Construcción de un entrenedor/simulador para enseñar el lenguaje de Marcas de Hipertexto HTML

Elfar Didier Morantes Sánchez

Tabla de contenido

[Resumen: 2](#_Toc137225656)

[Abstract: 3](#_Toc137225657)

[Introducción: 4](#_Toc137225658)

[Desarrollo 9](#_Toc137225659)

[Index.html 10](#_Toc137225660)

[Asignación de eventos onload 11](#_Toc137225661)

[Funciones para generar lienzos de trabajo 13](#_Toc137225662)

[Variables con el HTML de los lienzos de trabajo 15](#_Toc137225663)

[Funcionas relacionadas con la operación arrastrar 24](#_Toc137225664)

[Funciones relacionadas con la operación soltar 25](#_Toc137225665)

[Funciones relacionadas con generar nuevos identificadores a los elementos creados 26](#_Toc137225666)

[Funciones relacionadas con regenerar nuevos elementos en la sección de controles 33](#_Toc137225667)

[Funciones relacionadas con crear la sección de código de los elementos arrastrados 36](#_Toc137225668)

[Funciones para eliminar elementos 37](#_Toc137225669)

[Conclusiones 40](#_Toc137225670)

[Código 1. index.html 12](#_Toc137225681)

[Código 2. Sección para arrastrar y soltar y sección de código 12](#_Toc137225682)

[Código 3. Función para asignar eventos 14](#_Toc137225683)

[Código 4. Asignación de eventos a los controles de los formularios 14](#_Toc137225684)

[Código 5. Asignación de eventos para generar el lienzo de trabajo 14](#_Toc137225685)

[Código 6. Funciones para generar lienzos de trabajo 16](#_Toc137225686)

[Código 7. Asignación de Valores al lienzo padre 16](#_Toc137225687)

[Código 8. Variables con los valores que se asignarán al lienzo padre 24](#_Toc137225688)

[Código 9. Asignación de id especificos que representan filas y columnas 25](#_Toc137225689)

[Código 10. Función arrastrar() 25](#_Toc137225690)

[Código 11. Instrucción que almacena elementos de la operación arrastrar 26](#_Toc137225691)

[Código 12. Función permitirSoltar() 26](#_Toc137225692)

[Código 13. Función soltar() 27](#_Toc137225693)

[Código 14. Código que permite mover un control a una celda de destino 27](#_Toc137225694)

[Código 15. Función generarNuevoId() 33](#_Toc137225695)

[Código 16. Arreglos para llevar el conteo de los elementos arrastrados/tocados 34](#_Toc137225696)

[Código 17. Función regenerarElementoBase() 36](#_Toc137225697)

[Código 18. Función enviarCodigoHTML 38](#_Toc137225698)

[Código 19. Creación de caja de texto para alojar el código de los elementos arrastrados 38](#_Toc137225699)

[Código 20. Soltar para Eliminar 38](#_Toc137225700)

[Código 21. funcion EliminarDelArreglo() 40](#_Toc137225701)

[Código 22. EliminarDeLaCajaDeCodigo() 40](#_Toc137225702)

# Resumen:

¿Cómo facilitar el proceso de enseñanza-aprendizaje de los conceptos básicos de la programación en lenguaje de marcas HTML y superar el escollo para asimilar dichos conceptos mediante el uso de una base pedagógica en combinación con un entrenador? La propuesta de este trabajo de grado es, **desarrollar un entrenador** que le permita al estudiante, en especial al estudiante a distancia virtual, precisar y definir los conceptos abstractos de la programación en lenguaje de marcas HTML, de manera que tenga unas bases sólidas para continuar su aprendizaje de conceptos más avanzados de forma autónoma, utilizando las facilidades, ayudas y características que le ofrece el entrenador mediante el aprendizaje activo y colaborativo. Por tanto, el propósito de este artículo es describir el desarrollo de un sistema para generar componentes del tipo drag and drop –arrastre y suelte-, de manera que el estudiante inicialmente no tendrá que saber cómo construir algún elemento del lenguaje de marcas HTML, sino que sencillamente lo arrastrará de un menú que contiene todos los elementos esenciales para construir formularios.

**Palabras Claves:** Entrenador, Programación, Html, Formularios,

**Línea de investigación:** Pensamiento sistémico y educación

Abstract:

How to facilitate the teaching-learning process of the basic concepts of programming in HTML markup language and overcome the obstacle to assimilate these concepts by using a pedagogical base in combination with a trainer? The proposal of this grade work is to develop a trainer that allows the student, especially the virtual distance student, to specify and define the abstract concepts of programming in HTML markup language, so that they have a solid foundation to continue. their learning of more advanced concepts autonomously, using the facilities, aids and features offered by the trainer through active and collaborative learning. Therefore, the purpose of this article is to describe the development of a system to generate components of the drag and drop type -drag and drop-, so that the student initially will not have to know how to build some element of the HTML markup language, but rather which will simply drag you out of a menu containing all the essential elements for building forms.

**Keywords:** Trainer, Programming, Html, Forms,

**Research line:** Systemic thinking and education

# Introducción:

La enseñanza de los lenguajes de programación en las carreras técnicas, tecnológicas y profesionales relacionadas con los Sistemas, la Electrónica y las Computadoras es esencial para desarrollar competencias básicas en los aspirantes a estas titulaciones. Sin embargo, la enseñanza de estos lenguajes no es una actividad que se realice de manera orgánica ni didáctica, y muchas veces se reduce a remitirse a los libros y a practicar con el código. Como lo indican algunos investigadores del campo de la enseñanza de la programación “*Ver la solución de un problema en un libro, o ver al profesor construyéndola en el tablero, no garantiza la adecuada generación de habilidades en el estudiante. Y aunque en un porcentaje significativo de los casos este enfoque funciona adecuadamente (así se ha enseñado programación durante los últimos 20 años), el problema es que su éxito depende de factores que no controlamos. (Villalobos, Casallas, Marcos 2005).”* Posiblemente estas falencias se deban a que existen muy pocas herramientas que sean intuitivas para el usuario y le permitan apropiarse cómodamente de los conceptos asociados a los lenguajes de programación. Indudablemente, si se quiere aprender a hacer pan, se tendrá que interactuar con la masa, untarse las manos, amasar, y meter al horno dicha masa para obtener resultados. Así mismo, si se quiere aprender a programar, se tiene que interactuar con el código, manipularlo, depurarlo, y modificarlo una y otra vez para obtener resultados.

No obstante, surge una pregunta: ¿Habrá una forma diferente y más entretenida de enseñar los lenguajes de programación? ¿Existirá una forma más “amigable” de acercar los conceptos de la programación que sea diferente a tratar de imaginar los conceptos abstractos presentados por un libro o por un profesor? Como lo mencionan Villalobos, Casallas, Marcos 2005 con respecto a presentar conceptos en clase o demostrar la forma de utilizar alguna estructura de programación mediante la solución de un ejercicio,  *“Dicha manera de enseñar a programar se basa en la esperanza de que el estudiante sea capaz de detectar patrones en los problemas planteados y los logre asociar con las técnicas que usa el profesor cuando desarrolla un ejemplo en el tablero y que luego, el estudiante sea capaz de hacer la generalización adecuada en su cabeza, para poder aplicar esa asociación patrón-técnica en la resolución de otros problemas (Villalobos, Casallas, Marcos 2005).”*

Evidentemente existe un vasto acervo científico y documental sobre los lenguajes de programación, y existen tantos lenguajes de programación casi como dialectos en el planeta y, por tanto, es necesario delimitar este dilema a un solo lenguaje de programación, de manera que sea viable analizar la posibilidad de construir una herramienta, simulador o entrenador que facilite la adquisición de los conocimientos relacionados con dicho lenguaje de programación. Para este caso particular, se seleccionará el lenguaje de marcas de hipertexto HTML por su gran utilidad y versatilidad en el desarrollo de páginas de Internet. En el siglo XXI la mayoría de la información se encuentra disponible en la red, y por tanto, es pertinente seleccionar el lenguaje HTML como objeto de estudio para construir un simulador o entrenador que permita facilitar el aprendizaje de las nociones asociadas a este lenguaje, y allanar el proceso de enseñanza-aprendizaje relacionado.

Quien haya tenido experiencia o relación desarrollando software en lenguajes de marcas como HTML habrá notado que este lenguaje está conformado por una gran cantidad de conceptos abstractos tales como etiquetas, botones, cajas de texto, identificadores, eventos, entre otros conceptos fundamentales de este lenguaje. Para algunos estudiantes es difícil comprender estos conceptos que, inicialmente, no tienen muchos referentes palpables, y otros estudiantes tardan un tiempo considerable en comprender la esencia de estas ideas.

El asunto se torna más interesante cuando el desarrollo de estas competencias de programación se realiza a través de la modalidad a distancia virtual. Es el aprendiz quien por sí mismo debe construir sus definiciones y conceptos a partir del material de estudio proporcionado. Evidentemente tendrá la colaboración del tutor a través de foros y otros espacios destinados para la comunicación, pero es claro que el material proporcionado al estudiante debe ser de una calidad tal que le permita al estudiante desarrollar sus facultades intelectuales y construir conocimiento. Además, otro apuro a considerar en la programación en lenguaje de marcas HTML es el proceso de editar el código, y luego visualizarlo. Normalmente este es un proceso que consta de dos partes separadas en el tiempo, pues en primer lugar se debe escribir el código con un editor de texto plano, guardar los cambios con una extensión de archivo .html, y finalmente visualizar los productos con otra herramienta. Es decir, primero se debe escribir el código con una herramienta, y luego visualizar los frutos en otra. Esto implica que el estudiante debe realizar varios pasos, normalmente secuenciales y con diferentes herramientas, antes de poder percibir algún cambio en el proyecto web que esté desarrollando. En este punto, y habiendo identificado los distintos dilemas, se podría formular la siguiente pregunta: ¿Será posible construir una herramienta que le permita al estudiante apropiarse de estos conceptos abstractos, de manera que se vuelvan concretos, precisos y palpables? ¿Será factible desarrollar un instrumento que facilite el proceso de edición y visualización de código html, de manera que el estudiante experimente simultáneamente la escritura y la percepción de la página web?

Si se define someramente un simulador como una herramienta que permite reproducir sensaciones, experiencias o comportamientos, entonces, se podría pensar en la herramienta que se quiere desarrollar para el estudiante como un simulador. En esta investigación se podrá usar indistintamente la palabra simulador y entrenador, pues el simulador permitirá reproducir experiencias similares a las reales (programando una página web), y el entrenador le permitirá seguir aprendiendo y seguir entrenándose en los campos relacionados con el desarrollo de páginas web. Pero, ¿Será posible que un simulador/entrenador le permita al estudiante acercarse más a los conceptos básicos del lenguaje de marcas HTML? ¿Podrá el simulador/entrenador reproducir la experiencia de programar, pero haciéndolo de una manera más sencilla, y quizás, hasta intuitiva y divertida?

Como lo indica Villalobos 2009, “*Como parte importante de las estrategias que se desarrollan frente al problema de enseñar a programar, se encuentran las herramientas de soporte. Principalmente herramientas de visualización y animación de algoritmos. Los instructores y profesores han encontrado gran soporte en estas herramientas cuando se mide el impacto que tienen en las sesiones de clase y laboratorios. Estas herramientas involucran a los estudiantes en diversos niveles de compromiso con el aprendizaje. (Villalobos 2009)”*. Son muchos los casos que ilustran la utilidad de los simuladores y su habilidad para reproducir ciertos comportamientos o experiencias. Por ejemplo, muchos pilotos de carreras entrenan con simuladores de autos para mejorar sus tiempos e incrementar sus habilidades. Los pilotos de aviones y helicópteros entrenan en simuladores de vuelo para reproducir ciertas experiencias críticas y de esa manera, estar preparados para eventuales situaciones. Si los simuladores han tenido éxito en estos campos, entonces se podría pensar que un simulador de programación le permitirá al estudiante reproducir la experiencia de programar en un lenguaje de marcas HTML de una manera más amigable, haciendo los conceptos más relevantes, concretos y aplicables, coadyuvando al desarrollo de las competencias orientadas a la programación, no solo como una herramienta complementaria a la formación impartida por el tutor y la información que reside en los libros, sino como un elemento de aprendizaje autocontenido que le permite al estudiante por sí mismo apropiarse de los conocimientos.

Redondeando todas estas ideas, se puede afirmar que se ha identificado una problemática: ¿Cómo facilitar el proceso de enseñanza-aprendizaje de los conceptos básicos de la programación en lenguaje de marcas HTML y superar el escollo para asimilar dichos conceptos mediante el uso de un simulador?

La propuesta de este proyecto es desarrollar un simulador que le permita al estudiante, en especial al estudiante a distancia virtual, precisar y definir los conceptos abstractos de la programación en lenguaje de marcas HTML, haciendo énfasis en aquellos conceptos y nociones relacionados con los formularios web – herramienta primordial para el intercambio de información entre un usuario y un servicio-, de manera que tenga unas bases sólidas para continuar su aprendizaje de conceptos más avanzados de manera autónoma, utilizando las facilidades, ayudas y características que le ofrece el simulador. La forma en que este simulador le simplificará al estudiante el proceso de apropiarse de los conceptos será porque utilizará un sistema para generar componentes del tipo drag and drop –arrastre y suelte-, de manera que el estudiante inicialmente no tiene que saber cómo construir un botón o una caja de texto en el lenguaje de marcas html, sino que sencillamente lo arrastrará de un menú que contiene todos los elementos esenciales para construir formularios. Además, en tiempo real podrá observar el código html asociado a dicho elemento, y podrá cambiar sus propiedades y verificar su comportamiento simultáneamente, sin necesidad de guardar el código, y refrescar el navegador para percibir los cambios. Inclusive el estudiante podrá ir más allá del sistema de “arrastre y suelte” y podrá seguir escribiendo código html adicional, de manera que pueda añadir nuevas características y nuevas funcionalidades a su proyecto web. Finalmente, el estudiante podrá copiar el código generado a partir de arrastrar y soltar componentes, y podrá utilizarlo en entornos de producción de la vida real.

La intención es que al estudiante se le facilite el proceso enseñanza-aprendizaje de los conceptos relacionados con los formularios HTML, utilizando una única herramienta que le permita arrastrar/escribir código para componentes, verificar su funcionamiento, y realizar cambios en tiempo real, simplificando los procedimientos relacionados con la creación de los mismos, y aplicando a la vez un modelo pedagógico adecuado para la utilización de la herramienta, de manera que se puedan explotar las ventajas socio constructivistas del aprendizaje cooperativo centrado en el estudiante. Como lo indica Villalobos 2009 “*En la búsqueda de propuestas actuales frente al tema de innovar la enseñanza de la programación, muchos autores concuerdan con la necesidad de integrar la generación de habilidades más que la transmisión de conocimientos planos, especialmente en el nivel de educación superior. (Villalobos 2009)”.* Por tanto, se espera que el estudiante pueda desarrollar habilidades de creación de páginas web con el uso del simulador, más que transmitir el concepto de etiqueta, o el concepto de página web.Evidentemente, desarrollar un simulador que abarque todos los conceptos del lenguaje de marcas HTML es pretencioso y podría convertirse en una tarea compleja de realizar. Por tal razón, el simulador solamente incluirá conceptos relacionados con los formularios web, pues son la herramienta por antonomasia para la comunicación entre un ser humano y una página web, de manera que se pueda verificar la apropiación de los conceptos relacionados con la construcción de elementos HTML de una página web..

# Desarrollo

Se comienza diseñando una interfaz gráfica de la siguiente manera;

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ilustración 1. Interfaz Gráfica

El despliegue de la interfaz gráfica se puede visualizar en: <https://didiermorantes.github.io/AvanceMaestria/>

El código fuente desarrollado se puede observar en: <https://github.com/didiermorantes/AvanceMaestria>

Como se observa en la ilustración 1, se diseña un tablero de trabajo en el que la parte superior se selecciona el layout, es decir la visualización de filas y columnas. Se ofrece un diseño de 2columnasx12filas, de 3columnasx12filas, y de 12columnasx12filas. La siguiente sección es la de componentes, que posee los controles para configurar un formulario, y un contador que indica cuántos componentes se han alojado en la tercera sección, que se compone de dos columnas por colores. La columna de la izquierda, identificada de color verde, que será el área donde se arrastran y sueltan los componentes, y la columna de la derecha, identificada de color azul, donde aparecerá el código fuente de los elementos arrastrados a la zona de color verde. Se ha utilizado el framework de CSS denominado Bootstrap 5 para crear todo el diseño de filas y columnas que servirá como lienzo de trabajo para arrastrar y soltar elementos.

## Index.html

En el código 1 se observa el código html de la interfaz gráfica.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<link rel="stylesheet" href="assets/bootstrap5/css/bootstrap.css">

<link rel="stylesheet" href="assets/css/styles.css">

<link rel="icon" type="image/x-icon" href="assets/img/favicon.ico">

<body onload="asignarEventos()">

<div class="container text-center">

<div class="row">

<div class="col-md-12">

<div class="row">

<div class="col-md-12 seccion-layout">

<span class="text-center titulo-layout"><h3>Layout</h3></span><br>

<button id="btn2x12" class="btn btn-outline-success">2Columnas X 12Filas</button>

<button id="btn3x12" class="btn btn-outline-success">3Columnas X 12Filas</button>

<button id="btn12x12" class="btn btn-outline-success">12Columnas X 12Filas</button>

</div>

</div>

<div class="row">

<div id="contenedorControles" class="col-md-12 seccion-controles">

<span class="text-center titulo-controles"><h3>Componentes: <span id="spanCantidadElementos">0</span> </h3></span><br>

<input type="button" value="button" draggable="true" id="button0" >

<input type="text" value="text" size="8" draggable="true" id="text0" >

<label value="label" draggable="true" id="label0" >label</label>

<input type="radio" value="radio" name="grupoRadio" draggable="true" id="radio0" >

<input type="checkbox" value="checkbox" name="checkbox0" draggable="true" id="checkbox0" >

<input type="submit" value="submit" draggable="true" id="submit0" >

<input type="text" value="modificame" size="6" draggable="true" id="independiente0" >

</div>

</div>

</div>

</div>

<div class="row">

<div id='lienzoPadre' class="col-md-6 fondo-lienzo">

</div><!-- fin col-md-4 -->

<div class="col-md-6 fondo-codigo">

<!-- inicio del codigo html. Div para alojar cajas de texto con código html -->

<div id="cajaDeCodigo">

</div>

</div>

</div>

</div>

<script src="assets/bootstrap5/js/bootstrap.bundle.js"></script>

<script src="assets/js/scripts.js"></script>

</body>

</html>

Código 1. index.html

Lo que se observa en el código1 es el código completo de la página web con estructura HTML que soportará todo el desarrollo. Recordemos que el código fuente completo se puede encontrar en: <https://github.com/didiermorantes/AvanceMaestria>

Lo más representativo del código 1 es la sección presentada en el código 2.

<div id='lienzoPadre' class="col-md-6 fondo-lienzo">

</div><!-- fin col-md-4 -->

<div class="col-md-6 fondo-codigo">

<!-- inicio del codigo html. Div para alojar cajas de texto con código html -->

<div id="cajaDeCodigo">

</div>

</div>

Código 2. Sección para arrastrar y soltar y sección de código

Esta sección es importante porque es la que permitirá renderizar los elementos que se arrastren que está identificada con el color verde, y la que mostrará el código fuente de los elementos arrastrados, identificada con el color azul.

## Asignación de eventos onload

En el código 3 se observa la función que asignará eventos a los botones una vez se haya cargado la página. Esta función es invocada mediante el evento onload en el body del HTML.

function asignarEventos(){

// buscamos el contenedor de controles para permitir eliminar elementos que regresen de vuelta

let elContenedorControles = document.getElementById('contenedorControles');

elContenedorControles.addEventListener('dragover', permitirSoltar);

elContenedorControles.addEventListener('drop', soltarParaEliminar);

// hacemos que el contenedor de controles reaccione al evento tocar

elContenedorControles.addEventListener('touchstart', tocar);

// buscamos el botón de base en el DOM

let elBotonBase = document.getElementById('button0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elBotonBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elBotonBase.addEventListener('touchstart', tocar);

// buscamos la caja de texto de base en el DOM

let elTextoBase = document.getElementById('text0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elTextoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchmove para visualización mobile

elTextoBase.addEventListener('touchstart', tocar);

// buscamos el radio boton de base en el DOM

let elRadioBase = document.getElementById('radio0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elRadioBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchmove para visualización mobile

elRadioBase.addEventListener('touchstart', tocar);

// buscamos el label de base en el DOM

let elLabelBase = document.getElementById('label0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elLabelBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchmove para visualización mobile

elLabelBase.addEventListener('touchstart', tocar);

// buscamos el checkbox de base en el DOM

let elCheckBase = document.getElementById('checkbox0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elCheckBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchmove para visualización mobile

elCheckBase.addEventListener('touchstart', tocar);

// buscamos el submit de base en el DOM

let elSubmitBase = document.getElementById('submit0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elSubmitBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchmove para visualización mobile

elSubmitBase.addEventListener('touchstart', tocar);

// buscamos el independiente de base en el DOM

let elIndependienteBase = document.getElementById('independiente0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elIndependienteBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchmove para visualización mobile

elIndependienteBase.addEventListener('touchstart', tocar);

// buscamos el boton de 12x12

let elBoton12x12 = document.getElementById('btn12x12');

elBoton12x12.addEventListener('click', generarLienzo12x12);

// buscamos el boton de 2x12

let elBoton2x12 = document.getElementById('btn2x12');

elBoton2x12.addEventListener('click', generarLienzo2x12);

// buscamos el boton de 3x12

let elBoton3x12 = document.getElementById('btn3x12');

elBoton3x12.addEventListener('click', generarLienzo3x12);

// invocamos la función de ocultar controles. No se muestran hasta que se seleccione un layout

deshabilitarSeccionControles();

}

Código 3. Función para asignar eventos

Lo más representativo del código 3 se puede resumir en el código 4, donde se observa cómo se asigna el evento dragstart y el evento touchstart para hacer que los elementos o controles para construir formularios respondan al evento arrastrar, o al evento tocar si es que se están trabajando con dispositivos móviles. Estos eventos llamarán a las respectivas funciones de arrastrar y tocar que harán que el elemento se pueda desplazar a un contenedor de destino en el lienzo de trabajo identificado de color verde.

// buscamos el botón de base en el DOM

let elBotonBase = document.getElementById('button0');

// el evento dragstart tendrá asociado la función de callback de arrastrar

elBotonBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elBotonBase.addEventListener('touchstart', tocar);

Código 4. Asignación de eventos a los controles de los formularios

Hablando del lienzo de trabajo, en el código 3 también encontramos otra sección interesante y se muestra en el código 5.

// buscamos el boton de 3x12

let elBoton3x12 = document.getElementById('btn3x12');

elBoton3x12.addEventListener('click', generarLienzo3x12);

Código 5. Asignación de eventos para generar el lienzo de trabajo

## Funciones para generar lienzos de trabajo

En el código 5 se puede observar que cuando se presiona uno de los botones para generar un lienzo de trabajo, en el caso del código 5 el botón de 3columnasx12filas, se invoca una función denominada generarLienzo3x12(), que es la que poseerá toda la lógica para generar un área para arrastrar y soltar botones de 3 columnas por 12 filas. En el código 6 se muestran las funciones para generar lienzos de trabajo, con los respectivos elementos html que serán renderizados.

function generarLienzo12x12(){

// console.log(lienzo12x12);

// Borramos todos los arreglos para que comience de nuevo el conteo de elementos

// que sea lo primero para que luego se setee el limite

borrarTodosLosArreglos();

// establecemos el límite de elementos a arrastrar

limiteElementos = 144;

// console.log('generarLienzo12x12 limiteElementos: ', limiteElementos);

// mostramos la seccion de controles

habilitarSeccionControles();

// buscamos el lienzo padre en el DOM

let elLienzoPadre = document.getElementById('lienzoPadre');

// enviamos al lienzo padre la variable global con el html del lienzo 12x12

elLienzoPadre.innerHTML = lienzo12x12;

//Asignamos los eventos para el lienzo, de manera que se pueda arrastrar y soltar sobre ellos

// Se debe realizar dentro de este método, porque si lo hacemos en el onload, al no existir los elementos, no se asignará nada y no servirán para arrastrar y soltar

//buscamos los doce contenedores en el DOM donde se puede arrastrar el elemento

let arregloContenedores = document.getElementsByClassName('contenedoresArrastraSuelta');

// recorremos los doce contenedores y les asignamos los eventos dragover y drop

// el evento dragover tendrá asociado la función de callback de permitirSoltar

// el evento drop tendrá asociado la función de callback de soltar

for(let i=0; i<arregloContenedores.length;i++){

arregloContenedores[i].addEventListener('dragover', permitirSoltar);

arregloContenedores[i].addEventListener('drop', soltar);

// evento para tocar en version mobile para cada div generado dinamicamente

arregloContenedores[i].addEventListener('touchstart', tocar);

/\* No funciona bien cambiar colores de los div en los eventos mouseover y mouseout\*/

arregloContenedores[i].addEventListener('dragenter', dentroContenedor);

arregloContenedores[i].addEventListener('dragleave', fueraContenedor);

}

}

function generarLienzo2x12(){

// console.log(lienzo2x12);

// Borramos todos los arreglos para que comience de nuevo el conteo de elementos

// que sea lo primero para que luego se setee el limite

borrarTodosLosArreglos();

// establecemos el límite de elementos a arrastrar

limiteElementos = 24;

// console.log('generarLienzo2x12 limiteElementos: ', limiteElementos);

// mostramos la seccion de controles

habilitarSeccionControles();

// buscamos el lienzo padre en el DOM

let elLienzoPadre = document.getElementById('lienzoPadre');

// enviamos al lienzo padre la variable global con el html del lienzo 2x12de

elLienzoPadre.innerHTML = lienzo2x12;

//Asignamos los eventos para el lienzo, de manera que se pueda arrastrar y soltar sobre ellos

// Se debe realizar dentro de este método, porque si lo hacemos en el onload, al no existir los elementos, no se asignará nada y no servirán para arrastrar y soltar

//buscamos los doce contenedores en el DOM donde se puede arrastrar el elemento

let arregloContenedores = document.getElementsByClassName('contenedoresArrastraSuelta');

// recorremos los doce contenedores y les asignamos los eventos dragover y drop

// el evento dragover tendrá asociado la función de callback de permitirSoltar

// el evento drop tendrá asociado la función de callback de soltar

for(let i=0; i<arregloContenedores.length;i++){

arregloContenedores[i].addEventListener('dragover', permitirSoltar);

arregloContenedores[i].addEventListener('drop', soltar);

// evento para tocar en version mobile para cada div generado dinamicamente

arregloContenedores[i].addEventListener('touchstart', tocar);

/\* No funciona bien cambiar colores de los div en los eventos mouseover y mouseout\*/

arregloContenedores[i].addEventListener('dragenter', dentroContenedor);

arregloContenedores[i].addEventListener('dragleave', fueraContenedor);

}

}

function generarLienzo3x12(){

// console.log(lienzo3x12);

// Borramos todos los arreglos para que comience de nuevo el conteo de elementos

// que sea lo primero para que luego se setee el limite

borrarTodosLosArreglos();

// establecemos el límite de elementos a arrastrar

limiteElementos = 36;

// console.log('generarLienzo3x12 limiteElementos: ', limiteElementos);

// mostramos la seccion de controles

habilitarSeccionControles();

// buscamos el lienzo padre en el DOM

let elLienzoPadre = document.getElementById('lienzoPadre');

// enviamos al lienzo padre la variable global con el html del lienzo 2x12de

elLienzoPadre.innerHTML = lienzo3x12;

//Asignamos los eventos para el lienzo, de manera que se pueda arrastrar y soltar sobre ellos

// Se debe realizar dentro de este método, porque si lo hacemos en el onload, al no existir los elementos, no se asignará nada y no servirán para arrastrar y soltar

//buscamos los doce contenedores en el DOM donde se puede arrastrar el elemento

let arregloContenedores = document.getElementsByClassName('contenedoresArrastraSuelta');

// recorremos los doce contenedores y les asignamos los eventos dragover y drop

// el evento dragover tendrá asociado la función de callback de permitirSoltar

// el evento drop tendrá asociado la función de callback de soltar

for(let i=0; i<arregloContenedores.length;i++){

arregloContenedores[i].addEventListener('dragover', permitirSoltar);

arregloContenedores[i].addEventListener('drop', soltar);

// evento para tocar en version mobile para cada div generado dinamicamente

arregloContenedores[i].addEventListener('touchstart', tocar);

/\* No funciona bien cambiar colores de los div en los eventos mouseover y mouseout\*/

arregloContenedores[i].addEventListener('dragenter', dentroContenedor);

arregloContenedores[i].addEventListener('dragleave', fueraContenedor);

}

}

Código 6. Funciones para generar lienzos de trabajo

Lo más representativo del código 6 son las instrucciones de buscar en el DOM del index.html el lienzoPadre, que es uno de los elementos importantes identificados en el código 2, y cómo se le asignar a ese lienzo padre un valor determinado por las variables que poseen la estructura HTML del área de trabajo. En el código 7 se muestra lo más representativo del código 6, que asigna valores a los elementos del lienzo padre.

// buscamos el lienzo padre en el DOM

let elLienzoPadre = document.getElementById('lienzoPadre');

// enviamos al lienzo padre la variable global con el html del lienzo 2x12de

elLienzoPadre.innerHTML = lienzo3x12;

Código 7. Asignación de Valores al lienzo padre

## Variables con el HTML de los lienzos de trabajo

En el código 8 observamos el contenido de las variables que se asignan al lienzo padre, dependiendo del botón de layout que se haya presionado.

var lienzo12x12 = `

<!-- Contenedor de 12 columnas por 12 filas para arrastrar elementos-->

<div class="container">

<!-- fila 1-->

<div class="row">

<div id="f1c1" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f1c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c4" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f1c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 2-->

<div class="row">

<div id="f2c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c2" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f2c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c4" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f2c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 3-->

<div class="row">

<div id="f3c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c3" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f3c4" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f3c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 4-->

<div class="row">

<div id="f4c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f4c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f4c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 5-->

<div class="row">

<div id="f5c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f5c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f5c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 6-->

<div class="row">

<div id="f6c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f6c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f6c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 7-->

<div class="row">

<div id="f7c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f7c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f7c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 8-->

<div class="row">

<div id="f8c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f8c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f8c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 9-->

<div class="row">

<div id="f9c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f9c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c19" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f9c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 10-->

<div class="row">

<div id="f10c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f10c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f10c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 11-->

<div class="row">

<div id="f11c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f11c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f11c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 12-->

<div class="row">

<div id="f12c1" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c2" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c3" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c4" class="col-md-1 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f12c5" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c6" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c7" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c8" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c9" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c10" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c11" class="col-md-1 contenedoresArrastraSuelta">

</div>

<div id="f12c12" class="col-md-1 contenedoresArrastraSuelta">

</div>

</div>

</div><!-- fin container-->

`;

var lienzo2x12 = `

<!-- Contenedor de 2 columnas por 12 filas para arrastrar elementos-->

<div class="container">

<!-- fila 1-->

<div class="row">

<div id="f1c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f1c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 2-->

<div class="row">

<div id="f2c1" class="col-md-6 contenedoresArrastraSuelta">

</div>

<div id="f2c2" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

</div>

<!-- fila 3-->

<div class="row">

<div id="f3c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f3c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 4-->

<div class="row">

<div id="f4c1" class="col-md-6 contenedoresArrastraSuelta">

</div>

<div id="f4c2" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

</div>

<!-- fila 5-->

<div class="row">

<div id="f5c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f5c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 6-->

<div class="row">

<div id="f6c1" class="col-md-6 contenedoresArrastraSuelta">

</div>

<div id="f6c2" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

</div>

<!-- fila 7-->

<div class="row">

<div id="f7c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f7c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 8-->

<div class="row">

<div id="f8c1" class="col-md-6 contenedoresArrastraSuelta">

</div>

<div id="f8c2" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

</div>

<!-- fila 9-->

<div class="row">

<div id="f9c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f9c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 10-->

<div class="row">

<div id="f10c1" class="col-md-6 contenedoresArrastraSuelta">

</div>

<div id="f10c2" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

</div>

<!-- fila 11-->

<div class="row">

<div id="f11c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f11c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 12-->

<div class="row">

<div id="f12c1" class="col-md-6 contenedoresArrastraSuelta">

</div>

<div id="f12c2" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

</div>

</div><!-- fin container-->

`;

var lienzo3x12 = `

<!-- Contenedor de 3 columnas por 12 filas para arrastrar elementos-->

<div class="container">

<!-- fila 1-->

<div class="row">

<div id="f1c1" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f1c2" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f1c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 2-->

<div class="row">

<div id="f2c1" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f2c2" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f2c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 3-->

<div class="row">

<div id="f3c1" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f3c2" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f3c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 4-->

<div class="row">

<div id="f4c1" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f4c2" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f4c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 5-->

<div class="row">

<div id="f5c1" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f5c2" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f5c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 6-->

<div class="row">

<div id="f6c1" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f6c2" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f6c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 7-->

<div class="row">

<div id="f7c1" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f7c2" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f7c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 8-->

<div class="row">

<div id="f8c1" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f8c2" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f8c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 9-->

<div class="row">

<div id="f9c1" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f9c2" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f9c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 10-->

<div class="row">

<div id="f10c1" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f10c2" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f10c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 11-->

<div class="row">

<div id="f11c1" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f11c2" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f11c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

<!-- fila 12-->

<div class="row">

<div id="f12c1" class="col-md-4 contenedoresArrastraSuelta">

</div>

<div id="f12c2" class="col-md-4 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f12c3" class="col-md-4 contenedoresArrastraSuelta">

</div>

</div>

</div><!-- fin container-->

`;

Código 8. Variables con los valores que se asignarán al lienzo padre

El código 8 parece bastante voluminoso, sobre todo el que genera un lienzo de 12columnasx12 filas, debido a que se tiene que crear cada celda mediante código HTML. Sin embargo, analizando el lienzo de 2columnasx12filas, podemos encontrar lo más importante del código 8 y que es visualizado en el código 9.

<!-- fila 1-->

<div class="row">

<div id="f1c1" class="col-md-6 contenedoresArrastraSuelta">

&nbsp;

</div>

<div id="f1c2" class="col-md-6 contenedoresArrastraSuelta">

</div>

</div>

Código 9. Asignación de id especificos que representan filas y columnas

En el código 9 se puede observar que cada div o contenedor de división posee un id que consiste en una letra, un número, una letra, y un número. Esa codificación representa la fila, el valor de la fila, la columna, el valor de la columna. De esa manera es posible identificar cada contenedor de división y luego asignarle sus respectivos comportamientos cuando ocurra un evento.

## Funcionas relacionadas con la operación arrastrar

A continuación, se relacionan las porciones de código asociadas con las funcionalidades de arrastrar y soltar

function arrastrar(ev) {

// identificamos el id del elemento que está siendo arrastrado

// El método DataTransfer.setData() establece los datos de arrastre de la operación de arrastre en el tipo y datos especificados

// Se agrega el id del elemento arrastrado al objeto DataTransfer

// decidimos si el evento es arrastrar o por touch

if(ev.type=='touchmove'){

// console.log(ev);

// no existe la propiedad dataTransfer en los eventos touch

// extraemos los datos directamente de target

// console.log(arrastrar ev.target.id);

}

else if(ev.type == 'dragstart'){

// El objeto DataTransfer se utiliza para contener los datos que se arrastran durante una operación de arrastrar y soltar

ev.dataTransfer.setData("text/plain", ev.target.id);

// el idAnterior que está produciendo el evento de arrastrar

let idAnterior = ev.target.id;

// console.log('arrastrar idAnterior: ', idAnterior);

// extraemos el ultimo elemento del idAnterior, pues viene en formato letraNumero

let oldIdString = idAnterior.slice(-1);

// si el número es diferente de cero, almacenamos el evento en ultimoTargetId, de lo contrario no porque fue arrastrado de la caja de controles

if(oldIdString != 0){

// almacenamos el ultimoTargetId por si editamos un elemento y queda en blanco

ultimoTargetId = ev.target.id;

}

}

// console.log(ev.target.id);

// console.log(ev.dataTransfer);

}

Código 10. Función arrastrar()

En el código 10 lo más relevante es lo que se presenta en el código 11, donde encontramos la instrucción ev.dataTransfer.setData(), que sirve para almacenar información relevante a la operación de arrastrar(drag)

// El objeto DataTransfer se utiliza para contener los datos que se arrastran durante una operación de arrastrar y soltar

ev.dataTransfer.setData("text/plain", ev.target.id);

Código 11. Instrucción que almacena elementos de la operación arrastrar

## Funciones relacionadas con la operación soltar

En el código 12 encontramos la función permitirSoltar() que cancela eventos de recargar la página con la instrucción ev.preventDefault().

function permitirSoltar(ev) {

ev.preventDefault();

// console.log(ev);

}

Código 12. Función permitirSoltar()

function soltar(ev) {

// console.log(ev);

// console.log(ev.srcElement.firstElementChild.outerHTML);

// console.log(ev.srcElement.lastChild);

ev.preventDefault();

// borramos el borde de ubicación cuando se suelte el elemento

this.style.borderStyle = 'none';

try{

// si la cantidad de elementos es menor que el límite, se pueden seguir arrastrando más elementos

if(cantidadElementos < limiteElementos ){

// en el manejador soltar obtenemos el id del elemento que está siendo arrastrado y lo usamos para moverlo al contenedor destino

// data es el elemento que está siendo arrastrado

let data = ev.dataTransfer.getData("text");

// console.log('soltar data: ',data);

// console.log('soltar ev.target: ',ev.target);

// console.log('soltar ultimaTargeId: ', ultimoTargetId);

// evaluamos si data esta vacio. Si esta vacio es porque se ha editado el elemento

if(data == ''){

// console.log('soltar data esta vacio');

// puede que data este vacío por haber editado el elemento, así que lo reconstruimos

// El objeto DataTransfer no se puede modificar asi como en la siguiente linea

// ev.dataTransfer.setData("text/plain", ev.target.id);

// hacemos data igual al valor almacenado en el ultimoTargetId

data = ultimoTargetId;

// agregamos (appendChild) al target(destino) el elemento identificado mediante su id y referenciado en la variable data, que al estar vacia, enviamos el ultimo targetId

ev.target.appendChild(document.getElementById(data));

}

else{

// agregamos (appendChild) al target(destino) el elemento identificado mediante su id y referenciado en la variable data

ev.target.appendChild(document.getElementById(data));

}

// identificamos el destino (target), especificamente su id

let idDivDestino = ev.target.id;

// console.log('soltar idDivDestino: ',idDivDestino);

// invocamos al método generarNuevoId para que decida si se genera nuevo id y se reubique el elemento base en la sección de controles

generarNuevoId(data, idDivDestino);

}else{

// si la cantidad de elementos es mayor al limite mostramos alerta

alert('No puede arrastrar más elementos. Ha superado el límite');

}

}

catch(error){

console.log('error en la función arrastrar atrapado: ',error); // pasa el objeto de la excepción al manejador de errores

}

}

Código 13. Función soltar()

En el código 13 encontramos la función soltar() cuya línea más importante se ilustra en el código 14, y es la que permite mover a la celda de destino, representado mediante el objeto ev.target, el control u elemento que se esté moviendo y que está representado mediante el objeto data.

// agregamos (appendChild) al target(destino) el elemento identificado mediante su id y referenciado en la variable data

ev.target.appendChild(document.getElementById(data));

Código 14. Código que permite mover un control a una celda de destino

## Funciones relacionadas con generar nuevos identificadores a los elementos creados

La función soltar() que trabaja con el sistema de arrastra y suelte y la función tocar() que trabaja con el sistema de tocar la pantalla de un móvil invocan una función denominada generarNuevoId() que prácticamente es el corazón del desarrollo, pues genera un nuevo identificador para el control que se está alojando en el lienzo de trabajo, y así permitir que puedan ir generando dinámicamente nuevos elementos en el lienzo de trabajo cada vez que se arrastra y suelta un elemento del área de controles. En el código 15 se ilustra el código de la función generarNuevoId()

function generarNuevoId(idAnterior, idDivDestino){

// console.log('generarNuevoId: data idAnterior',idAnterior);

// extraemos el nombre del id sin contar el número

let nombreId = idAnterior.slice(0,-1);

// console.log(nombreId);

// extraemos el ultimo elemento del idAnterior, pues viene en formato letraNumero

let oldIdString = idAnterior.slice(-1);

// convertimos el ultimo elemento en número

let oldIdInt = Number(oldIdString);

// console.log('generarNuevoId oldIdInt: ', oldIdInt);

// variable para el nuevo elemento arrastrado

let elNuevoElemento = '';

// nuevoNombreId para el botón arrastrado

let nuevoNombreId = '';

// evaluamos el nombreId, pues nos indicará si es un botón, una caja de texto , o algún elemento funcional modelado

if(nombreId == 'button'){

// si el oldIdInt es diferente de cero, significa que es un botón que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un botón que se arrastró de la sección de controles, y debemos registrar dicho boton en el arreglo global de botones

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloBotones = arrBotones.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrBotones[arrBotones.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloBotones = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -boton0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloBotones;

// console.log(newIdInt);

// nuevoNombreId para el botón arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del botón de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrBotones.push(nuevoNombreId);

// console.log(arrBotones);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if boton

else if(nombreId == 'text'){

// si el oldIdInt es diferente de cero, significa que es un texto que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un texto que se arrastró de la sección de controles, y debemos registrar dicho texto en el arreglo global de texto

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloTexto = arrTexto.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrTexto[arrTexto.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloTexto = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -texto0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloTexto;

// console.log(newIdInt);

// nuevoNombreId para el texto arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del texto de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrTexto.push(nuevoNombreId);

// console.log(arrTexto);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if texto

else if(nombreId == 'radio'){

// si el oldIdInt es diferente de cero, significa que es un radio que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un radio que se arrastró de la sección de controles, y debemos registrar dicho radio en el arreglo global de radio botones

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloRadio = arrRadio.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrRadio[arrRadio.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloRadio = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -radio0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloRadio;

// console.log(newIdInt);

// nuevoNombreId para el radio arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del radio de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrRadio.push(nuevoNombreId);

// console.log(arrRadio);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if radio

else if(nombreId == 'label'){

// si el oldIdInt es diferente de cero, significa que es un label que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un label que se arrastró de la sección de controles, y debemos registrar dicho label en el arreglo global de label

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloLabel = arrLabel.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrLabel[arrLabel.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloLabel = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -label0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloLabel;

// console.log(newIdInt);

// nuevoNombreId para el label arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del label de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrLabel.push(nuevoNombreId);

// console.log(arrLabel);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if label

else if(nombreId == 'checkbox'){

// si el oldIdInt es diferente de cero, significa que es un checkbox que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un checkbox que se arrastró de la sección de controles, y debemos registrar dicho checkbox en el arreglo global de checkbox

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloCheckbox = arrCheckbox.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrCheckbox[arrCheckbox.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloCheckbox = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -checkbox0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloCheckbox;

// console.log(newIdInt);

// nuevoNombreId para el checkbox arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del checkbox de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrCheckbox.push(nuevoNombreId);

// console.log(arrCheckbox);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if checkbox

else if(nombreId == 'submit'){

// si el oldIdInt es diferente de cero, significa que es un submti que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un submit que se arrastró de la sección de controles, y debemos registrar dicho submit en el arreglo global de submit

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloSubmit = arrSubmit.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrSubmit[arrSubmit.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloSubmit = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -submit0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloSubmit;

// console.log(newIdInt);

// nuevoNombreId para el submit arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del submit de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrSubmit.push(nuevoNombreId);

// console.log(arrSubmit);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if submit

else if(nombreId == 'independiente'){

// si el oldIdInt es diferente de cero, significa que es un submti que ya existe en el lienzo, y no debemos incrementar ningún id

// si el oldIdInt es cero, significa que es un submit que se arrastró de la sección de controles, y debemos registrar dicho submit en el arreglo global de submit

if(oldIdInt==0){

//validamos el arreglo de elementos para incrementar en uno

// se diseña la lógica para que la cantidad de elementos almacenados en el array coincida con la cantidad de elementos en pantalla

// let longitudArregloSubmit = arrSubmit.length;

// se diseña la lógica para extraer el último elemento del arreglo, extraer el último caracter y luego incrementarlo en uno

let ultimoElementoArreglo = arrIndependiente[arrIndependiente.length - 1 ];

// extraemos el ultimo caracter del ultimoElementoArreglo, pues viene en formato letraNumero

let oldIdUltimoElementoString = ultimoElementoArreglo.slice(-1);

// convertimos el ultimo elemento en número

let oldIdUltimoElementoInt = Number(oldIdUltimoElementoString);

let longitudArregloSubmit = oldIdUltimoElementoInt+1;

// Generamos un nuevo id si se arrastra de la sección de controles, y ya hay elementos del mismo tipo en el lienzo

// le sumamos 1 al ultimo elemento (el numero) del id que haya en el arreglo global de elementos para generar un nuevo id

// con la propiedad .length del arreglo ya trae incrementado en uno la cantidad de elementos del arreglo, pues ya existe un primer elemento en el arreglo -submit0- no hay que hacer la operación matemática, solamente asignar el valor que entregue la propiedad .length

newIdInt = longitudArregloSubmit;

// console.log(newIdInt);

// nuevoNombreId para el submit arrastrado, que será el nombre anterior sin número concatenado con el nuevo número generado

nuevoNombreId = nombreId+newIdInt;

// console.log(nuevoNombreId);

// Buscamos el elemento recien arrastrado

elNuevoElemento = document.getElementById(idAnterior);

// console.log('elNuevoElemento', elNuevoElemento);

// le cambiamos el id por el nuevo id generado, es decir, el número incrementado en uno para diferenciarlo del submit de base

elNuevoElemento.id = nuevoNombreId;

// console.log('elNuevoElemento', elNuevoElemento);

// registramos el nuevo nombre del elemento en el arreglo global de elementos correspondiente

arrIndependiente.push(nuevoNombreId);

// console.log(arrSubmit);

// El orden para regenerar el elemento importa. Si regenero primero antes de cambiar el id del elemento recién arrastrado, se identificará en el DOM el elemento recién generado en la sección de controles

// si el ultimo elemento es cero, significa que es la primera vez que se arrastra de la sección de controles

// regeneramos el elemento base si es la primera vez que se arrastra

regenerarElementoBase(idAnterior);

}// fin oldInt

}// fin if independiente

// si no es un nuevo elemento (oldInt==0) no enviamos nada. No invocamos a enviarCodigoHTML

if(oldIdInt==0){

// invocamos al método que enviará el código HTML del nuevo elemento a la sección de código, junto con su id

enviarCodigoHTML(elNuevoElemento,nuevoNombreId, idDivDestino );

// si el oldIdInt es cero, actualizamos la referencia de ultimoTargetId actualizado con el nuevoNombreId

ultimoTargetId = nuevoNombreId;

// console.log('generarNuevoId ultimoTargetId: ',ultimoTargetId);

//incrementamos en uno la cantidad de elementos

cantidadElementos+=1;

// console.log('generarNuevoId cantidadElementos: ', cantidadElementos);

// invocamos la funcion que actualiza el span de la cantidad de elementos

actualizarSpanCantidadElementos(cantidadElementos);

}

else{

// si el ultimo numero del id no es cero, hay que actualizar la caja de código

// console.log('se movio elemento. Posiblemente haya cambiado codigo en la caja de codigo');

// la caja de codigo mantiene la referencia del primer lugar donde se movio y se comenzo a editar codigo

actualizarIdDivDestino(idAnterior, idDivDestino);

}

}// fin function

Código 15. Función generarNuevoId()

Como se ilustra en el código 15, la función generarNuevoId() identifica cuál elemento de la sección de controles se está arrastrando y soltando, para poder generar un nuevo elemento, y llevar el conteo en una serie de arreglos que servirán para llevar el registro de los elementos que se están alojando en la zona de trabajo. En el código 16 se observan los arreglos utilizados para llevar el registro de los elementos arrastrados al lienzo de trabajo.

// arreglo global para llevar registro de botones

var arrBotones = ['button0'];

// arreglo global para llevar registro de cajas de texto

var arrTexto = ['text0'];

// arreglo global para llevar registro de radio botones

var arrRadio = ['radio0'];

// arreglo global para llevar registro de labels

var arrLabel = ['label0'];

// arreglo global para llevar registro de checkbox

var arrCheckbox = ['checkbox0'];

// arreglo global para llevar registro de submit

var arrSubmit = ['submit0'];

// arreglo global para llevar registro de independiente

var arrIndependiente = ['independiente0'];

// variable para guardar el ultimoTargetId después de haber editado un elemento

var ultimoTargetId = '';

// variable para determinar limite de elementos a arrastrar

var limiteElementos = 0;

// variable para llevar el conteo de elementos en el lienzo

var cantidadElementos = 0;

// variable para guardar los elementos que se han tocado mobile

var elementoTouch1 = '';

// variable para guardar los elementos que se han tocado mobile

var elementoTouch2 = '';

// variable para llevar los conteos de toques

var contadorTouch=0;

Código 16. Arreglos para llevar el conteo de los elementos arrastrados/tocados

## Funciones relacionadas con regenerar nuevos elementos en la sección de controles

Una función igual de importante a la de generarNuevoId() se ilustra en el código 17. Se trata una función que regenera el elemento que se está arrastrando de la zona de controles, y que vuelve y le asigna el id original que tenía en la zona de controles, pues realmente el sistema drag and drop lo que hace es mover un elemento de una posición inicial a una posición final, y entonces lo que hacemos con la funcipon regenerarElementoBase() es crear dinámicamente un nuevo elemento en la sección de controles.

function regenerarElementoBase(idAnterior){

// extraemos el nombre del id sin contar el número, de esa manera sabremos que tipo de elemento es

let nombreId = idAnterior.slice(0,-1);

// identificamos en el DOM el contenedor de controles

let elContenedorControles = document.getElementById('contenedorControles');

if(nombreId == 'button'){

// creamos un nuevo elemento en el DOM de tipo input

let elElementoBase = document.createElement('input');

// asignamos al nuevo elemento un tipo botón

elElementoBase.setAttribute('type', 'button');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualizacion mobile

elElementoBase.addEventListener('touchstart', tocar);

// le agregamos un cero al nombre del elemento base para indicar que es el primero

elElementoBase.id=nombreId+0;

// le asignamos al nuevo elemento el valor boton

elElementoBase.value='button';

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

else if(nombreId == 'text'){

// creamos un nuevo elemento en el DOM de tipo input

let elElementoBase = document.createElement('input');

// asignamos al nuevo elemento un tipo texto

elElementoBase.setAttribute('type', 'text');

// asignamos al nuevo elemento un tipo texto

elElementoBase.setAttribute('size', '8');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elElementoBase.addEventListener('touchstart', tocar);

// le agregamos un cero al nombre del elemento base para indicar que es el primero

elElementoBase.id=nombreId+0;

// le asignamos al nuevo elemento el valor texto

elElementoBase.value='text';

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

else if(nombreId == 'radio'){

// creamos un nuevo elemento en el DOM de tipo input

let elElementoBase = document.createElement('input');

// asignamos al nuevo elemento un tipo radio

elElementoBase.setAttribute('type', 'radio');

// asignamos al nuevo elemento un name que permite el agrupamiento de los radio botones

elElementoBase.setAttribute('name', 'grupoRadio');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elElementoBase.addEventListener('touchstart', tocar);

// le agregamos un cero al nombre del elemento base para indicar que es el primero

elElementoBase.id=nombreId+0;

// le asignamos al nuevo elemento el valor radio

elElementoBase.value='radio';

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

else if(nombreId == 'label'){

// creamos un nuevo elemento en el DOM de tipo label

let elElementoBase = document.createElement('label');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elElementoBase.addEventListener('touchstart', tocar);

// le agregamos un cero al nombre del elemento base para indicar que es el primero

elElementoBase.id=nombreId+0;

// le asignamos al nuevo elemento el valor label

elElementoBase.value='label';

// le asignamos al nuevo elemento el texto de label

elElementoBase.innerText='label';

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

else if(nombreId == 'checkbox'){

// creamos un nuevo elemento en el DOM de tipo input

let elElementoBase = document.createElement('input');

// asignamos al nuevo elemento un tipo checkbox

elElementoBase.setAttribute('type', 'checkbox');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elElementoBase.addEventListener('touchstart', tocar);

// le agregamos un cero al nombre del elemento base para indicar que es el primero

let newId = nombreId+0;

elElementoBase.id=newId;

// le asignamos al nuevo elemento el valor radio

elElementoBase.value='radio';

// asignamos al nuevo elemento el mismo name que el id para los checkbox

elElementoBase.setAttribute('name', newId);

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

else if(nombreId == 'submit'){

// creamos un nuevo elemento en el DOM de tipo input

let elElementoBase = document.createElement('input');

// asignamos al nuevo elemento un tipo submit

elElementoBase.setAttribute('type', 'submit');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elElementoBase.addEventListener('touchstart', tocar)

// le agregamos un cero al nombre del elemento base para indicar que es el primero

elElementoBase.id=nombreId+0;

// le asignamos al nuevo elemento el valor submit

elElementoBase.value='submit';

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

else if(nombreId == 'independiente'){

// creamos un nuevo elemento en el DOM de tipo input

let elElementoBase = document.createElement('input');

// asignamos al nuevo elemento un tipo texto

elElementoBase.setAttribute('type', 'text');

// asignamos al nuevo elemento un tipo texto

elElementoBase.setAttribute('size', '6');

// asignamos al nuevo elemento la propiedad draggable en true

elElementoBase.setAttribute('draggable', 'true');

// asignamos al nuevo elemento el eventListener dragstart

elElementoBase.addEventListener('dragstart', arrastrar);

// asignamos al nuevo elemento el eventListener touchstart para visualización mobile

elElementoBase.addEventListener('touchstart', tocar);

// le agregamos un cero al nombre del elemento base para indicar que es el primero

elElementoBase.id=nombreId+0;

// le asignamos al nuevo elemento el valor texto

elElementoBase.value='modificame';

// añadimos al contenedor de controles el nuevo elemento recién creado con id 0, indicando que es un elemento de base

elContenedorControles.appendChild(elElementoBase);

}

}

Código 17. Función regenerarElementoBase()

Es importante aclarar en la función regenerarElementoBase() también crea dinámicamente los eventos que se asociarán al nuevo elemento, para que se comporte exactamente igual al elemento que fue arrastrado de la zona de controles.

## Funciones relacionadas con crear la sección de código de los elementos arrastrados

La función generarNuevoId() también invoca internamente a la función enviarCodigoHTML() cuyo trabajo es crear el código correspondiente del elemento arrastrado en la sección de código del lienzo de trabajo identificado de color azul.

// la funcion recibe el elemento que será editado, y su id, y el id del Div destino para modificarle su innerHTML

function enviarCodigoHTML(elemento, id, idDivDestino){

// console.log(elemento);

// console.log(typeof elemento);

// console.log('enviarCodigoHTML idDivDestino:', idDivDestino);

// Extraemos el HTML del elemento arrastrado con la propiedad outerHTML y la convertimos en String

let htmlExtraido = String(elemento.outerHTML);

// console.log('htmlExtraido',htmlExtraido);

// ubicamos la caja de código en el DOM

let laCajaDeCodigo = document.getElementById('cajaDeCodigo');

// identificamos el ancho de la pantalla para determinar el size de la caja que se usará para escribir código pensando en móviles

let anchoPantalla = window.innerWidth;

let sizeCajaCodigo = '';

if(anchoPantalla <= 768){

// es pantalla pequeña

sizeCajaCodigo = '30';

}

else{

// es pantalla grande

sizeCajaCodigo = '40';

}

// creamos una nueva caja de texto que poseerá el código recién creado

let nuevaCajaTexto = document.createElement('input');

// establecemos los atributos de la nueva caja de texto indicando que es de tipo texto

nuevaCajaTexto.setAttribute('type', 'text');

// establecemos los atributos de la nueva caja de texto para que tengan un tamaño definido de 40 caracteres

nuevaCajaTexto.setAttribute('size', sizeCajaCodigo);

// establecemos los atributos de la nueva caja de texto para que no tengan bordes aplicando una clase personalizda

nuevaCajaTexto.setAttribute('class', 'sin-bordes');

// generamos un id sintetico para la caja, de manera que se pueda ubicar en el DOM

let idSinteticoCajaEdicion = 'editar'+id;

// creamos una caja de span que identifique la posición del elemento html que se está arrastrando

let nuevoSpanPosicion = document.createElement('span');

let idSinteticoSpanPosicion = 'posicion'+id;

// asignamos el id sintetico al nuevo parrafo generado

nuevoSpanPosicion.id = idSinteticoSpanPosicion;

// asignamos un fontSize de 7 a la posicion

nuevoSpanPosicion.style.fontSize = '9px';

nuevoSpanPosicion.innerText = idDivDestino;

// asignamos el id sintetico a la nueva caja generada

nuevaCajaTexto.id = idSinteticoCajaEdicion;

// le asignamos un event listener para que resalte con un color azul claro la caja que se esta editando

nuevaCajaTexto.addEventListener('mouseover', resaltarCajaCodigo );

// le asignamos un event listener para que deje de resaltar la caja que se esta editando cuando se salga del foco

nuevaCajaTexto.addEventListener('mouseout', noResaltarCajaCodigo );

// hacemos que todas las cajas de texto reaccionen al evento onkeyup, es decir que reaccione cuando se escriba en las cajas pasando como argumentos el id del elemento que se va a editar y el id de la caja que edita

// cuando se requiere llamar una función de callback con parámetros, se llama una función anonima que internamente llama a la función de callback con parámetros

nuevaCajaTexto.addEventListener('keyup',function() { cambiarElemento(idDivDestino, idSinteticoCajaEdicion, id);} );

//asignamos como valor en la caja de código el html extraido

nuevaCajaTexto.value = htmlExtraido;

// creamos un nuevo salto de linea

let nuevoSalto = document.createElement('br');

// añadimos a la caja de codigo el nuevo elemento recién creado de span

laCajaDeCodigo.appendChild(nuevoSpanPosicion);

// añadimos a la caja de codigo el nuevo elemento recién creado

laCajaDeCodigo.appendChild(nuevaCajaTexto);

// añadimos a la caja de código el nuevo salto de linea

laCajaDeCodigo.appendChild(nuevoSalto);

}

Código 18. Función enviarCodigoHTML

En el código 18 la instrucción central es la que se muestra en el código 19, pues ilustra cómo se crea un nuevo elemento input en el DOM de tipo texto. Es esta caja de texto creada dinámicamente la que alojará el código de los elementos arrastrados en el lienzo de trabajo.

// creamos una nueva caja de texto que poseerá el código recién creado

let nuevaCajaTexto = document.createElement('input');

// establecemos los atributos de la nueva caja de texto indicando que es de tipo texto

nuevaCajaTexto.setAttribute('type', 'text');

Código 19. Creación de caja de texto para alojar el código de los elementos arrastrados

En el código 18 también se encuentra la lógica que extraerá el HTML del elemento arrastrado y que finalmente lo asignará a la caja de texto que alojará el código HTML del elemento arrastrado.

## Funciones para eliminar elementos

Finalmente presentamos las funciones relacionadas con eliminar elementos del lienzo de trabajo y de la caja de código. En el código 20 se observa la función soltarParaEliminar()

function soltarParaEliminar(ev) {

// console.log(ev);

// console.log(ev.srcElement.firstElementChild.outerHTML);

// console.log(ev.srcElement.lastChild);

ev.preventDefault();

// en el manejador soltar obtenemos el id del elemento que está siendo arrastrado y lo usamos para eliminarlo del DOM

let data = ev.dataTransfer.getData("text");

// console.log(data);

// extraemos el ultimo elemento del idAnterior, pues viene en formato letraNumero

let oldIdString = data.slice(-1);

// si el ultimo elemento del id es diferente de cero, eliminamos los elementos del lienzo, y de los arreglos

// de lo contrario no hacemos nada, pues si el ultimo elemento del id es cero, significa que es el elemento de base

// y no queremos eliminar el elemento de base

if(oldIdString != 0){

let elElementoAeliminar = document.getElementById(data);

// eliminamos(remove) del source(origen) el elemento identificado mediante su id y referenciado en la variable data

elElementoAeliminar.remove();

//eliminamos del arreglo correspondiente el elemento arrastrado

EliminarDelArreglo(data);

//eliminamos de la caja de codigo el elemento arrastrado

EliminarDeLaCajaCodigo(data);

}

}

Código 20. Soltar para Eliminar

Básicamente la función soltarParaEliminar() se invoca cuando se arrastra algún elemento del lienzo de trabajo nuevamente hacia la zona de controles. En ese momento eliminamos el elemento del DOM , de los arreglos que llevan el conteo y de la sección de cajas de código. Así mismo, se implementa la lógica para que los contadores resten un elemento al conteo que llevaban.

En el código 21 se ilustra la función eliminar del arreglo, que básicamente elimina cada elemento (botón, caja de texto, label, etc) de sus respectivos arreglos y poder llevar así un conteo actualizado de elementos que se encuentran en el lienzo de trabajo.

function EliminarDelArreglo(elementoAeliminar){

// console.log('Eliminar del Arreglo',elementoAeliminar);

// extraemos el nombre del id sin contar el número

let nombreId = elementoAeliminar.slice(0,-1);

// console.log(nombreId);

// extraemos el ultimo elemento del idAnterior, pues viene en formato letraNumero

let oldIdString = elementoAeliminar.slice(-1);

// convertimos el ultimo elemento en número

let oldIdInt = Number(oldIdString);

// evaluamos el nombreId, pues nos indicará si es un botón, una caja de texto , o algún elemento funcional modelado

if(nombreId == 'button'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrBotones.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrBotones.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrBotones);

}// fin if boton

else if(nombreId == 'text'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrTexto.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrTexto.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrTexto);

}// fin if texto

else if(nombreId == 'radio'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrRadio.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrRadio.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrRadio);

}// fin if radio

else if(nombreId == 'label'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrLabel.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrLabel.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrLabel);

}// fin if label

else if(nombreId == 'checkbox'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrCheckbox.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrCheckbox.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrCheckbox);

}// fin if checkbox

else if(nombreId == 'submit'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrSubmit.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrSubmit.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrSubmit);

}// fin if submit

else if(nombreId == 'independiente'){

// identificamos el indice del elemento a eliminar

let indiceAEliminar = arrIndependiente.indexOf(elementoAeliminar);

// eliminamos el elemento del arreglo, con el índice identificado

arrIndependiente.splice(indiceAEliminar, 1);

//mostramos el arreglo después de la eliminación

// console.log(arrSubmit);

}// fin if independiente

}// fin function

Código 21. funcion EliminarDelArreglo()

En el código 22 se ilustra la función que elimina la caja de código correspondiente al elemento que fue arrastrado fuera del lienzo de trabajo.

function EliminarDeLaCajaCodigo(elementoAeliminar){

// console.log('Eliminar del Arreglo',elementoAeliminar);

// identificamos la caja de código

// ubicamos la caja de código en el DOM

let laCajaDeCodigo = document.getElementById('cajaDeCodigo');

// construimos el id del elemento a eliminar anteponiendole la palabra editar al parámetro recibido

let idElementoAEliminar = 'editar'+ elementoAeliminar;

// identificamos el elemento a eliminar mediante el id construido

let elElementoAEliminar = document.getElementById(idElementoAEliminar);

// removemos de la caja de código el elemento hijo identificado con el id recien configurado

laCajaDeCodigo.removeChild(elElementoAEliminar);

// construimos el id del elemento a eliminar anteponiendole la palabra posicion al parámetro recibido

let idPosicionAEliminar = 'posicion'+ elementoAeliminar;

// identificamos el elemento de posicion a eliminar mediante el id construido

let elElementoPosicionAEliminar = document.getElementById(idPosicionAEliminar);

// removemos de la caja de código el elemento hijo identificado con el id recien configurado

laCajaDeCodigo.removeChild(elElementoPosicionAEliminar);

// Restamos uno de la variable global la cantidad de elementos en el lienzo

cantidadElementos-=1

// actualizar cantidad de elementos en el span. Se hace en este método porque es transversal a los eventos de arrastrar y tocar

actualizarSpanCantidadElementos(cantidadElementos);

}// fin function

Código 22. EliminarDeLaCajaDeCodigo()

Básicamente el código 22 ilustra que se identifica el id del elemento que se desea eliminar, y como la caja de código correspondiente posee un id compuesto por ese mismo nombre, más la palabra editar, entonces es factible eliminar la caja de código correspondiente inmediatamente después de haber eliminado el elemento del DOM.

Hay otras funciones accesorias que se pueden visualizar completamente en el repositorio público del código fuente, y que si las mencionáramos en este documento, ocuparían una mayor cantidad de páginas, por tanto, se invita al lector a consultar el código fuente el cual se encuentra suficientemente documentado mediante comentarios.

Por tanto, se invita al lector a revisar el código fuente desarrollado, cuyo repositorio se puede encontrar en: <https://github.com/didiermorantes/AvanceMaestria>

# Conclusiones

Se puede concluir que es posible construir el simulador propuesto en este trabajo de grado, utilizando el lenguaje de programación JAVASCRIPT, de manera que puede materializarse la solución planteada en la introducción de este trabajo ¿Será factible desarrollar un instrumento que facilite el proceso de edición y visualización de código html, de manera que el estudiante experimente simultáneamente la escritura y la percepción de la página web? Durante esta investigación era posible que no lograra construirse el simulador/entrenador, pues tenía que adaptarse a las situaciones planteadas en la introducción del trabajo, sin embargo, se pudo llegar a buen puerto y se pudo desarrollar una herramienta acorde a las necesidades del autor. Se esperaba que se pudiera arrastrar un control o elemento de formulario a un área de trabajo, y que se representara su respectivo código en una sección independiente del área de trabajo. Además, se esperaba que el código generado fuese interactivo, de manera que cuando el estudiante modificara dicho código, se viera su respectivo cambio o modificación en el elemento HTML manipulado.

Ya sería parte de otra investigación validar si este instrumento realmente facilita el proceso de aprendizaje del lenguaje de marcas HTML, y si se puede mejorar para convertirlo en una herramienta de enseñanza-aprendizaje que se pueda utilizar junto con un modelo pedagógico adecuado que coadyuve y fortalezca las habilidades y destrezas que se espera que alcancen los estudiantes del lenguaje de marcas HTML.

Como recomendaciones se podrían indicar las siguientes:

* Se puede continuar este proceso investigativo agregando otras secciones de edición de código, de manera que se pueda manipular la hojas de estilos CSS de manera independiente a la sección de edición del HTML, es decir, tener una representación similar a la hoja de edición de código HTML, pero para editar únicamente los estilos de un elemento HTML a través del lenguaje CSS.
* Se puede continuar este proceso investigativo agregando otras secciones de edición de código, de manera que se pueda manipular la funcionalidad Javascript de manera independiente a la sección de edición del HTML, es decir, tener una representación similar a la hoja de edición de código HTML, pero para editar únicamente funcionalidades Javascript con sus respectivos eventos.
* Se puede continuar este proceso investigativo validando si el simulador/entrenador desarrollado realmente facilita el proceso de aprendizaje del lenguaje de marcas HTML, y si realmente el modelo pedagógico propuesto(socio constructivismo, aprendizaje experiencial, aprendizaje cooperativo, aprendizaje basado en problemas) coadyuva a desarrollar conocimiento mediante el uso del entrenador.

Por tanto, se puede decir que este proceso de investigación se puede continuar desde la parte del desarrollo de software.