Unidad 01: Creando interacción con JavaScript

- 0. Opening: Haciendo tu sitio web interactivo
- 1. Document Object Model (DOM)
- 2. Browser Object Model (BOM)
- 3. Modificando el DOM
- 4. Eventos
- 5. Atributos data
- 6. Casos prácticos (videos)
- 7. Taller HSE: Presentaciones personales (parte 1)
- 8. Quiz #1
- 9. Clase de Conceptos
- 10. Clase Práctica
- 11. Retos
- **12. Solucionario Retos de Código**
- **13. Taller HSE: Presentaciones personales (parte 2)**
- **14. Quiz #2: _requizzing_**
- **15. Auto-evaluación**
- **16. Closing: Haciendo tu sitio web interactivo**

Retos

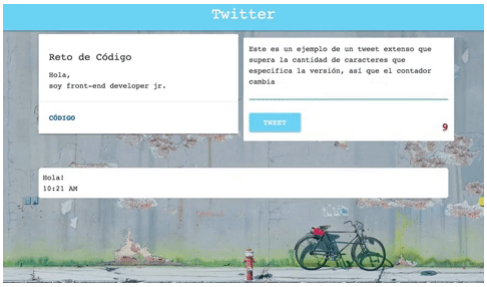
1200min

Tipo: práctica Formato: self-paced`

1. Twitter

En este reto, vamos a replicar el newsfeed de Twitter, para ello vamos a seguir un flujo de versiones que te guiarán para enfocarte de una manera más eficiente :)

Puedes adaptar el diseño como mejor te parezca, *pero no tardes mucho definiendo los colores o fuentes. Por último puedes replicar el diseño de la página oficial* :). Aquí un ejemplo:



Versión 0.0.1

1. Diseñar un formulario que permita ingresar un texto y un botón para "twittear".
2. Agregar un evento de click al botón o de submit al formulario.
3. En el evento, obtener el texto.
4. Agregar el texto al HTML.

Versión 0.0.2

1. No ingresar texto vacío (deshabilitar el botón de "twittear").
2. Contar la cantidad de caracteres de forma regresiva.

Versión 0.0.3

1. Si pasa los 140 caracteres, deshabilitar el botón.
2. Si pasa los 120 caracteres, mostrar el contador con OTRO color.
3. Si pasa los 130 caracteres, mostrar el contador con OTRO color.
4. Si pasa los 140 caracteres, mostrar el contador en negativo.

Versión 0.0.4

1. Al presionar enter(`/n`) que crezca el textarea de acuerdo al tamaño del texto.

Versión 0.0.5 (Extra)

1. Si la cantidad de caracteres ingresados (sin dar un enter), supera al tamaño del textarea por defecto, debe de agregarse una línea más para que no aparezca el scroll. (Si en caso aplica)

Versión 0.0.6 (Extra)

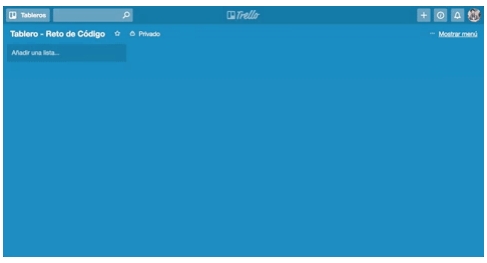
1. Agregar la hora en que se publicó el tweet. En el formato de 24 horas: `hh:mm` .

> Nota: Para dar formato a la fecha y hora, puedes crear tu propia función o usar una librería como `moment.js` .

2. Trello

En este reto, vamos a replicar el tablero de Trello, para ello vamos a seguir un flujo de versiones que te guiarán en dónde enfocarte desde un inicio.

Al igual que en el reto anterior, puedes usar tu propio estilo o seguir la del sitio original. A continuación puedes ver como debe quedar:



Versión 0.0.1

- Se mostrará el texto "Añadir una lista ...".

Versión 0.0.2

- Al dar click en el botón de "Guardar", se mostrará un nuevo cuadro donde estará el nombre de la lista agregada.
- Mostrar un texto de "Añadir una tarea" dentro de la lista.

Versión 0.0.3

- Al dar click en "Añadir una tarea", deberá mostrar un formulario con un textarea y un botón que diga "Añadir".

Versión 0.0.4

- Poner *focus* al input al dar click en "Agregar nueva tarea".
- Al dar click en el botón de "Añadir", deberá aparecer el texto de la tarea debajo del título de la lista.

Versión 0.0.5

- Mostrar el formulario nuevamente debajo de la última tarea añadida.

Versión 0.0.6 (Extra)

- Poder agregar múltiples listas con tarjetas. Para esto, el formulario de "Añadir una lista" debe aparecer a la derecha de la lista anteriormente creada.

3. Laberinto (Opcional)

Crea una página web que implemente el juego del Laberinto utilizando DOM y eventos. El juego del Laberinto es un divertido Puzzle donde podrás darle a cualquier jugador la capacidad de interactuar con el mapa hasta encontrar la salida.

Recuerda que en Laberinto el objetivo es ir de un lugar A hacia un lugar B únicamente por el camino correcto.

> El Laberinto tiene raíces tan profundas como el mito griego sobre Teseo, que fue enviado a un laberinto para matar al minotauro. Teseo usó una bola de hilo para ayudarse a encontrar su camino de regreso, una vez que había terminado con la bestia.

Para interactuar con Teseom el jugador tendrá que hacer uso de las flechas de su teclado para poder mover y ayudar a Teseo en su búsqueda de la salida.

Tips para la solución

A continuación, encontrarás tips que podrían ayudarte con la solución, mucha suerte!

[Tip 1. | El Programa]

Crea un programa que represente un Teseo en un Laberinto y que implemente el algoritmo de búsqueda de un camino de salida.

GitHub set up-w800

Para que sea más fácil para nosotros, asumiremos que nuestro laberinto está dividido en "cuadrados". Cada cuadrado del laberinto está abierto u ocupado por una sección de pared. Teseo sólo puede pasar a través de los cuadrados abiertos del laberinto. Si Teseo se topa con una pared debe intentar una dirección diferente.

[Tip 2. | Creando el Laberinto]

Para crear el Laberinto se puede usar un array de cadenas para poder representarlo. La notación que puedes usar para cada elemento dentro del mapa son las siguientes:

1. `*` Una pared
2. `_` Un espacio vacío
3. `o` Posición inicial del jugador
4. `W` Salida del laberinto.

```
var mazeMap = [
    "******************",
    "*_________*______*",
    "*_*****_____******",
    "*______***__*__*_*",
    "***_*____*____**_*",
    "*___*____**__*___*",
```

```
    "******************"
];
```

Ese mapa se puede traducir en lo siguiente con la función de renderización/dibujo adecuado:

GitHub set up-w400

[Tip 3. | Identificando Objetos y Funciones]

Es importante identificar los objetos presentes. Si hacemos un análisis a la imagen anterior nos daremos cuenta que los objetos presentes son los siguientes:

1. Maze: Objeto que representa el Laberinto
2. Player: Objeto que representa a Teseo
3. MazeInterface: Objeto que representa la interfaz de Juego usando DOM

Además es importante identificar propiedades asociadas a estos objetos así como las principales funciones presentes. A continuación presentamos la interfaz del código de estos objetos y funciones.

```
var maze = {
    matrix: undefined, // representa el mapa del laberinto
    startX: undefined,  // posición x inicial del laberinto
    startY: undefined,  // posición y inicial del laberinto
    endX: undefined,  // posición x que representa la salida
    endY: undefined,// posición y que representa la salida
    startOrientation: undefined // orientación inicial
};

var player  = {
    x: undefined, // posición x actual del jugador
    y: undefined, // posición x actual del jugador
    orientation: undefined // orientación actual del jugador
};

var ORIENTATION = {
  LEFT: 1,
  UP: 2,
  RIGHT: 3,
  DOWN: 4
};

// representa la inferfaz usando DOM del laberinto
var mazeInterface = [];

// dibujar laberinto
function renderMaze(maze, player);
// establecer estilo en la posición x, y
function setStyleAt(maze, x, y, style);

// preguntar si (x,y) representa un muro, es decir  '*'
function isWall(maze, x, y);
// preguntar si (x,y) representa un espacio vacio, es decir  '_'
function isSpace(maze, x, y);
// preguntar si (x,y) representa la salida del laberinto, es decir 'W'
function isEnd(maze, x, y);

//  rotar a la izquierda
function turnLeft(player) ;
//  rotar a la derecha
function turnRight(player) ;
//  mover una posición hacia adelante en la dirección de player.direction
function moveForward(player) ;

// algoritmo de búsqueda para encontrar un camino de salida para un
// laberinto cualquiera
function exitMaze(player);
```

[Tip 4. | Preguntas guía]

- ¿Cómo hacemos para detectar el teclado del usuario? En particular, ¿las flechas?
- ¿Cómo hacemos para mover al personaje en el tablero?
- ¿Cómo hacemos para limitar el movimiento del personaje en el tablero? (si hay una pared, por ejemplo)
- ¿Qué significa pintar o renderizar?
- ¿Cómo hacemos para renderizar el tablero?
