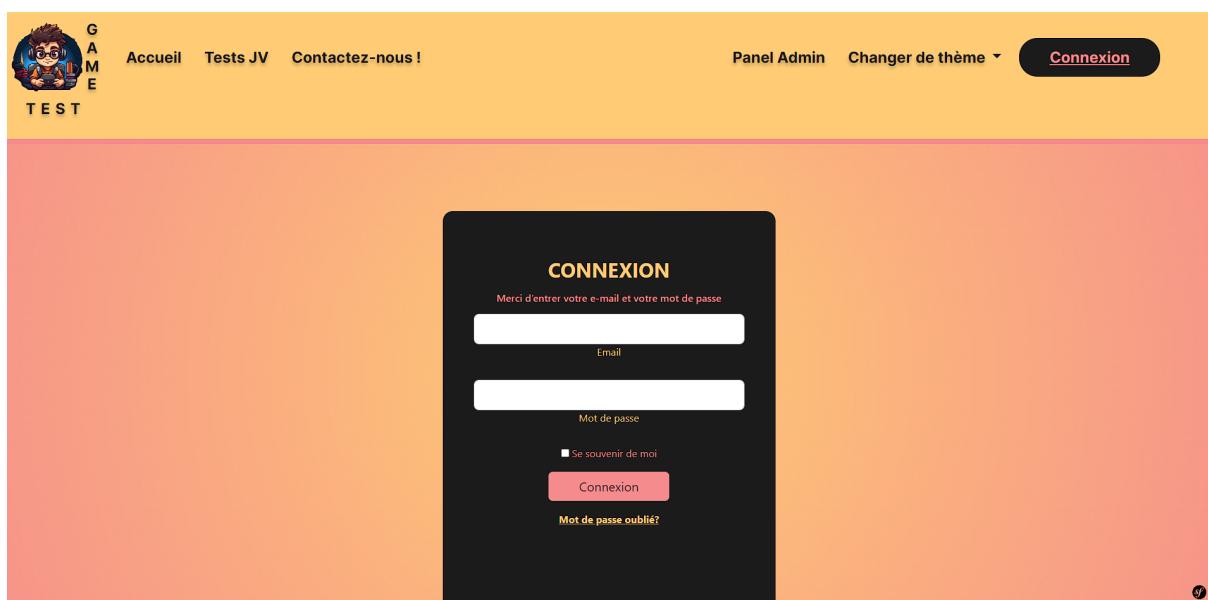
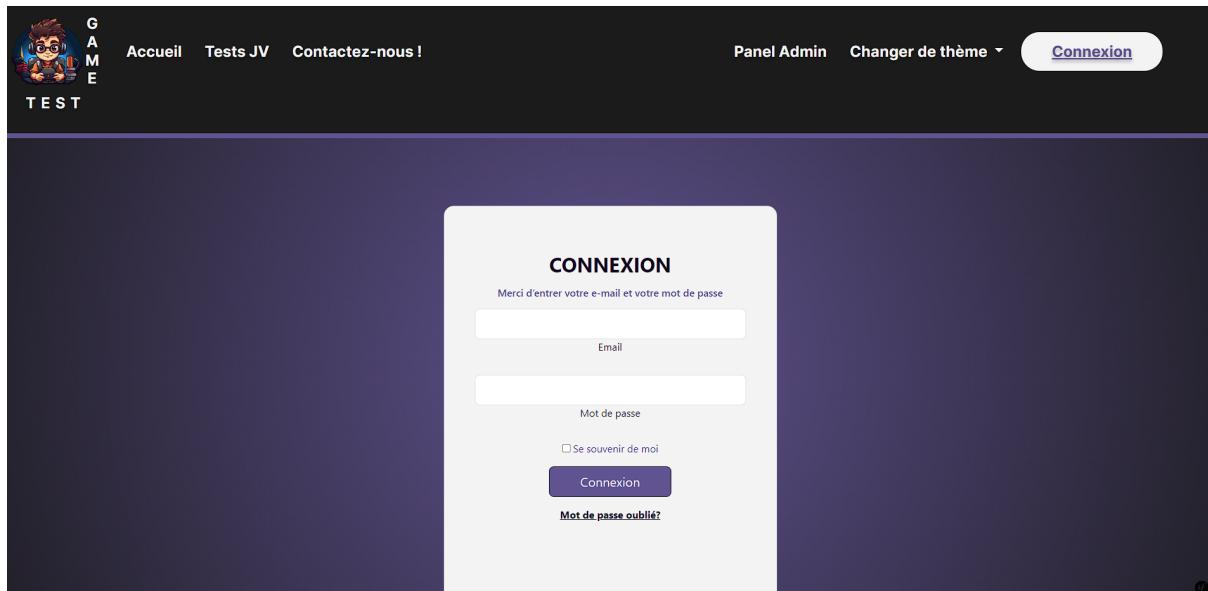


Et dans le Header l'utilisateur peut choisir son thème, et en fonction de son choix toutes les couleurs du site vont être différentes sur chaque page :

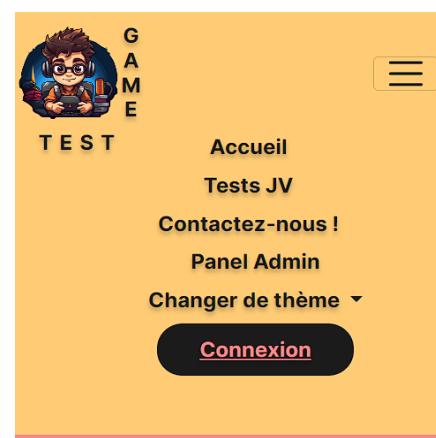
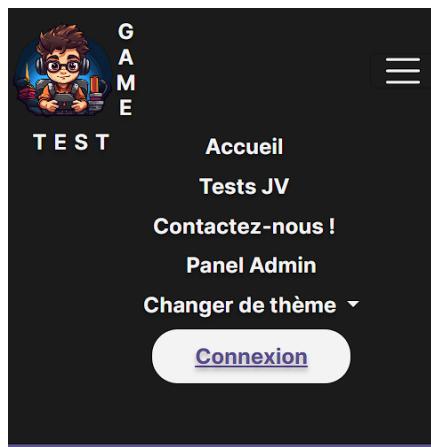
Voici le Header avec le choix de thèmes :



Et voici un autre exemple avec la connexion :



Bien sur le header est responsive :



Voici maintenant le code avec la logique de changement de thème. Ce script Javascript est dans le code de base.html.twig pour qu'il soit utilisé sur tout le site.

On définit la fonction et ses variables :

```
// Fonction pour définir le thème
function setTheme(theme) {
    // Récupérer les éléments du DOM nécessaires
    var themeCSS = document.getElementById('headerThemeCSS'); // Récupère le lien vers les styles du header
    var bodyThemeCSS = document.getElementById('bodyThemeCSS'); // Récupère le lien vers les styles du corps
    var appBody = document.getElementById('appBody'); // Récupère l'élément du corps de la page
    var currentPage = appBody.dataset.page; // Récupère le nom de la page actuelle à partir des données attribuées
```

Si thème = black panther alors modifier le css du header et du body (header_bp.css et body_bp.css) et il met le background dégradé (qui est une image) sur les pages renseigné dans le if currentPage... Ensuite il modifie la variable de noir car il y'avait un problème là dessus j'ai donc du mettre une variable appelé blackF et la changé par celle que je voulais.

Il fait de même pour l'autre thème.

```
// Vérifier le thème pour appliquer les styles appropriés
if (theme === 'black panther') {
    // Appliquer les styles spécifiques au thème Black Panther
    themeCSS.href = "{{ asset('css/header_bp.css') }}"; // Modifie le lien des styles du header
    bodyThemeCSS.href = "{{ asset('css/body_bp.css') }}"; // Modifie le lien des styles du corps

    // Appliquer un arrière-plan spécifique pour certaines pages
    if (currentPage === 'app_login' || currentPage === 'app_register' || currentPage === 'app_forgot_password_request' || currentPage === 'app_check_email' || currentPage === 'app_contact') {
        appBody.style.backgroundImage = "url('{{ asset('images/gradient_bp.png') }}')";
        appBody.style.backgroundSize = "cover";
        appBody.style.backgroundRepeat = "no-repeat";
        appBody.style.backgroundPosition = "center";
    }
    document.documentElement.style.setProperty('--blackF', 'var(--black)');
} else if (theme === 'stark industries') {
    // Appliquer les styles spécifiques au thème Stark Industries
    themeCSS.href = "{{ asset('css/header_si.css') }}"; // Modifie le lien des styles du header
    bodyThemeCSS.href = "{{ asset('css/body_si.css') }}"; // Modifie le lien des styles du corps

    // Appliquer un arrière-plan spécifique pour certaines pages
    if (currentPage === 'app_login' || currentPage === 'app_register' || currentPage === 'app_forgot_password_request' || currentPage === 'app_check_email' || currentPage === 'app_contact') {
        appBody.style.backgroundImage = "url('{{ asset('images/gradient_si.png') }}')";
        appBody.style.backgroundSize = "cover";
        appBody.style.backgroundRepeat = "no-repeat";
    }
}
```

Ensuite on enregistre le thème sélectionné dans un stockage local (localStorage) et enfin récupère le thème enregistré dans le stockage local pour l'appliquer.

```

    // Enregistrer le thème sélectionné dans le stockage local
    localStorage.setItem('theme', theme);
}

// Repérer le clic sur le bouton "Black Panther Theme"
document.getElementById('blackPantherTheme').addEventListener('click', function() {
    setTheme('black panther'); // Appelle la fonction pour définir le thème Black Panther
});

// Repérer le clic sur le bouton "Stark Industries Theme"
document.getElementById('starkIndustriesTheme').addEventListener('click', function() {
    setTheme('stark industries'); // Appelle la fonction pour définir le thème Stark Industries
});

// Récupérer le thème enregistré dans le stockage local et l'appliquer
var savedTheme = localStorage.getItem('theme'); // Récupère le thème enregistré
if (savedTheme) {
    setTheme(savedTheme); // Applique le thème enregistré
}
```

La base de données

Schéma relationnel

Avant de créer les entités, et la base de données il est important de faire un schéma relationnel en premier temps pour étudier les besoins et savoir quelles entités vont être utilisées et reliées.

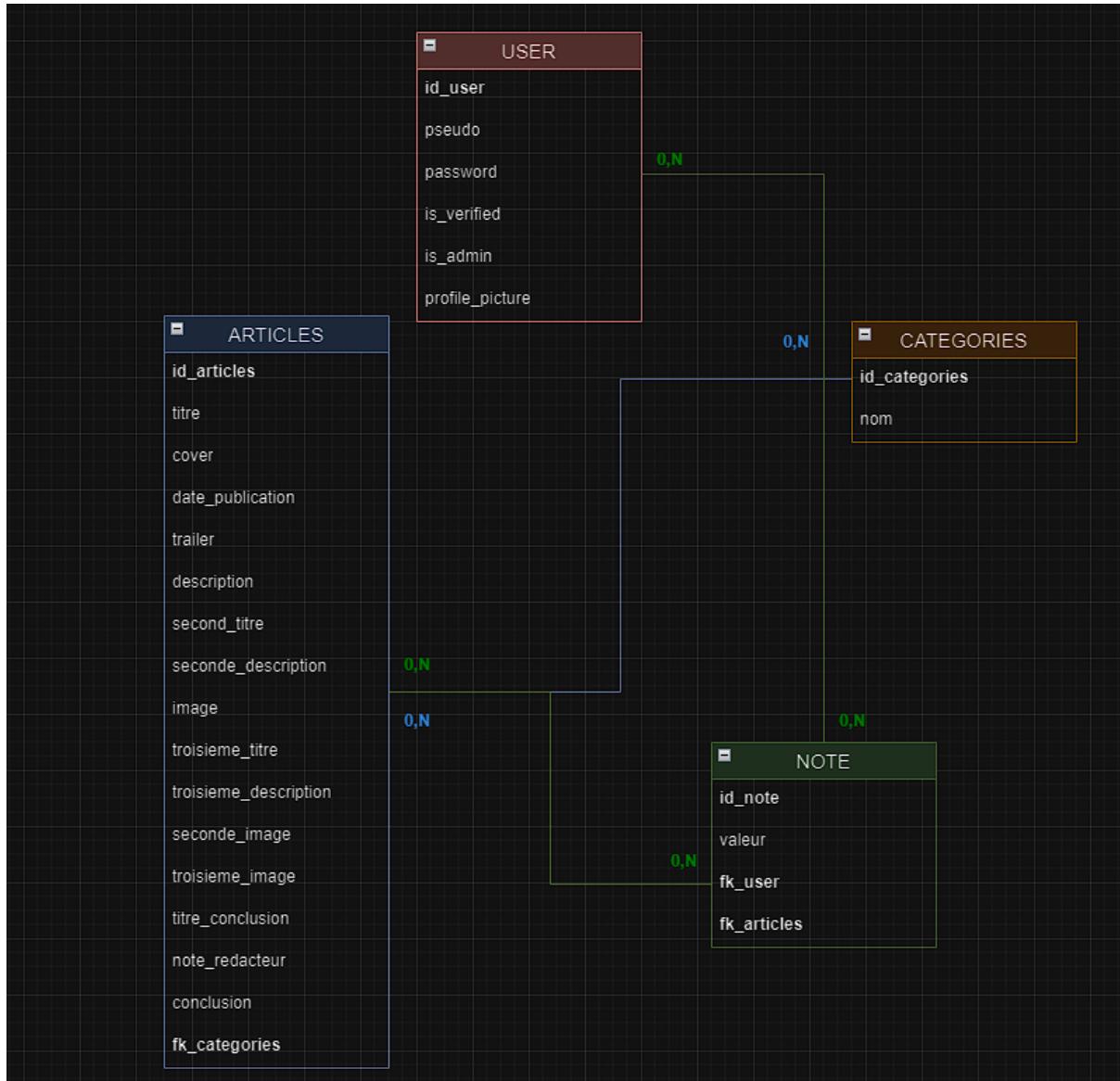
Voici donc le schéma relationnel :

USER	ARTICLES	CATEGORIES	NOTE
id_user	id_articles		
pseudo	titre		
password	cover		
is_verified	date_publication		
is_admin	trailer		
profile_picture	description		
	second_titre		
	seconde_description		
	image		
	troisieme_titre		
	troisieme_description		
	seconde_image		
	troisieme_image		
	titre_conclusion		
	note_redacteur		
	conclusion		
	fk_categories		

J'ai mis autant de duplication dans l'entité "articles" car je veux que sur l'article il y'aït plusieurs images, plusieurs titres et plusieurs descriptions. Comme on pourrait le voir sur d'autres sites de test en tout genre.

Et aussi nous le verrons plus tard mais cela rajoute un côté modelable à la page article. Et propose aussi aux rédacteurs de personnaliser un petit peu leurs articles. De ce fait ils peuvent ne pas mettre d'images, mettre 2 images et 1 seul titre etc....

Schéma MCD



- ❖ Les articles contiennent 0 à N catégories, et inversement.
- ❖ La note est attribuée à 0 à N articles, et inversée.
- ❖ Et enfin L'utilisateur peut mettre 0 à N notes et les notes sont émises par 0 à N utilisateurs.

Création des entités

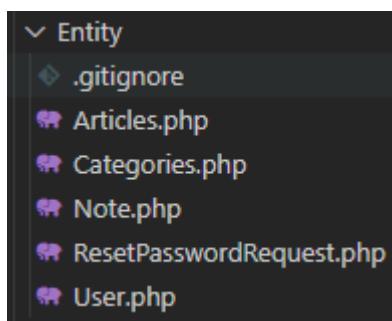
Pour la création des entités il est important d'avoir ouvert Xampp sinon on ne peut pas se connecter à la base de données de notre serveur local.

Maintenant que nous connaissons les entités et leurs relations, nous devons les créer dans notre base de données afin de pouvoir les gérer et les utiliser dans notre projet GameTest.

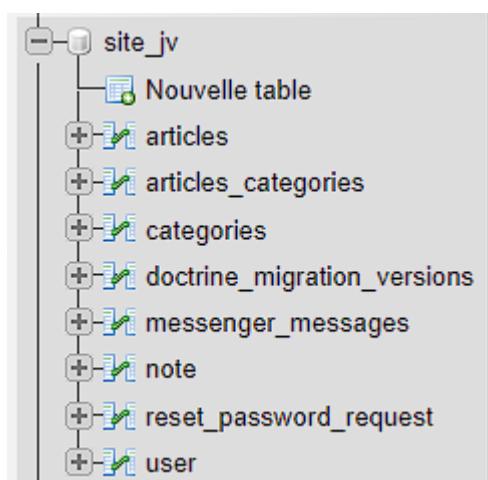
On commence donc à créer l'entité avec la commande symfony : “symfony console make:entity nom entité”, ensuite il faut remplir les propriétés que nous voulons mettre (par exemple pour articles on mets titre, string et non-null).

Pour l'entité de l'utilisateur c'est différent car nous faisons un “symfony console make:user” et ensuite il faut remplir les propriétés.

Quand les Entités sont créés elles sont stocké dans le dossier Entity de notre projet :



Ensuite nous faisons “php bin/console make:migration” puis “php bin/console doctrine:migrations:migrate” pour sauvegarder dans notre base de données.



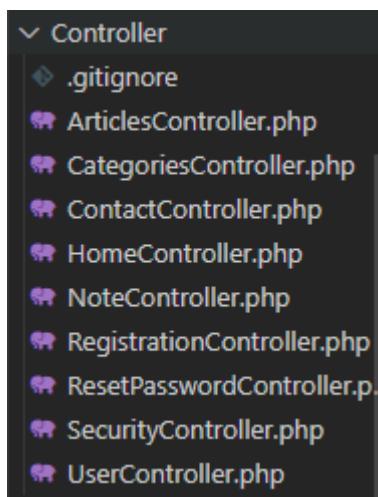
Si nous voulons maintenant créer des entités articles, les modifier ou les supprimer nous allons faire un CRUD (Create Read Update Delete).

Pour ce faire :

“symfony console make:crud nom_entité”

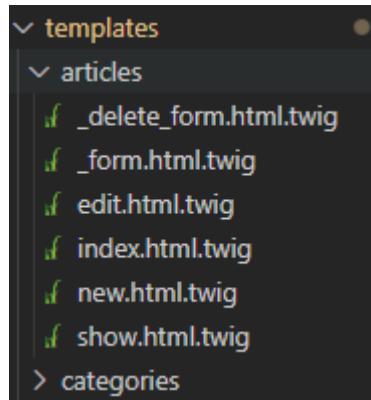
Nous aurons alors plusieurs fichiers de créés dans plusieurs dossiers.

Tout d'abord dans le dossier “Controller” :



Un contrôleur est généré pour gérer les différentes actions CRUD associées à l'entité.

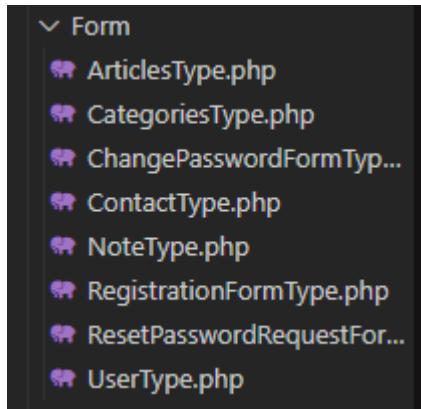
Ensuite dans le dossier "templates/nom_entité" :



Les fichiers de templates Twig sont créés pour l'affichage des pages HTML.
Chacun de ses fichiers à son action spécifique :

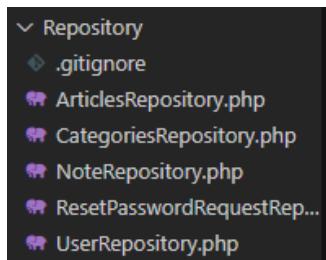
- ❖ index.html.twig: Affiche la liste des entités.
- ❖ new.html.twig: Affiche le formulaire pour créer une nouvelle entité.
- ❖ edit.html.twig: Affiche le formulaire pour éditer une entité existante.
- ❖ show.html.twig: Affiche les détails d'une entité.
- ❖ _delete_form.html.twig: Est un formulaire spécifique pour la suppression d'une entité.
- ❖ _form.html.twig: Est un template partiel pour le formulaire, utilisé dans les pages

Ensuite il y'a des fichiers dans le dossier "Form" :



Il définit les champs et les options du formulaire associé à l'entité.

Et enfin le dossier "Repository" :



Il fournit des méthodes et des requêtes prédéfinies pour interagir avec les entités et les données stockées dans la base de données.

L'utilisateur

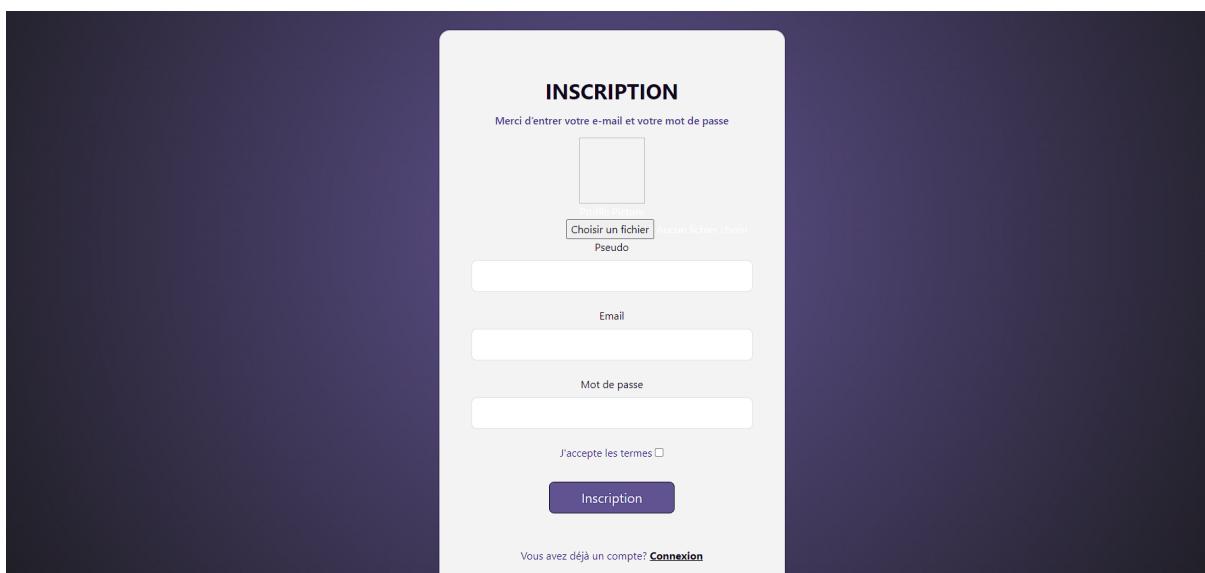
L'inscription

Avant de se connecter, l'utilisateur doit tout naturellement s'inscrire, car il ne possède pas de compte. De ce fait, il va devoir compléter un formulaire d'inscription.

Pour créer ce formulaire d'inscription il doit enter la ligne de commande suivante :

php bin/console make:registration-form, et remplir le formulaire. Plusieurs fichés vont être créées dont le RegistrationFormType, celui qui nous permet de mettre ce qu'il va être mis dans le formulaire d'inscription du site.

Le site acceptant l'anonymat, aucun besoin d'entrer son nom, prénom ou encore même adresse.



Ici donc l'utilisateur doit simplement entrer son pseudo, son image de profil s'il le souhaite, son e-mail (vérifié par un REGEX) et son mot de passe qui se doit d'être fort (aussi vérifié par un REGEX).

Les REGEX

Les REGEX sont une chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles. Dans notre cas à nous un REGEX sera utilisé en contrainte dans notre fichier RegistrationFormType, pour empêcher par exemple une fausse adresse mail :

```
->add('email', EmailType::class, [
    'constraints' => [
        new Regex('/^([a-zA-Z0-9._%+-]+@[outlook|gmail|yahoo]\.(com|fr)$)', 'L\'adresse e-mail n\'est pas valide')
    ]
],
```

Ici le REGEX va dire à l'utilisateur d'entrer une adresse mail qui comporte :

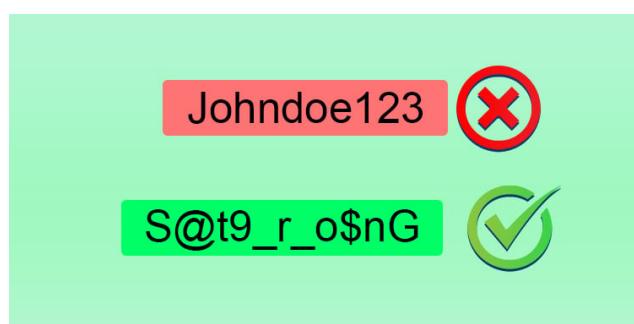
Au moins un caractère parmi les lettres majuscules et minuscules (a-z, A-Z), les chiffres (0-9), et certains caractères spéciaux comme ".", "_", "%", "+", et "-". Le symbole "@" suivi de outlook,gmail ou yahoo et enfin un .com ou .fr.

Les email comme Test@Test.fr ne permettent donc pas à l'utilisateur de s'inscrire.

Notre deuxième REGEX est celui du mot de passe :

```
->add('plainPassword', PasswordType::class, [
    'mapped' => false,
    'attr' => ['autocomplete' => 'new-password'],
    'constraints' => [
        new Regex('/^(?=.*[a-z])(?=.*[A-Z])(?=.*[\d])(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/', 'Il faut un mot de passe de 8 caractères avec une Majuscule, une minuscule, un chiffre et un caractère spécial.')
    ],
],
```

Celui-là, demande à l'utilisateur d'entrer un mot de passe “fort”, comprenant au moins 8 caractères avec une Majuscule, une minuscule, un chiffre et un caractère spécial.



L'envoi de mail

Lors de son inscription l'utilisateur va devoir confirmer son compte pour accéder à certains contenus comme par exemple donner sa note. Le mail est envoyé par un système Mailer. Symfony nous le fait pour nous lorsque nous faisons le formulaire d'inscription.

Mais nous avons quand même besoin de "composer require symfony/mailer".

Pour recevoir le mail j'ai utilisé le serveur SMTP (simple transfert Protocol) qui se nomme mailtrap.io. Mailtrap nous donne alors un mailer DSN que nous devons alors rentrer dans notre projet symfony :

```
MAILER_DSN=smtp://819cb164f44613:*****5128@sandbox.smtp.mailtrap.io:2525
```

Que nous mettons dans la ligne 40 de notre .env :

```
37 #####< symfony/messenger #####
38
39 #####> symfony/mailers #####
40 MAILER_DSN=smtp://819cb164f44613:*****5128@sandbox.smtp.mailtrap.io:2525encryption=tls&auth_mode=login
41 #####< symfony/mailers #####
42 |
```

L'envoi de mail lors de la création du formulaire d'inscription modifie le RegistrationController pour y ajouter la vérification de mail.

```
// generate a signed url and email it to the user
$this->emailVerifier->sendEmailConfirmation('app_verify_email', $user,
    (new TemplatedEmail())
        ->from(new Address('no-reply@MA2Tblog.fr', 'No Reply'))
        ->to($user->getEmail())
        ->subject('Please Confirm your Email')
        ->htmlTemplate('registration/confirmation_email.html.twig')
);
// do anything else you need here, like send an email

return $this->redirectToRoute('app_home');
}

return $this->render('registration/register.html.twig', [
    'registrationForm' => $form->createView(),
]);
}

#[Route('/verify/email', name: 'app_verify_email')]
public function verifyUserEmail(Request $request, TranslatorInterface $translator): Response
{
    $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

    // validate email confirmation link, sets User::isVerified=true and persists
    try {
        $this->emailVerifier->handleEmailConfirmation($request, $this->getUser());
    } catch (VerifyEmailExceptionInterface $exception) {
        $this->addFlash('verify_email_error', $translator->trans($exception->getReason(), [], 'VerifyEmailBundle'));
    }

    return $this->redirectToRoute('app_register');
}

// TODO Change the redirect on success and handle or remove the flash message in your templates
$this->addFlash('success', 'Your email address has been verified.');

return $this->redirectToRoute('app_register');
}
```

Le premier morceau de code génère une URL signée et l'envoie par e-mail à l'utilisateur. L'utilisateur va recevoir le template HTML TWIG : “registration/confirmation_email.html.twig”

Ensuite la deuxième partie du code quant à elle valide le lien de confirmation de l'e-mail, définit User::isVerified à true et persists, si l'utilisateur valide son email.

La photo de profil

La photo de profil n'est pas un élément très important du site, ni même de l'utilisateur, car le site ne propose pas aux utilisateurs de communiquer ensemble.

J'ai trouvé néanmoins que d'essayer de faire cette fonctionnalité que je n'avais jamais faite, était une bonne chose. De plus, ça permet au site d'avoir ce côté de personnalisation qui me tenait à cœur.

J'ai, lors de la création de l'utilisateur mis la propriété de l'image de profil que j'ai mis en string. Sauf que nous voulons une image.

Nous allons donc modifier cela dans le UserType :

```
->add('profilePicture', FileType::class, [
    'label' => 'Profile Picture',
    'mapped' => false,
    'required' => false,
    'constraints' => [
        new Image([
            'maxSize' => '1024k',
            'mimeTypesMessage' => 'Veuillez télécharger une image valide',
        ]),
    ],
]);
```

Le bout de code configure le téléchargement de l'image de profil dans le formulaire. Il définit des options telles que le label du champ, le fait que le champ n'est pas mappé automatiquement, et une contrainte pour valider que le fichier téléchargé est une image et respecte une taille maximale.

Il n'est pas mappé car nous devons gérer manuellement la manipulation du fichier téléchargé.

L'utilisateur doit importer sa photo de profil dès lors de son inscription, j'ai donc fait de même pour RegistrationFormType :

```
->add('profilePicture', FileType::class, [
    'label' => 'Profile Picture',
    'mapped' => false,
    'required' => false,
    'constraints' => [
        new Image([
            'maxSize' => '1024k',
            'mimeTypesMessage' => 'Veuillez télécharger une image valide',
        ]),
    ],
]);
```

Maintenant dans nos deux controller nous devons définir la méthode pour l'importation de l'image.

Dans le RegistrationController :

Le code en premier temps récupère le fichier envoyé depuis le formulaire

```
$uploadedFile = $form['profilePicture']->getData();
```

Ensuite, il vérifie si un fichier a été effectivement téléchargé.

```
if ($uploadedFile) {
```

Il Définit le répertoire de destination où les images de profil seront stockées

```
$destination =
$this->getParameter('kernel.project_dir').'/public/uploads/profile_pictures';
```

Que nous mettrons en paramètre dans services.yml pour une meilleure facilité

```
parameters:
    uploads.profile_pictures:
        '%kernel.project_dir%/public/uploads/profile_pictures'
```

Récupère le nom original du fichier sans l'extension

```
$originalFilename = pathinfo($uploadedFile->getClientOriginalName(),
PATHINFO_FILENAME);
```

Génère un nouveau nom de fichier unique en utilisant le nom original, un identifiant unique et l'extension originale

```
$newFilename =
$originalFilename.'-'.$uniqid().'.'.$uploadedFile->guessExtension();
```

Déplace le fichier téléchargé vers le répertoire de destination avec le nouveau nom de fichier

```
$uploadedFile->move(
    $destination,
    $newFilename
);
```

Met à jour le chemin de l'image de profil dans l'objet utilisateur

```
$user->setProfilePicture('/uploads/profile_pictures/' . $newFilename);
```

Persiste les modifications apportées à l'objet utilisateur dans la base de données

```
$entityManager->persist($user);
```

Et enfin il exécute les opérations persistées dans la base de données

```
$entityManager->flush();
```

Nous devons aussi ajouter la logique au contrôleur de l'utilisateur. Dans le new et dans le edit :

```
#[Route('/new', name: 'app_user_new', methods: ['GET', 'POST'])]
public function new(Request $request, UserRepository $userRepository): Response
{
    $user = new User();
    $form = $this->createForm(UserType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        /** @var UploadedFile $profilePictureFile */
        $profilePictureFile = $form->get('profilePicture')->getData();

        if ($profilePictureFile) {
            $originalFilename = pathinfo($profilePictureFile->getClientOriginalName(), PATHINFO_FILENAME);
            $safeFilename = \transliterator_transliterate('Any-Latin; Latin-ASCII; [^A-Za-z0-9_] remove; Lower()', $originalFilename);
            $newFilename = $safeFilename.'-'.$_uniqueid().'.'.$profilePictureFile->guessExtension();

            try {
                $profilePictureFile->move(
                    $this->getParameter('uploads.profile_pictures'),
                    $newFilename
                );
            } catch (FileException $e) {
            }
        }

        $user->setProfilePicture('uploads/profile_pictures/' . $newFilename);
    }

    $userRepository->save($user, true);

    return $this->redirectToRoute('app_user_index', [], Response::HTTP_SEE_OTHER);
}

return $this->renderForm('user/new.html.twig', [
```

Enfin ce n'est pas tout car maintenant nous voulons que l'utilisateur puisse choisir son fichier et qu'il soit prévisualisé lors de l'inscription. Il faut donc un petit peu de JS pour cela :

Nous allons d'abord attendre que le contenu de la page soit chargé

```
document.addEventListener('DOMContentLoaded', function() {
```

On ajoute un écouteur d'événement (EventListener) au champ de formulaire de téléchargement d'image de profil

```
document.getElementById('{{ registrationForm.profilePicture.vars.id }}').addEventListener('change', function(e) {
```

On récupère le fichier sélectionné dans le champ de formulaire

```
var file = e.target.files[0];
```

On crée un objet FileReader pour lire le contenu du fichier

```
var reader = new FileReader();
```

On définit une fonction à exécuter lorsque la lecture du fichier est terminée

```
reader.onloadend = function() {
```

Met à jour l'élément HTML avec l'ID 'image_preview' pour afficher l'aperçu de l'image

```
document.getElementById('image_preview').src = reader.result;
```

Vérifie si un fichier a été sélectionné

```
if (file) {
```

Lit le contenu du fichier en tant que URL de données

```
reader.readAsDataURL(file);  
} else {
```

Et enfin si aucun fichier n'est sélectionné, on réinitialise l'aperçu de l'image

```
document.getElementById('image_preview').src = "";
```

Nous allons mettre ce script dans user/_form.html.twig pour l'édition également.

La Connexion

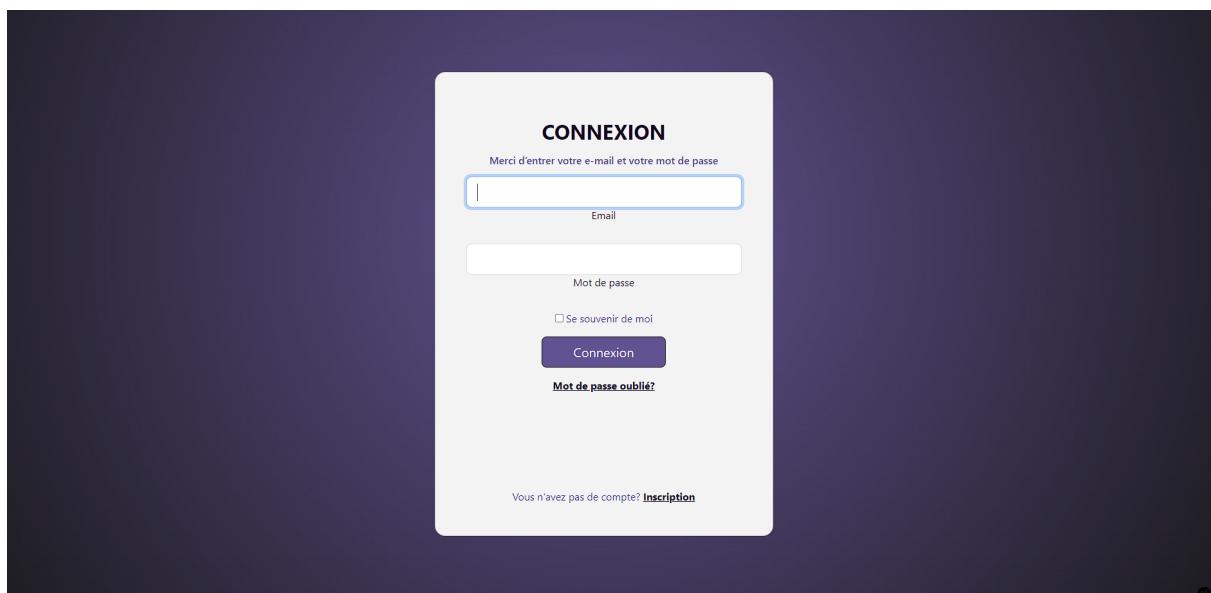
Après s'être inscrit, l'utilisateur doit pouvoir se connecter. Pour ajouter une connexion à notre projet symfony nous devons rentrer la ligne de commande suivante :

```
php bin/console make:auth
```

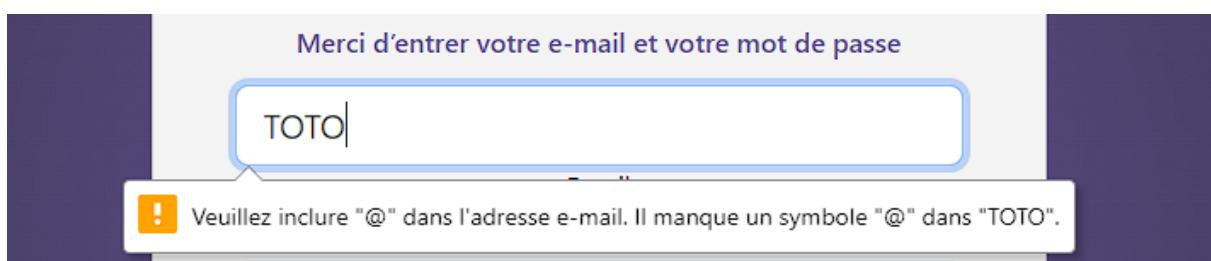
Cela crée donc un formulaire de connexion que l'on peut personnaliser par la suite.

Le formulaire de connexion ne comporte aucun paramètre plus compliqué, c'est l'inscription le plus complexe:

L'utilisateur doit donc entrer e-mail et mot de passe :



Comme l'inscription si un champ est mauvais une erreur viendra avertir l'utilisateur :



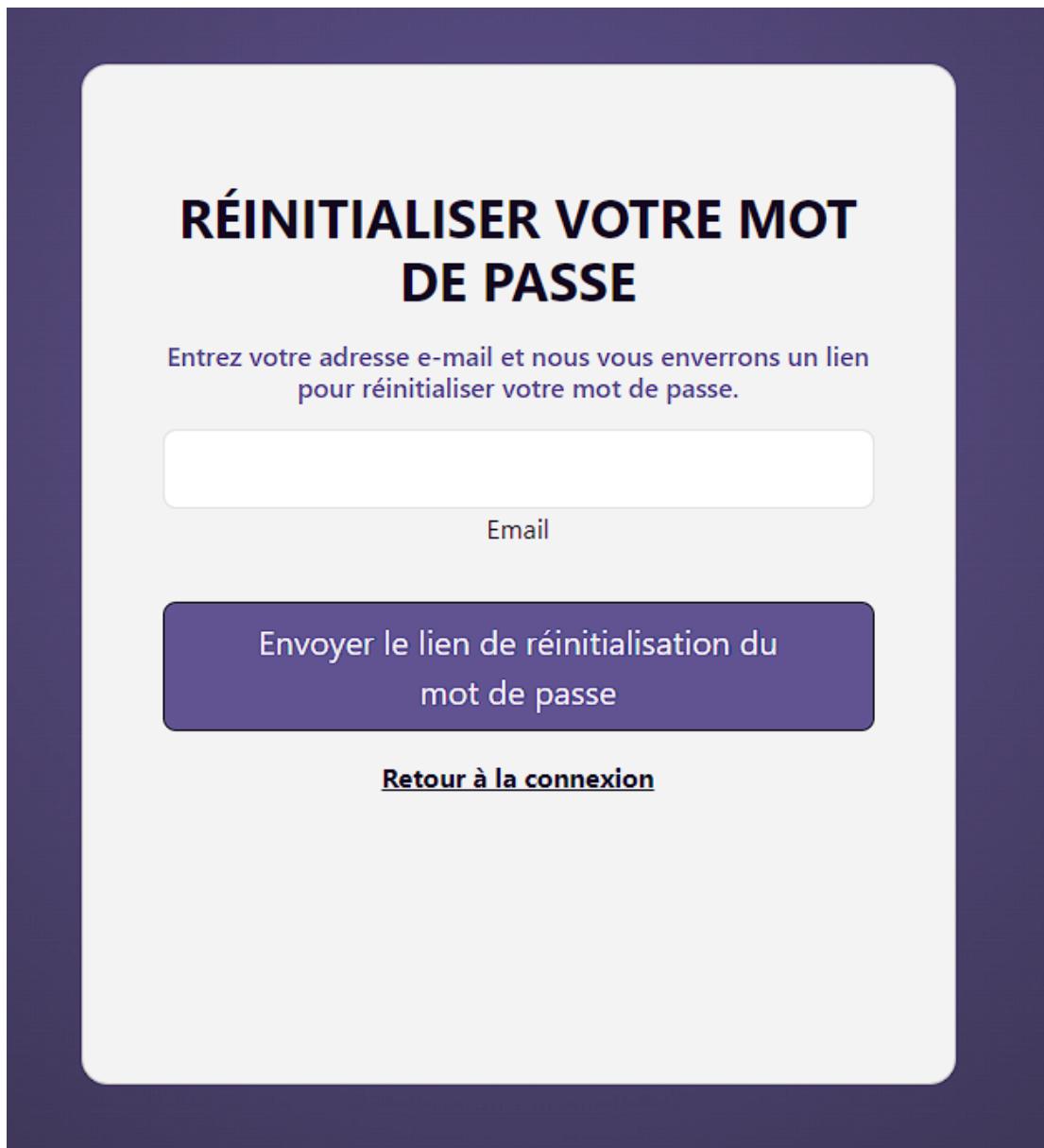
La réinitialisation du mot de passe

Si l'utilisateur oublie son mot de passe alors il peut le réinitialiser.

Pour l'ajouter à notre projet symfony nous devons entrer dans le terminal :

“php bin/console make:reset-password”

Et voilà le formulaire est créé, l'utilisateur doit simplement envoyer son adresse mail et un formulaire lui sera alors envoyé :



Les articles

Création de l'entité et du CRUD

Avant de voir réellement les articles, nous devons les créer et leur crud, avec les commandes vues précédemment :

“symfony console make:entity nom entité” - pour créer l'entité

“symfony console make:crud nom_entité” - pour créer le crud

On enregistre ensuite dans la base de données :

“php bin/console make:migration”

“php bin/console doctrine:migrations:migrate”

Nous avons donc les articles :

		id	titre	cover	date_publication	trailer	description	second_titre	seconde_description	image	troisieme_titre	troisieme_description	si
<input type="checkbox"/>	Éditer Copier Supprimer	1	TEST	gearsofwar4gamecover30650-6491c23d3c050.jpg	2018-01-01	<iframe width="560" height="315" src="https://www....">	Test de jeux :	TEST	Test de jeux : J'aime beaucoup	people_playground-64963b31b6c59.webp	TEST	Test de jeux : J'aime beaucoup	gi
<input type="checkbox"/>	Éditer Copier Supprimer	2	TEST2	gamingtta5cover-649b34d05ed11.jpg	2018-01-01	<iframe width="560" height="315" src="https://www....">	zflazflafaf	NULL	NULL	76af52d_1685522072454telera ma undefined12-6499d144e...	NULL	NULL	N

On fait pareil pour les catégories :

		id	nom	Action
<input type="checkbox"/>	Éditer Copier Supprimer	1	Tir	
<input type="checkbox"/>	Éditer Copier Supprimer	2	Tir	

Pour relier les deux ensemble pour devons, faire une relation dans le symfony make:entity :

```
New property name (press <return> to stop adding fields):
> category

Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> Category

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne
```

Symfony crée donc une relation et pour afficher une catégorie dans notre article, on peut l'ajouter de cette manière :

```
{% for category in article.categories %}
    <span class="badge category-color-bg">{ {
category.nom } }</span>
{% endfor %}
```

et pour l'ajouter dans un formulaire de création d'articles on l'ajoute dans ArticlesType.php :

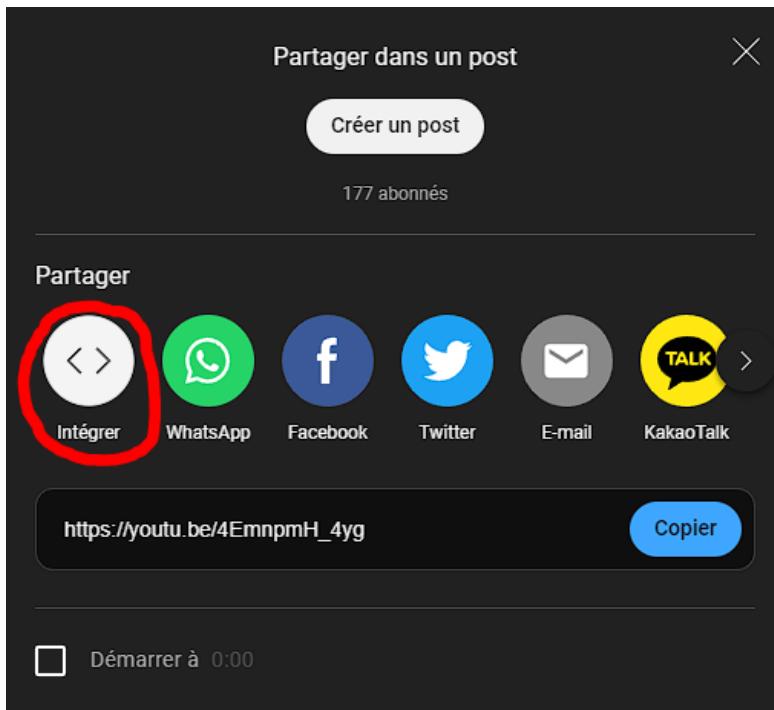
```
->add('categories', EntityType::class, [
    'class' => Categories::class,
    'choice_label' => 'nom',
    'multiple' => true,
    'expanded' => true
])
```

J'ai choisi expanded pour faire des cases à cocher et multiple pour dire que l'article peut avoir plusieurs catégories.

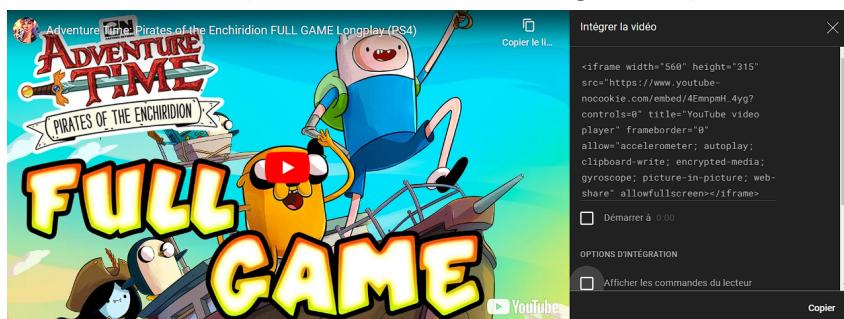
Gestions des images et vidéos

Dans nos articles, des images et une vidéo de bande annonce sont disponibles.

La vidéo elle est gérée par youtube, j'utilise le lien d'intégration donné par youtube :



Quand nous cliquons on a donc l'intégration par un iframe :



Ensuite je stocke ce code dans ma base de donnée :

```
trailer
<iframe
width="560"
height="315"
src="https://www....>
<iframe
width="560"
height="315"
src="https://www....>
```

Et pour une image moins grande j'ai remplacé sa hauteur :

```
<div class="embed-responsive embed-responsive-16by9">
  {{ article.trailer|replace({'height': 'height="396"'})|raw }}
</div>
```

Sa hauteur est maintenant à 396px, mais ça n'empêche pas de changer sa largeur si on le souhaite.

Concernant les images la logique est la même que celle de l'image de profil :

On commence par changer l'image dans le builder de l'ArticlesType.php :

```
->add('cover', FileType::class, [
    'label' => 'Image Cover',
    'mapped' => false,
    'required' => false,
    'constraints' => [
        new Image([
            'maxSize' => '1024k',
            'mimeTypesMessage' => 'Veuillez télécharger une image valide',
        ]),
    ],
])
```

Ensuite dans Articles Controller :

On commence par vérifier si le formulaire a été soumis et s'il est valide.

```
if ($form->isSubmitted() && $form->isValid()) {
```

On récupère ensuite les données du champ de formulaire nommé 'cover'.

```
$coverFile = $form->get('cover')->getData();
```

Vérifie si un fichier a été téléchargé via le champ 'cover'.

```
if ($coverFile) {
```

On récupère ensuite le nom d'origine du fichier sans l'extension.

```
$originalFilename = pathinfo($coverFile->getClientOriginalName(),
PATHINFO_FILENAME);
```

On transforme le nom original en une version sécurisée, compatible avec les caractères latins et minuscules.

```
$safeFilename = transliterator_transliterate('Any-Latin; Latin-ASCII;  
[^A-Za-z0-9_] remove; Lower()', $originalFilename);
```

On génère un nouveau nom de fichier unique en ajoutant un identifiant unique (uniqid) et en conservant l'extension d'origine.

```
$newFilename = $safeFilename . '-' . uniqid() . '.' .  
$coverFile->guessExtension();
```

Ensuite on déplace le fichier téléchargé vers le répertoire de destination spécifié (uploads.cover) avec le nouveau nom de fichier.

```
try {  
    $coverFile->move(  
        $this->getParameter('uploads.cover'),  
        $newFilename  
    );  
} catch (FileNotFoundException $e) {  
  
}
```

Et enfin on associe le nouveau nom de fichier à la propriété "cover" de l'objet article (ou d'une entité similaire).

```
$article->setCover($newFilename);
```

La logique est donc assez similaire et dans le services.yaml j'ai amélioré le paramètres pour rendre plus léger la donnée dans la base :

```
uploads.cover: '%kernel.project_dir%/public/uploads/covers'
```

Et dans la base de données

Utilisateur

profile_picture

uploads/profile_pictures/b1969b6b52bc00d215eae81e7...

Article

cover

gearsofwar4gamecoveri30650-
6491c23d3c050.jpg

Systèmes des notes utilisateurs

Le test d'un rédacteur étant totalement subjectif et prônant l'esprit critique de chacun et la liberté de pensée, j'ai voulu que les utilisateurs aient aussi la possibilité de donner leurs propres notes.

Le fonctionnement est donc très simple, à la fin de l'article, l'utilisateur peut voir un bouton pour ajouter une note, il ajoute donc une note entre 1 et 20 sans besoin de se justifier. A la fin il verra sa note ajouter aux autres pour former une moyenne et visible directement sur la note du jeu.

L'entité note doit être reliée avec l'article et l'utilisateur.

La relation de note et de Articles et de ManyToOne, également ManyToOne pour note et utilisateur.

Ensuite dans le contrôleur de Articles là où on pourra interagir et placer la note, il faut que l'on mette la logique de la note dans le show :

On commence par vérifier si il y'a un utilisateur

```
if (! $user) {  
    //  
}
```

Ensuite, on recherche dans la base de données s'il existe une note qui correspond à cet utilisateur et cet article.

```
$existingRating = $noteRepository->findOneBy([  
    'user' => $user,  
    'articles' => $article,  
]);
```

Si une note existe, la variable \$note la prend. Sinon, une nouvelle instance de la classe Note est créée.

```
$note = $existingRating ?? new Note();
```

On configure maintenant la note créée en définissant l'article et l'utilisateur associés à cette note.

```
$note->setArticles($article);  
$note->setUser($user);
```

Si aucune note existante n'est trouvée, on crée un formulaire pour la note sans options spécifiques.

```
if ($existingRating) {  
    $form = $this->createForm(NoteType::class, $note, [  
    ]);  
} else {  
    $form = $this->createForm(NoteType::class, $note);  
}
```

On traite ensuite les données soumises dans la requête HTTP par le formulaire.

```
$form->handleRequest($request);
```

On vérifie ensuite si le formulaire est valide.

```
if ($form->isSubmitted() && $form->isValid()) {
```

On obtient la valeur de la note :

```
$valeur = $note->getValeur();
```

Et on vérifie que la valeur de la note est entre 0 et 20

```
if ($valeur >= 0 && $valeur <= 20) {
```

Si aucune note n'a été trouvée, on met la nouvelle note dans la base de données. Sinon, on la met à jour.

```
if (!existingRating) {  
    $this->entityManager->persist($note);  
} else {  
  
    $existingRating->setValeur($valeur);  
}  
  
$this->entityManager->flush();  
  
}  
else {  
  
}  
}
```

Et enfin, on retourne le rendu du template 'articles/show.html.twig' avec l'article en question et le formulaire pour la note.

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

De ce fait, maintenant sur l'article, l'utilisateur peut ajouter sa note, seulement s'il est connecté.

Ajouter une note



Valeur

11

Fermer

Ajouter la note

Maintenant nous devons faire un calcul de moyenne pour enfin obtenir la moyenne des notes des utilisateurs :

Pour calculer la moyenne, cet extrait de code vérifie d'abord si l'article a des notes attribuées par les utilisateurs. S'il y en a, il calcule la moyenne de ces notes. Puis il affiche la moyenne et si aucune note n'est associée à l'article, il affiche un message indiquant qu'il n'y a pas de notes pour cet article.

```
{% if article.notes|length > 0 %}  
    {% set totalNotes = 0 %}  
    {% for note in article.notes %}  
        {% set totalNotes = totalNotes + note.valeur %}  
    {% endfor %}  
    {% set moyenne = totalNotes / article.notes|length %}  
  
    <p class="text-justify primary-color">Moyenne des notes des utilisateurs : {{ moyenne }}</p>  
{% else %}  
    <p class="text-justify primary-color">Aucune note des utilisateurs pour cet article pour cet article.</p>  
{% endif %}
```

Recherche d'articles

Pour faciliter l'accès aux articles pour les utilisateurs un système pour rechercher par nom, et/ou par genre est mis en place.

The screenshot shows a search interface with two main sections. On the left, under the heading 'TEST', there is a card for the game 'Gears of War 4'. The card features the game's cover art, which depicts a soldier in a dark, rainy environment with a large mechanical gear in the background. Below the cover art are two buttons: a purple one labeled 'Tir' with a gun icon, and a black one labeled 'Accéder'. On the right, under the heading 'TEST2', there is a card for the game 'Grand Theft Auto V'. This card features a collage of various in-game screenshots, including characters and vehicles. Below the collage are two buttons: a purple one labeled 'Action' with a bomb icon, and a black one labeled 'Accéder'. At the top of the interface, there are three buttons: 'Rechercher par titre', 'Trier par catégorie' (which is highlighted in blue), and 'Rechercher'.

Rechercher par titre

Action 🔍

Rechercher

Supprimer le filtre

TEST2



Action 🔍

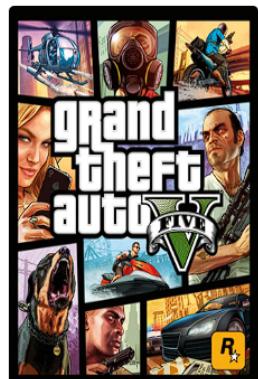
Accéder

2

Trier par catégorie

Rechercher

TEST2



Action 🔍

Accéder

Et voici la logique pour faire cette recherche par nom ou par filtre :

```
<form method="get" class="d-flex align-items-center">
    <div class="form-group mr-2">
        <input type="text" name="search" class="form-control" placeholder="Rechercher par titre" value="{{ app.request.get('search') }}"/>
    </div>
    <div class="form-group mr-2">
        <select name="category" class="form-control" onchange="this.form.submit()">
            <option value="">Trier par catégorie</option>
            {% for category in categories %}
                <option value="{{ category.id }}" {{ category.id == app.request.get('category') ? 'selected' : '' }}>
                    {{ category.nom }}
                </option>
            {% endfor %}
        </select>
    </div>
    <button type="submit" class="btn btn-primary">Rechercher</button>
    {% if app.request.get('category') %}
        <a href="{{ path('app_articles_index') }}" class="btn btn-secondary">Supprimer le filtre</a>
    {% endif %}
</form>
```

On crée un formulaire séparé en deux division (div), une pour la recherche du titre avec un input qui créer une recherche grâce à

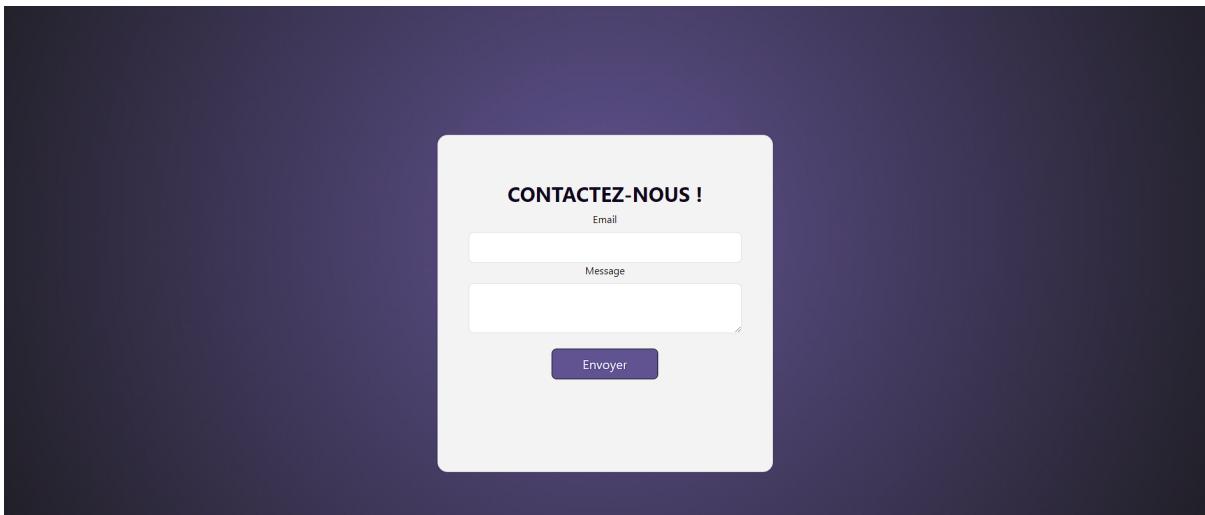
```
value="{{ app.request.get('search') }}
```

Ensuite la deuxième division permet la recherche par catégories avec la même logique mais pour les catégories

Formulaire de contact

Enfin le dernier élément du site est le formulaire de contact, il permet à l'utilisateur d'envoyer un email si jamais il rencontre un problème pendant sa navigation.

Le formulaire de GameTest est très simple et propose d'entrer son e-mail et son message :



Pour ce formulaire rien de plus simple, on crée un formulaire sous symfony grâce à la commande "php bin/console make:form Contact". Ainsi qu'un controller.

Et ensuite grâce au mailer précédemment utilisé dans la création de l'utilisateur et grâce à la documentation symfony.

```

class ContactController extends AbstractController
{
    #[Route('/contact', name: 'app_contact')]
    public function index(Request $request, MailerInterface $mailer): Response
    {
        $form = $this->createForm(ContactType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $data = $form->getData();

            $address = $data['email'];
            $content = $data['content'];

            $email = (new Email())
                ->from($address)
                ->to('admin@MA2T.fr')
                ->subject('Demande de contact')
                ->text($content);

            $mailer->send($email);

            $this->addFlash('success', 'Email envoyé avec succès !');

            return $this->redirectToRoute('app_contact');
        }

        return $this->render('contact/index.html.twig', [
            'controller_name' => 'ContactController',
            'formulaire' => $form->createView(),
        ]);
    }
}

```

Ce code vérifie si le formulaire a été soumis et s'il est valide. Si c'est le cas, il récupère les données soumises, et crée un e-mail en utilisant les données et les envoie à l'adresse 'admin@MA2T.fr' via le service de messagerie (\$mailer). Ensuite, il ajoute un message si l'opération a réussi, et si le formulaire n'a pas été soumis ou n'est pas valide, il renvoie le rendu du template 'contact/index.html.twig' avec le formulaire de contact.

Conclusion

Pour conclure, ce projet fut pour moi une réelle très belle expérience. J'ai pu acquérir de nouvelles compétences comme par exemple la gestion d'image et de mail et aussi développer mes connaissances en Symfony. Travailler seul est aussi un très bon moyen pour l'organisation avec l'obligation de planifier et de bien organiser le git ou encore même le Trello. J'ai aussi pu développer mon esprit critique et encore plus ma passion pour les jeux vidéo. Mes compétences en design pour la réalisation du site ont pu aussi être améliorées. J'ai appris aussi à utiliser d'avantages bootstrap que je considère comme réellement utile et puissant comme outil.

Je tiens à remercier toute l'équipe de Pop School qui m'ont réellement donné cette chance de pouvoir suivre ce cursus, me donner d'avantage de goûts pour la programmation et qui m'ont réellement ouvert les portes de ce domaine.

Merci d'avoir lu ce projet.

Matthieu NOWAK

