# Índice

# Documentación del proceso de desarrollo

Para el desarrollo de la página se han seguido los siguientes pasos:

## Gestor de paquetes

El punto de partida de cualquier proyecto de desarrollo web actual es un gestor de paquetes. En este caso he instalado NodeJS versión 14.15.5 que incorpora como gestor de paquetes npm vesrsión 6.14.11 en un entorno Windows10.

## Module bundler

Instalación en local de Parcel versión 1.12.3 como module bundler.
npm install parcel-bundler@1.12.3 –save-dev

Se ha configurado para trabajar en dos entornos:
Modo producción: realiza tareas de pre-compiler, concatenación y minificado de archivos css y jacascript.
Modo desarrollo: realiza tareas de pre-compiler, concatenación sin minificar y lanza un servidor de desarrollo que permite monitorizar los cambios en los archivos refrescando el navegador automáticamente.

## Dependencias

### Rimraf

Parcel después de realizar sus tareas deja el resultado en la carpeta dist y para el control del refresco del navegador en una serie de carpetas de cache. Una buena práctica para evitar errores inesperados es que cada vez que iniciamos el modo desarrollo se vacíen estas carpertas.
Para facilitar esta tarea se ha instalado la dependencia rimraf

npm install --save-dev rimraf

### Npm-run-all

Además para poder realizar la operación de limpieza antes del build en un solo comando se ha instalado npm-run-all. Esta dependencia permite la ejecutar varios scripts desde un solo comando.

Npm install --save-dev npm-run-all

Para el correcto funcionamiento de las dependencias deben configurarse los scripts en el archivo package.json:

```
"scripts": {
    "parcel:dev": "parcel PEC1/index.html",
    "parcel:build": "parcel build PEC1/index.html",
    "dev": "npm-run-all clean parcel:dev",
    "build": "npm-run-all clean parcel:build",
    "clean": "rimraf dist .cache .cache-loader",
}
```

## PostCSS

Los preprocesadores que generan CSS se usan entre otras cosas para añadir funcionalidades que no existen en CSS estándar. Para esta práctica utilizaré PostCSS que ya viene incluido por defecto en Parcel.

## Autoprefixer

Uno de los grandes problemas del css es la compatibilidad entre navegadores y sobretodo cuando entran en escena versiones antiguas. Para conseguir la compatibilidad es necesario añadir gran cantidad de lineas de código para que nuestros estilos estén soportados en diferentes entornos.
La dependencia autoprefixer se integra con PostCss y analiza los archivos CSS agregando prefijos de proveedores a las reglas utilizando una base de datos.
npm install --save-dev autoprefixer@9.8.6

Para configurar PostCss para que funcione en combinación con autoprefixer se ha añadido el archivo de configuración: .postcssrc con el siguiente contenido:

```
{
   "plugins": {
   "autoprefixer": true
   }
}
```

## Postcss-css-variables

Otro de los problemas que surge con los navegadores más antiguos de que no permiten trabajar con variables en css. Como autoprefixer no soluciona este problema se ha instalado una dependencia adicional: postcss-css-variables, que sustituye las variables por sus valores en tiempo de precompilación.

npm install --save-dev postcss-css-variables

Se ha modificado el archivo de configuración de postcss para que tenga en cuenta el nuevo módulo

```
{
   "plugins": {
   "autoprefixer": true,
   "postcss-css-variables": {}
   }
}
```

## Babel

De la misma forma que Parcel incorpora un preprocesador de Css también incorpora uno de JavaScript. En este caso es Babel, como es suficiente para el alcance de esta práctica no se ha instalado ninguno adicional.

Tanto autoprefixer como Babel generan código para que sea correctamente interpretado por los navegadores mas utilizados y a partir de las versiones que precisemos. Para ellos se debe configurar un archivo .browserslistrc

Para dar soporte a versiones antiguas de los navegadores más conocidos se han puesto directamente en el archivo de configuración, para los demás solo de ha dado soporte a las 4 últimas versiones:

last 4 version, IE 6, firefox 16, chrome 10, safari 5, opera 11, ios_saf 5

### Prettier

En algunos casos el editor no deja una buena identación y distribución del código. Esto hace que el desarrollador pierda tiempo ajustándolo para que tenga un estilo correcto.
Para facilitar el trabajo se ha añadido la dependencia prettier

npm install –save-dev

Además se ha añadido el script en el archivo package.json
"format": "prettier --write \"PEC1/**/*.*\""

En este caso no se ha añadido la ejecución del script en la automatización del build ya que no se considera necesaria su ejecución para cada modificación de código.

### Viewerjs

Al trabajar con muchas imagenes se ha añadido un módulo para poder visualizarlas en carrousel y se ha añadido a la página de presentación. Se llama vieverjs y se instala con:
npm install –save-dev viewerjs

El archivo js/visor.js contiene el javascript que permite su integración con la página. Se ha utilizado en la página de presentación.

## Editor de código

Se ha utilizado VisualStudioCode versión 1.54.1

## Repositorio de datos y servidor en producción

Se utiliza GitHub. Para la integración con el repositorio se ha instalado y configurado GitBash

Se ha añadido un archivo .gitignore para evitar enviar al repositorio los archivos de las carpetas node-modules y las caches de trabajo de Parcel

Como servidor de producción se utiliza Netlify

## Distribución de archivos

Se ha creado una carpeta PEC1 donde se encuentran todos los archivos de desarrollo de la web. Los archivos HTML de han ubicado directamente en este directorio. Se han creado subdirectorios para contener los archivos SCSS, JS y img respectivamente.

## Código

Se ha utilizado el diseño movile-first y para conseguir que las páginas sean responsive cajas flexibles  y en grid junto con media-querys.
Se ha intentado aplicar lo aprendido en la asignatura Herramientas CSS y HTML II en cuanto a la nomenclatura y formato de código de selectores CSS. En concreto la guía mdo y la nomenclatura propuesta por BEM.

## Proceso Desarrollo

Durante el desarrollo las modificaciones grandes se realizaban directamente en el código, con parcel en modo desarrollo para ver los resultados directamente en el navegador.
Los ajustes pequeños como cambios de tamaños, padding, margins, colores, etc.. se realizaban primero en las herramientas de desarrollo de Firefox para luego trasladarlas al código fuente.

Cuando concluía el desarrollo de una página se ejecutaba Parcel en modo producción para comprobar los cambios en el código.

Se ha utilizado Github como repositorio. Para ellos se ha tenido que instalar la herramienta Git Bash.
Desde la herramienta se han ejecutado los siguientes comandos:
Inicializar el directorio local como un repositorio Git.
$ git init -b main
Añadir los archivos al repositorio local
$ git add .
Commit de los archivos
$ git commit -m "Primer commit"
Configuración del repositorio remoto
$ git remote add origin  https://github.com/ptousd/eines2.git
# Vericación de la URL remota
$ git remote -v
Envio de los cambios del repositorio local al remoto
$ git push origin main

Se puede acceder al repositorio en:
https://github.com/ptousd/eines1

Para paso a producción se ha utilizado Netlify.
Se puede acceder al proyecto en la url:
https://laughing-feynman-33e088.netlify.app

Una vez en producción se ha analizado el código con las siguientes herramientas:
Para corregir problemas de html
https://validator.w3.org/

Para corregir problemas de css
https://jigsaw.w3.org/css-validator

Para validar accesibilidad de las páginas online
https://achecker.ca/checker/index.php

A partir de aquí se ha hecho Continuous Deployment hasta conseguir que los validadores no encontrasen problemas.

# Justificación de las decisiones tomadas en el desarrollo

## Decisiones sobre el  boilerplate

Se ha utilizado windows 10 a la espera de que se generasen problemas con npm, pero al no haberse producido no he cambiado a entorno Linux.

La versión 1.12.4 de Parcel si que me dio los mismos problemas que algunos compañeros con Ubuntu pero los solucioné instalando la versión 1.12.3.

Como he comentado he considerado los precompiladores babel y postcss adecuados para el tamaño del ejercicio.
Se ha utilizado la versión 9.8.6 de autoprefixer por los problemas de compatibilidad con Parcel 1 y postCSS 8.
Me gusta la idea de trabajar con variables para definir los parámetros principales del skin,pero estas no son interpretadas por navegadores. Postcss por si solo no hace ningún tipo de sustitución.
Para poder trabajar con variables he instalado un módulo que se integra con postcss: postcss-css-variables.

En cuanto al uso en navegadores antiguos, en pruebas he configurado el sistema para trabajar hasta con IE6. Como en la actualidad su uso es muy reducido he terminado configurándolo  para trabaja con IE10 , soporte para las últimas 4 versiones con soporte y para navegadores con al menos un 1% de cuota de accesos.

## Decisiones de estructura de las páginas

Para el desarrollo de la práctica se ha seguido la guía de estilos de mdo en combinación con la metodología BEM para el nombrado y definición de los selectores. Se pretende con ello reducir la especifidad de los selectores y aumentar la reutilización, además de generar un código comprensible para otros desarrolladores que conozcan dichas medodologías.

Se ha utilizado un desarrollo first-mobile ya que he considerado que es la mejor manera de que se vea bien en dispositivos pequeños.

Se han utilizado dos puntos de ruptura con media-queries: el primero es para pantallas de resolución 768px que son las que mas se aproximan a un entorno tablet, y el segundo para 1440px que es para equipos de escritorio. Se han utilizado esto puntos de ruptura ya que se consideran estándares.

Primero se ha desarrollado la web sin puntos de ruptura y probado en dispositivos tipo móvil. En segundo lugar se ha añadido el punto a 768px para adaptar el funcionamiento a tables, para finalmente añadir el punto de ruptura para visualizar en formato escritorio.

En todas las páginas se ha utilizado la semántica de html en la que se dividen las zonas en header, nav, main, footer y section para facilitar su accesibiliad.

### Index.html (Portada)

La página home es muy sencilla así que se ha decidido que se ajuste al 100% del viewport invalidando la posibilidad de realizar scrolls. Como el tamaño de la imagen de background es elevado se han creado tres versiones de jpg que se utilizan según las dimensiones del dispositivo por media-queries.

### Categorias.html

En la página de categorías, en la lista se ha utilizado display block para dispositivos pequeños, un grid de dos columnas para dispositivos medianos y un grid de tres columnas en dispositivos grandes. Se ha utilizado grid en lugar de flex ya que da un aspecto más homogéneo a toda la página. Para ajustar cada uno de los items de la lista de categorías se ha utilizado un display en flex. Se ha utilizado esta opción ya que de esta manera se rellena mejor el espacio.

### Detalle.html

En la página detalle se ha utilizado un formato de dos columnas en grid para dispositivos grandes y una columna en block para pequeños – medianos

### Presentacion.html

La página de presentación tienen un formato similar, utilizado un formato de dos columnas en grid para dispositivos grandes y una columna en block para pequeños – medianos. A esta página se le ha añadido la posibilidad de ver las imágenes en formato carrousel pulsando sobre cualquiera de ellas. En lugar de hacer un desarrollo propio se han buscado dependencias de npm y se ha utilizado finalmente: viewerjs. En esta página se encuentra también un pequeño script para lanzar la funcionalidad.

### Enlaces.html

Finalmente para la página enlaces se ha utilizado la misma composición que en la lista de categorías.

Se ha optado por utilizar los modernos mecanismos para la adaptación responsive: display flex y display grid ya que reducen mucho la complejidad de desarrollo frente al position: relative. Se ha considerado también para tomar esta decisión que actualmente está soportado por la mayoría de navegadores .

## Explica los resultados obtenidos al aplicar estas decisiones

La primera decisión y la mas importante es haber utilizado un boilerplate moderno cuyo resultado inmediato es la reducción de plazos de desarrollo.

La utilización de parcel como  module bundler reduce los errores de codificación al realizar un precompilado de archivos css y javascript, permitiendo la detección de estos antes de su ejecución.

La posibilidad de monitorizar los cambios en los archivos refrescando el navegador automáticamente que proporciona parcel en modo desarrollo, facilita mucho el trabajo de diseño de las páginas ya que se pueden visualizar inmediatamente todos los cambios.

Utilizar postcss con autoprefixer permite olvidarse de las reglas css específicas de navegadores en versiones mas antiguas.
Un ejemplo de conversión:

```
.receta__paso {
 display: flex;
 flex-direction: row;
 justify-content: space-between;
 margin-right: 4rem;
 border-bottom: 2px dotted #ddd;
}
```

Se convierte a:

```
.receta__paso {
 display: -webkit-box;
 display: -moz-box;
 display: -ms-flexbox;
 display: flex;
 -webkit-box-orient: horizontal;
 -webkit-box-direction: normal;
   -moz-box-orient: horizontal;
   -moz-box-direction: normal;
    -ms-flex-direction: row;
       flex-direction: row;
 -webkit-box-pack: justify;
   -moz-box-pack: justify;
    -ms-flex-pack: justify;
       justify-content: space-between;
 margin-right: 4rem;
 border-bottom: 2px dotted #ddd;
}
```

Otro de los problemas que surge con los navegadores más antiguos de que no permiten trabajar con variables en css. Como autoprefixer no soluciona este problema se ha instalado una dependencia adicional: postcss-css-variables, que sustituye las variables por sus valores en tiempo de precompilación.
De esta forma obtenemos los beneficios de trabajar con variables pero no perdemos compatibilidad con los navegadores antiguos que no las soportan.

Por ejemplo:

```
:root {
 --bg-color-1: #87ceeb;
 --bg-color-2: #05668b;
 --box-shadow-color: #023f57;
 --font-highlight-color: #a01127;
 --bg-highlight-color: rgba(255, 255, 255, 0.8);
}

body {
margin: 0;
background: linear-gradient(to right bottom,var(--bg-color-1),var(--bg-color-2));
font-family: 'Montserrat', serif;
}
```

se convierte a

```
body {
 margin: 0;
 background: -webkit-gradient(linear, left top, right bottom, from(#87ceeb), to(#05668b));
 background: -webkit-linear-gradient(left top, #87ceeb, #05668b);
 background: -o-linear-gradient(left top, #87ceeb, #05668b);
 background: linear-gradient(to right bottom, #87ceeb, #05668b);
 font-family: "Montserrat", serif;
}
```

El uso de un precompilador como babel también ha simplificado el trabajo ya que se puede trabajar con las versiones más actuales de javascript sin preocuparse de si será correctamente interpretado por los navegadores mas antiguos.

Veamos un ejemplo:
El javascript utilizado para el carrousel de fotos en la página de presentación está codificado en ES6:

```
const gallery = new Viewer(document.getElementById('images'), {
 title: [
  4,
  (image, imageData) => `${image.alt} (${imageData.naturalWidth} × ${imageData.naturalHeight})`,
 ],
});
```

y al tener configurado browserlist para navegadores más antiguos ha transformado el código en:

```
var gallery = new _viewerjs.default(document.getElementById('images'), {
 title: [4, function (image, imageData) {
  return "".concat(image.alt, " (").concat(imageData.naturalWidth, " \xD7 ").concat(imageData.naturalHeight,
")");
 }]
});
```

- Const se transforma en var
- Narrow Function a function
- La plantilla de string a cadena con concats

Para finalizar cuando utilizamos parcel se reduce el número de ficheros ya que compacta automáticamente los archivos scss en uno solo y además, cuando generamos con build, minifica los js y css, lo que reduce el tiempo de carga en los navegadors.

Github permitiría trabajar en equipo con un repositorio compartido.

Netlify nos permite tener un hosting de las páginas adaptado a node.js.