PEC 1 - HTML y CSS I

Desarrollo de un Sitio Web: Viajes por Europa

Estudiante: Federico Javier Martino **Asignatura:** HTML y CSS I **Universidad:** UOC (Universitat Oberta de Catalunya) **Fecha:** Noviembre 2025

Enlaces del proyecto:

- **Repositorio GitHub:** https://github.com/Federicojaviermartino/PEC_HTML_Y_CSS_I_Martino_Federico_Javier
- **Sitio web público:** https://pechtmlycssimartinofedericojavier.netlify.app/

Índice

---

1. Proceso de desarrollo con Parcel

1.1 Elección de Parcel como bundler

Para este proyecto se eligió Parcel como herramienta de empaquetado por varias razones técnicas:

- **Zero configuration:** No requiere archivos de configuración complejos como Webpack
- **Soporte nativo para SCSS:** Detecta y compila automáticamente archivos SASS/SCSS
- **Hot Module Replacement:** Recarga automática del navegador durante el desarrollo
- **Optimización automática:** Minificación y tree-shaking en modo producción

1.2 Instalación y configuración inicial

La instalación de Parcel se realizó mediante npm como dependencia de desarrollo:

```
npm install --save-dev parcel
```

Se configuraron dos scripts principales en el archivo `package.json`:

```
{
  "scripts": {
    "dev": "parcel src/*.html --open",
    "build": "parcel build src/*.html --public-url ./"
  }
}
```

El parámetro `--open` abre automáticamente el navegador en modo desarrollo, y `--public-url ./` asegura que las rutas sean relativas en producción.

1.3 Estructura del proyecto

Se organizó el código fuente en una carpeta `src/` con la siguiente estructura:

```
src/
├── index.html
├── categoria.html
├── det1.html
├── det2.html
├── links.html
├── styles/
│   ├── main.scss
│   ├── _variables.scss
│   ├── _mixins.scss
│   └── _layout.scss
├── scripts/
│   └── main.js
└── images/
```

Parcel procesa automáticamente todos los archivos HTML especificados y genera la carpeta `dist/` con el código optimizado para producción.

---

2. Entornos de desarrollo y producción

2.1 Entorno de desarrollo

El entorno de desarrollo se activa con el comando:

```
npm run dev
```

Características del entorno de desarrollo:

- **Servidor local:** Inicia un servidor en `http://localhost:1234`
- **Hot Module Replacement:** Los cambios en el código se reflejan instantáneamente sin recargar la página
- **Source maps:** Los archivos incluyen source maps para facilitar el debugging

- **Sin minificación:** El código permanece legible para facilitar la depuración

2.2 Entorno de producción

El build de producción se genera con:

```
npm run build
```

Optimizaciones aplicadas automáticamente por Parcel:

- **Minificación:** HTML, CSS y JavaScript se minimizan para reducir el tamaño
- **Tree shaking:** Eliminación de código no utilizado
- **Compresión de imágenes:** Optimización automática de assets
- **Hashing de archivos:** Los archivos generados incluyen hash para cache busting
- **Code splitting:** Separación de código para carga optimizada

La carpeta `dist/` resultante contiene todos los archivos listos para desplegar en un servidor web.

---

3. Soporte a navegadores antiguos

3.1 Configuración de browserslist

Para asegurar la compatibilidad con navegadores antiguos, se definió la lista de navegadores objetivo en `package.json`:

```
{
  "browserslist": [
    "defaults",
    "not IE 11",
    "maintained node versions"
  ]
}
```

Esta configuración indica:

- **defaults:** Navegadores con más del 0.5% de uso global
- **not IE 11:** Se excluye Internet Explorer 11 por razones de compatibilidad con módulos modernos
- **maintained node versions:** Versiones de Node.js con soporte activo

3.2 Transpilación automática

Parcel incluye transpilación automática de JavaScript mediante su transformador interno, que convierte características modernas de ES6+ a versiones compatibles con navegadores antiguos:

- Arrow functions → funciones tradicionales
- Template literals → concatenación de strings
- const/let → var cuando es necesario

- Destructuring → asignaciones tradicionales

3.3 Autoprefixing de CSS

El CSS generado incluye automáticamente vendor prefixes para propiedades que lo requieren:

```
/* Input */
.element {
  display: flex;
  transform: translateY(-2px);
}

/* Output con autoprefixing */
.element {
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
  -webkit-transform: translateY(-2px);
  transform: translateY(-2px);
}
```

---

4. Preprocesadores y postprocesadores

4.1 SCSS como preprocesador

Se utilizó SCSS (Sassy CSS) como preprocesador de CSS, lo que permitió aprovechar características avanzadas:

Variables

Definición centralizada de colores, fuentes y espaciados en `_variables.scss`:

```
$primary-color: #2c7da0;
$secondary-color: #f77f00;
$font-primary: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
$spacing-md: 1.5rem;
```

Mixins reutilizables

Creación de mixins para patrones comunes en `_mixins.scss`:

```
@mixin respond-to($breakpoint) {
  @if $breakpoint == mobile {
    @media (min-width: 576px) {
      @content;
    }
  }
```

```
}

@mixin button-style {
  display: inline-block;
  padding: $spacing-sm $spacing-lg;
  background-color: $primary-color;
  border-radius: $border-radius;
  transition: $transition;
}
```

Anidamiento y modularización

El uso de @use permite importar módulos de forma moderna:

```
@use 'variables' as *;
@use 'mixins' as *;
@use 'layout';
```

4.2 Autoprefixer

Parcel incluye Autoprefixer como postprocesador, que añade automáticamente los vendor prefixes necesarios basándose en la configuración de browserslist.

4.3 Ventajas obtenidas

- **Mantenibilidad:** El código CSS es más organizado y fácil de mantener
- **Reutilización:** Los mixins y variables evitan repetición de código
- **Consistencia:** Los valores centralizados aseguran coherencia visual
- **Productividad:** Escribir menos código gracias a las características de SCSS

---

5. Dependencia externa: Leaflet

5.1 Justificación de la elección

Se eligió Leaflet como librería externa por las siguientes razones:

- **Ligera:** Sólo 42KB comprimido, mucho más liviana que Google Maps
- **Open source:** Licencia BSD de código abierto
- **Compatible con móviles:** Diseñada para funcionar perfectamente en dispositivos táctiles
- **Flexible:** Fácil personalización y extensión
- **Sin API keys:** No requiere claves de API ni costes asociados

5.2 Instalación

```
npm install leaflet
```

5.3 Implementación

La integración se realizó en el archivo `main.js`:

```
import L from 'leaflet';
import 'leaflet/dist/leaflet.css';

function initMap() {
  const map = L.map('map').setView([41.3874, 2.1686], 13);

  L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '© OpenStreetMap contributors',
    maxZoom: 19
  }).addTo(map);

  const marker = L.marker([41.3874, 2.1686]).addTo(map);
  marker.bindPopup('<b>Barcelona</b>').openPopup();
}
```

5.4 Valor añadido al proyecto

Los mapas interactivos aportan:

- **Contexto geográfico:** Los usuarios pueden visualizar la ubicación exacta de los destinos
- **Interactividad:** Zoom, pan y marcadores clicables mejoran la experiencia
- **Información adicional:** Los popups muestran datos relevantes de cada ubicación
- **Profesionalidad:** Añade un componente visual atractivo y funcional

---

6. Semántica y accesibilidad

6.1 HTML semántico

Se utilizaron elementos HTML5 semánticos para estructurar correctamente el contenido:

```
<header>
  <nav aria-label="Navegación principal">
    <ul>
      <li><a href="index.html">Inicio</a></li>
    </ul>
  </nav>
</header>

<main id="main-content">
  <article>
    <section>
      <h2>Título de sección</h2>
      <p>Contenido...</p>
    </section>
  </article>
```

```
</main>

<footer>
  <p>&copy; 2025 Viajes por Europa</p>
</footer>
```

Elementos semánticos utilizados:

- `<header>`: Encabezado principal del sitio
- `<nav>`: Navegación principal
- `<main>`: Contenido principal de cada página
- `<article>`: Contenido independiente y autocontenido
- `<section>`: Secciones temáticas dentro del contenido
- `<footer>`: Pie de página con información complementaria

6.2 Cumplimiento WCAG 2.0 AA

Skip links

Enlace para saltar la navegación y acceder directamente al contenido:

```
<a href="#main-content" class="skip-link">Saltar al contenido principal</a>
```

Atributos alt en imágenes

Todas las imágenes incluyen texto alternativo descriptivo:

```
<img src="barcelona.jpg" alt="Vista de la Sagrada Familia en Barcelona">
```

Contraste de colores

Se verificó que todos los textos cumplan con el ratio mínimo de contraste 4.5:1:

- Texto principal sobre fondo claro: ratio 7.2:1
- Botones primarios: ratio 5.8:1
- Enlaces: ratio 6.1:1

Navegación por teclado

Todos los elementos interactivos son accesibles mediante teclado:

```
image.addEventListener('keypress', function(e) {
  if (e.key === 'Enter' || e.key === ' ') {
    e.preventDefault();
    this.click();
```

```
  }
});
```

Atributos ARIA

Se utilizaron atributos ARIA donde fue necesario:

```
<button class="menu-toggle"
        aria-label="Abrir menú de navegación"
        aria-expanded="false">
  <span aria-hidden="true">☰</span>
</button>

<div class="lightbox"
     role="dialog"
     aria-modal="true"
     aria-label="Visor de imágenes">
</div>
```

Idioma del documento

Se especificó correctamente el idioma:

```
<html lang="es">
```

6.3 Validación HTML

Todas las páginas HTML fueron diseñadas para pasar la validación del W3C, utilizando:

- Doctype correcto
- Elementos correctamente anidados
- Atributos requeridos presentes
- Valores de atributos válidos

---

7. Git y GitHub

7.1 Inicialización del repositorio

Se inicializó un repositorio Git local:

```
git init
```

7.2 Configuración de .gitignore

Se creó un archivo `.gitignore` para excluir archivos generados y dependencias:

```
node_modules/
dist/
.parcel-cache/
.cache/
.DS_Store
*.log
```

Esto asegura que sólo el código fuente se versione, no los archivos generados o las dependencias que pueden reinstalarse.

7.3 Estrategia de commits

Se realizaron commits descriptivos y atómicos:

```
git add .
git commit -m "Estructura inicial del proyecto web de viajes"
```

Cada commit representa un conjunto coherente de cambios relacionados.

7.4 Publicación en GitHub

El repositorio se publicó en GitHub siguiendo estos pasos:

1. Creación del repositorio en GitHub.com
2. Conexión del repositorio local con el remoto:

```
git remote add origin https://github.com/usuario/repositorio.git
git push -u origin master
```

7.5 Ventajas del control de versiones

- **Historial completo:** Registro de todos los cambios realizados
- **Colaboración:** Facilita el trabajo en equipo
- **Respaldo:** El código está almacenado de forma segura en la nube
- **Despliegue:** GitHub se integra con plataformas de hosting para deployment automático

---

8. Publicación en internet

8.1 Elección de plataforma: Netlify

Se eligió Netlify como plataforma de hosting por:

- **Continuous deployment:** Actualización automática al hacer push a GitHub

- **CDN global:** Distribución rápida del contenido
- **HTTPS gratuito:** Certificado SSL automático
- **Preview deployments:** Previsualización de cambios antes de publicar
- **Facilidad de uso:** Configuración simple y rápida

8.2 Configuración del deployment

Pasos realizados:

1. Registro en Netlify.com
2. Conexión con la cuenta de GitHub
3. Selección del repositorio del proyecto
4. Configuración de build:
    - **Build command:** `npm run build`
    - **Publish directory:** `dist`
5. Deploy del sitio

8.3 Continuous deployment

Una vez configurado, cada push a la rama principal de GitHub desencadena automáticamente:

1. Clone del repositorio
2. Instalación de dependencias (`npm install`)
3. Ejecución del build (`npm run build`)
4. Publicación de la carpeta `dist/`
5. Invalidación de caché del CDN

Este proceso asegura que el sitio web siempre refleja la última versión del código.

8.4 Verificación del sitio en producción

Se verificó que el sitio funciona correctamente:

- ✓ Todas las páginas se cargan correctamente
- ✓ Los estilos CSS se aplican
- ✓ JavaScript funciona (menú, lightbox, mapas, búsqueda)
- ✓ Las imágenes se muestran
- ✓ El diseño responsive funciona en diferentes dispositivos
- ✓ Los enlaces internos y externos funcionan
- ✓ El sitio es accesible mediante HTTPS

8.5 URL del sitio público

El sitio está disponible en: **https://pechtmlycssimartinofedericojavier.netlify.app/**

Conclusiones

Este proyecto ha permitido poner en práctica los conocimientos adquiridos en la asignatura HTML y CSS I, implementando:

- Un flujo de desarrollo moderno con Parcel
- Preprocesadores CSS para mejor organización del código
- JavaScript vanilla para interactividad
- Integración de librerías externas
- Buenas prácticas de accesibilidad y semántica
- Control de versiones con Git
- Despliegue continuo en producción

El resultado es un sitio web completamente funcional, responsive, accesible y desplegado en internet, cumpliendo con todos los requisitos de la práctica.