Symfony 4 :
Les commandes essentielles
==========

🟡 **NOUVEAU PROJET**

1) Avec Composer

Création d'un projet à partir d'une architecture complète

```
composer create-project symfony/website-skeleton
<project-name>
```

Création d'un projet à partir d'une architecture « microservice »

```
composer create-project symfony/skeleton <project-name>
```

Préciser la version de symfony
e.g.: la version 4.4

```
composer create-project symfony/website-skeleton
<project-name> "4.4.*"
composer create-project symfony/skeleton <project-name>
"4.4.*"
```

Liste des composants installés

```
composer show
```

2) Avec l'utilitaire Symfony

2.A) Installation

Installation de l'utilitaire Symfony (Linux)

```
wget https://get.symfony.com/cli/installer -O - | bash
```

Installation de l'utilitaire Symfony (Mac)

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

Installation de l'utilitaire Symfony (Windows)

Téléchargement de Symfony : https://get.symfony.com/cli/setup.exe

2.B) Création du projet

Création d'un projet à partir d'une architecture complète

```
symfony new <project-name> --full
```

Création d'un projet à partir d'une architecture « microservice »

```
symfony new <project-name>
```

3) Préparer le projet pour GIT

3.A) Préparation du fichier d'environnement

Créer le fichier d'environnement (Mac / Linux) :

```
cp .env .env.dist
```

Créer le fichier d'environnement (Windows) :

```
copy .env .env.dist
```

/!\ effacer les données sensible du fichier .env.dist

3.B) Initialisation de GIT

Protéger les fichier sensible :
> Ajouter au fichier .gitignore

```
.env
```

Initialiser GIT :

```
git init
git flow init
```

INSTALLER UN PROJET

1) Cloner le projet

Cloner le projet :

```
git clone <repository-url>
```

2) Définir l'environnement

Créer le fichier d'environnement (Mac / Linux) :

```
cp .env.dist .env
```

Créer le fichier d'environnement (Windows) :

```
copy .env.dist .env
```

3) Installer le projet

Créer le fichier d'environnement :

```
cp .env.dist .env
```

Installation des dépendances :

```
composer install
npm install
```

Installation des dépendances avec modification de la limite de mémoire :

```
COMPOSER_MEMORY_LIMIT=-1 composer install
```

Vider le cache de composer :

```
----------------------------
composer clear-cache
```

SERVEUR WEB INTERNE

Démarrer le serveur :

```
----------------------------
php -S localhost:8020 -t public
```

CONTROLLER

Démarrer le serveur :

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

BASE DE DONNEES / DOCTRINE

1) Configuration

Configurer la connexion, dans le fichier **.env** :

```
----------------------------
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/
db_name

Erreur : Metadata storage is not up to date...
Solution : DATABASE_URL=mysql://
db_user:db_password@127.0.0.1:3306/db_name?
serverVersion=mariadb-10.4.11
```

Multiple BDD :
---
https://symfony.com/doc/current/doctrine/multiple_entity_managers.html

2) Création

Créer une base de données :

```
php bin/console doctrine:database:create
```

3) Migration

Créer une migration :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

4) Reset

Reset de la base de données :

```
php bin/console doctrine:database:drop --force
rm -r src/Migrations
php bin/console doctrine:database:create
php bin/console make:migration
php bin/console doctrine:migrations:migrate
y
php bin/console doctrine:fixtures:load
yes
```

/!\ pour les propriétés « json » des entités passer en version 5.6 dans la phrase DATABASE_URL

LES ASSETS avec WebPack

1) Pré-requis

1.A) NodeJS

https://nodejs.org/

Vous avez besoin de NodeJS pour installer Yarn.
Ceci installera également NPM (normalement...)

Tester node est NPM :

```
----------------------------
node -v
npm -v
```

1.B) Yarn

https://yarnpkg.com/

Installation de l'utilitaire **Yarn** (Mac):

```
----------------------------
brew install yarn
```

2) Installation

Installation de la dépendance :

```
----------------------------
composer require encore
yarn install

yarn add sass-loader@^8.0.0 node-sass --dev
```

3) Configuration

3.A) Ajouter une application pour WebPack

Une application pour WebPack gère les scripts et les feuilles de styles.

Création de l'application « app.js »

L’application « app.js » va importer les dépendances JavaScript et CSS

Créer le fichier « **/assets/js/app.js** »

Ajouter l’application à la configuration **WebPack** « /webpack.config.js »:

```
.addEntry('app', './assets/js/app.js')
```

3.B) Importer les dépendances JavaScript

Importer des scripts JS dans l’application « **app.js** » pour le gestionnaire **WebPack** :

```
const $ = require('jquery');
require('bootstrap');
```

Importer l’application de scripts « **app.js** » dans la vue **Twig** :

```
{% block javascripts %}
    {{ encore_entry_script_tags('app') }}
{% endblock %}
```

3.C) Importer les dépendances CSS

Importer des feuilles de styles (css, sass, less) dans l’application « **app.js** » pour le gestionnaire **WebPack** :

```
require('../css/app.css');
require('../../node_modules/bootstrap/dist/css/
bootstrap.min.css');
```

Importer l’application de styles « **app.js** » dans la vue **Twig** :

```
{% block stylesheets %}
    {{ encore_entry_link_tags('app') }}
{% endblock %}
```

4) Compilation des Assets

Compilation des Assets (dev) :

```
----------------------------
yarn encore dev
yarn encore dev --watch
```

Compilation des Assets (prod) :

```
----------------------------
yarn encore production
```

LES ENTITES

1) Création & Modification

1.A) Créer ou Modifier

Créer une nouvelle entité ou Modifier une entité existante :

```
----------------------------
php bin/console make:entity
php bin/console make:entity Books
```

1.B) Créer une entité depuis une table de BDD

Générer les entités depuis une table existante:

```
----------------------------
php bin/console doctrine:mapping:import App\\Entity
annotation --path=src/Entity
```

2) Générer les Accesseurs (Getter / Setter)

Générer les accepteurs du namespace « App » :

```
----------------------------
php bin/console make:entity --regenerate App
```

3) Gestion des contraintes et validations

https://symfony.com/doc/current/validation.html

Installation de la dépendance :

```
composer require symfony/validator doctrine/annotations
```

Validation (Assert)

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

/**
 * @Assert\NotBlank
 */
```

LES CONTROLEURS

Créer un contrôleur :

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

LES FIXTUES

1) Installation

Installation de la dépendance :

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

2) Chargement des mixtures

Ajouter les fixtures :

```
php bin/console doctrine:fixtures:load
```

Ajouter les fixtures, en conservant les données de la basse :

```
----------------------------
php bin/console doctrine:fixtures:load --append
```

LES ROUTES

Affiche un tableau avec les informations de toutes les routes de l'application :

```
----------------------------
php bin/console debug:router
```

Affiche les informations d'une route donnée

```
----------------------------
php bin/console debug:router route_name
```

LES EVENEMENTS

1) kernel events

- kernel.request
- kernel.controller
- kernel.controller_arguments
- kernel.view
- kernel.response
- kernel.finish_request
- kernel.terminate
- kernel.exception

LES COMMANDES

Générer une commande :

```
----------------------------
php bin/console make:command <app:cmd>
```

LES TEST

xxx :

```
----------------------------
php bin/console make:command <app:cmd>
```

https://blog.dev-web.io/2018/01/04/symfony-4-gestion-des-utilisateurs-sans-fosuserbundle-chapitre-3-5/

MISE EN PRODUCTION

Charger le projet sur le serveur de prod :

```
----------------------------
git clone <http://depot.com>
composer install
yarn install
yarn encore production
```

Pousser les mises à jour :

```
----------------------------
git pull
php bin/console cache:clear
```

DEPENDANCES UTILES

Flex

```
----------------------------
composer require symfony/flex
```

Web Server

```
----------------------------
composer require symfony/web-server-bundle --dev
```

Maker

```
----------------------------
composer require symfony/maker-bundle --dev
```

Doctrine

```
----------------------------
composer require symfony/orm-pack
```

Annotations

```
----------------------------
composer require annotations
```

Form

```
----------------------------
composer require form
```

Form Validator

```
----------------------------
composer require validator
```

Twig

```
----------------------------
composer require twig-bundle
```

Security

```
----------------------------
composer require security-csrf
```

Swift Mailler

```
----------------------------
composer require symfony/swiftmailer-bundle
```

Serializer

```
----------------------------
composer require symfony/serializer
composer require symfony/serializer-pack
```

Xxxxx

```
----------------------------
composer require symfony/property-access
```

Security

```
----------------------------
composer require security
```

jSon Web Token

```
----------------------------
```

```
composer require lexik/jwt-authentication-bundle
```

CORS

```
-----------------------------
composer require nelmio/cors-bundle
```

Xxxxx

```
-----------------------------
composer require claviska/simpleimage
composer require cocur/slugify
composer require ramsey/uuid-doctrine
composer require symfony/translation
composer require vich/uploader-bundle


composer require doctrine/doctrine-fixtures-bundle --dev
composer require profiler --dev
```