Estilos y JavaScript

Ahora que ya tenemos una visión bastante completa de lo que el motor de plantillas Blade puede ofrecernos, llega el momento de terminar de perfilar nuestras vistas. Hasta ahora no hemos hablado nada de estilos CSS, y eso es algo que toda vista que se precie debe incluir. Además, también puede ser necesario en algunos casos incluir alguna librería JavaScript en el lado del cliente para ciertos procesamientos. Veremos cómo gestiona Laravel estos recursos.

1. Infraestructura para archivos CSS y JavaScript

Para poder añadir estilos CSS o archivos JavaScript a nuestro proyecto Laravel, el framework proporciona ya unos archivos donde centralizar estas opciones.

En primer lugar, debemos tener en cuenta que todas las dependencias de librerías en la parte del cliente (JavaScript) se centralizan en el archivo `package.json`, disponible en la raíz del proyecto. Inicialmente cuenta ya con una serie de dependencias pre-añadidas. Algunas de ellas son importantes, como `vite` (o `laravel-mix`, dependiendo de la versión de Laravel que tengamos instalada), y otras puede que no las necesitemos y las podamos borrar. Es recomendable instalar las dependencias cuando creamos el proyecto, para tenerlas disponibles, con este comando:

```
npm install
```

Se creará una carpeta `node_modules` en la raíz del proyecto con las dependencias instaladas. Esta carpeta es similar a la carpeta `vendor`, también en la raíz del proyecto, pero esta última contiene dependencias PHP (no JavaScript). Ninguna de estas carpetas debe subirse a un repositorio *git*, ya que ambas pueden reconstruirse con el correspondiente comando de instalación de *npm* o de *composer*, según el caso y, además, pueden ocupar mucho espacio.

Para centralizar los estilos CSS, tenemos el archivo `resources/css/app.css`, o bien `resources/sass/app.scss` (dependiendo de la versión de Laravel que usemos), donde podemos definir estilos CSS propios, o incorporar librerías externas como veremos después, utilizando o bien CSS plano o bien Sass. Por ejemplo, podemos editar este archivo para añadir algún estilo propio para el cuerpo del documento:

```
body
{
    background-color: #CCC;
    font-family: Arial;
    text-align: justify;
}
```

Por otro lado, tenemos el archivo `resources/js/app.js` para incluir nuestras propias funciones en JavaScript, o incluso funcionalidades externas (a través de jQuery, por ejemplo).

2. Generación automática de CSS y JavaScript

Estos dos archivos anteriores (`resources/css/app.css` y `resources/js/app.js`) necesitan ser procesados para generar el código resultante (CSS y JavaScript) que formará parte de la aplicación, aunando todas las librerías y funciones que hayamos especificado. Para esto, tenemos dos opciones dependiendo del gestor de *frontend* que se tenga instalado.

2.1. Generación con Laravel mix

Hasta las primeras versiones de Laravel 9, se empleaba el gestor `laravel-mix` como generador de toda la parte de *frontend* . En este caso, se tiene el archivo `webpack.mix.js` en la raíz del proyecto, que emplea la herramienta *WebPack* para compilar, empaquetar y minificar estos archivos resultado CSS y JavaScript.

```
mix.js('resources/js/app.js', 'public/js')
   .postCss('resources/css/app.css', 'public/css', [
    //
    ]);
```

Como podemos intuir, desde este archivo `webpack.mix.js` se tomará todo lo que hay en el archivo `resources/js/app.js` y se generará un archivo optimizado ubicado en `public/js/app.js` . De forma similar, se tomarán los estilos definidos en `resources/sass/app.scss` o en `resources/css/app.css` (dependiendo de la versión de Laravel) y se generará un archivo CSS optimizado en `public/css/app.css` . Para desencadenar este proceso, Laravel y WebPack se valen de la librería `laravel-mix` , incluida en el archivo `package.json` . Por eso es importante esta librería, y por eso debemos dejarla instalada previamente con el comando `npm install` que hemos explicado antes. Una vez instalada, para generar los CSS y JavaScript debemos ejecutar este comando desde la raíz del proyecto:

```
npm run dev
```

Esto generará los archivos `public/css/app.css` y `public/js/app.js`, y después ya podremos añadir estos archivos en nuestras vistas, con algo como esto, respectivamente:

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="/css/app.css">
        <script type="text/javascript" src="/js/app.js">
        </script>
...
```

2.2. Generación con Vite

Desde alguna versión intermedia de Laravel 9, el gestor *frontend* por defecto que se incorpora es `vite`, junto con un archivo de configuración en la raíz del proyecto llamado `vite.config.js`. Debemos especificar en este archivo las rutas hacia los archivos CSS y JavaScript que queramos incorporar a la aplicación, aunque vienen por defecto ya incluidos los que hemos mencionado anteriormente:

```
...

export default defineConfig({
    plugins: [
        laravel({
            input: ['resources/css/app.css', 'resources/js/app.js'],
            refresh: true
        }),
    ],
});
```

Si queremos incluir cualquiera de estos archivos en nuestras páginas, podemos emplear la directiva `@vite` con la URL relativa. Por ejemplo:

```
<!doctype html>
<head>
    ...
    @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
```

Ejecutando el comando `npm run dev` pondremos en marcha el servidor *Vite*, útil si estamos desarrollando y probando la web. Si ejecutamos `npm run build` generaremos ya las rutas preparadas para producción. El primer comando, aunque puede resultar útil, remite a una URL alternativa para el proyecto que no siempre funciona correctamente, así que lo más cómodo es lanzar `npm run build` y que genere las dependencias adecuadas.

3. Incluir estilos Bootstrap

Uno de los frameworks de diseño web más utilizados a la hora de elaborar una web es Bootstrap. En este curso no vamos a dar demasiadas nociones sobre él, pero sí utilizaremos algunas pinceladas para que nuestras vistas tengan un aspecto más profesional.

Para incluir este framework en Laravel, debemos incluir una librería en el servidor llamada *ui*, que se encarga de incorporar distintas herramientas para diseño de interfaces de usuario (*UI*, *User Interface*).

```
composer require laravel/ui:*
```

> **NOTA:** la expresión `:*` al final del comando es para que descargue la última versión disponible compatible con el proyecto Laravel actual.

Una vez añadida la herramienta, la podemos emplear a través del comando `artisan` para incorporar Bootstrap al proyecto:

```
php artisan ui bootstrap
```

Esto incorporará Bootstrap al archivo `package.json`, en la sección de dependencias...

```
"devDependencies": {
    ...
    "bootstrap": "^5.1.3",
    ...
}
```

... y también añadirá un enlace a dicha librería en el archivo `resources/sass/app.scss`, para que podamos generar un archivo CSS optimizado con Bootstrap incluido:

```
...
@import '~bootstrap/scss/bootstrap';
```

Si utilizamos Bootstrap, entonces el archivo `resources/css/app.css` dejará de tener efecto, ya que para poder incorporar los estilos de Bootstrap a nuestro proyecto, se trabaja con Sass a través de `resources/sass/app.scss`. Nuestro archivo `webpack.mix.js` o `vite.config.js` también se habrá actualizado en este sentido, y deberemos colocar todos nuestros estilos CSS propios en el archivo `app.scss`:

```
 ...
 @import '~bootstrap/scss/bootstrap';

body
{
    background-color: #CCC;
    font-family: Arial;
    text-align: justify;
}
```

En el caso de usar Vite, deberemos actualizar la URL correspondiente al archivo CSS en las cabeceras donde hayamos incluido la directiva `@vite`, de este modo:

```
<!doctype html>
<head>
    ...
    @vite(['resources/sass/app.scss', 'resources/js/app.js'])
</head>
```

Para finalizar, debemos ejecutar nuevamente las instrucciones:

```
npm install
npm run build
```

La primera instrucción debe ejecutarse sólo una vez, y descargará e instalará Bootstrap en el proyecto (en la subcarpeta *node_modules*), y la segunda generará de nuevo los archivos CSS y JavaScript optimizados, incluyendo en ellos la librería Bootstrap. Con esto ya tendremos disponibles las clases y estilos de Bootstrap para nuestras vistas.