PEC 3. Herramientas HTML y CSS II aula 1

Ignacio Casares Ruiz

Junio de 2023

El enlace al respositorio de github utilizado es el siguiente: https://github.com/nacaru-w/h-II-P3

La página web subida en Netlify puede consultarse aquí: https://main–famous-custard-f63314.netlify.app/

Instalación del boilerplate y creación del repositorio

Al igual que en las PECs anteriores, el primer paso que se llevó a cabo en la realización de esta PEC fue la instalación de UOC boilerplate.

Se creó un nuevo repositorio git local y lo vinculé a un repositorio remoto asociado a mi cuenta de Github. Este se puede consultar a través de la siguiente URL:

https://github.com/nacaru-w/h-II-P2

Esto permitiría tener un sistema de control de cambios, pudiendo revertir a un estado anterior en caso de necesitarlo. La habilitación de un repositorio público es necesaria para la publicación de la página a través de Netlify.

A continuación, instalé las dependencias del boilerplate a través de `npm` mediante el comando de terminal `npm install`. Se usaron las mismas dependencias que en la PEC anterior, eliminándose, además, `stylelint` y `bootstrap`. Después, se instaló Tailwind mediante el siguiente comando:

```
npm install tailwindcss --save-dev
```

Se añadió, por cuestiones de hábito, un script para el comando `npm run start`, que realiza las mismas funciones que `npm run build` en el archivo `package.json`.

Aunque para esta práctica no se utilizó una guía de estilo, se siguió usando el linter de código personalizado instalado localmente en mi dispositivo. Este realiza una serie de modificaciones (cambia aspectos como el indentado, el número de saltos de línea, etc) cada vez que realizo un guardado en la aplicación VSCode.

Configuración de Tailwind

Se comenzó por crear un archivo de configuración de Tailwind, esto se realizó a través del siguiente comando en la consola:

```
npx tailwindcss init
```

Una vez creado el *boilerplate* del archivo de configuración, se procedió a introducir los colores que se habían asignado en la PEC2 a variables SCSS al archivo `tailwind.config.js`. Para tal fin se configuró el objeto `colors` de la siguiente manera:

```
module.exports = {
  content: ['./src/**/*.{html,js}'],
  theme: {
    colors: {
      transparent: 'transparent',
      current: 'currentColor',
      black: colors.black,
      white: colors.white,
      gray: colors.gray,
      emerald: colors.emerald,
      indigo: colors.indigo,
      yellow: colors.yellow,
      'normal-bg-color': 'rgb(245 245 245)',
      'text-color': 'rgb(31 31 31)',
      'secondary-contrast-color': 'rgb(204 204 204)',
      'link-color': 'rgb(51 51 51)',
      'colorful-element-color': 'rgb(248 221 201)',
      'lighter-colorful-element-color': 'rgb(254 248 244)',
      'colorful-element-border-color': 'rgb(253 245 239)',
    }
  }
}
```

Se puede observar como se añadieron las variables usadas en los archivos `blog.html` y `members.html` (estas aparecen en las posiciones finales del objeto). También se añadieron colores nativos de Tailwind, con el objetivo de usarlos puntualmente para ciertos tonos fuera de la temática de colores de la página web. Para poder acceder a ellos, hubo de importarse el módulo `colors`:

```
const colors = require('tailwindcss/colors')
```

Asimismo, se configuraron las dos fuentes utilizadas en la tipografía de la pagína web. Estas se configuraron como parte del objeto `FontFamily`, de la siguiente manera:[1]

```
fontFamily: {
  normal: ['JetBrains Mono', 'monospace'],
  serif: ['EB Garamond', 'serif']
}
```

[1] La fuente EbGaramond se añadió porque se utilizaba en la portada del proyecto. Al no incluirse la portada en las dos páginas incluidas en esta práctica, no se hizo ningún uso práctico de ella en la elaboración de esta PEC.

También se configuró un valor estándar (`normal`) para la propiedad `border-radius`, con el objetivo de proporcionar estandarización a este tipo de estilo de bordes.

```
extend: {
  borderRadius: {
    'normal': '1em'
  }
}
```

Además, se realizaron las importaciones necesarias en el archivo `main.scss` para que la herramienta funcione correctamente:

```
/** Tailwind */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Configuración de Postcss

Tras su instalación, Tailwind no funcionaba en el proyecto. Esto fue así porque se debía de configurar como un plugin de Postcss y no como una dependencia aislada. Para realizar tal acción, se creó un archivo JSON `.postcssrc`. En este, se especificó la inclusión de Tailwind como plugin.

```
{
    "plugins": {
        "tailwindcss": {},
    }
}
```

Esto permitió el buen funcionamiento de la librería de utilidades.

Configuración de VSCode

Con el objetivo de prevenir los errores generados por VSCode ante la inclusión de *at-rules* propias de Tailwind como `@apply`, `@layer` y `@tailwind`, se modificó la configuración de VSCode para ignorar este tipo de inclusiones en el código SCSS. Esto se realizó a través de la modificación de las preferencias, cambiando las preferencias sobre actuación de SCSS ante reglas *at-rule* no conocidas de «warning» a «ignore».

Una vez realizado el cambio, VSCode no alertó de ninguna *at rule* desconocida.

Además, adicionalmente y como ayuda ante la redacción del código relativo a las *utilities* de Tailwind, se instaló la extensión de VSCode Tailwind CSS IntelliSense. Esta incluye:

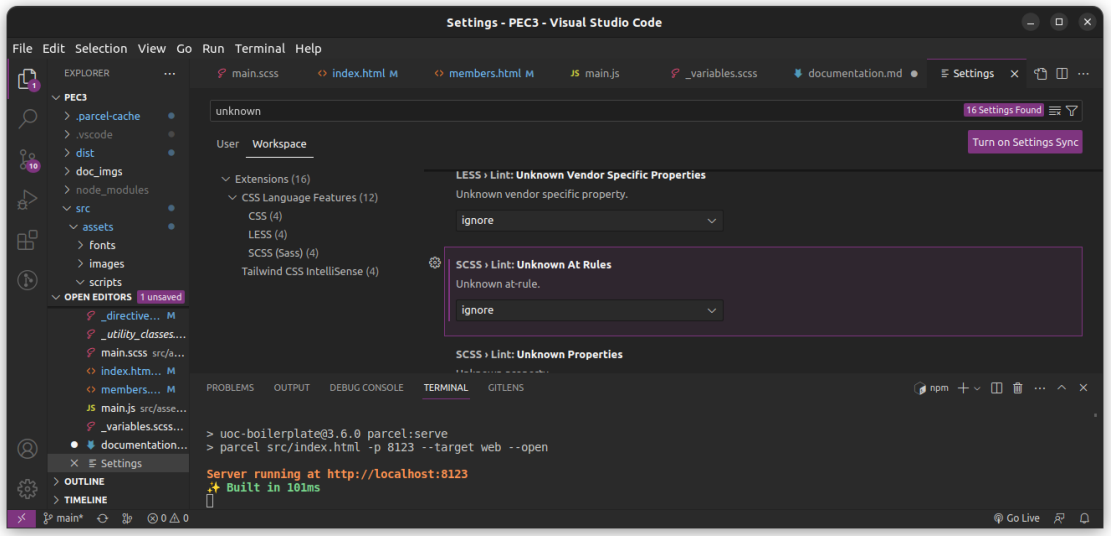

Imagen 1: menú en el que se configuran las alertas para las _at rules de VSCode

- Autocompletado de código relativo a Tailwind.
- Linteo, con resaltación de errores.
- Previsualización al realizar *hover* sobre el código.
- Modo de lenguaje Tailwind CSS.

La herramienta fue especialmente útil en la práctica, especialmente la opción de *hover* sobre el código, ya que te permite conocer el equivalente CSS al código redactado como *utility* de Tailwind.

Modificaciones en el sistema de ficheros

Se relizaron algunas modificaciones al sistema de ficheros respecto al del proyecto original. Se renombró la página SCSS en la que se situaba la hoja de estilos principal a `_directives.scss`, ya que es este tipo de contenido el que incluye en el proyecto actual. Se eliminaron los archivos de las páginas que no se recrearían, ya que el enunciado de la práctica requería la elaboración de dos de ellas. Se renombró `blog.html` a `index.html` para colocarla como página principal.

Preguntas específicas sobre utility-first CSS

- ¿Qué diferencias hay entre el enfoque de tipo CSS semántico (el que usaste en las otras PEC) y el CSS de utilidades? ¿Cómo afectó esto a tu proceso de desarrollo? ¿Y a tu código?

Se trata de enfoques muy diferentes. En el primer caso, aplicas los estilos mediante selectores CSS que encuentran elementos, clases, ids, etc a los que adjudicar las propiedades y sus valores descritos en las reglas. En el enfoque por utilidades, se utilizan sets preconfigurados por Tailwind (o personalizados) y estas se aplican a cada elemento en forma de clases. En casos en los que no resulta eficaz hacerlo así, existe la aplicación mediante la directiva `@apply`, que utiliza un enfoque más parecido al semántico.

Para mí esto supuso un enorme cambio de enfoque de desarrollo. Por una parte, el enfoque atómico permite aplicar la mayoría de estilos *on the go* a la vez que va describiéndose el código HTML, lo cual puede ser aventajoso a la hora de ahorrar tiempo. Por otro lado, haber utilizado Tailwind implica haber tenido que familiarizarse con la nomeclatura de sus utilidades, por lo que, de alguna forma, ha supuesto tener que volver a aprender

- ¿Qué diferencias encontraste entre usar una librería de componentes y una librería de utilidades?

Son conceptos ampliamente distintos. Una librería de componentes te construye elementos con funcionalidad propia que llamamos componentes, es decir, te construye elementos total o parcialmente, y esto puede incluir el código HTML, los estilos e incluso el javascript asociado. Una librería de utilidades, en este caso Tailwindcss, incluye formas predefinidas de aplicar estilos (hay otras librerías de utilidades que aportan funciones predefinidas, por ejemplo). En general, el sentido de una librería de utilidades es más ligero, con un uso más específico y complementario; por otro olado, una librería de componentes te da las piezas que componen la web ya creadas, y uno puede decidir si modificarlas o no a *posteriori*. Podríamos resumirlo en que las librerías de utilidades proporcionan herramientas y las de componentes aportan bloques o piezas de contenido.

- ¿Qué clases y componentes decidiste extraer y por qué?

Las clases extraidas se encuentran descritas exhaustivamente, incluida su justificación, en el desarrollo del código que se explica en el apartado posterior. De forma resumida, podemos afirmar que el criterio principal para extraer las clases y aplicarlas mediante directiva fue la existencia de un elemento que se repite con frecuencia a lo largo de la web,

y/o la presencia de una cantidad significativa de propiedades CSS que deben aplicarse a un solo elemento.

En cuanto a los componentes, se eligieron el set de reglas descrito en la página principal y los blogs de previsualización hallados al final de la página, ambos en `index.html`. La principal razón por la que se extrajeron estos componentes es porque puede considerarse que son de carácter cambiante, por lo que acceder individualmente a ese código permite realizar sus modificaciones frecuentes de manera más sencilla.

Recreación de la página web

Una vez realizada la instalación de las dependencias, se procedió a recrear los archivos `index.html` (antes `blog.html`) y `members.html`.

Ya que la práctica requería un enfoque basado en Atomic CSS, con el objetivo de estandarizar la redacción de la misma, se tomaron las siguientes decisiones antes de dar comienzo a la redacción del código:

- A la hora de desarrollar el código relativo a elementos únicos, se realizaría un enfoque atómico puro, es decir, realizando adiciones de estilo a través del atributo `class` dentro de la línea de código HTML. Dependiendo de la complejidad del elemento y su situación respecto al resto de elementos de la página, en ocasiones de dejó la clase original aún cuando esta no posee ningún estilo asignado a la misma. Esto se hizo de tal forma con el objetivo de identificar más fácilmente los elementos en el código.
- En cuanto al desarrollo de elementos que se repiten en la página web o que poseen una amplia cantidad de estilos, este se realizó a través de un abordaje mediante directivas. En concreto, estas se desarrollarían en el archivo `_directives.scss`, a través de las `at-rules`: `@layer`, que sirve para identificar la función del elemento al que se le aplican los estilos dentro del código, pudiendo ser este `@component` o `@base`; y `@apply`, que se usa para introducir las propiedades y sus valores.

Para el desarrollo del proyecto en su totalidad, se hizo uso de la página a la que se había subido el proyecto de la PEC2. Mediante el inspector de estilos, se identificaron las reglas CSS aplicadas a los diferentes elementos y, haciendo uso de la extensa descripción de las utilidades proporcionada en la página de documentación de Tailwind. Se aplicaron los estilos originales mediante Tailwind con la mayor fidelidad posible. Téngase en cuenta, también, que los pocos estilos que resultaban ininplementables o que daban error a través de Tailwind fueron descritos mediante código CSS.

Desarrollo en Tailwind de encabezado y pie de página

Al ser ambos elementos comunes a las dos páginas, estos se desarrollarán en esta sección individual.

Para el desarrollo del encabezado, Se sustituyeró el código original por clases Tailwind. Se dio preferencia a las unidades nativas de Tailwind, pero cuando estas no proporcionaban

un resultado satifactorio, se hizo empleo de *bracket notation* para establecer la unidad original. Por ejemplo:

```
<div class="mx-auto w-[90%] max-w-[1000px] flex justify-between
↪  items-center">
```

Se eliminó el código relativo a la apertura y cierre del menú de navegación. Esto se realizó así porque, al haber ahora solo dos opciones, el menú permanece visible y sin problemas de formato incluso cuando la página web se visualiza a través de dispositivos de menor tamaño.

```
<nav class="z-950" id="primary-navigation">
  <ul class="flex" role="list">
    <li class="menu-item"><a href="members.html">Members</a></li>
    <li class="menu-item"><a href="index.html">Blog</a></li>
  </ul>
</nav>
```

La sección de navegación principal del sitio web está ahora contenida dentro de un elemento `nav` con una clase `z-950`, que establece un índice que asegurar que la navegación se muestre por encima de otros elementos. El nav contiene una lista desordenada `<ul>` con una clase `flex`, que utiliza Flexbox para alinear los elementos en una fila. Cada elemento de la lista `<li>` tiene una clase `menu-item` y contiene un enlace `<a>` que lleva a las dos páginas del sitio web. A la clase `menu-item` se le aplica la directiva siguiente, mediante `@apply`:

```
.menu-item {
  @apply px-8;
}
```

En cuanto al pie de página, al componerse este de elementos mayormente únicos, se tomó un enfoque similar:

```
<footer class="bg-secondary-contrast-color py-8 text-[.9em]">
```

Aquí se establece la clase `bg-secondary-contrast-color` que se aplicará al fondo del pie de página y la clase `py-8` para agregar un *padding* vertical de 8 unidades. También se establece el tamaño del texto en 0.9em mediante *bracket notation*, ya que Tailwind no posee una unidad nativa que se corresponda a tal cantidad de em.

```
<div class="w-[90%] max-w-[1000px] justify-around my-0 mx-auto pt-4 flex
↪  items-center">
```

Este div posee el contenido principal del pie de página. Se utiliza la clase `w-[90%]` para establecer un `width` del 90% del contenedor principal. La clase `max-w-[1000px]` establece un ancho máximo de 1000 píxeles. En el proyecto original, ambas clases se correspondían a una única propiedad de función CSS `min(90%, 1000px)`, pero al no

admitirse en Tailwind, esta se tuvo que desarrollar así. Las clases `justify-around`, `my-0`, `mx-auto` y `pt-4` se utilizan para ajustar el diseño del contenido del pie de página. La clase `flex` crea un contenedor flexible que alinea los elementos horizontalmente.

```
<div class="px-2">
```

Este div contiene el contenido del lado izquierdo del pie de página. Se utiliza la clase `px-2` para agregar un relleno horizontal de 2 unidades, equivalente a `0.5em`.

```
<div class="button-box">
  <button class="button hover:bg-colorful-element-color type="
  ↪  button">Contact</button>
</div>
```

Aquí se encuentra un contenedor con la clase `button-box` que envuelve un botón. El botón tiene las clases `button` y `hover:bg-colorful-element-color` para aplicar estilos específicos y cambiar su fondo al pasar el cursor por encima.

```
<div class="py-4">
  <p class="my-4">
    Download our app
  </p>
  <div class="justify-evenly flex">
    <!-- Aquí se hallan los iconos -->
  </div>
</div>
```

Este div contiene un párrafo con la clase `my-4` para agregar un margen vertical de 4 unidades, lo que se corresponde a `1em`. Dentro del div hay otro div con las clases `justify-evenly` y `flex`, que se utilizan para alinear y distribuir equitativamente los elementos de su interior.

```
<div class="right-side-footer">
  <p class="font-bold my-4">Information</p>
  <div class="col3">
    <ul>
      <!-- Aquí se hallan los elementos de la lista -->
    </ul>
  </div>
</div>
```

En este div se encuentra el contenido del lado derecho del pie de página. El párrafo tiene la clase `font-bold` para aplicar un estilo de fuente en negrita. Dentro del `div` hay otro `div` con la clase `col3` que contiene una lista desordenada (`ul`).

```
<div class="sub-footer w-[90%] max-w-[1000px] border border-t-text-color
↪  border-b-transparent border-x-transparent flex justify-between mt-8
↪  mx-auto mb-0">
```

En este div se define la sección inferior del pie de página, que se coloca debajo del contenido principal. Se utilizan las clases `w-[90%]` y `max-w-[1000px]` para establecer el ancho de la sección. Además, se agrega un borde superior mediante las clases `border`, `border-t-text-color`, `border-b-transparent` y `border-x-transparent`. La clase `flex` se utiliza, de nuevo, para alinear los elementos del mismo horizontalmente, y la clase `justify-between` los separa `equitativamente`.

En el interior de este `div`, se halla el siguiente código:

```
<div class="logo-in-footer">
  <a href="index.html">
    <img class="main-logo w-[5em] h-auto"
    ↪  src="./assets/images/main_logo.svg" alt="Main logo of the
    ↪  website">
  </a>
</div>
<div class="pages-in-footer flex justify-center items-center">
  <ul class="flex">
    <li class="after:content-['|'] after:pl-4 pr-[1.25em]"><a
    ↪  href="">Members</a></li>
    <li class="after:content-['|'] after:pl-4 pr-[1.25em]"><a
    ↪  href="">Blog</a></li>
    <li><a href="">Contact & FAQ</a></li>
  </ul>
</div>
<div class="social-media-icons flex justify-evenly w-[8em]">
  <!-- Iconos -->
</div>
```

Que paso a explicar punto a punto:

- `logo-in-footer` contiene el logotipo principal del sitio web y se coloca en el lado izquierdo del subfooter. El logotipo está envuelto en un elemento `a` que conduce a la página principal `index.html`. El logotipo se engloba como elemento con la etiqueta `img`, con la clase main-logo que establece el ancho (`w-[5em]`) y la altura automática (`h-auto`).
- `pages-in-footer` muestra una lista de enlaces de las páginas del proyecto. La lista se coloca en el centro del subfooter utilizando las clases `flex`, `justify-center` e `items-center`, ya desarrolladas en elementos anteriores. Cada enlace de página se representa como un elemento de lista (`li`) y se muestra en línea utilizando la clase

`flex`. Además, se agrega un separador vertical utilizando la pseudo-clase `after` y algunas clases de espaciado (`after:content-['|']`, `after:pl-4` y `pr-[1.25em]`).
- `social-media-icons` describe los íconos de las redes sociales. Los íconos son representados por etiquetas `svg` y se muestran en línea utilizando las clases `flex` y `justify-evenly`. Cada ícono de red social tiene una clase `mx-2` para agregar un margen horizontal de 2 unidades y la clase `cursor-pointer` para establecer el cursor del mouse como un puntero al pasar por encima.

Desarrollo en Tailwind de la parte principal de `index.html`

Para la parte principal de esta página (*header* y *footer* han sido descritos en la sección anterior) se aplicó un elemento `main` que engloba todo el código. A este se le aplicaron las clases relativas al fondo `bg-normal-bg-color` (nótese que aquí se le aplica una clase definida en `tailwind.config.js`) y se estableció como fuente predeterminada para la página la fuente introducida en la configuración mediante `font-normal`.

La sección `checked-background` se configuró a través de una directiva, introducida con `@apply`:

```
.checked-background {
  @apply flex flex-col justify-center items-center text-center h-80
  ↪  w-full bg-[url('../images/backgrounds/checked_background.png')]
  ↪  bg-contain bg-repeat-round;
}
```

Esta aplica las siguientes utilidades:

- `flex`: Establece el contenedor como un *flex container*.
- `flex-col`: coloca los primeros elementos descendientes en una columna vertical dentro del contenedor flex.
- `justify-center`: centra verticalmente los elementos secundarios dentro del contenedor `flex`.
- `items-center`: centra horizontalmente los elementos secundarios dentro del contenedor flex.
- `text-center`: centra el texto dentro de los elementos secundarios.
- `h-80`: establece la altura del contenedor en 80 unidades de Tailwind.
- `w-full`: establece el ancho del contenedor al 100% del ancho disponible.
- `bg-[url('../images/backgrounds/checked_background.png')]`: establece la imagen de fondo del contenedor utilizando la URL proporcionada. En este caso, la URL apunta a la imagen `checked_background.png`, ubicada en la carpeta de imágenes del proyecto.
- `bg-contain`: ajusta el tamaño de la imagen de fondo para que se ajuste completamente dentro del contenedor sin distorsionarla, manteniendo la relación de aspecto original.

- `bg-repeat-round`: repite la imagen de fondo tanto horizontal como verticalmente, creando un efecto de mosaico redondeado.

A continuación se introduce la imagen que contiene el logo del blog, así como el título del mismo y los autores:

```
<div class="w-40 h-auto">
  <img class="w-full h-auto" src="assets/images/blog_logo.svg" alt="">
</div>
<h2 class="m-1 text-2xl font-bold"><span class="say-chess [text-shadow:
↪  2px 2px lightcoral]">Say chess!</span> a
  blog that
  keeps
  you
  in check
</h2>
<span>
  by Aleksandar Srebrenica and
  Catalina Borràs </span>
```

El primer elemento `div` tiene las clases `w-40` y `h-auto`, lo que establece su ancho en 40 unidades y ajusta automáticamente su altura en función del contenido. Sirve como contenedor para la imagen principal del logo.

El elemento `img` tiene las clases `w-full` y `h-auto`, lo que hace que se estire para ocupar todo el ancho de su contenedor.

El elemento h2 tiene las clases `m-1`, `text-2xl` y `font-bold`, lo que establece un margen de 1 unidad alrededor del título, aumenta el tamaño del texto a `2xl` y convierte el texto a negrita.

El elemento `span` incluye un estilo *inline* `text-shadow: 2px 2px lightcoral`. Se ha realizado de esta manera porque Tailwind no posee actualmente una forma de aplicar la propiedad CSS `text-shadow`. Esta propiedad agrega un efecto de sombra al texto, en este caso de color coral claro.

La siguiente sección se encuentra englobada dentro de otro `div`:

La clase `py-16`establece un relleno vertical (*padding*) de 16 unidades en el eje y, lo que crea un espacio adicional arriba y abajo del contenido dentro del contenedor. `W-[90%]` establece el ancho del contenedor en el 90% del ancho de su contenedor padre. En cuanto a `max-w-[1000px]`, establece un ancho máximo del contenedor en 1000 píxeles. Esto evita que el contenedor se vuelva demasiado ancho en pantallas grandes y ayuda a mantener un diseño más legible. La clase `mx-auto` centra el contenedor horizontalmente en su contenedor padre. Aplicando un margen izquierdo y derecho automático, el contenedor se alinea en el centro horizontal de su contenedor padre. La clase `text-text-color` establece el color temático de texto del contenido dentro del contenedor. `leading-6`

establece el espaciado entre líneas del texto dentro del contenedor. En este caso, se establece en 1.5 unidades, lo que aumenta ligeramente la separación vertical entre líneas de texto. Esto mejora la legibilidad del texto al proporcionar un espacio de lectura adecuado.

Respecto al código del rótulo del título, se describió a través de las siguientes clases:

Our rules and policies

La clase `text-center` alinea el texto en el centro horizontalmente, mientras que `text-[1.4em]` establece el tamaño del texto en 1.4 veces el tamaño base de la fuente. La clase `py-8` establece un relleno vertical de 8 unidades en el eje y, y la clase `font-semibold` convierte la fuente en negrita.

El código relativo a la última actualización incluye un centrado de texto a través de `text-center`, y un aumento del *padding* inferior a cuatro unidades.

A continuación, se describe los bloques del lema del club y la introducción:

```
<div class="flex flex-wrap py-4">
  <blockquote class="text-lg flex-auto w-1/4 m-[1.75em]">Our rules can
  ↪  be summed up in our motto: "Play with
    respect,
    compete with fairness."</blockquote>
  <p class="blog-paragraph flex-auto w-2/3">Our rules and policies are
  ↪  an essential component that
    outlines the expectations and regulations that govern the behavior
↪   of club members. It is important
    for
    the chess club to have clear and concise rules to ensure a safe and
↪   enjoyable experience for all
    participants. The following paragraphs will detail some typical
↪   rules and policies found on the
    webpage
    of a chess club for young people.</p>
</div>
```

En este código HTML, se crea un contenedor `<div>` con las clases `flex` y `flex-wrap`, lo que significa que los elementos contenidos se colocarán en una disposición de tipo flexbox y se envolverán en múltiples líneas si es necesario. Dentro del contenedor, hay dos elementos principales.

El primer elemento es un bloque de citas `<blockquote>` con las clases `text-lg`, `flex-auto`, `w-1/4` y `m-[1.75em]`. La clase `text-lg` establece el tamaño de fuente del bloque de citas en grande. Las clases `flex-auto` y `w-1/4` indican que el bloque de citas debe ocupar un ancho flexible y que debe ocupar el 25% del ancho del contenedor. La

clase `m-[1.75em]` establece un margen de 1.75 veces el tamaño de la fuente alrededor del bloque de citas.

El segundo elemento es un párrafo `<p>` con las clases `blog-paragraph`, `flex-auto` y `w-2/3`. Las clases `flex-auto` y `w-2/3` indican que el párrafo debe ocupar un ancho flexible y que debe ocupar el 66.67% del ancho del contenedor.

A continuación se describe el código relativo a las reglas:[1]

```
<div class="rule-set">
  <ol class="py-[1em] border-y-4 border-text-color m-1">
    <li>
      <h3 class="rule-title">1. Respectful behaviour</h3>
      <!-- Aquí una regla -->
    </li>
    <li>
      <h3 class="rule-title">2. Being there is essential</h3>
      <!-- Aquí una regla -->
    </li>
    <li>
      <h3 class="rule-title">3. Conflicts are resolved in civilised
      ↪  terms</h3>
      <!-- Aquí una regla -->
    </li>
  </ol>
</div>
```

Esta parte del código se desarrolló como directivas. La única parte en la que se aplicaron estilos en línea a través de clase corresponde al elemento `<ol>`, que posee con las clases `py-[1em] border-y-4 border-text-color m-1` adquiere un *padding* en el eje y de 1 em, se establece un border en el eje y de 4 unidades de anchura, y se establece el color del mismo como el color del texto, así como un margen global de 1 unidad.

En cuanto a las clases aplicadas mediante directivas a `rule-title-` y `rule-description`, estas son las siguientes:

```
.rule-title {
  @apply text-[1.2em] my-6 font-semibold;
}

.rule-description {
  @apply py-4 ml-8;
}
```

[1] Nótese que, aunque se describe en este apartado, el código HTML no se encuentra en el archivo `index.html`, sino que es transcluido a través de la dependencia `posthtml-include` a partir del archivo `component.html` mediante la etiqueta `<include>`.

La regla `.rule-title` utiliza la directiva `@apply` para aplicar un `text-[1.2em]` para establecer el tamaño del texto en 1.2 veces el tamaño base, `my-6` para agregar un margen vertical de 6 espacios, y `font-semibold` para establecer la fuente en negrita.

La regla `.rule-description`, por otro lado, utiliza la directiva `@apply` para aplicar `py-4` para agregar un relleno vertical de 4 espacios y `ml-8`para agregar un margen izquierdo de 8 espacios. Esto ayudará a dar formato y espaciado adecuados a los párrafos que contengan la descripción de las reglas.

Para el elemento `blockquote` siguiente, se aplicó el siguiente código:

```
<blockquote class="text-lg m-[1.75em]">
```

En este caso, se le ha asignado la clase `text-lg` que establece el tamaño del texto en grande. Además, se le ha asignado la clase `m-[1.75em]`, que establece un margen de espacio alrededor del bloque de texto citado, utilizando una medida relativa en `em`.

Para los párrafos, elementos con clase `.blog-paragraph`, se aplicó una directiva:

```
.blog-paragraph {
  @apply m-0 py-4;
}
```

Esta elimina el margen y aplica, a través de `py-4`, un *padding-block* de cuatro unidades de Tailwind.

Para la imagen siguiente, se describió el código:

```
<figure class="my-4">
  <div class="image-container">
    <img src="assets/images/chess_tournament.jpg" alt="People playing
    ↪  chess at a tournament">
  </div>
  <figcaption class="text-center text-[0.9em] my-2">
    So far, the club has upheld a healthy and friendly environment
  </figcaption>
</figure>
```

La clase “my-4” establece un margen vertical de 4 unidades de espacio alrededor de la figura. Es decir, agrega espacio tanto en la parte superior como en la parte inferior de la figura. `text-center` alinea el texto del elemento `<figcaption>` en el centro. `text-[0.9em]` establece el tamaño de fuente del texto del elemento `<figcaption>` en 0.9 unidades em. La clase `my-2` establece un margen vertical de 2 unidades de espacio alrededor del elemento.

A continuación se hallan dos párrafos más, que poseen las mismas características que el elemento con clase `blog-paragraph` descrito más arriba.

Después se describió el código relativo a la previsualización de otros *posts* del blog.

```
<div>
  <h4 class="text-center font-bold">Other posts</h4>
  <div class="flex flex-wrap">
    <div class="card">
      <!-- Código del interior de la tarjeta-->
    </div>
    <div class="card">
      <!-- Código del interior de la tarjeta-->
    </div>
    <div class="card">
      <!-- Código del interior de la tarjeta-->
    </div>
  </div>
</div>
```

En este caso, los elementos se dispusieron dentro de un contenedor que usa `flex`, para que estos se distribuyan de forma horizontal. Se aplicó `flex-wrap` para que en dispositivos de menor tamaño, estos se replieguen. El contenido de las tarjetas y su interior se definió mediante directivas:[2]

```
.card {
  @apply flex-grow-[1] flex-shrink-0 basis-[0%] border rounded-md
  ↪  bg-[#fff] flex-col flex-wrap min-w-[13em];
}

.card-img-top {
  @apply bg-[url('../images/backgrounds/checked_background.png')]
  ↪  bg-repeat-round bg-contain py-[10%] px-[20%] w-[100%];
}

.card-body {
  @apply p-4 text-[.85em];
}

.card-title {
  @apply font-bold underline text-center p-6;
}
```

Estas directivas se describen en los siguientes puntos:

- `.card` establece que la tarjeta debe crecer y encogerse de forma flexible (`flex-grow-[1]`, `flex-shrink-0`) sin ocupar espacio adicional (`basis-[0%]`).

[2]Téngase en cuenta que, a pesar de que se describe en este apartado, este código no se encuentra en el archivo `index.html`, sino que es transcluido a través de la dependencia `posthtml-include` a partir del archivo `component2.html` mediante la etiqueta `<include>`.

Además, agrega un borde y bordes redondeados (`border`, `rounded-md`), un fondo blanco (`bg-[#fff]`), y organiza los elementos internos en una columna flexible con capacidad de ajuste (`flex-col`, `flex-wrap`). También define un ancho mínimo de 13em (`min-w-[13em]`).

- `.card-img-top` configura el estilo de la imagen principal de la tarjeta. Utiliza una imagen de fondo con una URL específica (`bg-[url('...')]`) y se asegura de que la imagen se repita y se ajuste correctamente dentro del contenedor (`bg-repeat-round`, `bg-contain`). Además, establece un relleno vertical del 10% y un relleno horizontal del 20% (`py-[10%]`, `px-[20%]`), y asegura que la imagen ocupe el ancho completo de su contenedor (`w-[100%]`).
- `.card-body` define un espaciado interno de 4 unidades (`p-4`) para el cuerpo de la tarjeta y ajusta el tamaño del texto a 0.85em (`text-[.85em]`).
- Por último, `.card-title` establece que el título de la tarjeta debe ser negrita (`font-bold`), tener una línea subrayada (`underline`), estar centrado horizontalmente (`text-center`) y tener un espaciado interno de 6 unidades en todas las direcciones (`p-6`).

Este es el código principal de la página principal `index.html`, a continuación se describe el código de la página de miembros.

Desarrollo en Tailwind de la parte principal de `members.html`

Para la etiqueta `main`, que engloba el código principal de la página, se aplicaron las siguientes utilidades:

```
<main
    class="w-[90%] max-w-[1000px] text-text-color flex flex-col
↪   justify-center items-center my-0 mx-auto font-normal">
```

La clase `w-[90%]` establece que el ancho del elemento `<main>` será del 90% del ancho de su contenedor. La clase `max-w-[1000px]` establece un ancho máximo de 1000 píxeles para el elemento. `text-text-color` define el color del texto dentro del elemento según el valor configurado para `text-color` en la paleta de colores de Tailwind. Además, las clases `flex`, `flex-col`, `justify-center` y `items-center` aplican estilos de diseño flexbox para centrar vertical y horizontalmente el contenido. La clase `my-0` elimina cualquier margen vertical y la clase `mx-auto` centra el elemento horizontalmente en su contenedor. Por último, la clase `font-normal` establece el estilo de fuente del texto dentro del elemento `<main>` como normal.

El título de esta sección se describió mediante el siguiente código:

```
<h2 class="py-6 my-6 text-[1.75em] font-bold
↪   [text-shadow:_2px_2px_#ccc]">Chess Mates members</h2>
```

Las clases `py-6` y `my-6` establecen un relleno y margen vertical de 6 unidades de espacio, respectivamente, y `text-[1.75em]` define el tamaño de fuente del texto como 1.75 veces el tamaño de fuente base. `font-bold`, como ya se ha explicado, aplica un estilo de fuente en negrita. Por último, la clase `[text-shadow: 2px 2px #ccc]` crea una sombra de texto desplazada 2 píxeles hacia la derecha y 2 píxeles hacia abajo, con un color de sombra de `#ccc` (gris claro).

Posteriormente, el elemento que engloba todas las tarjetas de miembro se ha desarrollado como un `<div>` que posee las siguientes clases:

```
<div class="member-card-block flex flex-wrap items-center
↪  justify-around">
```

La clase `flex-wrap` permite que los elementos se envuelvan automáticamente en nuevas filas si no hay suficiente espacio horizontal disponible. Las clases `items-center` y `justify-around` se utilizan para alinear los elementos verticalmente y distribuir el espacio restante de manera equitativa entre ellos, creando así un diseño centrado y equidistante entre los elementos. En resumen, este código crea un bloque de tarjetas de miembros con un diseño flexible y distribución equitativa de los elementos dentro del contenedor.

Para las tarjetas, al repetirse frecuentemente el código, se usaron directivas:

```
.member-card {
  @apply w-[20em] flex flex-col py-8 items-center justify-center;
}

.member-img {
  clip-path: circle(50%);
}

.member-name {
  @apply text-[1.4em] font-bold my-4;
}

.member-description {
  @apply m-6 h-32;
}

.elo {
  @apply py-6;
}
```

`.member-card` se aplica al elemento que engloba toda la tarjeta para establecer un ancho fijo de 20em y utilizar un diseño de columna flexible con clases `flex` y `flex-col`, de forma que dispone los elementos de forma vertical. Además, se agregan clases `py-8` para

establecer un relleno vertical de 8 unidades en el eje y, y `items-center justify-center` para alinear vertical y horizontalmente los elementos dentro del contenedor. `.member-img` da forma de recorte circular a la imagen del miembro mediante la propiedad `clip-path` con la función `circle(50%)`. La clase `.member-name` se aplica para establecer el tamaño del texto en `1.4em`, utilizando una fuente en negrita (`font-bold`) y un margen vertical de 4 unidades (`my-4`). `.member-description` agrega un margen de 6 unidades en todos los lados (`m-6`) y establece una altura fija de 32 unidades (`h-32`) para la descripción del miembro. Esto puede utilizarse para crear un espacio adecuado para mostrar información adicional sobre el miembro. La última clase, `.elo`, agrega un relleno vertical de 6 unidades (`py-6`) al elemento.

Finalmente, existe un elemento que anima a los lectores a hacerse miembros, descrito así:

```
<p class="contact-us py-8 my-4">If you'd like to be a part of the club,
↪  don't hesitate to contact us through our form</a>!</p>
```

En este caso, se le aplica un estilo de *padding block* de 8 unidades (`2em`), así como un margen en el mismo eje de 4 unidades (`1em`).

Uso de posthtml-include

La práctica también requería el uso de esta dependencia, para poder utilizarla, se creó el archivo `.posthtmlrc` en la raíz del proyecto. A continuación, se describió el contenido del archivo JSON:

```
{
    "plugins": {
        "posthtml-include": {
            "root": "./src"
        }
    }
}
```

Esto establecería la raíz del proyecto en la carpeta `./src`. A continuación, se crearon dos archivos `component.html` y `component2.html`. Al primero se migró el código HTML relativo a las reglas, explicado en la sección anterior. En el segundo se incluyó la caja de previsualización de otros posts. Ambos en el archivo `index.html`.

Publicación del sitio web

Para la publicación de la web en internet se utilizó el servicio Netlify. Este permite la publicación de la página a partir de un repositorio público alojado en GitHub.

Se configuró el repositorio a través de Netlify. No tuvo que especificarse la carpeta raíz.

Netlify realizó, entonces, el *deploy* del proyecto de forma exitosa y sin generar errores de ningún tipo.

Este se publicó en la siguiente URL:

https://main–famous-custard-f63314.netlify.app/

Propiedad intelectual: atribución

Aquí se realiza la atribución debida a los autores de los elementos externos utilizados en el proyecto. Todos ellos poseen una licencia compatible con su uso por parte de terceros, con una serie de condiciones según el tipo de licencia Creative Commons asignada.

- En cuanto a los iconos utilizados en la página (incluído el favicon), se usó el paquete de Sports de ainul muttaqin en The Noun Project (disponible en este enlace), liberados bajo licencia CC BY 3.0. Sirva este punto como atribución.
- Las imágenes de los miembros del club halladas en el archivo `members.html` han sido generadas por IA a través de la página Generated.Photos. Estas se encuentran en dominio público.[1]
- Las foto de torneo «Players at a chess tournament» fue descargada de Wikimedia Commons. Su autor es Andreas Kontokanis, que la liberó bajo licencia CC BY-SA 2.0 (disponible en este enlace). Sirva este punto como atribución.

[1] Téngase en cuenta que, a pesar de que se describe en este apartado, este código no se encuentra en el archivo `index.html`, sino que es transcluido a través de la dependencia `posthtml-include` a partir del archivo `component2.html` mediante la etiqueta `<include>`.