**Clase 2**

En la clase anterior hicimos el bosquejo general de lo que serían las bases de app. En este bosquejo, nos quedo un pequeño error donde si el texto es muy largo, se sale de la nota y se muestra en otros lados, por lo que hay que agregar:

overflow-y: auto;

    overflow-x: hidden;

en la hojas de estilos a .card para evitar precisamente eso.

Quedamos en que una de las primeras cosas que tenemos que solucionar es poder agregar notas. Para ello vamos a usar un elemento llamado modal.

Para ello, fuera del **.container-fluid**, vamos a crear un nuevo cotenedor: **#contenedorModal**; y dentro de él colocaremos el modal:

<div id="contenedorModal">

    <div class="modal fade" id="modal" tabindex="-1">

        <div class="modal-dialog modal-fullscreen">

          <div class="modal-content bg-dark">

            <div class="modal-body">

              <button type="button" class="btn" data-bs-dismiss="modal">

                <i class="fa-solid fa-arrow-left-long" style="color: #ffffff;"></i>

              </button>

              <form action="">

                <div class="m-0">

                    <label for="tituloNotas" class="form-label" hidden></label>

                    <input type="text" class="w-100 border-0 bg-dark" id="tituloNotas" placeholder="Título">

                </div>

                <div class="m-0">

                    <label for="textoNotas" class="form-label" hidden></label>

                    <textarea class="w-100 border-0 bg-dark" id="textoNotas" placeholder="Contenido..." rows="10"></textarea>

                </div>

              </form>

            </div>

            <div class="modal-footer">

              <button type="button" class="btn btn-dark" id="btnAgregar" onclick="agregar()"

> Agregar</button>

            </div>

          </div>

        </div>

      </div>

</div>

Vamos por partes:

1. .fade es una clase de Bootstrap que se utiliza para dar una animación o un efecto de desvanecimiento para cuando aparece o se quita el modal.
2. .modal-dialog da la estructura básica del modal
3. . modal-fullscreen hace que se abra en toda la pantalla
4. Dentro del .modal-content tenemos normalmente un head, un body y un footer. En este caso eliminaremos el head
5. Dentro del body, tenemos primero un botón para cerrar el modal y también dibujamos un formulario que estará compuesto por un input para el titulo y un text-area para el contenido.
   1. Tanto el input como el tect-area van a tener .w-100 (para que tengan el mismo ancho que el modal, .bortder-0 (para que no tenga bordes)
6. Por ultimo tenemos el botón agregar con un evento onclick que llamar una funcion de js que ya crearemos.

Por supuesto, todo esto tiene que llamarse de algún lado. Por eso, usaremos el botón .btn-agregar para esto, asi que le agregaremos los atriburtos:

data-bs-toggle="modal" data-bs-target="#modal"

donde data-bs-target se refiere al id del modal que estamos trabajando

también agregamos los siguientes estilos:

#textoNotas{

    overflow-y: auto;

    overflow-x: hidden;

    resize: none;

}

#tituloNotas:hover,

#tituloNotas:focus,

#textoNotas:hover,

#textoNotas:focus{

    outline: none;

    text-decoration: none;

    cursor: default;

}

form input, textarea{

    color: aliceblue;

}

Donde evitamos que el text-area con id #textoNotas pueda modificar su tamaño y a su vez le permitimos general una barra de scroll vertical si lo necesita.

Además quitamos las decoraciones y cambios que surgen cuando escribimos en el input o en el text-area y le damos un color blanco a las letras de los mismos.

Ya con todo listo por aca, vamos al js.

Una vez allá, creamos la function agregar(), en ella vamos a dibujar las notas conforme se vayan agregando:

const contenedorNotas=document.getElementById('contenedorNotas');

function agregar(){

    const tituloNotas=document.getElementById('tituloNotas').value;

    const textoNotas=document.getElementById('textoNotas').value;

contenedorNotas.innerHTML='';

contenedorNotas.innerHTML+=`

        <div class="col">

            <div class="card">

                <div class="card-body">

                    <h5 class="card-title">${tituloNotas}</h5>

                    <p class="card-text">${textoNotas}</p>

                </div>

            </div>

        </div>

    `

}

Primero, tengo que llamar al contenedor donde voy a dibujar las notas por su id.

Luego, dentro de la funcion, estoy declarando las variables donde, atravez del id, voy a traer el value del input para el titulo y del text-area para el texto o contenido de la nota. Una vez obtenida la información, dibujo las notas en el contenedor. En este caso estoy usando comillas simples invertidas porque es mas sencillo concatenar las variables dentro del texto.

Nota: 1ro se limpia el contenedor y luego se esta usando “+=” después del innertHTML para que se vaya dibujando una nota seguida de la otra sin borrar la anterior.

Probamos….

Vemos que se esta agregando, pero tengo que cerrar el modal manualmente.

Basados en la documentación de Bootstrap, vamos a crear una instancia que me permita manejar el modal desde js. Para eso declaro el modal y luego en base a el hago la instancia, luego, al final de la funcion agregar() uso .hide() sobre la instancia para cerrar el modal

const ModalNotas=document.getElementById('modal');

const intanceModalNotas=new bootstrap.Modal(ModalNotas);

const contenedorNotas=document.getElementById('contenedorNotas');

function agregar(){

    const tituloNotas=document.getElementById('tituloNotas').value;

    const textoNotas=document.getElementById('textoNotas').value;

contenedorNotas.innerHTML='';

contenedorNotas.innerHTML+=`

        <div class="col">

            <div class="card">

                <div class="card-body">

                    <h5 class="card-title">${tituloNotas}</h5>

                    <p class="card-text">${textoNotas}</p>

                </div>

            </div>

        </div>

    `

intanceModalNotas.hide();

}

Ahora si, estamos agregando notas y el modal se cierra solo, pero… al intentar agregar una nuevo nota, vemos que el input y text-area tienen aun la información de la nota que acabamos de agregar. ¿Cómo lo arreglamos?

Para eso, vamos a hacer un **addeventlistener** para escuchar cuando se cierre el modal y se limpie el contenido:

ModalNotas.addEventListener('hidden.bs.modal',()=>{

    document.getElementById('tituloNotas').value='';

    document.getElementById('textoNotas').value='';

})

Ok, ya estamos agregando notas. Ahora, queremos que cuando las agregue, se queden ahí hasta que yo las elimine.

El **Local Storage** es una API del navegador web que permite almacenar datos de forma persistente en el dispositivo del usuario. Esta información se mantiene incluso después de cerrar y volver a abrir el navegador.

Para agregar elementos al local storage se usa el método setItem cuyos atributos serán el nombre de donde se va a guardar y el contenido. Sin embargo localstorage solo almacena strings, por lo que si vamos a trabajar con objetos, debemos primero convertir el objeto en un scrip y luego si guardarlo. Primero, vamos a crear un array vacio donde almacenar todo momentaniamente:

const notas=[];

function agregar(){

    const tituloNotas=document.getElementById('tituloNotas').value;

    const textoNotas=document.getElementById('textoNotas').value;

    notas.push({titulo:tituloNotas,texto:textoNotas});

const notasJSON= JSON.stringify(notas);

    localStorage.setItem('notas',notasJSON);

    contenedorNotas.innerHTML='';

contenedorNotas.innerHTML+=`

        <div class="col">

            <div class="card">

                <div class="card-body">

                    <h5 class="card-title">${tituloNotas}</h5>

                    <p class="card-text">${textoNotas}</p>

                </div>

            </div>

        </div>

    `

    intanceModalNotas.hide();

}

.push es un método con el cual puedo agregar un nuevo elemento a un array en su ultima posicion

.JSON.stringify convierte un array u objeto en una cadena de texto.

Ya estamos agregando notas nuevas, pero yo necesito poder mostrar lo que se ha guardado, es por eso que notas no puede ser igual a un array vacio, debe ser igual a lo que tengamos en localStorage:

const notas=obtenerNotas();

function obtenerNotas(){

    let notasGuardadas=JSON.parse(localStorage.getItem('notas'));

    if(notasGuardadas===null){

        result=[]

    }else{

        result=notasGuardadas;

    }

    return result;

}

En esta funcion, 1ro pedimos las notas que están en el localStorage mediante **.getItem()** pasandole como parámetro el nombre de donde esta guardada la información, transformando esa cadena de texto luego en un array de nuevo con **JSON.parse()**. Luego, validamos si la respuesta no es nula… si es nula devolveremos un array vacio y si si hay algo, devolveremos lo que haya.

Ya teniendo información con que trabajar. Vamos a mostrarla. Antes de eso, hay que entender que, por definición, una funcion solo debe hacer 1 cosa y vemos que agregar() recibe la información del html, la guarda y la muestra, asi que vamos a aligerarle la carga:

function agregar(){

    const tituloNotas=document.getElementById('tituloNotas').value;

    const textoNotas=document.getElementById('textoNotas').value;

    notas.push({titulo:tituloNotas,texto:textoNotas});

    localStorage.setItem('notas',JSON.stringify(notas));

    mostrarNotas();

    intanceModalNotas.hide();

}

function mostrarNotas(){

    contenedorNotas.innerHTML='';

    notas.forEach(nota => {

        contenedorNotas.innerHTML+=`

        <div class="col">

            <div class="card">

                <div class="card-body">

                    <h5 class="card-title">${nota.titulo}</h5>

                    <p class="card-text">${nota.texto}</p>

                </div>

            </div>

        </div>

    `

    });

}

En 1er lugar, a agregar() le dejamos solo la funcion de tomar la información de DOM y guardarla en el localStorage, luego llama a **mostrarNotas()** y cierra el modal.

En mostrarNotas(), vaciamos en contenedorNotas y luego para llenarlo recorremos el array notas con un forEach para asi mostrar el titulo y el texto de cada nota.

Por ultimo, falta un detalle… y es que al agregar una nota, bien… se muestra, pero… no se muestran desde un principio. Eso se debe a que solo estamos llamando a **mostrarNotas()** despues de agregar 1 nota. También hay que llamarlo fuera de las funciones para asegurarnos de se muestren las notas.

Probamos….

Ahora, la cosa se esta complicando. Como podemos hacer esto mas entendible y mas escalable?

Lo mejor seria llevarlo a orientación a objetos

En este caso, solo necesitamos 1 clase, la clase notas, la cual en principio debe intentar alimentarse del local storage, por ello el constructor será solamente lo que nos arroge el local storage y al igual que antes, dicho resultado debe comprobarse, para que en caso de que no haya nada, no me lance error

const contenedorNotas=document.getElementById('contenedorNotas');

const ModalNotas=document.getElementById('modal');

const intanceModalNotas=new bootstrap.Modal(ModalNotas);

class Notas{

    constructor(){

        this.notas=this.obtenerNotas();

    }

    obtenerNotas(){

        let notasGuardadas=JSON.parse(localStorage.getItem('notas'));

        return notasGuardadas || []

    }

}

Notese lo siguiente:

1. Para declarar una clase, se usa **class** seguido del nombre de la clase con la 1ra letra en mayuscula.
2. Cuando voy a llamar un elemento o un método de la misma clase, uso **this.** Seguido del nombre del elemento o el método.
3. Los métodos se declaran exactamente igual que las funciones, solo que sin la palabra function.
4. El método **obtenerNotas()**  es igual que la funcion del mismo nombre, solo que el condicional lo resumo en 1 sola linea.

Ya teniendo las notas, debo mostrarlas:

mostrarNotas(){

        contenedorNotas.innerHTML='';

        this.notas.forEach(nota => {

            contenedorNotas.innerHTML+=`

                <div class="col">

                    <div class="card">

                        <div class="card-body">

                            <h5 class="card-title">${nota.titulo}</h5>

                            <p class="card-text">${nota.texto}</p>

                        </div>

                    </div>

                </div>

            `

        });

    }

Probamos…

No se ve nada, porque?

Debemos instanciar un nuevo objeto de la clase Notas(), eso ya lo hicimos… con que?

Con el modal para poder cerrarlo.

Asi que fuera de la clase Notas() colocamos:

const notas=new Notas();

notas.mostrarNotas()

donde instanciamos un nuevo objeto de la clase notas y llamamos al método mostrarNotas()

ya viendo las notas, vamos a agregar nuevas:

agregarNota(){

        const titulo=document.getElementById('tituloNotas').value;

        const texto=document.getElementById('textoNotas').value;

        this.notas.push({titulo:titulo,texto:texto});

        this.guardarNota();

    }

Fíjense que en este caso estamos haciendo **.push** no a un array cualquiera, sino al array notas de esta clase, por eso el **this** y luego llamamos al método **guardarNota()** que quedaría asi:

guardarNota(){

        let notasString=JSON.stringify(this.notas)

        localStorage.setItem('notas',notasString);

        this.mostrarNotas();

        intanceModalNotas.hide();

    }

Probamos…

Vemos que no pasa nada…

Vamos a la consola y hay un error… no cambiamos la funcion que se llama con el evento onclick!!!

Como ahora agregarNota es parte de la clase Notas(), el evento debería quedar asi:

onclick="notas.agregarNota()"

volvemos a probar…

ahora si, todo funciona bien.

En dado caso que salga un error, antes de hacer cualquier cosa, dirigite a aplication/localStoraje y busca donde se están guardando los datos y borralos, recarga y prueba de nuevo.

Hasta aquí llegamos la clase de hoy. Como bono:

1. Agreguen el addeventlistener para borrar el contenido del modal al cerrarlo:

ModalNotas.addEventListener('hidden.bs.modal',()=>{

    document.getElementById('tituloNotas').value='';

    document.getElementById('textoNotas').value='';

})

1. En el html, a la div#foot le vamos a quitar la clase .mt-5 y le vamos a agregar las clases .mb-2 y .fixed-bottom para dejar el botón pegado abajo, sin importar que no hayan notas.

Para la próxima clase tenemos las siguientes preguntas:

1. ¿puedo hacer algo para que cuando vaya a crear una nota, por default tenga el cursor en el input?
2. ¿Cómo hacer el método eliminar?
3. ¿Cómo hacer el método editar?