

HTML Y CSS II

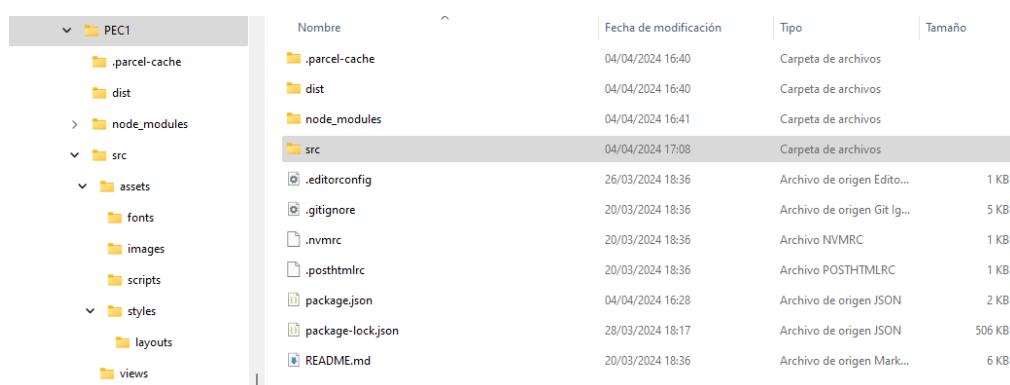
PEC1

ÁNGEL DE LUCAS GIL

1. ENTORNO DE DESARROLLO

Para el proceso de creación o desarrollo se ha empleado el boilerplate creado para esta asignatura y proporcionado junto con los recursos de la asignatura **UOC Boilerplate**. Como bien se explica en la teoría aportada para esta práctica, UOC Boilerplate se basa en parcel, por tanto, como punto de partida contamos con una estructura de ficheros y herramientas que facilitarán el desarrollo de la aplicación. Este module bundler es proporcionado por la UOC y simplemente debemos descomprimir el zip en el directorio de trabajo y ejecutar **npm install** para que se añadan todas las dependencias necesarias y se cree la carpeta `node_modules`.

La estructura de ficheros que ofrece UOC Boilerplate es la siguiente:



Nombre	Fecha de modificación	Tipo	Tamaño
.parcel-cache	04/04/2024 16:40	Carpeta de archivos	
dist	04/04/2024 16:40	Carpeta de archivos	
node_modules	04/04/2024 16:41	Carpeta de archivos	
src	04/04/2024 17:08	Carpeta de archivos	
.editorconfig	26/03/2024 18:36	Archivo de origen Edito...	1 KB
.gitignore	20/03/2024 18:36	Archivo de origen Git Ig...	5 KB
.nvmrc	20/03/2024 18:36	Archivo NVMRC	1 KB
.posthtmlrc	20/03/2024 18:36	Archivo POSTHTMLRC	1 KB
package.json	04/04/2024 16:28	Archivo de origen JSON	2 KB
package-lock.json	28/03/2024 18:17	Archivo de origen JSON	506 KB
README.md	20/03/2024 18:36	Archivo de origen Mark...	6 KB

Algunas de las herramientas que usaremos y que están incluidas en UOC Boilerplate son:

1. Pre y post procesadores de estilos. Como preprocesador de estilos contamos con Sass, que permite la organización del código css en ficheros o el uso de utilidades como funciones o reutilización de código mediante el uso de reglas de estilos definidas. PostCSS es un postprocesador de estilos que permite, entre otras tareas, la generación de código para navegadores antiguos.
2. Post procesadores de HTML. UOCBoilerplate usa PostHTML, que por defecto incluye el plugin posthtml-include para añadir ficheros HTML parciales.

Para el desarrollo de la PEC1 he decidido dividir la página en secciones (header, body, footer) y crear un fichero HTML parcial para cada una de ellas, de esta forma el resultado se compone de varios ficheros HTML parciales, independientes y fácilmente reutilizables en otras aplicaciones. He decidido realizar esta división debido a que se trata de una aplicación con una sola página y de esta forma se puede apreciar fácilmente las ventajas de desarrollar con un module bundler y usar herramientas de pre y post procesado. En mi caso no he alojado los ficheros html y scss en subcarpetas puesto que es un proyecto pequeño con un número bajo de elementos.

Así, la estructura del fichero index.html queda del siguiente modo:

```
<body>

<include src="./views/header.html"></include>
<include src="./views/icon-list-links.html"></include>

<main class="main-container">

  <include src="./views/submenu.html"></include>
  <include src="./views/intro.html"></include>
  <include src="./views/recent-articles.html"></include>
  <include src="./views/main-input.html"></include>
  <include src="./views/sites.html"></include>

</main>

<include src="./views/footer.html"></include>

</body>
```

El encabezado formado por el fichero header.html, el cuerpo de la aplicación incluido en el elemento main y el footer.html. Además he incorporado un fichero adicional llamado icon-list-links.html para añadir unos iconos flotantes a la aplicación. Como se puede ver en la imagen, el elemento main contiene a su vez varios ficheros html parciales. El objetivo de incluir los ficheros html dentro del main, es tener el contenido de la web alojado en un mismo elemento, para facilitar las tareas de centrado y alineamiento de los diferentes elementos dentro de la página.

Para mantener una estructura lógica y sencilla de entender dentro del proyecto, para cada uno de los ficheros html he creado un fichero de estilos (.scss) con el fin de conseguir crear una página totalmente modular y un código completamente reutilizable. De esta forma el fichero main.css, que contiene todos los ficheros de estilos resulta como se puede ver en la siguiente imagen:

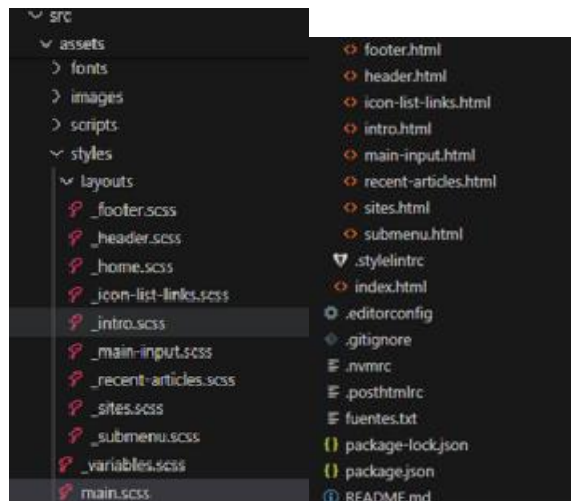
```
@import "variables";

/**
 * Import npm dependencies
 * see: https://v2.parceljs.org/features/module-resolution/
 * see commented examples below
 */
@import "npm:@fortawesome/fontawesome-free/css/all.css";
// @import "some-node-module";
// @import "@some-company/some-node-module";

/**
 * Import layouts
 * It's a best practice to use a different partial for each page
 */
@import "layouts/header";
@import "layouts/submenu";
@import "layouts/intro";
@import "layouts/home";
@import "layouts/icon-list-links";
@import "layouts/recent-articles";
@import "layouts/main-input";
@import "layouts/sites";
@import "layouts/footer";

/**
 * Do NOT include anything else in this file!
 */
```

La estructura de ficheros de UOC Boilerplate se respeta en todo el desarrollo y simplemente se añaden los ficheros nuevos necesarios:



El fichero main.scss contiene todos los **@import** de los ficheros de estilos parciales y el fichero index.html contiene todos los fichero html parciales que componen la página.

Para lanzar la web tenemos la opción de desarrollo y la opción de compilar para producción y ambas son proporcionas por UOC Boilerplate en la sección de scripts del fichero package.json:

Para desarrollo: **"dev": "npm-run-all clean parcel:serve"**

Para compilar: **"npm-run-all clean stylelint parcel:build"**

Lanzando la aplicación para desarrollo parcel levanta un servidor que permite visualizar la página en un navegador a través del puerto 8123 en la IP local (http://localhost:8123/).

Al compilar la aplicación se ejecuta la verificación del código mediante stylelint y si hay algún error podríamos ver los logs de los mismos y corregirlos de forma manual o dejar que lo haga stylelint a través de otro comando que se ha incluido en la sección de scripts del package.json al realizar la configuración de stylelint. El resultado de la compilación se almacena en una carpeta llamada **dist** dentro del directorio de trabajo.

2. DEFINICIÓN DE ESTILOS Y CONFIGURACIÓN STYLELINT

La definición de estilos la he realizado siguiendo el modelo **“mobile first”** definiendo breakpoints para cada uno de los html parciales en función de la información que quiero mostrar y del modo en el que la quiero mostrar. Esta decisión viene motivada por el alto uso de los dispositivos móviles frente a los PC’s para navegar por la red, por tanto, la página se diseña en base a este criterio y este diseño se va adaptando en función del tamaño de la pantalla.

Para esta PEC he aplicado una guía de estilos basada en **BEM** porque bajo mi punto de vista es la metodología que más aporta a la hora de leer la hoja de estilos puesto que permite conocer la estructura del HTML. Cada elemento se identifica por una clase con unas determinadas reglas y siguiendo el patrón de nombres establecido por BEM (**B**loque__**E**lemento--**M**odificador). En los ficheros scss de definición de estilos he

mantenido el anidamiento resultante en los correspondientes ficheros html, de forma que al consultar cualquier regla definida en estos ficheros se pueda ver la estructura que forman los elementos en el DOM. Como resultado de aplicar este estilo en el código css, los nombres de los elementos se componen por el nombre del bloque inmediatamente superior seguido del nombre del elemento. Como punto negativo, se puede destacar el alto nivel de anidamiento que se puede generar si el código html fuera complejo, no obstante, he decidido aplicar este criterio por ser una página sencilla con pocos elementos. Para proyectos con mayor envergadura creo que puede ser más eficiente establecer bloques de referencia de orden superior del cual cuelguen el resto de elementos y establecer unos límites de anidamiento en stylelint para asegurar su cumplimiento.

De esta forma, la guía de estilos definida para esta PEC sigue las siguientes pautas (adjunto imagen del fichero .stylelintrc con la configuración de cada pauta):

- **Comentarios vacíos:** regla para evitar comentarios sin contenido.

```
"comment-no-empty": true,
```

- **Anidamiento:** sin límite en el número de bloques o elementos anidados.

```
"selector-max-compound-selectors": null,  
"max-nesting-depth": null,
```

- **Nombres de clases:** metodología BEM.

```
"selector-class-pattern": [  
  "^[a-z]([a-z0-9-]+)?(_([a-z0-9-]+)?)(--[a-z0-9-]+)?{0,2}$",  
  {  
    "message": "Expected custom property name to be Block__Element--Modifier"  
  }  
,  
,
```

- **Orden de las propiedades** definidas en las reglas por bloques. El plugin "stylelint-order" sirve para ordenar las propiedades por bloques. Con el extend "stylelint-config-clean-order/error" si no se cumple esta regla se categoriza como error. Con order/properties-order" se establece un orden personalizado para las propiedades definidas.

```
"extends": [  
  "stylelint-config-clean-order/error",  
  {  
    "plugins": [  
      "stylelint-order"  
    ],  
    "order/properties-order": [  
      "width",  
      "height",  
      "padding",  
      "padding-top",  
      "padding-right",  
      "padding-bottom",  
      "padding-left",  
      "border",  
      "margin",  
      "margin-top",  
      "margin-right",  
      "margin-bottom",  
      "margin-left",  
      "display",  
      "flex-direction",  
      "position",  
      "top",  
      "right",  
      "bottom",  
      "left"  
    ]  
  }  
,
```

- Iteración usuario.
- Posicionamiento. Siguiendo el patrón:

- Position.
- Top.
- Right.
- Bottom.
- Left.
- Layout. Siguiendo el patrón:
 - Width/height.
 - Border.
 - Padding.
 - Margin.
- Modelo de cajas. Siguiendo el patrón:
 - Display.
 - Flex-direction.
- Tipografía.
- Apariencia.
- No se permiten líneas vacías entre bloques.

```
"custom-property-empty-line-before": [
  "never",
  {
    "message": "Empty line before custom property not allowed"
  }
],
"declaration-empty-line-before": [
  "never",
  {
    "message": "Empty line before declaration not allowed"
  }
],
```

- **Unidades permitidas:**

```
"unit-allowed-list": [ "%", "deg", "px", "rem", "ms", "s", "vh", "vw", "fr"],
```

- **Unidades empleadas para bordes y tamaño de fuentes:**

```
"declaration-property-unit-allowed-list": {
  "^border/": [
    "px"
  ],
  "^font-size/": [
    "rem"
  ]
},
```

- Bordes: px.
- Tamaños de letra: rem.

- No se permiten los **valores desconocidos** para las propiedades de las reglas.

```
"declaration-property-value-no-unknown": [
  true,
  {
    "message": "Property value not allowed"
  }
],
```

La configuración de stylelint está hecha en base a estas normas para garantizar su aplicación y generar un código css más uniforme y legible. Estas son normas personalizadas, pero además de estas estoy aplicando la configuración por defecto de stylelint, por lo que se aplican todas las normas estándar y las propias para este desarrollo arriba descritas.

A parte, he añadido mensajes de error customizados para algunas de las normas incluidas en stylelint.

Para la configuración de stylelint he seguido los siguientes pasos:

1. npm install --save-dev stylelint

2. **npm install --save-dev stylelint-scss stylelint-config-recommended-scss**
3. **npm install --save-dev stylelint-config-recess-order**
4. **npm install stylelint-config-clean-order**
5. Añadir en el fichero package.json, en la sección de scripts, para ejecutar el chequeo del código y la corrección de errores.

```
"stylelint": "stylelint src/**/*.scss",  
"stylelint_fix": "stylelint src/**/*.scss --fix",
```

En el paso número 4 he instalado el **plugin “stylelint-order”** que permite mantener siempre el mismo orden en las propiedades en las reglas de estilos. Este plugin junto con la regla "order/properties-order" me permite ordenar según mi criterio dichas propiedades.

En el fichero _variables.scs se incluyen los **@mixin**, **@function** y variables globales usadas para el desarrollo. En el caso de las variables se incluyen los tipos de letra y los colores principales para textos y contrastes.

Como dependencia externa he añadido **fontawesome**, que permite incluir iconos prediseñados en nuestra aplicación. Para usarla, en el fichero main.css debemos incluir la siguiente declaración:

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

3. HTML

UOC Boilerplate usa PostHTML para poder crear ficheros parciales para modularizar nuestro código y hacerlo más sencillo de seguir y reutilizar.

El proyecto se compone de 8 ficheros html parciales y sus correspondientes hojas de estilos. La división en ficheros de todas las secciones que componen la página es crear un código fácilmente reusable y modular, que permita una fácil reorganización del mismo en el contenedor principal:

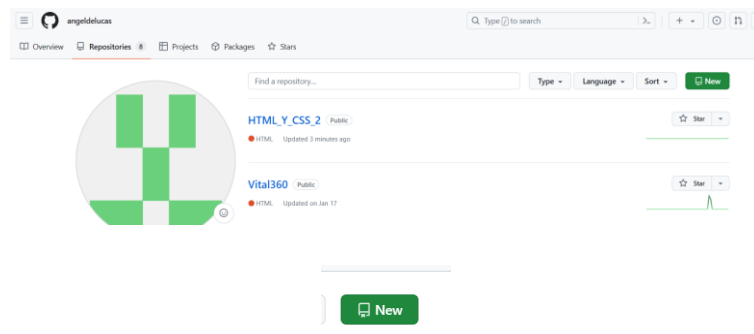
- **header.html**: cabecera de la página compuesta por imágenes.
- **submenu.html**: menú de la página que se encuentra fijo en la parte superior de la página.
- **icon-list.html**: lista de iconos flotantes con enlaces a sitios de interés.
- **intro.html**: sección que aparece al comienzo de la página y que muestra una sucesión de fotos cambiantes.
- **main-input.html**: cuerpo de la página con la información principal.
- **recent-articles.html**: sección de artículos o información reciente y de importancia.
- **sites.html**: sección de cosas que no te puedes perder si visitas la localidad.
- **footer.html**: pie de la página.

4. CONTROL DEL CÓDIGO

GitHub

Para el desarrollo de la PEC1 es necesaria la creación de un repositorio en GitHub para subir el código y posteriormente para desplegar la aplicación en producción a través de la herramienta Netlify. El repositorio alberga el código fuente de la aplicación sin compilar para producción y para crearle hay que seguir los siguientes pasos:

1. Desde la vista de nuestros repositorios en GitHub clicamos en el botón New, deberemos darle el nombre que queramos y que luego usaremos para enlazar el entorno de Netlify con GitHub:



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * angeldelucas / Repository name *

⚠ New repository name must not be blank

Great repository names are short and memorable. Need inspiration? How about [studious-octo-waddle](#)?

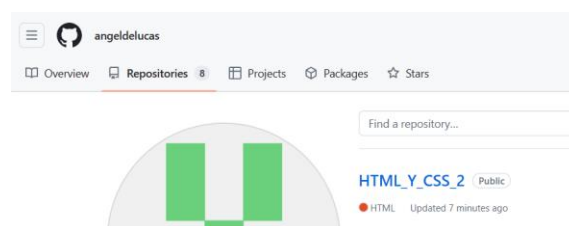
Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

NOTA: Al crear el repositorio hacerlo con la opción de Public seleccionada para permitir el acceso a terceros.

2. Clicar en Create Repository y a continuación podremos visualizar el nuevo repositorio en nuestro perfil.



3. A continuación, debemos clonar el repositorio en nuestro entorno local para trabajar en el y poder controlar el código. Para esto en un directorio en nuestro ordenador ejecutamos el siguiente comando:

git clone https://github.com/angeldelucas/HTML_Y_CSS_2.git

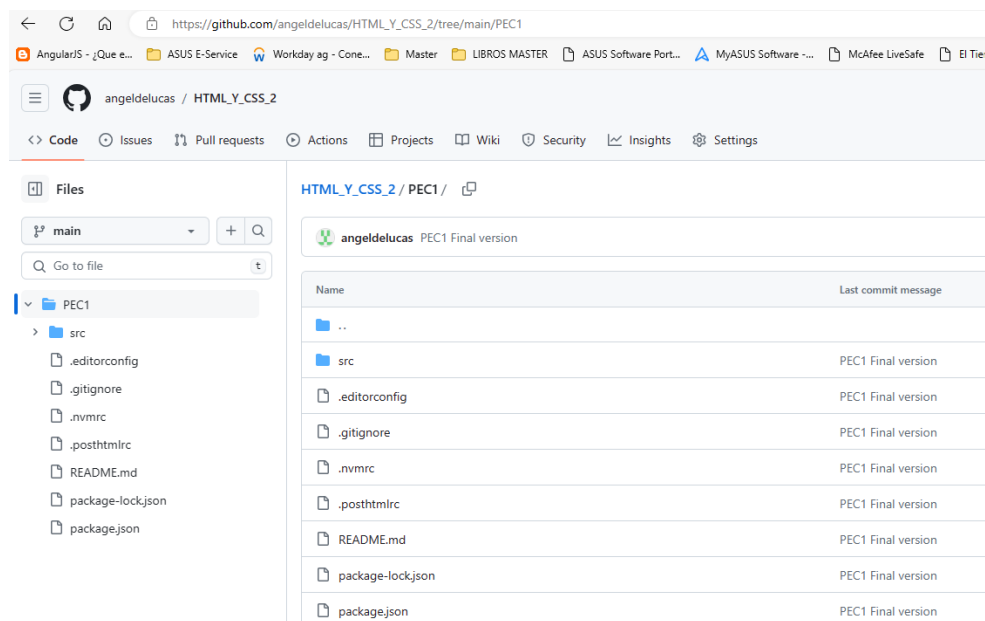
Después ya podremos añadir código e ir subiendo nuestros cambios al nuevo repositorio, para esto debemos ejecutar los siguientes comandos:

git add .

git commit -m “comentario”

git push

Al final del proceso en el repositorio podremos ver nuestros últimos cambios:



El enlace al repositorio en GitHub es el siguiente:

- https://github.com/angeldelucas/HTML_Y_CSS_2.git

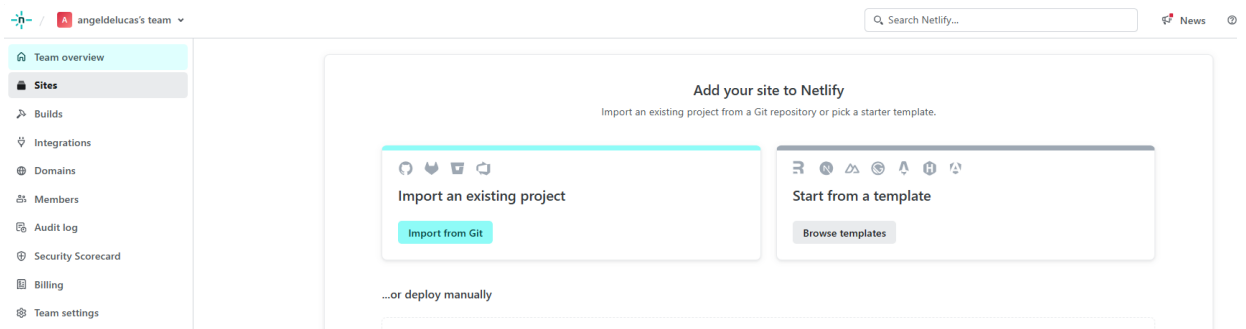
5. PUBLICACIÓN DE LA WEB

Netlify

Después de añadir nuestro código al repositorio creado en GitHub ya tenemos todo preparado para poder publicar nuestra página a través de la herramienta Netlify. Para ello debemos crear un perfil y configurar las opciones para desplegar la web.

1. Al abrir Netlify clicamos en Deploy to Netlify y a continuación clicamos Sing up with GitHub para enlazar Netlify con nuestros repositorios de GitHub.

2. En la opción Sites crearemos el nuevo sitio importando un proyecto desde GitHub. Para ello clicamos Import form Git.



3. Opción Deploy with GitHub. Se abrirá una vista con los repositorios que tengamos en la cuenta que hemos enlazado y seleccionamos el que hemos creado para esta PEC.

Let's deploy your project.



Get started another way

[Try Netlify Drop](#) or [choose a template](#)

4. Configuramos los parámetros necesarios para desplegar la web.

Review configuration for HTML_Y_CSS_2

Deploy as angeldelucas on angeldelucas's team team from main branch using parcel build src/index.html --target web --no-source-maps --no-cache command and publishing to dist

Team
angeldelucas's team ▼

Site name
descubremondejar
https://descubremondejar.netlify.app
[Check availability](#) ✔ Site name is available

Branch to deploy
main ▼

Build settings

Specify how Netlify will build your site.

[Learn more in the docs](#) ↗

Base directory

The directory where Netlify installs dependencies and runs your build command.

Build command

Examples: jekyll build, gulp build, make all

Publish directory

Examples: _site, dist, public

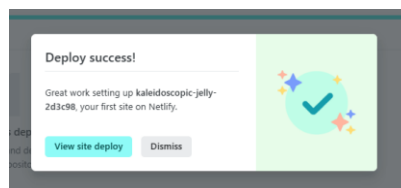
Functions directory

Examples: my_functions

Add environment variables

Deploy descubremondejar

5. Clicamos Deploy HTML_Y_CSS_2 para desplegar el sitio y esperamos hasta ver un mensaje avisándonos de que el proceso ha finalizado.



6. Al final del proceso podremos ver el estado del sitio:

- Site overview
- Site configuration
- Deploys**
- Logs
- Integrations
- Metrics
- Domain management
- Forms
- Blobs

Upgrade

Published deploy for descubremondejar Permalink

Today at 1:51 PM
Production: main@HEAO [Download](#)

[Open production deploy](#) [Lock to stop auto publishing](#) [Options](#)

Deploy summary

- 2 new files uploaded
1 generated page and 1 asset changed.
- No redirect rules processed
This deploy did not include any redirect rules. [Learn more about redirects](#)
- No header rules processed
This deploy did not include any header rules. [Learn more about headers](#)
- No functions deployed
This deploy did not include any functions. [Learn more about Netlify Functions](#)
- No edge functions deployed
This deploy did not include any edge functions. [Learn more about Edge Functions](#)
- Build time: 46s, Total deploy time: 46s
Build started at 1:52:18 PM and ended at 1:53:02 PM. [Learn more about build minutes](#)

Deploy log [Preview](#) [Full](#) [Up](#) [Down](#) [Maximize log](#)

> Initializing	Complete
> Building	Complete
> Deploying	Complete
> Cleanup	Complete
> Post-processing	Complete

Deploy file browser [Search for files in this deploy](#)

about 1

De este modo, cuando hagamos cualquier cambio en el código alojado en el repositorio, al ejecutar el comando git push, netlify recompilará la web y podremos ver el cambio que hemos realizado aplicado a nuestra página.

7. Ya tendremos configurado nuestro sitio web y será totalmente accesible desde cualquier dispositivo. En mi caso la URL del sitio es la siguiente:

URL: **<https://descubremondejar.netlify.app>**

6. PROPIEDAD INTELECTUAL

La información mostrada en el sitio web (fotos y textos) está obtenida de diversos sitios web listados a continuación, en el footer de la página se incluyen los enlaces a estas páginas dentro del link de Aviso Legal:

<https://zascandileando.com/guadalajara/mondejar/>

https://es.wikipedia.org/wiki/La_Alcarria

<https://cultura.castillalamancha.es/patrimonio/catalogo-patrimonio-cultural/iglesia-parroquial-de-mondejar>

<https://mondejar.eu/>

https://es.wikipedia.org/wiki/Viaducto_de_Mond%C3%A9jar

[https://es.wikipedia.org/wiki/Convento_de_San_Antonio_\(Mond%C3%A9jar\)](https://es.wikipedia.org/wiki/Convento_de_San_Antonio_(Mond%C3%A9jar))

[https://es.wikipedia.org/wiki/Ermita_de_San_Sebasti%C3%A1n_\(Mond%C3%A9jar\)](https://es.wikipedia.org/wiki/Ermita_de_San_Sebasti%C3%A1n_(Mond%C3%A9jar))

7. ENLACES

- URL de la web publicada en Netlify
 - o **<https://descubremondejar.netlify.app/>**
- URL del repositorio en GitHub
 - o **https://github.com/angeldelucas/HTML_Y_CSS_2.git**