M4.258 - Herramientas HTML y CSS II

PEC 1

Aníbal Santos Gómez

ÍNDICE: DOCUMENTACIÓN.

1. Objetivos.

- Diseñar y ejecutar un pequeño sitio web responsive de una sola página. Partiendo de un boilerplate.

- Escoger criterios de desarrollo y arquitectura reutilizable CSS.

- Utilizar y configurar Stylelint como linter CSS para mejorar nuestra escritura de código CSS

- Generar una documentación práctica que siga la elección de criterios, proceso de desarrollo y el proceso de despliegue.

2. Herramientas utilizadas.

Para el desarrollo de una web moderna, necesitamos herramientas modernas que nos permitan fácilmente el desarrollo de la misma. El principal objetivo de este desarrollo es centrarnos en la construcción de una web y perder el mínimo tiempo posible en otras actividades.

Para ello se eligen un conglomerado de herramientas que nos facilitarán, el desarrollo en local, el pase a producción, el control de versiones, la maquetación, y otra serie de herramientas que se justifican continuación:

- Editor de código: **Visual Studio Code**.

  Se utilizará **VS Code** como editor de código, ya que nos permite instalar plugins que nos permiten utilizar snippets para agilizar nuestra forma de programar. Además es totalmente personalizable en función del proyecto que se vaya a realizar, en cuestión de herramientas. Actualmente se han instalado plugins como auto close tag, rename tag, css formatter, highlight matching tag, JS snippers, Prettier, Stylint plugin.

- Control de versiones y repositorio: **Git y Github.**

  Para el control de versiones utilizaremos **Git** y para almacenar nuestro repositorio en la nube utilizaremos el servicio que nos brinde **Github.**

- Gestor de paquetes: **Npm.**

  Como gestor de paquetes de Node.js utilizaremos **Npm,** que contiene un sin fin de dependencias desarrolladas por otros programadores que nos facilitarán la configuración de los diversos paquetes que utilizaremos.

- Module bundler: **Parcel.**

  Para empaquetar todo nuestro proyecto utilizaremos **Parcel** ya que es por defecto el module bundler que se encuentra definido en el pacakge.json de UOC boilerplate.

- Preprocesadores de código: **PostCSS y Sass**.

  PostCSS es una herramienta de desarrollo de software que utiliza complementos basados en JavaScript para automatizar las operaciones de rutina de CSS.

  Sass que nos permitirá extender funcionalidades CSS que no están contenidas de forma nativa, como funciones, variables… y que nos permitirán programar nuestras hojas de estilos de una manera mucho más ágil.

- Dependencias: **Fontawesome Free**, **Stylelint**

  Fontawesome, es un conjunto de herramientas de fuentes e iconos basado en CSS.

  Stylelint, es un linter que nos ayudará a escribir mejor código CSS en base a unas reglas predefinidas según una configuración que utilicemos.

- Publicación: **Netlify.**

  Por último para el despliegue se utilizará **Netlify**, que nos permite vincular nuestra cuenta de **Github**, y el repositorio en cuestión, y ejecutar la compilación en nuestro servidor en cuestión de un comando.

3. Metodologías utilizadas.

Para el desarrollo de esta pequeña aplicación web, hemos utilizado como metodologías CSS, dos metodologías que se complementan muy bien y nos permitirán reducir nuestro código css de una manera recursiva y sustancial.

Las metodologías elegidas son:

- **Mobile First:** es la forma que utilizaremos para construir el diseño, partiendo siempre desde pantallas pequeñas a pantallas más grandes. Por defecto construiremos orientado a dispositivos móviles y abordaremos el resto de comportamientos en función del ancho de la pantalla para definir comportamientos para tablet y desktop.

- **ITCSS:** como base para el desarrollo se escogé este tipo de metodología o arquitectura, que nos permite establecer un pequeño estándar a desarrollar para nuestra aplicación.

  Esta metodología nos ayudará a definir determinados tipos de selectores, clases y configuraciones en función, en primer lugar, de la magnitud o alcance, la especificidad y según la claridad respecto a la abstracción.

- **OOCSS:** como arquitectura complementaria hemos elegido OOCSS, para poder realizar abstracción de ciertos elementos que se repiten a lo largo de nuestro proyecto y que tienen una base común, pero que de ninguna manera son ajustes, herramientas, configuraciones genéricas o elementos nativos HTML.

A continuación se detallan las dependencias que componen la arquitectura mencionada en estos dos puntos anteriores. Para llevar a cabo la misma, hemos partido de la configuración de dependencias del boilerplate assets/styles. Los ficheros/dependencias a tener en cuenta por impacto de mayor a menor son:

- **main:** este archivo carga todos los módulos de la arquitectura ITCSS. En él procedemos a importar cada módulo de dicha arquitectura:

```
/**
 *! BASE CONTENTS FOR ITCSS
 *
 ** SETTINGS - 01_settings"
 *
 ** TOOLS - 02_tools
  *
 ** GENERIC - 03_generic
 *
 ** ELEMENTS - 04_elements
 *
 ** OBJECTS - 05_objects
 *
 ** COMPONENTS - 06_components
 *
 ** TRUMPS - 07_trumps
 */
```

- **01_settings:** este módulo cargará variables de configuración reutilizables a lo largo de todo el proyecto, en este caso hemos definido, la **fuente** base, su tamaño base, y su interlineado base. Además hemos definido los tamaños aplicables de las fuentes, xs, sm, base, md, lg, xl.

```
$base-font: "Courier New", Courier, monospace;
$base-font-size: 1rem;
$base-line-height: 1.25 * $base-font-size;

$text-settings: (
  "xs": (
    font-size: 0.75rem,
    // 12px
    line-height: $base-line-height,
  ),
  "sm": (
```

```
    font-size: 0.875rem,
    // 14px
    line-height: $base-line-height,
  ),
  "base": (
    font-size: 1rem,
    // 16px
    line-height: $base-line-height,
  ),
  "md": (
    font-size: 1.125rem,
    // 18px
    line-height: $base-line-height * 2,
  ),
  "lg": (
    font-size: 1.25rem,
    // 20px
    line-height: $base-line-height * 2,
  ),
  "xl": (
    font-size: 1.5rem,
    // 24px
    line-height: $base-line-height * 2,
  ),
);
```

También hemos definido la **paleta de colores** seteable diferenciada en primarios y herramientas.

```
$theme-colors: (
  "primary": (
    "base": #ffd700,
    "light": #fffb00,
    "mid": #ffbb00,
    "dark": #ffae00,
  ),
```

```
  "tools": (
    "black": #1d1d1d,
    "full-black": #050505,
  ),
);
```

Y por último **variables de contraste** para poder modificar los diferentes tipos de contraste en la maquetación de los textos, fondos y cualquier otra propiedad que nos pueda ser útil.

```
$contrast-colors: (
  "dark": (
    "primary": rgb(255, 255, 255),
    "secondary": rgba(255, 255, 255, 0.7),
    "disabled": rgba(255, 255, 255, 0.5),
    "hint": rgba(255, 255, 255, 0.12),
  ),
  "light": (
    "primary": rgba(0, 0, 0, 0.87),
    "secondary": rgba(0, 0, 0, 0.54),
    "disabled": rgba(0, 0, 0, 0.38),
    "hint": rgba(0, 0, 0, 0.12),
  ),
);
```

- **02_tools:** este módulo define funciones y mixins globales reutilizables a lo largo de todo el proyecto, que además utilizará la base predefinida en las variables configuradas en el módulo “01_settings”.

  Para los **textos** hemos creado las siguientes funciones:

  Para **escalar** los textos en función de los ajustes y el nivel:

```
@function text-scale($level) {
  @return map-get(map-get($text-settings, $level), "font-size");
}
```

Para modificar el **interlineado** de los textos:

```
@function line-height($level) {
  @return map-get(map-get($text-settings, $level), "line-height");
}
```

Mixin reutilizable en las funciones anteriores según el nivel requerido:

```
@mixin text-setting($level) {
  font-size: text-scale($level);
  line-height: line-height($level);
}
```

Mixin reutilizable para el desborde de los textos:

```
@mixin truncateText($overflow: ellipsis) {
  overflow: hidden;
  white-space: nowrap;
  text-overflow: $overflow; // values are: clip, ellipsis, or a string
}
```

Para **la paleta de colores y los contrastes** hemos creado las siguientes funciones y mixins:

Para setear los **colores de la paleta**:

```
@function theme-color($key: "primary", $variant: "base") {
  $map: map-get($theme-colors, $key);
  @return map-get($map, $variant);
}
```

Para generar el **fondo con un tipo de contraste:**

```
@function contrast($background: "light", $type: "primary") {
  $map: map-get($contrast-colors, $background);
  @return map-get($map, $type);
}
```

Para generar un **color con contraste**:

```
@mixin contrast($background: "light", $type: "primary") {
  color: contrast($background, $type);
}
```

Además para poder definir un comportamiento en la maquetación entre diferentes dispositivos hemos establecido unos mixins de media queries según dispositivo. Como estamos trabajando mediante la metodología de diseño **Mobile First**, por defecto construimos para dispositivos móviles, por lo que se entiende que no necesitaremos una media query para esto. Los mixines definen el contenido aplicable según, **tablet, desktop y desktop-large**:

```
@mixin tablet {
  @media only screen and (min-width: 768px) {
    @content;
  }
}

@mixin desktop {
  @media only screen and (min-width: 992px) {
    @content;
  }
}

@mixin large {
  @media only screen and (min-width: 1200px) {
    @content;
  }
}
```

Para poder alinear el contenido de las cajas en función de las media queries, hemos creado unos mixins que nos permiten alinear el contenido de las cajas, algo que es común y muy reutilizable:

```
@mixin align-center {
  display: flex;
  justify-content: center;
}

@mixin align-start {
  display: flex;
  justify-content: flex-start;
}

@mixin align-end {
  display: flex;
  justify-content: flex-end;
}
```

Hemos creado un par de mixins más para **editar los estilos de los enlaces** cuando se hace hover y se visitan, y otro para la animación de subrayado:

```
@mixin hover-link {
  text-decoration: none;
  color: #000000;

  &:visited {
    color: #000000;
  }
}
```

En el **mixin del subrayado** hemos añadido una clase modificadora que nos permite quitar la animación y dejar plano el enlace:

```
@mixin underline {
  position: relative;
  &:before {
```

```
    content: "";
    position: absolute;
    width: 0;
    height: 2px;
    bottom: 0;
    left: 0;
    background-color: theme-color("tools", "black");
    visibility: hidden;
    transition: all 0.3s ease-in-out;
  }
  &:hover:before {
    visibility: visible;
    width: 100%;
  }

  &.no-underline {
    &:before {
      height: 0;
    }
  }
}
```

- **03_generic:** este módulo define el conjunto de normas normalizadoras que se ejecutará como primera capa para que podamos partir de elementos iguales sin tener en cuenta el navegador que utilicemos.

- **04_elements:** en este módulo daremos estilo base a los diferentes elementos html por defecto. Para ello hemos importado el módulo de herramientas y lo hemos utilizado de la siguiente manera:

  Para los **elementos de título:**

```
h1,
h2 {
  @include text-setting("xl");
  text-shadow: 3px 3px 0 theme-color("primary", "dark");
```

```
  font-weight: bold;
}

h3 {
  @include text-setting("lg");
  font-weight: bold;
}

h4,
h5,
h6 {
  @include text-setting("md");
}
```

Para el **resto de elementos** que contengan textos:

```
p,
a,
span,
ul,
ol,
li {
  @include text-setting("base");
}
```

Para los elementos **header, main y footer:**

```
header,
footer,
main {
  padding: 20px 0;
}
```

Para los **enlaces** utilizamos mixins de hover y underline:

```
a {
  @include hover-link;
  @include underline;
}
```

Para los **párrafos** metemos un margen top y bottom:

```
p {
  margin: 10px 0;
}
```

- **05_objects:** en este módulo realizaremos la arquitectura OOCSS, el patrón de objetos que utilizamos básicamente se compone de un base, que será un selector de clase global en el que utilizaremos, fondo, text base y fondo, posteriormente las clases utilizadas como objetos corresponden a contenedores, items, filas, columnas y contenido como título, bloque de contenido y sub-contenido.

  Clase **base:**

```
.base {
  font-family: $base-font;
  @include text-setting("base");
  background-color: theme-color("primary", "base");
}
```

  Contenedores:

```
.container {
  width: 100%;
  margin: 0 auto;
  padding: 20px;
```

```
  @include tablet {
    max-width: 768px;
  }

  @include desktop {
    max-width: 992px;
  }

  @include large {
    max-width: 1200px;
  }
}

.item {
  padding: 10px;
  margin: 10px 0;
}

.sub-item {
  padding-top: 10px;
}


.row {
  display: flex;
  flex-direction: column;

  &.no-column {
    flex-flow: wrap;
    flex-direction: row;
  }
  @include desktop {
    flex-flow: wrap;
    flex-direction: row;
  }
}
```

```
.column {
  padding: 20px;
  width: 100%;

  @include desktop {
    &.one {
      width: 33%;
    }

    &.two {
      width: 50%;
    }

    &.two-third {
      width: 66%;
    }

    &.full {
      width: 100%;
    }
  }
}
```

Contenidos:

```
.title {
  text-transform: capitalize;
}

.content {
  padding: 15px;
}

.sub-content {
  padding: 15px 0;
}
```

- **06_components:** continuando con nuestra arquitectura ITCSS, en este módulo declararemos los elementos que actúan como componentes dentro de nuestra User Interface, los cuales sobre todo incluyen estilos a nivel de funcionalidad y describen patrones reutilizables de forma genérica.

  Hemos creado un pequeño loader para simular una pequeña carga mediante Javascript, esto nos condiciona la utilización de dos ids para la identificación de los componentes genéricos de la aplicación, que serán **Cv y Loading:**

```
#cv {
  display: none;
}

#loading {
  display: flex;
  justify-content: center;
  flex-direction: column;
  align-items: center;
  position: absolute;
  top: 0;
  left: 0;
  z-index: 1;
  width: 100%;
  height: 100%;
  background-color: theme-color("tools", "black");
}

#loading .container-img {
  width: 200px;
  height: 200px;
  background-image: url("/src/assets/images/icons/stars.svg");
  background-repeat: no-repeat;
  background-position: center;
  animation: heartbeat 4s infinite;
}
```

```
@keyframes heartbeat {
  0% {
    transform: scale(0.75);
  }
  20% {
    transform: scale(1);
  }
  40% {
    transform: scale(0.75);
  }
  60% {
    transform: scale(1);
  }
  80% {
    transform: scale(0.75);
  }
  100% {
    transform: scale(0.75);
  }
}
```

Componentes Header y Footer:

```
.header {
  @include align-end;

  & h1 {
    @include underline;
  }

  & span {
    font-weight: normal;
    font-style: italic;
  }
}
```

```
.footer {
  @include align-end;
  align-items: center;

  & > * {
    margin: 0 10px;
  }
}
```

Componente fecha:

```
.date {
  font-style: italic;
  color: contrast("light", "secondary");
}
```

Componente Github:

```
.github {
  @include align-center;
  @include desktop {
    @include align-start;
  }
}
```

Componente Icon:

```
.icon {
  width: 1.5rem;
  height: 1.5rem;
  transition: transform 0.2s;

  &:hover {
    transform: scale(1.5);
  }
}
```

Componente Avatar:

```
.avatar {
  border-radius: 50%;
  width: 150px;
  height: 150px;
}
```

Componente Button:

```
.button {
  position: relative;
  width: 230px;
  padding: 10px 0;
  border-radius: 10px;
  background-color: theme-color("tools", "full-black");
  color: theme-color("primary", "dark");
  box-shadow: 3px 3px theme-color("primary", "dark");
  text-align: center;
  font-weight: bold;
  cursor: pointer;

  & span {
    margin: 0 10px;
  }

  &:hover {
    box-shadow: none;
    top: 3px;
    left: 3px;
  }
}
```

Componente Chip:

```
.chip {
  position: relative;
  padding: 10px 20px;
  margin: 5px;
  border-radius: 20px;
  background-color: theme-color("primary", "mid");
  box-shadow: 3px 3px contrast("light", "hint");
  text-align: center;
  font-weight: bold;

  &:hover {
    box-shadow: none;
    top: 3px;
    left: 3px;
  }
}
```

- **07_trumps:** por último hemos utilizado este módulo como utilidades y helpers que sobreescribir clases y modifican algunos estilos que por base vienen definidos de otra manera genérica. Lo hemos utilizado para setear tipos de texto y bordes custom de las cajas contenedoras:

```
.italic {
  font-style: italic;
}

.bold {
  font-weight: bold;
}

.b-dash {
  border: 2px;
  border-style: dashed;
}
```

```
.b-inset {
  border: 2px;
  border-style: inset;
  border-color: theme-color("primary", "dark");
}
```

4. Desarrollo:

Para explicar el desarrollo completo de este proyecto, hemos dividido esta sección en las siguientes fases, que a continuación resumen el desarrollo del mismo:

1. **Descargar boilerplate.**

   En primer lugar descargamos el boilerplate facilitado para esta práctica. En mi caso realicé previamente un fork del boilerplate de la UOC a mi cuenta de usuario en **Github.**

   Podemos descargarlo haciendo git clone en cualquier dependencia de nuestro equipo que deseemos mediante terminal, introduciendo lo siguiente:

   git clone https://github.com/ansango/uoc-boilerplate

   o

   git clone https://github.com/uoc-advanced-html-css/uoc-boilerplate

2. **Instalación y configuración de dependencias.**

   En primer lugar instalaremos el proyecto mediante el siguiente comando:

   npm i

   Ya tendremos instaladas las dependencias base del boilerplate instaladas. A continuación instalaremos las dependencias **Fontawesome y Stylint**.

Para instalar **Fontawesome** introduciremos en nuestro terminal en la raiz del proyecto lo siguiente:

npm install --save @fortawesome/fontawesome-free

Para utilizar **Fontawesome** en nuestro proyecto necesitaremos importar el módulo en src/assets/scripts/main.js escribiendo lo siguiente:

import "@fortawesome/fontawesome-free/css/all.css";

Verificaremos en nuestro package.json que se ha instalado correctamente nuestra dependencia:

```
"dependencies": {
   "@fortawesome/fontawesome-free": "^5.15.2"
  }
}
```

A continuación procederemos a instalar y configurar **Stylelint,** en primer lugar introduciremos en nuestro terminal, en la raíz del proyecto:

npm install --save-dev stylelint-scss stylelint-config-recommended-scss

En nuestro package.json deberíamos ver algo como esto:

```
"devDependencies": {
   ...
   "stylelint": "^13.12.0",
   "stylelint-config-recommended-scss": "^4.2.0",
   "stylelint-scss": "^3.19.0"
 },
```

Posteriormente crearemos un archivo de configuración para Stylelint, “.stylelintrc.json”, que contendrá las reglas que utilizaremos para la validación. La base para este archivo es la siguiente:

```
{
  "extends": "stylelint-config-recommended-scss",
  "rules": {}
}
```

En este caso hemos incluido las siguientes reglas:

```
    "at-rule-no-unknown": null,
    "scss/at-rule-no-unknown": true,
    "comment-no-empty": true,
    "no-empty-source": true,
    "font-family-no-duplicate-names": true,
    "color-no-invalid-hex": true,
    "string-no-newline": true,
    "unit-no-unknown": true,
    "property-no-unknown": true,
    "declaration-block-no-duplicate-custom-properties": true,
    "declaration-block-no-duplicate-properties": true,
    "declaration-block-no-shorthand-property-overrides": true,
    "block-no-empty": null,
    "selector-pseudo-class-no-unknown": true,
    "selector-pseudo-element-no-unknown": true,
    "selector-type-no-unknown": true,
    "media-feature-name-no-unknown": true,
    "no-extra-semicolons": true,
    "no-invalid-double-slash-comments": true,
    "color-named": "never",
    "length-zero-no-unit": true,
    "font-weight-notation": "named-where-possible",
    "declaration-block-no-redundant-longhand-properties": true,
    "declaration-no-important": true,
    "color-hex-length": "long",
    "function-name-case": "lower",
    "unit-case": "lower",
    "number-no-trailing-zeros": true,
    "number-leading-zero": "always",
    "property-case": "lower",
```

```
  "declaration-colon-space-after": "always-single-line",
  "declaration-block-semicolon-newline-after": "always",
  "block-closing-brace-newline-after": "always",
  "selector-type-case": "lower",
  "selector-pseudo-element-case": "lower",
  "selector-pseudo-class-case": "lower",
  "selector-descendant-combinator-no-non-space": true,
  "selector-list-comma-newline-after": "always",
  "at-rule-semicolon-newline-after": "always",
  "at-rule-name-space-after": "always",
  "at-rule-name-case": "lower",
  "comment-whitespace-inside": "always"
```

A groso modo, estas reglas nos permiten escribir código CSS que, no esté duplicado, que no esté vacío, que no haya colores que no sean hexadecimales, que no existan propiedades desconocidas, bloques vacíos, etc…

Para poder ejecutar y probar toda nuestra configuración debemos añadirlo a nuestro package.json como un nuevo script:

```
"scripts": {
  "build": "npm-run-all clean stylelint parcel:build",
  "stylelint": "stylelint src/**/*.scss"
 },
```

Para ejecutarlo:

npm run stylelint

Para agilizar nuestras pruebas hemos configurado un plugin muy útil en nuestro IDE, que se comentará a continuación.

3. Entorno de desarrollo y configuración.

Una vez instaladas y configuradas las dependencias procederemos a abrir nuestro proyecto con el IDE que queramos, en este caso se utilizará VSCode.
Dentro de la amalgama de plugins y extensiones que pueden añadirse a VSCode, hemos elegido los siguientes, que nos facilitarán la escritura de código:

- Better Comments: Comentarios coloreados de todo tipo. Personalmente he instalado este plugin para poder documentar de una manera mucho más llamativa toda la documentación y hacerla menos pesada.

- Colorize: para destacar los colores seleccionados en CSS. Es complicado memorizar colores hexadecimales, por lo que es útil tener un selector gráfico de los mismos.

- Mithril Emmet: son snippets inteligentes para crear código de manera rápida en HTML, CSS y Javascript.

- Prettier - Code formatter, nos permite formatear e indentar todo nuestro código, tanto html, js, css. Es instalable como dependencia, pero por el momento no hemos considerado está opción y hemos preferido adaptarlo en el plugin de nuestro IDE.

- Stylelint, este plugin nos permite ejecutar Stylelint en tiempo real, por lo que en principio no necesitaremos ejecutar el comando de npm en desarrollo. Veremos directamente los fallos en nuestro código y las sugerencias de cambio.

4. Desarrollo.

Para abordar el desarrollo propiamente dicho lo primero que haremos es iniciar git en nuestro proyecto con el siguiente comando:

git init

Después añadiremos todos los cambios y haremos commit de los mismos:

git add *

git commit -m "first commit"

Cambiaremos de rama:

git branch -M main

Accederemos a Github y crearemos un nuevo repositorio remoto e introduciremos por consola:

git remote add origin https://github.com/ansango/cv-uoc.git

Por último haremos push del commit establecido:

git push -u origin main

Una vez creado nuestro repositorio remoto y configurado git podremos empezar a escribir código. Para abordar correctamente este proyecto hemos establecido las siguientes fases:

- Contenido HTML: en primer lugar escribiremos todo el contenido html sin clases para hacernos una idea de como vamos a estructurarlo

- Arquitectura CSS:

    - Módulos ITCSS: aquí llevaremos la arquitectura ITCSS explicada en el apartado metodología, que nos ayudará a ir creando las clases selectoras de más reutilizable a menos.

    - Objetos OOCSS: estableceremos la abstracción de objetos y la creación de componentes.

- Javascript: por último añadiremos animaciones o cualquier otra cosa que requiera Javascript.

5. **Compilación para producción.**

   Después de haber realizado la fase de codificación, llevaremos a cabo nuestro primer despliegue de la aplicación. Para ello debemos verificar que todo ha ido bien compilando el proyecto. Ejecutaremos en la raíz del proyecto el siguiente comando:

   npm run build

   Este comando nos generará en la raíz del proyecto, una dependencia llamada **dist,** donde podremos ver toda la compilación. Podremos ver que todo funciona abriendo el index.html en cualquier navegador. Esta dependencia está lista para ser servida en un servidor.

6. **Publicación.**

   Por último una vez compilado el proyecto y viendo que el proceso ha ido correctamente, vamos a desplegarlo en un servidor.

   Para ello hemos utilizado **Netlify**, que nos ayudará a automatizar el sistema de despliegues de sitios estáticos.

   En primer lugar deberemos haber hecho push de los últimos cambios.

   Para configurar **Netlify**, seguiremos los siguientes pasos:

   1. Crearemos un nuevo sitio seleccionando el botón: "**New site from git**"
   2. En "**Continuous Deployment"** seleccionaremos **Github.**
   3. De la lista de repositorios seleccionaremos en que creamos anteriormente.
   4. Seleccionamos la rama **main.**
   5. Escribimos como comando de build: **npm run build.**
   6. Escribimos como directorio de publicación: **dist/**
   7. Y por último seleccionamos el botón "**Deploy Site**"

5. Publicación.

Como comentamos en el último subpunto del punto anterior, después del despliegue tendremos accesible nuestra aplicación en una url real.

En este caso hemos generado la siguiente en **Netlfy** desde el repositorio de **Github:**

https://github.com/ansango/cv-uoc

https://cv-uoc.netlify.app/

El resultado final es el siguiente:

anibalsantos *frontend developer*

Me

Aníbal Santos Gómez

Salamanca, Spain

anibalsantosgo@gmail.com

ansango

Creative and self-starting Frontend Developer with 2 years experience websites and apps in fast-paced, collaborative environments. **Passionate** about usability and possess working knowledge of Javascript & CSS Frameworks.

order, and in particular of the normative contents of the different branches of law.

Skills

HTML5 CSS3 Sass

Javascript Typescript

Git Node Express

Laravel SQL MongoDB

Webpack Parcel Vue.js

Nuxt.js Angular

Tailwind Bootstrap

Bulma Jest Mocha

Figma Sketch UX/UI