https://doi.org/10.22201/dgtic.ctud.2024.2.3.63

Vol. 2, Núm. 3. julio-septiembre 2024, págs. 63 - 75

Criterios de implementación de Tailwind CSS en desarrollos frontend

Edgar Vargas Zermeño

Dirección General de Cómputo y de Tecnologías de Información y Comunicación

Universidad Nacional Autónoma de México

talos@unam.mx

ORCID: 0009-0006-4772-653X

Información del reporte:

Para citar este reporte técnico:

Vargas Zermeño, E. (2024) Criterios de implementación de Tailwind CSS en desarrollos frontend. Cuadernos Técnicos Universitarios de la DGTIC, *2* (3) páginas (63 - 75).

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63

Resumen

El manejo particularizado de los estilos que la Separación de Intereses (Separation of Concerns, SoC) trajo consigo, puede volverse demasiado complicado en proyectos para desarrollo de software a gran escala que son intervenidos por muchas personas. Bajo esa circunstancia surgió la necesidad de mantener un trabajo colaborativo organizado en tres proyectos a cargo de la DGTIC. La incorporación de un *framework CSS* para construcción *frontend* basado en clases de utilidad resolvió parcialmente el problema, porque permitió afrontar nuevos retos para el equipo de desarrollo. Se implementó una nueva forma de trabajo respecto de los estilos, con designación de criterios para configurar y utilizar reglas y clases.

Palabras clave:

CSS, frontend, Tailwind CSS, interfaz gráfica de usuario

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 64 - 75

1. INTRODUCCIÓN

Después de analizar las metodologías para escritura de estilos BEM (*Block Element Modifier*), SMACSS (*Scalable and Modular Architecture for CSS*), OOCSS (*Separating container and content with CSS "objects"*), ITCSS (*Inverted Triangle architecture for CSS*) y SUITCSS (*Structured class names and meaningful hyphens*), se valoraron sus ventajas y desventajas (Bastidas, 2020), y la facilidad de entendimiento y adaptación que proveen para el equipo de trabajo, si se considera el manejo específico de los estilos en la Separación de Intereses (*Separation of Concerns*, SoC, que se refiere a la diferenciación en el desarrollo de software entre el manejo de datos, la lógica, y la forma de presentar y manipular los datos en la interfaz de usuario). Se encontró que no son suficientes para resolver el crecimiento y la complejidad de la escritura de reglas de estilo, el aumento de peso, disminuir la cantidad de archivos de estilo, ni solventar la sobreescritura[1] de reglas para especificidad (el criterio que determina qué estilos aplicará el navegador a un elemento afectado por varias reglas de estilo que se contradicen o se sobreescriben).

Para tener entornos con mejor desempeño en despliegue de interfaz de usuario y código CSS[2] (*Cascading Style Sheets*), mantenible para tres proyectos de desarrollo de software a la medida a cargo de la DGTIC en 2023, se buscaron nuevos enfoques y se encontró un nuevo paradigma de escritura de estilos que prioriza las clases de utilidad reutilizables sobre la escritura exagerada de reglas y archivos CSS: *Utility First*, que implica priorizar la utilidad y responde al desafío de mantener las hojas de estilo sin saturación de reglas. En vez de usar estilos personalizados para cada elemento, se emplean clases predefinidas que son reutilizables. Este cambio de paradigma plantea un proceso de desarrollo mucho más fácil y ágil (Bhayani, 2023).

Una clase de utilidad es una regla de estilo que designa la afectación de una sola propiedad. Por ello, para entender una clase de utilidad, hay que considerar que un selector[3] de clase aplica en la apariencia lo que su nombre indica: se declaran sobre el HTML los estilos como se hacía anteriormente, pero no en múltiples atributos[4], sino bajo el procedimiento estandarizado del atributo *class*, que es un atributo que lista las reglas de estilo por aplicar en un elemento HTML. En este contexto, se eligió de un conjunto de *frameworks*[5] *Utility First* existentes (Amaechi, 2021) a Tailwind CSS por su popularidad, documentación clara y peso bajo en archivos compilados.

El objetivo del trabajo presentado en este reporte es incorporar la apariencia visual en desarrollos *frontend* (es decir, la parte visible de un software o desarrollo web que dispone los elementos de interacción con los usuarios), de manera ágil e intuitiva, para lograr construcciones responsivas consistentes y mantenibles, desde una metodología replicable, que tome el enfoque *Utility First* implementado con Tailwind CSS.

1 Efecto de una regla CSS que anula o modifica la declaración previa de apariencia de una propiedad o elemento en una hoja de estilo.

2 Hojas de Estilo en Cascada o CSS (del inglés Cascading Style Sheets), es un lenguaje de estilos que describe cómo deben representarse los elementos HTML en la pantalla, en una impresión u otros medios.

3 Un selector es una palabra mediante la cual se agrupan las reglas de estilo que se aplicarán a uno o varios elementos.

4 Un atributo es una palabra reservada que se coloca en las etiquetas de apertura HTML para asignar determinados valores y comportamientos a los elementos.

5 Un framework es un conjunto de herramientas, disposiciones y estructuras que se utilizan para desarrollar un proyecto de desarrollo de software, a partir de elementos predefinidos.

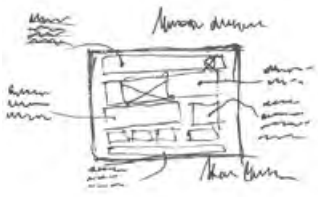
https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 65 - 75

2. DESARROLLO TÉCNICO

La documentación de Tailwind CSS aborda procedimientos técnicos, pero no presenta una metodología generalizada que proponga criterios para la integración del *framework* a los proyectos. Todo proceso de desarrollo pasa por varias etapas, el diseño de la interfaz tiene sus propias fases, donde una vez entendidos los requerimientos se bocetan ideas (Figura 1) que se transformarán en esquemas del contenido representado en pantallas (Altamira Team, 2021).

Figura 1

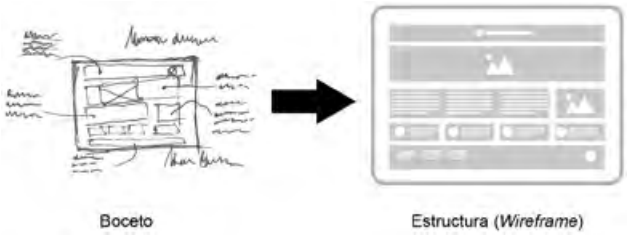
Boceto y apuntes



Fase de estructuración: Comprender los requerimientos y partir de prototipos de baja fidelidad es esencial. Un *wireframe* (representación visual básica de la estructura y funcionalidad de una página web o pantalla de aplicación móvil) es excelente para hacerlo (Figura 2), ya que mediante formas, líneas y marcadores de posición, se representan los elementos de la interfaz, priorizando la estructura y la funcionalidad sobre la estética visual (Joyce, 2023).

Figura 2

Del boceto al wireframe



Fase de maquetación: A partir de los *wireframes* se concreta el aspecto visual (Figura 3) para definir la interfaz gráfica de usuario (Joyce 2023).

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 66 - 75

Figura 3

Del wireframe al diseño visual

Fase de prototipado: Se obtiene un producto digital para probarlo desde la perspectiva del usuario y detectar con ello cualquier problema o frustración que puedan experimentar (Joyce, 2023). Con el diseño gráfico aprobado, se puede "pensar Tailwind", es decir, imaginar los elementos estructurales, el espaciado, los componentes, los casos especiales y excepcionales, partiendo del diseño traducido a clases de Tailwind CSS.

Las clases de utilidad implican llevar a cabo nuevas prácticas. Es necesario tener un conjunto de criterios de implementación para integrar el estilo al sistema y determinar bajo qué reglas configurar los estilos CSS.

2.1 METODOLOGÍA

Una práctica favorable para un equipo de desarrollo es comenzar a construir estilos bajo pautas uniformes y concisas que todos los colaboradores conozcan. Previo a abordar los criterios a usar con Tailwind CSS, se revisan los espacios y los elementos de los que se dispone para convertir las ideas de interfaz gráfica en prototipos que se convertirán en las vistas del sistema en producción.

2.1.1 ÁMBITOS DE PROTOTIPADO

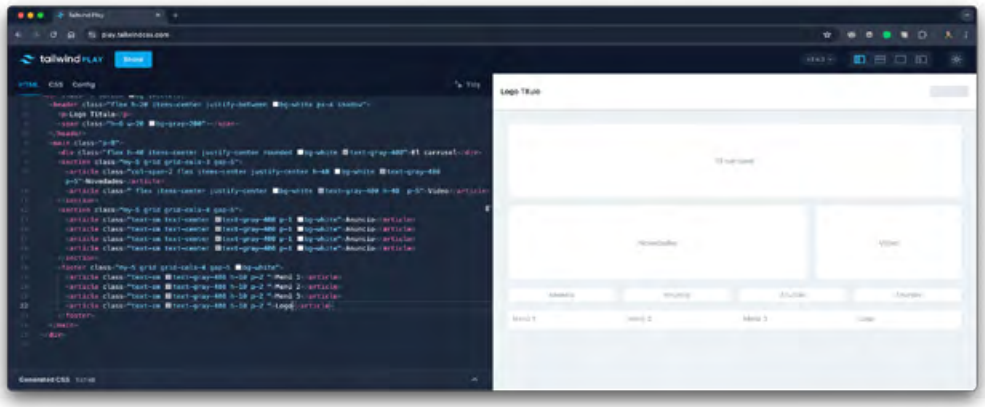
Se cuenta con tres ámbitos de prototipado para conformar la maquetación con Tailwind CSS:

- **Playground:** Espacio público gratuito para ensayar la composición, apariencia y comportamientos (figura 4). Permite guardar y mostrar en línea lo maquetado.

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 67 - 75

Figura 4

Ejemplo del Playground



Nota. Disponible en: https://play.tailwindcss.com/

- **CDN:** Este ámbito requiere implementar un desarrollo local que puede replicarse a un servidor para obtener la opinión de los colaboradores (figura 5).

Figura 5

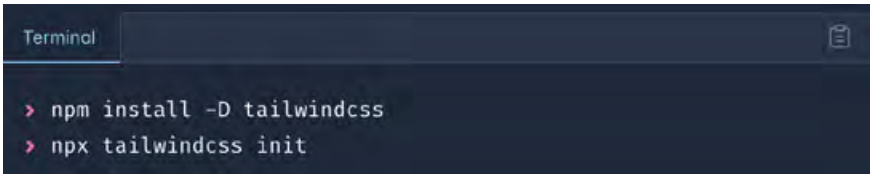
Ejemplo de inserción de CDN de Tailwind CSS

Nota. Disponible en: https://tailwindcss.com/docs/installation/play-cdn

- **CLI:** Este caso necesita compilación, es el ámbito más parecido a un desarrollo (figura 6) con archivos específicos semejantes a los de un sistema en producción.

Figura 6

Comandos para generar un proyecto



Nota. Disponible en: https://tailwindcss.com/docs/installation

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 68 - 75

2.1.2 CRITERIOS DE ORDEN PARA LA CONFIGURACIÓN

Existen tres elementos de configuración en los ámbitos de prototipado (figura 7):

a) **Archivo de configuración JS:** En él se indican parámetros, comportamientos y apariencias no consideradas por defecto.

b) **Reglas en atributo class:** Son todas las reglas implementadas con atributo *class* sobre las etiquetas HTML.

c) **Archivo fuente CSS :** Contiene las directivas de Tailwind CSS y será compilado con los anteriores.

Figura 7

Elementos de configuración en un desarrollo

El criterio de orden del listado anterior obedece a que:

- **Primero:** se dispone de colores, fuentes y comportamientos en configuraciones no predefinidas por Tailwind CSS (archivo de configuración JS).
- **Segundo:** se agregan las clases predefinidas por Tailwind CSS a las etiquetas HTML (reglas en atributo *class*).
- **Tercero:** en el archivo de estilo (archivo fuente CSS) se conjuntan clases o se escriben nuevas reglas no predefinidas por Tailwind CSS.

2.1.3 CRITERIOS DE IMPLEMENTACIÓN DE LA CONFIGURACIÓN

Tailwind CSS permite reconfigurar su tema gráfico sobre más de 40 propiedades en el archivo tailwind.config.js (https://github.com/tailwindlabs/tailwindcss/blob/master/stubs/config.full.js). El criterio por emplear es que, se agreguen o no parámetros nuevos, se debe considerar a profundidad los siguientes tres aspectos:

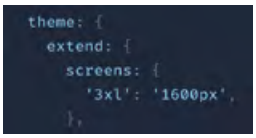
2.1.3.1 PUNTOS DE RUPTURA (BREAKPOINTS)

Tailwind CSS sólo considera 5 *breakpoints*, que pueden extenderse con prefijos no considerados (figura 8).

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 69 - 75

Figura 8

Ejemplo de punto de ruptura agregado



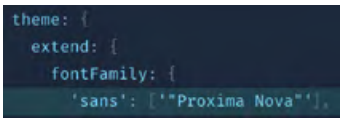
Nota. Documentación en: https://tailwindcss.com/docs/screens

2.1.3.2 FUENTES TIPOGRÁFICAS

De ser necesario, las familias de fuentes disponibles pueden extenderse (figura 9).

Figura 9

Ejemplo de fuente agregada



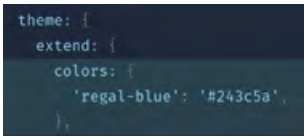
Nota. Documentación en: https://tailwindcss.com/docs/font-family#using-custom-values

2.1.3.3 GAMA CROMÁTICA

Es pertinente integrar tantos matices como sea necesario a los colores predeterminados (figura 10).

Figura 10

Ejemplo de color agregado



Nota. Documentación en: https://tailwindcss.com/docs/customizing-colors#adding-additional-colors

2.1.4 CRITERIOS DE IMPLEMENTACIÓN PARA EL USO DE REGLAS EN ATRIBUTO CLASS

Para abordar los criterios en atributo *class*, se dividen los elementos en dos categorías: esenciales y accesorios. Los elementos esenciales pueden o no ser declarados, pero están de forma implícita en su valor por defecto; los elementos accesorios pueden omitirse y no están de forma implícita.

Observar con disciplina el orden propuesto en la figura 11, ayudará a ubicar dónde hacer modificaciones al realizar actividades de mantenimiento y corrección. El equipo de trabajo se acostumbrará a ubicar con prontitud cada clase, los prefijos responsivos (palabras que anteceden a la declaración de una clase Tailwind CSS con afectación en un tamaño de pantalla específico), así como las *pseudoclases* (palabras

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 70 - 75

clave que se añaden a los selectores CSS y que especifican el estado o comportamiento de un elemento). Lo anterior se muestra en la figura 12.

2.1.4.1 ELEMENTOS ESENCIALES

1. **Posición:** Define la posición del elemento.
2. **Dimensiones:** Indica alto y ancho y, de ser necesario, profundidad.
3. **Modelo de *display:*** Indica qué modelo de *display* se usará y, en consecuencia, sus clases.
4. **Fondo y borde:** Designa el aspecto cromático que tomarán los contenedores.
5. **Características de texto:** Aborda las características del texto: puntaje, interlineado, alineación y color.
6. **Espaciado:** Al desplegar contenedores y textos es oportuno agregar márgenes para balancear los espacios del conjunto.

Figura 11

Orden de ubicación de elementos esenciales

2.1.4.2 ELEMENTOS ACCESORIOS

- **Pseudoclases:** Se pueden intercalar las pseudoclases de manera posterior inmediata al elemento que modifican, para no romper con el orden establecido.
- **Prefijos responsivos:** También se deben intercalar los prefijos que adaptan los elementos al ancho de pantalla, después de la propiedad que condicionan, respetando el orden ascendente *Mobile First* (concepto que propone comenzar un desarrollo considerando primero el dispositivo móvil de menor tamaño para ajustar después a pantallas de mayor tamaño).
- **Redondeado y sombras:** Estas propiedades inciden en efectos estéticos y no son indispensables.
- **Transición y animación:** Definitivamente al final de toda regla,son las declaraciones que permiten indicar apariencia mediante palabras (selectores) que representan un grupo de propiedades para ser aplicadas en un elemento HTML, se procede a insertar las clases que disponen la animación de elementos.

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 71 - 75

Figura 12

Ubicación de elementos accesorios

2.1.5 CRITERIOS DE IMPLEMENTACIÓN PARA LA INCORPORACIÓN DE NUEVAS REGLAS

Las metodologías de escritura para CSS brindan una idea de organización, pero no funcionan para aplicarse de forma directa al usar Tailwind CSS, puesto que este *framework* pretende que no se escriban más reglas de estilo; en cambio, alienta a utilizar las directivas para incorporar nuevas declaraciones en forma de reglas personalizadas que serán compiladas en un archivo CSS optimizado (figura 13).

2.1.5.1 CLASES CON DIRECTIVA @APPLY

Si se considera que una capa o layer (CSS) es una declaración que agrupa código CSS en capas para organizar con más facilidad las reglas de estilo, es importante respetar el orden en capas del que se dispone en las directivas de Tailwind CSS: primero *Base*, luego *Components*, y al final *Utilities*, para colocar nuestras propias clases de utilidad sobrescribiendo la apariencia base y de componentes. Dado lo anterior, se procede a asignar clases con la directiva *@apply* para los casos más comunes y repetitivos, como son los títulos, párrafos, tablas, formularios, botones, vínculos, entre otros.

2.1.5.2 CLASES CONVENCIONALES

En algunos proyectos no basta con usar Tailwind CSS; si se integra algún componente que no prevé su utilización, se le integra con poco esfuerzo al reinterpretar las reglas de estilo propuestas por su autor a reglas Tailwind CSS.

Figura 13

Reinterpretación a clases de utilidad

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 72 - 75

En lo general, puede aparecer la "tentación" de generar nuestras propias reglas contradiciendo el espíritu de Tailwind CSS de no escribir ya más código. Siempre es posible confiar en que la solución se encontrará en la documentación del *framework* y otros espacios.

2.1.6 FASE DE INTEGRACIÓN DEL DISEÑO AL SISTEMA DE INFORMACIÓN

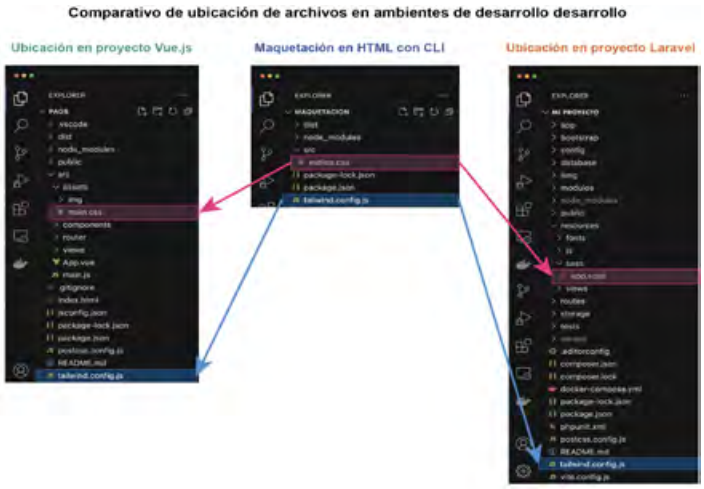
En las secciones anteriores se definió la estrategia de conformación de los estilos. Aunque el prototipo es la referencia para trasladar la apariencia en el ambiente de desarrollo del sistema, usualmente surgirán cambios no previstos. Aunque se pueden maquetar soluciones en el curso del desarrollo, tener un ambiente de prototipado brinda la opción de lograr un vaivén entre el desarrollo y el prototipado, pues en todo proyecto surgen nuevos requerimientos no previstos u omitidos que pueden ser ensayados antes de su integración.

2.1.7 UBICACIÓN DE ARCHIVOS EN DESARROLLO

Es la parte crucial que conecta la maquetación con el desarrollo (figura 14). La traslación de las reglas, configuraciones y directivas no debiera ser compleja, sin embargo, hay desarrollos que por su naturaleza presentan retos particulares que requieren trabajo que va más alla de la tarea de copiar y pegar a partir de los archivos de la maquetación.

Figura 14

Comparativo de ubicación de archivos en ambientes de desarrollo



Al igual que en el punto 2.1.2, se ubican los tres elementos de configuración:

- **Archivo de configuración JS:** Se trasladan los parámetros, comportamientos y apariencias no consideradas por defecto.
- **Reglas en atributo class:** Se transfieren las clases maquetadas a los archivos de la vista o componente.
- **Archivo fuente CSS:** Se colocan las reglas que se crearon al maquetar y se ajustan los detalles de apariencia que por la naturaleza del entorno hayan variado.

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 73 - 75

2.1.8 CRITERIO DE IMPLEMENTACIÓN PARA COMPONENTES WEB

Los componentes no aparecen en los ámbitos de prototipado propuestos, pues no son ambientes formales de desarrollo, sino entornos para disponer la apariencia, composición y espaciado. Si el proyecto los incluye, el criterio a usar es simple: se han de sustraer las clases respectivas para colocarlas en el archivo del componente, o en su caso indicar la clase con la directiva @apply correspondiente, como se muestra en el punto 2.1.5.2.

2.2 RESULTADOS

Implementar clases de utilidad con Tailwind CSS en tres proyectos de desarrollo de software en 2023 tuvo varias ventajas ya que aumentó progresivamente la sinergia y entendimiento en el equipo de desarrollo al permitir con más facilidad que cada integrante comprendiera los estilos empleados y los replicara.

También redujo notoriamente la incertidumbre sobre el desempeño en pantalla al contar con un procedimiento ensayado para implementar el diseño gráfico en nuevos proyectos, confiando en la herramienta empleada y en la metodología.

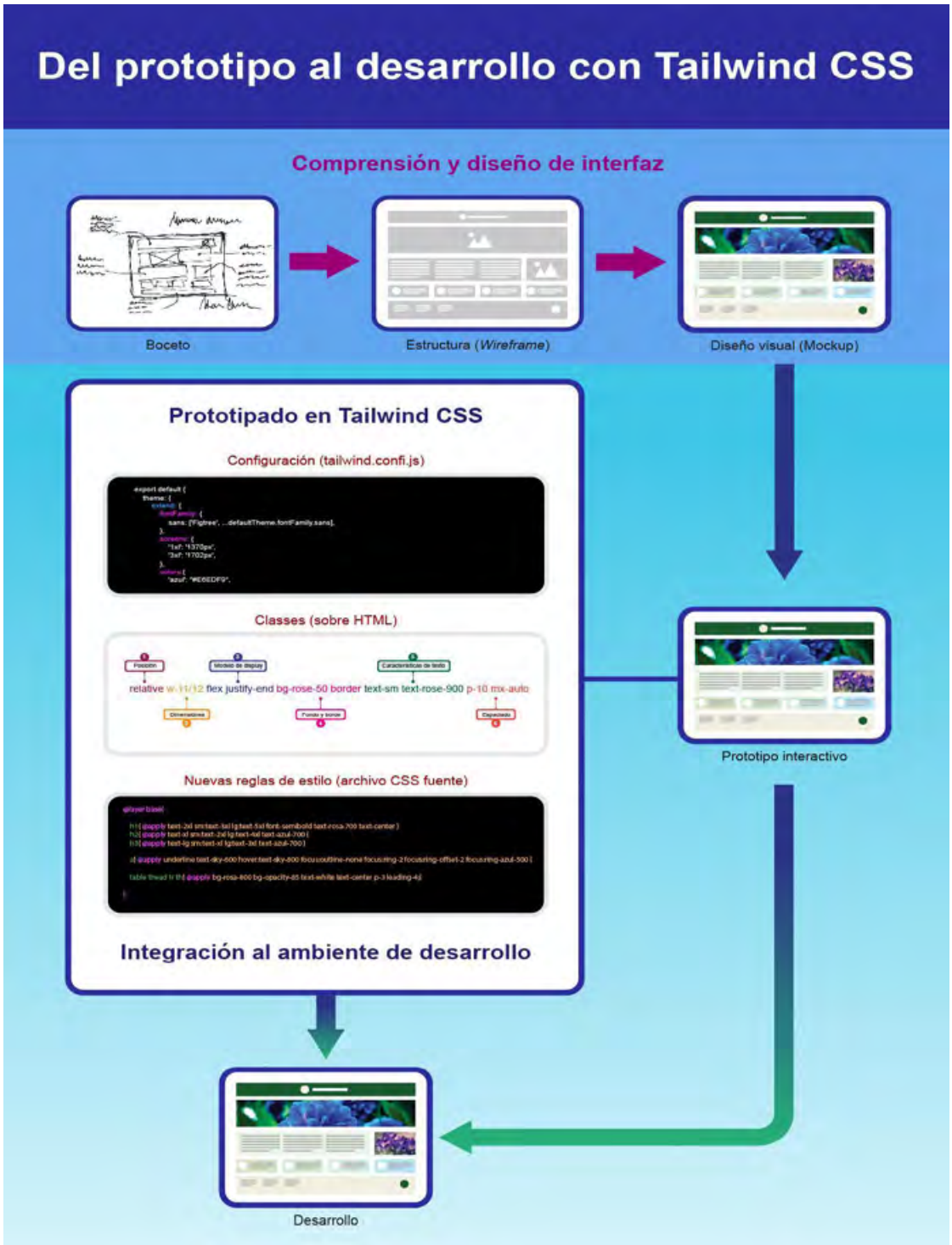
Se obtuvo un mejor manejo de fuentes y gama cromática con incorporación de matices no considerados por Tailwind CSS bajo nombres de clase personalizados que se integran de forma natural al desarrollo. Así mismo, en pocas líneas de código CSS se centraliza la apariencia de encabezados, tablas y formularios.

En la figura 15 se muestra un resumen del proceso de prototipado e integración al ambiente de desarrollo con Tailwind CSS.

https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 74 - 75

Figura 15

Prototipo y desarrollo con Tailwind CSS



https://doi.org/10.22201/dgtic.ctud.2024.2.3.63 Vol. 2, Núm. 3. julio-septiembre 2024, págs. 75 - 75

Con base en la experiencia, se estima que cualquier persona con al menos un año de experiencia en CSS, HTML y JavaScript puede superar la curva de aprendizaje, generar un prototipo y hacer configuraciones para Tailwind CSS.

Finalmente, se corrigieron las incidencias reportadas en pruebas de despliegue para la corrección de errores de forma ágil al tener ubicadas las configuraciones y las reglas de estilo.

CONCLUSIONES

Al implementar estilos CSS con un *framework Utility First* se encontró que no hay una sola manera de escribir las clases; en cambio, se concentró el esfuerzo en obtener el mejor provecho para la naturaleza de cada proyecto y conformar una estructura entendible y mantenible para todo el equipo de desarrollo, al agregar una metodología ordenada y repetible.

Trabajar con un *framework* que se considera *Utility First* no es limitativo. Se agiliza el arranque, desarrollo y conclusión en lo que a un *frontend* se refiere, y se tiene la oportunidad de escribir reglas nuevas.

El riesgo de practicar en menor grado la escritura convencional de reglas de estilo es real, para contrarrestarlo se recomienda buscar la experimentación del uso actualizado de los módulos de CSS para seguir actualizados. No depender de una sola tecnología o enfoque es lo más pertinente.

REFERENCIAS

Amaechi, Amarachi (2021). Top utility-first CSS frameworks. LogRocket . Recuperado el 12 de mayo de 2024 de: https://blog.logrocket.com/top-utility-first-css-frameworks/

Altamira Team (2021). Wireframe vs mockup vs prototype – What are the differences?. Altamira. Recuperado el 13 de mayo de 2024 de: https://www.altamira.ai/blog/how-do-wireframe-mockup-and-prototype-differ/

Bastidas, William (2020). Metodologías o Arquitecturas CSS (OOCSS, BEM, SMACSS, ITCSS, Atomic Design). Medium. Recuperado el 03 de mayo de 2024 de: https://medium.com/williambastidasblog/metodolog%C3%ADas-o-arquitecturas-css-oocss-bem-smacss-itcss-atomic-design-a1a3cfbfa6c9

Bhayani, Parul (2023). Embracing Utility-First CSS: A Shift in Web Development Paradigm. DhiWise. Recuperado el 4 de mayo de 2024 de: https://www.dhiwise.com/post/embracing-utility-first-css-a-shift-in-web-development

Joyce, Lisette (2023). The concept of Wireframes, Mockups, and Prototypes. Medium Bootcamp. Recuperado el 12 de mayo de 2024 de: https://bootcamp.uxdesign.cc/the-concept-of-wireframes-mockups-and-prototypes-5c73ba22ecd0