I. Développement front-end avec WEBPACK ENCORE

Permet de gérer (et générer) les fichiers ressources utilisées pour le front (js, css). Les fichiers seront compilés et placés dans le dossier `public/build`. Dans la suite, tu vas savoir comment ajouter les technologies suivantes à ton projet : jQuery, Bootstrap, FontAwesome.

```
C:\> composer require encore
```

II. Gestionnaire de dépendances : YARN

Pour utiliser webpack, il va falloir installer **yarn** sur ta machine. Tu trouveras l'installateur pour Windows ici. Tu pourras aussi trouver les commandes pour l'installer sur un Mac sur le site officiel de Yarn. Quoiqu'il arrive, il faut que ***npm*** (= ***nodejs***) soit auparavant installé.

Une fois **yarn** installé sur la machine, il va permettre d'installer les dépendances nécessaires à la génération des fichiers ressources.

1. Préparer l'utilisation de yarn dans le projet Symfony :

```
C:\> yarn install
```

2. Pour compiler le Sass (requis pour les frameworks développé en Sass comme **Bootstrap**) :

```
C:\> yarn add node-sass
```

```
C:\> yarn add sass-loader
```

3. Pour utiliser la police d'icônes FontAwesome :

```
C:\> yarn add @fortawesome/fontawesome-free
```

4. Pour utiliser jQuery, Popper.js[1] :

```
C:\> yarn add jquery
C:\> yarn add popper.js
```

5. Pour utiliser Bootstrap (framework CSS) :

```
C:\> yarn add bootstrap
```

III. Configurer la compilation des fichiers ressources

Pour utiliser ces bibliothèques js/css, il va falloir modifier certains fichiers :

IV. webpack.config.js

V. Décommenter la ligne 69 pour activer la compilation du SASS

```
// enables Sass/SCSS support
.enableSassLoader()
```

VI. Décommenter la ligne 69 pour éviter un bug lors de l'utilisation de jQuery

```
// uncomment if you're having problems with a jQuery plugin
.autoProvidejQuery()
```

VII. Pour ajouter dans le dossier `public/images` des fichiers images placées dans le dossier `assets/images` :

```
.copyFiles({
    from: './assets/images',

    // optional target path, relative to the output dir
    to: '../images/[path][name].[ext]',

    /*
    // if versioning is enabled, add the file hash too
    //to: 'images/[path][name].[hash:8].[ext]',

    // only copy files matching this pattern
    //pattern: /\.(png|jpg|jpeg)$/
```

[1] Bibliothèque JavaScript requise par Bootstrap. On peut n'installer que jQuery si Bootstrap n'est pas utilisé.

```
    */
})
```

Remarque le [path]… Cela signigie que l'arborescence du dossier `assets/images` sera respectée dans le dossier `public/images`.

Bien entendu, rien ne t'empêches de renommer les dossiers soulignés dans l'exemple ou de décommenter les autres options.

`assets/styles/app.css`

a. Modifie l'extension du fichier par `.scss` pour pouvoir écrire du SASS dans ce fichier.

b. Pour utiliser FontAwesome, ajoute au début du fichier :

```
@import "~@fortawesome/fontawesome-free/css/all.min.css";
```

c. Pour utiliser Bootstrap, ajoute :

```
@import "~bootstrap/scss/bootstrap";
```

d. S'il tu veux redéclarer les variables SASS, il faut le faire avant d'importer le framework SASS. Par exemple, pour modifier la valeur *primary* de Bootstrap

```
$primary: #5887ff;
```

`assets/app.js`

a. Modifie *app.css* par *app.scss* (seulement si tu as dû renommer ce fichier, bien entendu).

b. Pour jQuery, ajouter :

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

c. Pour éviter les risques d'erreurs en utilisant jQuery, ajouter :

```
// ⚠ fixe le problème d'utilisation tardive de jQuery
global.$ = global.jQuery = $;
```

d. Pour utiliser le js de Bootstrap, ajoute :

```
require('bootstrap');
```

Les fichiers templates

Pour créer les liens vers les fichiers css et js créés, il faut ajouter dans un template les fonctions twig qui vont générer les balises correspondantes (link, script) :

- Lien pour les fichiers CSS :

```
{{ encore_entry_link_tags("app") }}
```

- Lien pour les fichiers JS :

```
{{ encore_entry_script_tags("app") }}
```

Générer les fichiers ressources :

Les fichiers à modifier se trouvent dans le dossier `assets`. Dès qu'un de ces fichiers est modifié, il faut lancer la commande pour compiler et générer les fichiers :

```
yarn encore dev
```

↳ La dernière partie de la commande correspond à l'environnement sur lequel les fichiers vont être compilés. Sur un serveur de production, remplacer **dev** par **prod**.

Pour éviter de lancer continuellement cette commande lors du développement frontend, il faut lancer la commande :

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

↳ Tant que le processus sera actif, dès qu'une modification sera enregistrée, les fichiers ressources seront générés.

Avec Symfony CLI, on peut lancer cette commande en tâche de fond :

```
symfony run -d yarn encore dev --watch
```

Mettre à jour une dépendance

Ponctuellement, il faudra peut-être mettre à jour une dépendance installée avec yarn. La commande suivante permettra d'installer la version désirée :

```
yarn upgrade dependance@^1.x
```

↳ Remplacer ***dependance*** par le nom de la dépendance à actualiser

↳ Remplacer ***1.x*** par la version minimum à installer