Symfony

Nouveau projet

- `composer create-project symfony/website-skeleton my_project_name` créé un projet complet via composer
- `composer create-project symfony/skeleton my_project_name` créé un projet minimal via composer
- `symfony new nameOfTheProjectHere --full` créé un projet complet avec l'exécutable Symfony
- `symfony new nameOfTheProjectHere` créé un projet minimal avec l'exécutable Symfony

Problèmes

- Faire un `dd($maVariable)` pour débugger (dump and die)
- 404 alors que les routes sont bien annotées sur un serveur Apache
  - Installer le `symfony/apache-pack`
- Erreur en prod et pas en dev
  - Vider le cache
  - Changer `$kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);` en `$kernel = new Kernel($_SERVER['APP_ENV'], true);` dans `index.php` pour identifier l'erreur
- On peut voir le moment où :
  - des infos sont stockés dans la session dans `Symfony\Component\HttpFoundation\Session\Attribute\AttributeBag::set()`
  - les classes sont chargées dans `Symfony\Component\ClassLoader\ClassLoader::loadClass()`

CLI

- `php bin/console debug:router` afficher toutes les routes
- `php bin/console debug:container --show-private` afficher tous les services disponibles dans le conteneur de services
- `php bin/console debug:autowiring` afficher tous les services disponibles avec autowiring
- `php bin/console debug:event-dispatcher nomEvenement` avec `nomEvenement` en option (exemple : `kernel.controller`), afficher tous les listeners avec leur priorité
- Créer un système de connexion
  - `php bin/console make:auth`
  - `php bin/console make:user`
  - `php bin/console make:voter`

Injecter des paramètres manuellement via la config et le fichier .env

- https://symfony.com/doc/current/service_container.html#manually-wiring-arguments
- https://symfony.com/doc/current/configuration.html#configuration-based-on-environment-variables

Exemple sur le projet 7 de la formation OpenClassrooms :

Fichier `services.yaml`

```
App\EventListener\HttpCacheListener:
```

```
        arguments:
            $cacheExpiration: '%env(CACHE_EXPIRATION)%'
```

Fichier `.env`

```
CACHE_EXPIRATION='+10 minutes'
```

Récupérer un utilisateur

- Dans un controller héritant de AbstractController
    - `$this->getUser()`
- Via un objet Security
    - `$security->getUser()`
- Via un objet implémentant TokenInterface
    - `$token->getUser()`

Configurer son serveur

Documentation

Le dossier `public/` est la racine. Le virtual host doit donc pointer vers ce dossier.

Apache

Installer *symfony/apache-pack* en exécutant `composer require symfony/apache-pack`

> Installe au passage un `.htaccess` dans le dossier `public\`

Utiliser Webpack Encore

Documentation

Prérequis :

- Installer yarn

Installation

Exécuter `composer require encore` puis `yarn install`

Utilisation

Exécuter `yarn encore dev` pour compiler les *assets* (fichiers situés dans le dossier `assets/` situé à la racine du projet).

Exécuter `yarn encore dev --watch` pour recompiler automatiquement les fichiers si on fait des modifications.

Configuration

Voir dans le fichier `webpack.config.js` situé à la racine du projet.

> Si vous avez exécuté `yarn encore dev --watch` alors il faudra l'arrêter et le relancer après vos modifications.

Modifier PHP

- Dans `bin/console` :

```
// On modifie des parties de php.ini
ini_set('date.timezone', 'UTC');
ini_set('display_errors', (int) $debug);
ini_set('memory_limit', -1);

if ($debug) {
    Debug::enable(E_ERROR); // On change la sensibilité des erreurs, warnings...
}
```

Utiliser le serveur de développement

- Lancer le serveur avec la commande `symfony server:start`
  - On peut ajouter `-d` pour faire tourner le serveur en arrière plan
- Activer https avec la commande `symfony server:ca:install`
- Voir les logs avec la commande `symfony server:log`
- Utiliser une version particulière de php en créant un fichier `.phpversion` contenant le numéro de version
  - `echo "7.2" > .php-version`
- Créer un fichier `php.ini` personnalisé à la racine du projet pour modifier la configuration de php du serveur
- Le fichier `php.ini` utilisé dans mon cas est situé dans `c/wamp64/bin/php/php7.3.5/`

Différences Symfony 3 et 4

- Choix de l'environnement prod ou dev
  - sf4 : remplir `APP_ENV=prod ou dev` dans le fichier `.env.local`
  - sf3 : ajouter à l'URL `/app.php` pour le mode prod ou `/app_dev.php` pour le mode dev
- Aborescence
  - sf4 :
    - `config/` contient les fichiers de configuration
    - `src/` contient les services, controleurs, entités, repository...
    - `public/` contient `index.php`, `css/`, `js/`
    - `templates/` contient les fichiers twig
  - sf3 :
    - `app/` contient :
      - les fichiers de configuration dans `app/config/`
      - les ficheirs twig dans `app/Resources/views/`
    - `src/` contient le dossier `AppBundle` contenant les controleurs, entités, repository...
    - `web/` contient `index.php`, `css/`, `js/`

Evénements lancés par le kernel

1. kernel.request
2. kernel.controller
3. kernel.controller_arguments
4. kernel.view
5. kernel.response
6. kernel.finish_request
7. kernel.terminate
8. kernel.exception