Documentación de la Práctica Final

- **Estudios:** Grado en Multimedia
- **Asignatura:** Lenguajes y Estándares Web
- **Proyecto:** Práctica Final
- **autor:** Carlos Rodríguez López
- **email:** crodrigue1011@uoc.edu
- **Profesor:** César Pablo Córcoles Briongos
- **Profesor Colaborador:** Carlos Salgado Werner

Tabla de Contenido

0. Publicación

La página web está disponible a través de los siquientes enlaces:

- Servidor FTP de la UOC
- Repositorio Github

1. Presentación

He planteado la actividad como si se tratase de un encargo real. El encargo empezaba con las siguientes indicaciones:

> Nos han encargado, desde el ayuntamiento de nuestra ciudad, que hagamos una propuesta de la maquetación de una Biblioteca de la Red de Bibliotecas Públicas. El objetivo es que sea el modelo para actualizar el diseño de todas las bibliotecas de la Red, en el cual se ofrece un catálogo actualizado de los libros y recursos, información de las actividades del centro, y la opción de hacer una reserva de libros. El diseño debe ser funcional, usable y accesible, pero también se busca que sea atractivo para el usuario.

Por lo tanto, el objetivo del encargo es crear **un modelo** que sirva para **actualizar las webs de toda una red de bibliotecas.** Eso quiere decir que **se personalizará** el diseño propuesto **y se crearán muchas páginas** a partir de nuestra propuesta.

Teniendo eso en cuenta, me he decantado por utilizar **un preprocesador CSS y un preprocesador HTML.** Aprender a utilizarlos y solventar los problemas que iban surgiendo ha requerido **tiempo y esfuerzo,** pero ha sido a favor de un **valioso aprendizaje,** que era mi objetivo.

Las herramientas utilizadas han sido: **SASS** y **PUG.** Para compilar SASS he utilizado la extensión de VS Code: **Live Sass Compiler** y para compilar PUG, la aplicación **Prepros.**

El **código final** generado, tanto el html, como el css, como las imágenes que habría que publicar, se encuentra en la carpeta: `\procesado`.

Empezaré por explicar el código **html**, pero lo haré sobre la sintaxis de **pug**, que es mucho más limpia y clara. Si no se conoce, basta con saber algunas cosas para seguir la lectura:

- Pug no utiliza etiquetas de cierre, sino identado.
- Incorpora la sintaxis de selectores de CSS: `#id`, `.class`, etc.

Veamos un pequeño **ejemplo de código `pug`:**

```
.container
  h1#titulo Esto sería un título.
  p Esto sería un párrafo normal.
  p.aviso Este párrafo tiene la clase "aviso".
```

El html que genera ese código es el siguiente:

```
<div class="container">
  <h1 id="titulo">Esto sería un título.</h1>
  <p>Esto sería un párrafo normal.</p>
  <p class="aviso">Este párrafo tiene la clase "aviso".</p>
</div>
```

Nótese que la etiqueta `<div class="container">` se crea simplemente **indicando el selector css** de la clase.

Pues bien, vamos a empezar por analizar la plantilla:

Subir

---

2. layout.pug

La estructura básica de la web se construye con la plantilla **`_layout.pug`:**

```
// _layout.pug

- var biblioteca = "Carlos Rodríguez López"
block variables
include partials/mixins
doctype html
html(lang="es")
  include partials/head
  body
    include partials/header
    main
      div.contenedor
        include partials/nav

        if ariadna !== undefined
          p.ariadna!= ariadna

        h1#titulo-pagina!= h1

        block contenido

    include partials/footer
    include partials/scripts
    block scripts
```

Lo primero que hacemos es definir la **variable** `biblioteca`, que será lo primero que habrá que personalizar para cada biblioteca. **Cambiado aquí** el nombre de la biblioteca, estará **cambiado en toda la web.**

A continuación tenemos dos líneas con palabras clave de pug. La primera establece un punto en el que podremos **inyectar contenido** cuando extendamos la plantilla. En este caso, podemos declarar más variables en la página que extienda la plantilla `_layout.pug`.

La otra línea hace lo contrario, **incorpora contenido** que hemos escrito en otro archivo. *Más adelante veremos qué contiene el archivo mixins.*

Sobre la **estructura de la web**, vemos que el elemento **body tiene tres hijos:** `header | main | footer`**.** *(Hay un cuarto elemento,* `scripts`*, pero no forma parte como tal del contenido de la web. Sólo lo usamos para cargar el javascript que hace funcionar el menú desplegable en la versión móvil, aunque para dar más versatilidad a la plantilla, además de incluir (* `include` *) el script necesario, creamos un bloque por si fuera necesario añadir algún otro script desde alguna página.)*

Dentro del elemento `main` hay un `div.container`, que contiene el **menú de navegación principal** (`nav`), el **hilo de Ariadna** o miga de pan (`p.ariadna`), el título de la página `h1#titulo-pagina` y el contenido propio de cada página.

Como no todas las páginas muestran el hilo de ariadna, pero para todas las que lo muestran es un elemento común, se incluye en la plantilla utilizando un **condicional**. Si hemos establecido en una variable el contenido del hilo de ariadna, mostramos el contenido en un párrafo.

El título de la página (`h1`), también es un **elemento común a todas las páginas.** El contenido también lo pasamos a través de una **variable.**

El operador `!=` añade el código *sin escapar.* De modo que podemos pasar código html en esas variables.

Desde esta plantilla cargamos algunos *partials* mediante la etiqueta `include`. Vamos a verlos con más detalle.

Subir

---

3 Partials

El primer `include` que nos encontramos, carga el archivo `mixins.pug`. Sin embargo, se trata de un **archivo especial** que veremos cuando analicemos la página `colecciones.pug`.

3.1 head.pug

El primer *partial* propiamente dicho que nos encontramos es `head.pug`:

```
// partials/head.pug

head
  meta(charset='utf-8')
  meta(name='viewport' content='width=device-width, initial-scale=1.0')
  meta(http-equiv='X-UA-Compatible' content='ie=edge')
  title= titulo
  meta(name='description', content=descripcion )
  // Cargamos nuestro CSS
  link(rel='stylesheet' href='css/estilos.css')
  // Cargamos FontAwesome
  link(rel="stylesheet" href="https://use.fontawesome.com/...")
  // Espacio para cargar otros CSS desde cada página:
  block head
```

Con este código establecemos las etiquetas meta y además usamos una variable para definir el contenido de la etiqueta `title` y otra para el contenido de la *meta description*.

A continuación cargamos nuestro CSS y el necesario para utilizar FontAwesome.

Por último, añadimos un bloque de contenido que podremos extender desde cada página para añadir más contenido dentro de la etiqueta `head`, aunque en el trabajo no ha sido necesario.

3.2 header.pug

Descendiendo directamente de la etiqueta `body`, incluimos el *partial* `_header.pug`:

```
// partials/header.pug

header#header
 .contenedor
   section#logo
     a(href="index.html")
       svg(version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg"...)
         path(d="...")

     p
       a(href="index.html") #{biblioteca}

   section#header-acciones
     section#entrar-buscar
       a#login(href='' title='Acceder') Login

       .buscar
         input.buscar-texto(type="search",name="buscar", placeholder="Buscar...")
         a.buscar-lupa(href="" title="Buscar")

     nav#menu-cabecera
       ul
         li: a(href='informacion.html') La biblioteca
         li: a(href='') Tramites y servicios
         li: a(href='') Hazte el carnet
         li: a(href='') Contacto
```

Como podemos tener otras etiquetas `header` en nuestra web (por ejemplo dentro de un `<article>`), pero la que abre aquí es la más importante, el encabezado de la web, le asignamos una `id` que nos servirá para darle **estilos css**.

Dentro tenemos un `div.contenedor`, que nos servirá para ajustar el ancho del contenido del `header`. Así, aunque el `header` fluirá a todo el ancho del navegador, el contenido lo podremos centrar dándole el ancho que queramos.

A continuación establecemos dos secciones con **entidad semántica propia.** Por un lado tenemos la sección que identifica a la biblioteca *(logo + nombre)* y por otro lado, una sección con enlaces y herramientas.

El logo, en lugar de integrarlo mediante `<img src=''>`, lo he añadido directamente al código. La razón es que de ese modo puedo cambiar la propiedad `fill` y asignar el color mediante **CSS**.

Vemos también que usamos la variable `#{biblioteca}` para establecer el nombre de la biblioteca. Ya hemos visto que esa variable la declaramos directamente en el `_layout.pug`.

Luego tenemos un enlace de **login**, un `input` con un **buscador** y un botón hecho con un enlace al que le asignaremos el icono de la lupa mediante CSS.

Por último, definimos el menú de navegación de la cabecera mediante una lista desordenada contenida en una etiqueta `<nav>`. El código pug que utilizamos se podría llamar `inline`, ya que utilizamos dos etiquetas html en la misma línea, algo que es posible per por facilidad de lectura, se reserva para casos en los que la lectura sigue siendo clara.

3.3 nav.pug

La forma habitual de definir una lista de enlaces sería mediante identado, como veremos en el siguiente código:

```
// partials/nav.pug

.navbar
  nav#menu-principal
    ul
      li
        a(href='colecciones.html') Colecciones
      li
        a(href='') Cat&aacute;logo
      li
        a(href='') Agenda
      li
        a(href='') Salas de estudio
      li
        a(href='') &Aacute;rea infantil
      li
        a(href='') Recursos digitales

    .hamburguesa
      i.fas.fa-bars.fa-2x
```

En este código definimos un contenedor `<div class="navbar">` y dentro de él un `<nav>` con dos elementos: una lista desordenada y un icono de **hamburguesa** *(mediante FontAwesome)*.

3.4 footer.pug

```
// partials/footer.pug

footer#footer
  .contenedor
    ul#pie-biblioteca
      li
        a(href="") Red de bibliotecas
      li
        a(href="") Contacto
      li
        a(href="") Trámites y servicios
    ul#pie-legal
      li
        a(href="") Aviso Legal
      li
        a(href="") Accesibilidad
      li
        a(href="") Protección de datos
```

Para el `footer`, que también será **común a todas las páginas**, un código muy sencillo. Al igual que hicimos con el header, definimos un contenedor que nos facilitará disponer el contenido como necesitemos y dentro de ese contenedor, dos listas de enlaces: una de temática más relacionada con información y servicios de la biblioteca y otra sobre aspectos legales.

Subir

---

4 paginas

A continuación vamos a empezar a explicar el contenido independiente de cada página.

4.1 index.pug

```
// index.pug

extends _layout

block variables
  - var titulo = 'Biblioteca ' + biblioteca + ' - Página principal'
  - var descripcion = 'Página principal de la Biblioteca ' + biblioteca + '.'
  - var h1 = 'Novedades de otoño (...): <a href="ficha.html">El ala izquierda.</a>'

block contenido

  article#noticia-destacada
    a(href="ficha.html")
      img(src="images/destacada.jpg", alt="Portada del libro (...)")

  section#noticias-secundarias

    article
      img(src="images/noticia_1.jpg" alt="Estanterías de una biblioteca.")
      h1: a(href="colecciones.html") Colecciones.
      h2 Novedades del catálogo.
      p Consulta las colecciones de la biblioteca, están disponibles en la web...
```

La página contiene otros cuatro artículos con código similar al primero, por lo que *no merece la pena* copiar aquí todo el contenido.

La primera instrucción indica que el presente archivo usará la plantilla `_layout.pug` que ya hemos analizado. Si recordamos, la plantilla definía una **variable** `biblioteca` y creaba un **bloque de contenido** `variables`. Pues bien, en ese bloque introducimos ahora las variables propias de esta página. Definimos el **título** y la **descripción**, que ya hemos visto que se usarán en el partial `head.pug` y definimos el contenido de la etiqueta `<h1>` principal.

A continuación inyectamos código dentro del bloque `contenido`: Una noticia destacada, que simplemente incluirá una **imagen enlazada**, y una sección de artículos con noticias secundarias.

Cada artículo tendrá una imagen, un título enlazado a una **sección** de la web y un texto.

4.2 colecciones.pug

En esta página probé una característica de **pug** muy interesante:

```
// colecciones.pug
extends _layout

block variables
  - var titulo = 'Biblioteca ' + biblioteca + ' - Colecciones'
  - var descripcion = 'Página que muestra (...) en la biblioteca ' + biblioteca + '.'
  - var h1 = 'Colecciones'
  - var ariadna= '<a href="index.html">Inicio</a> / Colecciones'

block contenido

  section.colecciones
    // Llamamos al mixin colecciones y le pasamos un array con los datos
    +colecciones([
      {
        col: "1",
        titulo: "Mujer",
        alt: "Mujer leyendo sentada en el suelo de una biblioteca",
        parrafo: "Fondo especializado que recoge y difunde todo tipo de información..."
      },
      {
        col: "2",
        titulo: "Cocina",
        alt: "Libro de rectas abierto sobre una encimera con algunos ingredientes",
        parrafo: "Fondo especializado en cocina según alimentos, cocina de autor,..."
      },
      {
        col: "3",
        ...
      },
      {
        col: "4",
        ...
      },
      {
        col: "5",
        ...
      }
    ])
```

Al igual que en la página `index.pug` *(y que en las otras dos páginas)*, empezamos

extendiendo la plantilla `_layout.pug` y definiendo las **variables**. Como en esta página sí que tendremos un hilo de **ariadna**, definimos su valor utilizando **código html**.

Lo interesante de esta página está en el código `+colecciones...`. Lo que hacemos es llamar al `mixin colecciones` pasándole como argumento un **array de objetos**. Cada objeto contiene la información de una de las colecciones que se mostrarán en la página. Es equivalente, por ejemplo, al resultado que obtendríamos mediante una consulta **sql** a una base de datos, o muy similar a los datos en formato **JSON** que obtendríamos como respuesta al usar una **API REST**.

Pues bien, vamos ahora a ver qué hace exactamente el `mixin` con esos datos:

4.2.1 mixins.pug

En este archivo podríamos ir definiendo varios *mixins*, pero ahora mismo, sólo tenemos uno:

```
// partials/mixins.pug

mixin colecciones(items)
    each item in items
        article.coleccion
            img(src='images/col_' + item.col +'.jpg', alt=item.alt)
            .texto
                h1= item.titulo
                p= item.parrafo
```

Un pequeño código muy sencillo, veamos qué hace:

la primera línea es la declaración del *mixin* (lo que en otros lenguajes llamaríamos función) y a todos los datos que recibe, los llamamos: `items`. Es decir, que en la variable `items` tenemos el array de colecciones. Lo que hacemos a continuación es recorrer ese arrary mediante un bucle: *Para cada elemento de items*:

Creamos una etiqueta con la clase .coleccion y dentro ponemos una imagen y un div con un título y un párrafo.

Con estas 7 líneas, generamos todo el contenido de la página `colecciones.html`.

4.3 ficha.pug

En esta página mostramos la ficha de un libro:

```
// ficha.pug

extends _layout

block variables
  - var titulo = 'Biblioteca ' + biblioteca + ' - Ficha libro: El ala izquierda'
  - var descripcion = 'Ficha del libro El ala izquierda. Contiene la portada,...'
  - var h1 = 'Ficha libro'
  - var ariadna = '<a href="index.html">Inicio</a> / Ficha libro: El ala izquierda'

block contenido

  article.ficha

    img.portada(src="images/portada.jpg", alt="Portada del libro El ala izquierda...")

    section
      h1 El ala izquierda
      h2 Mircea Cărtărescu
      ul.datos-tecnicos
        li EDITORIAL: Impedimenta
        li ISBN: 9788417115869
        li FECHA DE PUBLICACIÓN: 3/11/18
      h3 Sinopsis
      p El ala izquierda es el volumen que abre «Cegador», la monumental trilogía...

      section#compartir
        ul
          li#facebook: a(href="http://www.facebook.com/sharer.php?u=http://multimed...")
          li#twitter: a(href="https://twitter.com/home?status=Ya%20disponible%20en%...")
          li#pinterest: a(href="https://pinterest.com/pin/create/button/?url=http%3...")
          li#instagram: a(href="http://instagram.com/_u/biblioteques_bcn/")

  include partials/formulario
```

Al igual que en las páginas anteriores, empezamos extendiendo la plantilla `_layout.pug` y continuamos definiendo las variables.

En el bloque de contenido, ponemos una imagen, que será la **portada** del libro y una **sección de información** sobre el mismo.

A continuación tenemos una sección de iconos para compartir en las **redes sociales**. Se utiliza un *sprite* png para mostrar los logos de las 4 plataformas, así como para hacer el efecto de `:hover` de las mismas, variando su `background-position` para mostrar el contenido adecuado.

En cuanto a la funcionalidad, he buscado la forma de hacer que los botones **funcionen**, que sirvan realmente para compartir el contenido. Lo he conseguido en todos salvo en **Instagram**, ya que se trata de una plataforma que se basa en la publicación de contenido propio y original y su **API** no permite compartir publicaciones. De modo que me he decantado por colocar un enlace a la cuenta de las bibliotecas de Barcelona.

Al final de todo hay un formulario que añadimos incluyendo un *partial*:

4.3.1 formulario.pug

En un archivo aparte, tenemos el formulario de reserva online:

```
// partials/formulario.pug

section#formulario-reserva

  h1 Reserva online

  form(name="reserva" action="#" method="post")

    #div-nombre
      label(for="nombre") Nombre:
      input#nombre(type="text" name="nombre" placeholder="Nombre..." required)

    #div-apellidos
      label(for="apellidos") Apellidos:
      input#apellidos(type="text" name="apellidos" placeholder="Apellidos...")

    #div-dni
      label(for="dni") DNI/NIE:
      input#dni(type="text" name="dni" placeholder="DNI..." maxlength="10" required)

    #div-fecha
      label(for="fecha") Fecha de reserva:
      input#fecha(type="date" name="fecha" required)

    #div-aviso
      label ¿Quieres que te avisemos cuando esté disponible?
      .respuestas
        label #[input#aviso-si(type="radio", name="aviso" value="Sí")] Sí
        label #[input#aviso-no(type="radio", name="aviso" value="No")] No

    #div-recogida
      label(for="recogida") Quiero recogerlo en la siguiente biblioteca:
      select#recogida(name="recogida" required)
        option(value="" disabled selected) Selecciona una biblioteca
        option(value="biblioteca1") Buenaventura Durruti
        option(value="biblioteca2") Francisco Ascaso
        option(value="biblioteca3") García Oliver

    #div-botones
      input(type="submit", value="Enviar")
      input(type="reset", value="Borrar")
```

Se trata de un formulario bastante sencillo. Está compuesto por una serie de *divs* que agrupan los campos del formulario y sus etiquetas.

4.4 informacion.pug

La última de las páginas que forman este boceto es la de información y contacto. El código es el siguiente:

```
// informacion.pug

extends _layout

block variables
  - var titulo = 'Biblioteca ' + biblioteca + ' - Información de la biblioteca'
  - var descripcion = 'Página de información de la biblioteca. Contiene...'
  - var h1 = 'Información'
  - var ariadna = '<a href="index.html">Inicio</a> / Biblioteca'

block contenido

  article#informacion

    section#descripcion
      p La Biblioteca es un equipamiento cultural y de proximidad abierto a todos...
      p La Biblioteca está situada en el interior de una isla de la Derecha del...
      p La primera Biblioteca Sofia Barat abrió sus puertas el año 1971 en un...

    section#otros-datos
      table#horario
        tr
          th Días
          th Horarios
        tr
          td Lunes y Viernes
          td de 16:00h a 20:30h
        tr
          td Martes y miércoles
          td de 10:00h a 14:00h y,
            br
            | de 16:00h a 20:30h
        tr
          td Jueves
          td de 10:00h a 20:30h
        tr
          td Sábado
          td de 10:00h a 14:00h

      section#contacto
        p Dirección:
          span C Girona, 64*68 LI
        p Distrito:
          span Eixample
        p Barrio:
          span La Derecha del Eixample
```

```
        p Código postal:
          span 08009
        p Población:
          span Barcelona
```

En el bloque de contenido de esta página, tenemos dos secciones que se dispondrán en un diseño de dos columnas. En la columna de la derecha hay una **tabla** con los **horarios de apertura** y un bloque de **información de contacto.**

Hay un elemento de la sintaxis de `pug` que todavía no habíamos visto: En la tercera fila, el símbolo `|` se utiliza para indicar que el contenido de esa fila contínua desde la anterior. No se trata de una etiqueta nueva. El problema es que al querer añadir un `<br />` para dividir el texto de la celda en dos líneas, si el br se coloca en la fila de arriba, lo interpretará como texto plano, y si se escribe texto a continuación, lo interpretará commo una etiqueta: `<br>texto</br>`. Para solucionar este problema, se coloca esa barra vertical.

Subir

---

5 SCSS

Para **generar, mantener y agilizar** la escritura del código CSS, opté, como comentaba al principio, por utilizar **SCSS.**

En este caso se ha optado por escribir todo el código en un mismo archivo, aunque seguramente sería interesante haberlo separado en distintos archivos. En cualquier caso, se ha estructurado el contenido interno para que sea **fácil navegar** por sus más de 800 líneas.

La mayor ventaja de SCSS para este proyecto es que podemos, fácil y rápidamente, hacer cambios que afectan a **todo el diseño**, sin tener que ir buscando por todo el código.

Para ello, nos valemos del uso de **variables.** Es lo primero que definimos y las que he usado son las siguientes:

```
// ------------------------------------------------
//
//                    Variables
//
// ------------------------------------------------
//   Aquí se definen variables que permiten cambiar el
//   aspecto de muchos elementos rápida y cómodamente.
// ------------------------------------------------
//
//   Paleta de Colores:
//
$principal: #7a2800;
$secundario: #661900;
$resaltado: #c95100;
$oscuro: #2e0f00;
$claro: #ffc6aa;
//
// - - - - - - - - - - - - - - - - - - - - - - - - -
//
//   Fuentes Tipográficas:
//
@import url('https://fonts.googleapis.com/css?family=Asul|Chivo:900');
$font1: 'Asul', sans-serif;
$font2: 'Chivo', sans-serif;
//
// - - - - - - - - - - - - - - - - - - - - - - - - -
//
//   Valor para los border-radius:
//
$radius: 20px;
//
```

Como se puede apreciar, ahí definimos **los colores, las fuentes y el radio que** se le dará a los bordes. Por lo que, si el que se ha elegido de `20px` resultase excesivo, sólo con cambiar ese valor, se cambiarían todos los bordes de la web.

Del mismo modo, si se quieren modificar los colores o las tipografías, sería tan fácil como cambiarlas en **un único lugar.**

Lo siguiente he usado ha sido:

```
//
//   Contenedores flex
//
%flex-justify-between {
  display: flex;
  justify-content: space-between;
}

%flex-justify-center {
  display: flex;
  justify-content: center;
}
```

Más como práctica que por la utilidad real, ese código permite aplicar dos propiedades css escribiendo una sólida línea. Sin duda, su utilidad radicaría en aplicar muchas más propiedades, pero con ello también se pierde **versatilidad.**

El resto del código, es muy similar al CSS, lo único que varía es el uso del **anidamiento,** (o *nesting*). Por ejemplo, el código:

```
a {
  color: $principal;
  font-family: $font2;
  &:hover {
    color: $resaltado;
  }
}
```

Se convierte en:

```
a {
  color: $principal;
  font-family: $font2;
}
a:hover {
  color: $resaltado;
}
```

Eso hace que el código sea mucho más fácil de leer, de interpretar y de mantener.

He utilizado **flexbox** para posicionar todos los elementos. También lo he usado para la **versión móvil**, para lo cual, muchas veces ha sido suficiente con añadir en el *media querie*

correspondiente: `flex-direction: column;`.

El código es demasiado **extenso** para comentarlo pormenorizadamente, pero sí comentaré a grandes rasgos algunos estilos utilizados que habría que tener en cuenta si se quisiera implementar el diseño en un **proyecto real.**

Las dos fuentes elegidas, contrastan bastante entre ellas. Una, la `$font2: 'Chivo', sans-serif;`, se ha seleccionado de entre su familia tipográfica, aquella versión con un **mayor peso.** La otra, se utiliza con distintos tamaños, consiguiendo un buen contraste entre título y cuerpo.

Es decir, en general se ha usado para todo el contenido la `$font1`. La `$font2`, únicamente se ha utilizado allí donde quería crearse un **mayor contraste** que llamase la atención.