PEC 3

Herramientas HTML y CSS 2

PEC 2
https://herramientas-html-css-2-p2.netlify.app/
https://github.com/imsantigarcia/htmlcss-2

PEC 3
https://htmlcss3final.netlify.app/
https://github.com/imsantigarcia/htmlcss3

Tabla de contenido

1. Preparación del Entorno y Configuración

Para el desarrollo de esta práctica, se ha partido de un nuevo proyecto basado en UOC Boilerplate, siguiendo los requisitos del enunciado.

```
~/Library/CloudStorage/OneDrive-Personal/Máster/Herramientas HTML y CSS 2/P3 git:(main)  6 files changed, 39 deletions(-)  (0.927s)
git clone https://github.com/uoc-advanced-html-css/uoc-boilerplate.git

Cloning into 'uoc-boilerplate'...
remote: Enumerating objects: 395, done.
remote: Counting objects: 100% (188/188), done.
remote: Compressing objects: 100% (62/62), done.
remote: Total 395 (delta 159), reused 126 (delta 126), pack-reused 207 (from 3)
Receiving objects: 100% (395/395), 1.16 MiB | 5.95 MiB/s, done.
Resolving deltas: 100% (214/214), done.
```

1.1. Instalación de Dependencias

El proyecto utiliza la versión más reciente de Tailwind (v4.1.18), lo que implica un cambio de paradigma respecto a versiones anteriores. La configuración se ha realizado siguiendo el enfoque "CSS-first":

- Dependencias: Se instalaron tailwindcss y @tailwindcss/postcss para integrarse con el flujo de compilación de Parcel.

- Configuración: En lugar de un archivo JavaScript extenso, la configuración del tema se define directamente en CSS nativo dentro de src/assets/styles/main.css. Se utilizó la directiva @theme para declarar las variables de color personalizadas (usando el espacio de color oklch) y sobreescribir los valores predeterminados del framework.

```
v24.10.0 ~/Library/CloudStorage/OneDrive-Personal/Máster/Herramientas HTML y CSS 2/P3/uoc-boilerplate git:(main)  (2.72s)
npm install -D tailwindcss postcss autoprefixer

added 66 packages, removed 1 package, changed 8 packages, and audited 437 packages in 3s

259 packages are looking for funding
  run `npm fund` for details

4 vulnerabilities (3 moderate, 1 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
v24.10.0 ~/Library/CloudStorage/OneDrive-Personal/Máster/Herramientas HTML y CSS 2/P3/uoc-boilerplate git:(main)  (6.057s)
npm install -D @tailwindcss/postcss

added 91 packages, changed 2 packages, and audited 463 packages in 6s

262 packages are looking for funding
  run `npm fund` for details

4 vulnerabilities (3 moderate, 1 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

1.2. Definición del tema

Se trasladaron las variables de diseño de la PEC 2 al sistema de Tailwind. Se definieron los colores principales utilizando el espacio de color moderno OKLCH:

- --color-primary: Color tierra/naranja principal.
- --color-secondary: Tono complementario.
- --color-accent: Verde para elementos destacados.
- --color-fondo: Color de fondo general de la página.

Esto permite utilizar clases como bg-primary, text-accent o border-secondary directamente en el HTML.

2. Desarrollo

2.1. Estructura

Se ha mantenido una estructura organizada separando las vistas parciales de las páginas principales:

- src/index.html: Página de navegación principal (Landing).
- src/ponentes.html: Recreación de la página de ponentes de la PEC 2.
- src/extra.html: Segunda página recreada (ej. Recetas/Detalle).
- src/ia.html: Página generada mediante Inteligencia Artificial.
- src/views/: Carpeta que contiene los componentes reutilizables (header.html, footer.html, etc.).

2.2. Componentes

Para mejorar la mantenibilidad del proyecto, se han extraído tres bloques funcionales independientes utilizando el plugin posthtml-include. Estos componentes se almacenan en la carpeta src/views/ y se pueden instanciar en cualquier página mediante la etiqueta <include>.
Los componentes seleccionados para esta abstracción son:

1. Navegación de Migas de Pan (breadcrumbs.html) Se ha modularizado la barra de navegación secundaria para mantener la consistencia en la jerarquía del sitio.

- **Justificación:** Permite situar al usuario en la estructura de navegación sin duplicar el bloque <nav> en cada archivo HTML.

- **Implementación:** Utiliza una lista ordenada <ol> con clases de utilidad para la alineación (flex, items-center) y colores del tema (text-secondary). Se ha implementado siguiendo las buenas prácticas de accesibilidad con aria-label="breadcrumb".

2. Sección de Preguntas Frecuentes (faq.html) Se extrajo el bloque de FAQs para limpiar el código de las páginas principales, ya que es una sección extensa en contenido de texto.

- **Justificación:** Al ser un bloque de texto denso, separarlo facilita la lectura del archivo principal.

- **Técnica destacada:** Se ha construido utilizando exclusivamente HTML5 semántico (<details> y <summary>) y clases de Tailwind (group-open:rotate-180, transition) para lograr el efecto de acordeón interactivo sin necesidad de escribir ni una sola línea de JavaScript.

3. Bloque de Suscripción (newsletter.html) El formulario de captación de suscriptores ("Newsletter") es un elemento transversal que suele aparecer en múltiples puntos (al pie de artículos, en la home, etc.).

- **Justificación:** Al ser un "Call to Action" (CTA) recurrente, su extracción permite modificar el diseño o los campos del formulario en un único archivo y que los cambios se reflejen globalmente.

- **Diseño:** Implementa un diseño responsive que pasa de columna a fila en pantallas grandes (flex-col sm:flex-row) y utiliza los colores corporativos (bg-primary para el botón) para destacar la acción.

2.3. Extracción de clases

Aunque la filosofía de Tailwind CSS fomenta el uso de clases de utilidad directamente en el HTML, el enunciado requiere la extracción de patrones repetitivos para limpiar el código. Se ha utilizado la directiva @apply dentro de la capa @layer components en el archivo src/assets/styles/main.css para crear abstracciones de componentes complejos.

Se han extraído, entre otras, las siguientes tres clases clave:

1. Tarjeta de Ponente (.speaker-card) Se encapsuló toda la lógica visual de las tarjetas de presentación en una única clase para evitar repetir una cadena de más de 10 utilidades en cada elemento de la lista.

- **Justificación:** Esta clase agrupa el diseño de caja (padding, bordes, sombras), la disposición flexbox (centrado vertical) y los micro-interacciones (animación hover y elevación), simplificando drásticamente el HTML de la página ponentes.html.

2. Botón de Enlace (.gallery-link) Se creó una clase para transformar enlaces estándar <a> en botones visuales coherentes.

- **Justificación:** Garantiza que todos los botones de llamada a la acción ("Call to Action") tengan el mismo tamaño, color (bg-fondo) y comportamiento al pasar

el ratón, sin necesidad de definir estas propiedades manualmente en cada botón.

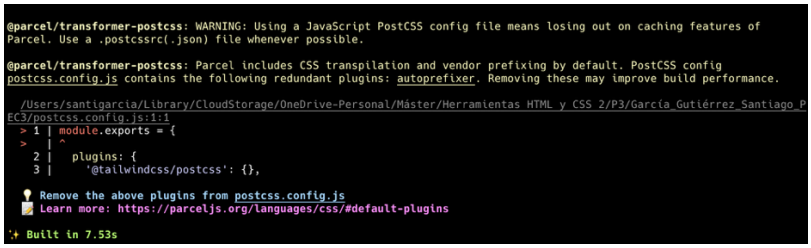
3. Tarjeta de Galería (.gallery-card) Similar a la tarjeta de ponentes, pero con especificaciones distintas para la página de contenido extra/recetas.

- **Justificación:** Permite mantener una consistencia visual en el sitio (usando las mismas sombras y transiciones que los ponentes) pero ajustando el *padding* y la estructura interna para adaptarse a contenidos más pequeños.

Beneficio obtenido: Gracias a esta extracción, el código HTML resultante es mucho más semántico y legible. En lugar de tener etiquetas con largas cadenas de clases, se utilizan identificadores claros como class="speaker-card", facilitando el mantenimiento futuro del diseño.

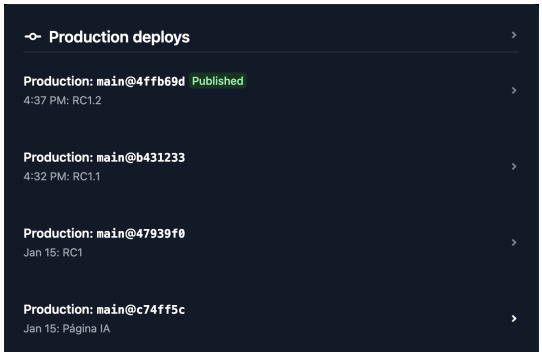
2.4. Compilación de producción y despliegue en Netlify

Para preparar el proyecto para su despliegue, se ejecutó el comando de compilación definido en el package.json, npm run build. Como resultado, se generó la carpeta /dist conteniendo los archivos estáticos minificados y optimizados, listos para ser subidos a Netlify. El proceso finalizó sin errores, como se muestra en la siguiente captura:



Una vez verificado el correcto funcionamiento de la compilación local, se procedió al despliegue continuo (CI/CD) vinculando el repositorio de GitHub con la plataforma Netlify.

Netlify ejecutó el script de compilación en sus servidores sin incidencias, publicando la versión de producción del sitio web. El resultado final es accesible públicamente y es totalmente funcional, incluyendo la navegación y los recursos estáticos.



3. Preguntas específicas sobre utility-first CSS

¿Qué diferencias hay entre el enfoque de tipo CSS semántico (el que usaste en las otras PEC) y el CSS de utilidades? ¿Cómo afectó esto a tu proceso de desarrollo? ¿Y a tu código?

En las PECs anteriores, utilizando un enfoque semántico (BEM o clases tradicionales), la metodología consistía en asignar nombres de clase basados en el significado del contenido (como .menu-principal, .tarjeta-noticia) y luego escribir todas las reglas de estilo en los archivos correspondientes. Esto separaba estrictamente la estructura del estilo. En cambio, con el enfoque Utility-First de Tailwind, he invertido este proceso. En lugar de crear nombres abstractos, aplico clases predefinidas que describen directamente la apariencia (ej: flex, p-4, text-center) directamente en el HTML.

La principal diferencia técnica es que en el enfoque semántico el HTML es "limpio" y el CSS es complejo –a pesar del uso de BEM– y específico; mientras que en Tailwind el HTML es más "verboso" (lleno de clases), pero el CSS generado es mínimo y genérico, evitando el problema de que la hoja u hojas de estilos crezcan infinitamente.

¿Qué diferencias encontraste entre usar una librería de componentes y una librería de utilidades?

Bootstrap (PEC 2): Me proporcionaba componentes ya construidos y opinionados (un botón ya tenía color, padding y hover definidos). Fue muy rápido para empezar, pero difícil de personalizar cuando quería un diseño que se saliera de lo estándar; tenía que pelear contra los estilos por defecto usando !important o selectores muy específicos.

Tailwind (PEC 3): No me da componentes, sino "ladrillos" (utilidades) para construirlos. Esto me ha obligado a construir mis propios botones y tarjetas desde cero, lo que lleva un poco más de tiempo inicial, pero el resultado es un diseño 100% personalizado y fiel al prototipo de Figma sin tener que sobrescribir estilos de terceros. Siento que tengo el control total del diseño final.

¿Qué clases y componentes decidiste extraer y por qué?

Respecto a la decisión de qué clases y componentes extraer (pregunta específica del enunciado), el detalle técnico y la justificación de cada uno se encuentran desarrollados en los apartados 2.2. Componentes y 2.3.

Analiza el código generado por la IA y proporciona toda la secuencia de prompts que has utilizado hasta llegar a la solución. ¿Ha cometido la herramienta errores? Cuáles son y cuál es la dificultad de corregir el código en contraposición a escribirlo de cero.

Conversación: https://gemini.google.com/share/2ba78868acfb

El prompt original fue:

Actúa como un desarrollador experto en Tailwind CSS. Necesito crear una página web completa llamada 'ia.html' para un sitio de conferencias gastronómicas llamado 'Tradikus'. Es el que puedes encontrar en el repositorio de Github. Así tienes contexto.

Requisitos Técnicos:
Usa HTML5 semántico y Tailwind CSS (versión CDN o clases estándar).

No uses CSS personalizado (<style>), solo clases de utilidad de Tailwind.

El diseño debe ser limpio, moderno y responsive.

Estructura de la página:
Header: (Déjalo comentado indicando).

Hero Section: Un título grande centrado que diga "Inteligencia Artificial en la Cocina" y un subtítulo "Cómo la tecnología está revolucionando la gastronomía tradicional". Fondo de color suave.

Grid de Contenidos: Una sección con 3 tarjetas (cards) en columna para móvil y en fila para escritorio (grid-cols-3). Cada tarjeta debe tener:

Un espacio para imagen (placeholder).

Un título (ej: "Recetas Generativas", "Optimización de Residuos", "Maridaje Algorítmico").

Un párrafo corto de texto.

Formulario de Contacto: Una sección sencilla con campos para Nombre, Email y Mensaje, con un botón estilizado.

Footer: (Déjalo comentado indicando).

Estilos:
Usa una paleta de colores cálida (naranjas, cremas) similar a un sitio de comida.
Usa tipografía sans-serif.
Añade sombras suaves (shadow-lg) a las tarjetas.

Por favor, dame solo el código HTML dentro del <body> (sin head ni scripts, ya los tengo).

Al principio, me temo que Gemini ha alucinado, mezclando las páginas del repositorio con las del diseño de Figma, de ahí que añadiera elementos y colores que no estaban

con éste. Quizá esta sea la razón por la que no ha entendido que las tarjetas o cards formaban parte de una lista no numerada, escribiendo código poco semántico.

Luego he ido dándole instrucciones de partes concretas del diseño. A medida que el rango de mejora disminuía, era más tentador y eficiente modificar el código por mi propia cuenta que pedirle que lo hiciera, más si consideramos que piensa durante algunos segundos. Esto es lo que ha sucedido, por ejemplo, con el texto uppercase.

¿Cuál es tu opinión sobre el uso de estas herramientas?

Tras realizar esta práctica, mi visión sobre herramientas como ChatGPT/Claude para el desarrollo web es que actúan como "potenciadores de productividad", pero no como sustitutos del conocimiento técnico. Es más, sin conocimiento técnico, es más difícil obtener el resultado deseado.

Sobre los puntos fuertes: Destaco principalmente la capacidad de la IA para acelerar el "andamiaje" inicial del proyecto, permitiendo generar estructuras complejas y responsivas en cuestión de segundos que rompen el bloqueo de la hoja en blanco. Además, actúa como un eficaz "diccionario en tiempo real" para Tailwind, traduciendo descripciones en lenguaje natural a clases de utilidad específicas sin necesidad de consultar constantemente la documentación oficial.

Sobre los retos y dificultades: La principal dificultad radica en la necesidad imperiosa de supervisión técnica, ya que el código generado suele ser visualmente correcto, pero semánticamente pobre o con fallos de accesibilidad. Esto obliga a cambiar el rol de desarrollador por el de "revisor de código", donde a veces resulta más costoso auditar y corregir la lógica de la IA que haber escrito el componente desde cero.

4. Autoría de las imágenes

Las imágenes, procedentes de la PEC anterior, fueron generadas con Gemini.