

TP “StudentHub” Symfony

L'objectif du tp:

L'objectif de notre TP est de concevoir une application Symfony spécialisée dans la gestion d'un répertoire étudiant, avec l'intégration d'un système d'authentification.

----- à titre informatif -----

La commande `docker-compose exec php bash` est utilisée pour exécuter une session bash (ou un autre shell) à l'intérieur d'un conteneur Docker.

Décomposition de cette commande pour une meilleure compréhension :

- `docker-compose exec`: Cela signifie qu'on utilise la fonction d'exécution de Docker Compose, qui nous permet d'exécuter des commandes dans un service spécifique défini dans notre fichier `docker-compose.yml`.
- `php`: C'est le nom du service dans lequel on souhaite exécuter la commande. Dans notre cas, c'est le service PHP défini dans notre fichier `docker-compose.yml`.
- `bash`: C'est la commande qu'on exécute à l'intérieur du conteneur. Dans ce cas, on lance un shell bash interactif à l'intérieur du conteneur PHP.

symfony console list ⇒ Cette commande vous permet de lister les commandes utiles qui vous permettront de créer votre application .

Partie 0 : Préparation de l'environnement de travail

Veillez suivre les étapes ci-dessous :

1. Clonez le dépôt suivant : <https://github.com/JOEEIHAJJ/symfony.git>
2. Ouvrez le projet dans votre environnement de développement intégré (par exemple, VSCode, IntelliJ, etc.).
3. Pour Lancer Docker dans les machines du M5 : `systemctl --user start docker`
4. Accédez au terminal et naviguez jusqu'au dossier ".docker".
5. Lancez la commande *`docker-compose up`*
6. Maintenez ce terminal ouvert et ouvrez un nouveau terminal à côté pour les prochaines étapes.
7. Si vous accédez au fichier ".docker/.env", vous y trouverez les informations de connexion à la base de données, qui a été créée automatiquement avec la commande "docker-compose ;)".
8. Vous pouvez vous connecter à la base de données et la visualiser à l'aide de DBeaver (ou de votre IDE).
9. Dans le 2ème terminal, il faut lancer la commande en dessous :

`docker compose exec php bash`

10. Lancez la commande : *`composer install`*

IMPORTANT 🚨 : Toutes les commandes qui suivront doivent être exécutées dans ce même bash.

11. Testez *`http://localhost:1027`*
12. Lancez les 2 commandes suivantes à la suite dans le bash:

⚠️ `curl -sS https://get.symfony.com/cli/installer | bash`

⚠️ `export PATH="$HOME/.symfony5/bin:$PATH"`

⚠️ `composer require twig`

`Vous êtes prêts à commencer à créer votre premier projet Symfony !`

Partie 1 : HELLO WORLD !

L'objectif de cette partie est donc de créer une première page qui affiche Hello world.

Pour cela, nous allons créer un controller HelloController .

- Trouvez la commande qui permet de créer un controller à l'aide de

symfony console list

Une fois le controller créé, vous devriez avoir le contenu ci-dessous dans votre terminal:

```
created: src/Controller/HelloController.php
created: templates/hello/index.html.twig

Success!

Next: Open your new controller class and add some pages!
```

On remarque qu'avec la simple création du HelloController, une méthode index est créée avec une route associée. Cette méthode nous renvoie vers le template créé automatiquement également.

- Visualisez le contenu des fichiers créés.

 **NB:** Tous les templates que nous créerons durant le tp devront étendre base.html.twig.

- Vous pouvez désormais visualiser votre première page en tapant le lien:

<http://localhost:1027/hello>

Partie 2 : Installation d'une certification SSL sur Symfony

Trouver la commande qui permet d'installer le certificat auto-signé nécessaire pour activer le protocole HTTPS.

Une fois la commande exécutée vous remarquerez que notre application est maintenant sécurisée par un cadenas, indiquant le passage du protocole HTTP à HTTPS.

⚠ NB: sur les machines du M5 cette option est restreinte .

Partie 3 : Première entité et CRUD

1. Création de l'entité Etudiant

créez une entité "Etudiant" dotée des propriétés suivantes : un nom, un prénom, et un numéro INE . Faites en sorte que ces propriétés ne soient jamais nulles.

La création de cette entité a également engendré un répertoire "repository" contenant une classe "EtudiantRepository". Analysez rapidement cette classe.

1.1 Migrations :

Créez les tables dans la base de données en utilisant les bonnes commandes , Vérifiez que votre table "etudiant" est présente avec ses propriétés.

Partie 4 : Installation du dashboard EasyAdmin

Lancez les commandes vu en cours pour :

⚠ installer *EasyAdminBundle*

⚠ créer un tableau de bord d'administration

⚠ créer des opérations CRUD

Analyser le code généré dans `controller/Admin/DashboardController` puis tester la route

<https://localhost:1027/admin>

Dans `Controller/Admin/DashboardController` Ajoutez les bouts de code manquants, comme expliqué dans les **commentaires bleus** `TODO` ;)

```
//TODO : Ajouter les imports nécessaire

class DashboardController extends AbstractDashboardController
{
    //Ajouter ce constructeur ;)
    public function __construct(
        private AdminUrlGenerator $adminUrlGenerator
    ) {}

    #[Route('/admin', name: 'admin')]
    public function index(): Response
    {
        // TODO: Générer l'URL pour rediriger vers EtudiantCrudController

    }

    public function configureDashboard(): Dashboard
    {
        return Dashboard::new()
            ->setTitle('StudentHubTP');
```

```

    }

    public function configureMenuItems(): iterable
    {
        yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');

        // TODO: Ajouter des liens vers le tableau de bord et les entités
        Etudiant et Note

    }
}

```

Partie 5 : Installation du bundle de la sécurité (authentification + création de compte)

⚠ composer require security

Lancez :

⚠ symfony console make:user


Cette commande crée une classe utilisateur de base avec des fonctionnalités prêtes à l'emploi, comme la gestion des e-mails observez ce qu'elle génère .

Puis Lancez :

⚠ symfony console make:auth

Cette commande facilite la mise en place d'un système d'authentification en générant automatiquement des fichiers tels que des contrôleurs et des templates, tout en offrant des options de personnalisation.(Nommez la classe : *AppAuthenticator*)

Testez la route */login*

 symfony console make:registration

NB : Nous n'allons pas gérer l'envoi de mail de confirmation pour ce TP.

Cette commande génère automatiquement les fichiers et le code nécessaires pour mettre en place un système d'inscription dans une application Symfony

NB : N'oubliez pas de relancer les commandes de migration en BDD pour mettre à jour votre table User.

Testez la route */register*

Dans `|src|Security|AppAuthenticator.php`

```
    public function onAuthenticationSuccess(Request $request, TokenInterface
$token, string $firewallName): ?Response
    {
        if ($targetPath = $this->getTargetPath($request->getSession(),
$firewallName)) {
            return new RedirectResponse($targetPath);
        }

        //Todo rediriger vers le dashboard
        throw new \Exception('TODO: provide a valid redirect inside
' . __FILE__);
    }

    protected function getLoginUrl(Request $request): string
    {
        return $this->urlGenerator->generate(self::LOGIN_ROUTE);
    }
}
```

Partie 6 : Pour aller plus loin (les Relations entre les tables)

Reproduisez les étapes précédentes pour instaurer une nouvelle entité nommée 'Note', dotée des propriétés suivantes ainsi qu'une relation ManyToOne avec l'entité 'Étudiant':

- note (type float)
- remarque (type string)
- etudiant (relation)

Cela implique d'exécuter les commandes nécessaires pour générer l'entité 'Note', définir ses propriétés, et établir une relation ManyToOne avec l'entité 'Étudiant', en accord avec les démarches préalablement entreprises.