Repositorio PAC3 GitHub
https://github.com/AlbertViSi/PAC3Eines
Repositorio PAC2 GitHub
https://github.com/AlbertViSi/PAC2Eines
Netlify PAC3
https://lambent-horse-e09ed3.netlify.app/
Netlify PAC2
https://pac2.netlify.app/

Preparacion entorno

Mediante el comando de git clone se realiza un clon de UOC boilerplate (https://github.com/uoc-advanced-html-css/uoc-boilerplate).
He optado por simplemente copiar el clon y eliminar el archivo .git y no realizar un segundo clon de un repositorio vacío.

Preparacion tailwind CSS
Para la instalación se ha seguido la guia que se encuentra en "https://tailwindcss.com/docs/installation/using-postcss" junto con la información dada en la actividad 2 del modulo 4.2.

No se comentará todas las configuraciones personalizadas añadidas a tailwind dado que es innecesario al realizarse para cosas como el min-with:500px, colores o crear un flex-grow personalizado.

Nota sobre tailwind: Se ha usado ChatGPT para ayudar a la hora de configurar las distintas clases de tailwind tanto para configuraciones personalizadas como para crear los estilos para emular las clases de PAC2 en Atomic CSS con tailwind para la PAC3.

Elección de páginas a replicar de la PAC2
La elección de páginas a replicar de la PAC2 han sido la de título (index.html) y la de eventos (pagethree.html), estas se han elegido por las siguientes razones:

1. Título: Es una página que contiene elementos que están sujetos a @support, además de trabajar con grids verticales con los que no he trabajado mucho. También posee un fondo de pantalla que no

ocupa toda la pantalla y también algunas reglas específicas que solo aplican a algunos elementos de esta página, elementos que en este caso corresponden al header, elemento extraído.
2. Eventos: Esta página contiene algunas animaciones por lo que será interesante ver qué desafíos lleva el adaptarlas a tailwind CSS.

Extracción clases de tailwind

@apply se ha usado para los casos siguientes:

1. Usado en clases que se encuentran dentro de @support, para evitar el tener que añadir plugins extras en tailwind.
2. Para un cambio en la regla general de a:hover para el título se ha creado una clase específica llamada index que se le ha aplicado una regla especial, se ha intentado hacer de forma manual con tailwind pero al no funcionar se ha decidido extraer la regla al final.
3. Se ha decidido extraer el body como una clase bloody-style; se ha hecho eso debido a que el body es la configuración que todas las páginas tienen, por lo que al extraerlo como una configuración aparte conseguimos limpiar el código de HTML, que de otra forma iríamos repitiendo en todas las páginas.

Extraccion elementos HTML

Se ha extraído el elemento contacto, este ya se extrajo en el documento original (PAC2). Este ha sido extraído al ser un elemento recurrente al estilo del footer y el header y repetirlo en cada fichero es innecesario.

Se ha extraído también el elemento correspondiente a la frase coletilla del título (index.html). Se ha elegido esta no por ser muy repetida, sino por ser la mejor elección al no tener que eliminar todo el código de una de las páginas, en cualquiera de las otras, de haberse escogido, prácticamente habría habido un <include> raro o una página con tres <include> seguidos siendo cabecera, la extracción y el footer. Personalmente, de no requerirse para el ejercicio, esta segunda extracción no se habría realizado al no tener mucho sentido, dado que no es un elemento recurrente que se vaya a reutilizar en otras partes de la página.

Comentario sobre adaptaciones
Debido a que la página es muy similar a la entregada en la PAC2 no se hará mucho hincapié en el desarrollo sino más bien se comentarán los cambios importantes respecto a la PAC2.

Adaptaciones relevantes hechas al elemento index.html que se han tenido que realizar
En la pantalla de index se ha debido de añadir una altura mínima dado que quedaba de tamaño muy pequeño y se ha tenido que añadir para arreglarlo visualmente.
En la portada se ha tenido que añadir “h-full w-full flex flex-col” y en el texto de titulo.html añadir “h-full w-full” debido a que no ocupaba todo el espacio disponible, al añadir esto se ha arreglado este error.
Las imagenes del logo no se alineaban correctamente y se ha debido de añadir elementos “mx-auto sm:mx-0 sm:text-left” sm siendo el mínimo de ancho de pantalla definido para el cambio de móvil a sobremesa.

Adaptaciones importantes hechas a pagethree.html
Una adaptación que se ha tenido que hacer ha sido añadir un cierto margen tanto en las tarjetas (con my-{value}) como en el título (ha sido un margen regular m-{value}), margen que no tenía en PAC2, de lo contrario los elementos estaban muy pegados entre sí.
También ha habido un problema con las animaciones dado que transition-all no funcionaba y se ha tenido que hacer mediante animación específica transition-max-height.

Notas extras
Se ha decidido el uso preferible de tailwind CSS antes que añadir style en la línea. Se ha mantenido el formato CSS del body al ser un elemento omnipresente en todas partes

Preguntas:

- **¿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?**

A mi punto de vista no hay muchas diferencias entre el enfoque de tipo CSS semántico y el enfoque de tipo CSS de utilidades, las principales serían que, en el primero, creas un nombre para la clase y luego le das las propiedades pertinentes, mientras que con el CSS de utilidades, usando tailwind dado que es el método usado en esta PAC, no hay necesidad de crear esas clases y simplemente es poner las características de los elementos en cuestión. Otra de las diferencias, y la principal a mi punto de vista, es que, en el CSS semántico sueles desarrollar la parte del código apuntando a los componentes concretos, mientras que con el CSS de utilidades, quizás en vez de apuntar tanto al componente se apunta a un rango más general.

En mi caso particular, como ya tenía marcados los márgenes se ha traducido a, simplemente, convertir las clases ya existentes a Atomic CSS, el desarrollo se ha limitado a la investigación y adaptación de las clases existentes, cuando aplicar los @apply por ejemplo, en casos como los @supports que no se pueda usar el codigo en linea de tailwind al no estar incluido en el propio tailwind; otra parte del desarrollo que me ha afectado ha sido el hecho de tener que adaptar el código con algunas funcionalidades que, tal y como estaban planteados en la PAC2, no funcionaban y se ha tenido que investigar cómo adaptar el código para mejorar la funcionalidad y emular los resultados de la PAC2.

La afectación al código ha sido sorprendente, al tener un CSS muchísimo más limpio y reducido, teniendo en el solo el código esencial; mientras que el HTML está lleno de código “extra” para definir todas esas propiedades que ya no definimos en CSS, aunque en mi caso el aumento de código en el HTML no lo noto en tanta medida como noto la limpieza del codigo CSS en sí. Otro punto que considero mencionar es la configuración que se debe hacer de ciertas clases en el .tailwind.js para ciertos colores, al haber notado éste bastante más limitado que con el CSS habitual, aunque sigue siendo muy ordenado y bastante más claro.

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

La verdad es que yo no use una librería de componentes, pero entiendo que una de las diferencias principales es que las librerías de componentes están enfocadas a componentes concretos, como pueden ser los botones por ejemplo, eso limita en cierta medida su aplicación y, supongo, que pueden ser un poco más rígidas en ese sentido.
Por otro lado, las librerías de utilidades me ha parecido que reducían la cantidad de código de una forma enorme, aunque no estaban libres de problemas, uno de ellos el hecho de no tener soporte para el @support (valga la redundancia), pero considero que la reducción de código que ha implicado supera con creces los problemas que me ha dado.

- **¿Qué clases y componentes decidiste extraer y por qué?**
  Los componentes que decidí extraer fueron el header, footer y información de contacto, esto lo hice debido a que todos ellos son componentes recurrentes que se encuentran en todas las páginas, por lo que, su extracción es una elección lógica, al reducir la cantidad de código redundante en las distintas páginas de la web.
  Otro componente que se ha extraído ha sido la frase de la pantalla principal junto con el grid en el que está esta. No ha habido ninguna razón de peso para extraerse, simplemente se ha extraído porque se pedía en el ejercicio.
  Respecto a las clases extraídas se han extraído, un grupo de ellas eran las que, en el código original de la PAC2, formaban parte de un @support, esto es debido a que @support no funciona con tailwind sin necesidad de añadir plugins extras, por lo que he decidido extraer las clases debido a que CSS si que entiende los @supports. Otra clase ha sido un color especial para el enlace del header, esta se ha extraído dado que, después de probar múltiples opciones no se ha conseguido que funcionara si no se extraía, esa clase lo que hace es dar un estilo personalizado a los enlaces de la página index.html.
  Finalmente la última clase es la extraido ha sido el body, usado en todas las páginas, por lo que ha sido más práctico extraerlo y usarlo como clase.

- **Analiza el código generado por la IA. ¿Ha cometido la herramienta errores? Cuáles son y cuál es la dificultad de corregir el código en contraposición a escribirlo de cero. ¿Cuál es tu opinión sobre el uso de estas herramientas?**
  Para la generación del código con una IA se ha escogido ChatGPT como IA y, para que generará la primera iteración del código se le ha proporcionado una imagen de la página Figma proporcionada.
  Al principio es una aproximación relativamente buena, pero faltan elementos como las imágenes de las tarjetas y otros no se parecen a la original, como el pie de página o la imagen principal que no se adecua al contenedor como lo hace con la página Figma proporcionada.

  Pie de página:

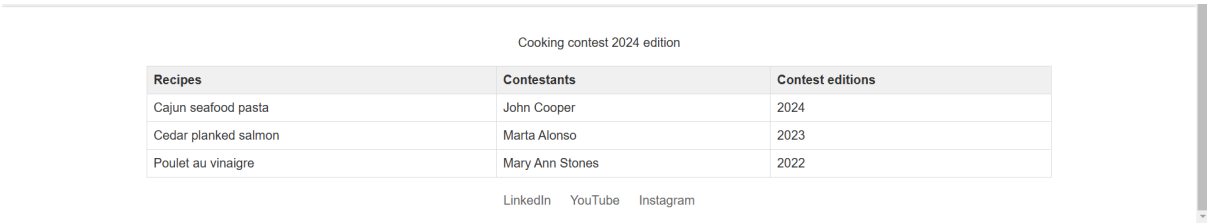
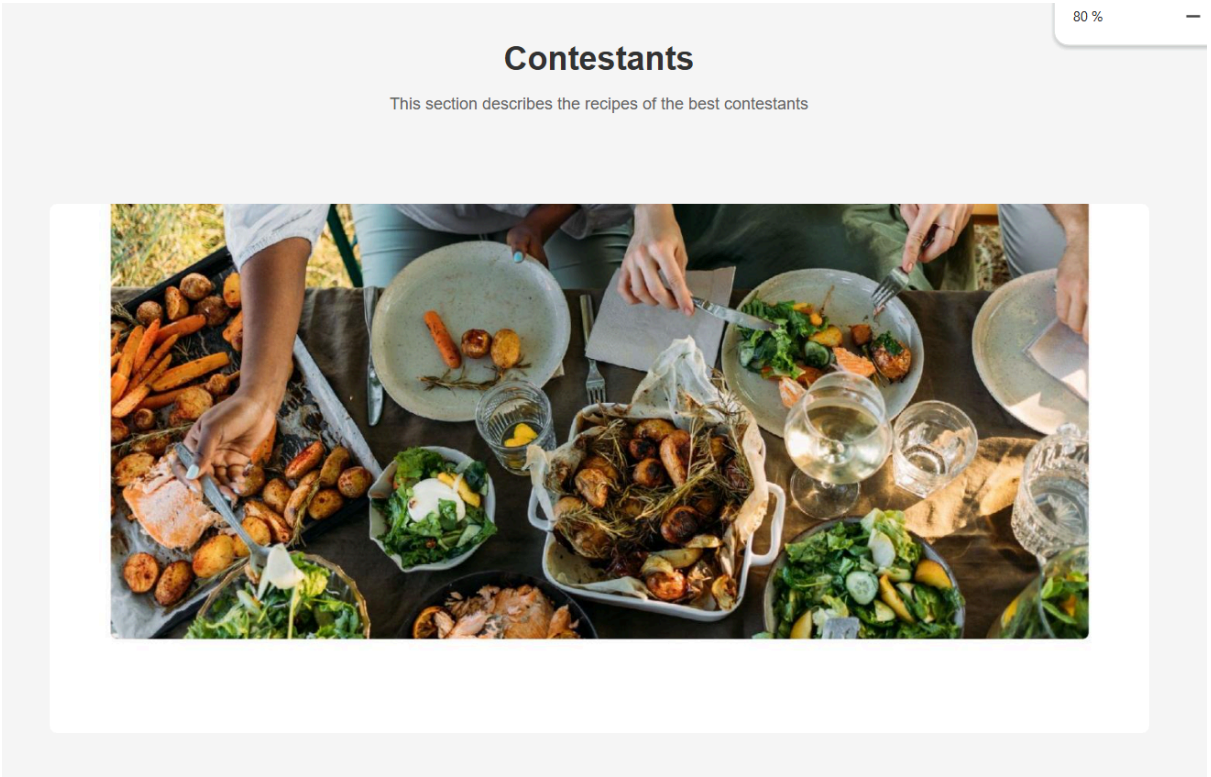


  Imagen central:



En la segunda iteración se le ha pedido que añada un placeholder para las imágenes de las tarjetas.
En la tercera iteración se le ha pedido que haga la página responsive, al no ser así en la primera iteración que nos ha proporcionado.
La cuarta iteración se le ha pedido que modifique el pie de página como grid o flex según considere que sea mejor, al final usó grid.
A continuación se le ha pedido convertir las tarjetas del final de la página en links que redirigirían a la receta de la tarjeta.
Finalmente le he pedido que arreglara el encabezado dado que en móvil los elementos se solapan.
Ya no le he dado más indicaciones dado que era muy cercano al resultado final, no he conseguido que la imagen central se adaptara al contenedor dando órdenes a la IA por lo que así se queda.

Como conclusión creo que la IA es una muy buena herramienta para ayudar, entre otras cosas, al desarrollo del código, aunque esta puede cometer errores, la mayoría de veces por una mala explicación de lo que se necesita exactamente, o simplemente por trabajar en un entorno no ideal.

También está el problema de la corrección, si se le pide a una IA escribir todo el código tienes menos control sobre el mismo, al igual que intentar entender código escrito por un tercero puede ser caótico hasta cierto punto, también lo es en el caso que te lo escriba una IA y, para finalizar, reiterar que la IA es una muy buena herramienta para ayudar en el desarrollo de las páginas web entre otras funcionalidades.