

HTML Y CSS II

# PEC2

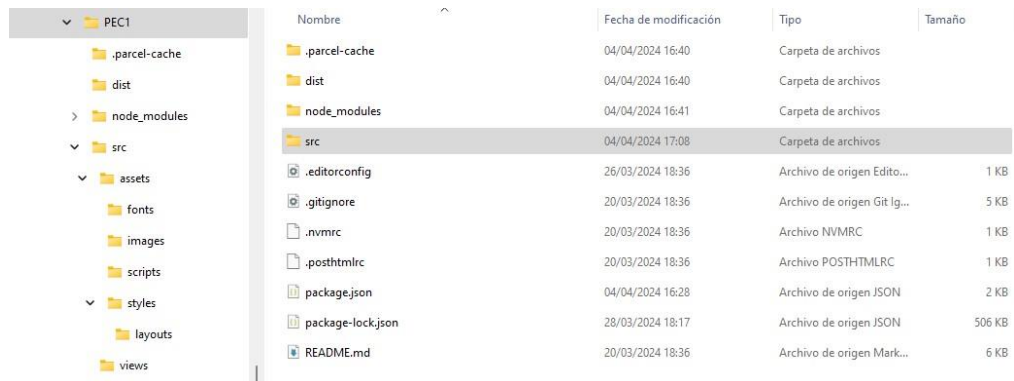
ÁNGEL DE LUCAS GIL

---

## 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:

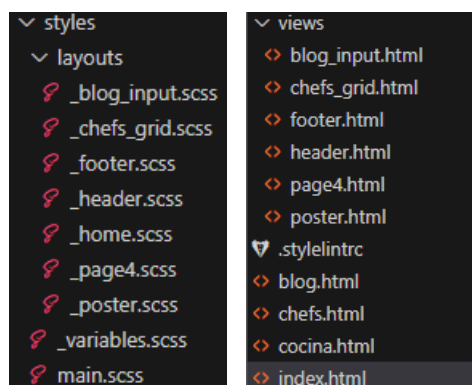


Nombre	Fecha de modificación	Tipo	Tamaño
.parcel-cache	04/04/2024 16:40	Carpeta de archivos	
dist	04/04/2024 16:40	Carpeta de archivos	
node_modules	04/04/2024 16:41	Carpeta de archivos	
src	04/04/2024 17:08	Carpeta de archivos	
.editorconfig	26/03/2024 18:36	Archivo de origen Edito...	1 KB
.gitignore	20/03/2024 18:36	Archivo de origen Git Ig...	5 KB
.npmrc	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.
3. Procesadores de código JavaScript.

El desarrollo de esta PCE2 se basa en la creación de 4 vistas que componen la página web del concurso de cocina. Cada una de ellas se divide en header, body y footer, los cuales a su vez están implementados en diferentes ficheros HTML parciales, los cuales son independientes y cuentan con su propia hoja de estilos. De este modo, la estructura de carpetas del proyecto queda como se muestra a continuación:



styles	views
layouts	blog_input.html
_blog_input.scss	chefs_grid.html
_chefs_grid.scss	footer.html
_footer.scss	header.html
_header.scss	page4.html
_home.scss	poster.html
_page4.scss	.styleintrc
_poster.scss	blog.html
_variables.scss	chefs.html
main.scss	cocina.html
	index.html

En este caso hay 4 ficheros .html principales los cuales se corresponden con las 4 vistas que debe tener la web y la estructuras de todos ellos sigue el mismo patrón (header, body, footer) como se puede ver en la siguiente imagen, en este caso corresponde con la vista de los chefs:

```
<!DOCTYPE html>

<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <link rel="stylesheet" type="text/css" href="./assets/styles/main.scss"><!-- do not remove this line! -->
    <title>Cook Master - 2024</title>
  </head>
  <body>

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

    <script type="module" src="./assets/scripts/main.js"></script><!-- do not remove this line! -->
  </body>
</html>
```

El resto de ficheros html contienen el código que implementa cada vista y están incluidos, a su vez, en los ficheros principales.

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 "some-node-module";
// @import "@some-company/some-node-module";
@import "npm:@fortawesome/fontawesome-free/css/all.css";
@import "bootstrap/scss/bootstrap";
@import ".././../node_modules/bootstrap-icons/font/bootstrap-icons.min.css";

/**
 * Import layouts
 * It's a best practice to use a different partial for each page
 */

@import "layouts/home";
@import "layouts/header";
@import "layouts/footer";
@import "layouts/poster";
@import "layouts/chefs_grid";
@import "layouts/blog_input";
@import "layouts/page4";
```

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. Además, también se incluyen las dependencias añadidas, en este caso fontawesome, Bootstrap y Bootstrap icon.

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 (**Bloque\_\_Elemento--Modificador**). 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:** en este caso se añaden las nuevas unidades para el view port dinámico vista en la PEC2.

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

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

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

- o Bordes: px.
- o 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.

Como excepción para algunas reglas de estilo he marcado el css con las etiquetas “stylelint-disable/enable” para que stylelint ignore ciertas definiciones. Por ejemplo en el fichero `_variables.scss` se puede ver un ejemplo:

```
@mixin width($list) {  
  @each $element in $list {  
    @media (min-width: #{$element}) {  
      /* stylelint-disable */  
      width: calc-width($element, 48%);  
      /* stylelint-enable */  
    }  
  }  
}
```

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.

Como dependencia externa he añadido **fontawesome, Bootstrap y Bootstrap icon**, que permite incluir iconos prediseñados en nuestra aplicación. Para instalar Bootstrap es necesario ejecutar los siguientes comandos para añadir las dependencias:

Bootstrap: **npm install @popperjs/core bootstrap --save**

Librería de iconos: **npm i bootstrap-icons.**

Para usarlas, en el fichero main.css debemos incluir la siguiente declaración:

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

```
@import "bootstrap/scss/bootstrap";
```

```
@import "../..../node_modules/bootstrap-icons/font/bootstrap-icons.min.css";
```

Y en el fichero main.js:

```
import * as bootstrap from 'bootstrap';
```

### 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 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.

Cabe destacar que se ha realizado un diseño distinto del poster de la vista principal para navegadores que no soporten el uso de grid.

En la vista de contenido libre he implementado una especie de galería con los ganadores de los últimos concursos. Un “hall of fame” donde aparece en primer lugar un contenedor con las fotos de los ganadores y si se pulsa o se pasa el ratón por encima muestra cierta información que se puede ampliar al clicar en el botón que aparece con la imagen. Si se clica, la pantalla se mueve hacia abajo mostrando una imagen con la receta ganadora y una sección con una breve descripción y un vídeo de la elaboración. **NOTA:** EN ESTE APARTADO EL CONTENIDO DE LOS VÍDEOS ESTÁ RELACIONADO CON LA ELABORACIÓN DE ALIMENTOS, PERO NO CORRESPONDEN CON LA RECETA.

Los cuatro componentes empleados de Bootstrap son:

1. Botón
2. Card
3. Breadcrumb
4. Paginador.

## 4. CSS

En este caso se emplea SASS haciendo uso de variables, funciones, nesting parciales e importación. Además de estos elementos se hace uso de las psedoclases `:is`, `:where` y `:has`, así como de las container queries y la organización de las reglas en `@layers`.

1. **@layer.** He empleado la organización de las reglas en capas en cascada en la vista de blog (`_blog_input.scss`) y en la página de contenido libre (`_page4.scss`). En el caso de la página para la entrada de blog se puede apreciar que hay dos capas definidas llamadas *mobile* y *desktop*, en la primera se puede ver una regla con una especificidad mayor que en la segunda capa, pero según están definidas las prioridades para ambas capas las reglas de la capa más prioritaria se aplican pisando las reglas definidas en la de menor prioridad.

```
@layer mobile, desktop;

@layer mobile {
  :is(.input_recents-articles) {
    margin-top: 10%;
    display: flex;
    flex-direction: column;

    >h2 {
      top: -5%;
    }

    >section:nth-child(3) {
      margin-right: 2%;
      margin-left: 2%;
    }
  }
}

@layer desktop {
  @media (min-width: 576px) and (max-width: 768px) {
    :where(.input_recents-articles) {
      flex-direction: row;

      >h2 {
        top: -20%;
      }
    }
  }
}
```

En la página de contenido libre se organizan las reglas de estilos en capas para cambiar los estilos de varios elementos cuando están o no seleccionados. En este caso he definido dos capas, una que se aplica por defecto a los elementos cuando arranca la aplicación y otra para los elementos seleccionados por el usuario. Para ello, el usuario puede clicar sobre ciertos elementos y con código JavaScript se añade una nueva clase a la lista de clases del elemento.

2. **Pseudoclases.** He empleado las pseudoclases en varias partes del proyecto como por ejemplo en el fichero `_home.scss` para definir estilos para elementos que aparezcan en cualquier punto del sitio web. En este fichero se puede ver un ejemplo de uso de `:where` e `:is`. Para ilustrar el funcionamiento de ambas defino con `:where` estilos para los elementos `p` y `li`, a continuación defino dentro de otra regla que solo aplica a los elementos `figure` un estilo propio para los `p` dentro de los `figure`, en este caso lo hago con `:is` para lograr que esta regla pise a la anterior definida con `:where` puesto que `:is` tiene mayor especificidad que `:where` (`:where` anula cualquier especificidad). En definitiva, usando ambas reglas podemos definir estilos generales y aplicar excepciones en los puntos que nos interese.



```

:where(p, li) {
  font-size: 1.2rem;
}

:is(figure) {
  position: relative;

  :is(img) {
    width: 100%;
  }

  :is(figcaption) {
    @include author();
  }
}

~:is(p) {
  font-weight: bold;
  font-size: 1.4rem;
}

~* {
  padding: 0% 2%;
}

```

Nótese que en la vista blog (`_blog_input.scss`) he maquetado todo el contenido organizando las reglas con pseudoclases puesto que era una vista sencilla con poca variedad de elementos, mostrando de este modo, el potencial de estas herramientas que ofrece css.

En la vista principal (`_psoter.scss`) he empleado la pseudoclase `:has` para el `:hover` sobre las fotos del grid. Con `:has` detecto que elementos dentro del grid no tienen `hover` activo. Además, esta pseudoclase se emplea en más puntos del proyecto.

3. **Container queries.** En las 4 páginas del sitio web se aplican container queries para dimensionar las vistas en función del tamaño del elemento seleccionado. En todas las vistas donde he empleado container queries he definido un elemento que actúa como referencia para distribuir el contenido en función del width de dichos elementos de referencia.
4. **Variables y funciones:** se definen variables con los nuevos formatos de colores y se incluye una función para el cálculo de ancho en función del ancho de la pantalla. Esto se incluye en el fichero `_variables.scss`.
5. **Nuevas unidades.** En la página de contenido libre se emplean las nuevas unidades (`dvh`) para adaptar el tamaño de una imagen al tamaño del view port en función de si se muestra o no la barra de navegación en el teléfono móvil.
6. **Bootstrap.** A lo largo de todo el código html he incluido reglas predefinidas en Bootstrap para y en el fichero `_variables.scss` se incluye la modificación de parámetros de los cuatro componentes a través de variables. Cabe destacar que para el elemento botón he empleado un `@mixin` predefinido en Bootstrap para personalizar botones.

## 5. DECISIONES DE DISEÑO

Para la página de contenido libre he implementado la selección y mostrado de imágenes y vídeos con JavaScript. El código es algo muy sencillo y es una funcionalidad que tenía bastante complejidad sin incluir JavaScript.

He combinado la guía de estilos junto con clases predefinidas de Bootstrap, las segundas incluidas directamente en el HTML.

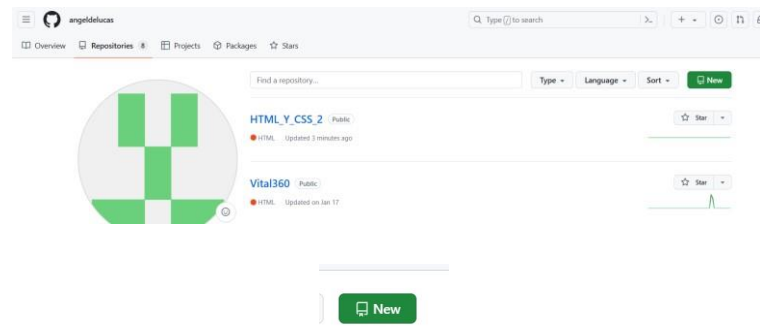
El enlace “Aviso Legal” del footer abre una vista con los links a los sitios donde he sacado la información.

## 6. CONTROL DEL CÓDIGO

### GitHub

Para el desarrollo de la PEC2 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](#)?

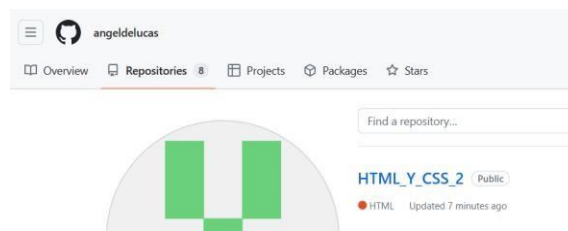
Description (optional)

☒ **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.

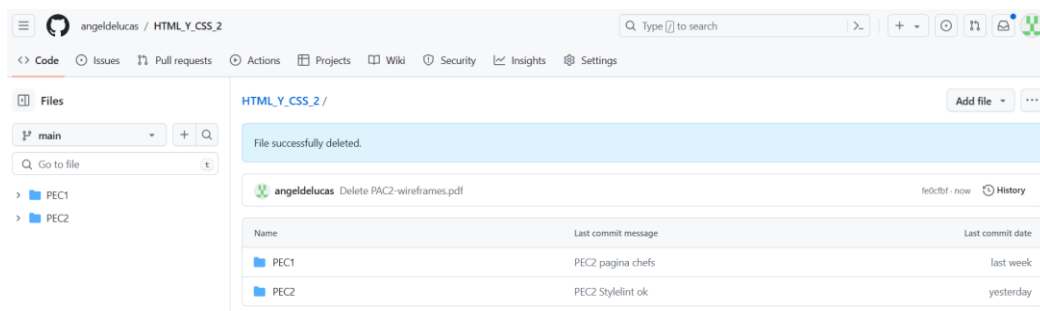


3. A continuación, debemos clonar el repositorio en nuestro entorno local para trabajar en el y poder controlar el código. Para esto en un directorio en nuestro ordenador ejecutamos el siguiente comando:

**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**

En este caso el repositorio ya lo tenía creado de la PEC1 y la vista actual del mismo contiene ambas PEC's



El enlace al repositorio en GitHub es el siguiente y dentro del mismo se encuentra la PEC1 y la PEC2:

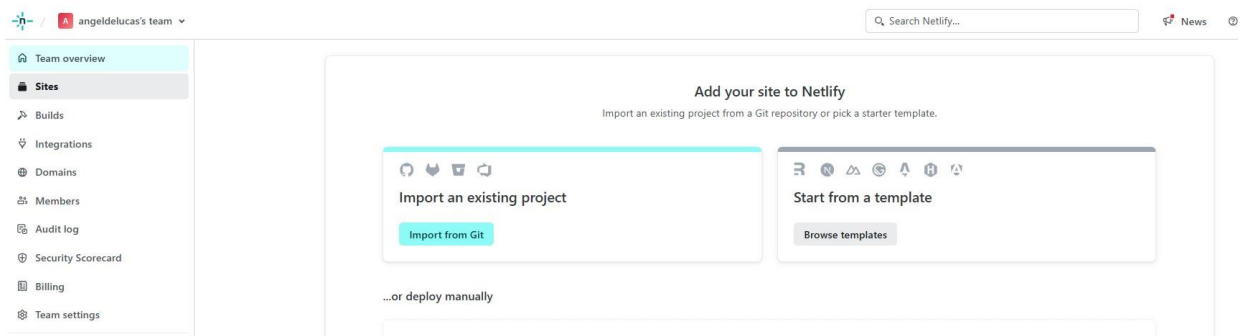
- [https://github.com/angeldelucas/HTML\\_Y\\_CSS\\_2.git](https://github.com/angeldelucas/HTML_Y_CSS_2.git)

## 7. 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.



- 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

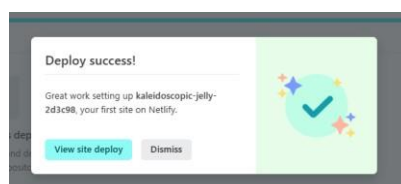
[Try Netlify Drop](#) or [choose a template](#)

- Configuramos los parámetros necesarios para desplegar la web.

## Build settings

Runtime:	Not set
Base directory:	PEC2
Package directory:	Not set
Build command:	parcel build src/index.html --no-source-maps --no-cache
Publish directory:	PEC2/dist
Functions directory:	PEC2/netlify/functions
Deploy log visibility:	Logs are public
Build status:	Active

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



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.

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

URL: **cookmasterangel.netlify.app**

## 8. 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:

- Logos e imágenes: <https://www.freepik.es/>
- Textos incluidos en la vista blog generados con ChatGPT.
- Primera imagen de la vista blog generada con CopilotIA.

## 9. ENLACES

- URL de la web publicada en Netlify
  - o **<https://cookmasterangel.netlify.app>**
- URL del repositorio en GitHub
  - o **[https://github.com/angeldelucas/HTML\\_Y\\_CSS\\_2.git](https://github.com/angeldelucas/HTML_Y_CSS_2.git)**