PEC 2. Herramientas HTML y CSS II aula 1

Ignacio Casares Ruiz

Mayo de 2023

Instalación del boilerplate, creación del repositorio y guía de estilo

Al igual que en la PEC1, 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`. Después, se instaló Stylelint mediante el siguiente comando:

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

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

Se creó el archivo `.stylelintrc.json`, con la configuración adaptada a los criterios de la guía de estilo, descritos en el siguiente párrafo y en la siguiente sección.

Como guía de estilo principal, tanto para HTML y CSS, se decidió utilizar la guía de https://codeguide.co/. Se utilizó esta guía de estilo teniendo en cuenta que incluye directrices para el código HTML y CSS y que estas no entran mucho en profundidad. Se hizo de esta forma teniendo en cuenta que el tamaño de la tarea para la PEC 1 no es lo suficientemente voluminoso como para requerir la aplicación de una guía de estilo más elaborada.

Independientemente de esta guía de estilo, yo como usuario utilizo un linter de código personalizado que 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. Esto puede entrar en conflicto con algunas de las reglas especificadas en la guía de estilo elegida. El linter personalizado que uso lleva aplicándose en todos mis proyectos durante años, así que decidí dar prioridad a las modificaciones realizadas por este ante las recomendaciones de codeguide.co.

Se llevó a cabo una revisión de las normas que se podrían agregar a .stylelintrc.json para cumplir con los criterios establecidos.

Inicialmente, se estudiaron y memorizaron los criterios, y se realizaron evaluaciones periódicas para verificar su cumplimiento. Además, se corrigieron los errores reportados por Stylelint durante la compilación para producción utilizando el comando npm run start.

Configuración de `stylelint` y configuración de las reglas: ejecución práctica de criterios

Se modificó el archivo de configuración de Stylelint en `.stylelintrc.json`. Por defecto, se aplican las reglas especificadas en la configuración estándar SCSS de Stylelint (accesible aquí). Además, con el objetivo de estandarizar el uso de unidades en el código de la hoja de estilo, se decidió añadir una regla específica que restringiese la cantidad de unidades posibles:

```
{
    "extends": "stylelint-config-standard-scss",
    "rules": {
        "unit-allowed-list": [
            "em",
            "rem",
            "px",
            "%",
            "deg",
            "vw",
            "fr",
            "s",
            "vmax"
        ]
    }
}
```

Asimismo, se modificaron los scripts del archivo `package.json` para añadir la ejecución de Stylelint cara vez que se ejecute `npm run build`, de la siguiente forma:

```
{
    "parcel:serve": "parcel src/index.html -p 8123 --target web --open",
    "parcel:build": "parcel build src/index.html --target web
    ↪  --no-source-maps --no-cache",
    "clean": "rimraf dist .cache .cache-loader .parcel-cache",
    "dev": "npm-run-all clean parcel:serve",
```

```
    "build": "npm-run-all clean stylelint parcel:build",
    "test": "echo 'Everything is working as expected \nStart working on
    ↪  your project by running \\033[1mnpm run dev\\033[0m'",
    "stylelint": "stylelint src/**/*.scss"
  }
```

También se añadió una regla que tiene como objetivo estandarizar el nombre de las clases, tal y como se recomienda en la guía de estilo elegida. Esto se hizo a través de una regla que permite especificar el patrón de los selectores de clase de Stylelint con la especificación de una expresión regular como valor:

```
"selector-class-pattern": "^[a-z0-9]+(-[a-z0-9]+)*$"
```

Asimismo, se añadió una regla para obligar a los desarrolladores a utilizar comilllas dobles (`""`) en lugar de comillas simples (`''`) en la denotación de los `string` del proyecto. Esto también se realizó a través de la adición de una regla de Stylelint:

```
"string-quotes": "double"
```

En la ejecución del comando `stylint` aparece un mensaje que avisa de que la regla se encuentra obsoleta (*deprecated*), pero para los objetivos de esta práctica, después de una revisión de su funcionamiento, se decidió mantener.

Algo que se realizó también fue la adición de la obligación de aplicar la notación moderna en la especificación de colores. Esto se realizó mediante la introducción de la siguiente línea de código:

```
"color-function-notation": "modern"
```

Como excepción se introdujo la eliminación de la regla `no-descending-specificity`, que se aplica por defecto en la configuración estándar `stylelint-config-standard-scss`. Esto se hizo porque daba falsos errores al no identificar correctamente la especificidad de los elemetos debido al `nesting` de SCSS. Lo recomienda la documentación de la siguiente forma en proyectos que usan mucho *nesting*:

> This rule enforces that practice as best it can, reporting fewer errors than it should. It cannot catch every actual overriding selector, but it can catch certain common mistakes. We recommend turning this rule off if you use a lot of nesting.

Las recomendaciones de *nesting* aplicadas fueron las descritas en el apartado «Nesting in Sass and Less» de la guía de estilo utilizada.

También se desactivó la regla estándar `comment-no-emptyp` que forma parte del conjunto de reglas `stylelint-config-standard-scss`. Esta regla emite errores cuando, en la compilación, encuentra que existen comentarios `//` que no son seguidos de ningún carácter. Para poder jerarquizar los comentarios conforme a las pautas de la guía de estilo de

codeguide.co, los comentarios que se construyen como títulos de secciones se disponen englobados de este tipo, razón por la que se añadió esta excepción. Esto se hizo igualmente a través de una modificación en el archivo `.stylelintrc.json` de la siguiente forma:

```
"scss/comment-no-empty": null
```

Asimismo, se añadió una excepción a la regla `declaration-block-no-duplicate-properties`. Esto se hizo porque interfería en las variables que se declararon para realizar el *override* de los estilos por defecto de los componentes de Bootstrap. Se hizo de forma similar al resto de excepciones:

```
"declaration-block-no-duplicate-properties": null
```

Justificación y aplicación según la guía de estilo

El set de reglas estándar de CSS de stylelint `stylelint-config-standard-scss` se eligió porque ese sería el lenguaje utilizado para la confección de la hoja de estilos de la página. Se comprobaron que las reglas especificadas por el mismo no entraban en conflicto con la guía de estilo de codeguide.co elegida. En los casos en los que esto se daba, se añadieron excepciones, tal y como se han descrito en el apartado anterior.

La restricción del número de unidades utilizadas se usó con el objetivo de que la especificación de las dimensiones de los elementos de la página fuera más homogénea, con la justificación de que, a mayor cantidad de unidades, más difícil es cuadrar y maquetar el proyecto. Las unidades fueron elegidas en base a lo más utilizado por mi parte en el pasado. Su elección se encuentra más desarrollada en profundida en la sección «Desarrollo del código SCSS» posterior.

La regla `selector-class-pattern`, que limita el nombre de las clases a aquellas con minúsculas, números y un guión, se realizó en base a las recomendaciones especificadas en la guía de estilo de codeguide.co, en la sección «Class names», que recomienda lo siguiente:

> Keep classes lowercase and use dashes (not underscores or camelCase). Dashes serve as natural breaks in related class (e.g., `.btn` and `.btn-danger`).

La especificación del tipo de comillas permitido también se extrajo de las recomendaciones de la guía de estilo utilizada. En concreto, lo especificado en el apartado «Syntax»:

> Always use double quotes, never single quotes, on attributes.

La regla que introduce la obligatoriedad de utilizar la notación moderna para la designación de colores se añadió porque la guía de estilo elegida recomienda el uso de la notación `rgba()` en la sección «Colors».

La decisión de eliminar la regla `no-descending-specificity` se hizo en base a la cantidad de *nesting* empleado en el proyecto. Esta regla se anuló de forma posterior al comprobar que el funcionamiento de la misma no era el adecuado y de que existían una gran cantidad de falsos errores en el proceso de análisis de Stylelint.

Instalación de dependencia externa: `w3c-html-validator`

Como depedencia externa a instalar según lo exigía el enunciado, se decidió instalar una dependencia que permite ejecutar el servicio de validación de W3C de forma local a través de la consola de comandos. Existe una herramienta actualizada recientemente y soportada en la actualidad para este fin llamada W3C HTML Validator. Esta fue instalada a través de la ejecución del siguiente código:

```
npm install --save-dev w3c-html-validator
```

Para facilitar su uso, se configuró el archivo `package.json`, añadiendo el siguiente comando al objeto `scripts`:

```
"scripts": {
  "validator": "html-validator --ignore='Element "include" not allowed
  ↪  as child of element "body"'"
}
```

La lína que usa el comando opcional `--ignore` se añadió para evitar errores de compatibilidad entre el análisis realizado por esta extensión y el uso de la dependencia `posthtml-include`, que permite unificar varios archivos `html` distintos mediante el elemento `include`, tal y como se realizó en la elaboración de los archivos hallados en la carpeta `src/views`.

El script se ejecutaría por tanto mediante el siguiente comando:

```
npm run validator
```

Una vez ejecutado el código, aparecerían en consola los errores a solucionar. Una vez solucionados todos, la herramienta de validación arroja un mensaje positivo que indica que el archivo pasa el análisis de la W3C.

Esta análisis se realizó una vez terminada la redacción de todo el código `html` de la página. Una vez considerados todos los archivos como válidos, se mantuvo este código de forma definitiva.[1]

[1] Se ignoraron los errores en los archivos de la carpeta `src/views`, al ser estos trozos de HTML que serían incorporados en otros archivos HTML y que por tanto daban errores relacionados con la ausencia de elemento `head`.

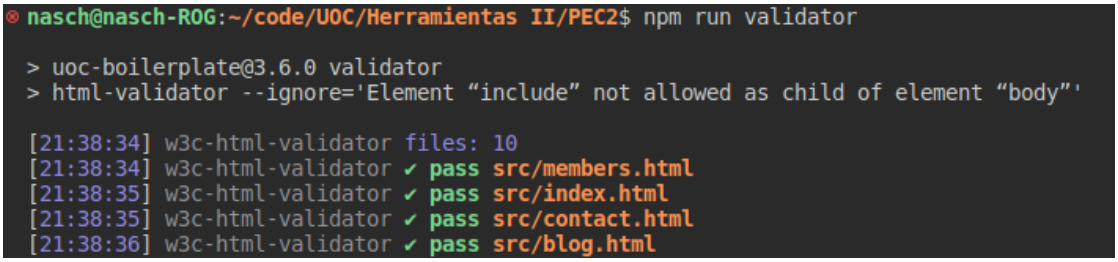


Imagen 1: Mensaje de éxito que aparece al no darse errores en el archivo `html`

Instalación e implementación de bootstrap

Se integró Bootstrap en el repositorio local, ejecutando el siguiente comando:

```
npm i --save bootstrap @popperjs/core
```

Para poder importar el javascript necesario para algunas de las funcionalidades que incluyen los componentes de Bootstrap, se añadió el siguiente código al archivo `main.js`:

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

Respecto a la importación de módulos de Bootstrap, en un principio se decidió realizar una importación completa, añadiendo el código `@import '~bootstrap/scss/bootstrap';` al archivo de importación `main.scss`. Esto, sin embargo, se descartó posteriormente porque aumentó de forma considerable el tiempo de carga de la web. En su lugar, se decidió activar módulos de Bootstrap selectivamente, de forma que se realizaron las siguientes importaciones:

```
@import "../../../node_modules/bootstrap/scss/functions";
@import "../../../node_modules/bootstrap/scss/variables";
@import "../../../node_modules/bootstrap/scss/maps";
@import "../../../node_modules/bootstrap/scss/mixins";
@import "../../../node_modules/bootstrap/scss/root";
@import "../../../node_modules/bootstrap/scss/alert";
@import "../../../node_modules/bootstrap/scss/accordion";
@import "../../../node_modules/bootstrap/scss/card";
@import "../../../node_modules/bootstrap/scss/transitions";
```

Estas correspondían a los módulos mínimos necesarios para que los componentes implementados en el proyecto funcionaran adecuadamente.

Uso de componentes de Bootstrap

Una vez realizada la importación de los módulos, se decidió utilizar los siguientes componentes:

- Un acordeón. Este se colocó en la página `contact.html` y serviría para definir las FAQ (Frequently Asked Questions) del club.
- Un mensaje de alerta. Este se posicionó también en la página `contact.html`, inmediatamente inferior a la sección de las FAQ.
- Un botón de cerrado, que se incluyó como parte del mensaje de alerta mencionado en el punto anterior, al cual se le aplicó javascript personalizado para que tuviera una funcionalidad.
- Un grupo de tarjetas, empleado en la parte final de la página `blog.html`. El objetivo de este componente es contener previsualizaciones con enlaces a otros posibles artículos del blog.

Para la personalización de los componentes, además de realizarse a través de la hoja de estilos general `_home.scss` mediante la modificación de diferentes propiedades, se creó el archivo `variable_overrides.scss`. Este, que sería importado a `main.scss`, contiene el código correspondiente a la sobreescritura de las variables por defecto de Bootstrap que permiten la modificación del aspecto de los componentes.

En las siguientes secciones se describe la aplicación y personalización de los componentes:

Acordeón

El componente de acordeón se modificó ampliamente para adaptarlo a las expectativas y al estilo de la web:

- Se cambiaron los atributos, añadiendo `collapse` a todas las clases de los tres segmentos que lo componen para que estos apareciesen cerrados por defecto. Se hizo de forma similar con el valor del atributo `aria-expanded`, que se determinó como `false` por defecto, con el objetivo de que los lectores de pantallas y otros dispositivos orientados a la accesibilidad puedan interpretar el menú como cerrado.
- Se crearon variables para sobreescribir las que Bootstrap aplica por defecto. En este caso se añadieron las siguientes, aunque no todas acabaron utilizándose:

```
$accordion-button-bg: unset;
$accordion-button-active-bg: rgb(254 248 244);
$accordion-icon-color: pink;
$accordion-button-active-color: unset;
$accordion-button-focus-box-shadow: rgb(248 221 201);
```

- Se modificaron aspectos de la hoja de estilos SCSS que no se pudo modificar mediante la sobreescritura de variables. Específicamente destaca el caso del color del botón de cerrado del acordeón, que, tras intentarlo a través del método descrito en el punto anterior, se decidió que era menos complejo cambiar el estilo mediante la implementación de la propiedad `background-image` del selector `button.accordion-button:not(.collapsed)::after`, cuyo `path fill` se configuró mediante el valor del color deseado. Entre otros aspectos, también se modificó el padding por defecto.

El resultado final permitió obtener un acordeón dinámico que se adaptaba bien a la estética elegida para la web.

Mensaje de alerta

Para el mensaje de alerta se usó el de tipo «info», configurado con las clases de Bootstrap `alert alert-info`. Este se configuró de forma que utilizase un color de fondo y letra diferentes. Para esto, Bootstrap realiza un mapeo de colores en base a lo definido como color primario, por lo que para llevar a cabo el *override*, tuvo que hacerse otro mapeo que sustituyese al mapeo original. Esto se realizó definiendo un color para el tema «info» nuevo y aplicando `map` sobre el mismo:

```
$info: rgb(248 221 201);

// Mapping
$theme-colors: (
    "info": $info
);
```

Esto no solo sustituyó el color de fondo, sino que aplicó un color distinto a la letra y al borde.

Botón de cerrado

Bootstrap incluye como parte de sus componentes un icono de botón cerrado con ciertas opciones de interactividad: posibilidad de convertirlo en botón desactivado fácilmente, aplicación de ciertas propiedades ante un *hover*, etc.

Se decidió utilizar este tipo de botón como forma estándar para el proyecto. Aunque en la realización de esta práctica solo se usó una vez, puede contribuir a una estandarización más sencilla en proyectos de mayor envergadura.

Entre sus modificaciones, se eliminó el cuadrado que ocupaba en el fondo mediante la aplicación de la declaración `background-color: transparent`, y se le aplicó la variable `$darker-pink` definida en el archivo `_variables.scss`.

Grupo de tarjetas

Como parte de la sección final del archivo `blog.html`, se añadió una serie de enlaces a otros *posts* ficticios del blog organizados dentro de un grupo de tarjetas.

Estos se aplicaron como un `div` de clase de Bootstrap `card-group`. No se realizó ningun modificación a través del *overriding* de variables, pero se cambiaron algunos estilos a través del código SCSS. Además, se añadieron imágenes para cada tarjeta y se adaptó el fondo de este espacio a través de la propiedad `background-image`.

Utilización de las características de Sass

Variables

Esta característica se ha utilizado de forma habitual en la elaboración del proyecto. Concretamente se han englobado dentro del archivo `_variables.scss` y `_variable_overrides.scss`. El segundo contiene las variables que sobreescriben las variables por defecto que se aplican a los componentes de Bootstrap.

Se han definido variables para las fuentes y para los colores estándar.

Anidado

El anidado se aplicó en las reglas SCSS incluídas en el archivo `_home.scss`. Este se desarrolló de forma ámplia en la estructuración de las mismas, siguiendo la estructura de anidado utilizada en la confección del archivo `html`.

Funciones

Al comienzo del código en `_home.scss` se ha implementado una función de Sass que invierte el color de las cosas. Para ello, se ha tenido que importar igualmente `@sass:color`. La función es la siguiente:

```
@function invert($color) {
  $red: 255 - color.red($color);
  $green: 255 - color.green($color);
  $blue: 255 - color.blue($color);

  @return rgb($red, $green, $blue);
}
```

Se ha aplicado para generar algunas sombras de títulos.

Parciales

El código SCSS se ha desarrollado a lo largo de varios archivos: `_home.scss`, `_variable_overrides.scss`, `_utility_classes.scss` y `_variables.scss` que se compilan de forma conjunta en el archivo `main.scss`.

Importación

Para la compilación de los *partials* definidos en la sección anterior, se utiliza `@import`.

Elección de la paleta de colores

Para la paleta de colores, se probaron varias combinaciones utilizando al aplicacion coolors.co. Finalmente, se decidió usar la siguiente paleta de colores:

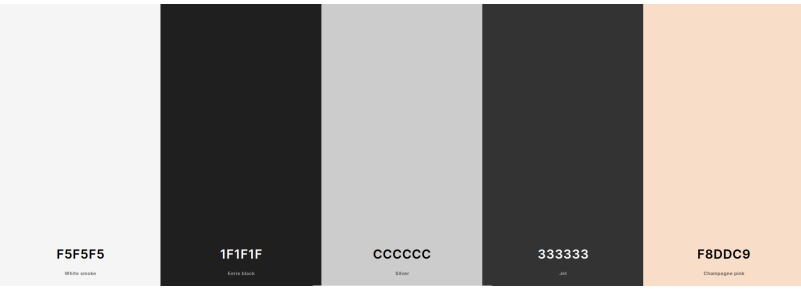


Imagen 2: Paleta de colores del sitio

- El color #F5F5F5 (whitesmoke) se asignó a los fondos básicos
- El color #1f1f1f (eerie black) se asignó al color de los textos
- El color #cccccc (silver) se asignó a
- El color #333333 (jet) se asignó a
- El color #f8ddc9 (champagne pink) se asignó a elementos de contraste, creándose una tonalidad más oscura mediante el módulo SCSS `@color`, la cual se aplicó a los enlaces a través de la pseudoclase `hover`.

Estos colores fueron asignados a variables SCSS en el archivo `_variables.scss`.

También se han usado ocasionalmente algunos de los colores propuestos en Bootstrap, cuando se ha considerado que estos encajaban bien con la paleta de colores elegida y el contexto.

Iconografía e imágenes

Para los iconos de la web, decidió usarse el paquete de Sports por ainul muttaqin en The Noun Project (disponible en este enlace), liberados bajo licencia CC BY 3.0.

Para las imágenes de los miembros del club se usó la página Generated.Photos. Estas se modificaron a través de la propiedad `clip-path` con el valor `circle(50% at 50% 50%)`. Se utilizó la herramienta clippy para ello.

Elaboración de la página web

La estructura de los elementos de toda la página web se realizó en función a lo descrito en el modelo de *wireframes* proporcionado en el enunciado, con ligeras modificaciones en apartados puntuales.

Desarrollo de encabezado y pie

Desarrollo del `header`

Para el header se decidió utilizar HTML y SCSS puro. Se siguió un diseño basado en los *wireframes* proporcionados en el enunciado de la práctica, con una barra superior en la que se mostrara, a la izquierda, el logo principal del club, y, a la derecha, el menú de navegación con las diferentes páginas que conforman el sitio web.

Para el código HTML, se utilizó el elemento `header` para englobal todo el contenido. Este contiene un `div` con la clase `container` para poder modificar fácilmente los elementos de dentro. Estos son tres, el logo principal de la página, configurado como un enlace que lleva a la página principal; el botón que abre el menú de navegación *responsive* (desarrollado en los párrafos posteriores), configurado como un elemento `button`; y el menú de navegación en sí, configurado como un elemento `nav`.

En cuanto al código SCSS, estos elementos fueron distribuidos fácilmente a través de la propiedad `display: flex` y situados adecuadamente mediante `align-items:center` y `justify-content: space between`. Sin embargo, se decidió, desde un enfoque orientado hacie una mayor *responsiveness*, crear un menú *responsive* para cuando el tamaño de pantalla dispositivo fuera menor.

Para su funcionamiento, se añadió una amplia `@media` *query* que se aplicaría en dispositivos de anchura menor a `40em`. En estos casos, aparecería una imagen de «hamburguesa» en la esquina superior derecha de la pantalla (configurada a través de CSS como un fondo). Esta imagen, ligada a un *event listener* de javascript que aplica una clase `data-visible`, ligada a un código css `display:block` o `display:none` a través de un *toggle*, abriría un menú configurado a través de `display: grid`, que mostraría las opciones de navegación. Para conseguir el efecto de sombreado que se aplica al resto de elementos de la página, se añadió un borde expandido a `1000vh` con una opacidad del 50% a través de la propiedad `box-shadow`. Para que el menú no tuviese colisión con el resto de elementos de la página, se le añadió al propiedad CSS `position: fixed`. Además, para que este apareciese en

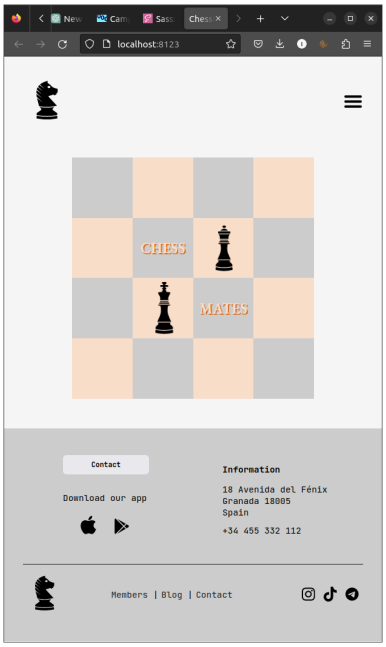


Imagen 1: Muestra del menù de navegación *responsive* que aparece en dispositivos pequeños

una posición (en cuanto a profundidad) más prioritaria respecto al resto de elementos, se aplicó la propiedad `z-index: 950`. Se usó la propiedad `gap` para configurar la distancia entre los elementos de la lista que conforma el menú.

Desarrollo del `footer`

Para el desarrollo del *footer*, se siguió, de igual forma, la distribución proporcionada por los *wireframes* del enunciado de la práctica. Sin embargo, para esta sección estos no eran demasiado específicos, por lo que se decidió seguir un enfoque propio en su diseño y elaboración.

De esta forma, se creó un *footer* dividido en dos secciones. Una sección superior, determinada mediante un elemento `div` con clase `main-footer`, y una interior, determinada con un elemento `div` con clase `sub-footer`. El `main-footer` contiene, a su vez, dos elementos `div`: uno `left-side-footer` y otro `right-side-footer`. En el espacio izquierdo de este footer se halla un botón que lleva a la página de contacto e información sobre una posible aplicación para el club, configurados como elementos `button` y `div`, respectivamente. En el otro lado, se halla información relativa a la dirección, configurada dentro de una lista no ordenada. El `sub-footer` contiene tres secciones, distribuidas en elementos `div`. De izquierda a derecha, estas corresponden al logo principal de la página, una sección de

enlaces a las diferentes paáginas de la web, conformadas como una lista no ordenada, y enlaces a redes sociales a través de iconos `svg`.

Desarrollo del `main`: parte principal de las páginas

Desarrollo del `main` de `index.html`

Para la parte principal de la portada, englobada dentro de un elemento main. Se ha decidido realizar un formato póster que utiliza CSS grid.

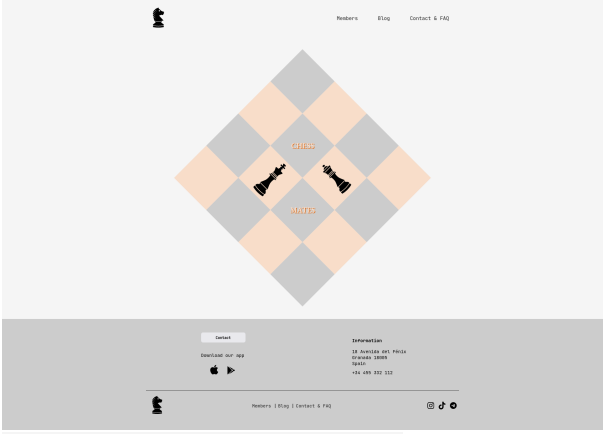
En cuanto al HTML, este simplemente consiste en una serie de elementos `div` contenidos dentro de uno de clase `wrapper`. Estos elementos poseen clases distintas con el objetivo de poder aplicar el CSS necesario para darle el aspecto de un tablero de ajedrez.

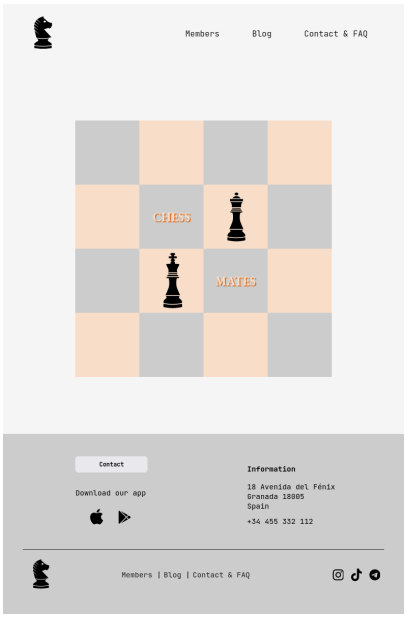
Para conseguir esto, se configuró la propiedad `display` con el valor `grid` y se añadió la propiedad `grid-template-columns` junto con el valor `1fr 1fr 1fr 1fr`. Esto conseguiría que los elementos `div` mencionados anteriormente se dispusieran en forma de cuadrícula de cuatro columnas y cuatro filas. Además, se añadió un efecto de rotación de 45 grados al elemento `wrapper` que solo se activa en dispositivos de mayor tamaño. Esto se realizó mediante el uso de la propiedad `transform` con el valor `rotate(45deg)`.[1] También se realizó la rotación de los bloques a los que se le aplicó un fondo con una pieza de ajedrez para que tomaran una posición oblícua, usando la misma propiedad. Para asegurarse de que estos se hallaban bien encajados en la cuadrícula correspondiente, se usaron las propiedades `background-size` y `background-position` con los valores `contain` y `center` respectivamente.

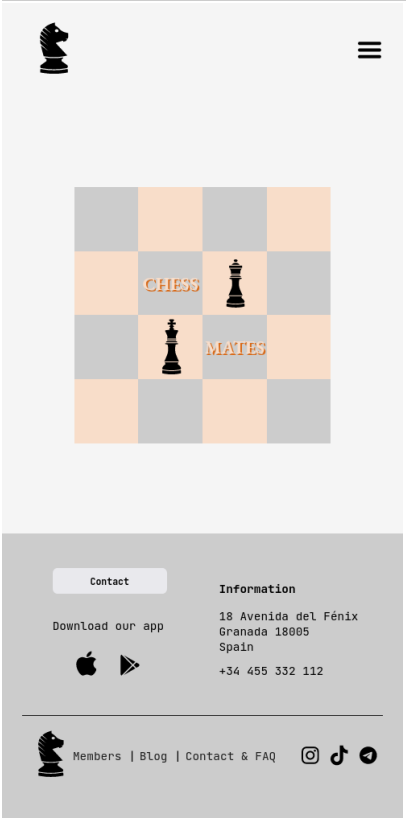
Además, se añadió un rótulo con el nombre del club «Chess Mates». Esto se realizó en las posiciones adyacentes a las cuadrículas con las piezas. Se configuró el estilo, usando la función nativa de CSS `clamp` para conseguir un tamaño de texto que se adapte al tamaño del dispositivo, y otorgando a la tipografía un color de sombra y tipo de fuente únicos.

Con le objetivo de asegurarnos de que los cuadrados del tableron eran exactamente cuadrados, se utilizó la propiedad `aspect-ratio`, a la cual se le otorgó el valor `1`. Los diferentes aspectos pueden visualizarse en las imágenes siguientes.

[1] `deg` no era una unidad admitida por nuestro código en `.stylelintrc.json`, por lo que tuvo que ser incluida para que no diese errores.


Members
Blog
Contact & FAQ
CHESS
MATES
Contact
Download our app
Information
18 Avenida del Fénix
Granada 18005
Spain
+34 455 332 112
Members | Blog | Contact & FAQ


Members
Blog
Contact & FAQ
CHESS
MATES
Contact
Download our app
Information
18 Avenida del Fénix
Granada 18005
Spain
+34 455 332 112
Members | Blog | Contact & FAQ


CHESS
MATES
Contact
Download our app
Information
18 Avenida del Fénix
Granada 18005
Spain
+34 455 332 112
Members | Blog | Contact & FAQ

Uso de `@supports`

Con el objetivo de adaptar la hoja de estilos a navegadores que no soportan CSS grid, se hizo uso de la *query* `@supports`. Esta permite implementar código solo en las instancias en las que el navegador posea soporte para esa característica.

En este caso, se utilizaron dos *queries* distintas: una `@supports (display: grid)`, para dispositivos que sí soportan este rasgo; y una `@supports not (display:grid)`, que cubre los casos en los que esta característica no es implementable. Se testeó el código para comprobar que la visualización, aún siendo significativamente más sencilla, era la adecuada en estos dispositivos.

Desarrollo del `main` de `members.html`

La sección `main` de la página en la que aparecen los miembros del club se desarrolló de la siguiente manera:

En cuanto al HTML, este se organizó de forma que el contenido principal, con todos los miembros del club, estuviese contenido en un elemento `div` con una clase `member-card-block`. En el código SCSS, se creó una regla en la que se le atribuyó la propiedad `display` con el valor `flex`. Además, con el objetivo de adaptarlo a diferentes tamaños de pantalla sin tener que realizar `@media` *queries*, se atribuyó al mismo elemento las declaraciones siguientes: `flex-wrap: wrap`, que haría que los elementos se plegasen en caso de colisión; `justify-content: space-around`, que permitiría que los elementos quedasen separados entre sí en la misma fila, otorgando espacio y mejorando la estética; y `align-items: center`, que conseguiría que estos quedasen encajados en el mismo plano vertical. Gracias a esta configuración, el número de miembros que aparece por cada fila varía (entre 1 y 3) dependiendo del tamaño del dispositivo desde el que se visualiza la página.

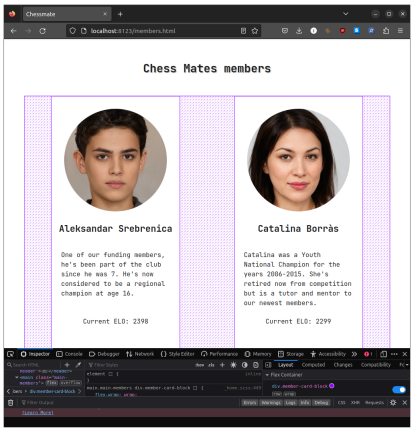


Imagen 2: Muestra de la visualización del elemento desde el inspector de Firefox

A cada elementos representando la tarjeta de miembro, sin embargo, se le aplicó la declaración `display: flex` pero conjuntamente con `flex-direction: column`. Esto se hizo así para asegurarnos de que los elementos *interiores* dentro de cada tarjeta se conforma verticalmente.

La imagen de cada miembro del club se insertó dentro de un `div` específico con la clase `image-container`. Para darles un aspecto más dinámico, al elemento `img` se le aplicó la declaración `clip-path: circle(50% at 50% 50%)`, realizada a través de clippy, que recortaría cada fotografía, otorgándoles un marco circular.

Desarrollo del `main` de `blog.html`

La página `blog.html` se configuró según el modelo aportado en el enunciado de la práctica.

En la parte superior de la página, se realizó una portada del blog, englobada dentro de un elemento `div` con clase `checked-background`, en la que se utiliza una imagen de fondo que sigue el mismo patrón de colores y formas que la utilizada en la página `main.html`. Esta imagen se introdujo mediante código SCSS a través de la propiedad `background-image` y no como elemento `img` en el HTML. Se utilizaron las declaraciones `background-size: contain` y `background-repeat: round` para permitir que la imagen se adapte fácilmente a los distintas variaciones de tamaño de los diferentes dispositivos desde los que se visualiza. Inmediatamente superior al título, se colocó una imagen representativa del blog, englobada dentro de un elemento `img` con su propio *container*. Para que los elementos de esta parte del blog se dispusieran de forma adecuada, se usaron las declaraciones `display: flex` y `flex-flow: column`.

Imagen 3: Visualización de la portada del blog

El resto del código se concibió dentro de un gran elemento `div` con la clase `narrative-blog`. Este contiene todos los elementos de esta página que no son la portada.

Tal y como se puede visualizar en los *wireframes* inmediatamente inferior al título de la entrada del blog, se observa un elemento que muestra dos elementos en una misma fila. Para conseguir esto, se englobó a los elementos en un `div.motto-and-intro` y se utilizó, de nuevo, la declaración `display: flex`, y a cada elemento (un elemento `blockquote` y un `p`) se le otorgó una anchura porcentual correspondiente a la que se ve en los *wireframes*. Con el objetivo de que, ante una reducción del tamaño del *viewport* estos no quedasen

demasiado aplastados, se otorgó al elemento *flex* la declafración `flex-wrap: wrap`, y a cada hijo de esta se le aplicó `flex: auto`. Esta última permite que, cuando se dé el *wrap* de los elementos en dispositivos de mayor tamaño, la anchura de los elementos crezca hasta adaptarse a la anchura total del elemento en el que se encuentran.

La siguiente parte corresponde a la lista de reglas. Esta se ha concebido como un elemento `ol` (lista ordenada). Cada elemento `li` contiene el título del a regla y la descripción, conformados como elementos `h3` y `p` respectivamente. Se aplicaron algunos estilos a través de la hoja de estilo con el objetivo de seguir la línea estética general de la página.

Para la fotografía que se puede visualizar en la parte media-inferior de la página, se usó el elemento `figure`. Este contiene un `div` que sirve como contenedor para la imagen, y un elemento `figcaption`, para el pie de foto. Se realizaron algunas modificaciones de estilo.

Al final de la página puede hallarse el componente de Bootstrap descrito en la sección «Uso de componentes de Bootstrap: Grupo de tarjetas». El elemento correspondiente a la imagen de la parte superior de cada tarjeta se engloba fue modificado con el siguiente código para poder implementar el fondo de tablero de ajedrez deseado:

```
img.card-img-top {
  padding-block: 10%;
  padding-inline: 20%;
  fill: $text-color;
  background-image:
   ↪  url("../images/backgrounds/checked_background.png");
  background-size: contain;
  background-repeat: round;
}
```

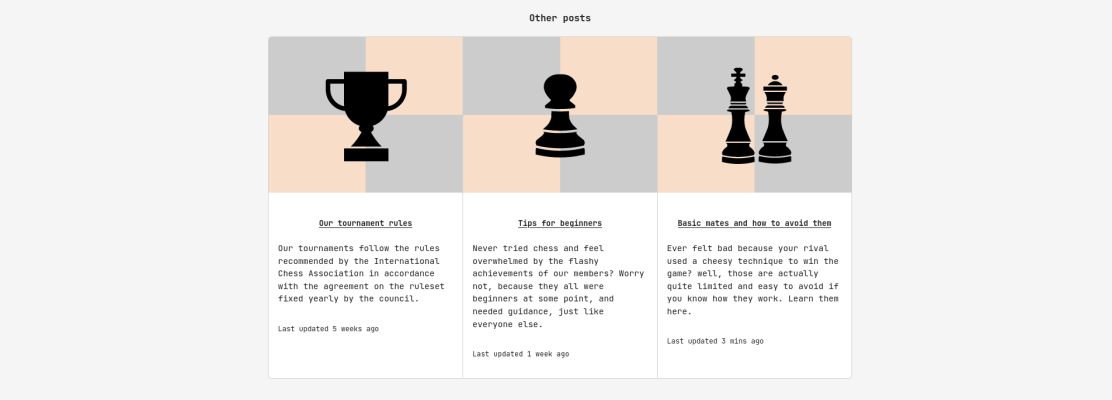


Imagen 4: Muestra del grupo de tarjetas en dispositivos de mayor tamaño

El icono `svg` añadido se integró dentro de ese elemento, de forma que apareciese junto al fondo implementado a través de SCSS.

Desarrollo del `main` de `contact.html`

El código de esta parte de la página corresponde a aquel en el que se describen las FAQ del sitio y se halla el formulario de contacto del club.

Este corresponde, por tanto, al acordeón (componente de Bootstrap mencionado en la sección pertinente), el mensaje informativo y el formulario de contacto.

El acordeón fue desarrollado ampliamente en la sección hallada más arriba.

El mensaje informativo, que incluye en sí mismo dos componentes de Bootstrap (el mensaje en sí y el botón de cierre del mensaje), se adaptó a la estética deseada de la página. Aunque el mensaje en sí recibió casi todas sus modificaciones a través del *overriding* de variables por defecto de Bootstrap mediante mapeo, el botón de cerrado sí tuvo que recibir una personalización significativa a través del código SCSS:

```
button.close {
  position: absolute;
  font-size: 1.5em;
  top: 1px;
  right: 5px;
  border: none;
  background-color: transparent;
  align-items: right;
  justify-content: right;
  color: $darker-pink;
  cursor: pointer;
}
```

En cuanto al formulario, hay que tener en cuenta que la funcionalidad del mismo está limitada al no haber programado el código correspondiente al *server-side*. Este se desarrolló como un elmeento `form`, que contiene elementos `input` y `textarea` asociados a etiquetas, configuradas como elementos `label`. Se añadió un elemento `div` en el que contener la `checkbox` junto al texto explicativo. Se modificó ampliamente la hoja de estilos para adaptar su diseño a los estándares de la página.

Publicación del sitio web

Propiededad intelectual: atribución

- En cuanto a iconos, se usó el paquete de Sports de ainul muttaqin en The Noun Project (disponible en este enlace), liberados bajo licencia CC BY 3.0.
- 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).

[1] La falta de elegibilidad de las imágenes generadas por inteligencia artificial para la protección de derechos de autor se debe a la ausencia de autoría humana. Aunque los algoritmos utilizados para generar estas imágenes son el resultado de la creatividad y el esfuerzo intelectual humano, la imagen resultante en sí misma es considerada por expertos como carente de originalidad humana y, por lo tanto, no cumpliría los requisitos para la protección de derechos de autor. Esta situación podría cambiar dependiendo del sentido de la jurisprudencia, pero al no existirla, se interpreta que tal protección no existe.