Introducción

En Symfony 4 se recomienda utilizar WebPack Encore como gestor de assets en lugar de Assetic.

Instalación

Para utilizar Encore, debemos tener instalado Node.js

Encore lo podemos instalar con yarn o con npm

> *yarn add @symfony/webpack-encore --dev*

> *npm install @symfony/webpack-encore --save-dev*

Para inicializar Encore en nuestro proyecto, utilizaremos Flex

> *composer require encore*
> *yarn install*

Esto creará un fichero webpack.config.js y el directorio assets/ y añadirá el directorio node_modules/ al fichero .gitignore.

Ejemplo de configuración

Vamos a imaginar que tenemos un proyecto sencillo con un archivo css y un archivo js que hemos puesto en el directorio assets/

- assets/js/app.js
- assets/css/app.css

Configurar Encore/Wepback

Tenemos que crear un nuevo fichero llamado webpack.config.js en el raíz de nuestro proyecto. Dentro del fichero utilizaremos Encore para ayudar a Webpack a generar la configuración.

```
// webpack.config.js
var Encore = require('@symfony/webpack-encore');

Encore
    // El directorio del proyecto en donde se compliarán y
 guardarán los assets
    .setOutputPath('public/build/')

    // El path público para acceder al directorio anterior
    .setPublicPath('/build')

    // creará ficheros app.css y app.js en public/build/
    .addEntry('js/app', './assets/js/app.js')
```

```
    .addStyleEntry('css/app', './assets/css/app.css')

    // permite a aplicaciones legacy utilizar $/jQuery com
o una variable global
    .autoProvidejQuery()

    // habilita los source maps durante el desarrollo
    .enableSourceMaps(!Encore.isProduction())

    // vacía el directorio del outputPath antes de cada bu
ild
    .cleanupOutputBeforeBuild()

    // muestra notificaciones al hacer el build
    .enableBuildNotifications()

    // crea archivos con hash (por ejemplo: app.abc123.css
)
    // .enableVersioning()

    // permite procesar archivos sass/scss
    // .enableSassLoader()
;

// exporta la configuración
module.exports = Encore.getWebpackConfig();
```

Para construir los assets, se utiliza el comando de encore:

```
# compila los assets
./node_modules/.bin/encore dev

# compila los assets automáticamente cada vez que modifica
mos algún fichero
./node_modules/.bin/encore dev --watch

# compila los assets, pero también los minifica y los opti
miza
./node_modules/.bin/encore production

# Estos comandos son atajos de los comandos anteriores
yarn run encore dev
yarn run encore dev --watch
yarn run encore production
```

Si cambiamos la configuración de webpack, hay que volver a compilar los assets.

Después de haber ejecutado el comando de Encore, podemos enlazar los assets en nuestros twigs con el helper **asset()**.

```
{# base.html.twig #}
<!DOCTYPE html>
<html>
```

```
    <head>
        <!-- ... -->
        <link rel="stylesheet" href="{{ asset('build/app.c
ss') }}">
    </head>
    <body>
        <!-- ... -->
        <script src="{{ asset('build/app.js') }}"></script
>
    </body>
</html>
```

Archivos Sass y Less

Sass

En vez de utilizar archivos css, podemos utilizar Sass. Los cambios serían mínimos.

Lo primero sería instalar las dependencias:

> *yarn add --dev sass-loader node-sass*

y en la configuración de webpack:

```
// webpack.config.js
Encore
    // ...

    // allow sass/scss files to be processed
-    // .enableSassLoader()
+    .enableSassLoader()
```

LESS

Con Less el procedimiento es el mismo. Las dependencias se instalarían así:

> *yarn add --dev less-loader less*

Y para habilitarlo:

```
// webpack.config.js
// ...

Encore
    // ...
    .enableLessLoader()
;
```

Habilitar Source Maps

Los Source maps permiten a los navegadores acceder al código original de un asset (por ejemplo, al código Sass que fue compilado a css o al código TypeScript que fue compilado a JavaScript). Son útiles para depuración durante el desarrollo, pero son innecesarios cuando la aplicación está en producción.

Encore incluye source maps en los assets compilados solamente en el entorno de desarrollo, pero podemos controlar dicho comportamiento con la función **enableSourceMaps()**:

```
// webpack.config.js
// ...

Encore
    // ...

    // this is the default behavior...
    .enableSourceMaps(!Encore.isProduction())
    // ... but you can override it by passing a boolean va
lue
    .enableSourceMaps(true)
;
```

Habilitar TypeScript (ts-loader)

Primero, como siempre, instalar las dependencias:

> *yarn add --dev typescript ts-loader*

Y segundo, activar el loader correspondiente en webpack.config.js:

```
// webpack.config.js
// ...

Encore
    // ...
    .addEntry('main', './assets/main.ts')

    .enableTypeScriptLoader()
;
```

Se pueden configurar las opciones del ts-loader con un callback.

```
.enableTypeScriptLoader(function (typeScriptConfigOptions)
 {
    typeScriptConfigOptions.transpileOnly = true;
    typeScriptConfigOptions.configFileName = '/path/to/tsc
onfig.json';
});
```

Versionado de assets

Para evitar la cache de assets de los navegadores, se pueden compilar los assets con versionado llamando a **enableVersioning()**.

Cada fichero incluirá un hash en su nombre con lo que cuando el fichero se modifique el hash será distinto y el nombre del asset también.

```
// webpack.config.js
// ...

Encore
    .setOutputPath('public/build/')
    // ...
     .enableVersioning()
```

Para enlazar estos assets, Encore se crea un archivo manifest.json que mapea los nombres de los assets

```
{
    "build/app.js": "/build/app.123abc.js",
    "build/dashboard.css": "/build/dashboard.a4bf2d.css"
}
```

En nuestra aplicación Symfony habrá que activar la estrategia de versionado de assets **json_manifest_file**.

```
# config/packages/framework.yaml
framework:
    # ...
    assets:
        # feature is supported in Symfony 3.3 and higher
        json_manifest_path: '%kernel.project_dir%/public/build/manifest.json'
```

Así nuestros enlaces en twig seguirán siendo los de antes.

```
<script src="{{ asset('build/app.js') }}"></script>

<link href="{{ asset('build/dashboard.css') }}" rel="stylesheet" />
```

Uso de CDNs

Si utilizamos CDN para alojar nuestros assets, una vez que hemos construido y subido al CDN nuestros assets, hay que configurar Encore para que utilice el CDN.

```
// webpack.config.js
// ...

Encore
    .setOutputPath('public/build/')
    // in dev mode, don't use the CDN
    .setPublicPath('/build');
    // ...
;


 if (Encore.isProduction()) {
     Encore.setPublicPath('https://my-dominio-cdn.es');

     // nos aseguramos que nuestros mapeos en manifest.jso
n siguen prefijados con build/
     // (ej: "build/dashboard.js": "https://https://my-dom
inio-cdn.es/dashboard.js")
     Encore.setManifestKeyPrefix('build/');
 }
```

Los enlaces a los assets siguen siendo los de antes gracias al manifest.json:

```
<script src="{{ asset('build/dashboard.js') }}"></script>
```

Cómo pasar información de Twig a JavaScript

A veces necesitamos que nuestros archivos javascripts (.js) sean dinámicos. Pero no es posible incluir código twig en ficheros js.

La técnica recomendada es utilizar atributos *data* para almacenar información y luego recogerla desde javascript.

```
<div class="js-user-rating" data-is-authenticated="{{ app.user ? 'true' : 'false' }}">
    <!-- ... -->
</div>
```

En el fichero javascript:

```
document.addEventListener('DOMContentLoaded', function() {
    var userRating = document.querySelector('.js-user-rating');
    var isAuthenticated = userRating.dataset.isAuthenticated;
});
```

NOTAS:

- Los nombres de los atributos se convierten de dash-style a

camelCase.

- No hay límite de tamaño del valor de un atributo data
- Es recomendable escapar el contenido del atributo para no romper el html.

```
<div data-user-profile="{{ app.user ? app.user.profileData
|json_encode|e('html_attr') : '' }}">
    <!-- ... -->
</div>
```