Symfony UX avec webpack-encore-bundle

Depuis sa version 6, symfony a décidé de s'intéresser un peu plus au front, ils ont alors décidé de mettre en place "**webpack-encore-bundle**".

Celui ci utilise "**yarn**" ou "**npm**" donc faites bien attention d'avoir installé un de ces gestionnaires.
Par la suite j'utiliserais npm. Passons maintenant à l'installation de notre bundle :

```
composer require symfony/webpack-encore-bundle
npm install
```

Après avoir installé Encore, il nous aura créé entre autre chose un fichier "**package.json**" qui contient toute une liste de paquet dont il aura besoin, lançons la seconde commande pour installer tous ce JS.

Nous avons maintenant en plus de notre gros dossier "**vendor**", un gros dossier "**node_modules**" mais aussi un dossier "**assets**".

Jusqu'à maintenant on devais placer notre CSS et JS dans notre dossier "**public**" mais maintenant ils iront dans le dossier "**assets**". En effet un des rôle de "**Encore**" est de regrouper les scripts, css et certaines images au même endroit puis faire un build pour avoir une version minimisé en public.

On trouvera aussi un fichier "**webpack.config.js**", on y trouvera beaucoup de choses différentes mais ce qui va nous intéresser c'est la partie suivante :

```
Encore
    .setOutputPath('public/build/')
    .setPublicPath('/build')
    .addEntry('app', './assets/app.js')
```

- setOutputPath() indique dans quel dossier le "Encore" créera les builds finaux.
- setPublicPath() permet de paramétrer le chemin qui devra être utilisé pour atteindre les fichiers.
- addEntry() permet de nommer les build et d'indiquer à quel fichier ils font référence.

On pourra ajouter d'autres entrées si le besoin s'en fait sentir. Par exemple on utilisera celle par défaut pour notre JS et CSS principal, mais si certaines pages ont besoin de leur propre fichier, on créera une autre entrée.

D'ailleurs en parlant de ces entrées, regardons un instant les lignes suivantes dans le fichier "**base.html.twig**" :

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

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

Ces deux lignes étaient déjà présente mais ne servent qu'à cette extension, elles attendait l'installation de celle ci. En parlant de cela, depuis qu'on a installé l'extension, si on actualise, notre site nous renvoi une erreur. Effectivement, l'extension tente d'atteindre les builds que promet ces deux lignes mais ils n'existent pas. Voyons les commandes que nous apporte cette extension :

```
npm run dev
npm run watch
npm run build
```

La première commande permet de faire un build non définitif mais rapide.
La seconde fait de même mais en watch, prenant en compte tous les changements. La dernière fait un build plus long à produire mais prêt pour la prod.

Passons maintenant au CSS et JS eux même dans le dossier "**assets**". Le dossier "controllers" et les fichiers bootstrap.js et controllers.json sont lié à une bibliothèque appelé "Stimulus". Si vous vous intéressez à Symfony UX, je vous conseille d'aller lire la documentation, ici nous ne le verrons pas.

Ensuite nous avons un dossier "**styles**" qui contient un fichier css. celui ci est là à titre d'exemple pour voir comment fonctionne Encore.

Enfin voyons app.js (si vous vous souvenez, il c'est lui qui est appelé en tant qu'entrée dans le fichier de config de Encore).

Ce fichier n'a pour rôle que d'importer les fichiers JS et CSS de notre Projet. Actuellement on y trouve que deux lignes :

```
import './styles/app.css';
import './bootstrap';
```

Le fichier css que l'on a vu précédemment et le fichier bootstrap de Stymulus.

Si vous créez vos propres script et ou fichiers css, vous n'avez qu'à les importer ici et ils seront automatiquement placé dans votre "head" lors du prochain build.

Par exemple retirons notre CDN de bootstrap et installons le via npm maintenant qu'on l'utilise pour d'autres choses.

```
npm install bootstrap
```

Il ne me reste plus qu'à l'importer dans notre fichier **app.js**;

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Cela fonctionne de même pour n'importe quel bibliothèques ou pour mes fichiers JS ou CSS personnel.

Optionnellement, je peux venir activer certaines options en bas du fichier "**webpack.config.js**"

```
.enableSassLoader()
.enableTypeScriptLoader()
```

Permettent d'écrire et d'importer directement des fichiers sass/scss ou typescript sans avoir besoin de les compiler puisque Encore s'en occupera lui même lors du build.

Conclusion

Symfony est immense et possède encore de nombreux outils et options que nous n'avons pas vu ensemble, je ne les connais pas toute non plus cela dit. Mais vous avez maintenant les bases pour le pousser plus loin ou l'abandonner selon l'envie.