Clase 4

Comenzamos donde nos quedamos la ultima vez.

Hacer repaso de las imágenes y del offcanvas

Agregamos un método para generar un color de forma aleatoria y se guarde con las notas  
colorRandom() {

        const matiz = Math.floor(Math.random() \* 360);

        const saturacion = Math.random() \* 0.4 + 0.6;

        const ligereza = Math.random() \* 0.3 + 0.7;

        return `hsl(${matiz}, ${saturacion \* 100}%, ${ligereza \* 100}%)`;

    }

Se llama ese método para almacenar su resultado en una variable y lo pasamos al array en guardar nota y en editar nota. También lo usamos para que la card salga de ese color

Instanciamos un objeto del offcanvas que se creo para cerrarlo

Vamos con otro plugin que me pidieron, este es el de **ubicación:**

**Instalación:**

cordova plugin add cordova-plugin-geolocation

hay dos formas que se puede trabajar este plugin, una es la forma gratuita y la otra es paga. Dentro de la forma gratuita puedo acceder al gps, obtener la ubicación actual, incluso seguir mi ubicación, pero es un poco complicado para mostrarla. En la versión paaga, se puede mostrar en un iframe, ver cosas del cima, tiendas cerca, etc…

Vamos primero con la paga:

1. Api de Google maps

Para poder acceder a la forma paga, debemos ir a Google cloud console y crear una cuenta.

Creamos un proyecto y le activamos la api de js

Creamos una credencial (si queremos la restringimos) y la agregamos a nuestro proyecto.

Antes de agregarla, hay que activar la facturación, para ello neesitaremos una tarjeta de crédito en dólares que no sea prepagada (no se hace ningún cobro sin autorización). Una vez activada la facturación, vamos al código:  
  
agregamos esto al html:

        <div id="map" style="height: 400px; width: 100%;"></div>

        <button onclick="openGoogleMaps()">abrir mapa</button>

        <script src="https://maps.googleapis.com/maps/api/js?key=suClave "></script>

Y luego esto al js:

    var Latitude = undefined;

    var Longitude = undefined;

    // Obtener coordenadas geográficas

    function getMapLocation() {

        navigator.geolocation.getCurrentPosition(onMapSuccess, onMapError, { enableHighAccuracy: true });

    }

    // Callback de éxito para obtener coordenadas

    var onMapSuccess = function (position) {

        Latitude = position.coords.latitude;

        Longitude = position.coords.longitude;

        // Aquí puedes llamar a la función para mostrar el mapa

        showMap();

    }

    // Callback de error

    function onMapError(error) {

        console.log('code: ' + error.code + '\n' + 'message: ' + error.message + '\n');

    }

    // Función para observar el cambio de posición

    function watchMapPosition() {

        return navigator.geolocation.watchPosition(onMapWatchSuccess, onMapError, { enableHighAccuracy: true });

    }

    // Evento DOMContentLoaded

    document.addEventListener('DOMContentLoaded', () => {

        getMapLocation();

    });

// Mostrar mapa usando coordenadas

function showMap() {

    var mapOptions = {

        center: new google.maps.LatLng(Latitude, Longitude),

        zoom: 15,

        mapTypeId: google.maps.MapTypeId.ROADMAP

    };

    var map = new google.maps.Map(document.getElementById("map"), mapOptions);

    var latLong = new google.maps.LatLng(Latitude, Longitude);

    var marker = new google.maps.Marker({

        position: latLong,

        map: map

    });

    map.setCenter(marker.getPosition());

}

function openGoogleMaps() {

    if (Latitude !== undefined && Longitude !== undefined) {

        var zoom = 15; // Nivel de zoom deseado

        var googleMapsLink = `https://www.google.com/maps?q=${Latitude},${Longitude}`;

        window.open(googleMapsLink, '\_blank');

    } else {

        console.log('Coordenadas no disponibles.');

    }

}

Ya con este código tenemos dos cosas:

1. El iframe con el mapa
2. Un btn para abrir el mapa en Google

Ahora pasemos a otra forma: **la gratuita**

1. Plugin en version gratuita:

Dentro de la versión gratuita tenemos 3 metodos para este plugin:

1. navigator.geolocation.getCurrentPosition: toma la posición actual
2. navigator.geolocation.watchPosition: sigue la posición
3. navigator.geolocation.clearWatch: cancela la anterior

en este proyecto vamos a trabajar con la anterior.

Para ello vamos a agregar unas cosas:

1. en el html:
   1. al inicio del footer del modal:

<div class="me-auto">

  <button id="ubicacion" class="btn" hidden>

    <i class="fa-solid fa-location-dot fa-2xl" style="color: #ffffff;"></i>

  </button>

</div>

* 1. al fial del off-canvas:

<div class="row mt-1 pb-2">

                                <button type="button" class="btn" id="btnUbicacion" onclick="manejador.tomarUbicacion()">

                                  <i class="fa-solid fa-location-dot" style="color: #ffffff;">

                                      <span class="ms-2">Guardar Ubicación</span>

                                  </i>

                                </button>

                            </div>

1. en el js:
   1. en la class Manejador:

tomarUbicacion(){

        var options={ maximumAge: 3000, timeout: 5000, enableHighAccuracy: true };

        function onSuccess(position) {

            var ubicacion=`https://www.google.com/maps?q=${position.coords.latitude},${position.coords.longitude}`;

            notas.ponerUbicacion(ubicacion);

        }

        function onError(error) {

            console.log('code: '    + error.code    + '\n' +

                'message: ' + error.message + '\n');

        }

        // Options: throw an error if no update is received every 30 seconds.

        //

        var watchID = navigator.geolocation.getCurrentPosition(onSuccess, onError, options);

    }

* 1. en la class Notas:

ponerUbicacion(ubicacion){

    const btnUbicacion=document.getElementById('ubicacion');

    btnUbicacion.hidden=false;

    btnUbicacion.dataset.ubicacion=ubicacion;

    this.instOffCanvas.hide();

    if(notas.modal.hasAttribute('name')){

        const index=notas.modal.getAttribute('name');

        notas.editarNota(index);

    }

}

* + 1. dentro de agregar notas: declaramos la variable ubicación, luego nos preguntamos si el elemento del html que debería tener la ubicación tiene un data-ubicación, si es asi… ubicación va a ser igual a eso que este ahí, si no, ubicación va a ser igual a un string vacio. Como sea, lo que valga ubicación, lo pusheamos al array this.notas

agregarNota(){

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

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

        const contenedorImg=document.getElementById('imgNotas');

        const imagenes=contenedorImg.getElementsByTagName('img');

        const imagen=[];

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

            imagen.push(imagenes[i].src);

        }

        const color=this.colorRandom();

        var ubicacion;

        document.getElementById('ubicacion').dataset.ubicacion ? ubicacion=document.getElementById('ubicacion').dataset.ubicacion : ubicacion="";

        this.notas.push({ titulo, texto, imagen, color, ubicacion });

        this.guardarNota();

        this.instModal.hide();

    }

* + 1. en seleccionar notas: buscamos el elemento por su id, luego metemos un condicional para ver si nota.ubicacion es diferente a vacio, si es asi, le quitamos el hidden, le agregamos un data-ubicación con lo que valga ubicación y escuchamos si se toca el botón. Si si se esta tocando, llamamos a this.activarLink()

    seleccionarNota(index){

    const btnElimnar=document.getElementById('btnEliminar');

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

    const btnUbicacion=document.getElementById('ubicacion');

    btnElimnar.hidden=false;

    btnAgregar.hidden=true;

    const nota=this.notas[index];

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

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

    const imagen=document.getElementById('imgNotas');

    this.modal.setAttribute('name',`${index}`)

    titulo.value=nota.titulo;

    texto.value=nota.texto.replace(/<br>/g, '\n');

    imagen.innerHTML=`${nota.imagen ? nota.imagen.map(img => `

            <div>

                <img src="${img}" alt="Imagen de la nota" class="img-fluid mt-2">

                <button type="button" class="btn-close close-img " onclick="notas.quitarFoto(event)"></button>

            </div>`

        ).join('') : ''}`;

    if(nota.ubicacion!==""){

        btnUbicacion.hidden=false;

        btnUbicacion.dataset.ubicacion=nota.ubicacion;

        btnUbicacion.ontouchstart=()=>{

            this.activarLink();

        }

    }

    this.instModal.show();

    titulo.oninput=()=>{

        if (this.modal.hasAttribute('name')) {

            const indice = this.modal.getAttribute('name');

            this.editarNota(indice);

        }

    }

    texto.oninput=()=>{

        if (this.modal.hasAttribute('name')) {

            const indice = this.modal.getAttribute('name');

            this.editarNota(indice);

        }

    }

    btnElimnar.onclick=()=>{

                this.eliminarNota(index);

    }

}

* + 1. en editar hacemos los mismos cambios que en agregar.
    2. Creamos el método activarLink:

activarLink(){

    const btnUbicacion = document.getElementById('ubicacion');

    let timeoutId;

    timeoutId = setTimeout(() => {

        console.log(‘borrable’)

    }, 500); // Espera 500 ms antes de llamar a confirmacion

    btnUbicacion.onmouseup=()=>{

        clearTimeout(timeoutId);

        window.open(btnUbicacion.dataset.ubicacion, "\_blank");

    };

}

Con este método, si solo pulsamos rápido el icono que se creo cuando agregamos la ubicación, se abre el mapa en Google maps, si lo dejamos pulsado brevemente…. Ahí vamos a meter la lógica para quitar la ubicación.

Hay que recordar ir a limpiarModal y colocar lo propio para que se limpie también el botón de la ubicacion

En base a esto, vamos con otro plugin: **diálogos**

cordova plugin add cordova-plugin-dialogs

este plugin tiene 6 metodos:

1. navigator.notification.alert:

en suma, es un alert personalizado, contiene los siguientes parámetros:

1. mensaje : mensaje de diálogo. (Cadena).
2. alertCallback : devolución de llamada que se invocará cuando se cierre el cuadro de diálogo de alerta. (Función).
3. título : título del cuadro de diálogo. (Cadena) (opcional, el valor predeterminado es Alert)
4. buttonName : Nombre del botón. (Cadena) (Opcional, el valor predeterminado es OK)

ejemplo:

 navigator.notification.alert(

            'Acabas de eliminar una nota',  // message

            this.mensaje,         // callback

            'Atencion',            // title

            'ok'                  // buttonName

        );

mensaje(){

        console.log('aaaaa')

    }

1. navigator.notification.confirm

este ya es un cuadro de dialogo que me permite hacer una pregunta y obtener una confirmación

en si, se ve de la siguiente manera

navigator.notification.confirm(message, confirmCallback, [title], [buttonLabels])

1. mensaje : mensaje de diálogo. (Cadena)
2. confirmCallback : devolución de llamada que se invocará con el índice del botón presionado (1, 2 o 3) o cuando se cierre el diálogo sin presionar un botón (0). (Función)
3. título : título del cuadro de diálogo. (Cadena) (opcional, el valor predeterminado es Confirm)
4. buttonLabels : Matriz de cadenas que especifican las etiquetas de los botones. (Matriz) (Opcional, el valor predeterminado es [ OK,Cancel])

tomar nota: el mensaje no admite mas de 3 lineas y Android admite un máximo de 3 botones. En el navegador solo admite 2 botones

ejemplo:

vamos a hacer la confirmación para poder eliminar una nota:

1. En notas.seleccionarNotas() en vez de llamar a eliminarNota, llamaremos a Manejador.confirmacion(index), pero lo haremos de otra forma, lo llamaremos mediante un fetch, esperando la resolución de la promesa y dependiendo de eso, veremos que haremos:

btnElimnar.onclick=()=>{

            manejador.confirmacion()

            .then(res=>{

                if(res===1){

                    this.eliminarNota(index);

                }else{

                    this.instOffCanvas.hide();

                }

            })

            .catch(error=>{

                console.log(error);

            });

        }

1. Creamos ese método:

    confirmacion(){

        return new Promise((res,rej)=>{

            navigator.notification.confirm(

            '¿Desea Continuar??',

                (btnIndex)=>{

                    res(btnIndex);

                },

                    'Atención',

                    ['Si','no']

                );

        });

    }

Como se puede ver, lo estamos haciendo como una promesa y cuando se resuelva dicha promesa yo la recojo en seleccionar nota y sabre que hacer. El hacer este método de esta forma, nos permite reutilizarlo para otras confirmaciones necesarias, como para eliminar una imagen y para quitar la ubicación… agreguemosla

1. navigator.notification.prompt

en este caso, me permite obtener una entrada de texto del usuario y su ejemplo podría ser este:

function preguntar(){

    navigator.notification.prompt(

        'introduzca su nombre',  // message

        onPrompt,                  // callback to invoke

        'Registro',            // title

        ['Ok','Exit'],             // buttonLabels

        'Desconocido'                 // defaultText

    );

}

function onPrompt(results) {

    alert("haz seleccionado el boton numero " + results.buttonIndex + " y haz ingresado: " + results.input1);

}

Para probar esto, usaremos un botón en el index

1. navigator.notification.beep

este es bastante sencillo, se llama asi:

navigator.notification.beep(2);

en este ejemplo se utiliza para que suene dos veces

1. navigator.notification.dismissPrevious
2. navigator.notification.dismissAll

estos dos últimos, se utilizan para cerrar mensajes abiertos, pero dado a que las notificaciones perse se cierran automáticamente, no los profundizaremos dado su irrelevancia

**botón atrás**

la clase pasada hablamos un poco de la api de js que usa cordova para detectar los estados del ciclo de vida de la app. Dicha api también sirve para manejar los botones del teléfono y podemos manejarlos escuchando los eventos:

backbutton

menubutton

searchbutton

startcallbutton

endcallbutton

volumedownbutton

volumeupbutton

y se hace tal cual como cualquier addeventlistener

1ro, vamos a preparar el terreno… como diría mate, para ir para atrás, debes saber donde haz estado, asi que agregaremos algo al constructor de la clase manejador:

constructor(){

        this.pantallaActual='';

        this.start=document.addEventListener('deviceready',()=>{

            this.iniciarEventos();

        });

    }

Ahora, sabemos que tenemos algunos escenarios en los que podemos usar el botón de atrás:

1. cuando estamos en el modal y queremos regresar a ver todas las notas.
2. Cuando abrimos el offcanvas para borrar una nota o agregar la ubicación.
3. Cuando estamos buscando algo.

Para el 1er punto, vamos a iniciarEventos() y agregamos un escuchador de eventos para cuando se abra el modal:

notas.modal.addEventListener('show.bs.modal',()=>{this.pantallaActual='modalAbierto'});

para el 2do, hacemos algo parecido (lógicamente, hay que llamar ese elemento del html dentro del constructor de Notas:

notas.Offcanvas.addEventListener('show.bs.offcanvas',()=>{this.pantallaActual='offCanvas'})

y para el tercero, vamos a buscarNota() y agregamos al principio:

manejador.pantallaActual='buscarNota';

ya preparado el terreno, vamos de nuevo a iniciarEventos() y agregamos el escuchador para el evento que queremos:

document.addEventListener("backbutton",()=>{

    this.atras()

});

Creamos el método atrás()

atras(){

    switch (this.pantallaActual) {

        case 'modalAbierto':

            notas.instModal.hide();

            this.pantallaActual='';

            break;

        case 'buscarNota':

            notas.cancelarBusqueda();

this.pantallaActual='';

            break;

        case 'offCanvas':

            notas.instOffCanvas.hide();

            this.pantallaActual='modalAbierto';

            break;

        default:

            navigator.app.exitApp();

            break;

    }

}

Y hay que agregar un escuchador en iniciareventos para cuando se cierre el offcanvas cambie el valor de pantallaActual

Dentro de dicho método usamos un swich tomando this.pantalla como variable para ver que se ejecutara y aprovechamos y actualizamos los valores de dicha variable. Por supuesto, en caso de un valor desconocido, se cierra la app.