CSS IN JS

CSS-in-JS es una técnica de estilo en la que se usa JavaScript para diseñar componentes. Cuando se analiza el código JavaScript, se genera CSS (generalmente como un <style>elemento) y se adjunta al DOM. Permitiendo abstraer CSS al nivel del componente en sí, utilizando JavaScript para describir estilos de forma declarativa y mantenible.

Hay múltiples implementaciones de este concepto en forma de bibliotecas, la mayoría diseñadas para react, sin embargo, ya hay varios frameworks populares como vue y stvelt también están implementando esta línea de trabajo.

Beneficios

- Pensando en componentes. Ya no tienes que mantener un montón de hojas de estilo. CSS-in-JS abstrae el modelo CSS al nivel del componente, en lugar del nivel del documento (modularidad).
- CSS-in-JS aprovecha todo el poder del ecosistema JavaScript para mejorar CSS.
- Aislamiento de las reglas verdaderas. Los selectores con alcance no son suficientes. CSS tiene propiedades que se heredan automáticamente del elemento principal, si no se definen explícitamente
- Selectores con alcance. CSS tiene un solo espacio de nombres global. Es imposible evitar colisiones de selectores en aplicaciones no triviales. CSS en JavaScript genera nombres de clase únicos de forma predeterminada, cuando se compila en CSS.
- Prefijo de proveedor. Las reglas de CSS tienen automáticamente el prefijo del proveedor, por lo que no tiene que pensar en ello.
- Compartir código. Comparta fácilmente constantes y funciones entre JavaScript y CSS.
- Solo los estilos que están actualmente en uso en su pantalla están en el DOM.
- Eliminación de código muerto.
- Manejo de estilos totalmente dinámicos, definidos por arreglos globales, atributos propios de las clases o modelos de datos.

Implementación

Esta propuesta también incluye una implementación propia de esta librería y busca tener un estilo minimalista de este por medio de la implementación de la POO.

Esta propuesta parte de la utilización de la estructura que hasta el momento estamos construyendo, por lo que iniciaremos por crear dentro de WDevCore/Wmodules un archivo llamado WStyledRender.js.

Dentro de esta haremos uso del WRender, por lo que procederemos a importarlo.

```
import { WRender } from "./WComponentsTools.js";
```

posteriormente será necesario tener alguna especie de intelliSense, dado que no tendremos la ayuda por defecto que aportan los editores/IDES en el manejo de archivos CSS, se hará complicado gestionar los nombres de propiedades existentes en CSS, por ello crearemos una clase que incluya esta lista de nombres y la usaremos como guía para implementar nuestros propios estilos.

Esta clase tendrá dentro de sus atributos la lista completa de propiedades CSS, con un valor por defecto en NULL(es posible que haya propiedades faltantes no incluidas en la lista)

```
class CSSProps {
    "align-content" = null;
    "align-items" = null;
    "align-self" = null;
    "all" = null;
    "animation" = null;
    "animation-delay" = null;
    "animation-direction" = null;
    "animation-duration" = null;
    "animation-fill-mode" = null;
    "animation-iteration-
count" = null;
    "animation-name" = null;
    "animation-play-state" = null;
    "animation-timing-
function" = null;
    "caption-side" = null;
    "caret-color" = null;
    //"@charset" = null;
    "clear" = null;
    "clip" = null;
    "color" = null;
    "column-count" = null;
    "column-fill" = null;
    "column-gap" = null;
    "column-rule" = null;
    "column-rule-color" = null;
    "column-rule-style" = null;
    "column-rule-width" = null;
    "column-span" = null;
    "column-width" = null;
    "columns" = null;
    "content" = null;
    "counter-increment" = null;
    "counter-reset" = null;
    "cursor" = null;
    "direction" = null;
    "display" = null;
    "empty-cells" = null;
    "filter" = null;
    "flex" = null;
    "flex-basis" = null;
    "flex-direction" = null;
    "flex-flow" = null;
    "flex-grow" = null;
    "flex-shrink" = null;
    "flex-wrap" = null;
    "float" = null;
    "font" = null;
    "font-family" = null;
    "font-feature-settings" = null;
    "font-kerning" = null;
    "font-language-
override" = null;
    "font-size" = null;
    "font-size-adjust" = null;
    "font-stretch" = null;
    "font-style" = null;
    "font-synthesis" = null;
    "font-variant" = null;
    "font-variant-
alternates" = null;
    "font-variant-caps" = null;
    "font-variant-east-
asian" = null;
    "font-variant-
ligatures" = null;
    "font-variant-numeric" = null;
    "font-variant-position" = null;
    "font-weight" = null;
    "grid" = null;
    "grid-area" = null;
    "grid-auto-columns" = null;
    "grid-auto-flow" = null;
    "grid-auto-rows" = null;
    "grid-column" = null;
    "grid-column-end" = null;
    "grid-column-gap" = null;
    "grid-column-start" = null;
    "grid-gap" = null;
    "grid-row" = null;
    "grid-row-end" = null;
    "grid-row-gap" = null;
    "grid-row-start" = null;
    "grid-template" = null;
    "grid-template-areas" = null;
    "grid-template-columns" = null;
    "grid-template-rows" = null;
    "hanging-punctuation" = null;
    "height" = null;
    "hyphens" = null;
    "image-rendering" = null;
    "isolation" = null;
    "justify-content" = null;
    "left" = null;
    "letter-spacing" = null;
    "line-break" = null;
    "line-height" = null;
    "list-style" = null;
    "list-style-image" = null;
    "list-style-position" = null;
    "list-style-type" = null;
    "margin" = null;
    "margin-bottom" = null;
    "margin-left" = null;
    "margin-right" = null;
    "margin-top" = null;
    "max-height" = null;
    "max-width" = null;
    "min-height" = null;
    "min-width" = null;
    "mix-blend-mode" = null;
    "object-fit" = null;
    "object-position" = null;
    "opacity" = null;
    "order" = null;
    "orphans" = null;
    "outline" = null;
    "outline-color" = null;
    "outline-offset" = null;
    "outline-style" = null;
    "outline-width" = null;
    "overflow" = null;
    "overflow-wrap" = null;
    "overflow-x" = null;
    "overflow-y" = null;
    "padding" = null;
    "padding-bottom" = null;
    "padding-left" = null;
    "padding-right" = null;
    "padding-top" = null;
    "page-break-after" = null;
    "page-break-before" = null;
    "page-break-inside" = null;
    "perspective" = null;
    "perspective-origin" = null;
    "pointer-events" = null;
    "position" = null;
    "quotes" = null;
    "resize" = null;
    "right" = null;
    "scroll-behavior" = null;
    "tab-size" = null;
    "table-layout" = null;
    "text-align" = null;
    "text-align-last" = null;
    "text-combine-upright" = null;
    "text-decoration" = null;
    "text-decoration-color" = null;
    "text-decoration-line" = null;
    "text-decoration-style" = null;
    "text-indent" = null;
    "text-justify" = null;
    "text-orientation" = null;
    "text-overflow" = null;
    "text-shadow" = null;
    "text-transform" = null;
    "text-underline-
position" = null;
    "top" = null;
    "transform" = null;
    "transform-origin" = null;
    "transform-style" = null;
    "transition" = null;
    "transition-delay" = null;
    "transition-duration" = null;
    "transition-property" = null;
    "transition-timing-
function" = null;
    "unicode-bidi" = null;
    "user-select" = null;
    "vertical-align" = null;
    "visibility" = null;
    "white-space" = null;
    "widows" = null;
    "width" = null;
    "word-break" = null;
    "word-spacing" = null;
    "word-wrap" = null;
    "writing-mode" = null;
    "z-index" = null;
    "background" = null;
    "background-color" = null;
    "background-image" = null;
    "border" = null;
    "border-radius" = null;
    "border-top" = null;
    "border-right" = null;
    "border-left" = null;
    "border-bottom" = null;
    "box-shadow" = "";
}
```

Posteriormente procederemos a crear una clase bastante sencilla que servirá como modelo de estructura de selector CSS. El constructor de esta clase solamente recibirá como parámetro el nombre del clase o selector, esto deberá ser una cadena de texto la cual puede incluir cualquier forma de escritura de selector CSS (ejemp: “#componetId” / “.ClassName” / “tagName”), así mismo recibirá un segundo parámetro que incluirá la lista de propiedades de ese selector y este tendrá un valor por defecto definido por un objeto del tipo CSSProps.

```
class WCssClass {
    constructor(ClassName, PropsList = (new CSSProps())) {
        this.Name = ClassName;
        this.CSSProps = PropsList;
    }
}
```

Una vez construidas las estructuras necesarias, crearemos un WebComponent que defina todo el comportamiento del bloque de estilos, que pueda distinguir entre estilos normales, mediaquerys y keyframes. Para este caso no utilizaremos shadowRoot dado que la intención de este componente es que afecte a elementos externos a este ya sea elementos globales del DOM o componentes que lo incluyan.

```
class WStyledRender extends HTMLElement {
    constructor() {
        super();
    }
    attributeChangedCallBack() {
        this.DrawStyle();
    }
    connectedCallback() {
        if (this.innerHTML != "") {
            return;
        }
        this.DrawStyle();
    }
    DrawStyle() {
        let styleFrag = {
            type: "style",
            props: {},
            children: []
        }
        if (this.ClassList != undefined && this.ClassList.__proto__ == Array.prototype) {
            styleFrag.children.push(this.DrawClassList(this.ClassList));
        }
        if (this.MediaQuery != undefined && this.MediaQuery.__proto__ == Array.prototype) {
            this.MediaQuery.forEach(MediaQ => {
                let MediaQuery = `@media ${MediaQ.condicion}{
                    ${this.DrawClassList(MediaQ.ClassList)}
                }`;
                styleFrag.children.push(MediaQuery);
            });
        }
        if (this.KeyFrame != undefined && this.KeyFrame.__proto__ == Array.prototype) {
            this.KeyFrame.forEach(KeyF => {
                let KeyFrame = `@keyframes ${KeyF.animate} {
                    ${this.DrawClassList(KeyF.ClassList)}
                }`;
                styleFrag.children.push(KeyFrame);
            });
        }
        this.append(WRender.createElement(styleFrag));
    }
    DrawClassList(ClassList) {
        let bodyStyle = "";
        ClassList.forEach(Class => {
            let bodyClass = "";
            if (Class.__proto__ == Object.prototype) {
                for (const prop in Class.CSSProps) {
                    bodyClass = bodyClass + `${prop}: ${Class.CSSProps[prop]};`;
                }
                bodyClass = `${Class.Name} {${bodyClass}}`;
                bodyStyle = bodyStyle + bodyClass;
            }
        });
        return bodyStyle;
    }
}
```

Esta clase tendrá algunas características interesantes que podremos revisar parte por parte. Lo primero es que la creación de toda la lógica del componente style dependerá de la funcionalidad de WRender.createElement, esto permitirá compactar de manera muy significativa el código requerido para poder construir un componente tan complejo. Por otro lado, como es costumbre, no basta con solo verificar que las propiedades existen para proceder la construcción, sino que se debe verificar que sean del tipo correcto para evitar errores lógicos dentro de la aplicación, por ello siempre usamos la

validación por medio de la propiedad __proto__ la cual nos permite determinar si el objeto enviado es del prototipo correcto.

```
if (this.ClassList != undefined && this.ClassList.__proto__ == Array.prototype)
```

```
if (this.MediaQuery != undefined && this.MediaQuery.__proto__ == Array.prototype)
```

```
if (this.KeyFrame != undefined && this.KeyFrame.__proto__ == Array.prototype)
```

este tipo de validaciones son indispensables para evitar comportamientos inesperados dentro de cualquier app. Así mismo se debe verificar que cada objeto de la lista también posee un prototipo adecuado por ello dentro de la función DrawClassList se realiza una verificación similar, esta vez utilizando la definición del prototipo de WCssClass.

```
    if (Class.__proto__ == WCssClass.prototype)
```

Por último, solo habrá que definir el componente como customElement y exportar las clases requeridas para facilitar la escritura de estas estructuras.

```
    customElements.define("w-style", WStyledRender);
    export { WCssClass };
```

Uso de Estilos Normales.

para hacer uso del componente, basta con solo referéncialo y renderizar un w-style dentro de los componentes que necesitemos.
La estructura simple seria la siguiente:

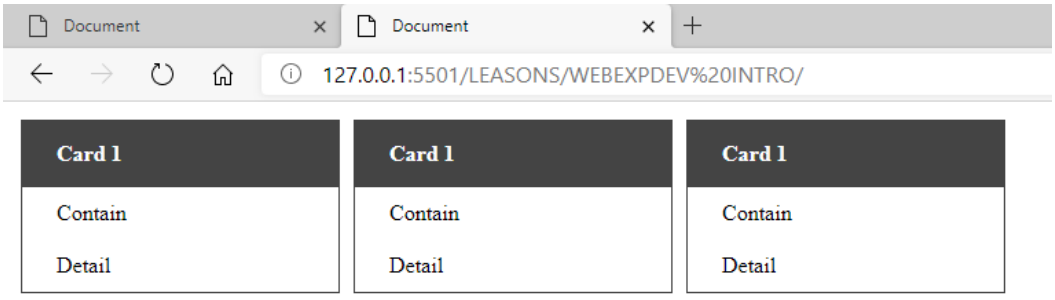
```
{ type: "w-style", props: {
            ClassList: [
                new WCssClass(".card", {
                    display: "inline-block",
                    border: "solid 1px #444",
                    margin: "5px",
                }), new WCssClass(".title", {
                    "font-weight": "bold",
                    padding: "15px 25px",
                    "background-color": "#444",
                    color: "#fff",
                    display: "block",
                }), new WCssClass("section", {
                    padding: "10px 25px",
                    width: "180px"
                }),
            ]
        }
}
```

En este ejemplo se incluyen dos clases **card** y **title,** con propiedades comunes de estilo y la modificación de un selector por medio del tagName **section** para incluirlo

directamente en nuestro componente, haríamos la modificación en este de la siguiente forma (archivo CardComponnet.js, ubicado dentro WDevCore/Wcomponents):

```
import { WRender } from "../WModules/WComponentsTools.js";
import { WCssClass } from "../WModules/WStyledRender.js";
class WCardComponent extends HTMLElement {
    constructor() {
        super();
        this.attachShadow({ mode: "open" });
    }
    connectedCallback() {
        this.shadowRoot.append(this.DrawCard());
        const style = {
            type: "w-style",
            props: {
                ClassList: [
                    new WCssClass(".card", {
                        display: "inline-block",
                        border: "solid 1px #444",
                        margin: "5px",
                    }), new WCssClass(".title", {
                        "font-weight": "bold",
                        padding: "15px 25px",
                        "background-color": "#444",
                        color: "#fff",
                        display: "block",
                    }), new WCssClass("section", {
                        padding: "10px 25px",
                        width: "180px"
                    }),
                ]
            }
        }
        this.shadowRoot.append(WRender.createElement(style));
    }
    DrawCard() {
        return WRender.createElement({
            type: "label",
            props: { className: "card" },
            children: [
                { type: "label", props: { className: "title", innerText: this.element.title } },
                { type: "section", props: { innerText: this.element.Contain } },
                { type: "section", props: { innerText: this.element.Detail } },
            ]
        });
    }
}
customElements.define("w-card", WCardComponent);
```

al incluir el w-style dentro del shadowRoot del componente, evitamos directamente que este tenga afectación sobre elementos externos al propio CardComponent. Asi mismo evitamos que otros estilos independientes afecten a sus elementos.



Uso de MediaQuerys

Para hacer uso de MediaQuerys debemos incluir una propiedad con su mismo nombre, esta deberá ser un arreglo de objetos, donde cada elemento del arreglo deberá tener la propiedad condición la cual determinará qué tipo de MediaQuerys implementaremos.
y una ClassList con sus estilos WCssClass, que determinaran cual será el comportamiento de los elementos una vez que se cumpla la condición.

```
        MediaQuery: [{
               condicion: "(max-width: 1200px)",
               ClassList: [
                   new WCssClass(`.myClass`, {
                       display: "grid",
                   }),
               ]
        },{
               condicion: "(max-width: 600px)",
               ClassList: [
                   new WCssClass(`.myClass`, {
                       display: "flex",
                   }),
               ]
        }],
```

La estructura completa de cómo se vería un w-style con MediaQuerys incluidos, seria la siguiente (Modificando la variable style del CardComponent):

```
    const style = {
         type: "w-style",
         props: {
             ClassList: [
                 new WCssClass(".card", {
                     display: "inline-block",
                     border: "solid 1px #444",
                     margin: "5px",
                 }), new WCssClass(".title", {
                     "font-weight": "bold",
                     padding: "15px 25px",
                     "background-color": "#444",
                     color: "#fff",
                     display: "block",
                 }), new WCssClass("section", {
                     padding: "10px 25px",
                     width: "180px"
                 }),
             ], MediaQuery: [{
                 condicion: "(max-width: 700px)",
                 ClassList: [
                     new WCssClass(`.card`, {
                         display: "block",
                     })
                 ]
             }]
         }
     }
```

Al aplicarlo a nuestro CardComponent tendremos el siguiente resultado:

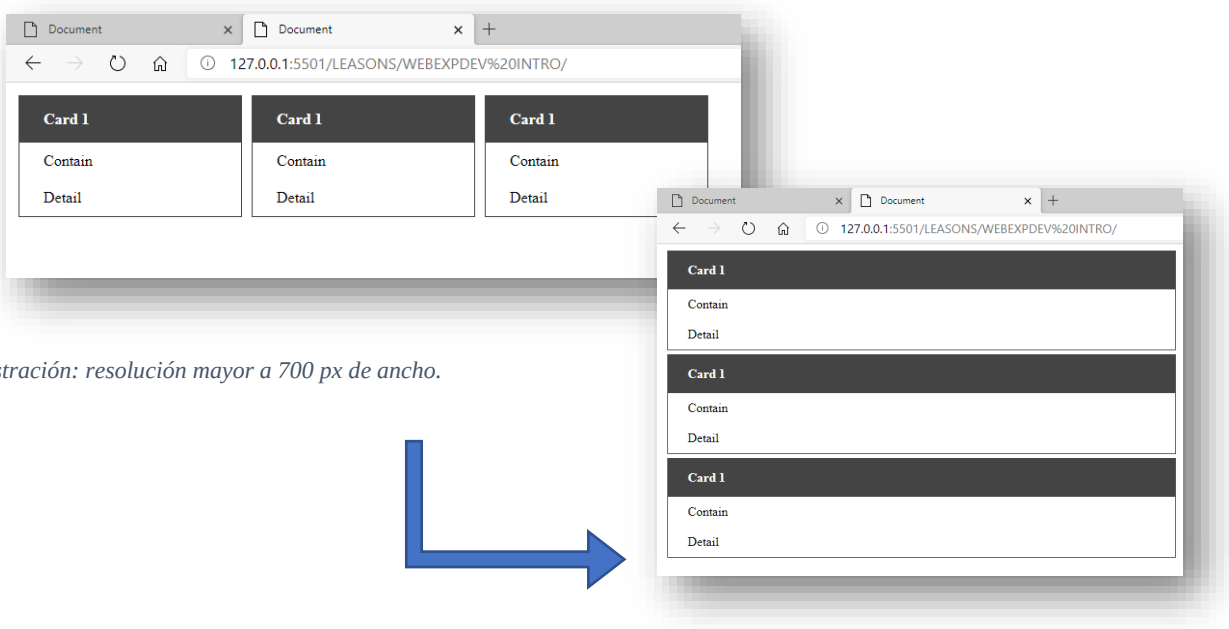


Ilustración: resolución mayor a 700 px de ancho.

Ilustración: Resolución menos a 700px

Uso de KeyFrames

Al igual que los mediaquery los keyframe se rigen bajo la misma lógica que implica un arreglo de objetos que poseen una propiedad anímate y una ClassList que determinan que propiedades CSS se animaran. Ejemplo:

```
KeyFrame: [{
        animate: "slide",
        ClassList: [
            new WCssClass("0%", {
                transform: "translateX(-25%)"
            }), new WCssClass("100%", {
                transform: "translateX(25%)"
            })
        ]
 }]
```