|  |
| --- |
| Herramientas HTML y CSS II |
| PEC 1 |
|  |

|  |
| --- |
| Alarcon Moya, Teo |

**Index**

**Información sobre el repositorio y la web** 3

**1.Creación de Proyecto utilizando UOC Boilerplate** 4

**2.Desarrollo Página Web sobre Tokyo con UOC Boilerplate** 5

**3.Estructura y Metodología de Diseño con SCSS** 6

**4.Deployment** 9

**Información sobre el repositorio y la web**

|  |  |
| --- | --- |
| **Repositorio Remoto** | <https://github.com/Moyat89/VisitaTokyo> |
| **Link Netlify** | <https://tokyocap.netlify.app/> |

**1. Creación de Proyecto utilizando UOC Boilerplate**

1. Antes de comenzar, me he asegurado de tener [Node.js](https://nodejs.org/en/download/) instalado en el sistema mediante el comando node -v. Cómo ya lo tenía instalado para otra asignatura de este Máster, no ha hecho falta que lo descargase e instalara de nuevo.
2. He creado una carpeta nueva para este proyecto. Con Windows, he abierto la consola de comandos (cmd), he navegado hacia la ubicación donde quería tener mi proyecto y con el comando mkdir he creado una carpeta nueva para este.
3. Una vez en la carpeta del proyecto, he ejecutado el comando git clone https://github.com/uoc-advanced-html-css/uoc-boilerplate.git para clonar el repositorio con el código del *boilerplate* de la UOC que se requiere para esta PEC en la carpeta de mi proyecto.
4. Al acabar de clonar el repositorio, he instalado npm para poder gestionar las dependencias necesarias (npm install). En ese sentido, he añadido Font Awesome como una dependencia externa (npm install --save @fortawesome/fontawesome-free), ya que me permitía enriquecer el front-end de la página web mediante la representación de diferentes iconos. He explorado el añadir otras dependencias externas, pero muchas se basaban en funcionalidades que no iban a destacar en el tipo de proyecto web simple que se propone para este ejercicio (centradas en el back-end).
5. Además, para asegurar la calidad y consistencia del código SCSS y seguir con las pautas marcadas del ejercicio, he incorporado *Stylelint* como herramienta de *linting*. Lo he instalado (junto a las recomendaciones de configuración para SCSS) con el siguiente comando: npm install --save-dev stylelint-scss stylelint-config-recommended-scss. Tras su instalación, he creado el archivo .styleintrc en la carpeta raíz del proyecto con la siguiente configuración para asegurarme de que Stylelint comprobaba esas reglas:

{

"extends": ["stylelint-config-recommended-scss"],

"rules": {}

}

**2.Desarrollo Página Web sobre Tokyo con UOC Boilerplate**

Para crear la página web propuesta, se ha elegido la capital japonesa de Tokyo como temática. El sitio se web se compone de una sola página, con una cabecera que sirve como panel con la información básica del sitio (titulo, logo, redes sociales) y que podría reciclarse para que sirva de menú de navegación en un futuro. Como contenido principal nos encontramos un texto describiendo a la capital japonesa y, por último, encontramos un *footer* con imágenes varias de la ciudad.

Para desarrollar su HTML, he partido los archivos html originales del Boilerplate (index.html y footer.html) y he modificado su contenido.

La página web de Tokyo consta de los siguientes elementos:

1. **Encabezado (Header):**

* Contiene tres clases distintas.
* La primera clase alberga un logo, obtenido a través de Font Awesome.
* La segunda clase incluye un elemento <h1> que sirve como título principal de la página.
* La tercera clase contiene tres iconos de redes sociales, también obtenidos mediante Font Awesome, que actúan como enlaces a las cuentas de la página en diferentes plataformas.

1. **Contenedor Principal (main-container):**
   * Se ha creado una clase llamada "main-container" que funciona como el contenedor principal del contenido de la página web.
2. **Footer (Pie de Página):**
   * Se ha reciclado el elemento <include> propio del plugin *posthtml-include* presente en el boilerplate y en código HTML original para incorporar el contenido HTML del archivo src/views/footer.html como pie de página.
   * El archivo footer.html se ha modificado para incluir una serie de imágenes de Tokyo, que proporcionan un complemento visual y representativo al pie de la página.

**3. Estructura y Metodología de Diseño con SCSS**

En la carpeta *styles*, se encuentran los archivos SCSS que contienen el diseño de la página web. El archivo principal es el main.scss, el cual importa todos los parciales necesarios. Estos parciales incluyen:

* \_home.scss: Contiene el diseño principal de la página index.html.
* \_variables.scss: Define todas las variables de color y tamaño utilizadas con frecuencia en \_home.scss.
* \_functions.scss: Contiene funciones que se aplican al código SCSS.

Para el diseño de la web, se ha decidido seguir la metodología mobile-first. Esto significa que la página se ha diseñado teniendo en mente dispositivos pequeños, y luego se han aplicado reglas mediante *media queries* para modificar el diseño cuando se visualiza a través de dispositivos medianos y grandes.

En términos de estilo, he decidido utilizar las siguientes *best practices,* algunas solamente posibles utilizando SASS:

* **Uso de Variables (SASS)**: Se han utilizado variables para cualquier valor que se repitiese más de una vez en el código, lo que mejora la mantenibilidad y la coherencia del diseño.
* **Funciones para calcular tamaños (SASS)**: Se ha implementado una función para determinar automáticamente el tamaño del contenedor principal del contenido dependiendo de la altura del encabezado. Esto permite una adaptación dinámica del diseño a diferentes tamaños de encabezado en dispositivos móviles y de escritorio.
* **Viewport Units:** Se han utilizados unidades de viewport (vh y vw) para definir las dimensiones de anchura y altura de los contenedores principales, garantizando una presentación consistente en diferentes tamaños de pantalla.
* **Anidamiento (SASS)**: Se ha intentado aprovechar al máximo el anidamiento proporcionado por SCSS. Con la excepción del elemento body, todo el código CSS queda anidado dentro de los tres contenedores principales (header, main-container, footer), lo que mejora la organización y legibilidad del código.

En cuanto a Stylelint, se han aplicado reglas específicas de OOCSS (Object-Oriented CSS). Estas reglas las he escogido porque me parece que hacen una perfecta simbiosis entre el anidamiento y las funciones de SCSS, creando una estructura de código similar a la programación orientada a objetos, como Java. Además, todas las reglas escogidas fomentan a que el código sea más modular y flexible, siendo de esta manera más fácil de modificar, mantener y leer en un futuro.

"selector-class-pattern": "^[a-z][a-zA-Z0-9]\*(-[a-zA-Z0-9]+)\*$",

"selector-max-specificity": "0,4,0",

"declaration-block-no-duplicate-properties": true,

"declaration-block-no-shorthand-property-overrides": true,

"at-rule-no-unknown": [true, {

"ignoreAtRules": [

"function" , "return"

]

**Selector-class-pattern**: He aplicado esta regla para establecer un patrón consistente para los nombres de las clases CSS. Este se compone en que los nombres de las clases comiencen con letra mínuscula; puedan contener mayúsculas, minúsculos y números después de esta y que se puedan incluir guiones seguidos de más caracteres alfanuméricos.

**Selector-max-specificity***:* Establece un límite de especificidad que un selector CSS puede tener en su código. Con los valores descritos, no se permite utilizar IDs (#) en los selectores, se permite un máximo de 4 clases, pseudoclases o atributos en estos y no se permiten elementos y pseudoelementos.

**Declaration-block-no-duplicate-properties**:Con el valor true en esta regla, evitamos que tengamos la misma propiedad repetida varias veces dentro de un selector.

**Declaration-block-no-shorthand-property-overrides":** Con el valor true, no permite que una propiedad individual sobreescriba a una individual (por ejemplo border-top sobreescriba los valores de border si estos ya han sido especificados).

**At-rule-no-unknown": [true, {"ignoreAtRules": ["function" , "return"]:**No permitimos que reglas desconocidas se puedan aplicar. Como en nuestro scss hemos definido funciones he añadido que ignore function y return.

Además de estas reglas, se han seguido las otras recomendaciones de [Stylelint para SCSS](https://github.com/stylelint-scss/stylelint-config-standard-scss/blob/HEAD/index.js), como por ejemplo, la regla kebab-case para las variables, que no me ha parecido oportuno especificar en este documento.

**4. Deployment**

Tras terminar de completar el desarrollo de la página web, he revisado el archivo .gitignore y he observado que ya se habían especificado las carpetas que debían ser ignoradas al subir el código a GitHub, dejando solo las indispensables para publicar la web. También he revisado el JSON, pero no he modificado su información, ya que mucha pertenecía al boilerplate utilizado y no estaba seguro de modificar los campos de autoría. A continuación, he registrado todos los cambios en los archivos utilizando el comando git add ., añadiendo todos los archivos modificados y nuevos para el siguiente commit. Luego, he ejecutado el comando git commit. En este paso, también he escrito un mensaje (-m”mensaje”) para explicar los cambios realizados en esta versión del código. Finalmente, he subido el código al repositorio remoto en GitHub utilizando el comando git push. El código completo se puede encontrar en mi repositorio de GitHub en la siguiente URL: <https://github.com/Moyat89/VisitaTokyo>.

Al tenerlo en GitHub, ya disponía de todos los elementos para publicar la web en Netlify. Como ya la había utilizado en la anterior asignatura de HTML y CSS, solamente he tenido que utilizar mi cuenta y añadir un nuevo sitio web. El resultado se puede visualizar en <https://tokyocap.netlify.app/>.