**Estilos avanzados con css**

**Introducción**

🙌 Tras entender el valioso trabajo de los diseñadores UX/UI, es momento de adentrarnos aún más en nuestra herramienta principal: **react**. En esta clase, repasaremos lo aprendido sobre estilos y exploraremos enfoques avanzados, así como la aplicación de bibliotecas CSS-in-JS.

**Enfoques avanzados**

**Conceptos**

CSS se ha convertido en un mundo completamente vasto, razón por la que cada desarrollador basa su expertise sobre el uso del lenguaje en temas particulares. Estos temas van desde animaciones, hasta grid systems altamente complejos.  
  
Conozcamos algunos conceptos avanzados que darán pie a la exploración de diferentes ramas que pueden aplicarse a distintos proyectos.

**Metodología de trabajo**

Para evitar conflictos en proyectos con múltiples desarrolladores, es crucial adoptar una metodología CSS que **establezca reglas sobre la organización del código**. Esto garantiza una estructura coherente y una jerarquía clara en los estilos. Entre las opciones populares se encuentran **OOCSS**, **SMACSS** y **BEM**.

**Texto

Descripción generada automáticamente**

**OOCSS (Object-Oriented CSS)** se centra en la creación de estilos reutilizables y modulares, separando la estructura del estilo.

* Hace uso del mismo Principio de Responsabilidad Única de React, pero aplicado a los estilos: cada estilo tendrá una sola tarea y la hará bien.

bject-Oriented CSS (OOCSS) es una metodología de desarrollo que se aplica a la escritura de hojas de estilo (CSS) para páginas web con el objetivo de mejorar la escalabilidad, mantenibilidad y reutilización del código. La idea fue introducida por Nicole Sullivan alrededor de 2008 y desde entonces, ha ganado popularidad entre los desarrolladores frontend debido a sus principios orientados a optimizar y organizar el CSS. OOCSS se basa en dos pilares fundamentales: la separación de estructura y diseño, y la modularidad.

Separación de estructura y diseño

La primera regla de OOCSS implica separar lo más posible la estructura (layout) de un objeto de su diseño (skin). Esto significa que debes separar los estilos que afectan la geometría del objeto (como el ancho, el margen, o la posición) de aquellos estilos que definen su apariencia (como el color, los bordes o las sombras).

Estructura: Se refiere a cómo se organiza el contenido, cómo se posiciona en la página, y cómo interactúan los elementos entre sí en términos de layout. Esto incluye propiedades como width, margin, padding, y display.

Diseño: Se centra en cómo se ven los elementos, es decir, su apariencia visual. Esto cubre propiedades como color, background, border, y box-shadow.

Por ejemplo, si tienes varios botones en tu sitio web que comparten la misma estructura pero varían en color, deberías crear una clase común para la estructura y clases separadas para cada variación de color. Esto facilita la reutilización de estilos y hace que tu CSS sea más fácil de mantener.

Modularidad

El segundo pilar de OOCSS promueve la creación de componentes modulares, lo que significa diseñar la interfaz de usuario como un conjunto de piezas reutilizables. Cada módulo o componente debe ser independiente, de modo que pueda usarse en diferentes contextos sin necesidad de duplicar código. Esto ayuda a construir interfaces complejas a partir de componentes más simples, promoviendo la reutilización de código y la eficiencia.

Objetos: En OOCSS, un "objeto" es un patrón de diseño reutilizable que puede ser utilizado en diferentes contextos. Por ejemplo, un objeto podría ser un botón, una tarjeta de producto, o un formulario de búsqueda.

**BEM (Block Element Modifier)** define una convención de nomenclatura para clases CSS, dividiéndolas en bloques, elementos y modificadores para una fácil comprensión y mantenimiento.

Texto

Descripción generada automáticamente

**SMACSS (Scalable and Modular Arquitecture for CSS)** propone una estructura de estilos modular y escalable, clasificándolos en categorías como base, diseño, módulos, estados y temas.

* Hace uso inteligente de envoltorios y contenedores para hacer que los componentes encajen perfectamente en todas partes.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Preprocesadores**

Un **preprocesador CSS** extiende las capacidades de CSS convencional, permitiendo agregar nuevas características y funcionalidades al código de los estilos. Algunos de los más conocidos son **SASS, LESS**o**Stylus**.

Los preprocesadores CSS como Sass y Less han revolucionado la forma en que escribimos CSS, proporcionando características que el CSS estándar no ofrece, como variables, mixins, funciones y herencia. Estos preprocesadores permiten escribir código de manera más eficiente, modular y mantenible, lo cual es especialmente útil en proyectos grandes y complejos.

Sass (Syntactically Awesome Stylesheets)

Sass es un preprocesador CSS muy poderoso y flexible que ayuda a escribir CSS de manera más eficiente y con menos esfuerzo.

Sass permite usar características como variables, anidamiento, mixins, herencia y otras funciones avanzadas que no están disponibles en CSS puro.

**Variables**: Permiten almacenar valores que se pueden reutilizar en todo el documento, como colores, tamaños de fuente o márgenes.

**Mixins**: Son bloques de código reutilizables que pueden incluirse en otros selectores para evitar la repetición de código.

**Anidamiento:** Sass permite anidar selectores de manera jerárquica dentro de otros selectores, lo que facilita la organización del código y mantiene las relaciones de parentesco claras.

**Herencia (@extend):** Permite que un selector herede los estilos de otro, facilitando la reutilización de estilos y manteniendo el CSS DRY (Don't Repeat Yourself).

**Funciones:** Sass incluye funciones para realizar operaciones y cálculos, permitiendo una mayor dinámica y flexibilidad en la definición de estilos.

Sass ofrece dos sintaxis: SCSS (Sassy CSS), que usa el formato de archivo .scss y tiene una sintaxis similar a CSS, y la sintaxis original, conocida como Sass (o "indent syntax"), que utiliza el formato .sass y omite llaves y puntos y coma, utilizando la indentación para definir las estructuras.

Los preprocesadores permiten resolver el problema de la repetitividad del código CSS con un sistema de variables mejor definido, mixins, functions, y un conjunto de implementaciones que conllevan a un desarrollo más robusto y eficiente.

Texto

Descripción generada automáticamente

La razón por la cual hacemos referencia a tener algún preprocesador, es debido a que nosotros no tomaremos el enfoque de preprocesador para los proyectos a desarrollar a futuro, sino que abordaremos y profesionalizaremos una práctica diferente para estilizar nuestros componentes.

Less (Leaner Style Sheets)

Aunque Less y Sass comparten muchas funcionalidades, existen algunas diferencias en su sintaxis y en algunas características específicas.

Variables: Al igual que Sass, Less permite definir variables para almacenar valores comunes que se pueden reutilizar en todo el CSS.

Mixins: En Less, los mixins permiten incluir clases completas dentro de otras clases, lo que puede incluir sus propiedades o incluso selectores anidados.

Anidamiento: Less también soporta el anidamiento de selectores, lo que permite estructurar el CSS de una manera que refleje la estructura HTML.

Funciones: Less proporciona funciones incorporadas para manipular colores y realizar operaciones aritméticas.

La sintaxis de Less es muy similar a CSS, lo que hace que su curva de aprendizaje sea relativamente baja. Se usa el formato de archivo .less y, al igual que Sass, necesita ser compilado a CSS puro para que los navegadores puedan interpretarlo.

**Diferencias de sintaxis**

**La sintaxis de Sass y Less comparten muchas similitudes porque ambos preprocesadores extienden las capacidades de CSS estándar con características adicionales como variables, mixins y anidamiento. Sin embargo, hay diferencias clave en la sintaxis y en cómo se utilizan algunas de estas características avanzadas.**

**Variables**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**Sintaxis de Archivos**

**Sass: Tiene dos sintaxis: Sass (indentada) y SCSS (similar a CSS). Los archivos Sass usan .sass y los SCSS .scss.**

**Less: Tiene una única sintaxis que es muy similar a CSS y usa la extensión de archivo .less.**

**Compatibilidad**

Cuando los usuarios visitan nuestros sitios web desde diferentes navegadores, pueden surgir problemas con la interpretación de estilos debido a los motores de renderizado que estos utilizan.

Para solventar este problema, se hace uso de los **vendor-prefixes**. Estos permiten a un navegador específico entender correctamente directivas de estilo como rotación, traslación y transformación.

**Texto

Descripción generada automáticamente**

💪A partir de este punto, nos enfocaremos directamente en el modelo de estilos que estaremos usando a lo largo del módulo...

**Introducción a CSS-in-JS**

**¡Y listo! Así nace JSS 😎**una librería que nos permite definir estilos directamente en el código JavaScript.

Su objetivo es guardar en el mismo archivo js toda la stylesheet en un objeto para, finalmente, incorporarlo en las etiquetas HTML.

**Styled Components**

La carrera de librerías CSS-in-JS tuvo una curva bastante pronunciada. Sin embargo, hay uno en especial que ha sabido mantenerse en el podio: **Styled Components**.

Recapitulo de Styled Components:

Styled Components es una biblioteca para React y React Native que permite escribir CSS real en JavaScript, ofreciendo una forma poderosa y única de estilizar aplicaciones. Esta biblioteca utiliza técnicas de CSS-in-JS, lo que significa que los estilos son escritos dentro de archivos JavaScript o TypeScript, permitiendo aprovechar toda la potencia de JavaScript para definir y manipular estilos.

**Creación de un Componente Estilizado**

**Una vez instalada, puedes comenzar a crear componentes estilizados. Los componentes estilizados son básicamente componentes de React que tienen estilos adjuntos a ellos. Se crean utilizando la función styled, seguida de un tipo de elemento HTML. Los estilos se escriben entre backticks (`).**

**Texto

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**En este ejemplo, el color de fondo del Boton cambia dependiendo de si la prop primary es verdadera.**

**Texto

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**Conclusión**

**Styled Components es una herramienta poderosa y flexible para estilizar aplicaciones React. Te permite escribir CSS real dentro de tu JavaScript, lo que significa que puedes usar todas las características de JavaScript para crear estilos dinámicos y reutilizables. Al integrar estilos directamente en tus componentes, facilita la gestión de tus estilos y evita problemas comunes de CSS como el conflicto de nombres de clases. Además, al aprovechar las props y el sistema de temas, puedes crear interfaces de usuario altamente dinámicas y personalizables.**

**¿Te suena familiar? 🧐**Debería ser así, pues te fue presentado desde el módulo 3. Este es el momento en el que lo aplicaremos con mayor profundidad. Para ello, tomaremos un formulario y lo convertiremos en tres componentes: **Button**, **Input**y **Form**, en Styled Components, para mantener un formato de estilos homogéneo.

Lo primero es posicionarnos sobre nuestro proyecto, sobre el cual realizaremos la instalación de **styled-components** y ejecutar el comando

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

También recomendamos ampliamente instalar la **extensión**de styled components en tu VSCode, pues este nos permitirá entender y recomendar prellenado a partir de la sintaxis de **template string** que utilizaremos más adelante.

**Hagámoslo ahora más interesante**. Planteemos algunos escenarios para profundizar un poco más sobre los styled Components y así aprovechar más su potencial. Supongamos que deseamos que el botón cambie de color cuando hagamos **hover**. Esto significa, que queremos aplicar un **estilo específico** cuando el cursor esté sobre el botón

**¿Dónde agregamos esa lógica?**  
En **Styled Components**, podemos manejar esta lógica usando el símbolo ampersand ( & ).

Otro caso a considerar es en el que deseamos que nuestro componente Input únicamente sea morado si es de tipo password. Sino, que sea blanco. ¿Cómo resolverlo? 🧠

Styled Components nos permite definir una propiedad a partir de las **props**que se haya enviado a dicho componente, por lo que podemos ejecutar funciones que definan el estilo resultante al momento de la construcción del componente:

**Texto

Descripción generada automáticamente**

**Theme & customization**

**Implementación de temas en React**

Los **Styled Components** en react ofrecen la capacidad de crear temas para gestionar estilos, colores y otras propiedades de apariencia de manera coherente en una aplicación.

🎨 Estos temas representan conjuntos de **estilos** con paletas de colores, tipos de fuente y, en casos avanzados, librerías de íconos e imágenes.  
  
Al igual que en tu computadora o navegador, definir temas en una aplicación facilita el mantenimiento y la consistencia del diseño.

**Styled Components** nos ayuda a tener una gestión más organizada  
de estilos a partir de su **ThemeProvider**.

**Diseño responsive**

**Principios**

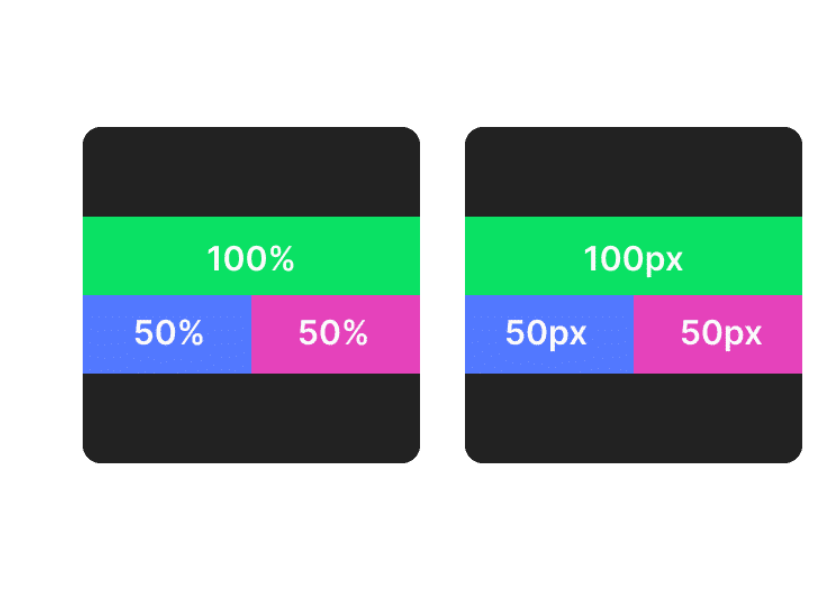
Seguramente ya habrás escuchado o visto varias veces este concepto dado que se volvió algo muy popular en la última década. El **responsive design** engloba todas las prácticas que permiten hacer un sitio web adaptable a diferentes dispositivos y tamaños de pantalla.

* Mantiene una óptima experiencia de usuario tanto en computadoras como tabletas y dispositivos móviles.

**Unidades realtivas**

Parecen ser los mismo expresado en otras unidades, ¿no?  
  
Sin embargo, cuando el dispositivo desde el cual estamos usando el componente cambia, o cuando las dimensiones de la ventana del navegador se reajustan, puede sucedernos esto...

📣 Las**unidades relativas** se ajustan a las dimensiones de la pantalla o del contenedor padre dentro del que esté.

****

📣 Las**unidades relativas** se ajustan a las dimensiones de la pantalla o del contenedor padre dentro del que esté.

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

**Max & Min**

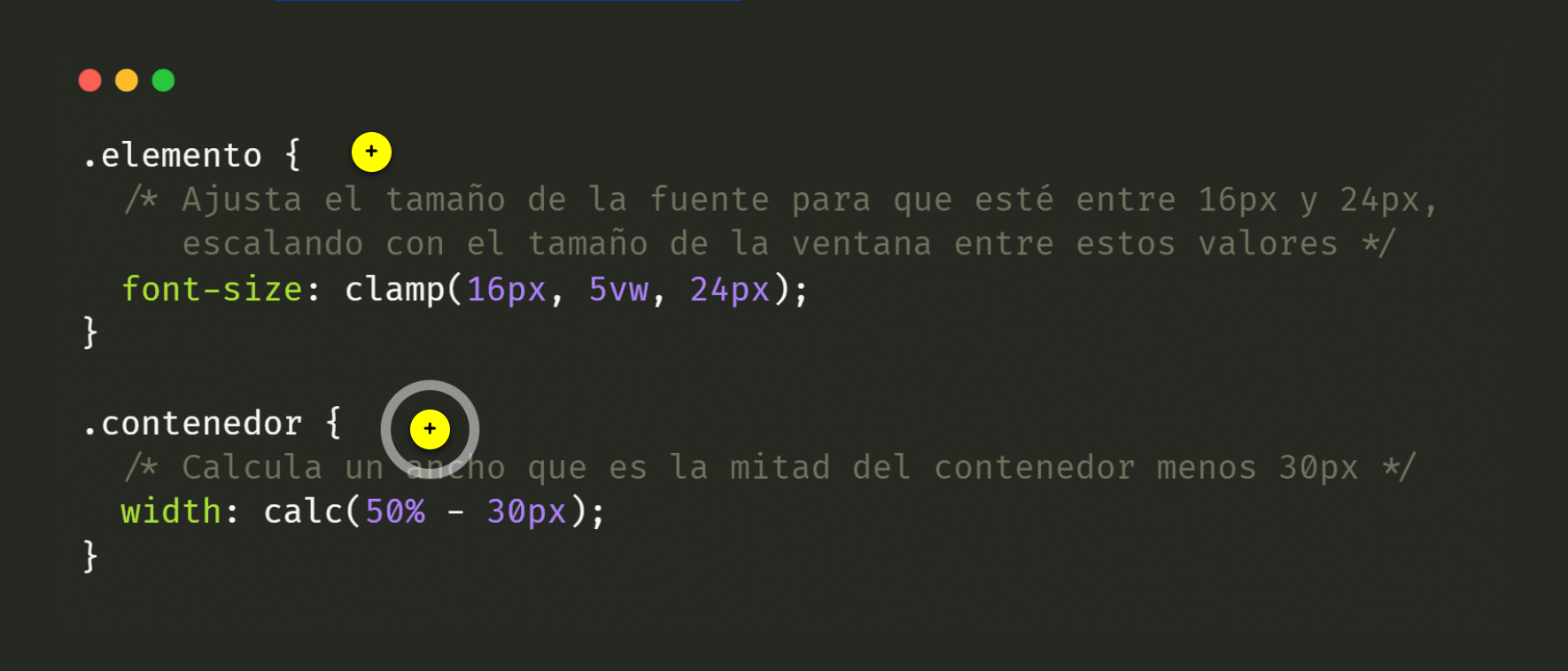
Una vez que comenzamos a usar unidades relativas nos encontramos con el problema de que un componente podría ajustarse al tamaño de la pantalla tanto como le sea posible, pero no siempre deseamos que un componente **se encoja sin límites**. Utiliza **max**y **min**para delimitar hasta dónde puede crecer o encogerse un elemento.

**Texto

Descripción generada automáticamente**

**Funciones matemáticas**

Las funciones **clamp**y **calc**nos permiten realizar ciertas operaciones matemáticas para determinar la dimensión de algún elemento.

****

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

Descripción generada automáticamente**

**Layouts con grid y flexbox**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**Imagen que contiene Diagrama

Descripción generada automáticamente**

Los media queries son útiles para poder trabajar con diferentes dispositivos, pues en ocasiones necesitamos realizar un reajuste de UI dependiendo desde dónde se esté visitando nuestro sitio web. Este cambio de estilos en un punto específico se le conoce como breakpoint.

**Técnicas**

Para terminar de reestructurar el estilo que tiene nuestro formulario, vamos a brindar un estilo responsive a los elementos Input y Form. Veamos cómo realizar esto en el siguiente video.

**Animaciones y transiciones**

**Definciones**

Dentro de los últimos aspectos a revisar para tener una descripción general del CSS aplicado en react, están las animaciones y transiciones.**🚨**Estos conceptos suelen tomarse de manera indiscriminada, pero**no son lo mismo**.

Diremos que las **animaciones**son secuencias de cambios visuales aplicadas a elementos HTML mediante reglas CSS. Permite crear efectos dinámicos y fluidos en nuestra web. Las **transiciones**, por su parte, son cambios graduales en propiedades de estilo como, por ejemplo, color o tamaño.

**Diagrama

Descripción generada automáticamente**

Una vez clara esta diferencia, iniciemos con un ejemplo de cómo se realiza una transición. Tomaremos el formulario con el que ya contamos y haremos que este se expanda en vertical cuando hacemos **hover**. Veamos esto en el siguiente video...

Las transiciones de CSS permiten cambiar propiedades de manera suave durante un periodo de tiempo. Al aplicarlas a un estado **:hover**, puedes crear efectos visuales interactivos cuando el usuario pasa el cursor sobre un elemento.

Aquí tienes un ejemplo simple que ilustra cómo cambiar el color y el tamaño de un botón con una transición suave al pasar el cursor sobre él:

css  
**Texto

Descripción generada automáticamente**

En este ejemplo:

.button es la clase aplicada al botón. Aquí, se define el color de fondo inicial, el color del texto, el padding, y se especifica la transición. transition se utiliza para indicar qué propiedades queremos animar (background-color y transform), la duración de la transición (0.5s) y el tipo de efecto de transición (ease).

.button:hover define los estilos que se aplican cuando el cursor está sobre el botón. Cambia el color de fondo a rojo y aumenta el tamaño del botón (transform: scale(1.1)) de forma suave, gracias a la transición definida en .button.

El resultado es un cambio gradual del color de fondo y un leve aumento de tamaño del botón durante medio segundo cuando pasas el cursor sobre él, creando una experiencia interactiva y visualmente atractiva para el usuario.

**Animaciones**

Las animaciones suponen un tema más complejo, pues están pensadas para poder desarrollar diferentes movimientos, escalamientos o rotaciones a lo largo de un tiempo definido. Para tener un mayor control sobre las animaciones, utilizamos una herramienta importante llamada **keyframes**.

Un **keyframe**es un punto específico en una secuencia de animación  
donde se define un estado o posición.

Al momento de trabajar con animaciones, debemos considerar que cada una de estas va desde un inicio hasta un final, por lo que se puede trabajar con from - to , o bien con porcentajes desde 0% a 100%.

Texto

Descripción generada automáticamente

**En conclusión...**

En esta clase hemos repasado algunos de los conceptos básicos de **estilos** en react, así como la implementación de **styled components**, una librería**css-in-js** que permite la creación de estilos dentro de cada componente de nuestro proyecto.

🚀 Nos adentramos en la implementación del **diseño responsive**, lo que conlleva a mejorar la experiencia del usuario al adaptar el contenido web a distintos dispositivos y tamaños de pantalla.

⚡Finalmente, vimos una introducción a las**animaciones** y **transiciones**que generan mayor dinamismo en nuestros sitios web y obtener aplicaciones modernas