MEMORIA PEC1

Descargar/clonar el repositorio

A partir del enlace que nos brinda el módulo 1 de Herramientas HTML y CSS II he descargado del repositorio de GitHub el código del Boilerplate de la UOC mediante el botón verde CODE, que nos da la opción. A partir de aquí el .zip obtenido lo he descomprimido.

Instalación de dependencias.

1. FontAwesome

En el terminal ejecutamos:

```
npm install --save @fortawesome/fontawesome-free
```

La dependencia se añade en el fichero **package.json y también en** el directorio **node_modules.**

Seguidamente incorporamos las dependencias FontAwesome al **bundle,** es decir al paquete final que irá a producción en el fichero main.scss. Añadimos:

```
@import "npm:@fortawesome/fontawesome-free/css/all.css";
```

en la ruta y en la línea correspondiente de código dentro del fichero:

src/assets/styles/main.scss

De esta manera al añadir un icono gratuito de la dependencia en el código HTML de nuestra página se mostraría en pantalla ejecutando:

```
npm run dev
```

2. cssnano

Instalación del plugin cssnano de PostCSS para realizar la **minificación** de ficheros CSS. cssnano requiere de la instalación previa de PostCSS. Instalamos cssnano teniendo en cuenta que trabajamos con el module bundler **parcel**.

```
npm install @parcel/optimizer-cssnano
```

3. htmlnano

Instalación del plugin **htmlnano** de PostHTML para realizar la **minificación** de ficheros HTML. htmlnano requiere de la instalación previa de PostHTML. Instalamos htmlnano teniendo en cuenta que trabajamos con el module bundler **parcel**.

```
npm install @parcel/optimizer-htmlnano
```

4. Fichero configuración de Sharp

La dependencia de optimización de imágenes Sharp está contamplado en el boilerplate de guía de la UOC y hemos añadido un fichero de configuración ***sharp.config.json*** en el cual he reducido la calidad del jpeg para producción a 75 consiguiendo reducir el tamaño de la imagen teniendo buena calidad.

Sharp permite trabajar con varias formatos de imágenes modernos.

5. Dependencia Babel.

He instalado la dependencia babel para una transpilación de ES6 a ES5, no obstante con el fichero de configuración .babelrc y poniendo la configuración adecuada **sería equivalente,** ya que parcel tiene babel integrado.

Entorno de desarrollo

IDE para realizar el proyecto web: **VisualStudioCode.**

El gestor de paquetes utilizado es **NPM** con la versión:

```
npm -v  =  v8.5.0
```

y **node** con la versión:

```
node -v  =  v16.14.2
```

Para instalar la versión anterior de node he instalado **nvm** (gestor de versiones de node).

Compilación para producción

Para compilar y obtener los ficheros resultado para producción ejecutamos:

```
npm run build
```

mediante **run** indicamos que vamos a ejecutar un script (en este caso build).

En nuestro package.json tenemos configurado:

“build”:”npm-run-all clean stylelint parcel:build

para que ejecute clean (limpia cache y directorios) stylelint para que valide las reglas antes de generar los ficheros de producción y por último parcel, que genera los ficheros resultado.

En los ficheros generados después de compilar para producción observamos:

– Un único fichero de estilos CSS.
– El tamaño de la imagen en el curriculum se ha reducido.
– cssnano y htmlnano han minificado los ficheros y sale todo el código en una sola línea y de menor tamaño que los de desarrollo.
– Genera dos ficheros .js. Uno de para la especificación ES6 y otro para la ES5. Se puede apreciar porque en el include se implementa el **type=”module”** correspondiente a la ES6.

He tenido un problema en producción porque la parte javascript no funcionaba. He podido averiguar que era debido a que en el index.html al incluir el javascript indicaba (porque he seguido el index.html del boilerplate) **‘type=module’. He quitado esta premisa y la parte javascript en producción funciona correctamente.** No he tenido tiempo pero creo que poniendo la configuración adecuada posiblemente en el fichero **.posthtmlrc** se resuelva el problema sin tener que eliminar esta premisa.

Publicación en Netlify mediante GitHub

Subir el código al repositorio GitHub

1. Creamos un repositorio en GitHub desde este sitio web.

<u>Desde el terminal realizamos las siguientes acciones y/o comandos:</u>

1. No situamos en la carpeta ‘PEC1’ donde se encuentran todos los ficheros que queremos subir al repositorio.

2. Ejecutamos **git init** (Comando que inicializa un repositorio local de git).

He copiado el fichero .**gitignore** del proyecto de la UOC proporcionado en el enunciado y he **eliminado la entrada dist de este fichero** para que al realizar el comando del siguiente punto 3 el directorio **dist se añada al repositorio.**

3. **git add .** (Añade todos los ficheros del directorio actual ‘PEC1’ al repositorio local y los prepara para poder realizar el commit).

4. **git commit -m “version1.0.0”** (Hacemos commit de los ficheros añadidos al repositorio local). (Puede ser que nos pida autentificación en local, seguid los pasos).

5. **git branch -m main** (El commit anterior ha provocado la creación de una rama llamada *‘master’*, pero en nuestro repositorio remoto de GitHub la rama principal se llama *‘main’* así que cambiamos el nombre de la rama en local para que coincidan y no haya conflicto con este comando).

6. **git remote add origin <remote url>**
(Asignamos la dirección de nuestro repositorio remoto que en nuestro caso es “https://github.com/oscar1delguila/Herramientas1-PEC1.git” a ‘origin’).

7. **git remote -v** (Comprobamos que realmente se ha asignado bien la dirección remota).

8. **git push origin main** (Subimos a nuestro repositorio remoto los ficheros desde nuestro repositorio local, en la rama main).

Publicación en Netlify

1. Realizar una vinculación entre el repositorio remoto en github y netlify.

2. Indicar en **Settings** del apartado **Deploys de Netlify** el directorio de publicación (donde se encuetran los ficheros de producción), en nuestro caso **'dist'**. Netlify también puede construir nuestro proyecto con nuestro package.json desde cero , pero es tontería teniendo en cuenta que lo hemos hecho en local y que los ficheros de producción están listos para 'usarse' en el directorio dist. Por lo tanto en Netlify no es necesario incluir el comando **npm run build.**

.

3. Desplegar el proyecto.

4. Si tenemos más de una versión de deploy (despliegue) del sitio, indicar cuál queremos publicar en internet.

5. Indicar mediante un botón en netlify que detecte subidas nuevas de cambios en el **github** y refrescar la web publicada en internet automáticamente.

Justificar la elección de metodología y/o guía de estilo en base al tipo de encargo, tus conocimientos, la metodología de desarrollo aplicada y la avenencia de los conceptos estudiados con el proyecto y tu estilo de código.

Justificación de Guía de estilo escogida @mdo.

La guía de estilo @mdo es menos "severa" que la guía de estilo de Harry Roberts. La guía de estilo de Harry Roberts está más enfocada a proyectos grandes con gran seriedad y responsabilidad para mantener las hojas de estilo del proyecto; de tal forma que indica hasta los espacios que se han de dejar para comentarios, alineados en las diferentes propiedades de la regla CSS, y la de @mdo no.

He escogido la guía de @mdo porque me ha parecido muy interesante que las propiedades de las reglas CSS se declaren en un orden establecido, que creo que es muy importante a la hora de leer y mantener la hoja de estilo sobre todo en costes de tiempo, análogamente para el documentos HTML, en cuanto al orden de declaración de los atributos del tag HTML, donde repito se gana mucho tiempo si se ha de leer la regla CSS.

No obstante en el proyecto web del CV he adoptado algunas reglas de la guía de **Harry Roberts** para realizar los **comentarios** de las distintas secciones de estilos que se han declarado en el CSS.

En cuanto a mi estilo de código se asemeja más a la guía de @mdo y no me ha resultado 'muy caro' escribir conforme la guía.

La metodología que he utilizado para codificar las hojas de estilo es ITCSS con nomenclatura BEM (ITCSS + BEM = BEMIT).

La metodología SMACSS (Arquitectura modular y escalable para CSS) tiene como objetivo fundamental dar significado semántico a nuestro proyecto para facilitar la legibilidad y ser más inteligible, pero esta metodología es anticuada porque no está adaptada a los lenguajes modernos como Angular que se caracteriza por dividir el proyecto en componentes o módulos. He pensado en utilizar OOCSS que también vendría bien ya que en el proyecto se distinguen los distintos contenedores que dividen el curriculum en idiomas, experiencia, … y el diseño, pero he preferido aplicar ITCSS para dar más semántica al proyecto y mayor escabilidad.

ITCSS divide nuestras hojas de estilo en diferentes capas, donde el peso recae en la capa de componentes (donde hay más hojas de estilo), de forma que si hubiera que hacer una migración de estilos a un proyecto Angular sería mucho menos costoso. Además la idea de tener las hojas de estilo divididas en diferentes directorios cada uno de los cuales indicando un propósito diferente es útil porque permite que el proyecto sea escalable y fácil de mantener al igual que con sentido semántico.

El proyecto web de la PEC1 es muy pequeño y se puede pensar que no vale la pena aplicar ITCSS, pero en producción las hojas de estilo en desarrollo se convierten en una sola, no afecta el rendimiento en producción tener tantas hojas de estilo y ITCSS *funciona bien* por los motivos antes explicados.

Explicar detalladamente cómo aplicaste los criterios escogidos a tu código, así como la configuración realizada en Stylelint para que se aplicaran estos criterios.

Aplicación de los criterios escogidos al código

Para aplicar la regla de estilo de la guía de @mdo de ***orden de las declaraciones de las propiedades*** en las reglas que conforman el CSS he añadido la dependencia:

```
npm install --save-dev stylelint-config-recess-order
```

y también escribiendo en el fichero de configuración **.stylelintrc.json:**

```
“extends”: [
                    ...
        “stylelint-config-recess-order”,
                    ...
  ]
```

Configuracióin de Stylelint

Stylelint **es un linter** que se compone de un montón de reglas para hojas de estilo CSS y de plugins (también se puede crear reglas personalizadas) con el **objetivo de evitar errores y forzar el buen uso de las convenciones de estilos en CSS** que existen actualmente y/o personalizados. Para configurar stylelint primeramente lo instalamos:

```
npm install --save-dev stylelint
```

Stylelint soporta la sintaxis de hojas de estilo SCSS, pero añadimos un plugin llamado **stylelint-scss** para SCSS que permite introducir reglas SCSS. Además **activamos un conjunto de reglas** recomendadas mediante la dependencia **stylelint-**

config-recommended-scss (conjunto de reglas scss que sea aplican a nuestros estilos tan solo por instalar esta dependencia), que también instalamos:

```
npm install --save-dev stylelint-scss stylelint-config-recommended-scss
```

Creamos en la raíz del proyecto el fichero **.stylelintrc** con la estructura siguiente:

```
{
  "extends": ["stylelint-config-recommended-scss"],
  "rules": {}
}
```

en donde en **rules** escribiremos las reglas o bien anularemos algunas que no nos interesen o con las que pueda haber conflicto.

Codificación en el proyecto para la metodología ITCSS

ITCSS no aprueba la anidación en SCSS de más de dos niveles por lo que escribimos en el fichero **.stylelintrc.json** la siguiente regla:

```
“max-nesting-depth”:2
```

También en este fichero **para evitar conflictos de la anidación de estilos que permite SCSS** con los estilos CSS propiamente dichos **anulamos la regla “no-descending-specifity”** que determina el orden de la especificidad en los selectores de la forma siguiente:

```
.component1 a:hover {}
.component2 a {}
```

Stylelint daría error porque el primer selector tiene mayor especificidad que el segundo, declarado más abajo y para que no diera error deberían de estar declarados en orden inverso.

), **poniendo su valor a null.**

He codificado de forma que los componentes sean independientes los unos de los otros.

Para aplicar la nomenclatura de **BEM** he implementado la regla **“selecto-class-pattern”** con la siguiente expresión regular

“^[a-z]([a-z0-9-]+)?(__([a-z0-9]+-?)+)?(--([a-z0-9]+-?)+){0,2}$"

donde no hace falta poner el punto que indica el inicio de clase.

Explicar cualquier decisión de diseño y desarrollo que tomaras (por ejemplo: qué dependencias añadiste, si en tus hojas de estilo seguiste una aproximación desktop first o mobile first, …).

En mis hojas de estilo **he seguido** una aproximación **desktop first** para realizar el diseño de las hojas de estilo de la página HTML. El motivo es que para la versión mobile no resulta complicado adaptar las hojas de estilo desde la versión desktop, teniendo en cuenta que he utilizado elementos CSS de carácter responsive como **flex** y **grid**, que son fáciles de trabajar desde la versión desktop.

He creado un **punto de interrupción** mediante una consulta al medio en las hojas de estilo para **dos tipos de dispositivo**. Los que superan el ancho de 500px y los que no, donde adoptarán unos estilos u otros en función de su tamaño. Teniendo en cuenta que no es una web complicada y solamente de una página no veo necesario añadir más consultas a los medios (media query) para implementar una buena hoja de estilos. Las consultas al medio se realizan en las propias reglas CSS, tal y como indica la guía @mdo (mucho menos tedioso y más cómodo para realizar el mantenimiento del proyecto).

La dependencia que cabe destacar en este punto que he añadido es la de **FontAwesome,** que importa todo el CSS gratuito. Sería mejor importar el único icono que utilizo en el proyecto que es un *sombrero de graduación* en la web en formato SVG y embeber su código en el HTML para no tener que importar todo el fichero CSS de FontAwesome que relentiza la carga de la página (aunque está implementado como indico al principio de este párrafo).

Incluir los enlaces al repositorio de GitHub y la URL pública de Netlify.

– URL Curriculum vitae

https://curriculum-vitae-oag.netlify.app

– Repositorio github:

https://github.com/oscar1delaguila/herramientasHTML-CSS-II-PEC1