

Germain Denis

Concepteur Développeur d'Applications
SOFIP : Roubaix

Jte dis quoi

L'application des voyageurs du Nord



Source : Pixabay Mrdidg

Sommaire

| | |
|---|----|
| • Présentation du projet | 3 |
| • Partie 1 : Bien s'organiser | 4 |
| ◦ Les différents outils | 4 |
| ◦ Environnement de travail | 5 |
| • Partie 2 : Conception de l'application | 6 |
| ◦ Le modèle Conceptuel de données | 6 |
| ◦ Les UML : Diagramme de classe et d'activité | 7 |
| ◦ La création de la maquette | 9 |
| • Partie 3 : Installation et configuration : | 10 |
| ◦ Présensation de Symfony | 10 |
| ◦ Utilisation du terminal, création du projet | 10 |
| ◦ gitHub | 10 |
| • Partie 4 : Developpement de l'application | 12 |
| ◦ Création de la base de données | 12 |
| ◦ Nos premières entités | 13 |
| ◦ Les fixtures | 14 |
| ◦ Sécurité | 15 |
| ◦ Entity listenner | 16 |
| ◦ Le FireWall | 17 |
| ▪ Login | 17 |
| ▪ Logout | 18 |
| ▪ Formulaire d'inscription | 18 |
| ▪ Edition du profil | 19 |
| ▪ Modification du mot de passe | 21 |
| ◦ Relations entre entités | 22 |
| ◦ Le CRUD | 23 |
| ◦ Noter un lieu | 27 |
| ◦ Ajouter un commentaire | 28 |
| ◦ Restreindre l'accès aux routes | 29 |
| ◦ Formulaire de contact | 30 |
| ◦ Système d'administration avec Easy admin | 31 |
| • Partie 4 : Le Front-end | 32 |
| ◦ Le header / La barre de navigation | 32 |
| ◦ La feuille de style CSS | 33 |
| ◦ Le footer / Le responsive | 34 |
| ◦ Conclusion et remerciements | 36 |
| • Annexes | 37 |

Présentation

Dans le cadre de la formation Concepteur Développeur d'Applications du 14/07/2023 au 20/02/2023 à Roubaix, nous avons dû présenter un projet final afin d'obtenir notre certificat.

L'application est interactive car chaque utilisateur peut ajouter un lieu qu'il a visité. C'est une sorte de réseau social de voyageurs. L'utilisateur peut commenter et s'il le souhaite uploader une photo du lieu visité. L'utilisateur peut ajouter une note à un lieu qui a été proposé par un autre membre. Une note ne peut être attribuée qu'une seule fois par utilisateur et par lieu. Une moyenne des notes sera affichée sur la page qui détaille le lieu en question. Le nombre total des lieux enregistrés sera affiché sur la page d'accueil.

Un utilisateur pourra interagir avec l'application seulement s'il a préalablement créé un compte. En revanche, un utilisateur non connecté pourra envoyé un message à l'administrateur via le formulaire de contact.

Pour la création du compte, l'utilisateur devra renseigner son nom, prénom, email et un mot de passe. Le pseudo sera facultatif. Son mot de passe sera haché. L'utilisateur sera en mesure de modifier ses informations, ainsi que son mot de passe.

Une fois le compte créé, il pourra se connecter et accéder à toutes ces fonctionnalités mais aussi à la liste des lieux qu'il aura lui-même ajoutés. Il pourra enfin, modifier les lieux qu'il a enregistrés.

Un admin a le contrôle total de l'application (il peut y en avoir plusieurs) et va aussi servir de modérateur. Il pourra modérer les nouveaux lieux ajoutés, les commentaires et sera en mesure de bannir des utilisateurs.

L'application va être développée avec le framework Symfony 6. Pour la partie front-end, j'ai utilisé Bootstrap.

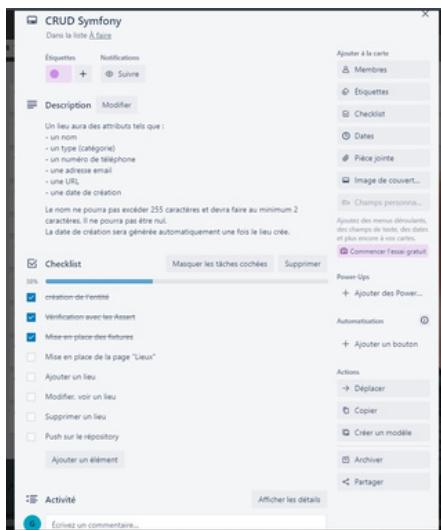
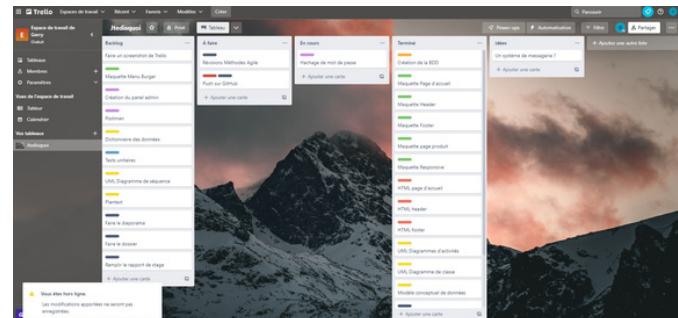
Partie 1 : Bien s'organiser

Pour bien démarrer un projet il faut bien s'organiser. J'ai essayé de respecter plusieurs principes lors du développement du projet à savoir :

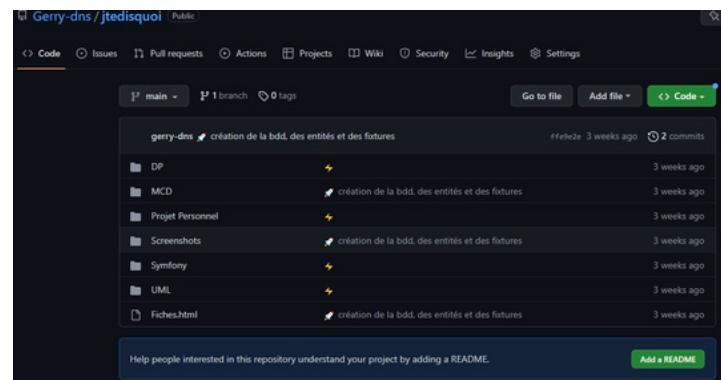
- Etablir un plan clair : définir les objectifs et les délais de mon projet
- Utiliser un outil de planification pour m'aider à suivre le projet
- Définir les priorités : décider ce qui est important et ce qui peut être reporté
- Etre discipliné
- Eviter les distractions
- Etre flexible : j'ajuste mon plan en fonction des imprévus
- Prendre soin de soi et s'accorder suffisamment de pauses

Les différents outils

Trello : C'est un outil de **gestion de projet** en ligne basé sur des tableaux Kanban. Il permet de visualiser les tâches à accomplir, leur progression et les organiser selon leur priorité. Il sera donc très utile pour mon projet.



GitHub : Un service de contrôle de version et de gestion de projet. Il m'a permis de stocker en toute sécurité mon code. Je pourrai partager mon profil à mes futurs employeurs.



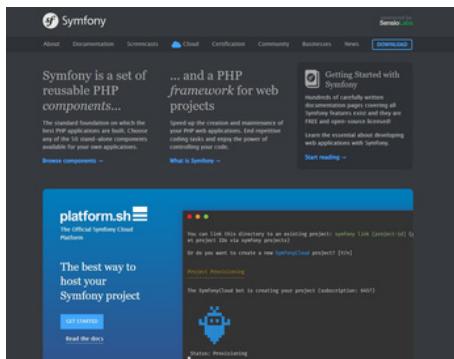
Nous verrons plus tard comment bien l'utiliser

Environnement de travail

Ce qui m'a agréablement surpris durant cette formation, c'est la communauté de développeurs. J'ai beaucoup apprécié cet entraide et ce système d'Open-Source. J'espère pouvoir moi aussi un jour aider de nouveaux développeurs.

Voici quelques ressources et sites d'entraides qui m'ont été indispensables durant le développement de l'application

Symfony: Indispensable pour mon projet



PlantUML : pour la conceptualisation

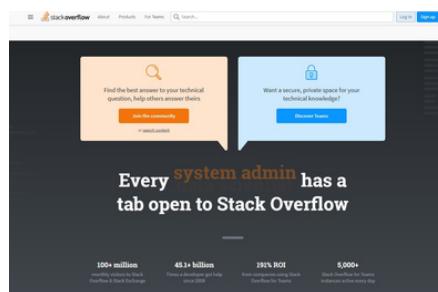
PlantUML est un composant qui permet de dessiner rapidement des :

- diagrammes de séquence
 - diagrammes de cas d'utilisation
 - diagrammes de classes
 - diagrammes d'objet
 - diagrammes d'aktivité (*ici l'ancienne syntaxe*)
 - diagrammes de composant
 - diagrammes de déploiement
 - diagrammes d'état
 - diagrammes de temps

Certains autres diagrammes (hors UML) sont aussi possibles.

- données au format JSON
 - données au format YAML
 - Extended Backus-Naur Form (EBNF) diagram
 - diagrammes de réseaux (nwdiag)
 - maquettes d'interface graphique (salt)
 - diagrammes Archimate
 - diagrammes de langage de description et de spécification (LDS) ou *Specification and Description Language (SDL)*
 - diagrammes d'atlas
 - diagrammes d'État
 - diagrammes d'édit (minidrag)
 - organigrammes ou *Flowchart Breakdown Structure (WBBS)*
 - notation mathématique avec AsciiMath ou LaTeX/Math
 - diagrammes social network (CD-HR)

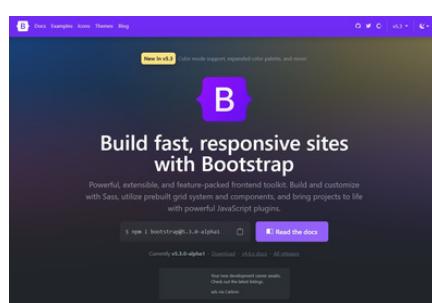
Entraide : Stack over flow qui m'a sauvé plus d'une fois



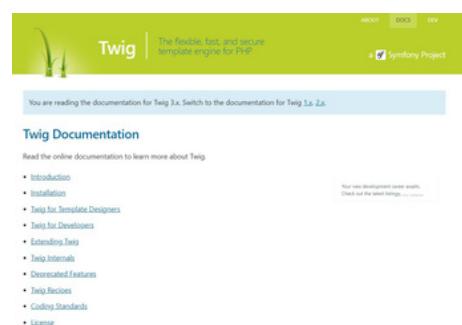
SQL.sh: Pour réviser le requêtage



Bootstrap : la révolution du responsive



Twig: Pour la dynamique



Le modèle conceptuel de données



Un modèle conceptuel de données est une représentation graphique qui permet de comprendre comment les différents éléments sont liés entre eux. Il permet de décrire clairement la structure de données d'une application afin de faciliter la communication entre les membres d'une équipe de développeurs et de garantir que la base de données soit construite de manière cohérente et logique.

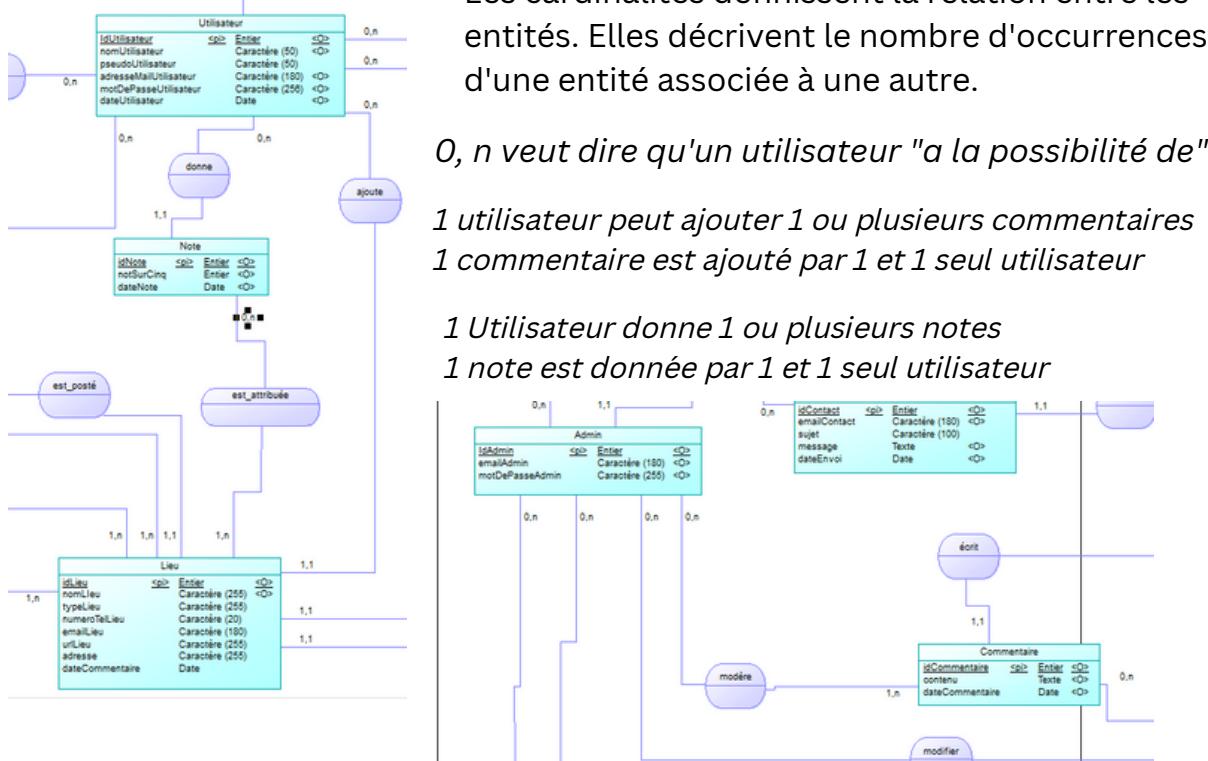
Dans notre cas, nous avons au total six entités :

- Utilisateur
- Administrateur
- Contact
- Note
- Commentaire
- Lieu

Pour ne rien oublier, nous allons créer un dictionnaire de données.

Chacune de ces entités possèdent des attributs. Ce sont les caractéristiques, des propriétés qui définissent les entités. Par exemple l'entité utilisateur aura pour attributs, un nom, un pseudo, un email, un mot de passe.

Voici une miniature du MCD. La totalité ainsi que le dictionnaire de données se trouvent dans les **annexes (1)**.



Les cardinalités définissent la relation entre les entités. Elles décrivent le nombre d'occurrences d'une entité associée à une autre.

0..n veut dire qu'un utilisateur "a la possibilité de"

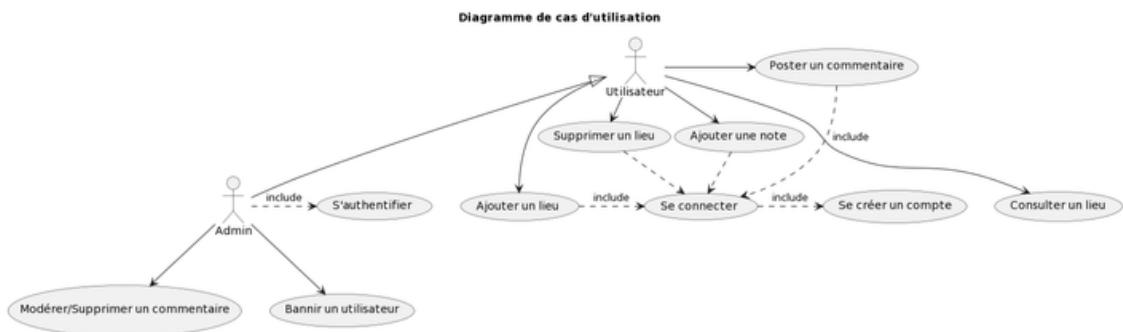
*1 utilisateur peut ajouter 1 ou plusieurs commentaires
1 commentaire est ajouté par 1 et 1 seul utilisateur*

*1 Utilisateur donne 1 ou plusieurs notes
1 note est donnée par 1 et 1 seul utilisateur*

*1 admin modère 1 ou plusieurs commentaires
1 commentaire est modéré par 1 ou plusieurs admins*

Les UML

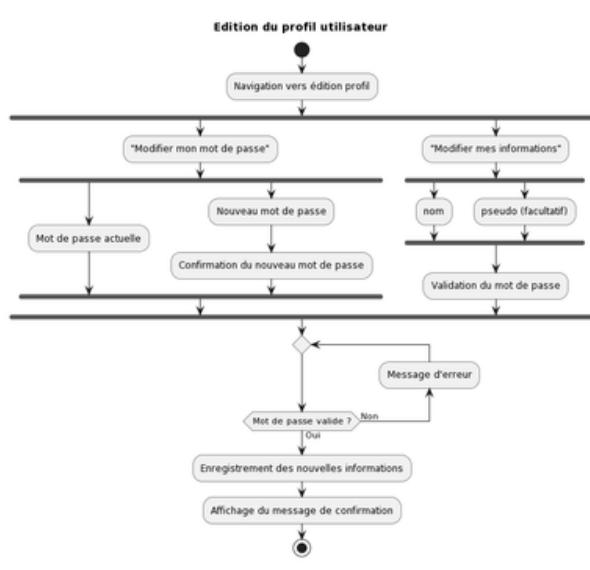
Un **UML** (Unified Modeling Language) est un langage de modélisation graphique utilisé pour la **description** et la **documentation** des systèmes informatiques et logiciels. Il aide à décrire les différents aspects de l'application, tel que le processus, les comportements et relation entre les différents éléments d'un système. En voici des exemples :



Ici un **diagramme de cas d'utilisation** qui représente les **interactions entre les utilisateurs et le système**. Il montre les actions que les utilisateurs peuvent effectuer avec l'application. En l'occurrence ici, un utilisateur lambda peut consulter un lieu. En revanche, pour ajouter un lieu, poster un commentaire ou bien ajouter une note, il va devoir se créer un compte et se connecter.



A noter qu'un Administrateur va **hériter** des droits d'un utilisateur. C'est-à-dire qu'il va pouvoir profiter de toutes les fonctionnalités de l'application.

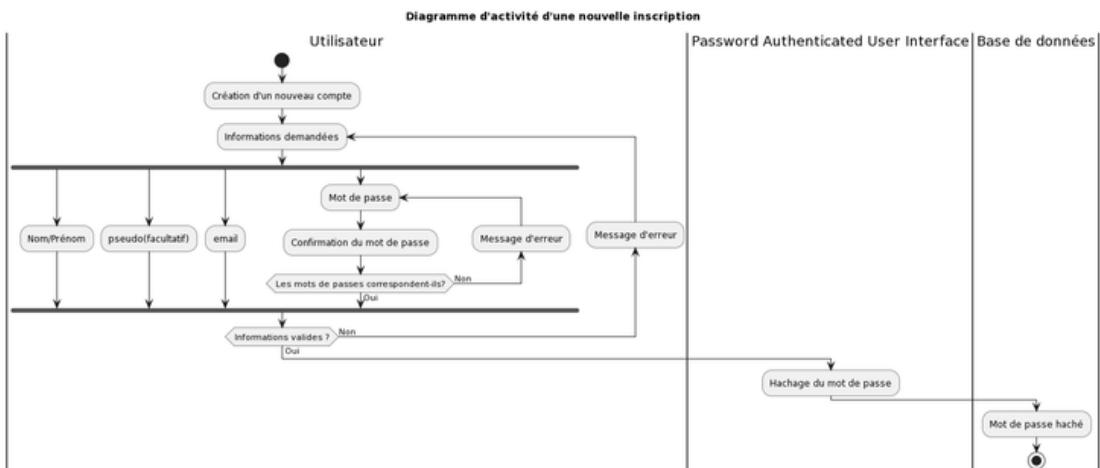


Sur notre gauche se trouve un **diagramme d'activité**. Le diagramme d'activité va démontrer la logique d'un algorithme. Il décompose les étapes d'un processus.

Le diagramme d'activité est caractérisé par des :

- **actions** : symbolisées par des rectangles aux bords arrondis
 - **Nœuds** (de départ, de fin et de décisions),
 - **Divisions** : matérialisées par des symboles d'embranchement.

Voici un autre exemple diagramme d'activité. Les suivants se trouveront dans les **annexes (2)**



Password Authenticated User Interface est une fonctionnalité de sécurité qui va permettre de hacher le mot de passe d'un utilisateur. Dans la base de données il apparaîtra sous cette forme :

```
password
$2y$13$vRwlhAt6ZdnY/TohXUffiewUPaJILqbET6qiHTIQwZN...
```

Pour créer ce diagramme, j'ai utilisé PlantText :

```
@startuml
Title ::= Diagramme d'activité d'une nouvelle inscription
|Utilisateur| --> start
start --> Creation:Création d'un nouveau compte
Creation --> InformationsDemandees:Informations demandées
InformationsDemandees --> NomPrénom:Nom/Prénom
NomPrénom --> Fork1
Fork1 --> PseudoFacultatif:pseudo(facultatif)
PseudoFacultatif --> Fork1
Fork1 --> Email:email
Email --> Fork1
Fork1 --> MotDePasse:Mot de passe
MotDePasse --> Confirmation:Confirmation du mot de passe
Confirmation --> Décision{Les mots de passe correspondent-ils ?}
Décision -- Non --> MessageErreur1:Message d'erreur
Décision -- Oui --> InformationsValides{Informations valides ?}
InformationsValides -- Non --> MessageErreur2:Message d'erreur
InformationsValides -- Oui --> Hachage:Hachage du mot de passe
Hachage --> MotDePasseHaché:Mot de passe haché
MotDePasseHaché --> stop

|Password Authenticated User Interface| --> Hachage
|Base de données| --> MotDePasseHaché
```

PlantText est un outil open-source en ligne qui a pour particularité de créer toutes sortes de diagrammes sans l'aide de la souris.

Il faut en revanche connaître la syntaxe afin de générer correctement le diagramme. Comme nous l'avons vu plus haut je me suis servi de la bibliothèque **PlantUML** afin de connaître les commandes.

La création de la maquette



Une maquette (ou prototype) est utilisée dans le développement web pour présenter une **version préliminaire** de l'interface utilisateur (UI) d'un site web ou d'une application. Elle **permet de visualiser l'apparence et le comportement** de l'application avant de développer le code final.

La maquette est indispensable pour avoir une idée précise de l'apparence et de la disposition de l'interface utilisateur. Pour la réalisation de la maquette j'ai utilisé **FIGMA** qui est un logiciel permettant la collaboration en temps réel entre designers et l'équipe de développeurs.

En voici un exemple. La totalité de la maquette se trouvera dans les **annexes (3)**

LOGO Accueil Tous les lieux Contact Connexion Inscription

Message de bienvenue sur Jte dis quoi !
Description de l'application
Invitation à l'inscription
Call to action : Inscription

CARTE de Lille (API)

Nombre de lieux déjà ajoutés
Bouton pour partager un lieu

Exemples de lieux

Nom Type Description Voir

A PROPOS DE NOUS LES SERVICES PROFILE CONTACT

Copyright 2023 Tous droits réservés par : Jte dis quoi

Sur notre gauche, la page d'accueil d'un utilisateur non connecté.

A droite, la même page sur mobile. Lorsque l'on clique sur le menu burger, un menu déroulant apparaît sur la gauche (ci-dessous)

LOGO

Accueil Tous les lieux Contact Connexion Inscription

Message de bienvenue sur Jte dis quoi !
Description de l'application
Invitation à l'inscription
Call to action : Inscription

CARTE de Lille (API)

Nombre de lieux déjà ajoutés
Bouton pour partager un lieu

Exemples de lieux

Nom Type Description Voir

A PROPOS DE NOUS LES SERVICES PROFILE CONTACT

Copyright 2023 Tous droits réservés par : Jte dis quoi

LOGO Accueil Tous les lieux Mes lieux Ajouter un lieu Contact

Nom Utilisateur ▾

A PROPOS DE NOUS LES SERVICES PROFILE CONTACT

Copyright 2023 Tous droits réservés par : Jte dis quoi

Ci-dessus, une barre de navigation lorsque l'utilisateur est connecté. De nouvelles fonctionnalités apparaissent telles que "**Mes lieux**" ou "**Ajouter un lieu**".

Après avoir cliqué sur le nom de l'utilisateur un menu se déroule

Nom Utilisateur ▾

Modifier mes informations
Modifier mon mot de passe
Déconnexion

Partie 3 : Installation et configuration :

Présentation de Symfony

Symfony est un **framework** de développement web open source qui aide les développeurs à construire des applications web plus rapidement en fournissant une multitude de **composants**. Il est basé sur le langage de programmation **PHP**.

Check requirement

Dans un premier temps, rendons-nous sur la documentation de Symfony.

Dans un terminal nous pouvons lancer cette commande :

symfony check:requirements

```
> PHP is using the following php.ini file:  
C:\xampp\php\php.ini  
  
> Checking Symfony requirements:  
  
.....WW.....W..  
  
[OK]  
Your system is ready to run Symfony projects
```

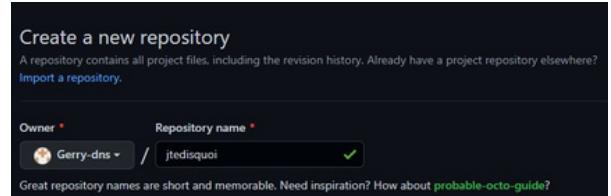
Notre système est prêt pour commencer un nouveau projet Symfony. Nous allons dans notre cas créer une application web traditionnelle.

```
# run this if you are building a traditional web application  
$ symfony new my_project_directory --version="6.2.*" --webapp
```

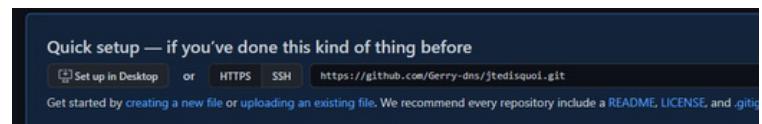
Avant de lancer cette commande, nous allons tout d'abord "pointer" dans le bon emplacement. Dans un terminal **Git Bash**, nous allons nous déplacer dans **l'arborescence** de notre ordinateur. Nous créons ensuite un dossier portant le nom de notre projet : "jtedisquoi"

```
MINGW64:/e/Documents/Programmation/jtedisquoi  
Gerry@Gerry MINGW64 /e/Documents  
$ ls  
Images/ Musiques/ Programmation/ Vidéos/  
Gerry@Gerry MINGW64 /e/Documents  
$ cd Programmation/  
Gerry@Gerry MINGW64 /e/Documents/Programmation  
$ mkdir jtedisquoi  
Gerry@Gerry MINGW64 /e/Documents/Programmation  
$ cd jtedisquoi/  
Gerry@Gerry MINGW64 /e/Documents/Programmation/jtedisquoi  
$ pwd  
/e/Documents/Programmation/jtedisquoi  
Gerry@Gerry MINGW64 ~/Desktop/Germain Projet final/Symfony  
$ symfony new jtedisquoi --version="6.2.*" --webapp
```

Le projet est installé sur notre ordinateur. Nous pouvons dès lors et déjà publier notre dossier local vers le cloud. Dans notre cas nous allons utiliser **GitHub**. Il faudra dans un premier temps créer un nouveau dossier.



Maintenant, récupérons le liens GitHub du "**repository**"



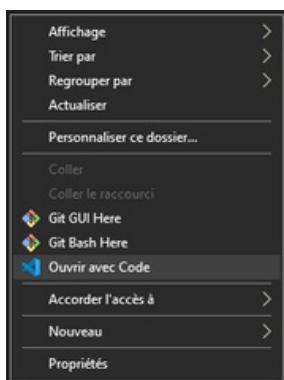
Retournons dans le **terminal**, il faut être sûr d'être dans le bon dossier (dans le doute on peut utiliser la commande "**pwd**". Puis, faisons un "**git init**". Cette commande va créer un nouveau dépôt local. Les commandes suivantes sont :

- **git add "nom_du_fichier"** (elle sélectionne le fichier du dossier)
- **git commit -m "premier push"** Le commit ajoute les modifications au dépôt local
- **git remote add origin https://github.com/Gerry-dns/jtedisquoi.git**
- **git branch "nom_de_la_branche"**
- **git check out "nom_de_la_branche"**
- **git push** (cette commande permet d'envoyer les fichiers sur le dépôt distant)

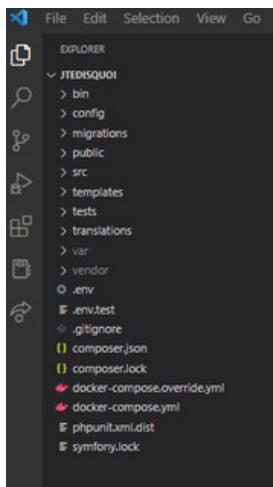


On ne "push" jamais directement sur la "main" ou "master"

Nous pouvons copier ces commandes dans une "**cheatSheet**" et les réutiliser plus tard. Maintenant nous pouvons ouvrir notre IDE (**environnement de développement intégré**) dans notre cas : **Visual studio Code**



Lancer le **serveur** et ouvrir notre page Symfony



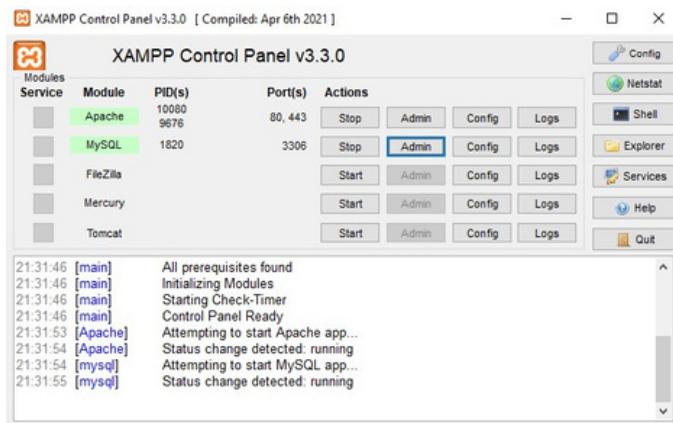
Sur notre gauche, voici à quoi ressemble les dossiers Symfony:

- **bin** : fichiers de commandes, mise à jour
- **config** : la configuration des paquets et des services
- **migrations** : transforme le code PHP en commandes SQL
- **public** : Le point d'entrée de l'application
- **scr** : le cœur du projet (entités, controllers etc.)
- **templates** : les 'vues'
- **test** : pour les tests
- **translation** : pour les traductions
- **var** : dossier de données temporaires
- **vendor** : contient le code source

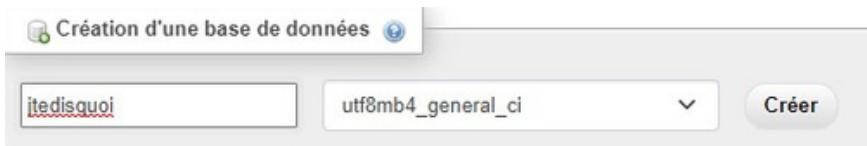
Partie 4 : Développer l'application

Création de la base de données

Dans notre cas nous allons utiliser **Xampp** afin de tester le code grâce notamment aux logiciels **Apache** qui est un serveur web et **MySQL** qui est un système de gestion de bases de données (**SGBD**) et permet donc de stocker et d'accéder à ces données.



En cliquant sur "Admin" une page s'ouvre. C'est **phpMyadmin**. Nous allons pouvoir créer une base de données.



 Si on souhaite utiliser des caractères spéciaux dans dans la base de données, il est préférable de choisir '*utf8mb4_unicode_ci*'

Maintenant retournons dans le projet Symfony. Dans le fichier **.env** nous allons **décommenter** une ligne et rajouter le nom d'utilisateur de connexion, le mot de passe (s'il y en a) et le nom de la base de données à laquelle nous voulons nous connecter.



Il s'agit là de connecter notre projet Symfony à la base de données. 127.0.0.1:3306 correspond à l'**adresse IP** et le **port** sur lequel la base de données est hébergée.

 Il est préférable de créer un **.env.local** et de ne pas le partager sur GitHub puisque ces données sont confidentielles. Il faut aller configurer le fichier **.gitignore**

Notre première entité </>

Dans un nouveau terminal, faire la commande : `php bin/console make:entity`

Un certain nombre de questions vont être posées :

- Le nom de l'**entité**
- le nom de ses **attributs**
- le **type** d'attributs (est-ce une chaîne de caractère ? un nombre entier ?)
- Si ce champ peut être "**null**" en base de données

Nous allons donc nous référer à notre MCD et commencer par créer l'entité "lieu".

 Une adresse email est un "**string**" qui ne devrait pas dépasser 180 caractères.
Un numéro de téléphone est aussi un "string" (il y a parfois des codes indicatifs "+33" par exemple)

Le createdAt ("créé le") est de type **DateTimeImmutable**. Pour qu'elle soit automatiquement générée lors de la création d'un nouveau lieu, nous allons nous rendre dans l'entité et rajouter une fonction :

```
public function __construct()  
{  
    $this->createdAt = new \DateTimeImmutable();  
}
```

Success!

L'entité est créée. Il faut maintenant **transformer** notre code **PHP** en commandes **SQL**. C'est grâce à la commande **make:migration**. Dans le dossier "migration", nous retrouvons bien des commandes SQL :

```
$this->addSql('CREATE TABLE lieu (id INT AUTO_INCREMENT NOT NULL,
```

Avant de continuer nous allons **valider** l'entité : `composer require symfony/validator`

```
use Symfony\Component\Validator\Constraints as Assert;
```

```
##[Assert\Length(min: 2, max: 255)] Il existe un grand nombre de contraintes sur la  
#[Assert\NotBlank()] documentation de Symfony.
```

Une fois importé, on va sur chaque propriété de l'entité apporter des **contraintes**. Dans notre exemple : un nom ne peut pas être vide ou nul :

Pour confirmer l'envoie en base de données tapons la commande :
`php bin/console doctrine:migrations:migrate`

 Pour aller plus vite, on peut aussi écrire :
`php bin/console d:m:m`

Les fixtures

Les fixtures sont utilisées pour remplir la base de données avec de **fausses données** lorsque l'application est en développement. Nous allons devoir ouvrir notre terminal et taper la commande suivante :

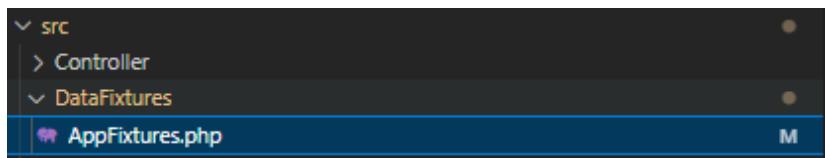
Pour rendre les données encore plus réelles, nous allons utiliser "**FakerPHP**".

Pour cela, il faut taper : `composer require Fakerphp/faker`

- [FakerPHP / Faker](#)
- [Faker](#)
- [Available Formatters](#)
 - [Formatters](#)
 - [Numbers and Strings](#)
 - [Text and Paragraphs](#)
 - [Date and Time](#)
 - [Internet](#)
 - [User Agent](#)
 - [Payment](#)
 - [Color](#)
 - [File](#)
 - [Image](#)
 - [UUID](#)
 - [Barcode](#)
 - [Miscellaneous](#)
 - [Biased](#)
 - [HTML Lorem](#)
 - [Version](#)
 - [Locales](#)
 - [ORM Integration](#)

FakerPHP nous propose un large éventail de fausses données. Nous serons intéressés par la génération de fausses adresse email, de sites web, et de numéros de téléphone.

Nous allons aussi créer un **tableau** dans lequel nous allons mettre les différents types de lieux qu'un utilisateur pourrait ajouter : restaurant, bars, parcs, associations etc. Nous allons utiliser un logiciel tiers en ligne pour générer de faux noms de lieux. Puis grâce à des formules **Javascript** nous allons créer les fixtures. Rendons-nous dans le dossier



```
$typeLieu = ['Bar', 'Restaurant', 'Musée', 'Salle de concert', 'Hôte  
'Jardins', 'Parc', 'Association'];  
$nomLieu = ['Le Palais Clair', 'Le Château de la Plage', 'La Fable de  
'La Légende', 'Le Colibri','Le Mur','Séduction','la Niche','Lueur de  
"Le Nuage d'Orange","La Table Chaude","Le Lieu de Sarriette","Le Nuage  
"Le Balcon de Cuisson","Le GastroGnome", "Le Calme", "Le Saphir", "Le  
"La Capture Rose". "La Cabane Italienne". "La Vallée Ovale". "Le Mor
```

Grâce à une **boucle "for"** nous allons créer 50 nouveaux lieux qui seront de type (bar, restaurant, etc.) générés aléatoirement. Quant aux lieux on leur assignera un à un les noms dans le tableau crée ci-dessus.

```
for ($i=0; $i < 50 ; $i++) {
    $lieu = new Lieu();
    $lieu->setTypeLieu($typeLieu[mt_rand(0, count($typeLieu) -1)]);
    $lieu->setNumeroTelLieu($this->faker->randomNumber(9, true));
    $lieu->setNomLieu($nomLieu[$i]);
    $lieu->setEmailLieu($this->faker->email());
    $lieu->setUrlLieu($this->faker->domainName());
    // a user will be assigned to a random user
    $lieu->setUser($users[mt_rand(0, count($users) -1)]);
    $lieu->setDescription($this->faker->sentence(mt_rand(10, 30)));

    $lieux[] = $lieu;
    $manager->persist($lieu);
}
```

mt_rand sert à assigner
aléatoirement un index.

 Ne pas oublier le
\$manager->flush

Pour **charger les fixtures**
dans la bdd :

```
doctrine:fixtures:load
```

Pour les autres fixtures **cf. annexe (4)**

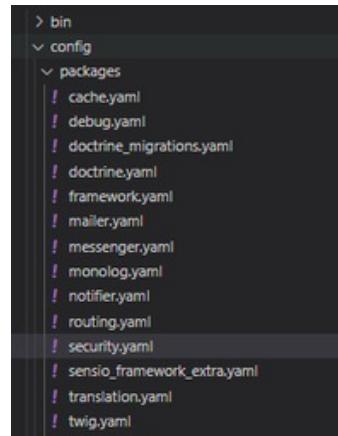
Pour voir le résultat en base de données **cf. annexe (5)**

Sécurité

Normalement, le **SecurityBundle** est déjà installé sinon il faudra faire la commande suivant : `composer require symfony/security-bundle`

Dans le fichier "**Security.yaml**" on retrouve trois éléments principaux :

- Les Providers : le fournisseur d'utilisateur qui va nous permettre de fournir les utilisateurs à partir de la base de données
- Les Firewalls : les pares-feux, le cœur de la sécurisation, car chaque requête va passer par ces firewalls, ce qui va nous permettre de déterminer si un utilisateur doit être identifié pour avoir accès à certaines données)
- L'Access control : c'est le vérificateur qui contrôle les permissions requises par rapport aux URL.



D'après la documentation de Symfony, si nous souhaitons sécuriser et restreindre l'accès d'une partie de notre application nous allons devoir créer un utilisateur via la commande `php bin/console make:user`

Pour valider notre utilisateur, nous allons devoir répondre à un certain nombre de questions : si on veut stocker les données de l'utilisateur dans la base de données. Si on veut utiliser l'email comme propriété pour identifier l'utilisateur et enfin si on veut hacher les mots de passe. La réponse sera oui pour chacune des questions.

Une entité "Utilisateur" est créée et va implémenter un **Password Authentificated User Interface** et un **User Interface** ce qui va permettre à Symfony de gérer correctement la sécurité. De nouvelles méthodes sont utilisées :

- **getUserIdentifier()** qui va donc prendre l'email comme identifiant
- **getRoles()** qui va définir le rôle (USER ou ADMIN)
- **eraseCredentials()** qui va supprimer les données de manière sécurisée

Avant de faire la migration, nous allons compléter notre entité avec les différents attributs que nous avons assignés dans notre MCD, à savoir, un nom, un pseudo et une date de création.

Nous allons aussi ajouter une propriété : `private ?string $plainPassword = null;`

Cette propriété sert à stocker temporairement un mot de passe saisi par l'utilisateur lors de son inscription ou de la modification de son compte. Il sera ensuite haché dans la base de données.

Ne pas oublier **d'importer** les **Getter et Setter**



Ne pas mettre un **ORM\Column** car on ne veut pas que le plainPassword soit dans la base de données

Ensuite, nous allons créer un Provider. Ce qui va permettre de récupérer un utilisateur depuis une zone de stockage.

```
app_user_provider:  
    entity:  
        class: App\Entity\User  
        property: email
```

Normalement, Symfony devrait déjà avoir ajouté ces lignes dans le **security.yaml**.

L'entity listenner ↴

Nous allons exporter la logique du hachage de mot de passe grâce à l'entity listenner.

Ce sont des fichiers qui vont écouter ce qui se passe au niveau des entités et implémenter des fonctionnalités (dans notre cas, le hachage de mot de passe). Nous allons donc créer cette entité dans les **services.yaml** :

```
App\EntityListener:  
    resource: "../src/EntityListener"  
    tags: ["doctrine.orm.entity_listener"]
```

Une fois créée, nous allons l'appeler dans l'entité User :

```
#[ORM\Entity(repositoryClass: UserRepository::class)]  
#[ORM\EntityListeners(['App\EntityListener\UserListener'])]
```

Enfin, nous allons créer un fichier **entityListeners.php** dans lequel nous allons implémenter la fonction d'hachage de mot de passe. **cf. annexes (6)**

```
password  
$2y$13$vRwlhAt6ZdnY/TohXUffiewUPaJLqbET6qiHTIQwZN...
```

Le Firewall

Le Firewall va définir les parties de l'application qui sont sécurisées et comment les utilisateurs vont avoir la possibilité de se connecter.

La création du formulaire de connexion

Pour ce faire, on va créer un **Controller**, on va ensuite compléter le **security.yaml** avec le paramètre **form_login**. Enfin, on va créer le formulaire de connexion.

Voici la commande pour créer un controller :

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

```
class SecurityController extends AbstractController
{
    #[Route('/connexion', name: 'security.login', methods: ['GET', 'POST'])]
    public function login(): Response
    {
        return $this->render('pages/security/login.html.twig', [
            'controller_name' => 'SecurityController',
        ]);
    }
}
```

Dans le fichier **security.yaml**, nous allons devoir rajouter le **form_login** avec les chemins.

```
main:
    lazy: true
    provider: app_user_provider
    form_login:
        login_path: security.login
        check_path: security.login
```

 A noter que le nom du **login** et **check path** doit correspondre avec le nom du **Controller**.

Enfin, nous allons créer le formulaire de connexion. Vous retrouverez l'imprime écran dans les **annexes (7)**. En revanche, pour la partie Back, deux lignes de code très importantes à retenir :

```
<form action="{{ path('security.login') }}" method="post"></form>
<input type="email" class="form-control" id="userEmail" name="_userEmail">
```

La déconnexion

Pour créer un "log out", il faut retourner dans le SecurityController et ajouter :

```
#[Route('/deconnexion', name: 'security.logout')]
public function logout()
{
    // Nothing to do here...
}
```

Puis dans le security.yaml

```
logout:
    path: security.logout
```

Le formulaire d'inscription

On va d'abord créer un formulaire : **php bin/console make:form**

```
The name of the form class (e.g. TinyKangarooType):
> RegistrationForm

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> User

created: src/Form/RegistrationFormType.php

Success!
```

```
->add('email', EmailType::class, [
    'attr' => [
        'class' => 'form-control',
        'minlength' => '2',
        'maxlength' => '180',
    ],
    'label' => 'Adresse email',
    'label_attr' => [
        'class' => 'form-label'
    ],
    'constraints' => [
        new Assert\NotBlank(),
        new Assert\Email(),
        new Assert\Length(['min' => 2, 'max' => 180])
    ]
])
```

Pour le mot de passe, il devra être répété plusieurs fois, notamment grâce à "**RepeatedType**". Un message d'erreur sera affiché si le client ne rentre pas le bon mot de passe.

Dans le **RegistrationFormType**, nous allons créer notre formulaire en sécurisant les inputs. Par exemple en mettant **EmailType**, en définissant le nombre minimum et maximum de caractères et enfin en ajoutant des **"Assert"** tels que "**NotBlank**", "**Email**" et "**Length**"

```
->add('plainPassword', RepeatedType::class, [
    'type' => PasswordType::class,
    'first_options' => [
        'attr' => [
            'class' => 'form-control'
        ],
        'label' => 'Mot de passe',
        'label_attr' => [
            'class' => 'form-label mt-4'
        ]
    ],
    'second_options' => [
        'attr' => [
            'class' => 'form-control'
        ],
        'label' => 'Confirmation du mot de passe',
        'label_attr' => [
            'class' => 'form-label mt-4'
        ]
    ],
    'invalid_message' => 'Les mots de passe ne correspondent pas'
])
```

Edition du profil

Pour éditer le profil d'un utilisateur, il va falloir créer un formulaire qu'on nommera **userType** en se basant sur l'entité "User". L'utilisateur pourra changer son pseudo (ou en ajouter un) et son nom.

```
->add('pseudo', TextType::class, [
    'attr' => [
        'class' => 'form-control',
        'minLength' => '2',
        'maxLength' => '50',
    ],
    'required' => false,
    'label' => 'Pseudo Facultatif',
    'label_attr' => [
        'class' => 'form-label mt-4'
    ],
    'constraints' => [
        new Assert\Length(['min' => 2, 'max'=> 50])
    ]
])
```



'**required**' => **false** nous permet de ne pas obliger l'utilisateur à mettre un pseudonyme

Ensuite on crée un controller : **php bin/console make:controller User**

Dans le Controller, nous allons renommer la Route et mettre les méthodes GET et POST. `#[Route('/utilisateur/edit/{id}', name: 'user.edit', methods:['GET', 'POST'])]`
GET va récupérer les informations dans l'URL tandis que dans POST, elles sont écrites directement dans la requête HTTP.

Dans la fonction "**edit**", nous allons passer en argument l'entité **User**, la **Request** et l'**Entity Manager Interface**.

```
public function edit(User $user, Request $request, EntityManagerInterface $manager,
```

```
if(!$this->getUser()) {
    // if not logged, the user will be redirected to login
    return $this->redirectToRoute('security.login')
```

Si l'utilisateur courant n'est pas connecté on le redirige vers la page de login

Si l'utilisateur courant n'a pas le même ID du profil. En d'autres termes, on vérifie si l'utilisateur n'essaie pas de modifier un autre profil.

```
if($this->getUser() !== $user) {
    return $this->redirectToRoute('lieux.index');
```

Ensuite, nous créons le formulaire en vérifiant s'il est soumis et valide. Puis on le return dans la page **Twig** qu'on nommera "**edit.html.twig**"

```
$form = $this->createForm(UserType::class, $user);
```

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

On peut aussi vérifier le mot de passe avant que le manager n'apporte les modifications à la base de données avec la méthode : **isPasswordValid** avec pour argument la variable **\$user**, les données du formulaire et le **plainPassword**

```
// If the password = password of current user
if($hasher->isPasswordValid($user, $form->getData()->getPlainPassword())){
    $user = $form->getData();
    $manager->persist($user);
    $manager->flush();
    //adding a succes message
    $this->addFlash(
        'success',
        'Les informations de votre compte ont bien été modifiées'
    );
} else {
    $this->addFlash(
        'warning',
        'Le mot de passe est incorrect.'
    );
}
```

Dans le template Twig, on affiche les messages d'alerte ainsi que le formulaire

```
<div class="container mt-5">
    <h1>Modification de mon profil</h1>
    {% for message in app.flashes('warning') %}
        <div class="alert alert-warning mt-4">
            {{message}}
        </div>
    {% endfor %}
    {% for message in app.flashes('success') %}
        <div class="alert alert-success mt-5">
            {{ message }}
        </div>
    {% endfor %}

    {{ form(form) }}
</div>
```

Modification du mot de passe

Pour que l'utilisateur puisse modifier son mot de passe, nous allons créer un nouveau formulaire : **UserPasswordType**.

Nous allons utiliser la classe **RepeatedType** et **PasswordType** avec la "first_options" et "second_options" :

```
->add('plainPassword', RepeatedType::class, [
    'type' => PasswordType::class,
    'first_options' => [
        'attr' => [
            'class' => 'form-control'
        ],
        'label' => 'Mot de passe',
        'label_attr' => [
            'class' => 'form-label mt-4'
        ]
    ],
    'second_options' => [
        'attr' => [
            'class' => 'form-control'
        ],
        'label' => 'Confirmation du mot de passe',
        'label_attr' => [
            'class' => 'form-label mt-4'
        ]
    ],
    'invalid_message' => 'Les mots de passe ne correspondent pas.'
])
```

Et le nouveau mot de passe :

```
->add('newPassword', PasswordType::class, [
    'attr' => [
        'class' => 'form-control'
    ],
    'label' => 'Nouveau mot de passe',
    'label_attr' => [
        'class' => 'form-label mt-4'
    ],
    'constraints' => [new Assert\NotBlank()]
])
```



Ne pas oublier
d'importer les classes

On retourne dans le **UserController**, on crée une fonction **editPassword** qui va retourner une Response la page twig correspondante. Les arguments de la fonction seront : l'entité **User**, la **Request**, le **ManagerInterface**, le **UserPasswordHasherInterface**.

On va décorer la fonction d'une Route **/utilisateur/edition-mot-de-pass/{id}**. Les méthodes seront **GET** et **POST**.

```
if ($form->isSubmitted() && $form->isValid()) {
    if ($hasher->isPasswordValid($user, $form->getData()['plainPassword'])) {
        $user->setPassword(
            $hasher->hashPassword(
                $user,
                $form->getData()['newPassword']
            )
        );
    }
}
```

Les relations entre entités

Après avoir créé l'entité "User" nous allons donc créer sa relation avec l'entité "lieu". Comme nous l'avons spécifié dans le MCD un utilisateur peut ajouter plusieurs lieux. Un lieu est ajouté par un et un seul utilisateur.

```
Class name of the entity to create or update (e.g. GentleKangaroo):
> User

Your entity already exists! So let's add some new fields!
> lieu

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

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

What class should this entity be related to?
> lieu

What type of relationship is this?

Type Description
-----
ManyToOne Each User relates to (has) one lieu.
          Each lieu can relate to (can have) many User objects.

OneToMany Each User can relate to (can have) many lieu objects.
           Each lieu relates to (has) one User.

ManyToMany Each User can relate to (can have) many lieu objects.
           Each lieu can also relate to (can also have) many User objects.

OneToOne Each User relates to (has) exactly one lieu.
          Each lieu also relates to (has) exactly one User.

-----
```

Nous allons donc créer un nouvel attribut à l'entité "User" qui s'appellera "lieux"
Le type sera cette fois-ci "**relation**"
On nous demande alors à quelle classe cette entité est-elle reliée :
Nous sélectionnerons donc "lieu"
Dans notre cas, nous allons choisir la relation **OneToMany**

Est-ce que "lieu.user" peut être "**null**" en base de données : "**no**"
Est-ce que l'on veut que l'entité "lieu" disparaisse si l'utilisateur supprime son compte ? Dans notre cas : "**yes**"

La relation est créée. Nous pouvons refaire une migration pour valider ce changement.

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

A new property will also be added to the lieu class so that you can access and set the related User object from it.

New field name inside lieu [user]:
>

Is the lieu.user property allowed to be null (nullable)? (yes/no) [yes]:
> no

Do you want to activate orphanRemoval on your relationship?
A lieu is "orphaned" when it is removed from its related User.
e.g. $user->removeLieu($lieu)

NOTE: If a lieu may "change" from one User to another, answer "no".
Do you want to automatically delete orphaned App\entity\lieu objects (orphanRemoval)? (yes/no) [no]:
> yes

updated: src/Entity/User.php
updated: src/Entity/Lieu.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!
```

Dans la base de données nous aurons comme résultat :

| id | user_id | nom_lieu | type_lieu | numero_tel_lieu | email_lieu | url_lieu | created_at (DC2Type:datetime_immutable) |
|----|---------|----------|-----------|-----------------|------------|----------|--|
|----|---------|----------|-----------|-----------------|------------|----------|--|

Le CRUD

Le CRUD correspond aux quatre opérations les plus courantes que l'on retrouve sur les applications. Le C pour **Create**, le R pour **Read**, le U pour **Update** et le D pour **Delete**.

CREATE

Tout d'abord, nous allons devoir créer un **Controller**. Dans notre cas ce sera le **MesLieuxController**. Rappelons-nous qu'un utilisateur ne peut ajouter un lieu seulement s'il est connecté (nous verrons plus tard comment se connecter) :

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

Nous reviendrons plus tard dans le Controller. Nous allons dans un premier temps créer un formulaire grâce à la commande **php bin/console make:form**. Ce formulaire sera lié à l'entité "lieu". Nous le retrouverons dans le dossier **scr/form**. Voici le résultat :

```
ass LieuFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('nomLieu')
            ->add('typeLieu')
            ->add('numeroTelLieu')
            ->add('emailLieu')
            ->add('urlLieu')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Lieu::class,
        ]);
    }
}
```

D'après la documentation de Symfony, il va falloir ajouter les 'types'.

- "nomLieu" sera un "**TextType**"
- "typeLieu" sera un "**ChoiceType**"
- "numeroTelLieu" un "**TelType**"
- "emailLieu" un "**EmailType**"
- "urlLieu" un "**UrlType**"
- "imageFile" un "**VichImageType**" - annexe (8)
- "submit" un "**SubmitType**"

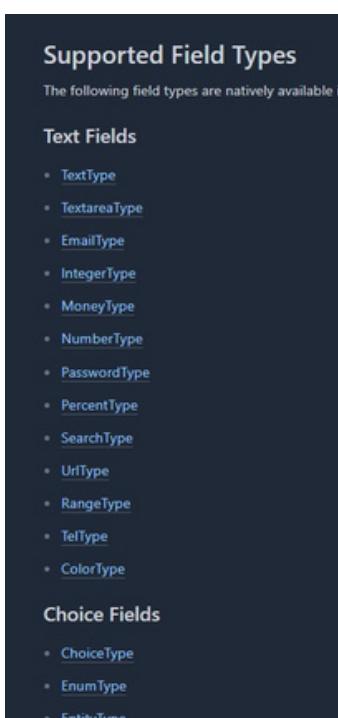
Nous assignons des attributs ("**attr**") avec un nombre de caractères minimum à 2 jusqu'à 255. Nous lui donnons un 'label' ainsi que des class **Bootstrap mt-4**

 Ne pas oublier d'**importer les classes**

C'est ici que nous allons aussi ajouter des contraintes de validation :

```
'constraints' => [
    new Assert\Length(['min' => 2, 'max' => 255]),
]
```

L'ordre dans lequel nous mettons les champs sera l'ordre d'apparition dans la "**vue**".



Pour le **Controller**, nous allons devoir créer une **Route** c'est à dire une adresse. Dans ce cas ce sera **/lieu/nouveau**. Le nom de la Route sera "**lieu.new**", ce nom nous servira pour naviguer dans l'application. La **méthode** HTTP sera **GET** et **POST**

GET est utilisé pour **récupérer les données** du serveur. Ces données vont s'inclure dans l'URL. '**POST**' sert à **la transmission des informations et des données** de l'utilisateur.

Ensuite, il va falloir créer une "**public function**" dans laquelle nous allons faire une injection de dépendance. Nous allons appeler les classes "**Request**" qui va contenir les informations des données de la requête et "**EntityManagerInterface**", qui va servir à gérer les objets, dans ce cas précis pour ajouter un lieu.

Cette fonction va créer un nouveau "lieu" en tant qu'entité, elle sera stockée dans une **variable** que l'on va nommer **\$lieu**. Ensuite, nous allons créer une variable **\$form** qui va faire appel au formulaire que nous avons créé préalablement. Cette variable va être soumise à une requête via la méthode "**handleRequest**".

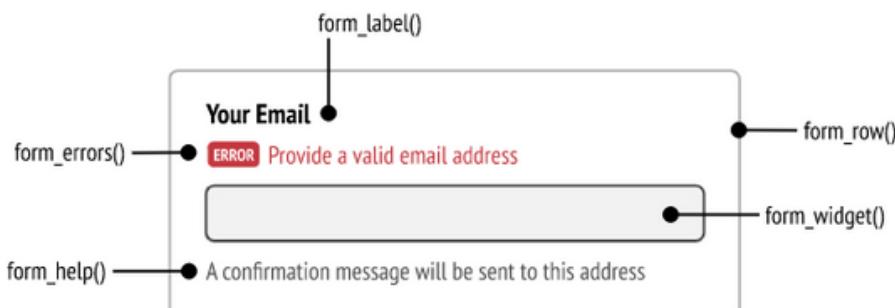
Puis, nous allons créer une **condition** : si le formulaire est soumis et qu'il est valide alors grâce à la méthode "**getData()**" les données du formulaire seront récupérées.

On peut aussi faire en sorte que l'utilisateur qui ajoute ce lieu soit attribué à l'objet. La méthode "**persist()**" va alors être enregistrée puis envoyé dans la base de données avec la méthode "**flush()**".

Pour une meilleure **expérience UI**, nous allons ajouter un message flash grâce à la méthode "**addFlash()**" dans laquelle nous allons donner un nom, en l'occurrence "**success**" pour rester cohérent ainsi qu'un message : "*Votre lieu a bien été enregistré. Merci pour votre aide !*"

Pour finir, nous allons dans le **return** qui va retourner un résultat d'une requête HTTP, avec la méthode "**render()**" qui pointe vers le fichier **Twig** associé, ajouter un argument, qui est un tableau associatif qui va créer la "vue" dans Twig de l'objet "form". (cf. annexe)

Il ne nous reste plus qu'à remplir le template Twig. Regardons la documentation :



L'entièreté du code se trouvera dans les **annexes (9)**

Dans le "`{%block body%}`", nous allons ajouter un "`{{ form_start(form) }}`" dans lequel nous mettrons un "`{{ form_label(form.nomLieu) }}`", un "`{{ form_widget(form.nomLieu) }}`" ainsi qu'un `{{ form_errors(form.nomLieu) }}`. Nous allons répéter cette structure pour chacun des attributs de l'entité "lieu" :

```
<div class="form-group">
    {{ form_label(form.typeLieu) }}
    {{ form_widget(form.typeLieu) }}
    <div class="form-error">
        {{ form_errors(form.typeLieu) }}
    </div>
</div>
```

Les `{{form}}` sont entourés par des classes de Bootstrap "**form-group**" afin de styliser les groupes d'éléments. Nous finirons par `{{form_row(form.submit)}}` et `{{form_start(end)}}`



Un utilisateur non connecté pourra accéder à tous les lieux proposés.

Pour ce faire nous allons **créer un Controller "lieux"** que nous allons décorer d'une **Route /lieux**. La méthode cette fois-ci sera **GET**. Nous allons ensuite créer une fonction qui importera le **LieuRepository** (le dossier de la base de données où sont enregistrés tous les lieux) On utilisera la méthode **findAll()** pour aller chercher tous les lieux.

```
#[Route('/lieux', name: 'lieux.index', methods:['GET'])]
public function displayAllLieux(LieuRepository $repository, PaginatorInterface $paginator)
{
    $lieux = $paginator->paginate(
        // display 'lieux' that are assigned to the current user
        $repository->findAll(),
        $request->query->getInt('page', 1), /*page number*/
        4 /*limit per page*/
    );
    return $this->render('pages/lieu/lieux.html.twig', [
        'controller_name' => 'LieuxController',
        'lieux' => $lieux,
    ]);
}
```

Installation de **PaginatorInterface** :



```
composer require knplabs/knp-paginator-bundle
```

Pour rendre le contenu plus agréable à voir nous allons utiliser l'argument **PaginatorInterface**, ce qui va nous permettre de paginer les données. Nous afficherons quatre lieux par page.

Au niveau de Twig, nous allons créer une **boucle "for"** qui va aller chercher dans la base de données tous les lieux. Puis on va les afficher dans une "**div**" avec un paragraphe en appelant la **variable** avec le **symbole twig** : `{}{}`. A l'intérieur de ces symboles nous allons entrer le nom de la table et le nom de la colonne que l'on veut afficher. Dans ce cas ce sera : `{}{lieu.nom}`, le type, etc.

```

<div class="container">
    <div class="row">
        <div class="col-12 d-flex mt-5">
            {% for lieu in lieux %}
                <div class="card m-3" style="width: 18rem;">
                    <div class="card-body d-flex justify-content-between flex-column">
                        <h2 class="card-title">{{lieu.nomLieu}}</h2>
                        <h3 class="card-subtitle mb-2 text-muted">{{lieu.typeLieu}}</h3>
                        <p class="card-text">{{lieu.description}}</p>
                        <a class="btn btn-info" href="{{ path('lieux.show', { id : lieu.id }) }}>Voir</a>
                    </div>
                </div>
            {% endfor %}
        </div>
    </div>
</div>

```

Il faudra aussi afficher la barre qui permet de passer d'une page à l'autre :

```

        </div>
        {# display navigation #}
        <div class="navigation d-flex justify-content-end mt-3">
            {{ knp_pagination_render(lieux)}}
        </div>

```

UPDATE

La subtilité du Update est de rajouter un **{id}** à la fin de la route.

```
#[Route('/lieu/edit/{id}', 'lieu.edit', methods: ['GET', 'POST'])]
```

On va rappeler le formulaire que nous avions créé pour le CREATE et procéder de la même manière. Cependant le addFlash va cette fois-ci afficher "Votre lieu a bien été modifié". Voici comment afficher le message dans le template Twig :

```

    {% for message in app.flashes('success') %}
        <div class="alert alert-success mt-5">
            {{ message }}
        </div>
    {% endfor %}

```

DELETE

Nous utiliserons aussi un **{id}** dans la route. Avant de "**flush()**" on appellera le manager pour **remove()** la variable \$lieu



Une fois la requête validée, on peut utiliser la méthode "**redirectToRoute**" pour rediriger l'utilisateur sur une autre page.

```
return $this->redirectToRoute('mesLieux.index');
```

Noter un lieu

Un utilisateur peut attribuer une note à un ou plusieurs lieux. Un lieu dispose d'une note par utilisateur. Nous avons donc la relation 1,n - 1,1. La note attribuée devient donc une entité. Après avoir créé l'entité, nous allons rajouter les contraintes. Un utilisateur pourra ajouter une note de un à cinq. Cette note est donc forcément positive :

```
#ORM\Column(type: 'integer')
#[Assert\Positive()]
#[Assert\LessThan(6)]
private ?int $mark = null;
```

Rendons-nous maintenant dans l'entité "Lieu". Il va falloir rajouter une fonction qui va calculer la moyenne des notes attribués par les utilisateurs.

```
public function __construct()
{
    $this->marks = new ArrayCollection();
```

La classe ArrayCollection est une classe fournie Symfony qui permet de gérer des collections d'objets

Puis on va créer une fonction "getAverage". cf. annexe (10)

D'abord, la variable \$marks est initialisée avec les propriétés marks de l'objet en cours.

Ensuite, un contrôle est effectué pour vérifier si la note (\$marks) est vide en utilisant la méthode "toArray()" et la comparaison avec un tableau vide. Si c'est le cas, la moyenne est définie sur "null" et retournée immédiatement.

Si la collection de notes n'est pas vide, une variable \$total est définie pour tenir compte de la somme des notes. Elle est ensuite incrémentée pour chaque note dans la collection en utilisant une boucle foreach et la méthode getMark sur chaque objet de note.

Enfin, la moyenne est calculée en divisant le total des notes par le nombre de notes (obtenu avec count(\$marks)) et est assignée à la propriété average de l'objet.

La moyenne finale est retournée en fin de fonction.

Pour qu'un utilisateur note qu'une seule fois un même lieu nous allons ajouter dans l'entité "mark" ceci : `use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;`

```
#[UniqueEntity(
    fields: ['user', 'lieu'],
    errorPath: 'user',
    message: 'Cet utilisateur a déjà noté ce lieu'
)]
```

Il va ensuite falloir créer un formulaire qui sera lié à l'entité "mark" et ajouter un **ChoiceType** de un à cinq.

Rendons-nous ensuite dans le LieuxController

Cette note va apparaître sur la "show.html.twig" avec la Route : /lieux/{id}

Nous allons créer deux conditions. Comme précédemment, nous vérifions si le formulaire est soumis et valide. On va aussi automatiquement assigné l'utilisateur connecté à la note. On vérifie ensuite si l'utilisateur a déjà noté le lieu. Si c'est le cas, un message addflash apparaîtra pour informer l'utilisateur. Sinon le \$manager va valider la note dans la base de données.

Pour finir, dans la "vue", nous allons ajouter la moyenne arrondie à deux virgules grâce à un filtre Twig

```
<p>Note : {{(lieu.average|number_format(2, '.', ','))}}/5</p>
```

Ajouter un commentaire

Pour l'affichage des commentaires, c'est un peu le même principe. Cependant un utilisateur pourra ajouter plusieurs commentaires. Nous allons donc créer une entité "**Commentaire**" qui aura lui aussi une relation **ManyToOne** avec l'entité "lieu"

On va créer un formulaire qui sera lié à l'entité "lieu" et qui aura un **TextareaType**. Il aura aussi une contrainte, il sera "**NotBlank**".

De retour dans le "**LieuxController**" au niveau de la Route /lieux{id} qui aura pour méthode **GET** et **POST**. Nous allons appeler faire une **injection de dépendance** du **CommentaireRepository** dans la fonction. Puis nous allons créer une nouvelle instance de classe "Commentaire" avec pour argument "lieu". Ensuite, on crée un formulaire en utilisant ce nouvel objet. Ce formulaire est ensuite géré en fonction de la requête HTTP.

```
EntityManagerInterface $manager, CommentaireRepository $commentaireRepository) : Response
{
    $comment = new Commentaire($lieu);
    $commentForm = $this->createForm(CommentType::class, $comment);
    $commentForm->handleRequest($request);
```

Le commentaire doit être valide et soumis pour que le \$manager l'ajoute dans la base de données. On assigne automatiquement l'utilisateur connecté au commentaire : `$comment->setUser($this->getUser())
 ->setLieux($lieu);`

Sur la page "show.html.twig" on va afficher le nombre total de commentaires grâce au filtre Twig : `{{lieu.commentaires|length}}`

Enfin, on va afficher les commentaires grâce à la "**boucle for in**". cf. annexe (11)

Restreindre l'accès aux routes

Afin d'être sûr qu'un utilisateur non connecté ne puisse accéder aux fonctionnalités destinées aux utilisateurs connectés, nous allons restreindre l'accès aux Route. Nous pouvons nous rendre dans la documentation Symfony au niveau de la sécurité. Nous allons utiliser **isGranted** qui permet de gérer les accès par les **ROLES**

```
use Symfony\Component\Security\Http\Attribute\IsGranted;  
  
#[Route('/mesLieux', name: 'mesLieux.index', methods:['GET'])]  
#[IsGranted('ROLE_USER')]
```

Pour l'Update ou Delete de CRUD, c'est un peu plus compliqué, car non seulement on veut que ce soit un utilisateur connecté qui puisse modifier ou supprimer un lieu, mais aussi qu'il puisse modifier uniquement les lieux qui lui appartiennent. On va dans ce cas utiliser l'annotation **Security**

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;
```

On vérifie que l'utilisateur courant est égal à l'utilisateur responsable du lieu en question `#[Security("is_granted('ROLE_USER') and user === lieu.getUser()")]`

On va vérifier en tapant dans la barre d'adresse **lieu/edition/{id}** qui correspond à un lieu d'un autre utilisateur.

Nous avons bien la réponse **HTTP 403 Forbidden**



Dans **les annexes (19)**, nous verrons les erreurs les plus courantes.

Nous allons procéder à la même chose dans le UserController au niveau de l'édition du profile et du mot de passe.

Le formulaire de contact

Nous allons créer une nouvel entité : **php bin/console make:entity Contact**

Le nom est de type "string", il ne peut pas excéder 50 caractères. Il peut en revanche être "null" dans la base de données.

A l'inverse, on veut absolument avoir un **email** enregistré dans la base de données.

```
Add another property? Enter the property name (or press <return> to stop adding fields):
> email

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

Field length [255]:
> 180

Can this field be null in the database (nullable) (yes/no) [no]:
> no
```

On va avoir ensuite besoin d'un attribut "**sujet**" qui fera 100 caractères et "null" en base de données et un **message** qui sera cette fois-ci un "**text**" qui ne pourra pas être "null" en base de données. Enfin on aura une **date d'envoi** qui sera un `datetime_immutable`.

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

On va ensuite valider les entités. Pour la date d'envoi, on peut utiliser la méthode `__construct()`

On fait ensuite les migrations pour l'insérer dans la base de données.

On crée un Controller que l'on va appeler : **ContactController**

```
#[Route('/contact', name: 'contact.index')]
public function index(EntityManagerInterface $manager, Request $request, MailerInterface $mailer): Response
{
    $contact = new Contact();

    // if there is a current user
    if($this->getUser()) {
        // FullName and Email will be already filled-in
        $contact->setFullName($this->getUser()->getFullName())
            ->setEmail($this->getUser()->getEmail());
    }
}
```

Si la personne qui écrit le message est un utilisateur courant, on va récupérer son nom et son email

```
$form = $this->createForm(ContactType::class, $contact);

$form->handleRequest($request);
if($form->isSubmitted() && $form->isValid()) {
    $contact = $form->getData();

    $manager->persist($contact);
    $manager->flush();
```

Ci-contre la soumission et validation du formulaire. Juste après la méthode `flush()`, nous y mettrons le code pour utiliser Mailtrap - **annexe (14)**

On crée un **formulaire** `ContactType` qui se base sur l'entité `Contact` :

- le nom sera un `TextType` ('`required`' => false (pas obligatoire))
- l'email sera un `EmailType`
- le sujet un `TextType`
- le message un `TextareaType`
- le recaptcha un `Recaptcha3Type` - **annexe (15)**

Système d'administration avec Easy admin



On va utiliser le **bundle** EasyAdmin :

```
composer require easycorp/easyadmin-bundle
```

Dashboard `php bin/console make:admin:dashboard`

```
# [Route('/admin', name: 'admin')]
#[IsGranted('ROLE_ADMIN')]
public function index(): Response
{
    return $this->render('admin/dashboard.html.twig');
```

On va restreindre la Route par un **isGranted** en mettant le **ROLE_ADMIN**

Configurer le dashboard la totalité dans les annexes (16)

On peut ajouter du HTML brute dans le template Twig :

```
{% block content %}
<h1>Administration Jte dis quoi</h1>
{% endblock %}
```

Pour rajouter des éléments dans le menu du dashboard, on va devoir créer des **CRUD Controllers**

```
php bin/console make:admin:crud
```

```
Which Doctrine entity are you going to manage with this CRUD controller?:
[0] App\Entity\Commentaire
[1] App\Entity>Contact
[2] App\Entity\Lieu
[3] App\Entity\Mark
[4] App\Entity\User
[5] VichUploaderBundle\Entity\File
> []
```

Nous allons commencer par gérer l'entité User.

On peut changer le nom de l'entité.
->setEntityLabelInSingular : sert pour la page détail de l'utilisateur
On peut aussi changer le nom de la page et afficher un certain nombre d'utilisateur par page

```
// this function configure the crud (design, titles, entities...)
public function configureCrud(Crud $crud): Crud
{
    return $crud
        // changing the name of the Entity (User) in French
        ->setEntityLabelInPlural('Utilisateurs')
        // when clicking on one single User, the name will be "Utilisateur"
        ->setEntityLabelInSingular('Utilisateur')

        ->setPageTitle("index", "Jte dis quoi - Administration des utilisateurs")
        ->setPaginatorPageSize(10);
}
```

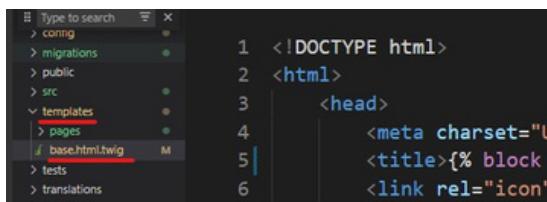
```
// display the different fields
public function configureFields(string $pageName): iterable
{
    return [
        // on the dashboard, there will be displayed :
        // id is the name of the property of the Entity
        IdField::new('id')
            // id won't be display when clicked on editing option
            ->hideOnForm(),
        TextField::new('fullName'),
        TextField::new('pseudo'),
        TextField::new('email')
            // email will be displayed, but cannot be changed
            ->setFormTypeOption('disabled', 'disabled'),
        ArrayField::new('roles'),
        DateTimeField::new('createdAt')
            ->hideOnForm()
    ];
}
```

Pour afficher certains éléments dans la page détail par exemple le nom on va utiliser les **TextField**. Si on veut en revanche cacher l'id, on peut utiliser le ->**hideOnForm()**.
On peut aussi faire en sorte de ne pas pouvoir changer l'email avec
->setFormTypeOption('disabled', 'disabled')

Partie 4 : Le Front-end

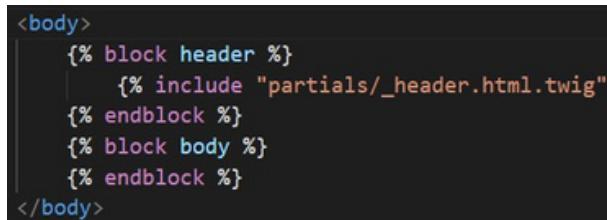
Le Header

Dans les projets Symfony, il est commun de **séparer les éléments** les uns des autres afin d'avoir un code à manipuler et à lire. Par exemple, le fichier "header" se trouvera dans le dossier **template/partials/_header**.



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="l
5     <title>{% block
6       <link rel="icon"
```

Dans le fichier **base.html.twig**, nous allons donc créer un "**block header**" dans lequel je vais inclure mon header comme ceci :



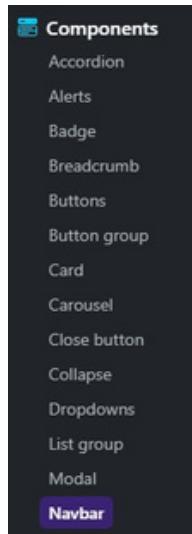
```
<body>
  {% block header %}
    {% include "partials/_header.html.twig"
  {% endblock %}
  {% block body %}
  {% endblock %}
</body>
```

A noter : C'est aussi dans ce fichier que nous allons coller le lien Bootstrap et le lien vers un fichier .CSS

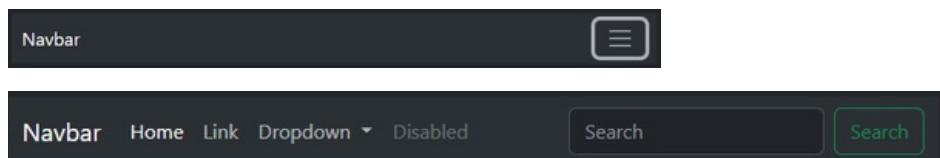
Maintenant, dans chacune des pages que je vais créer au fur et à mesure de mon projet je vais devoir écrire **{% extends 'base.html.twig' %}**

La barre de navigation

Une fois notre fichier "header" créé, nous pouvons nous rendre sur la documentation de Bootstrap et copier un template.



La documentation de **Bootstrap** est un excellent outil pour les développeurs (surtout ceux qui débutent comme moi) puisqu'il propose un large éventail de **composants** déjà codés et open-source. L'intérêt de Bootstrap est aussi du fait que le **"Responsive"** a déjà été travaillé comme nous voyons ici :



L'élément "search" ne nous intéressera pas. Il suffit de le supprimer. On va aussi modifier le nom des éléments.

La feuille de style CSS

Pour changer de couleur il va falloir créer un fichier **.css** que l'on va placer dans le répertoire "**public**" de notre projet.

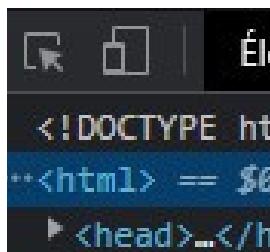
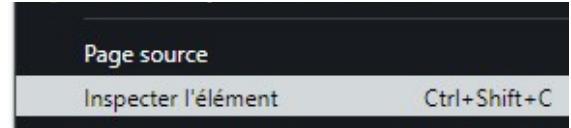
Il est recommandé de créer un sous-répertoire "css". Maintenant, dans le fichier base.html.twig on va écrire dans le block stylesheet :

```
<link rel="stylesheet" href="{{ asset('css/styles.css') }}>
```

Conventionnellement, cette ligne est placée au-dessus de celle de Bootstrap. Cela permet de s'assurer qu'il n'y ait pas de conflit et que notre feuille de style a la priorité.

Puis dans notre fichier .css, il faudra venir appeler le nom de la classe correspondant à la barre de navigation et de modifier sa couleur grâce à la propriété **"background-color"**.

Pour faciliter le choix de la couleur et avoir un rendu en temps réel, on peut faire un clique droit sur la page et "**Inspecter l'élément**". Voici les étapes



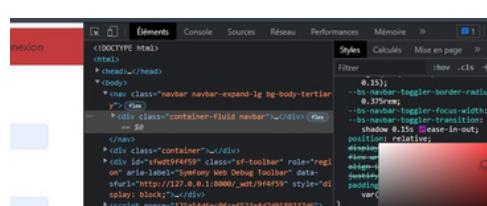
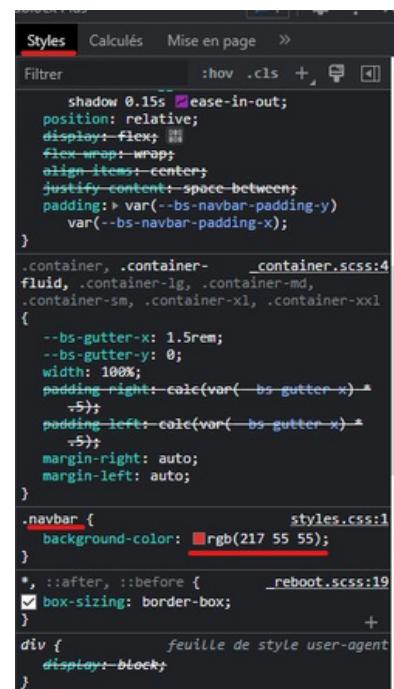
Ensuite, **cliquons sur la flèche** afin de sélectionner un élément de la page. Je clique donc sur la barre de navigation.

Sélectionnons ensuite la barre de navigation. Un onglet "style" apparaît alors sur la droite. Venons ensuite chercher

```
.navbar {  
    background-color :  
}
```

C'est précisément ce que nous avons marqué dans notre feuille de style.css

Pour finir, **cliquons sur le carré** où se trouve la couleur. Nous pouvons la modifier et le résultat sera en direct.



Le footer : le responsive

```
<div class="container">
  <div class="row text-center">
    <div class="col-sm-12 col-md-2 col-lg-2 c">
      A PROPOS DE NOUS
      "Jte dis quoi" est un réseau social d'entraide de voyageurs. Vous avez visité un bar sympa, vous faites partie d'un association et vous avez envie de le partager autour de vous ? Inscrivez vous et partagez votre lieu favori !
    </div>
    <div class="col-md-2">
      NOS SERVICES
      Tous les lieux
      Ajouter un lieu
      Modifier un de vos lieux
    </div>
    <div class="col-md-2">
      TENTEZ L'EXPÉRIENCE
      Inscription
      Connexion
    </div>
    <div class="col-md-2">
      CONTACT
      fa-envelope jtedis@quoi.com
      fa-user admin@jtedisquoi.com
      fa-phone Formulaire de contact
    </div>
  </div>
</div>
```

Copyright 2023 Tous droits réservés par : [Jte dis quoi](#)

[f](#) [t](#) [g](#) [in](#)

```
<div classe="col-sm-12">
  A PROPOS DE NOUS
  "Jte dis quoi" est un réseau social d'entraide de voyageurs. Vous avez visité un bar sympa, vous faites partie d'un association et vous avez envie de le partager autour de vous ? Inscrivez vous et partagez votre lieu favori !
  <div class="col-sm-12">
    NOS SERVICES
    Tous les lieux
    Ajouter un lieu
    Modifier un de vos lieux
  </div>
  <div class="col-sm-12">
    TENTEZ L'EXPÉRIENCE
    Inscription
    Connexion
  </div>
  <div class="col-sm-12">
    CONTACT
    jtedis@quoi.com
    admin@jtedisquoi.com
    Formulaire de contact
  </div>
</div>
```

[f](#) [t](#) [g](#) [in](#)

Même principe pour le footer, nous le plaçons dans le dossier "partials" et nous l'incluons dans le "base.html.twig".

Pour créer le footer nous allons utiliser les **classes Bootstrap**. (cf. annexe 18)

La classe "**container**" permet de centrer le contenu à l'écran en y ajoutant des marges horizontales automatiques.

La classe "**row**" va disposer les éléments sur une ligne verticale. La classe "**col**" va quant à elle disposer les éléments en colonne. La largeur de la colonne sera définie par son attribut.

Pour les icônes, on va utiliser la classe "**fa**" (il faudra importer le lien "**Font-awesome**" dans le **base.html.twig**).

Pour que le footer soit responsive, j'ai utilisé la classe "col-sm-12". C'est à dire que sur un petit écran (small), le contenu aura pour unité d'espacement de 12. (l'équivalent de 12 colonnes).

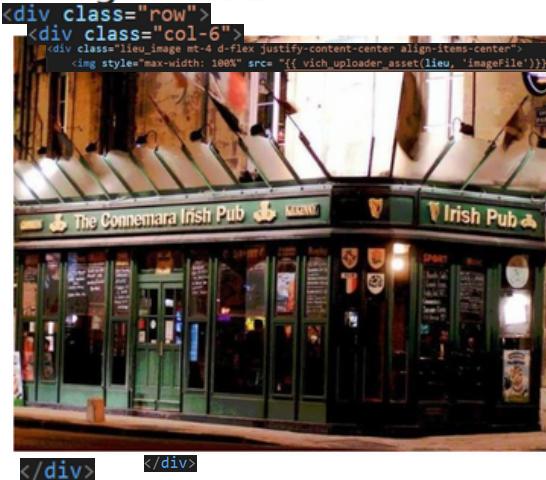
Un breakpoint (point d'arrêt) est un point de discontinuité, modification ou interruption. Dans le contexte de conception et de développement frontend, un point d'arrêt est le point à partir duquel la mise en page change. Généralement, un nombre de pixels représente la largeur de la taille d'écran affichée à laquelle le point d'arrêt se produit.

Les autres classes Bootstrap utilisées dans ce code sont

- "mx-auto" est utilisé pour centrer les colonnes dans la section du pied de page.
- "text-uppercase", pour mettre les caractères en majuscule.
- "font-weight-bold" pour mettre les caractères en gras
- "text-info" change la couleur en bleu
- "hr" est une ligne
- "d-flex" pour activer le "flex" qui fait référence à "flexbox" de CSS

```
<div class="container mt-4">
    <h1>{{lieu.nomLieu}}</h1>
```

Le Nuage Privé 2



```
<div class="row">
    <p>{{lieu.description}}</p>
```

Description :

```
Ratione culpa sed qui sit eos veniam iure repellat aut modi ut perspiciatis eos maiores sit modi dolores quis qui quos eos sed accusamus consequuntur nulla.vdfv
</div>
```

• Nicolas Bazin-Alves

Super !

Le 20/01/2023 à 21:49

```
</div>
```

```
<div class="row">
    <div class="col-6 mt-4">
        <ul class="comment-list">
            {# for commentaire in commentaires %}
            <li>
                <div class="">
                    <h4>{{commentaire.user.fullName}}</h4>
                    <p>{{commentaire.content}}</p>
                    <small>Le {{commentaire.createdAt|date("d/m/Y")}} à
                        {{commentaire.createdAt|format_datetime('none', 'short', locale='fr')}}</small>
                </div>
            </li>
            {# endfor %}
        </ul>
    </div>
```

Pour changer le style de la pagination, nous allons utiliser le style de Boostrap.
Pour ce faire, il faut se rendre dans le fichier "**knpPaginator.yaml**" qui se trouve
dans le dossier "packages" et coller cette ligne :

```
packages
  ! cache.yaml
  ! debug.yaml
  ! doctrine_migrations.yaml
  ! doctrine.yaml
  ! fosckeditor.yaml
  ! framework.yaml
  ! karsen_recaptcha3.yaml
  ! knpPaginator.yaml
  ! mailer.yaml
```

```
pagination: '@KnpPaginator/Pagination/bootstrap_v5_pagination.html.twig'
```

Voici le résultat :



Vous trouverez des
exemples d'autres
pages dans les
annexes (19).

Conclusion

L'application est développée. La prochaine étape a été de tester entièrement l'application, de corriger certains bugs et d'optimiser le code. C'est aussi dans cette étape que l'on va s'attarder sur la mise en page et le design, vérifier que le rendu est bien responsive en utilisant plusieurs supports.

L'application étant destinée au web, pour la déployer, il faudrait acheter un nom de domaine et l'héberger sur, par exemple OVH.

Pour de futurs améliorations de l'application, il serait intéressant qu'un utilisateur puisse directement ajouter sur l'API mapbox des lieux qu'il a aimé.

Remerciements

Je tiens à remercier mon formateur Alexandre Dessoly pour son investissement et ses qualités humaines. Cette formation a été très intense et il a su trouver les mots pour m'encourager.

Je remercie aussi les autres apprenants ainsi que toute la communauté de développeurs, car sans eux je serais encore en train de chercher cette erreur de syntaxe dans mon code.

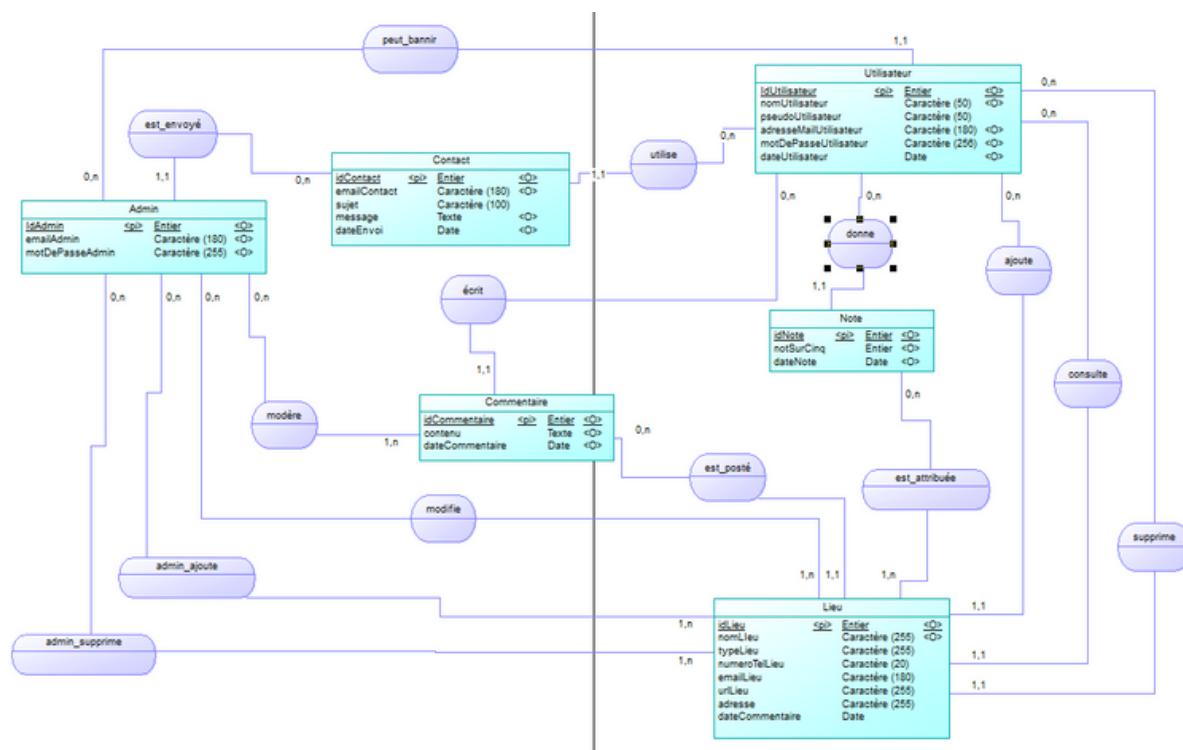
Annexes

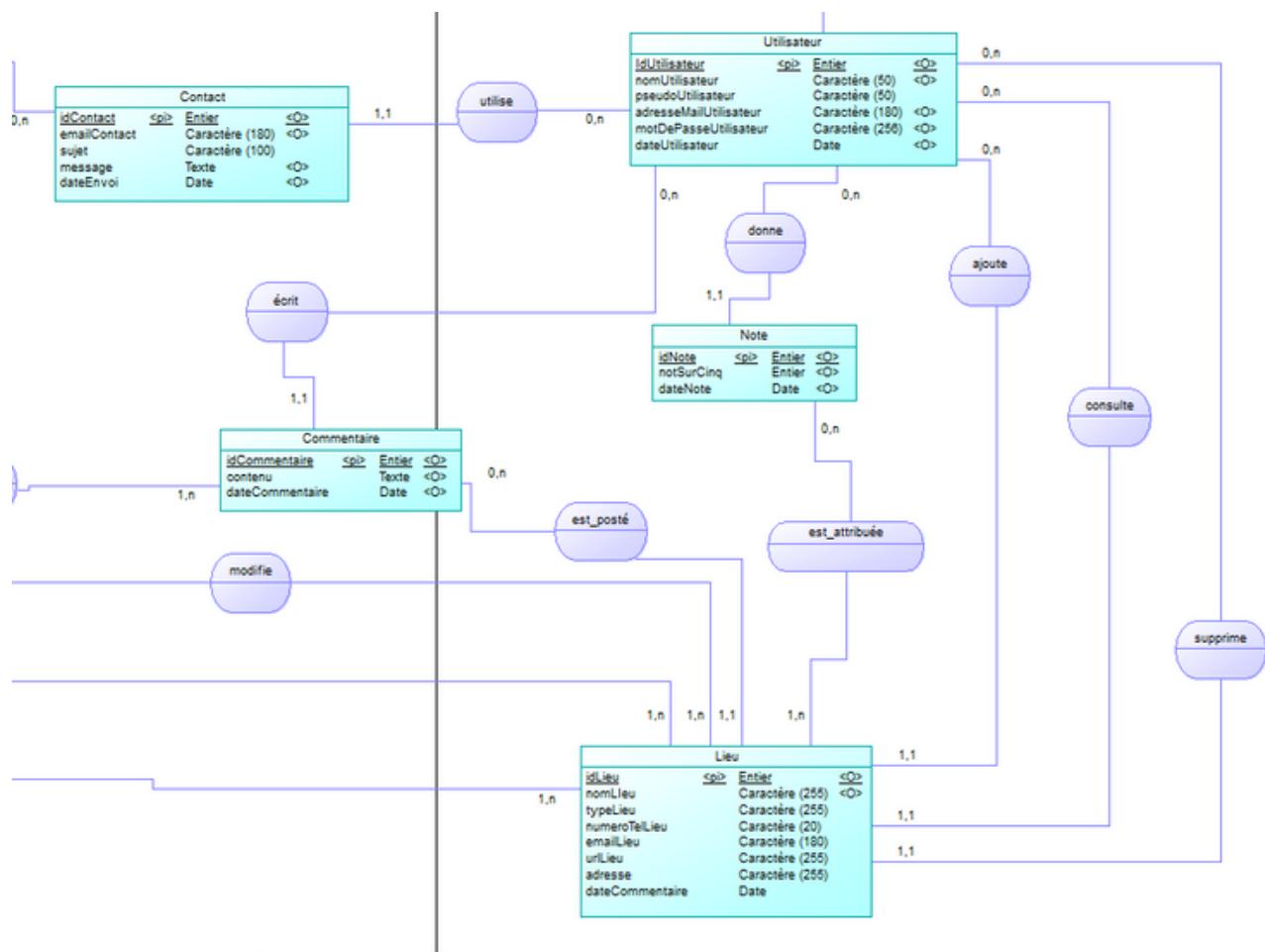
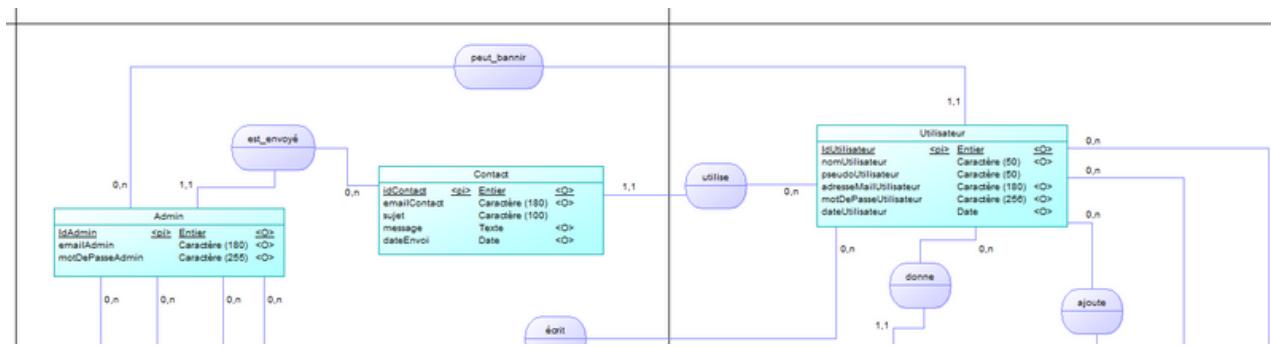
| | |
|---|----|
| Annexe 1 - MCD | 38 |
| Annexe 2 - Dictionnaire de données et UML | 40 |
| Annexe 3 - Maquettes | 42 |
| Annexe 4 - Fixtures | 44 |
| Annexe 5 - Base de données | 46 |
| Annexe 6 - entityListeners | 48 |
| Annexe 7 - Formulaire de connexion | 49 |
| Annexe 8 - VichUploader | 50 |
| Annexe 9 - CRUD - Create | 51 |
| Annexe 10 - Noter un lieu | 54 |
| Annexe 11 - Commentaires | 55 |
| Annexe 14 - Mailtrap | 56 |
| Annexe 15 - Recaptcha | 57 |
| Annexe 16 - EasyAdmin | 61 |
| Annexe 17 - Navigation | 63 |
| Annexe 18 - Bootstrap | 64 |
| Annexe 19 - Exemples pages | 65 |
| Annexe 20 - Bonus : Api Mapbox | 66 |
| Annexe 21 - Erreurs courantes | 67 |

Le dictionnaire de données et le modèle conceptuel de données (MCD)

Dictionnaire de données Jte dis quoi

| | A | B | C | D |
|----|--|-------------------------|---------------------------|--|
| 1 | Nom | ID | Type | Commentaires |
| 2 | utilisateur | idUtilisateur | integer | obligatoire, principal |
| 3 | nom de l'utilisateur | nomUtilisateur | string (min: 2, max: 50) | |
| 4 | pseudo utilisateur | pseudoUtilisateur | string (min: 2, max: 50) | facultatif |
| 5 | email utilisateur | emailUtilisateur | string (min: 2, max: 180) | |
| 6 | rôle de l'utilisateur | roleUtilisateur | json | ROLE_USER ou ROLE_ADMIN |
| 7 | date de création de l'utilisateur | dateCreationUtilisateur | dateTimeImmutable | automatique |
| 8 | mot de passe | passwordUtilisateur | password, string | |
| 9 | lieu | idLieu | integer | obligatoire, principal |
| 10 | nom du lieu | nomLieu | string (min: 2, max: 255) | |
| 11 | type de lieu | typeLieu | string (min: 2, max: 255) | |
| 12 | numéro de téléphone du lieu | numeroTelLieu | string (min: 2, max: 20) | |
| 13 | email du lieu | emailLieu | string (min: 2, max: 180) | |
| 14 | site web du lieu | urlLieu | string (min: 2, max: 255) | |
| 15 | date de création du lieu | dateCreationLieu | dateTimeImmutable | automatique |
| 16 | note du lieu | idNote | integer | obligatoire, principal, doit être positif et inférieur à 6 |
| 17 | date de la note | dateCreationNote | dateTimeImmutable | automatique |
| 18 | commentaire | idCommentaire | integer | obligatoire, principal |
| 19 | contenu | contenuCommentaire | text | |
| 20 | date du commentaire | dateCreationCommentaire | dateTimeImmutable | automatique |
| 21 | Formulaire de contact | idContact | integer | obligatoire |
| 22 | nom de l'utilisateur souhaitant envoyer un msg | nomContact | string (min: 2, max: 50) | |
| 23 | email de l'utilisateur | emailContact | string (min: 2, max: 180) | obligatoire |
| 24 | sujet | sujetContact | string (min: 2, max: 100) | obligatoire |
| 25 | contenu du message | contenuMessage | text | |
| 26 | date d'envoi | dateEnvoy | dateTimeImmutable | automatique |





UML

Diagramme d'activité d'un utilisateur authentifié

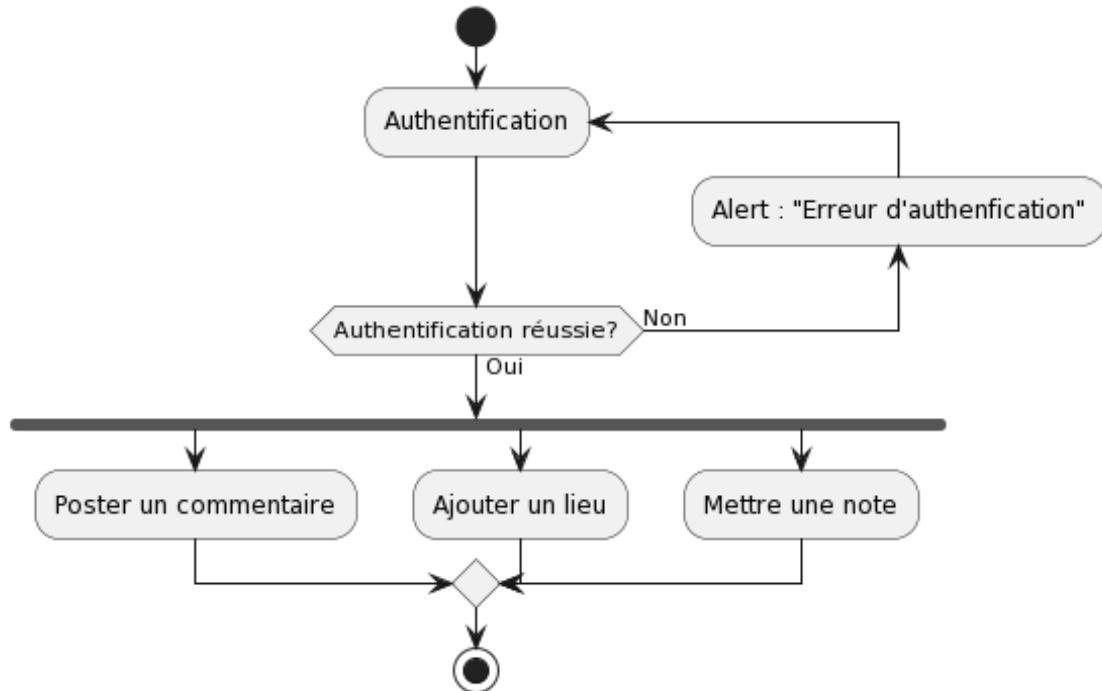


Diagramme d'activité Commentaire lieu

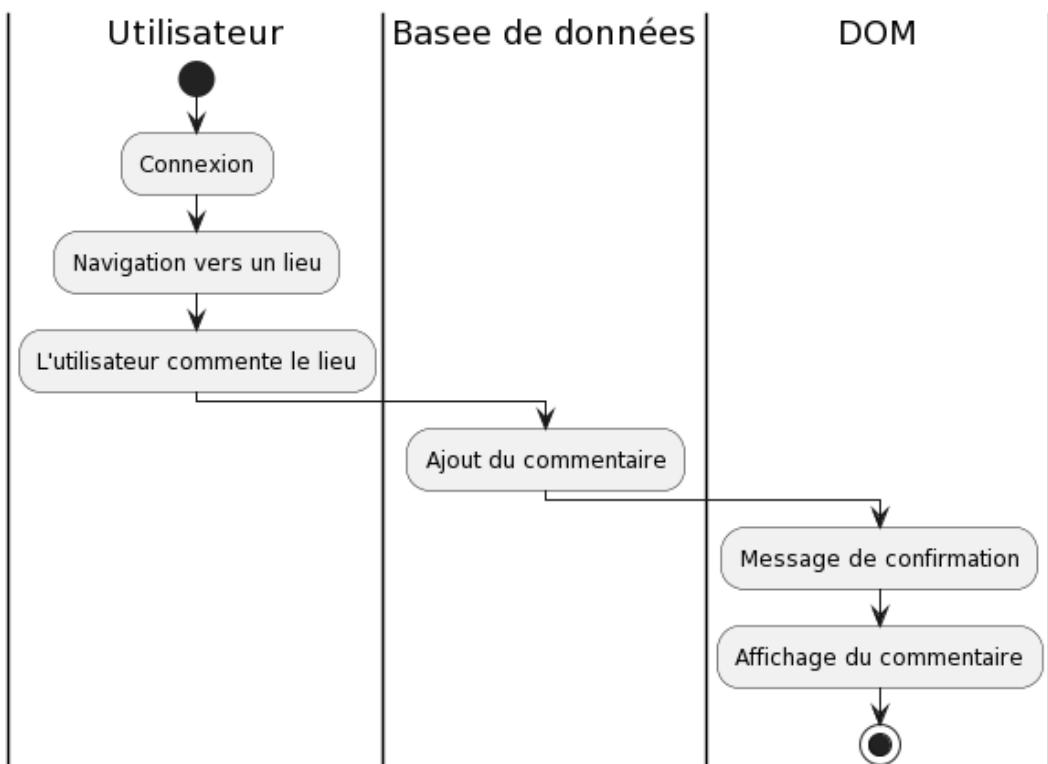


Diagramme d'activité d'un utilisateur attribuant une note à un lieu

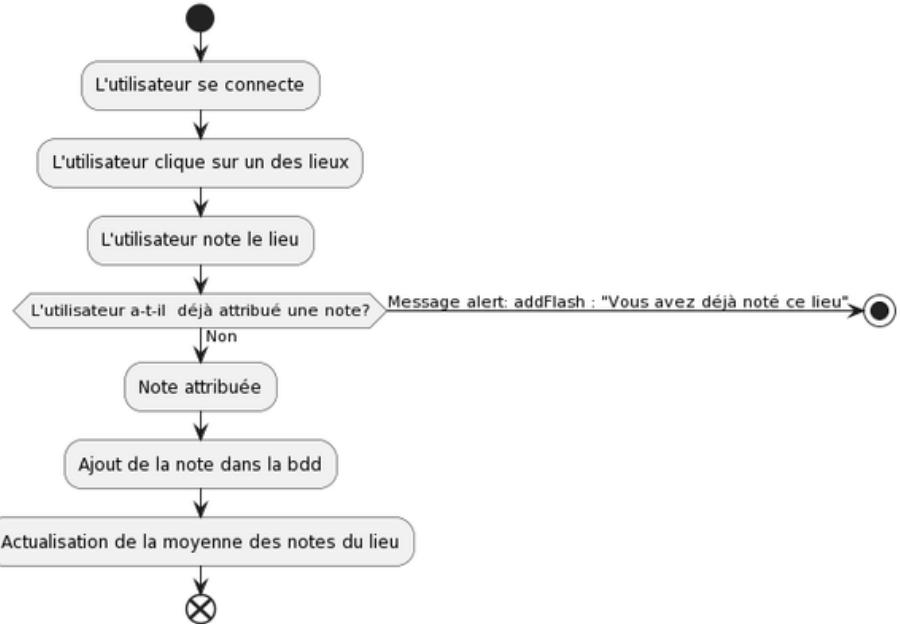
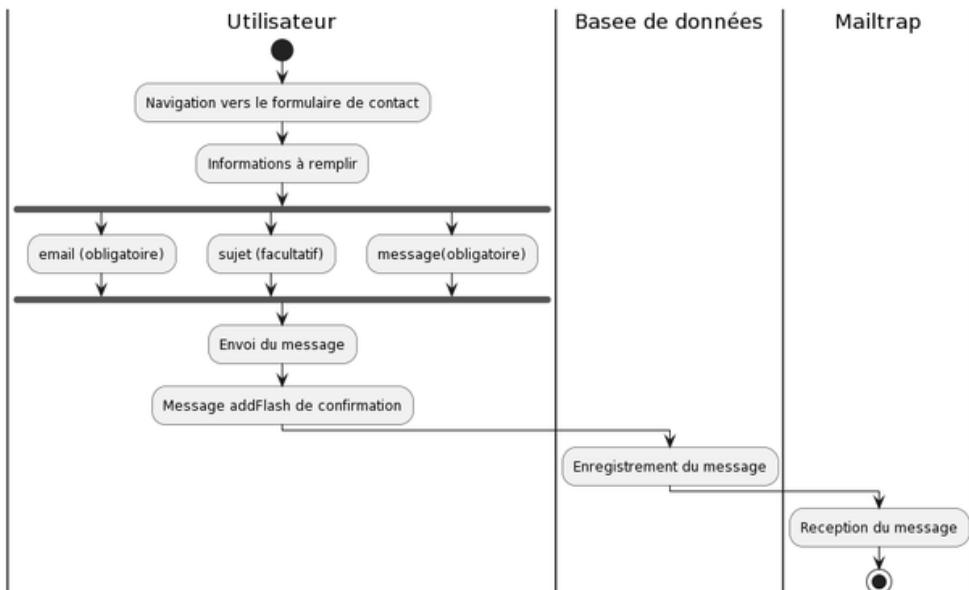
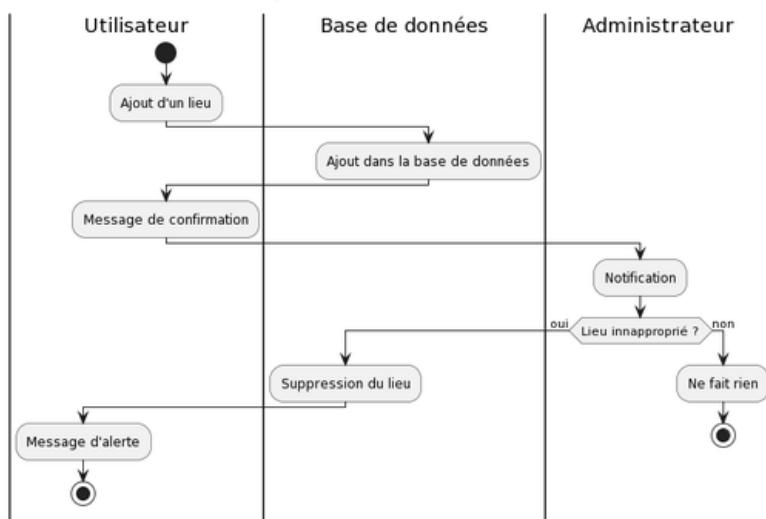


Diagramme d'activité Envoi d'un message via formulaire de contact



Ajout d'un lieu et modération



Maquette

Mes lieux

LOGO Accueil Tous les lieux Mes lieux Ajouter un lieu Contact Nom Utilisateur ▾

Mes lieux

Ajouter un nouveau lieu.

Il y a 5 lieux enregistrés

| Nom | Type | Numéro de téléphone | Email | Site web | Date de création | Édition | Suppression |
|----------------|------------|---------------------|-----------------|-----------------|------------------|---------------------------|----------------------------|
| Jte dis quoi ! | Restaurant | +33 5 56 22 14 15 | jtedisquoil.com | jtedisquoil.com | 19/05/2023 | <button>Modifier</button> | <button>Supprimer</button> |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

A PROPOS DE NOUS

Description de l'application

LES SERVICES

Ajouter un lieu
Noter un lieu
Commenter
Tous les lieux

PROFILE

Inscription
Connexion

CONTACT

email de contact
redirection vers formulaire

Copyright 2023 Tous droits réservés par : Jte dis quoi !

Formulaire d'inscription

LOGO Accueil Tous les lieux Mes lieux Ajouter un lieu Contact Nom Utilisateur ▾

Formulaire d'inscription

Nom / Prénom

Pseudo

Adresse email

Mot de passe

Confirmation du mot de passe

Valider

A PROPOS DE NOUS

Description de l'application

LES SERVICES

Ajouter un lieu
Noter un lieu
Commenter
Tous les lieux

PROFILE

Inscription
Connexion

CONTACT

email de contact
redirection vers formulaire

Copyright 2023 Tous droits réservés par : Jte dis quoi !

Formulaire de connexion

Adresse email

Mot de passe

[Se connecter](#)

A PROPOS DE NOUS

Description de l'application

LES SERVICES

Ajouter un lieu
Noter un lieu
Commenter
Tous les lieux

PROFILE

Inscription
Connexion

CONTACT

email de contact
redirection vers formulaire

Copyright 2023 Tous droits réservés par : Jte dis quoi

Formulaire de contact

Nom / Prénom

Adresse email

Sujet

Message

[Envoyer](#)

A PROPOS DE NOUS

Description de l'application

LES SERVICES

Ajouter un lieu
Noter un lieu
Commenter
Tous les lieux

PROFILE

Inscription
Connexion

CONTACT

email de contact
redirection vers formulaire

Copyright 2023 Tous droits réservés par : Jte dis quoi

Fixtures

```
1 <?php
2
3 namespace App\DataFixtures;
4
5 use Faker\Factory;
6 use App\Entity\Lieu;
7 use App\Entity\Mark;
8 use App\Entity\User;
9 use Faker\Generator;
10 use Doctrine\Persistence\ObjectManager;
11 use Doctrine\Bundle\FixturesBundle\Fixture;
12
13 class AppFixtures extends Fixture
14 {
15     /**
16      * Faker Generator
17      *
18      * @var Generator
19      */
20     private Generator $faker;
21
22     public function __construct()
23     {
24         $this->faker = Factory::create('fr_FR');
25     }
26 }
```

```
public function load(ObjectManager $manager): void
{
    // creating users variable into an empty array
    $users = [];

    //setting admin
    $admin = new User();
    $admin->setFullName('Administrateur')
        ->setPseudo(null)
        ->setEmail('admin@jtedisquoi.fr')
        ->setRoles(['ROLE_USER', 'ROLE_ADMIN'])
        ->setPlainPassword('password');
    $user[] = $admin;
    $manager->persist($admin);
    // creating 10 users
    for ($i=0; $i < 10; $i++) {
        $user = new User();
        // using faker to generate a name
        $user->setFullName($this->faker->name())
        // mt_rand generate a random value (between 0 or 1 in this example) if = 1 then faker generate a nickname esle null
        ->setPseudo(mt_rand(0, 1) === 1 ? $this->faker->firstName() : null )
        ->setEmail($this->faker->email())
        ->setRoles(['ROLE_USER'])
        ->setPlainPassword('password');
        $users[] = $user;
        $manager->persist($user);
    }
}
```

```

$lieux = [];
$typeLieu = ['Bar', 'Restaurant', 'Musée', 'Salle de concert', 'Hôtel',
'Jardins', 'Parc', 'Association'];
$nomLieu = ['Le Palais Clair', 'Le Château de la Plage', 'La Fable de Bronze', 'Le Mélange du Quai',
'La Légende', 'Le Colibri', 'Le Mur', 'Séduction', 'la Niche', 'Lueur des Étoiles',
'Le Nuage d'Orange', 'La Table Chaude', 'Le Lieu de Sarrette', 'Le Nuage Privé',
'Le Balcon de Cuisson', 'Le GastroGnome', 'Le Calme', 'Le Saphir', 'L'Amusement', 'Le Lis',
'La Capture Rose', 'La Cabane Italienne', 'La Vallée Ovale', 'Le Morceau du Canal',
'Le Hall Solaire', 'Le Dépôt', 'Élémentaire', 'La Tulipe', 'La Gemme', 'L'Échange de Cannelle',
'Le Balcon de la Plage', 'Le Pétales Violet', 'L'Usine Argentée', 'Le Boulevard Lunaire',
'La Perle rare', 'Bambino', 'L'île', 'Lueur des Songes', 'La Caverne', 'Le Lis de Paume',
'La Saveur Thailandeuse', 'Le Piment Doré', 'Le Moulin d'Hiver', 'Le Sanglier Silencieux',
'La Chance', 'Le Coquin', 'le Marmonnement', 'Piccolo', 'Révélations', 'Chez Anouk'];

for ($i=0; $i < 50 ; $i++) {
    $lieu = new Lieu();
    $lieu->setTypeLieu($typeLieu[mt_rand(0, count($typeLieu) -1)]);
    $lieu->setNumeroTeliu($this->faker->randomNumber(9, true));
    $lieu->setNomLieu($nomLieu[$i]);
    $lieu->setEmailLieu($this->faker->email());
    $lieu->setUrlLieu($this->faker->domainName());
    // a user will be assigned to a random user
    $lieu->setUser($users[mt_rand(0, count($users) -1)]);
    $lieu->setDescription($this->faker->sentence(mt_rand(10, 30)));

    $lieux[] = $lieu;
    $manager->persist($lieu);
}

```

```

foreach ($lieux as $lieu) {
    // giving between 0 and 4 marks
    for ($i=0; $i < mt_rand(0, 4) ; $i++) {
        $mark = new Mark();
        // mark will be between 1 and 5
        $mark->setMark(mt_rand(1, 5))
        // by random users
        ->setUser($users[mt_rand(0, count($users) -1)])
        // to lieu
        ->setLieu($lieu);

        $manager->persist($mark);
    }
}
$manager->flush();
}

```

Base de données

Serveur : 127.0.0.1 > Base de données : tledisque

Filtres

Contenant le mot :

| Table | Action | Lignes | Type | Interclassement | Taille | Perte |
|-----------------------------|--|--------|--------|--------------------|-----------|-------|
| commentaire | Parcourir Structure Rechercher Insérer Vider Supprimer | 21 | InnoDB | utf8mb4_unicode_ci | 48,0 kio | - |
| contact | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_unicode_ci | 16,0 kio | - |
| doctrine_migration_versions | Parcourir Structure Rechercher Insérer Vider Supprimer | 3 | InnoDB | utf8_unicode_ci | 16,0 kio | - |
| lieu | Parcourir Structure Rechercher Insérer Vider Supprimer | 44 | InnoDB | utf8mb4_unicode_ci | 32,0 kio | - |
| mark | Parcourir Structure Rechercher Insérer Vider Supprimer | 48 | InnoDB | utf8mb4_unicode_ci | 48,0 kio | - |
| messenger_messages | Parcourir Structure Rechercher Insérer Vider Supprimer | 0 | InnoDB | utf8mb4_unicode_ci | 64,0 kio | - |
| user | Parcourir Structure Rechercher Insérer Vider Supprimer | 10 | InnoDB | utf8mb4_unicode_ci | 32,0 kio | - |
| 7 tables | Somme | 126 | InnoDB | utf8mb4_general_ci | 256,0 kio | 0 o |

Tout cocher Avec la sélection :

SELECT * FROM `user`

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Crée le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par clé : Aucun(e)

Options supplémentaires

| id | full_name | pseudo | email | roles (DC2Type:json) | password | created_at (DC2Type:datetime_immutable) |
|----|----------------------|-----------|------------------------------|----------------------------|--|---|
| 89 | Administrateur | NULL | admin@tledisque.fr | ["ROLE_USER","ROLE_ADMIN"] | \$2y\$13\$svRwhA6ZdnY/TonXUffewUPaJLqbET6qHTQw/ZN... | 2023-01-19 10:13:02 |
| 90 | Nicolas Bazin-Alves | NULL | david@rousseau.net | ["ROLE_USER"] | \$2y\$13\$sdGig6/huOQCxFcMtgYZledircXMv5ukR7Luck/U9sq... | 2023-01-19 10:13:02 |
| 92 | Suzanne Guyot | Thibault | zrenaud@bouvet.org | ["ROLE_USER"] | \$2y\$13\$YqqAddJBhgvr0YLwvB09H4nu0zN/gAvpA6VBo... | 2023-01-19 10:13:04 |
| 93 | Céline Blanchard | NULL | legende.sophie@joseph.org | ["ROLE_USER"] | \$2y\$13\$h7pQuA/OX8MWKYYkSyelidxSiZucBYMrYvnDfeaNT5... | 2023-01-19 10:13:05 |
| 94 | Georges-Marcel Gomez | Margaud | mfoucher@delanay.com | ["ROLE_USER"] | \$2y\$13\$GqOCPE1xRf4wchH54l1TepPFzmlnxYdcavZPvpgj... | 2023-01-19 10:13:05 |
| 95 | Augustin Rodrigues | André | bouchet.matthieu@laposte.net | ["ROLE_USER"] | \$2y\$13\$Fbabe9SM2bcCcoVd3n2uIzBvGI/GXhT2WICVVoc... | 2023-01-19 10:13:06 |
| 96 | Richard Godard | Geneviève | evrard.virginie@daniel.com | ["ROLE_USER"] | \$2y\$13\$QuxF_FKEM2VO2q263YDeDl8hvxdMLV7qydc04f... | 2023-01-19 10:13:07 |
| 97 | Thierry-Marc Caron | NULL | ufrancois@perret.com | ["ROLE_USER"] | \$2y\$13\$zgenJIBraA9Od5RNPa3aTuZYY6Eu2oKCA9TyBV0eKN... | 2023-01-19 10:13:07 |
| 98 | Céline du Dupuis | Constance | iklein@huet.com | ["ROLE_USER"] | \$2y\$13\$WLtk0fb55s0RK7B9K1NuceE9Qk3gCyuJH54Ny7zN... | 2023-01-19 10:13:08 |
| 99 | René Thierry | William | josephine82@david.org | ["ROLE_USER"] | \$2y\$13\$EhJ3qTYzIKvsQTANIFP1XuMV.flbE0plhVmVNaK7... | 2023-01-19 10:13:09 |

Tout cocher Avec la sélection : Éditer Copier Supprimer Exporter

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par clé : Aucun(e)

SELECT * FROM `commentaire`

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Crée le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par clé : Aucun(e)

Options supplémentaires

| id | lieux_id | content | created_at (DC2Type:datetime_immutable) | user_id |
|----|----------|------------|---|---------|
| 7 | 401 | Super ! | 2023-01-19 15:46:09 | 90 |
| 16 | 401 | Incroyable | 2023-01-19 16:36:16 | 90 |
| 28 | 414 | génial ! | 2023-01-31 16:12:08 | 90 |

Tout cocher Avec la sélection : Éditer Copier Supprimer Exporter

| | <input type="checkbox"/> Profilage | Éditer en ligne | Éditer | Expliquer SQL | Créer le code source PHP | Actualiser |
|--|------------------------------------|---|-------------------------|-------------------------------|--|---|
| | | 1 | > | >> | <input type="checkbox"/> Tout afficher | Nombre de lignes : 25 |
| | | | | | | Filtrer les lignes : <input type="text"/> |
| | | Options supplémentaires | | | | |
| | | Éditer | Copier | Supprimer | id | user_id |
| | | Éditer | Copier | Supprimer | 630 | 90 |
| | | Éditer | Copier | Supprimer | 631 | 90 |
| | | Éditer | Copier | Supprimer | 632 | 94 |
| | | Éditer | Copier | Supprimer | 633 | 95 |
| | | Éditer | Copier | Supprimer | 634 | 95 |
| | | Éditer | Copier | Supprimer | 635 | 98 |
| | | Éditer | Copier | Supprimer | 637 | 90 |
| | | Éditer | Copier | Supprimer | 638 | 94 |
| | | Éditer | Copier | Supprimer | 639 | 97 |
| | | Éditer | Copier | Supprimer | 640 | 96 |
| | | Éditer | Copier | Supprimer | 641 | 94 |
| | | Éditer | Copier | Supprimer | 642 | 93 |
| | | Éditer | Copier | Supprimer | 643 | 98 |
| | | Éditer | Copier | Supprimer | 644 | 95 |
| | | Éditer | Copier | Supprimer | 648 | 98 |

entityListeners

```
<?php

namespace App\EntityListener;

use App\Entity\User;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;

class UserListener
{
    private UserPasswordHasherInterface $hasher;

    public function __construct(UserPasswordHasherInterface $hasher)
    {
        $this->hasher = $hasher;
    }

    public function prePersist(User $user) {

        $this->encodePassword($user);

    }
}
```

```
 /**
 * Encode password based on plainpassword
 */

public function encodePassword(User $user)
{
    if($user->getPlainPassword() === null ) {
        return;
    }

    $user->setPassword(
        $this->hasher->hashPassword(
            $user,
            $user->getPlainPassword()
        )
    );
}
```

Formulaire de connexion

```
1  {% extends 'base.html.twig' %}  
2  
3  {% block title %}Jte dis quoi - Connexion{% endblock %}  
4  
5  {% block body %}  
6    <div class="container">  
7      <h1 class="mt-5">Formulaire de connexion</h1>  
8      {% for message in app.flashes('success') %}  
9        <div class="alert alert-success mt-5">  
10       |          {{ message }}  
11       |        </div>  
12       {% endfor %}  
13      {% display error message, user not found %}  
14      {% if error %}  
15        <div class="alert alert-dismissible alert-danger" role="alert">  
16          |          {{ error.messageKey|trans(error.messageData, 'security') }}  
17        </div>  
18      {% endif %}  
19  
20
```

```
<form action="{{ path('security.login') }}" method="post">  
  <div class="mb-3">  
    <label for="username" class="form-label">Adresse email</label>  
    <input type="email" class="form-control" id="username" name="_username" placeholder="exemple@jtedisquoi.fr" value="{{ last_username }}"/>  
    <div id="emailHelp" class="form-text">Nous ne partagerons jamais votre email</div>  
  </div>  
  <div class="mb-3">  
    <label for="password" class="form-label">Mot de passe</label>  
    <input type="password" class="form-control" id="password" name="_password" placeholder="*****"/>  
  </div>  
  <button type="submit" class="btn btn-primary mt-4">Se connecter</button>  
</form>  
</div>  
{% endblock %}
```

VichUploader



Il serait en effet dommage de ne pas pouvoir ajouter des photos. Nous allons donc installer le bundle : VichUploader `composer require vich/uploader-bundle`

Rendons-nous dans le dossier **packages** puis sélectionnons le fichier **vich_uploader.yaml**

```
vich_uploader:  
    db_driver: orm  
    metadata:  
        type: attribute  
#mappings tells to the bundle vich where to save files  
    mappings:  
        jtedisquoi_images:  
            #path  
                uri_prefix: /images/lieux  
                upload_destination: '%kernel.project_dir%/public/images/lieux'  
                namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
```

Dans ce fichier, on va décommenter le mapping, ajouter un nom, ainsi que dans le `uri_prefix` et le `upload_destination`. En suivant la documentation, on lit qu'il faut aussi ajouter la ligne **namer** et les **metadata**

Ensute, on va le lier à une entité (dans notre cas, l'entité "lieu") et importer les "use"

```
# [Vich\Uploadable]  
class Lieu
```

```
use Vich\UploaderBundle\Mapping\Annotation as Vich;  
use Symfony\Component\HttpFoundation\File\File;
```

Le `Vich\Uploadable` indique à l'entité qu'il va y avoir un système d'upload

Puis dans les attributs, on va copier ceci :

```
[Vich\UploadableField(mapping: 'jtedisquoi_images', fileNameProperty: 'imageName',  
size: 'imageSize')]  
private ?File $imageFile = null;  
  
#[ORM\Column(type: 'string')]  
private ?string $imageName = null;  
  
#[ORM\Column(type: 'integer')]  
private ?int $imageSize = null;
```

Il faut configurer le mapping et ne pas oublier les Getter et Setter

[Insert PHP Getter & Setter](#)

L'ImageFile correspond à l'endroit où l'image va être stockée.

L'imageName est le nom de l'image qui sera stockée dans la base de données.

Dans le formulaire, il faudra rajouter une section **VichImageType** et dans le template twig associé un `form_row` :

```
->add('imageFile', VichImageType::class, [  
    'label' => 'Image du lieu',  
    'label_attr' => [  
        'class' => 'form-label mt-4'  
    ]  
])
```

```
<div class="form-group">  
|   {{ form_row(form.imageFile) }}  
</div>
```

CRUD - Create

```
/* we create a route we will see an action and the methods are GET and POST (read to us)
#[Route('/lieu/nouveau', 'lieu.new', methods: ['GET', 'POST'])]
#[IsGranted('ROLE_USER')]
/* we call the EntityManagerInterface, it will help to get into the database */
public function new(Request $request, EntityManagerInterface $manager) : Response {
    $lieu = new Lieu();
    $form = $this->createForm(LieuFormType::class, $lieu);
    // get the request
    $form->handleRequest($request);
    // is form is submitted and valid
    if($form->isSubmitted() && $form->isValid()) {
        $lieu = $form->getData();
        /* when a 'lieu' is created, it will be assigned to the current user */
        $lieu->setUser($this->getUser());
        /* manager has to add ('persist' and 'flush') the data into the database */
        /* Persist is kind of "commit", flush is a "push" */
        $manager->persist($lieu);
        $manager->flush();

        $this->addFlash(
            'success',
            'Votre lieu a bien été enregistré. Merci pour votre aide !'
        );

        /* To redirect to the page 'lieu' */
        return $this->redirectToRoute('mesLieux.index');
    }

    return $this->render('pages/lieu/new.html.twig', [
        'form' => $form->createView()
    ]);
}
```

```
return $this->render('pages/lieu/new.html.twig', [
    'form' => $form->createView()
]);
}
```

```

class LieuFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nomLieu', TextType::class, [
                'attr' => [
                    'class' => 'form-control',
                    'minlength' => '2',
                    'maxlength' => '255'
                ],
                'label' => 'Nom :',
                'label_attr' => [
                    'class' => 'form-label mt-4'
                ],
                'constraints' => [
                    new Assert\Length(['min' => 2, 'max' => 255 ]),
                    new Assert\NotBlank()
                ]
            ])
            ->add('typeLieu', ChoiceType::class, [
                'choices' => [
                    'Bar' => 'Bar',
                    'Restaurant' => 'Restaurant',
                    'Association' => 'Association',
                    'Parc' => 'Parc',
                    'Musée' => 'Musée',
                    'Salle de concert' => 'Salle de concert',
                    'Hôtel' => 'Hôtel'
                ],
                'attr' => [
                    'class' => 'form-select'
                ],
                'label' => 'Type de lieu',
                'label_attr' => [
                    'class' => 'form-label mt-4'
                ]
            ])
            ->add('numeroTelLieu', TelType::class, [
                'attr' => [
                    'class' => 'form-control',
                    'minlength' => '2',
                    'maxlength' => '20'
                ],
                'label' => 'Numéro de téléphone :',
                'label_attr' => [
                    'class' => 'form-label mt-4'
                ],
                'required' => false,
                'constraints' => [
                    new Assert\Length(['min' => 2, 'max' => 20 ]),
                ]
            ])
    }
}

```

```

        ->add('emailLieu', EmailType::class, [
            'attr' => [
                'class' => 'form-control',
                'minlength' => '2',
                'maxlength' => '255'
            ],
            'required' => false,
            'label' => 'Adresse Email :',
            'label_attr' => [
                'class' => 'form-label mt-4'
            ],
            'constraints' => [
                new Assert\Length(['min' => 2, 'max' => 255 ]),
            ]
        ]);

        ->add('urlLieu', UrlType::class, [
            'attr' => [
                'class' => 'form-control',
                'minlength' => '2',
                'maxlength' => '255'
            ],
            'required' => false,
            'label' => 'Site web :',
            'label_attr' => [
                'class' => 'form-label mt-4'
            ],
            'constraints' => [
                new Assert\Length(['min' => 2, 'max' => 255 ]),
            ]
        ]);

        ->add('description', TextType::class, [
            'attr' => [
                'class' => 'form-control',
                'minlength' => '2',
                'maxlength' => '255'
            ],
            'required' => false,
            'constraints' => [
                new Assert\Length(['min' => 2, 'max' => 255 ]),
            ]
        ]);

        ->add('imageFile', VichImageType::class, [
            'required' => false,
            'label' => 'Image du lieu',
            'label_attr' => [
                'class' => 'form-label mt-4'
            ]
        ]);

        ->add('submit', SubmitType::class, [
            'attr' => [
                'class' => 'btn btn-primary mt-4'
            ],
            'label' => 'Valider'
        ]);
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Lieu::class,
        ]);
    }
}

```

Noter un lieu

```
#Route('/lieux/{id}', name: 'lieux.show', methods:['GET', 'POST'])

public function show(Lieu $lieu,
Request $request,
MarkRepository $markRepository,
EntityManagerInterface $manager,
CommentaireRepository $commentaireRepository) : Response
[

    $mark = new Mark();
    $form = $this->createForm(MarkType::class, $mark);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // current user will give a mark
        $mark->setUser($this->getUser())
            ->setLieu($lieu);

        // has the user already given a mark?
        $existingMark = $markRepository->findOneBy([
            'user' => $this->getUser(),
            'lieu' => $lieu
        ]);
        // if existingMark doesn't exist
        if (!$existingMark) {
            // then manager persist and flush $mark
            $manager->persist($mark);
        } else {
            $this->addFlash(
                'fail',
                'Vous ne pouvez pas noter plusieurs fois'
            );
        }
        $manager->flush();
        // display a succes message
        $this->addFlash(
            'success',
            'Votre note à a bien été prise en compte'
        );
    }

    // redirect to lieu.show according the its ID
    return $this->redirectToRoute('lieux.show', ['id' => $lieu->getId()]);
}

return $this->render('pages/lieu/show.html.twig', [
    'lieu' => $lieu,
    'form' => $form->createView(),
    'commentForm' => $commentForm->createView(),
    'commentaires' => $commentaireRepository->findBy(['lieux' => $lieu])
]);
}

<p>Note : {{(lieu.average|number_format(2, '.', ','))}}/5</p>
```

Les commentaires

```
<div class="row">
    <div class="col-6 mt-4">
        <ul class="comment-list m-0">
            {% for commentaire in commentaires %}
                <li>
                    <div class="">
                        <h4> {{commentaire.user.fullName}}</h4>
                        <p>
                            {{commentaire.content}}
                        </p>
                        <small>Le {{commentaire.createdAt|date("d/m/Y")}} à
                            {{commentaire.createdAt|format_datetime('none', 'short', locale='fr')}}</small>
                    </div>
                </li>
            {% endfor %}
        </ul>
    </div>
</div>
```

```
class CommentType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('content', TextareaType::class, [
                'attr' => [
                    'class' => 'form-control',
                ],
                'label' => 'Votre commentaire',
                'label_attr' => [
                    'class' => 'form-label mt-4'
                ],
                'constraints' => [
                    new Assert\NotBlank()
                ],
            ])
            ->add('submit', SubmitType::class, [
                'attr' => [
                    'class' => 'btn btn-primary mt-3'
                ],
                'label' => 'Poster'
            ]);
    }

}
```

```
public function show(Lieu $lieu, Request $request,
MarkRepository $markRepository,      // to display te comment into the view
EntityManagerInterface $manager, CommentaireRepository $commentaireRepository) : Response
{
    $comment = new Commentaire($lieu);
    $commentForm = $this->createForm(CommentType::class, $comment);
    $commentForm->handleRequest($request);

    if ($commentForm->isSubmitted() && $commentForm->isValid()) {
        $comment->setUser($this->getUser())
            ->setLieux($lieu);

        $manager->persist($comment);
        $manager->flush();
        $this->addFlash('success', 'Votre commentaire a bien été posté. Merci !');
    }
}
```

Mailtrap ➤

Mailtrap est un outil en ligne qui permet de tester l'envoi et la réception d'emails dans un environnement de développement. Il faut dans un premier temps se créer un compte Mailtrap, puis nous allons le configurer. Il faudra copier le **MAILER_DNS** et le coller dans le .env

Symfony 5+ +

Symfony uses Symfony's Mailer to send emails. You can find more information on how to send email on [Symfony's website](#).
To get started you need to modify .env file in your project directory and set MAILER_DSN value:

```
MAILER_DSN=smtp://33645558c6c15a:567e2b41ab3c76@smtp.mailtrap.io:2525?encryption=tls&auth_mode=login
```

Copy

Dans le dossier **config/messenger.yaml**, il faudra aussi commenter cette ligne :

```
routing:  
    # Symfony\Component\Mailer\Messenger\SendEmailMessage: async  
    Symfony\Component\Notifier\Message\ChatMessage: async  
    Symfony\Component\Notifier\Message\SmsMessage: async
```

Comme indiqué dans la documentation de symfony, il faudra importer dans le **ContactController**:

```
use Symfony\Component\Mailer\MailerInterface;    use Symfony\Component\Mime\Email;
```



Ne pas oublier de faire une injection de dépendance de **MailerInterface** dans la fonction

Il faudra ensuite rajouter des lignes de code suivante :

```
// Email  
  
$email = (new Email())  
->from($contact->getEmail())  
->to('mtx@revov.com')  
->subject($contact->getSubject())  
->html($contact->getMessage());  
  
$mailer->send($email);
```

Voici le résultat dans Mailtrap :

Search... 🔍 ✉️ ⟳ ✉️ ⚙️

Sujet Test

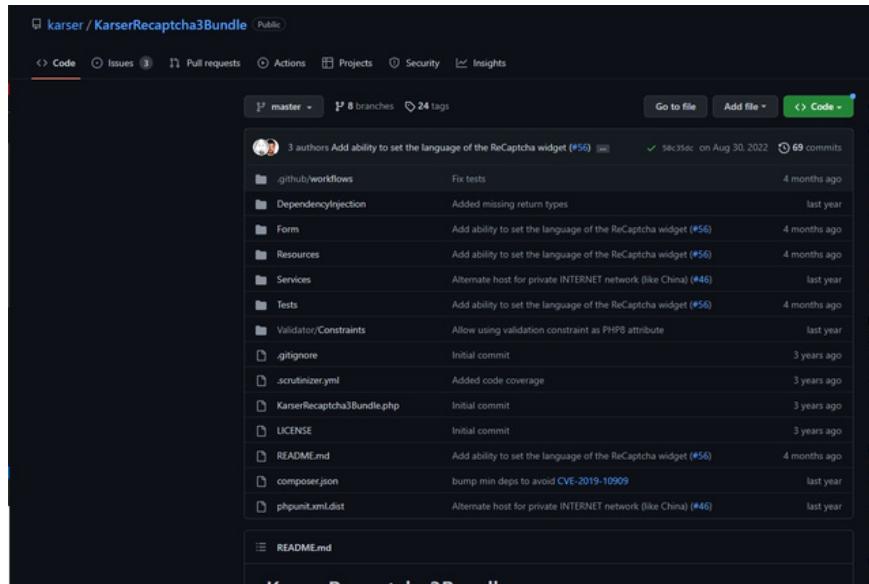
From: <test@test.fr>
To: <mtx@revov.com>

Show Headers

HTML HTML Source Text Raw Spam Analysis HTML Check

Demande de renseignements

Recaptcha



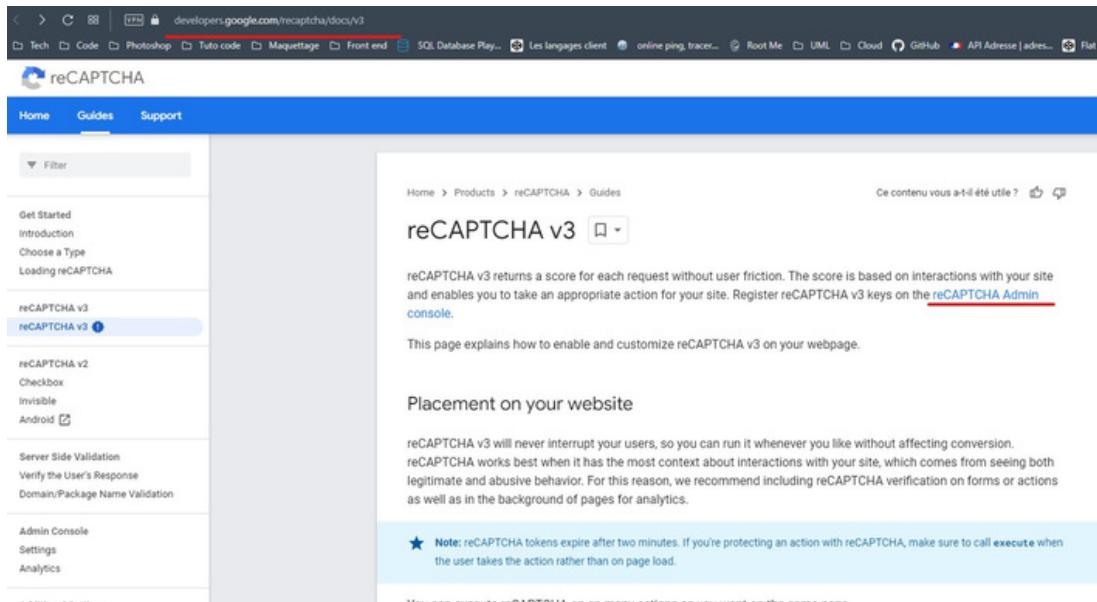
Installation

With `composer`, require:

```
composer require karser/karser-recaptcha3-bundle
```

You can quickly configure this bundle by using `symfony/flex`:

- answer `no` for `google/recaptcha`
- answer `yes` for `karser/karser-recaptcha3-bundle`
 - `WARNING` `google/recaptcha (>=1.1)`: From `github.com/symfony/recipes-contrib:master`
The recipe for this package comes from the "contrib" repository, which is open to community contributions.
Review the recipe at <https://github.com/symfony/recipes-contrib/tree/master/google/recaptcha/1.1>
 - Do you want to execute this recipe?
 - [`y`] Yes
 - [`n`] No
 - [`a`] Yes for all packages, only for the current installation session
 - [`p`] Yes permanently, never ask again for this project
 - [`defaults to n`):
 - `WARNING` `karser/karser-recaptcha3-bundle (>=0.1)`: From `github.com/symfony/recipes-contrib:master`
The recipe for this package comes from the "contrib" repository, which is open to community contributions.
Review the recipe at <https://github.com/symfony/recipes-contrib/tree/master/karser/karser-recaptcha3-bundle/0.1>
 - Do you want to execute this recipe?
 - [`y`] Yes
 - [`n`] No
 - [`a`] Yes for all packages, only for the current installation session
 - [`p`] Yes permanently, never ask again for this project
 - [`defaults to n`):
 - Configuring `karser/karser-recaptcha3-bundle (>=0.1)`: From `github.com/symfony/recipes-contrib:master`
Executing script `cache:clear [OK]`
Executing script `assets:install public [OK]`



Google reCAPTCHA

← Enregistrer un site

Obtenez des évaluations illimitées avec [reCAPTCHA Enterprise](#)

Libellé [i](#)

par exemple, example.com

0 / 50

Type de reCAPTCHA [i](#)

- reCAPTCHA version 3 Valider les requêtes à l'aide d'un score
- reCAPTCHA version 2 Valider les requêtes à l'aide d'un test

Domaines [i](#)

+ Ajouter un domaine, par exemple, example.com

Libellé [i](#)

jtedisquol.com

14 / 50

Type de reCAPTCHA [i](#)

- reCAPTCHA version 3 Valider les requêtes à l'aide d'un score
- reCAPTCHA version 2 Valider les requêtes à l'aide d'un test

Domaines [i](#)

+ localhost

Propriétaires

(vous)

+ Saisir des adresses e-mail

[Accepter les conditions d'utilisation de reCAPTCHA](#)

Vous acceptez d'indiquer explicitement à vos visiteurs que vous avez intégré reCAPTCHA version 3 sur votre site, et que l'utilisation de cette fonctionnalité est soumise aux [Règles de confidentialité](#) et aux [Conditions d'utilisation](#) de Google. La fonctionnalité reCAPTCHA ne doit servir qu'à lutter contre le spam et les abus sur votre site. Elle ne doit pas être utilisée à d'autres fins, par exemple pour déterminer la solvabilité, l'éligibilité à l'emploi, la situation financière ou l'assurabilité d'un utilisateur.

En utilisant les API reCAPTCHA ou en y accédant, vous acceptez les [Conditions d'utilisation](#) des API Google, les [Conditions d'utilisation](#) Google, ainsi que les [Conditions d'utilisation](#) supplémentaires ci-dessous. Avant d'accéder aux API, veuillez prendre connaissance de toutes les conditions et règles applicables.

Conditions d'utilisation de reCAPTCHA [▼](#)

[Envoyer des alertes aux propriétaires](#) [i](#)

ANNULER

ENVOYER

Ajoutez la clé reCAPTCHA à votre site

"jtedisquoi.com" a bien été enregistré.

Utilisez cette clé de site dans le code HTML de votre site destiné aux utilisateurs. [Voir l'intégration côté client](#)

[COPIER LA CLÉ DU SITE](#)

6LeN5_UjAAAAAJsxS7B1D02oI4e0hJH-cWIX4hJ

Utilisez cette clé secrète pour la communication entre votre site et le service reCAPTCHA. [Voir l'intégration côté serveur](#)

[COPIER LA CLÉ SECRÈTE](#)

6LeN5_UjAAAAAD0K2-SugKaHerMopQJvadtssRWm

[ACCÉDER AUX PARAMÈTRES](#)

[ACCÉDER À ANALYTICS](#)

"jtedisquoi.com" a bien été enregistré.

Utilisez cette clé de site dans le code HTML de votre site destiné aux utilisateurs. [Voir l'intégration côté client](#)

[COPIER LA CLÉ DU SITE](#)

6LeN5_UjAAAAAJsxS7B1D02oI4e0hJH-cWIX4hJ

Utilisez cette clé secrète pour la communication entre votre site et le service reCAPTCHA. [Voir l'intégration côté serveur](#)

[COPIER LA CLÉ SECRÈTE](#)

6LeN5_UjAAAAAD0K2-SugKaHerMopQJvadtssRWm

[ACCÉDER AUX PARAMÈTRES](#)

[ACCÉDER À ANALYTICS](#)

```
> security
  < user
    < edit_password.html.twig
    < edit.html.twig
  > partials
    < base.html.twig
  > tests
  > translations
  > var
  > vendor
  < .env
```

```
47 | #####> karser/karser-recaptcha3-bundle #####
48 | # Get your API key and secret from https://g.co/recaptcha/v3
49 | RECAPTCHA3_KEY=6LeN5_UjAAAAAJsxS7B1D02oI4e0hJH-cWIX4hJ
50 | RECAPTCHA3_SECRET=6LeN5_UjAAAAAD0K2-SugKaHerMopQJvadtssRWm
51 | #####< karser/karser-recaptcha3-bundle #####
52 | 
```

Usage

How to integrate re-captcha in Symfony form:

```
<?php

use Karsen\Recaptcha3Bundle\Form\Recaptcha3Type;
use Karsen\Recaptcha3Bundle\Validator\Constraints\Recaptcha3;

class TaskType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('captcha', Recaptcha3Type::class, [
            'constraints' => new Recaptcha3(),
            'action_name' => 'homepage',
            'script_nonce_csp' => $nonceCSP,
            'locale' => 'de',
        ]);
    }
}
```

```
  ...
  ->add('submit', SubmitType::class, [
      'attr' => [
          'class' => 'btn btn-primary mt-3'
      ],
      'label' => 'Envoyer'
  ])
  ->add('captcha', Recaptcha3Type::class, [
      'constraints' => new Recaptcha3(),
      'action_name' => 'contact',
      'locale' => 'de',
  ]);
}
```

Formulaire de contact

Nom / Prénom

Adresse Email

Sujet

Message



EasyAdmin

```
<?php

namespace App\Controller\Admin;

use App\Entity\Lieu;
use App\Entity\User;
use App\Entity>Contact;
use App\Entity\Commentaire;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use EasyCorp\Bundle\EasyAdminBundle\Config\MenuItem;
use EasyCorp\Bundle\EasyAdminBundle\Config\Dashboard;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractDashboardController;

class DashboardController extends AbstractDashboardController
{
    #[Route('/admin', name: 'admin')]
    #[IsGranted('ROLE_ADMIN')]
    public function index(): Response
    {
        return $this->render('admin/dashboard.html.twig');
    }

    public function configureDashboard(): Dashboard
    {
        return Dashboard::new()
            ->setTitle('J\'etends quoi - Administrateur')
            ->renderContentMaximized();
    }

    public function configureMenuItems(): iterable
    {
        yield MenuItem::linkToRoute('Retourner vers le site', 'fa fa-undo', 'home.index');
        yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
        yield MenuItem::linkToCrud('Utilisateurs', 'fas fa-solid fa-users', User::class);
        yield MenuItem::linkToCrud('Demandes de contact', 'fas fa-solid fa-envelope', Contact::class);
        yield MenuItem::linkToCrud('Lieux', 'fas fa-solid fa-location-dot', Lieu::class);
        yield MenuItem::linkToCrud('Commentaires', 'fas fa-comment', Commentaire::class);
    }
}

class UserCrudController extends AbstractCrudController
{
    // this function is mandatory
    public static function getEntityFqcn(): string
    {
        return User::class;
    }

    // this function configure the crud (design, titles, entities...)
    public function configureCrud(Crud $crud): Crud
    {
        return $crud
            // changing the name of the Entity (User) in French
            ->setEntityLabelInPlural('Utilisateurs')
            // when clicking on one single User, the name will be "Utilisateur"
            ->setEntityLabelInSingular('Utilisateur')

            ->setPageTitle("index", "Jte dis quoi - Administration des utilisateurs")
            ->setPaginatorPageSize(10);
    }

    // display the different fields
    public function configureFields(string $pageName): iterable
    {
        return [
            // on the dashboard, there will be displayed :
            // id is the name of the property of the Entity
            IdField::new('id')
                // id won't be display when clicked on editing option
                ->hideOnForm(),
            TextField::new('fullName'),
            TextField::new('pseudo'),
            TextField::new('email')
        ];
    }
}
```

```

//setting admin
$admin = new User();
$admin->setFullName('Administrateur')
->setPseudo(null)
->setEmail('admin@jtedisquoi.fr')
->setRoles(['ROLE_USER', 'ROLE_ADMIN'])
->setPlainPassword('password');
$user[] = $admin;
$manager->persist($admin);



Administration


```

| ID | Full Name | Pseudo | Email | Roles | Created At | ... |
|----|----------------------|-----------|------------------------------|-----------------------|---------------------------|-----|
| 89 | Administrateur | Null | admin@jtedisquoi.fr | ROLE_USER, ROLE_ADMIN | Jan 19, 2023, 10:13:02 AM | ... |
| 90 | Nicolas Bazin-Alves | Null | david@rousset.net | ROLE_USER | Jan 19, 2023, 10:13:02 AM | ... |
| 92 | Suzanne Guyon | Thibault | zrenaud@bouvet.org | ROLE_USER | Jan 19, 2023, 10:13:04 AM | ... |
| 93 | Céline Blanchard | Null | legende.sophie@joseph.org | ROLE_USER | Jan 19, 2023, 10:13:05 AM | ... |
| 94 | Georges-Marcel Gomez | Margaud | mfoucher@defaunay.com | ROLE_USER | Jan 19, 2023, 10:13:05 AM | ... |
| 95 | Augustin Rodrigues | André | bouchet.matthieu@laposte.net | ROLE_USER | Jan 19, 2023, 10:13:06 AM | ... |
| 96 | Richard Godard | Genevieve | evrard.virgine@daniel.com | ROLE_USER | Jan 19, 2023, 10:13:07 AM | ... |
| 97 | Thierry-Marc Caron | Null | ufrancois@perret.com | ROLE_USER | Jan 19, 2023, 10:13:07 AM | ... |
| 98 | Céline du Dupuis | Constance | iklein@huet.com | ROLE_USER | Jan 19, 2023, 10:13:08 AM | ... |
| 99 | René Thierry | William | josephine@2@david.org | ROLE_USER | Jan 19, 2023, 10:13:09 AM | ... |

10 results < Previous 1 Next >

Full Name*

Pseudo

Email*

Roles*

+ Add a new item

| ID | Content | Created At | ... |
|----|------------------------------|--------------------------|-----|
| 7 | View content | Jan 19, 2023, 3:46:09 PM | ... |
| 8 | View content | Jan 19, 2023, 3:48:24 PM | ... |
| 9 | View content | Jan 19, 2023, 3:48:32 PM | ... |
| 10 | View content | Jan 19, 2023, 3:48:54 PM | ... |
| 12 | View content | Jan 19, 2023, 3:51:43 PM | ... |
| 13 | View content | Jan 19, 2023, 3:51:54 PM | ... |
| 14 | View content | Jan 19, 2023, 3:52:07 PM | ... |
| 15 | View content | Jan 19, 2023, 3:52:39 PM | ... |
| 16 | View content | Jan 19, 2023, 4:36:16 PM | ... |

Content

Super !

Navigation

```
<li class="nav-item">
    <a class="nav-link active" aria-current="page" href="{{ path('home.index')}}">Accueil
</a>
</li>
<li class="nav-item">
    <a class="nav-link active" aria-current="page" href="{{ path('lieux.index')}}">Tous les lieux</a>
</li>
{% if app.user %}
<li class="nav-item">
    <a class="nav-link" href="{{ path('mesLieux.index') }}>Mes lieux</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="{{ path('lieu.new') }}>Ajouter un lieu</a>
</li>
{% endif %}
<li>
    <a class="nav-link active" href="{{ path('contact.index')}}">Contact</a>
</li>
```

```
#[Route('/', name: 'home.index', methods:['GET'])]
#[Route('/lieux', name: 'lieux.index', methods:['GET'])]
#[Route('/mesLieux', name: 'mesLieux.index', methods:['GET'])]
#[Route('/lieu/nouveau', 'lieu.new', methods: ['GET', 'POST'])]
#[Route('/contact', name: 'contact.index')]
```

```
<a class="dropdown-item" href="{{path('user.edit', {id : app.user.id})}}>Modifier mes informations</a>
<a class="dropdown-item" href="{{path('user.edit.password', {id : app.user.id})}}>Modifier mon mot de passe</a>

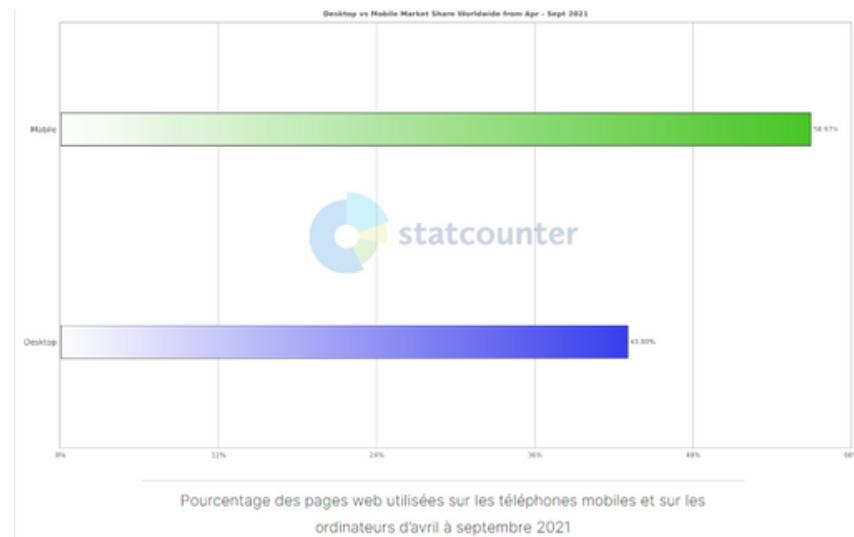
#[Route('/utilisateur/edit/{id}', name: 'user.edit', methods:['GET', 'POST'])]
#[IsGranted('ROLE_USER')]

#[Route('/utilisateur/edit-mot-de-passe/{id}', name:'user.edit.password', methods:['GET', 'POST'])]
```

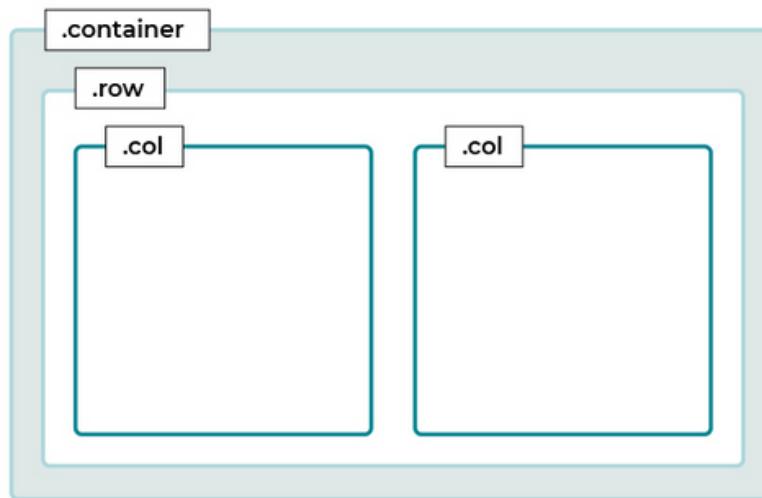
```
// redirecting to login page
    return $this->redirectToRoute('security.login');
```

```
  * @param AuthenticationUtils $authenticationUtils
  */
#[Route('/connexion', name: 'security.login', methods: ['GET', 'POST'])]
```

Bootstrap



Source : OpenClassrooms



Imbrication des éléments de la grille de Bootstrap 5

Source : OpenClassrooms

| Breakpoint | Class infix | Dimensions |
|-------------------|-------------|------------|
| X-Small | None | <576px |
| Small | sm | ≥ 576px |
| Medium | md | ≥ 768px |
| Large | lg | ≥ 992px |
| Extra large | xl | ≥ 1200px |
| Extra extra large | xxl | ≥ 1400px |

Source : Bootstrap

Exemples pages

Logo Accueil Tous les lieux Mes lieux Ajouter un lieu Contact Nicolas Bazin-Alves ▾

Mes lieux

Ajouter un nouveau lieu

Il y a 5 lieux enregistrés !

| Nom | Type | Numéro de téléphone | Email | Site web | Date de création | Édition | Suppression |
|----------------------|-------------|---------------------|-------------------------------|-------------------|------------------|---------------------------|----------------------------|
| Le Nuage Privé 2 | Bar | 960503705 | wbrun@bonnet.net | http://fleury.org | 19/01/2023 | <button>Modifier</button> | <button>Supprimer</button> |
| Le Lis | Association | 467390086 | aurore26@tele2.fr | marques.com | 19/01/2023 | <button>Modifier</button> | <button>Supprimer</button> |
| Le Pétalement Violet | Parc | 453456279 | gauthier.edouard@martinez.com | carlier.com | 19/01/2023 | <button>Modifier</button> | <button>Supprimer</button> |
| La Perle rare | Association | 808385438 | agnes47@dbmail.com | maillard.com | 19/01/2023 | <button>Modifier</button> | <button>Supprimer</button> |
| Révélations | Restaurant | 205004098 | marcelle79@laposte.net | joubert.com | 19/01/2023 | <button>Modifier</button> | <button>Supprimer</button> |

Modification du lieu

Nom :

Le Nuage Privé 2

Type de lieu

Bar

Numéro de téléphone :

960503705

Adresse Email :

wbrun@bonnet.net

Site web :

http://fleury.org

Description

Ratione culpa sed qui sit eos veniam iure repellat aut modi ut perspiciatis eos maiores sit modi dolores quis qui quos eos sed accusamus consequuntur nulla.vdfv

Image
du
lieu

Valider

Image du lieu

Choisir un fichier Aucun fichier choisi

Delete:



[Download](#)

Logo Accueil Tous les lieux Mes lieux Ajouter un lieu Contact Nicolas Bazin-Alves ▾

Formulaire de contact

Nom / Prénom

Nicolas Bazin-Alves

Adresse Email

dvidal@roussel.net

Sujet

Message

Envoyer



Api Mapbox

Bienvenue sur Jte dis quoi !

Jte dis quoi est un réseau social de personnes vivant dans la région Nord et voulant faire découvrir des lieux atypiques et bienveillant aux personnes qui désiraient s'aventurer dans la région

Rejoins la communauté afin de pouvoir profiter pleinement de l'application !

[Inscription](#)



La communauté a déjà ajouté 44 lieux !

Envie de partager un endroit ?

[Clique ici !](#)

Venir récupérer l'api et le coller dans le block javascripts du fichier base.html.twig

```
{% block javascripts %}
{{ encore_entry_script_tags('app') }}
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
<script src="https://api.mapbox.com/mapbox-gl-js/v2.9.1/mapbox-gl.js"></script>
```

Se créer un compte sur mapbox et venir récupérer le token

jtedisquoi

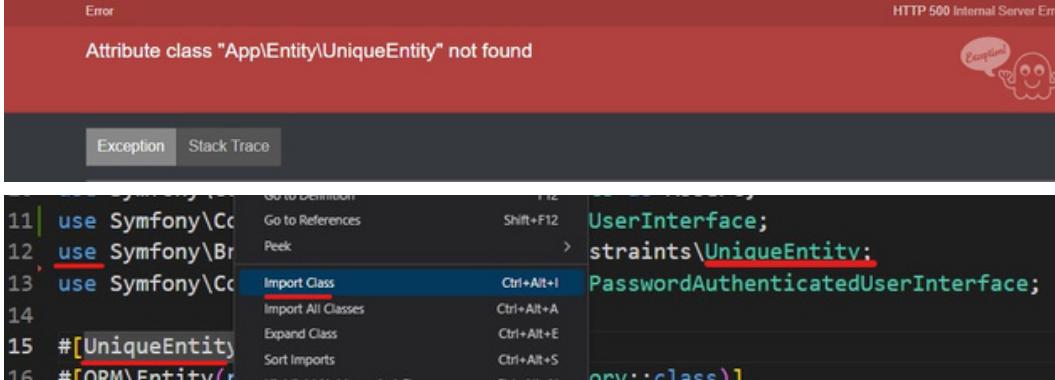
pk.eyJ1IjoiZ2VycnkzMzkwiIiwiYSI6ImNsY3c5ZWZweDFjZnUzb214dDIwNG13MTQifQ.w2EVN-uw6IYUxFQGz1AFVg



Le coller dans le script

```
<div class="row ms-5 mt-5">
  <div class="col-sm-12 col-md-6 ">
    <div id='map' style='width: 400px; height: 300px;'>
      <script>
        mapboxgl.accessToken = 'pk.eyJ1IjoiZ2VycnkzMzkwiIiwiYSI6ImNsY3c5ZWZweDFjZnUzb214dDIwNG13MTQifQ.w2EVN-uw6IYUxFQGz1AFVg';
        var map = new mapboxgl.Map({
          container: 'map',
          style: 'mapbox://styles/mapbox/streets-v11',
          zoom: 11,
          center: [3.060937, 50.631471], //lng, lat
          // http://bboxfinder.com to get exact lng and lat
          projection: 'globe'
        });
      </script>
    </div>
  </div>
</div>
```

Erreurs courantes



The screenshot shows a Symfony error page with a red header and a black body. The header includes the word "Error", the status "HTTP 500 Internal Server Error", and a cartoon ghost icon with a speech bubble saying "Exception". The main content is a stack trace:

```
11 | use Symfony\Component\String\String;
12 | use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
13 | use Symfony\Component\Validator\Constraint;
14 |
15 | #[UniqueEntity]
16 | #[ORM\Entity(r...
```

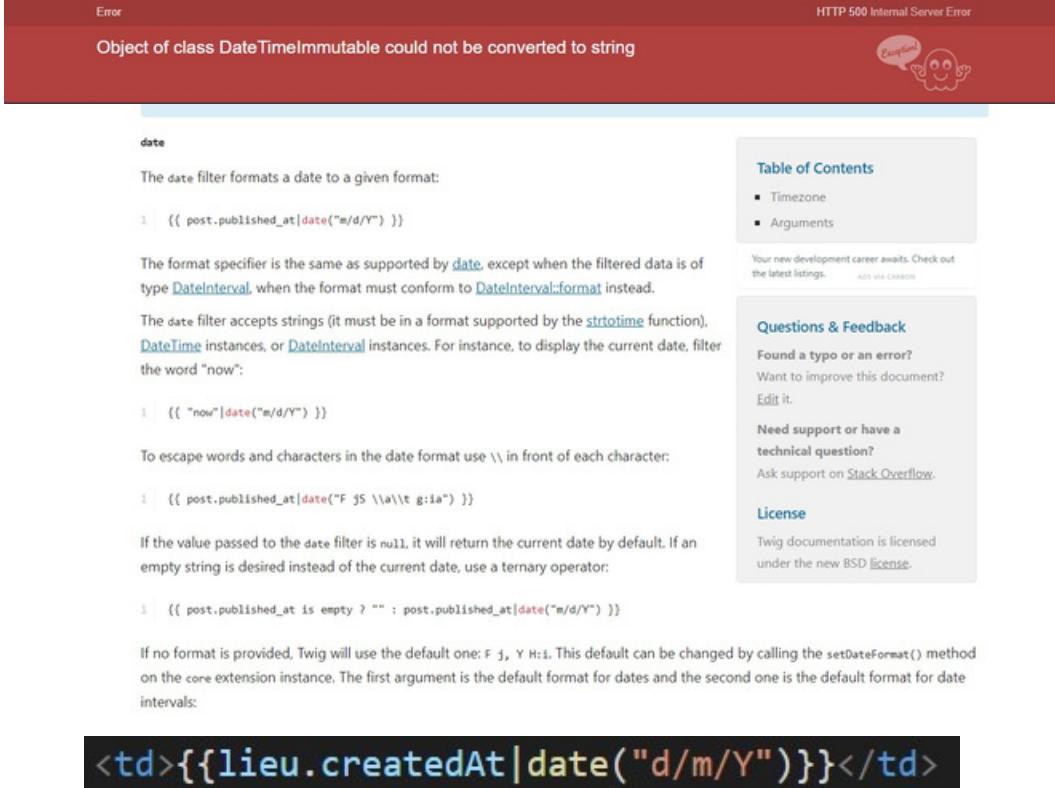
A context menu is open over the line `#[UniqueEntity]` at line 15, with "Import Class" highlighted.



The screenshot shows a Symfony error page with a red header and a black body. The header includes the word "Error", the status "HTTP 500 Internal Server Error", and a cartoon ghost icon with a speech bubble saying "Exception". The main content is an exception message:

An exception occurred in the driver: SQLSTATE[HY000] [2002] Aucune connexion n'a pu être établie car l'ordinateur cible l'a expressément refusée

Solution : Xammp n'est pas lancé



The screenshot shows the Twig documentation page for the `date` filter. The header includes the word "Error", the status "HTTP 500 Internal Server Error", and a cartoon ghost icon with a speech bubble saying "Exception". The main content is a detailed explanation of the `date` filter:

date

The `date` filter formats a date to a given format:

```
1 {{ post.published_at|date("m/d/Y") }}
```

The format specifier is the same as supported by `date`, except when the filtered data is of type `DateInterval`, when the format must conform to `DateInterval::format` instead.

The `date` filter accepts strings (it must be in a format supported by the `strtotime` function), `DateTime` instances, or `DateInterval` instances. For instance, to display the current date, filter the word "now":

```
1 {{ "now"|date("m/d/Y") }}
```

To escape words and characters in the date format use `\` in front of each character:

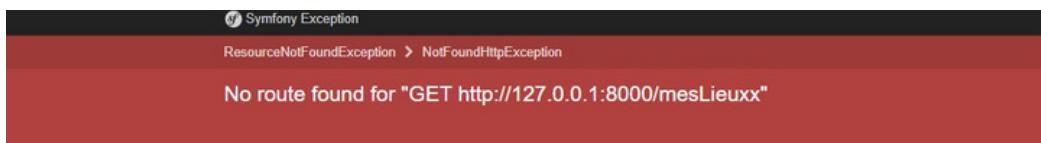
```
1 {{ post.published_at|date("F \S \\a\\t g:i:a") }}
```

If the value passed to the `date` filter is `null`, it will return the current date by default. If an empty string is desired instead of the current date, use a ternary operator:

```
1 {{ post.published_at is empty ? "" : post.published_at|date("m/d/Y") }}
```

If no format is provided, Twig will use the default one: `F j, Y H:i`. This default can be changed by calling the `setDateFormat()` method on the `core` extension instance. The first argument is the default format for dates and the second one is the default format for date intervals:

```
<td>{{ lieu.createdAt|date("d/m/Y")}}</td>
```



```

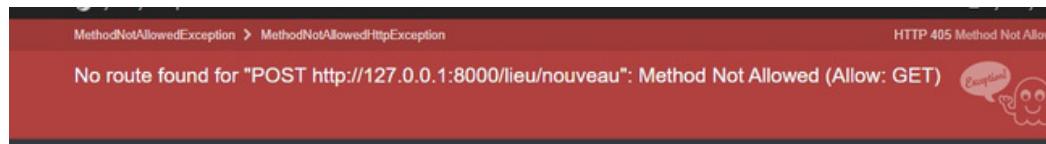
<?php

namespace App\Controller;

use AdminController;
use AddressController;
use ContactController;
use HomeController;
use LieuxController;
use MesLieuxController;
use SecurityController;
use UserController;
use DataFixtures;
use Entity;
use EntityListener;
use Form;

class MesLieuxController extends Controller
{
    /**
     * @param LieuRepository $repository
     * @param PaginatorInterface $paginator
     * @param Request $request
     * @return Response
     */
    #[Route('/meslieux', name: 'meslieux.index', methods:['GET'])]
    #[IsGranted('ROLE_USER')]
    // injection de dépendance -> injecter un service dans les paramètres de la fonction du contrôleur
    public function findByUser(LieuRepository $repository, PaginatorInterface $paginator, Request $request)
    {
        $lieux = $paginator->paginate(
            // display 'lieux' that are assigned to the current user
        );
    }
}

```



```

<?php

namespace App\Controller;

use AdminController;
use AddressController;
use ContactController;
use HomeController;
use LieuxController;
use MesLieuxController;
use SecurityController;
use UserController;
use DataFixtures;
use Entity;
use EntityListener;
use Form;

class MesLieuxController extends Controller
{
    /**
     * @param Request $request
     * @param EntityManagerInterface $manager
     * @return Response
     */
    #[Route('/lieu/nouveau', 'lieu.new', methods: ['GET', 'POST'])]
    #[IsGranted('ROLE_USER')]
    /* we create a Route we write it in French and the methods are GET
     * and POST
    */
    public function new(Request $request, EntityManagerInterface $manager)
    {
        /* we call the EntityManagerInterface, it will help to get into the
         * database
        */
    }
}

```



```

<?php

namespace App\Controller;

use AdminController;
use AddressController;
use ContactController;
use HomeController;
use LieuxController;
use MesLieuxController;
use SecurityController;
use UserController;
use DataFixtures;
use Entity;
use EntityListener;
use Form;
use Repository;
use Kernel;
use Twig\Environment;
use Twig\Error\LoaderError;
use Twig\Error\RuntimeError;
use Twig\Error\SyntaxError;

class MesLieuxController extends Controller
{
    /**
     * @param Environment $twig
     */
    public function index(Environment $twig)
    {
        $lieux = $this->getDoctrine()
            ->getRepository(Lieu::class)
            ->findAll();
        return $this->render('meslieux/index.html.twig', [
            'lieux' => $lieux,
        ]);
    }

    /**
     * @param Environment $twig
     */
    public function new(Environment $twig)
    {
        $lieu = new Lieu();
        $form = $this->createForm(LieuType::class, $lieu);
        return $this->render('meslieux/new.html.twig', [
            'form' => $form->createView(),
        ]);
    }

    /**
     * @param Environment $twig
     */
    public function edit(Environment $twig, Lieu $lieu)
    {
        $form = $this->createForm(LieuType::class, $lieu);
        return $this->render('meslieux/edit.html.twig', [
            'form' => $form->createView(),
        ]);
    }

    /**
     * @param Environment $twig
     */
    public function delete(Environment $twig, Lieu $lieu)
    {
        $em = $this->getDoctrine()
            ->getManager();
        $em->remove($lieu);
        $em->flush();
        return $this->redirectToRoute('meslieux.index');
    }
}

```