| TP guidé | Création d'un blog |

3x2h + 1h30 finition et debrief

Objectifs du TP

- Découvrir le principe de Framework et notamment le fonctionnement de Symfony
- Comprendre les commandes de bases et l'architecture MVC en PHP
- Créer sa première application en Symfony

Contexte

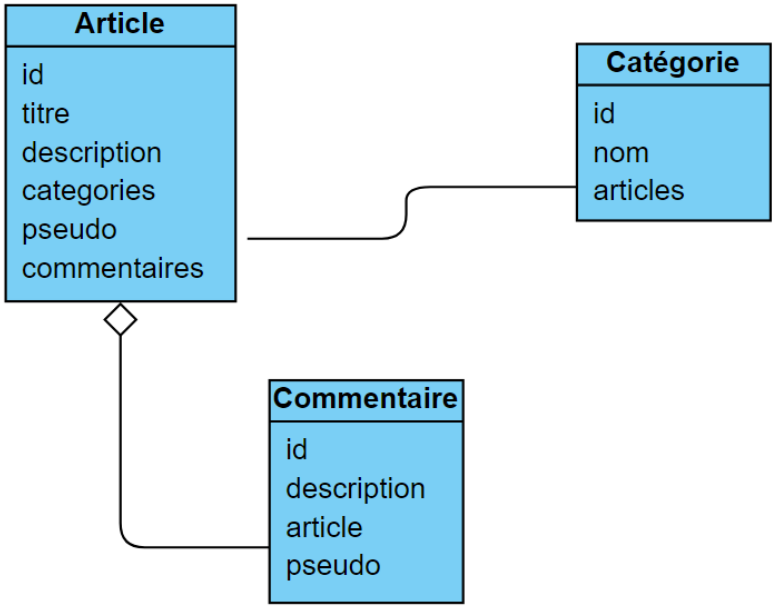
Pour ce premier TP, l'application web sera un simple projet de type « Blog » et qui sera composée des fonctionnalités suivantes :

- Une page qui liste les articles (page d'accueil)
- Une page qui affiche un article et ses commentaires (avec un formulaire pour ajouter un commentaire avec un pseudo)
- Une page de création d'article avec un pseudo et une ou plusieurs catégories (et les autres attributs)
- Un CRUD pour les catégories (avec un préfix « /admin »)

En supplément, vous pouvez ajouter :

- Un formulaire qui prend un pseudo et affiche l'ensemble de ses articles et commentaires
- Possibilité de supprimer un article ou un commentaire dans cette liste
- Ajouter un module pour utiliser des « slug » sur les articles
- Utiliser webpack et générer du CSS via NPM

Diagramme de classe rapide



1. Etape 0

Vérifier votre ordinateur en ayant bien :

- Node.js (pour utilisation de npm) (https://nodejs.org/en)
- Composer (https://getcomposer.org)
- PHP accessible en ligne de commande (test en faisant php -v dans un terminal, si KO mettre PHP dans vos variables d'environnements)

2. Etape 1

Création d'un dossier sur votre serveur via la commande :

composer create-project symfony/skeleton *iutBlog*

cd iutBlog

On installe ensuite le *kit* pour avoir l'ensemble des modules utiles pour une application web (à faire dans votre dossier du projet) :

composer require webapp

Suivant la version qui sera installée, vous devrez ajouter le package *Profiler* qui est très utile en dev :

composer require --dev symfony/profiler-pack

3. Git

Initier votre git à partir de là (en ayant créé votre répertoire dans votre forge)

Création d'un repo sur votre système de versioning (ici iutBlog)

git -c http.sslVerify=false clone https://forge.univ-lyon1.fr/alexis.monnet/iutblog.git
git remote add origin https://forge.univ-lyon1.fr/alexis.monnet/iutblog.git
git branch -M main
git –c http.sslVerify=false push -uf origin main
git add .
git commit –m « Initial commit »

4. (optionnel mais préférable) Création d'un fichier .htaccess

Ajouter un fichier .htaccess afin de « réécrire » les urls et éviter d'ajouter /index.php dans toutes vos urls :

- Dans le dossier /public dans votre projet
- Créer un fichier **.htaccess**
- Ajouter les lignes suivantes

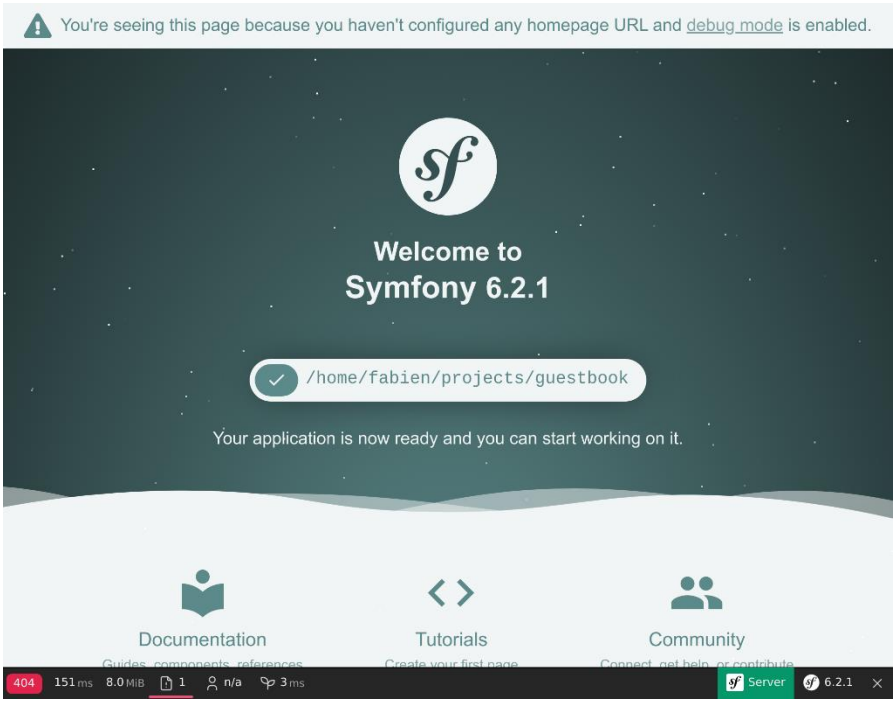
```
<IfModule mod_rewrite.c>
    Options -MultiViews
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php [QSA,L]
</IfModule>
```

- Si possible, créer un vhost sur votre ordinateur pour avoir une url de type (pour ceux et celles travaillant sur leur PC) : **http://tp-blog.local**

5. Configuration fichier *.env*

Avant de commencer à coder, nous allons configurer le fichier *.env* présent à la racine de votre projet pour connecter notre application web PHP à une base de données SQL : **DATABASE_URL** sera la variable à modifier (en reprenant le lien de connexion de PDO que vous avez utilisé précédemment ; enfin presque).

6. Votre projet Symfony est prêt même si aucun code n'a été encore codé. Essayez d'accéder à votre projet sur la route / ; Si votre configuration est bonne vous avez la page d'accueil par défaut de Symfony



7. Notre application est maintenant bien chargée. Le travail va pouvoir commencer !

COMPOSANT *MAKER*

Ce composant va être ultra utile pour la génération de plusieurs types de fichier comme les Controller, les commandes, les entités, les formulaires, les CRUD, etc…

8. Créons chaque entité suivant le schéma précédent grâce à la commande
   php bin/console make:entity
   La commande va vous guider sur les types des attributs de l'Entity, etc…
   / !\ Chaque entity se doit d'avoir une date de création et une date de modification (non présentes dans le schéma)

9. L'ensemble de nos entités sont maintenant créées. Nous allons les générer dans la base de données sans faire une ligne de SQL ! 2,5 commandes essentielles :

php bin/console doctrine:migrate:diff -> On regarde la différence entre notre code et la base de données et cela génère un fichier de *migration* dans le dossier *migrations* à la racine de votre projet

php bin/console doctrine:migrate:migrate ou **migrate:execute --up NUM_VERSION** -> On lance la ou les migrations en base de données (on exécute le SQL présent dans le fichier de migration généré précédemment)

10. Avant de créer nos controllers, services, etc… Nous allons ajouter un nouveau module pour gérer nos routes au niveau des controllers (ou utiliser les attributs PHP8 donc sans ce module) :

composer require annotations

Grâce à ce package, nous allons pouvoir gérer les routes via les controller que nous créerons tout à l'heure. Normalement, vous devez modifier le fichier /config/packages/sensio_framework_extra.yml et ajouter ces quelques lignes :

```yaml
sensio_framework_extra:
    router:
        annotations: true
```

/!\ les fichiers .yml sont sensibles aux indentations. Pensez-y bien.

11. Nos entités sont parfaites, notre base de données est à jour. Nous allons pouvoir générer le CRUD sur l'entity *Categorie*

php bin/console make:crud Categorie

Stupéfaction, nous avons maintenant un controller, un formulaire et des fichiers .html.twig qui ont été généré.

12. Nous allons pouvoir créer nos différents controllers, nos différentes vues et formulaires via le maker et en allant coder un peu tout ça. N'hésitez pas à explorer la commande *make* et ses possibilités.

Bonus

Pour gérer votre JS, CSS via du SCSS, vous devez ajouter le package *webpack* :

composer require symfony/webpack-encore-bundle

npm install (on installe les packages npm liés ; par défaut jquery, bootstrap sont installés)

3 commandes sont importantes à connaitre pour travailler dessus :

- **npm run dev** -> Génère les fichiers pour une utilisation en phase de développement
- **npm run watch** -> Génère les fichiers à chaque changement JS/SCSS
- **npm run build** -> Génère les fichiers pour une MEP