

Sección 26

181) Guardar las nuevas citas

- Para evitar la duplicación de registros, que es cuando se toma como si todas las citas actuales fueran la ultima que estuvimos agregando.
- Para resolverlo en lugar de pasarle el original, le pasamos una copia, no le vamos a pasar la referencia de todo el objeto
- → `administrarCitas.agregarCita({...citaObj})`
- Y arriba en la clase queda:
- → `class Citas{`
- `constructor(){`
- `this.citas=[];`
- `}`
- `agregarCita(cita){`
- `this.citas=[...this.citas, cita];`
- `}`
- `→}`

Se manda como valor la ultima copia, no se repite como antes

Spread es útil porque te evitas acceder o estar modificando el valor global.

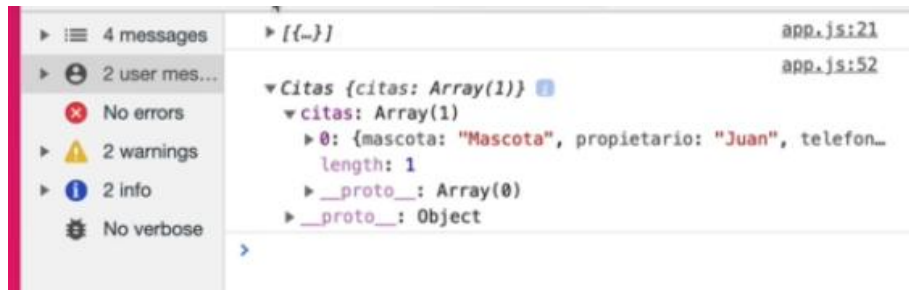
- se tiene que resetear el formulario pero también el objeto donde almacenamos
- Reiniciando el objeto:

- \rightarrow function reiniciarObjeto(){
- citaObj.mascota="";
- citaObj.propietario="";
- citaObj.telefono="";
- citaObj.fecha="";
- citaObj.hora="";
- citaObj.sintomas="";
-

\rightarrow }

182. Mostrar las Citas en el HTML

- Puedes aplicar destructuring desde el paréntesis.
- Si tenemos citas, y después citas



- No se puede utilizar destructuring porque se llaman igual
- Por lo tanto extrayendo las citas de objeto aplicando destructuring en el paréntesis
- → imprimirCitas({citas}){
- console.log(citas);
- }
- Las etiquetas **H2** corresponden a los subtítulos.
- → <h2>

183. Eliminar las citas

- Para encontrar iconos pagina
- Heroicons, solo das click en el icono en la pagina y lo pegas, después de eliminar y ya puedes tener el icono
- <https://heroicons.dev/>
- → btnEliminar.innerHTML='Eliminar <svg class="w-6 h-6" fill="none" stroke="currentColor" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M10 14l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2m7-2a9 9 0 11-18 0 9 9 0 0118 0z"></path></svg>';

184. Editar una Cita

- Tal vez me equivoque y no quiero eliminar la cita y volverla a capturar
- Icono para cerrar en la pagina de herocoins X
- Icono para editar pen
- Agregando icono:
- →`btnEditar.innerHTML='Editar <svg class="w-6 h-6" fill="none" stroke="currentColor" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M15.232 5.232l3.536 3.536m-2.036-5.036a2.5 2.5 0 113.536 3.536L6.5 21.036H3v-3.572L16.732 3.732z"></path></svg>'`
-
- Editar los datos reasignar los datos de la cita en el formulario para su edición
- Para entrar en modo edición generamos una variable `let` , luego le asignamos el valor de `true` en alguna parte del código, y para entrar en el modo generamos un `if`

185. Modificar la Cita en el objeto

- Actualizar cita
- Si cita.id es igual a citaActualizada entonces el objeto completo pasa a ser citaActualizada, o sea que reescribimos lo que tenga esa cita, caso contrario,vamos a retornar la cita actual o sea cita, para no perder ninguna perdida de las citas
- → editarCita(*citaActualizada*){
- //.map para que cree un nuevo arreglo que se asigna a citas
- this.citas=this.citas.map(*cita* => cita.id === citaActualizada.id ? citaActualizada : cita)
- → }

Sección 27: Sets Maps y Symbols

186. Sets y sus Caracteristicas

- Los sets en javascript: son pregunta para obtener un trabajo

Te permite crear una lista de valores sin duplicados, cuando maneja un gran volumen de datos, tiende a ser mas rapido que un objeto o que un arreglo. Algunos métodos que existen en arreglos se pueden utilizar con los sets pero no todos, tiene su propia sintaxis y tiene sus propios metodos (set methods). Los set solamente son valores, no es llave valor como un objeto. Detecta mayusculas y minuscula como diferentes


- Crear un set:

→ `const carrito= new Set();`

→ `console.log(carrito);`

- Agregar valor a set

→ `carrito.add('Disco #3');`

- Para saber cuantos elementos hay  Elemento a agregar

→ `console.log(carrito.size);`

- Comprobar si un valor existe en el set

→ `carrito.has('camisa');`

Nombre del set y si tiene camisa

→ `console.log(carrito.has('camisa')); // true or false`

- Eliminar un elemento del set:

→ `carrito.delete('Disco #3');`

→ `console.log(carrito.delete('Disco #3')); // true or false, si tratamos de eliminar un valor que no Existe imprime un false y aparece un true si si elimina algo`

- Para eliminar todos los elementos del set

→ `carrito.clear();`

- Los sets son iterables

→ `carrito.forEach(producto => {`

- `console.log(producto);`

→ `})`

- Llave y valor en un set imprime lo mismo, por ejemplo si tratamos de imprimir el indice, imprime lo mismo

→ `carrito.forEach((producto, index) => {`

- `//console.log(producto);`

- `console.log(index);`

→ `})`

- Si le pasamos 3 valores, aparece el set completo

→ `carrito.forEach((producto, index, pertenece) => {`

- `//console.log(producto);`

- `//console.log(index);`

- `console.log(pertenece);`

→ `})`

Ej) del siguiente arreglo elimina los duplicados

→ `const numeros= [10, 20, 30, 40, 50, 10, 20];`

-

`const noDuplicados= new Set(numeros); // le pasamos el arreglo a set`

→ `console.log(noDuplicados);`

187. Qué es un WeakSet y en que se diferencia de un Set. Set débil

- Crear un set debil:
→ `const weakset = new WeakSet();`

A diferencia del set, en el set le puedes pasar cualquier valor (objetos, numeros, booleanos, etc.) en el WeakSet solamente le puedes pasar u agregar objetos

- `.size` en este no existe
- No son iterables no puedes utilizar `.foreach`
- Agregar objeto
→ `weakset.add(cliente);`
- Verificar si Existe un objeto
→ `console.log(weakset.has(cliente)); // true or false`
- Eliminar un objeto
→ `weakset.delete(cliente);`

188. Que son los Maps

- Los maps son listas ordenadas en llave y valor (un objeto con una sola propiedad(1 llave y un valor) y la llave y el valor pueden ser cualquier tipo de dato (arreglo, numero etc.).
- Son especialmente diseñados para agregar , quitar elementos o recorrerlos y cuando son muy grandes tienen mucho mejor performance que un objeto
- Para agregar un elemento
- Crear un Map:
→ `const cliente=new Map();`
- Agregar elementos a un Map:
→ `cliente.set('nombre', 'karen');` //llave valor en (), al principio nombre del map
- (Map llave valor, mientras que el set, solamente el valor)
- Cuantos elementos hay en el map:
→ `console.log(cliente.size);`
- Si existe un elemento dentro
→ `console.log(cliente.has('nombre'));` //true or false
- Obtener un valor, por ejemplo lo que este en nombre
→ `console.log(cliente.get('nobre'));` // si no existe nombre, marca undefined
- Eliminar un elemento
→ `cliente.delete('saldo');`
- Eliminar todos los elementos del map
→ `cliente.clear();`

- Iniciar un map con valores ej 2 arreglos dentro de un arreglo
→ `const paciente= new Map(['nombre','paciente'], ['cuarto', 'no definido']); // llave , valor en []`
- Agregar valores despues de haber inicializado con map
→ `paciente.set('dr', 'Dr Asignado');`
- Si pones la misma llave, eso va a reescribir un valor
→ `paciente.set('nombre', 'Antonio');`
- Los map son iterables, para imprimir los valores de las llaves:
→ `paciente.forEach(datos => {`
 - `console.log(datos);``→})`
- Para imprimir las llaves:
→ `paciente.forEach(datos, index => {`
 - `console.log(index);``→})`

189. Que son los WeakMaps o mapas debiles

- Menos utilizados
 - Crear weakmap:
→ `const weakmap= new WeakMap();` //crear weak map
 - Agregando valores a weakmap:
→ `weakmap.set(producto, "Monitor");` //agregando objeto y string a weakmap
 - Saber si tiene el objeto producto
→ `console.log(weakmap.has(producto));` // true or false
 - Oculta info:
→ `console.log(weakmap.get(producto));` // oculta lo que hay en el objeto y solo imprime string "monitor"
 - Eliminar producto
→ `console.log(weakmap.delete(producto));` // true or false
-
- Para algo que sirven muy bien los WeakMap es para mantener una serie de datos como privados (no guardar datos muy sensibles como los datos de una tarjeta de crédito o el password de un usuario , porque solo no se pueden acceder a esos valores con `.get`, pero si imprimimos WeakMap, si aparece todo el contenido de WeakMap)
 - También son llave valor
 - No se pueden iterar, no puedes utilizar `.forEach` , `.size` tampoco existe
 - Para eliminar:
 - Solamente acepta objetos, no acepta string o numero.

190. Symbols y sus Características

- Fueron nuevos en ES6 permiten crear una propiedad única, no hay 2 symbols que sean iguales (mas usado en librerías) entrevistas de trabajo
- Crear symbol:
→ `const sym2= Symbol();`
- No importa que tengan el mismo valor, siempre son diferentes
→ `Console.log(Symbol()===Symbol()); //false`
- Agregar nombre y apellido como llaves del objeto
→ `const nombre= Symbol();`
- `const apellido=Symbol();`
- `const persona={`
- `//Agregar nombre y apellido como llaves del objeto`
- `persona[nombre]='Juan';`
- `persona[apellido]='De la torre';`
- `persona.tipoCliente='premium';`
- `persona.saldo=500`
- Para acceder al valor, tenemos que utilizar los corchetes
→ `console.log(persona[nombre]);`
- Las propiedades que tengas definidas por medio de un simbol no son iterables, si iteramos solo muestra los que no son Symbol
→ `for(let i in persona){`
- `console.log(i);`
- `}`

- Cuando creas un symbol puedes agregar una descripcion de ese symbol
- `const nombreCliente= Symbol('Nombre del cliente')`
- `const cliente={};`
`cliente[nombreCliente]='Pedro'; //Asignar a cliente symbol y darle un valor`
- `console.log(cliente[nombreCliente]); //acceder al valor pedro`
- `console.log(nombreCliente); //Acceder a la descripcion 'Nombre del cliente'`

191. Iteradores en JavaScript

- Crear nuestro propio iterador
- Definir una función que itere sobre cada uno de estos :
→ `const carrito=['producto 1', 'producto 2 ', 'producto 3', 'Producto Nuevo'];`

- Todos los iteradores se construyen con un código similar a este
→ `function crearIterador(carrito){`

```
•  
    let i=0;  
•  
    return{  
•      siguiente:()=>{  
•        const fin=(i>= carrito.length);  
•        const valor= !fin ? carrito[i++] : undefined;  
•        return{  
•          fin,  
•          valor  
•        }  
•      }  
•    }  
•  }  
→ }
```

- Utilizando el iterador:
→ `const recorrerCarrito = crearIterador(carrito);`
- → `console.log(recorrerCarrito.siguiente());`
• `console.log(recorrerCarrito.siguiente());`
• `console.log(recorrerCarrito.siguiente());`
• `console.log(recorrerCarrito.siguiente());`
→ `console.log(recorrerCarrito.siguiente());`

192. Generadores en JavaScript

- Implementación de iteradores
- Un generador es una función que retorna un iterador
- Palabra reservada yield es para los valores que se pueden iterar
- Crear generador:
 - `function *crearGenerador(){`
 - `yield 1;`
 - `yield 'Juan';`
 - `yield 3+3;`
 - `yield true;`
 - `}`
- `next()`(permite ir iterando sobre el generador, si lo utilizas el generador despierta y despues vuelve a suspended) y `suspended`(el generador se queda dormido porque no lo estamos utilizando) estan relacionados
- Despertar generador:
 - `console.log(iterador.next())// despertar iterador`
 - Acceder al valor
 - `console.log(iterador.next().value);`
- Cada que pones `next()` se va al siguiente elemento `yield`, no importa para que utulices el `next`
- Cuando termina de llamar todos los `yield` y se vuelve a poner un `next()`, el generador se cierra `<closed>`
- `Done` va a ser igual a falso y se va a ejecutar con ese valor, hasta que ya no haya valores, solo así cambia de estado a `true` y cierra el generador
- Todo lo anterior con los `yield` que yo defini

- Si tenemos un arreglo con un carrito de compras y queremos utilizar un generador para ir iterando sobre los diferentes elementos

→ `const carrito= ['Producto1', 'Producto 2', 'producto 3'];`

- Creando generador

→ `function *generadorCarrito(carrito){`

•

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

- `yield carrito[i];`

•

`}`

→ `}`

- Utilizando generador:

→ `const iterador= generadorCarrito(carrito);`

- Iterando:

→ `console.log(iterador.next());`

193. Iteradores en JavaScript

- Entries iterator
 - Iterador entry: agrega una llave si no existe, e imprime llave valor (imprime todos los valores porque itera)
- `const ciudades = ['Londres', 'New York', 'Madrid', 'Paris'];`
-----nombre del arreglo (llave es el indice), set(con el set, llave y valor tienen el mismo valor) o map(llave valor)
- `for (let entry of ciudades.entries()){`
- `console.log(entry);`
- `}`
- Values Iterator : solo imprime los valores (lo mismo para arreglos, set, map)
- `for (let value of ciudades.values()){`
- `console.log(value);`
- `}`
- Keys iterator: imprime las llaves o sea las posiciones pero con set muestra los valores ya que las llaves son iguales a los valores
- `for (let keys of ciudades.keys()){`
- `console.log(keys);`
- `}`
- Cada uno de los anteriores tiene un iterador por default
 - El iterador por default de un map es el entries, el de un set y de un arreglo es values
 - Sintaxis para iterador por default :
- -----de donde extrae los valores
- `for (let ciudad of ciudades){`
- `console.log(ciudad);`
- `}`
- La linea se llama ciudad

Sección 28: Módulos en JavaScript

194. Básicos de los Modulos en ES6

- Módulos se utilizan en librerías y frameworks modernos
- Prevenir que se mezclen las diferentes variables, puedes colocar tu código en un iife o una función que se ejecuta inmediatamente. (mantiene las variables localmente en un archivo y no se mezclan con otro archivo)
- Si pongo un window dentro de un iife va a ser una variable global (librerías para no tener que importar nada) y se puede acceder desde otro archivo js, el problema es que llenamos la ventana global de código innecesario por lo que es ahí donde los módulos son bastante útiles
- IIFE:
 - `(function () {`
 - `console.log('Desde un IIFE');`
 - `window.nombreCliente='Juan'; //crea la variable globalmente`
 - `})(); //() es el que manda a llamar la función inmediatamente`
- Exportar variable para poderla importar en otros archivos
 - `export const nombreCliente='Juan'; //en un archivo`
 - Importar variable
 - `import {nombreCliente, ahorro} from './cliente.js' // en el otro archivo(archivo modulo)`
 - ./ indica que esta en la misma carpeta
 - hay 2 variable importadas
- Declarar el archivo modulo en el html
 - `<script src="js/app.js" type="module"></script> // type="module"`

195. Exportar e Importar Funciones

- Exportar una función
 - `export function mostrarInformacion(nombre, ahorro){`
 - `return `Cliente: ${nombre} - Ahorro: ${ahorro}`;`
 - `} //en un archivo`
- Importar una funcion
 - `import {nombreCliente, ahorro, mostrarInformacion, tieneSaldo} from './cliente.js' // desde archivo modulo`
 - `console.log(mostrarInformacion(nombreCliente, ahorro)); //llamando la función desde archivo modulo`
- Visual importa en automático las funciones

196. Exportar e Importar una clase

- Exportar una clase:
 - `export class Cliente{`
 - `constructor(nombre, ahorro){`
 - `this.nombre=nombre;`
 - `this.ahorro=ahorro;`
 - `}`
 - `mostrarInformacion(){`
 - `return `Cliente: ${this.nombre} - Ahorro: ${this.ahorro}``
 - `}`
 - `} // un archivo`
- Importar una clase:
 - `import{nombreCliente, ahorro, mostrarInformacion, tieneSaldo, Cliente} from './cliente.js' // en archivo modulo`
- Instanciando una clase:
 - `const cliente=new Cliente(nombreCliente, ahorro); // en archivo modulo`
- Llamando una clase con su metodo:
 - `console.log(cliente.mostrarInformacion()); // en archivo modulo`

197. Heredar una clase que esta siendo importada

- Como heredar una clase que estoy exportando desde cliente, heredarla en el archivo empresa y hacerla disponible también en el app.js
- La importamos al archivo empresa:
→ `import {Cliente} from './cliente.js';`
- Creamos una clase en archivo empresa que hereda cosas de clase Cliente y la hacemos disponible para donde decida utilizarla con export
→ `export class Empresa extends Cliente{`
- `constructor(nombre, ahorro, categoria){`
- `super(nombre, ahorro); // lo que hereda de Cliente(constructor)`
- `this.categoria=categoria;`
- `}`
- `mostrarInformacion(){`
- `return `Cliente: ${this.nombre}- Ahorro: ${this.ahorro}- Categoria:${this.categoria}`;`
- `}`
→ `}`
- Importamos la clase nueva de Empresa en el app.js (Recomendación: colocar todos tus imports en la parte superior)
→ `import{ Empresa} from './empresa.js';`
- La instanciamos
→ `const empresa=new Empresa ('Codigo con Juan', 100, 'Aprendizaje en linea'); //en archivo app.js`
- La llamamos :
→ `console.log(empresa.mostrarInformacion()); // en archivo app.js`

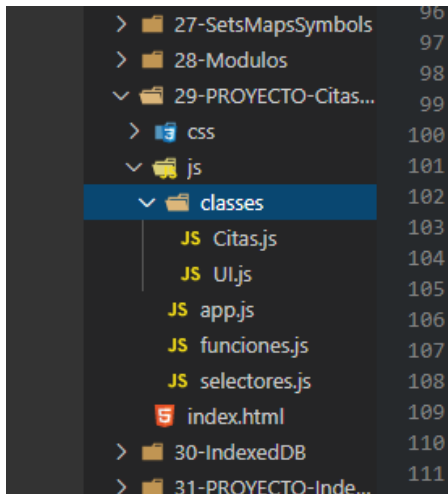
198. El Export Default y alias a los imports

- Export default, a la hora de importarlo no requiere estar dentro de las llaves
 - Los que están en las llaves son los export normales, el que esta por fuera es el export default
 - Solamente puede existir un export default, no puedes tener dos, le puedes dar un alias al importarlo y te lo va a reconocer, de hecho al declararlo puede ir sin nombre e igual te lo reconoce, porque solo existe uno en todo el archivo .
 - Creando export default en cliente.js
- ```
→ export default function nuevaFuncion(){
```
- `console.log('Este es el export default');`
- ```
→ }
```
-
- Importando en app.js
- ```
→ import nuevaFuncion, {nombreCliente, ahorro, mostrarInformacion, tieneSaldo, Cliente} from './cliente.js'
```
- 
- Llamando:
- ```
→ nuevaFuncion();
```
-
- Los export normales al importarlos también pueden tener un alias y así ser llamados en el archivo de importación
- ```
→ import nuevaFuncion, {nombreCliente as caca, ahorro, mostrarInformacion, tieneSaldo, Cliente} from './cliente.js'
```

## **Sección 29: PROYECTO: Agregando Modulos al Proyecto de Citas**

# 199. Creando los Diferentes Archivos

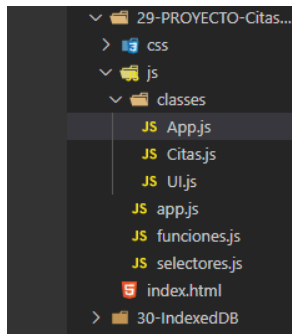
- Se puede exportar en la parte inferior (hacerlo disponible en los demás archivos)  
→ `export default Citas;`
- Dividimos nuestro proyecto en los siguientes archivos:



- Para UI y Citas con `export default` al final, para selectores con `export` en cada línea, no pasamos los selectores que se ocupan dentro de funciones, para el archivo funciones, pasamos todo lo que ocupan las funciones, las instancias las pusimos en la parte superior, y solo exportamos las funciones ya que las variables y las instancias no se exportan porque se ocupan dentro de este mismo archivo.

# 200. Últimos Ajustes

- Como importar las funciones para que todo funcione bien
- Vamos a ir abriendo cada archivo e importar lo que requiere
- Salirme de una carpeta para jalar algo de carpeta superior  
→ `import{ eliminarCita, cargarEdicion} from '../funciones.js';`
- Entrar en una carpeta:  
→ `import Citas from './classes/citas.js';`
- Definir codigo app.js en su propia clase (algo que eche a andar todo el contenido) > creamos carpeta App.js



- En esa carpeta creamos la clase: colocando los adEventListeners en su propio metodo

- Y ya no requiero la función del app.js original

```
→ class App{
• constructor(){
•
• this.initApp();
•
• }
•
• initApp(){
•
• mascotaInput.addEventListener('change', datosCita);
• propietarioInput.addEventListener('change', datosCita);
• telefonoInput.addEventListener('change', datosCita);
• fechaInput.addEventListener('change', datosCita);
• horaInput.addEventListener('change', datosCita);
• sintomasInput.addEventListener('change', datosCita);
•
• //formulario para nuevas citas
• formulario.addEventListener('submit', nuevaCita); //se puede colocar aquí tambien
• }
→ } //ya no requiero la funcion anterior
```

- E importamos los selectores y las funciones que faltan para que el código funcione

- Ahora en el app.js original importamos App e instanciamos y eso echa a andar todo el código

```
→ import App from './classes/App.js'
```

```
•
//instanciando
```

```
→ const app=new App();
```

- Se recomienda mas utilizar módulos cuando son reglas, es decir en los frameworks y librerías modernas y cuando es un proyecto grande

# **Sección 30: IndexedDB - Una base de datos real en JavaScript**



# 201. Creando la base de datos

- A diferencia de localStorage si es una BD real
- Entrar a indexDB para ver información de tu BD:  
→ Inspeccionar>application>indexDB
- En firefox development edition  
→ inspeccionar>>> almacenamiento> DB indexada
- En consola para ver todos los metodos que hay para indexDB  
→ window.indexDB
- Crear una base de datos.  
→ 

```
let crmDB = window.indexedDB.open('crm', 1);
```

```
-----version
-----nombre
```
- Si hay un error en la BD  
→ 

```
crmDB.onerror=function(){
• console.log('Hubo un error a la hora de crear la BD');
→ }
```
- Si se creo bien  
→ 

```
crmDB.onsuccess=function(){
• console.log('Base de datos Creada!');
→ }
```
- Cuando creas una BD creas columnas y ese codigo para crearlas se ejecuta una sola vez , porque si no reescribiria la info que tenemos (configuracion de la BD)  
→ 

```
crmDB.onupgradeneeded=function(){
• console.log('Este método solo se ejecuta una vez...')
→ } // buen metodo para definir las diferentes columnas
```

# 202. Creando las Tablas

- Ver BD que hemos creado
- `crmDB.onupgradeneeded=function(e){`
- `console.log(e.target.result);`
- `}`
- objectStore permite crear las columnas de nuestra BD
- No se recomienda almacenar password en una BD de este tipo
- Configuración de la BD
- `crmDB.onupgradeneeded=function(e){`
- `const db= e.target.result;`
- `-----base de datos`
- `const objectStore = db.createObjectStore( 'crm', {`
- `keyPath: 'crm',`
- `autoIncrement: true // objetos de configuración // va incrementando el id`
- `});`
- `//Definir las columnas`
- `objectStore.createIndex('nombre', 'nombre', {unique:false} );`
- `-----como vamos a hacer referencia para consultar la tabla del nombre (keyPath)`
- `-----opciones unique:false porque podemos tener 2 clientes que se llamen igual, y no son la misma persona, se pone true si es único, como email que solo puede tenerlo una persona`
- `console.log('Columnas creadas');`
- `}`
- `}`

# 203. Creando un Nuevo Cliente por Medio de una transacción

- Agregar registros, insertar registros en nombre, email y teléfono
- Para poder trabajar con las diferentes operaciones de una BD en indexdB, utilizas transacciones (cuando se completan correctamente pasos)

- Crear transacciones:

→ *function* crearCliente(){

```
•
 let transaction =DB.transaction(['crm '], 'readwrite');
•
 -----BD
•
 -----Modo tambien E readonly
•
 transaction.oncomplete=function() {
•
 console.log('transaccion completada'); // si se completa
•
 }
•
 transaction.onerror=function() {
•
 console.log('hubo un error en la transaccion');
•
 }
→ }
```

- Utilizar las transacciones :
- Crear un objeto en nuestra BD

```

→ let DB; // en alguna parte del codigo
→ B=crmDB.result; //resultado de la creacion de BD // en alguna parte del codigo

→ function crearCliente(){
•
• let transaction =DB.transaction(['crm'], 'readwrite');
• -----BD
• -----Modo tambien E readonly
• transaction.oncomplete=function(){
• console.log('transaccion completada'); // si se completa
• }
•
• transaction.onerror=function(){
• console.log('hubo un error en la transaccion');
• }
•
• //escribir un objeto en nuestra BD
• const objectStore= transaction.objectStore('crm');
•
• //creando objeto
• const nuevoCliente={
•
• telefono: 123455566,
• nombre: 'Julieta',
• email: 'correo@correo.com'
• }
•
• //Agregar nuevo cliente a BD
• const peticion = objectStore.add(nuevoCliente)
• -----si quieres actualizar pones .put y para eliminar .delete
• console.log(peticion);
→ }

```

- Para las diferentes operaciones, ya sea obtener registros, eliminarlos, actualizarlos, o agregarlos, siempre indexedDB utiliza lo que son las transacciones
- Todos los métodos de lo que se conoce como el krod, los soporta correctamente indexedDB

# **Sección 31: PROYECTO: Administrador de Citas con Indexe...**

# 204. El Proyecto Finalizado

- Como agregarle indexDB al proyecto de la citas
- Los datos de las citas se van a almacenar en indexDB
- IndexDB es una version mejorada de localStorage pero ofrece mucho mas porque es una BD real
- Si recargo no desáparecen las citas

# 205. Creando la Base de datos

- Otra forma de ejecutar una función cuando carga el html
- `window.onload={() => {`
- `console.log('Documento Listo'); //tambie se ejecuta cuando carga el html`
- `}`
- KeyPathes un índice, en este caso va a ser el id



# 206. Agregando una Cita a la BD

- Como agregar registros desde un formulario
  - Hacemos el procedimiento normal pero al agregar al objectStore, agregamos el objeto que contiene los datos del formulario  
→ `objectStore.add(citaObj);`
  - Al declarar las funciones de oncomplete también se puede utilizar la sintaxis de arrow function
  - Tambien puede ir en este orden (oncomplete)
- ```
→ //Insertar Registro en IndexedDB
•   const transaction = DB.transaction(['citas'], 'redwrite');
•
•   // escribir objeto en BD // habilitar el objectStore
•   const objectStore=transaction.objectStore('citas');
•
•   //insertar en la BD
•   objectStore.add(citaObj);
•
•   transaction.oncomplete= function(){
•       console.log('Cita Agregada');
•
•       // Mostrar mensaje de que todo esta bien...
•       ui.imprimirAlerta('Se agregó correctamente')
→ }
•
•   Vemos que se guarda en indexDB pero en la pagina al recargar ya no se muestra la cita por lo tanto sig video
```

207. Mostrar las Citas al cargar

- Que Cuando yo recargue tome las citas de indexDB y las muestre en el html
 - Ya no pasamos las citas para imprimir en el html
 - En la funcion imprimirCitas tenemos que leer todo lo que tenemos en la BD
 - Leer citas de BD
 - Instancia de objectStore de citas (obtener todos los registros) leer el contenido de la BD
- `const objectStore=DB.transaction('citas').objectStore('citas');`
- Iterando sobre los registros y traer lo que tenemos en las columnas (va imprimiendo los datos)
- `// itera sobre el contenido de la BD e imprime valores`
- `objectStore.openCursor().onsucces= function(e){`
 - `console.log(e.target.result);`
- `}`
- Mandar a llamar cuando el documento este listo.
 - Una vez que creo mi BD puedo mandar a llamar `ui.imprimirCitas()`, porque asi ya estamos seguros de que estamos leyendo indexDB (indexDB ya esta listo)

```

→ function crearDB (){
  • //crear la base de datos en version 1.0
  • const crearDB = window.indexedDB.open('citas', 1); //conexion
  • -----nombre BD
  •
  • //si hay un error
  • crearDB.onerror=function(){
  •   console.log('Hubo un error');
  • }
  •
  • // si todo sale bien
  • crearDB.onsuccess=function(){
  •   console.log('BD Creada');
  •
  •   DB=crearDB.result;
  •
  •   //Mostrar citas al cargar (pero indexedDB ya esta listo)
  •   ui.imprimirCitas();
→ }

```

- Cita antes era el objeto actual del forEach , ahora es cursor.value (linea 126)

- Esta cosa es un iterador pero tienes que indicarle dentro de él para que siga iterando después de que ejecute la primera iteración de código

```
→ objectStore.openCursor().onsuccess= function(e){
    const cursor = e.target.result;
    if(cursor){
        //ve al siguiente elemento
        cursor.continue();
    }
}
```

- Ver propiedades del contenido de la BD

```
→ const total=objectStore.count();
→ console.log(total);
```

- Para entrar a resultados si le pones

```
→ console.log(total.result); // no te deja entrar
```

- Por lo tanto tienes que poner para entrar lo siguiente:

```
→ total.onsuccess=function(){
    console.log(total.result);
}
```

- Si pones esto no manda llamar la función textoHeading(); porque se queda el código dentro del propio ciclo

→ `total.onsuccess=function(){`

- `console.log(total.result);`

- `this.textoHeading();`

→ `}`

- Hacemos la función global

→ `const fnTextoHeading=this.textoHeading; //la hacemos global`

- Y la llamamos

→ `total.onsuccess=function(){`

- `console.log(total.result);`

- `fnTextoHeading(total.result);`

→ `}`

208. Editar Citas

- Para editar cita
- ```
if(editando){
```
- ```
    // Estamos editando
```
- ```
 administrarCitas.editarCita({...citaObj});
```
- 
- ```
    //Edita en indexDB
```
- ```
 const transaction=DB.transaction(['citas'], 'readwrite');
```
- ```
    const objectStore=transaction.objectStore('citas');
```
- ```
 objectStore.put(citaObj); // permite editar un registro
```
- 
- ```
    transaction.oncomplete=()=>{
```
- ```
 ui.imprimirAlerta('Guardado Correctamente');
```
- ```
        formulario.querySelector('button[type="submit"]').textContent = 'Crear Cita';
```
-
- ```
 editando = false;
```
- ```
    }
```
-
- ```
 transaction.onerror=()=>{
```
- ```
        console.log('Hubo un error');
```
- ```
 }
```
- 
- ```
    } else {...
```
- Tambien cambiamos en el codio cita por cursor.value (linea 161) pero se repite la cita al edita, por lo tanto:
- ```
const cita=cursor.value; // para que cambie cada que edito diferente cita, si no se repite misma cita al editar porque es dinamico
```
- ```
btnEditar.onclick = () => cargarEdicion(cita);
```

209. Eliminar Citas

- Destruir un registro desde la BD
- ```
function eliminarCita(id) {
```
- ```
    const transaction=DB.transaction(['citas'], 'readwrite');
```
- ```
 const objectStore=transaction.objectStore('citas');
```
- ```
    objectStore.delete(id);
```
- ```
 transaction.oncomplete={()=>{
```
- ```
        console.log(`Cita ${id} eliminada...`);
```
- ```
 ui.imprimirCitas()
```
- ```
    }
```
- ```
 transaction.onerror={()=>{
```
- ```
        console.log('Hubo un error');
```
- ```
 }
```
- ```
}
```
- 4 operaciones que todo sistema debe de realizar Crod :
- Create
- Read
- Update(actualizar)
- Delete
- registros

Sección 32: PROYECTO: CRM con IndexedDB

210. El Proyecto Finalizado

- Como crear un proyecto que solamente utilice indexDB
- Crear un CRM herramienta para dar seguimiento a nuestros clientes
- Tenemos listado de clientes y podemos dar de alta a un nuevo cliente
- Tendremos validación
- Se pueden editar los clientes y eliminarlos
- Ventana emergente de si esta seguro de eliminarlo

211. Creando la base de datos

- IndexDB no supe bases de datos reales como SQL mySQL entre otras ya que estas nos ofrecen mas cosas
- Crearemos las 4 operaciones del crod por medio de indexDB
- Al agregar las columnas de BD agregamos una para el id
- **EJercicio) esto a modulos**

212. Primeros pasos para Agregar Nuevos Clientes

- Crear del crod, es decir insertar nuevos registros (nuevo cliente)
 - El codigo para conectar con una BD es el mismo que para crearla (si ya E la conecta, si no E la crea)
 - Para imprimir en consola un mensaje de error:
- `console.error('error');`

- Para que al mostrar las alertas no se llene de alertas:

```
→ const alerta=document.querySelector('.alerta');
```

```
if(!alerta){  
  
    //crear alerta  
    const divMensaje=document.createElement('div');  
    divMensaje.classList.add('px-4', 'py-3', 'rounded', 'max-w-log', 'mx-auto', 'mt-6', 'text-center', 'border', 'alerta'); // tailwind  
  
    if(tipo==='error'){  
        divMensaje.classList.add('bd-red-100', 'border-red-400', 'text-red-700');  
  
    }else{  
        divMensaje.classList.add('bg-green-100', 'border-green-400', 'text-green-700');  
    }  
  
    divMensaje.textContent= mensaje;  
    formulario.appendChild(divMensaje);  
  
    setTimeout(() => {  
        divMensaje.remove();  
    }, 3000);  
}
```

213. Agregando Nuevos Clientes en la BD

- Insertarlo en la BD
 - Como agregar un registro a la BD
 - Abrir una conexión y por medio de transacciones agregar o colocar ese nuevo registro
 - Para crear un objeto con la info vamos a utilizar el object literal enhancement
- ```
→ const cliente={
• nombre,
• email,
• telefono,
• empresa, //ya toman los valores porque se igualan a leer inputs de arriba, como se llaman igual ya no se pone llave
 // valor
→ }
```
- Agregar un nuevo registro a BD :
- ```
• function crearNuevoCliente(cliente){  
•     const transaction = DB.transaction(['crm'], 'readwrite');  
•  
•     const objectStore=transaction.objectStore('crm');  
•  
•     objectStore.add(cliente);  
•  
•     transaction.onerror=function(){  
•         console.log('Hubo un error');  
•     };  
•  
•     transaction.oncomplete=function(){  
•         console.log('Cliente agregado')  
•     }  
→ }
```

- Cuando se agregue el nuevo cliente, me lleve a lapagina de clientes

→ `setTimeout(() =>{`

- `window.location.href='index.html'; // despues de 3 seg se va a index html`

→ `}, 3000);`

- Enlistar los clientes
- Ejercicio enlistar los clientes:
- Pag 44

214. Query a la BD para obtener los
clientes

- Ejecutar una funcion en caso de que exista la BD

```
→ if(window.indexedDB.open('crm', 1)){
```

- obtenerClientes();

```
→ }
```


- Abrir una conexión para saber si se hace correctamente o existe un error con la base de datos
- <https://gist.github.com/juanpablogdl/7e31b69b50e837c25ea6b9beabaa4c63> // para listado clientes

- Agregar todos los elementos de la BD al HTML

```
→ listadoClientes.innerHTML += `
```

- <tr>
- <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200">
- <p class="text-sm leading-5 font-medium text-gray-700 text-lg font-bold"> \${nombre} </p>
- <p class="text-sm leading-10 text-gray-700"> \${email} </p>
- </td>
- <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 ">
- <p class="text-gray-700">\${telefono}</p>
- </td>
- <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 leading-5 text-gray-700">
- <p class="text-gray-600">\${empresa}</p>
- </td>
- <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 text-sm leading-5">
- Editar
- Eliminar
- </td>
- </tr>

```
→ `;
```


- `<tr>` define una fila de celdas en una tabla. Html

215. Query para obtener cada cliente

- Trabajar con la edición para editar cliente
- `<td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 text-sm leading-5">`
- `<a href="editar-cliente.html?id=${id}" class="text-teal-600 hover:text-teal-900 mr-`
`5">Editar`
- `Eliminar`
- `</td>` // enlace que lleva a editar cliente y luego pasa el parámetro id a la url con el signo ?, se conoce como query String

- Leer el valor de la URL para editar.
 - Tiene su propio archivo html editar cliente
 - Ver la parte de query String después del signo de interrogación, no da el archivo en el cual estamos solo muestra la info id
- `//verificar el ID de la URL`
- `const parametrosURL = new URLSearchParams(window.location.search);`
- `//Para obtener el id:`
- `const idCliente = parametrosURL.get('id');`
-
- Si hago esto sale un error porque tarda en verificar
- `function obtenerCliente(id){`
- `const transaction= DB.transaction(['crm'], 'readwrite');`
 - `const objectStore= transaction.objectStore('crm');`
 - `console.log(objectStore);`
- `}`
- Por lo tanto antes de llamar la funcion esperamos 1 segundo:
- `if(idCliente){`
- `setTimeout(() =>{`
 - `obtenerCliente(idCliente);`
 - `}, 1000);`
- `} //(tambien se soluciona con promises o asycaway`

- Seleccionar un solo id y llenar el formulario para editar

```
→ cliente.onsuccess= function(e){
•     const cursor = e.target.result;
•
•     if(cursor){
•         if(cursor.value.id=== Number(id)){
•             llenarFormulario(cursor.value);
•         }
•
•         cursor.continue();
•
•     }
→ }
```

- Llenar el formulario al editar:

```
• //variable global:
→ const nombreInput= document.querySelector('#nombre');
→ function llenarFormulario(datosCliente){
•
•     const {nombre}= datosCliente;
•     nombreInput.value=nombre;
→ }
```

216. Llenar el Formulario con la información del Cliente

- Llenar los demás campos y actualizar en la BD
- Crear un listener para cuando edite algo en el formulario
- Crear un archivo para poner las funciones que se comparten en los demás archivos, y así no reescribir código, pero eso lo tenemos que poner en el html de nuevo-cliente, lo mismo para editar-cliente (no importa el orden)
- Si el id viene como string y en la BD esta como numero, en el objeto donde estamos actualizando la info tenemos que convertir el id a numero o E error
- Actualizar en BD:
 - `const transaction=DB.transaction(['crm'], 'readwrite');`
 - `const objectStore=transaction.objectStore('crm');`
 - `objectStore.put(clienteActualizado);` // como el keypath es el id, busca, ve los que son iguales y actualiza los nuevos datos

217. Eliminar Clientes de la BD

- Eliminar registros
- Con innerHTML hacer el onclick no se puede utilizar
- Por lo tanto mandamos const a global
- `const listadoClientes=document.querySelector('#listado-clientes');`
- Agregamos un listener en la funcion de listeners, donde al hacer click se ejecuta una funcion
- `listadoClientes.addEventListener('click' , eliminarRegistro); //(delegation)`
- En el script del super textote del html agregamos una clase extra de eliminar y con delegation revisamos, si el elemento al que le estoy dando click tiene esa clase, entonces significa que el usuario presiono en este boton
- `Eliminar`
- Obtener la clase a la que le estoy dando click
- `function eliminarRegistro(e){`
- `console.log(e.target.classList);`
- `}`

- Eliminar un registro de BD
- ```
function eliminarRegistro(e){
```
- 
- ```
    if(e.target.classList.contains('eliminar')){
```
- ```
 const idEliminar = Number(e.target.dataset.cliente);
```
- 
- ```
        //ventanita nativa
```
- ```
 const confirmar=confirm('Deseas eliminar este cliente? ')
```
- 
- ```
        console.log(confirmar); // true or false
```
- ```
 if (confirmar){
```
- ```
            const transaction= DB.transaction(['crm'], 'readwrite');
```
- ```
 const objectStore= transaction.objectStore('crm');
```
- 
- ```
            objectStore.delete(idEliminar);
```
-
- ```
 transaction.oncomplete=function(){
```
- ```
                console.log('Eliminando...')
```
-
- ```
 e.target.parentElement.parentElement.remove(); //traversing para eliminar en html//Traversing, irse al padre y luego al padre para eliminar el elemento
```
- ```
                //en el DOM una vez que se completa la transaccion ( se elimina de la lista, después de que se da click)
```
-
- ```
 };
```
- 
- ```
            transaction.onerror=function(){
```
- ```
 console.log('hubo un error');
```
- ```
            } // eliminamos en BD
```
- ```
 }
```
- 
- ```
    }
```
-
- ```
}
```
- Para que funcionara el programa, solo pusimos
- ```
Let DB; // en la carpeta de funciones y la eliminamos de los demás archivos
```

Sección 33: Promises, Callbacks y Programación Asíncrona en JS

218. Ejemplo de Callbacks

- Supongamos que estamos descargando un listado de clientes, y en lo que estoy descargando los últimos 10 clientes, se agrega uno nuevo, el nuevo cliente no va a ser parte de mi descarga por lo tanto callback
- Mostrar nuevo elemento
- Si se utilizan demasiado sucede el callback hell

→ *function* nuevoPais(*pais*, *callback*){

- -----funcion que se va a ejecutar cuando nuevoPais sea mandado llamar
- setTimeout(() => {
- países.push(pais);
- callback();
- }, 2000);

→ }

- Llamando nuevo pais

→ nuevoPais('Alemania', mostrarPaíses);

219. El muy exagerado Callback Hell

- Cuando implementas mal los callbacks (siempre se exagera) , se hace una curva en el código y no es recomendable

```
→ function iniciarCallbackHell(){  
  •   setTimeout(()=>{  
  •     nuevoPais('Alemania', mostrarPaises);  
  •     setTimeout(()=>{  
  •       nuevoPais('Francia', mostrarPaises);  
  •       setTimeout(()=>{  
  •         nuevoPais('Inglaterra', mostrarPaises);  
  •       }, 3000);  
  •     }, 3000);  
  •   }, 3000);  
→ }
```

220. Creando un Promise y .then y .catch

- Si entro a la API de la empresa y comienzo a descargar los ultimos clientes, una vez que se han descargado todos los clientes, ese promise se cumple. Si por algo intento entrar al sistema y esta caido el servidor no se cumple el promise porque no pude descargar los clientes
- // declarando promise
- `const aplicarDescuento = new Promise ((resolve, reject) =>{`
 - -----nombre promise
 - -----
 - //lo que se va a ejecutar cuando el promise se ejecuta correctamente o cuando se cumple el promise
 - -----cuando tenemos un error en el promise
 - `const descuento = true ;`
 -
 - `if (descuento){`
 - `resolve('Descuento Aplicado')`
 - `}else {`
 - `reject('No se pudo aplicar el descuento')`
 - `}`
- `})`

- `console.log(aplicarDescuento);`
- `//Hay 3 valores posibles...`
- `//fulfilled- el promise se cumplio`
- `//rejected- El promise no se cumplio`
- `//pending- no se ha cumplido y tampoco fue rechazado (no he definido solve ni reject)`
- Como acceder a descuento aplicado para poderlo mostrar o para poder hacer alguna operación (ver el resultado del promise, accedes a la respuesta) para ver el resolve
- `aplicarDescuento`
- `.then(resultado => {`
- `console.log(resultado);`
- `})`
- Se hace lo mismo cuando se utiliza `fetchAPI` QUE ES una forma en la que vamos a poder consumir datos de alguna API que esta en otro servidor, podemos consumir esos datos y mostrarlos (la conexión es correcta, pude descargar esos datos por lo tanto se cumple el promise y lo mando a la consola)
- Para sacar lo que esta en el reject . `catch`
- `aplicarDescuento`
- `.then(resultado => {`
- `console.log(resultado);`
- `})`
- `.catch(error => {`
- `console.log(error);`
- `})`

- Se pueden borrar llaves por el arrow y mandar llamar funciones
- aplicarDescuento
- `.then(resultado => descuento(resultado))`
 - `.catch(error => console.log(error))`
 - `function descuento(mensaje){`
 - `console.log(mensaje);`
- `}`

221. De Callback Hell a Promises

- La funcion que nos hace la curva es la que va a ser nuestro promise
 - Pasando a promises codigo de video anterior
- `const paises=[];`
- - `const nuevoPais= pais => new Promise(resolve =>{`
 - `setTimeout(() => {`
 - `paises.push(pais);`
 - `resolve(`Agregado: ${pais}`)`
 - `}, 3000);`
 - `})`
 -
 - `nuevoPais('Alemania')`
 - `.then(resultado => {`
 - `console.log(resultado);`
 - `console.log(paises);`
 - `return nuevoPais('Francia')// volver a llamar de nuevo el promise`
 - `})`
 - `.then(resultado => {`
 - `console.log(resultado)`
 - `console.log(paises);`
 - `return nuevoPais('Inglaterra')`
 - `})`
 - `.then(resultado => {`
 - `console.log(resultado);`
 - `console.log(paises)`
- `})`

Sección 34: API's en JavaScript

222. Notification API

- Apis nativas en JavaScript
- Codigo de las notificaciones nativas en javascript
- Para poder enviarle notificaciones a los usuarios ellos tienen que aceptar recibir notificaciones por eso un boton para notificarme, y otro para ver las notificaciones
- Todas las apis modernas de JavaScript o casi todas utilizan promises, esa es la tendencia

- Notificaciones codigo

```
→ notificarBtn.addEventListener('click', ()=>{  
  • Notification // API de notificaciones  
  • .requestPermission() // correr metodo que le va a preguntar al usuario si quiere dar permiso para recibir notificaciones  
  • .then ( resultado => { // promise  
  •   console.log('El resultado es', resultado);  
  • })  
→ });
```

- Granted : cuando el usuario acepta las notificaciones

- Ver la notificacion:

```
→ const verNotificacion= document.querySelector('#verNotificacion');  
• verNotificacion.addEventListener('click', ()=> {  
  • if(Notification.permission==='granted'){  
  •   new Notification('Esta es la notificacion', {  
  •     icon: 'img/ccj.png', //agregar un icono personalizado a la notificacion  
  •     body:'Codigo con Juan, aprende con proyectos reales' // agrega texto a notificacion  
  •   });  
  • }  
→ });
```


- Al dar click a esa notificación lleve a alguien a ese sitio web o a otro sitio web
- `const verNotificacion= document.querySelector('#verNotificacion');`
- `verNotificacion.addEventListener('click', ()=> {`
- `if(Notification.permission==='granted'){`
- `const notificacion= new Notification('Esta es la notificacion', {`
- `icon: 'img/ccj.png',//agregar un icono personalizado a la notificacion`
- `body:'Codigo con Juan, aprende con proyectos reales'// agrega texto a notificacion`
- `});`
- `notificacion.onclick= function (){`
- `window.open('https://www.codigoconjuan.com')`
- `}`
- `}`
- `});`
- Muchas aplicaciones modernas utilizan el envío de notificaciones push nativas, no requieres una librería externa para poderlo implementar en tu sitio web
- No enviar notificaciones a los usuarios todo el tiempo preguntando si quieren recibir notificaciones
- A veces los antivirus bloquean las notificaciones

223. Intersection Observer

- Permite identificar cuando un elemento esta visible, Api nativa de javascript
 - Permite identificar un elemento una vez que es visible en el view port del navegador
 - Cada que este o no este visible nos va a reportar en consola
 - `isIntersecting: true`
 - `isIntersecting: false`
- ```
→ document.addEventListener('DOMContentLoaded', () => {
•
• // habilitamos intersection observer
• const observer = new IntersectionObserver(entries =>{
• console.log(entries[0]);
• });
•
• // le decimos que elemento vamos a observar y cuando elemento este visible o no este visible nos lo va a ir reportando //en el entries de arriba
• observer.observe(document.querySelector('.premium'));
→ })
```
- 
- Para imprimir algo cuando ya este visible:
- ```
→ document.addEventListener('DOMContentLoaded', () => {  
•  
•     // habilitamos intersection observer  
•     const observer = new IntersectionObserver( entries =>{  
•  
•         if(entries[0].isIntersecting){  
•             console.log('Ya esta visible');  
•         }  
•  
•     });  
•  
•     // le decimos que elemento vamos a observar y cuando elemento este visible o no este visible nos lo va a ir reportando en el entries de arriba  
•     observer.observe(document.querySelector('.premium'));  
→ })
```

- Útil para agregar lo que es el easy loading a las imágenes, es decir (las imágenes de hasta abajo en un sitio web, no se vean hasta que yo llegue a esa posición, también se puede usar para aplicar el scrol infinito (cuando entras a instagram das scrol sobre los post y va cargando mas)

224. Detectar si hay conexión a internet o no

- En consola si tengo internet o no:
→ `navigator.online // true/false`
- Ejemplo cuando utilizas la app de uber, estas buscando tu transporte y por algo pierdes tu conexión te dice que no estas conectado o tratas de enviar un post en facebook y te dice que no estas conectado y se envia cuando tengas una conexión
- Saber si el usuario tiene o no una conexión a internet
→ `window.addEventListener('online', actualizarEstado);`
• `window.addEventListener('offline', actualizarEstado);`
•
• *function* `actualizarEstado(){`
• `if(navigator.onLine){`
• `console.log('si estas conectado')`
• `}else{`
• `console.log('No estas conectado...')`
• `}`
→ `}`
- Base de lo que se conoce como progressive web application

225. Ejecutar pantalla completa con JS

- Agregar funcionalidad de abrir en pantalla completa o salir de pantalla completa (similar al icono de youtube)
- En consola todo el html
 - `document.documentElement` // trae todo el html
- Dependiendo del elemento que quieras hacer pantalla completa, ahí tendrías que colocar el elemento
- Yo quiero que todo el sitio corra en pantalla completa
 - `const abrirBtn= document.querySelector('#abrir-pantalla-completa');`
 - `const salirBtn=document.querySelector('#salir-pantalla-completa');`
 - - `abrirBtn.addEventListener('click', pantallaCompleta);`
 - `salirBtn.addEventListener('click', cerrarPantallaCompleta);`
 - - `function pantallaCompleta(){`
 - `document.documentElement.requestFullscreen();` // abrir pantalla completa
 - `}`
 - - `function cerrarPantallaCompleta(){`
 - `document.exitFullscreen();` // cerrar pantalla completa
 - `}`

226. Detectar cuando estamos viendo la página web actual

- En consola
- → `document.visibilityState // visible`
- Si esta o no visible una pagina

- Cuando cambio se ejecuta el hidden, cuando vuelvo se ejecuta el visible
- `document.addEventListener('visibilitychange', () => {`
- `----- cambia la visibilidad de la pagina`
- `console.log(document.visibilityState);`
- `})`

- Pausar o reproducir video por ejemplo
- `document.addEventListener('visibilitychange', () => {`
- `----- cambiar la visibilidad de la pagina`
- `if(document.visibilityState==='visible'){`
- `console.log('Ejecutar la función para reproducir el video...');`
- `}else{`
- `console.log('Pausar el video');`
- `}`
- `})`

- Hay extensiones que previenen que eso pase agregando un global que siempre tenga `visibilityState` como visible
- Existen muchas apis nuevas como la de geolocalizacion para saber en que lugar estan tus usuarios y si se estan moviendo, ideal para crear un clon de uber

227. Speech API

- Speech recognition API, permite que cuando yo hable javascript pueda saber o pueda trasladar lo que yo estoy diciendo en el microfono hacia mi pagina web (alexa o google home assistant) presente y futuro en el desarrollo
- Hay diferentes etapas, una es arrancar recognition, otra es comenzar a escuchar, otra es cuando el usuario termina de hablar, y otra es cuando mostramos el resultado, es decir lo que el usuario hablo
- Confidence que tan seguro esta de lo que escucho y escribio el valor mas alto es 1
- Transcript lo que dijiste en texto

```
→ const salida= document.querySelector('#salida');
• const microfono= document.querySelector('#microfono');
•
• microfono.addEventListener('click', ejecutarSpeechAPI);
•
• function ejecutarSpeechAPI(){
•   const SpeechRecognition= webkitSpeechRecognition; // esta en ventana global
•   const recognition= new SpeechRecognition(); // instanciando
•
•   recognition.start(); // comenzar speechRecognitionAPI
•
•   // funcion que se ejecuta cuando recognition comience a ejecutarse
•   recognition.onstart = function(){
•     salida.classList.add('mostrar'); // como tiene clase de ocultar, hay que agregarle la clase de mostrar esta en app.css
•     salida.textContent='Escuchando...';
•
•   };
•
•   // cuando hallamos terminado de hablar se ejecuta una funcion
•   recognition.onspeechend= function(){
•     salida.textContent='Se dejo de grabar...';
•     recognition.stop();
•
•   };
•
•   // lo que yo halla hablado en el microfono hacia texto
•   recognition.onresult= function(e){
•     console.log(e.results); // acceder a lo que dije .results
•   }
→ }
```

- Ver solo la confianza y lo que escribí
- `const salida= document.querySelector('#salida');`
- `const microfono= document.querySelector('#microfono');`
-
- `microfono.addEventListener('click', ejecutarSpeechAPI);`
-
- `function ejecutarSpeechAPI(){`
- `const SpeechRecognition= webkitSpeechRecognition; // esta en ventana global`
- `const recognition= new SpeechRecognition(); // instanciando`
-
- `recognition.start(); // comenzar speechRecognitionAPI`
-
- `// funcion que se ejecuta cuando recognition comience a ejecutarse`
- `recognition.onstart = function(){`
- `salida.classList.add('mostrar'); // como tiene clase de ocultar, hay que agregarle la clase de mostrar esta en app.css`
- `salida.textContent='Escuchando...';`
-
- `};`
-
- `// cuando hallamos terminado de hablar se ejecuta una funcion`
- `recognition.onspeechend= function(){`
- `salida.textContent='Se dejo de grabar...';`
- `recognition.stop();`
- `};`
-
- `// lo que yo halla hablado en el microfono hacia texto`
- `recognition.onresult= function(e){`
- `console.log(e.results[0][0]); // acceder a lo que dije .results`
- `}`
- `}`

- Poner texto que escucha en pantalla y tambien la seguridad en porcentaje
- `const salida= document.querySelector('#salida');`
- `const microfono= document.querySelector('#microfono');`
-
- `microfono.addEventListener('click', ejecutarSpeechAPI);`
-
- `function ejecutarSpeechAPI(){`
- `const SpeechRecognition= webkitSpeechRecognition; // esta en ventana global`
- `const recognition= new SpeechRecognition(); // instanciando`
-
- `recognition.start(); // comenzar speechRecognitionAPI`
-
- `// funcion que se ejecuta cuando recognition comience a ejecutarse`
- `recognition.onspeechend = function(){`
- `salida.classList.add('mostrar');` // como tiene clase de ocultar, hay que agregarle la clase de mostrar esta en app.css
- `salida.textContent='Escuchando...';`
-
- `};`
-
- `// cuando hallamos terminado de hablar se ejecuta una funcion`
- `recognition.onspeechend= function(){`
- `salida.textContent='Se dejo de grabar...';`
- `recognition.stop();`
- `};`
-
- `// lo que yo halla hablado en el microfono hacia texto`
- `recognition.onresult= function(e){`
- `console.log(e.results[0][0]); // acceder a lo que dije .results`
-
- `const {confidence, transcript}= e.results[0][0];`
-
- `const speech = document.createElement('p');`
- `speech.innerHTML= 'Grabado: ${transcript}';`
-
- `const seguridad= document.createElement('p');`
- `seguridad.innerHTML=`Seguridad: ${parseFloat(confidence*100)} %`;`
-
- `salida.appendChild(speech);`
- `salida.appendChild(seguridad);`
- `}`
- `}`

Sección 35: Fetch API - Para obtener datos de otros Servidores o API's

228. Como utilizar FETCH API

- Fetch API solo puede leer texto, por ejemplo un texto plano o JSON, no soporta xml
 - Fetch API es nativo de JavaScript
 - Se pueden recibir y enviar datos
 - Utiliza promises, si el promise se cumple, entonces nos va a resolver los datos, no tienes que escribir el resolve, ya fetch tiene el resolve y el reject
- `const cargarTxtBtn= document.querySelector('#cargarTxt');`
- `cargarTxtBtn.addEventListener('click', obtenerDatos);`
 -
- ```
function obtenerDatos(){
```
- - `const url= 'data/datos.txt';`
  - `fetch(url)`
  - `----- url de donde van a venir los datos o a donde se van a enviar los datos`
  - `.then(respuesta => {`
  - `console.log(respuesta)`
  - `})`
- `}`
- En consola status 200 significa que Se logro hacer correctamente la consulta, pudo encontrar el archivo y esta todo listo para el siguiente paso, si es otro numero puede que no encontró el archivo

- Para imprimir solo el numero  
→ `console.log(respuesta.status);`
- Para que te diga ok de que todo esta bien  
→ `console.log(respuesta.statusText);`
- Ver la url que estamos consultando  
→ `console.log(respuesta.url);`
- Ver el tipo de consulta que estamos realizando // basic  
→ `console.log(respuesta.type);`
- Proto son las funciones que podemos aplicarle (JSON y TEXT) muestra la información que tenemos en la consulta que estamos realizando
- Leer el contenido de datos.txt

```

→ const cargarTxtBtn= document.querySelector('#cargarTxt');
• cargarTxtBtn.addEventListener('click', obtenerDatos);
•
• function obtenerDatos(){
•
• const url= 'data/datos.txt';
• fetch(url)
• ----- url de donde van a venir los datos o a donde se van a enviar los datos
• .then(respuesta => {
• console.log(respuesta);
• console.log(respuesta.status);
• console.log(respuesta.statusText);
• console.log(respuesta.url);
• console.log(respuesta.type);
•
• return respuesta.text();// porque es texto // la respuesta la quiero como text
• })
• .then(datos => {
• console.log(datos)
• })// entonces imprime los datos
• .catch (error => {
• console.log(error); // este seria el reject del promise si no se lo pones y algo esta mal, el código no te va a decir que esta pasando
• })
→ }

```

# 229. Consultar un JSON

- Como cargar JSON con fetch API
  - JSON es bastante similar a un objeto de JavaScript la diferencia es que las llaves estan dentro de un string
- {
- `"id" : 1,`
  - `"nombre" : "Juan",`
  - `"empresa" : "Código Con Juan",`
  - `"trabajo" : "Desarrollador Web"`
- }
- Json se conoce como una tecnologia de transporte, es decir si tienes un proyecto hecho en node etc y consultas una BD para mostrar algunos clientes y quieres agregar algo de JavaScript, lo ideal seria que la respuesta no se imprima con las funciones que tienen esas tecnologias, si no que des una respuesta como JSON, para que la puedas consumir, porque por si solo JavaScript en la forma en la que la estamos viendo no puede consultar una BD
  - Un lenguaje que entiende bien JavaScript es Json y algo que va a permitir es que puedas conectar tus aplicaciones ya sea en php es json, esos datos deben estar en JSON para que JavaScript pueda leerlos, no importa mucho el lenguaje de backend que utilizas (ruby, python, Java, php etc) , todos los json son iguales
  - JavaScript convierte JSON en objeto (se crea un lenguaje de transporte para poder comunicar un backend hecho en java con un frontend hecho en JavaScript)

- Con json
- `const cargarJSONBtn= document.querySelector('#cargarJSON');`
- `cargarJSONBtn.addEventListener('click', obtenerDatos);`
- 
- `function obtenerDatos(){`
- `const url= 'data/empleado.json';`
- `fetch(url)`
- `.then( respuesta => respuesta.json())// regresar datos en json`
- `.then( resultado => console.log(resultado))// imprimir respuesta`
- `}`
- aplicando algún scripting para mostrar en pantalla
- `function obtenerDatos(){`
- `const url= 'data/empleado.json';`
- `fetch(url)`
- `.then( respuesta => respuesta.json())// regresar datos en json`
- `.then( resultado => mostrarHTML(resultado))// imprimir respuesta`
- `}`
- `function mostrarHTML({ empresa, id, nombre, trabajo }){`
- `const contenido= document.querySelector('.contenido');`
- `contenido.innerHTML=``
- `<p>Empleado: ${nombre}</p>`
- `<p>ID: ${id}</p>`
- `<p>Empresa: ${empresa}</p>`
- `<p>Trabajo: ${trabajo}</p>`
- ``;`
- `}`
- InnerHTML reemplaza lo que teníamos en la pagina anteriormente limpiando todo
- De esta forma puedes consultar un BD y mostrar los resultados con javascript

# 230. Consultar e Imprimir los Resultados de un Fetch

- Cargar un json que es un array
- No podemos aplicar destructuring porque es un arreglo en parentesis de mostrar HTML
- `const cargarJSONArrayBtn=document.querySelector('#cargarJSONArray');`
- `cargarJSONArrayBtn.addEventListener('click', obtenerDatos);`
- 
- `function obtenerDatos(){`
- `const url='data/empleados.json';`
- 
- `fetch(url)`
- `.then(respuesta => respuesta.json())`
- `.then (resultado => mostrarHTML(resultado))`
- `}`
- 
- `function mostrarHTML(empleados){`
- `const contenido=document.querySelector('.contenido');`
- 
- `let html=""`
- 
- `empleados.forEach(empleado => {`
- `const { id, nombre, empresa, trabajo}= empleado;`
- 
- `html +=``
- `<p>Empleado: ${nombre}</p>`
- `<p>ID:${id}</p>`
- `<p>Empresa:${empresa}</p>`
- `<p>Trabajo: ${trabajo}</p>`
- ``; // concatenando`
- `});`
- `contenido.innerHTML=html;`
- }



- Nosotros le damos click en el botón y se ejecuta el código, si lo quieres de forma automática
  - Entrar y mostrarse como en facebook que entras y se muestran los post
  - Al principio agregamos lo siguiente:
  - → `document.addEventListener('DOMContentLoaded', obtenerDatos );`
  - En lugar de los 2 de abajo
- `//const cargarJSONArrayBtn=document.querySelector('#cargarJSONArray');`
- `//cargarJSONArrayBtn.addEventListener('click', obtenerDatos);`

# 231. Consultar e Imprimir los Resultados de una API

- Cargar API
- Como consumir una api, la respuesta de una api
- Así se ve un json ya desde una api
- Para poder consumir bien esta info, poderla mostrar y tenerla de un mejor formato instalar la extension JSONview en explorador
- Muchos sitios web ofrecen apis que puedes utilizar, por ejemplo una api muy popular es la de youtube , puedes consumir algunos de esos videos
- Lo mejor de apis con respuesta json es que si eres desarrollador frontend, no es necesario que sepas en que lenguaje esta hecho el backend, la respuesta json es universal, puede estar hecha en java, en python y siempre se va a ver igual
- La api que estoy utilizando es un api de perfiles onsplash que te permite descargar imágenes y utilizarlas en tus proyectos
- Api que consume datos de internet y lo podemos mostrar en nuestro sitio web
- La mayoría de aplicaciones modernas tienen un backend en json y puedes consumir el frontend de esta forma, tendencia en desarrollo web y movil
- Enlace a la api <https://picsum.photos/list>

# **Sección 36: PROYECTO: API de Clima con Fetch API**

# 232. El Proyecto Finalizado

- Proyecto buscador de clima, le colocamos la ciudad y el país y obtenemos el clima desde una API
- La mayoría de la veces vas a tener que consumir o extraer datos que existen en un web service o en una api
- Poner una ciudad valida y un pais para cotizar el clima
- Validacion de formulario
- Aparece un spinner y el clima
- Proyecto que se conecta de forma externa y extrae datos de un servidor externo

# 233. Primeros pasos con la App

- Utilizaremos api openWeather o clima abierto, es gratis pero necesitas crear una cuenta
- Api keys> nombrar
- Tarda en habilitarla
- Validar porque muchas apis quieren la informacion como ellos la esperan
- Similar al DOMContentLoaded
- `window.addEventListener('load', ()=>{ // similar a DOMContentLoaded pero desde window`
- `formulario.addEventListener('submit', buscarClima);`
- `})`
- Enlace api <https://openweathermap.org/>
-

# 234. Validando el Formulario

- La api requiere que le envíes el país en el código de 2 dígitos
  - `<strong>` es el apropiado para marcar con especial énfasis las partes más importantes de un texto.
  - `<span>` Es un contenedor en línea. Sirve para aplicar estilo al texto o agrupar elementos en línea.
- `alerta.classList.add('bg-red-100', 'border-red-400', 'text-red-700', 'px-4', 'py-3', 'rounded', 'max-w-md', 'mx-auto', 'mt-6', 'text-center' );`
- `//tailwind`
  - `/*background red 100 (bg-red-100)`
  - `padding a los lados 4(px-4)`
  - `padding arriba y abajo de 3(py-3)`
  - `rounded esquinas redondeadas`
  - `(max-w-md )tamaño de la alerta`
  - `centrar horizontalmente con (mx-auto)`
  - `margintop (mt- 6) */`

# 235. Obtener una API Key

- Como enviar datos hacia la api
- Consultar la api, ver si esa ciudad existe o no
- La api de openweather requiere un id de la aplicación, no todas requieren el id, algunas si, previamente google maps no requería el id ahora si porque se dieron cuenta que era una buena forma de saber que sitios web consumen muchos recursos, y quien utilizara mucho podrían cobrarle
- Mandar los datos como la api lo espera (forma estructurada)
- Si estas en una empresa en frontend y hay un equipo de backend, ellos te tienen que decir que urls hay disponibles en la api para que puedas enviar los datos, tienen que documentar la aplicación de alguna forma
- Si vamos a api en pag, encontramos todo lo que soporta nuestra api >api doc> aparece la forma en la que tienes que llamar esta api
- Esta identificando quiénes somos nosotros y dice, esa api si es valida y consulta lo que estamos pidiendo
- Api tienen reglas de cómo se ponen a disposición los recursos, cada api tiene sus propias reglas
- Si pongo un país y le paso una ciudad que no existe , dice que no la encontró , no cae en el catch porque estamos enviando la consulta correctamente, el catch es cuando el promise no se cumple. Aquí si se cumple pero no hay esa ciudad, se cumple el promise pero el resultado no es correcto

```
// consultar API
→ function consultarAPI(ciudad, pais){
 • const appId= '1d4d8f241edf9c0befb12f0d09e01ad3';
 •
 • const url=`https://api.openweathermap.org/data/2.5/weather?q=${ciudad},${pais}&appid=${appId}`; // consulta lo que nosotros estamos pidiendo
 •
 • fetch(url)
 • .then(respuesta => respuesta.json())
 • .then(datos =>{
 • console.log(datos);
 • if(datos.cod==="404"){
 • mostrarError('Ciudad no encontrada') // solo para cuando no encuentra la ciudad
 • }
 • })
 • }
 • }
→ }
```

# 236. Mostrar la Temperatura Actual

- Sacar información de la consola y mostrarla en el html
- ```
function mostrarClima(datos){
```
- ```
 const {main: {temp, temp_max, temp_min}}=datos; // de nuestra api
```
- ```
  const centigrados= kelvinACentigrados(temp);
```
- ```
 const actual= document.createElement('p');
```
- ```
  actual.innerHTML=`${centigrados} &#8451;`; // forma de poner los grados
```
- ```
 actual.classList.add('font-bold', 'text-6xl');
```
- ```
  const resultadoDiv= document.createElement('div');
```
- ```
 resultadoDiv.classList.add('text-center', 'text-white');
```
- ```
  resultadoDiv.appendChild(actual);
```
- ```
 resultado.appendChild(resultadoDiv);
```
- ```
}
```
- Div destructuring: destructuring de un objeto que esta dentro de otro objeto
- ```
const {name, main: {temp, temp_max, temp_min}}=datos; // de nuestra api
```
- ```
&#8451;`; // forma de poner los grados
```
- ```
'text-6xl' // así es como tailwind define el tamaño de las fuentes
```
- Helpers pequeñas funciones que solamente hacen algo , una entrada de datos y un retorno de datos
- ```
/*function kelvinACentigrados(grados){
```
- ```
 return parseInt(grados-273.15);
```
- ```
  }*/
```
- ```
 //helper
```
- ```
const kelvinACentigrados= grados => parseInt(grados-273.15); // funcion mas pequeña
```
-

237. Mostrar más información de la API

- Como imprimir la temperatura max, la minima
- (font-bold) para que este en negritas

238. Mostrando un Spinner de Carga

- Mostrar un spinner que muestre que se esta cargando en lo que se obtienen los datos
- A veces las apis tardan un poco dependiendo de que estés obteniendo ejemplo descargar los últimos mil productos
- Código del spinner esta en styles.css sk-fading-circle
- Si quieres cambiar el estilo tienes que copiar el codigo y pegarlo en la hoja de estilo <https://tobiasahlin.com/spinkit/>
- Código para spinner

```
→ function Spinner(){  
•  
    limpiarHTML(); // limpie cualquier registro que halla  
•  
    const divSpinner = document.createElement('div');  
•  
    divSpinner.classList.add('sk-fading-circle'); // clase del spinner  
•  
    divSpinner.innerHTML=`  
•      <div class="sk-circle1 sk-circle"></div>  
•      <div class="sk-circle2 sk-circle"></div>  
•      <div class="sk-circle3 sk-circle"></div>  
•      <div class="sk-circle4 sk-circle"></div>  
•      <div class="sk-circle5 sk-circle"></div>  
•      <div class="sk-circle6 sk-circle"></div>  
•      <div class="sk-circle7 sk-circle"></div>  
•      <div class="sk-circle8 sk-circle"></div>  
•      <div class="sk-circle9 sk-circle"></div>  
•      <div class="sk-circle10 sk-circle"></div>  
•      <div class="sk-circle11 sk-circle"></div>  
•      <div class="sk-circle12 sk-circle"></div>  
•      `;  
•  
    resultado.appendChild(divSpinner);  
→ }  
•
```

Sección 37: PROYECTO: Buscador de Canciones con Fetch API

239. El Proyecto Finalizado

- Buscando la letra de la canción
- Consumiendo un segundo web service, una segunda api
- Validacion La api exige 2 campos
- Como leer 2 datos de un formulario y enviarlo de forma estructurada hacia una API
- Pagina de api <https://lyricsovh.docs.apiary.io/#reference/0/lyrics-of-a-song/search>

240. Primeros pasos de este Proyecto

- Vamos a utilizar modulos, por lo tanto en el html:
- → `<script src="js/app.js" type="module"></script>`
- Archivo interfaz hace algunos selectores
- Truco exportar todo lo que halla en un archivo, y todo va a tener el alias de UI
- → `import * as UI from './interfaz.js';`
- Para mandar a llamar
- → `UI.formularioBuscar.addEventListener('submit', buscarCancion);`
- Otra forma de eliminar el mensaje de error despues de 3 segundos
- `setTimeout(() =>{`
- `UI.divMensajes.textContent="";`
- `UI.divMensajes.classList.remove('error');`
- `}, 3000);`

241. Consultar la API con una Clase

- Consultar url

```
→ const url=`https://api.lyrics.ovh/v1/${this.artista}/${this.cancion}`
```

- Mostrar la letra

```
→ consultarAPI(){  
  • const url=`https://api.lyrics.ovh/v1/${this.artista}/${this.cancion}`;  
  •  
  • fetch(url)  
  • .then(respuesta => respuesta.json())  
  • .then(resultado => {  
  •  
  •   if(resultado.lyrics){  
  •     const {lyrics}=resultado;  
  •  
  •     UI.divResultado.textContent=lyrics;  
  •     UI.headingResultado.textContent=`Letra de la cancion: ${this.cancion} de ${this.artista}`;  
  •  
  •   }else{  
  •     UI.divMensajes.textContent='La cancion no existe, prueba con otra busqueda';  
  •     UI.divMensajes.classList.add('error');  
  •  
  •     setTimeout(() =>{  
  •       UI.divMensajes.textContent="";  
  •       UI.divMensajes.classList.remove('error');  
  •     }, 3000);  
  •   }  
  •  
  • })  
  •  
→ }
```

Sección 38: PROYECTO: Buscador de Imágenes con Pixabay API

242. El Proyecto Finalizado

- Crea buscador de imágenes utilizando API PIXABay
- Validacion
- Se puede ver imagen en tamaño completo
- Podemos descargar imágenes y utilizarlos en proyectos incluso comerciales
- Tiene paginacion (imágenes siguiente) utiliza funcionalidad llamada generadores

243. Primeros Pasos con el Proyecto

- Pixabay: banco de imágenes similar onsplash porque son gratis , se pueden usar en proyectos e incluso cobrar por ellos
- Pagina pixabay <https://pixabay.com/api/docs/>
- Ya logeado entrar aquí <https://pixabay.com/api/docs/>
- Solo voy a buscar imágenes por termino (imágenes relacionadas)

- Tailwind
- (max-w-lg) tamaño de ancho de este elemento
- (mx-auto) para que este centrado

- Mostrar alerta

```
→ function mostrarAlerta(mensaje){  
  •   const existeAlerta = document.querySelector('.bg-red-100');  
  •  
  •   if(!existeAlerta){  
  •     const alerta= document.createElement('p');  
  •     alerta.classList.add('bg-red-100', 'border-red-400', 'text-red-700', 'px-4', 'py-3', 'rounded', 'max-w-lg', 'mx-auto', 'mt-6', 'text-center');  
  •  
  •     alerta.innerHTML=`  
  •       <strong class="font-bold">Error!</strong>  
  •       <span class="block sm:inline">${mensaje}</span>  
  •       `;  
  •  
  •     formulario.appendChild(alerta);  
  •  
  •     setTimeout(() =>{  
  •       alerta.remove();  
  •     }, 3000);  
  •  
  •   }  
  •  
→ }
```

244. Consultando la API y viendo la información disponible

- Api key identifica cuantas consultas estas haciendo, y si te pasas te mandan un mensaje para pagar
 - Si otra persona revis tu api key y lo copia, te van a estar cargando esas consultas por lo tanto se utiliza un framework o una librería con react o angular que te permiten tener variables de entorno, de esa forma quedan como ocultas (mantiene esa info del lado del servidor)
 - Poner llave y url
- ```
→ function buscarImagenes(termino){
```
- `const key='20407877-bff5313c9288c8ff3739decb5';`
- ```
→ const url=`https://pixabay.com/api/?key=${key}&q=${termino}`
```
- En consola hits es cada uno de los resultados
 - Total cuantas imágenes en pixabay puedo encontrar
 - Totalhits de ese total solo te ponen a disposición ej 500 (esta info esta en la misma documentacion)
 - Para acceder a las imágenes
- ```
→ mostrarImagenes(resultado.hits);
```

# 245. Mostrar los Resultados de la consulta

- Como mostrar las imágenes
  - (mb-4) para que se separen hacia abajo
  - (w-full) para que tome todo el ancho que tenga disponible
  - (P-1) padding de 1 para que el botón se vea un poco mas grueso
  - (block) para que nuestras clases tomen efecto porque los enlaces son de tipo inline
  - Solo muestra 20 imágenes, como mostrar las demas
  - (per\_page) default 20
  - Para que traiga 100 imágenes (&per\_page=100`) por cada vez que se haga una consulta
- `const url=`https://pixabay.com/api/?key=${key}&q=${termino}&per_page=100`;`

```

→ //Iterar sobre el arreglo de imagenes y construir el html
• imagenes.forEach(imagen => { //imagenes es un arreglo
• const {previewURL, likes, views, largeImageUrl}= imagen;
• -----imagen de baja resolucion
• -----imagen de alta resolucion
•
• resultado.innerHTML += //toma la mitad, mediano una terceraparte, mas largo toma una cuarta parte, y se van acomodando las imagenes // dep
//endiendo del tamaño de la pantalla
•
• <div class="w-1/2 md:w-1/3 lg:w-1/4 p-3 mb-4">
• <div class="bg-white">
•
•
• <div class="p-4">
• <p class="font-bold"> ${likes} Me gusta </p>
• <p class="font-bold"> ${views} Veces Vista </p>
•
• <a
• class="block w-full bg-blue-800 hover:bg-blue-500 text-white uppercase font-bold text-center rounded mt-5 p-1"
• href="{largeImageUrl}" target="_blank" rel="noopener noreferrer"
• >
• Ver Imagen
•
• </div>
• </div>
• </div>
• `; //target="_blank" rel="noopener noreferrer"> // abrir la imagen en una nueva pestaña // arreglar problema de seguridad
• })
• }

```

# 246. Primeros pasos para crear un paginador

- Acceder a la imagen 101 requerimos a paginación
- Saber cuantos elementos hay disponibles para crear la paginacion porque no puedes paginar registros de mas pero tampoco puedes dejar registros que no sean visibles
- Paginacion debe ser dinamica de acuerdo a la cantidad de registros que halla disponibles
- Definir cuantos elementos vas a mostrar
- Calcular el total de paginas (redondear siempre hacia arriba)
- ```
function calcularPaginas(total){
```
- ```
 return parseInt(Math.ceil(total/registrosPorPagina));
```
- ```
}
```
- codigo
- ```
function buscarImagenes(termino){
```
- ```
  const key='20407877-bff5313c9288c8ff3739dec5';
```
- ```
 const url=`https://pixabay.com/api/?key=${key}&q=${termino}&per_page=100`;
```
- ```
  fetch(url)
```
- ```
 .then(respuesta => respuesta.json())
```
- ```
    .then(resultado =>{
```
- ```
 totalPaginas=calcularPaginas(resultado.totalHits);
```
- ```
      console.log(totalPaginas);
```
- ```
 mostrarImagenes(resultado.hits);
```
- ```
    })
```
- ```
 }
```
- ```
function calcularPaginas(total){
```
- ```
 return parseInt(Math.ceil(total/registrosPorPagina));
```
- ```
}
```

247. Generar un Paginador con un Generador

- Mostrar paginador en la parte inferior de acuerdo a la cantidad de paginas que halla disponibles usando un generador
 - Los generadores vienen con un iterador, y ese nos permite identificar cuando hemos llegado a la parte final del generador, no tenemos que estar revisando si ya estamos en la ultima pagina ya que el generador lo detecta
- `const registrosPorPagina= 40;`
- `let totalPaginas; //para que este disponible en todas la funciones`
 - `let iterador;`
-
- `// generador que va a registrar la cantidad de elementos de acuerdo a las paginas`
 - `function* crearPaginador(total){`
 - `for (let i=1; i<= total; i++){`
 - `yield i;`
 - `}`
 - `}`
 - `function imprimirPaginador(){`
 - `iterador=crearPaginador(totalPaginas);`
- `}`

248. Mostrando la cantidad de páginas del Paginador

- Registra las 17 paginas porque tomamos los valores de yield para mostrarlos en la función imprimir paginador
 - Codigo ve cuantos elementos hay registrados en el generador, y crea esa cantidad de paginas
- `console.log(iterador.next().value)// valor del iterador`
→ `console.log(iterador.next().done)// si ya llego al final //true or false nos dice si ya terminamos o ya recorrimos todo el iterador`
- En el html para que no se vaya a los lados, si no que fluya hacia abajo el paginador
- `<div id="paginacion" class="container mx-auto mb-10 text-center flex flex-wrap mx-auto justify-cente"></div>`
- El generador y el iterador solo nos muestran la cantidad de elementos que halla registrados en el generador, pero la paginacion tenemos que hacerla nosotros
- *function* imprimirPaginador(){
- `iterador=crearPaginador(totalPaginas);`
 -
 - `while(true){`
 - `const {value, done}= iterador.next();`
 - `if(done) return;`
 -
 - `//caso contrario, genera un boton por cada elemento en el generador`
 - `const boton= document.createElement('a');`
 - `boton.href='#'; // no lleva a ningun lado, solo nos lleva de una pagina a otra`
 - `boton.dataset.pagina=value;`
 - `boton.textContent=value;`
 - `boton.classList.add('siguiente', 'bg-yellow-400', 'px-4', 'py-1', 'mr-2', 'font-bold', 'mb-4', 'uppercase', 'rounded');`
 -
 - `paginacionDiv.appendChild(boton);`
 - `}`
- `}`

249. Navegar por las páginas

- Leer el valor al cual le estoy dando click, y me lleve a esa pagina
 - No todas las apis tienen paginacion, si la api no tiene paginacion va a ser muy dificil paginarla y ninguna solución va a ser muy optima en el uso de la memoria , de eso se encargaría backend
 - Para que nos mande a la pagina seleccionada
- ```
→ const url=`https://pixabay.com/api/?key=${key}&q=${termino}&per_page=${registrosPorPagina}&page=${paginaActual}`;
```
- Generador permite iterar sobre todos los registros y te dice cuando ha llegado al final
  - Codigo para hacer un generador :
- ```
→ // generador que va a registrar la cantidad de paginas
function* crearPaginador(total){
  for (let i=1; i<= total; i++){
    yield i;
  }
}
```
- ```
→ function imprimirPaginador(){
 iterador=crearPaginador(totalPaginas);

 while(true){// mientras no lleguemos al final del generador
 const {value, done}= iterador.next();
 if(done) return;

 //caso contrario, genera un boton por cada elemento en el generador
 const boton= document.createElement('a');
 boton.href='#';// no lleva a ningun lado, solo nos lleva de una pagina a otra
 boton.dataset.pagina=value;
 boton.textContent=value;
 boton.classList.add('siguiente', 'bg-yellow-400', 'px-4', 'py-1', 'mr-2', 'font-bold', 'mb-4', 'rounded');

 boton.onclick=()=>{
 paginaActual=value;

 buscarImagenes();// volvemos a consultar nuestra api con un numero de paginacion diferente
 }

 paginacionDiv.appendChild(boton);
 }
}
```
- ```
→ }
```


Sección 39: PROYECTO: Cotizador de Criptomonedas

250. El Proyecto Finalizado

- Buscador de los precios actuales de las criptomonedas, si tienes un sitio web, de bitcoin de temas relacionados, puedes agregar este cotizador actual que te diga en que precio esta el bitcoin, en que precio estan todas las diferentes monedas que existen
- Consumiendo api externa que nos pide una moneda y una criptomoneda
- Validación
- Agregar múltiples monedas, elegir entre 10, supuestamente las mas importantes
- Muestra un spinner
- La api tiene como 2000 criptomonedas
- Cotizaciones en tiempo real
- Link api <https://min-api.cryptocompare.com/documentation>

251. Primeros pasos con este proyecto

- [Single symbol price > https://min-api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD,JPY,EUR](https://min-api.cryptocompare.com/data/price?fsym=BTC&tsyms=USD,JPY,EUR)
 - -----codigo de 3 digitos de la criptomoneda
 - -----precio de una moneda tradicional
 - Buscar criptomonedas mas comunes:
 - Toplists > Market cap> execute call> response
 - Si trabajas en una empresa como frontend, tendras que comunicarte con backend para saber los diferentes endpoints y los parametros que tendrias que enviarle a esa API
 - Llenar select con las criptomonedas mas importantes:
- ```
→ //Crear un promise
• const obtenerCriptomonedas = criptomonedas => new Promise (resolve => {
• resolve(criptomonedas);
→ }); // descarga criptomonedas

→ function consutarCriptomonedas(){
• const url='https://min-api.cryptocompare.com/data/top/mktcapfull?limit=10&tsym=USD'; //url de criptomonedas populares (limit=10)
•
• fetch(url)
• .then(respuesta => respuesta.json())
• .then(resultado => obtenerCriptomonedas(resultado.Data))
• .then(criptomonedas => selectCriptomonedas(criptomonedas))
•
• }

• function selectCriptomonedas(criptomonedas){
• criptomonedas.forEach(cripto => {
• const {FullName, Name} = cripto.CoinInfo;
•
• const option= document.createElement('option');
• option.value= Name; //codigo de 3 digitos
• option.textContent= FullName; // texto que si entiende el usuario
• criptomonedasSelect.appendChild(option);
• })
•
→ }
```

# 252. Leer la moneda y criptomoneda seleccionada

- Como leer valores para saber que es lo que vamos a cotizar
- Para no volver a seleccionar la opción por default en el html:  
→ `<option value="" disabled selected>Elige tu criptomoneda</option>`
- Código:  
→ 

```
const criptomonedasSelect=document.querySelector('#criptomonedas');
const monedaSelect= document.querySelector('#moneda');
const formulario=document.querySelector('#formulario');
```
- ```
const objBusqueda=// se van a llenar conforme el usuario seleccione su búsqueda
moneda:"",
criptomoneda:"
}
```
- ```
document.addEventListener('DOMContentLoaded', ()=>{
consultarCriptomonedas();
```
- ```
formulario.addEventListener('submit', submitFormulario);
```
- ```
criptomonedasSelect.addEventListener('change', leerValor);
monedaSelect.addEventListener('change', leerValor); // esto pasa en automaticamente porque los select tienen el nombre name: "criptomoneda" y el otro name: "moneda" por eso es que se mapean los valores correctamente
})
```
- ```
function leerValor(e){
objBusqueda[e.target.name]=e.target.value; // ASIGNAR VALOR AL OBJETO name en el html igual a name en el objeto
console.log(objBusqueda);
}
```
- ```
function submitFormulario(e){
e.preventDefault();

// validar
const {moneda, criptomoneda}= objBusqueda;
```
- ```
if(moneda===" " || criptomoneda===""){
mostrarAlerta('Ambos campos son obligatorios');
return;
}
}
```
- ```
}
```

```
→ function mostrarAlerta(msg){
• console.log(msg);
→ }
```

# 253. Validación del Formulario

- Código para Mostrar alerta

# 254. Consultando la API

- Mostrar la api con los resultados
- Como consultar la api para obtener la cotización que el usuario desea
- Tiene mas info, viene el precio mas alto del dia, el mas bajo del dia, cuando fue actualizado >Multiple Symbols Full Data
- Vamos a utilizar el resultado de display porque tiene el signo de pesos
- ENCONTRAR la forma de entrar a las propiedades del objeto de forma dinamica

```
→ function consultarAPI(){
• const {moneda, criptomoneda}=objBusqueda;
•
• const url=`https://min-api.cryptocompare.com/data/pricemultifull?fsyms=${criptomoneda}&tsyms=${moneda}`;
•
• fetch(url)
• .then(respuesta => respuesta.json())
• .then(cotizacion => {
• mostrarCotizacionHTML(cotizacion.DISPLAY[criptomoneda][moneda]); //acceder al objeto de forma dinámica , leer desde objeto
• // de busqueda, objeto dinamico, le pasa los valores adecuados
• })
• }
•
• function mostrarCotizacionHTML(cotizacion){
• console.log(cotizacion);
• }
→ }
```

# 255. Imprimiendo el resto de la Información

- Mostrar cotización, sacarlo de la consola y mostrarlo en el html
- Uso innerHTML porque se de donde vienen los datos, y voy a mezclar un poco de html con código

```
→ function mostrarCotizacionHTML(cotizacion){
•
• limpiarHTML();
•
• const { PRICE, HIGHDAY, LOWDAY, CHANGEPC24HOUR, LASTUPDATE } = cotizacion;
•
• const precio = document.createElement('p');
• precio.classList.add('precio');
• precio.innerHTML = `El Precio es: ${PRICE}`;
•
• const precioAlto = document.createElement('p');
• precioAlto.innerHTML = `<p>Precio mas alto del dia: ${HIGHDAY}`;
•
• const precioBajo = document.createElement('p');
• precioBajo.innerHTML = `<p>Precio mas bajo del dia: ${LOWDAY}`;
•
• const ultimasHoras = document.createElement('p');
• ultimasHoras.innerHTML = `<p>Variacion ultimas 24 horas: ${CHANGEPC24HOUR}%`;
•
• const ultimaActualizacion = document.createElement('p');
• ultimaActualizacion.innerHTML = `<p>Ultima Actualizacion: ${LASTUPDATE}`;
•
• resultado.appendChild(precio);
• resultado.appendChild(precioAlto);
• resultado.appendChild(precioBajo);
• resultado.appendChild(ultimasHoras);
• resultado.appendChild(ultimaActualizacion);
•
•
• }
•
• function limpiarHTML(){
• while(resultado.firstChild){
• resultado.removeChild(resultado.firstChild);
• }
• }
→ }
```



# 256. Mostrando un Spinner de Carga

- Agregar un spinner <https://tobiasahlin.com/spinkit/>
- ```
→ function mostrarSpinner(){  
  • limpiarHTML();  
  •  
  • const spinner= document.createElement('div');  
  • spinner.classList.add('spinner');  
  •  
  spinner.innerHTML=`  
  • <div class="bounce1"></div>  
  • <div class="bounce2"></div>  
  • <div class="bounce3"></div>  
  • `;  
  •  
  resultado.appendChild(spinner);  
→ }
```

Sección 40: PROYECTO: API de Github Jobs para crear un buscado...

257. El Proyecto FINALIZADO

- Buscar trabajos de diferentes tecnologías
- Como enviar los datos que escribimos hacia la API de github jobs, y consumir sus resultados

258. Primeros pasos y agregando Axios

- Axios: es como un fetchAPI mejorado, el problema es que no es nativo a diferencia de fetch, todos los navegadores tienen FetchAPI, axios es una librería externa, para instalar librerías externas, puedes agregar el enlace en tu html
 - Cdnjs encuentras muchas librerías que puedes utilizar <https://cdnjs.com/>
 - Forma de instalar librerías externas: en html antes de `<script src="js/app.js"></script>`
- `<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.21.1/axios.min.js" integrity="sha512-bZS47S7sPOxkjU/4Bt0zrhEtWx0y0CRkhEp8lckzK+ItiflIE9EMIMTuT/mEzolMewUINruDBIR/jJnbguonqQ==" crossorigin="anonymous"></script>`
- `<script src="js/app.js"></script>`

259. Consultando la API de Github Jobs

- Realizar una búsqueda hacia la API
- Cuando sale error: has been blocked by CORS (intercambio de recursos de origen cruzado) policy: tratamos de consumir una api que esta en otro dominio, en este dominio 127.0.0.1
- Api key es para monitorear tu consumo, y asi poder cobrarte posteriormente si te pasas del consumo(identificar quien esta realizando esas consultas)
- Github no tiene Api keys por lo que estamos tratando de consumir un api que ellos protegieron de una forma para que nadie la muestre o utilice (backend se encargo de protegerla, y no hay forma de consumirla) el backend dice todos los request deben de venir de esta url, de otra forma no los aceptes
- En resumen, la api esta protegida

- Como saltarnos la proteccion, porque no hay forma de crear un api key con proxy

```
→ function consultarAPI(busqueda){  
  •   const githubUrl=`https://jobs.github.com/positions.json?search=${busqueda}`;  
  •   //saltando CORS  
  •   const url=`https://api.allorigins.win/get?url=${encodeURIComponent(githubUrl)}`;  
  •  
  •   -----cifrar pero no llega a tal grado ,  
  •   //      le aplica un encoding a la url que le pasemos  
  •  
  
  axios.get(url)  
  •   .then ( respuesta => mostrarVacantes(JSON.parse(respuesta.data.contents))); // convertir estructura que tiene la forma de json pe  
  //   ro es un string  
→ }
```

- Como usamos axios la respuesta del servidor te la retorna en data

- Imprime la compañía que esta contratando

```
→ resultado.innerHTML+= `
•         <div class="shadow bg-white p-6 rounded">
•             <h2 class="text-2xl font-light mb-4">${company}</h2>
•             </div>
•
→ `;//
```

- Grid en html: Los elementos se han colocado en el **grid** utilizando posicionamiento en línea.

```
→ function mostrarVacantes(vacantes){
•
•     while(resultado.firstChild){
•         resultado.removeChild(resultado.firstChild);
•     }
•
•     if(vacantes.length>0){
•         resultado.classList.add('grid');
•
•         vacantes.forEach(vacante =>{
•             const {company, title, type, url}=vacante;
•
•             resultado.innerHTML+= `
•             <div class="shadow bg-white p-6 rounded">
•                 <h2 class="text-2xl font-light mb-4">${title}</h2>
•                 <p class="font-bold uppercase">Compañia: <span class="font-light normal-case">${company} </span></p>
•                 <p class="font-bold uppercase">Tipo de Contrato: <span class="font-light normal-case">${type} </span></p>
•                 <a class="bg-teal-500 max-w-lg mx-auto mt-3 rounded p-2 block uppercase font-xl font-bold text-white text-center" href="${url}">Ver Vacante</a>
•             </div>
•
•             `;
•             //imprime todo lo de las vacantes
•         })
•     }
→ }
```

260. Mostrar un mensaje cuando no haya vacantes

- Mostrar que no hubo resultados.
 - El mensaje aparece a un lado porque ese espacio tiene la clase de grid, para que se centre hay que quitarle esa clase
 - A veces alguna clase que agregaste tendrás que limpiarla después
- `resultado.classList.remove('grid');`
- enlaces
 - <https://gist.github.com/juanpablogdl/e9c2a14364232b167785ff6513519f27> // gist
 - <https://jobs.github.com/positions.json?location=mexico> API de jobs

Sección 41: Async Await en JavaScript

261. Pero primero, que es Try Catch?

- Async Await complemento o alternativa hacia promises
 - Try catch se utiliza mucho con asyn await
 - Característica de JavaScript cuando tu código falla, deja de ejecutarse
 - Try catch intenta ejecutar una pieza de código, y si no se puede, obtenemos una excepción, podría ser un mensaje de error para poderlo debugiar, pero nuestro código no deja de funcionar.
 - Ej al intentar descargar un listado de alumnos, podemos escribir una función que consuma una api , pero en caso de que no se pueda o marque un error, podemos enviarle al usuario un mensaje de error, que diga: no se pudieron descargar, al colocarlo en un try catch el resto de código continua funcionando correctamente
 - Recomendado utilizarlo en partes criticas de tu aplicación, como puede ser conectar BD, consultar una API, autenticar un usuario, o acciones que nos permitan que en caso de que falle, nuestra aplicación continúe funcionando, pero que podamos obtener un mensaje de error
- `console.log(1+1);`
- `try {`
 - `autenticarUsuario();`
 - `} catch (error) {`
 - `console.log(error);`
 - `}`
- `console.log(2+2);`

262. Nuestro Primer Ejemplo de Async Await y que es lo que hace

- Bases de asyn await
- Recuerda que fetch son promises
- Asyn tiene que ser la función padre, la función sobre la cual se esta ejecutando el promise , en este caso, es ejecutar();
- Await : en la parte que va a esperar a que se ejecute el promise , detiene la ejecución del código, hasta que se resuelva el promise, digamos que la bloquea, por lo tanto la siguiente línea no se puede ejecutar hasta que tengamos un resultado de la función
- Ej descargar clientes, el usuario no puede hacer nada hasta que se descarguen todos esos clientes
- Una vez que el try detecta un error, automáticamente detiene la ejecución del código, y se salta automaticamente hacia el catch no permitiendo que se ejecuten las siguientes líneas en el try

- Recomendable para detener la ejecución al descargar clientes , pedidos, etc
- Function declaration:

→ `//Async await`

```
•  
  async function ejecutar(){  
•    try{  
•      console.log(1+1);  
•      const respuesta= await descargarClientes(); // si hay error en promise (reject) se va al catch  
•  
      console.log(2+2);  
•      console.log(respuesta);  
•  
    }catch(error){  
•      console.log(error);  
•    }  
→ }  
}
```

263. Async Await en function express y Declaration

- Otra forma de crear una funcion async await porque el anterior era el function declaration y ahora es con function expression
- Function expression:
 - `const ejecutar = async () =>{`
 - `try{`
 - `console.log(1+1);`
 - `const respuesta= await descargarClientes(); // si hay error en promise (reject) se va al catch`
 -
 - `console.log(2+2);`
 - `console.log(respuesta);`
 - `}`
 - `}catch(error){`
 - `console.log(error);`
 - `}`
 - `}`
 - `ejecutar();`

264. Como manejar múltiples awaits?

- Trabajar con 2 o mas promises en un mismo await
- Como consumir o utilizar 2 promises, es decir hacer 2 consultas a una api o consumir 2 endpoints etc, de una forma que sea muy eficiente de hacer

→

```
// si funciona pero no de la forma correcta porque esta descargando clientes, y mientras esta bloqueando la descarga de nuevos pedidos eso esta bie  
//n cuando quiere tener listos unos resultados para hacer la consulta en otros, pero en este caso son consultas independientes
```

```
•  
• /*const app= async () => {  
•   try{  
•     const clientes = await descargarNuevosClientes();  
•     console.log(clientes);  
•  
•     const pedidos= await descargarNuevosPedidos();  
•     console.log(pedidos);  
•   } catch( error){  
•     console.log(error);  
•   }  
• }  
• }  
→ app(); */
```

→ // para promises independientes para mejorar el perfomance

```
• const app= async () => {  
•   try{  
•     const respuesta= await Promise.all([ descargarNuevosClientes(), descargarNuevosPedidos()]); // toma un arreglo de diferentes promises que qui/  
•     //eres ejecutar al mismo tiempo porque no tienen nada que ver uno con otro  
•     //console.log(respuesta)// salen las respuestas de ambos promises  
•     console.log(respuesta[0]);  
•     console.log(respuesta[1]); // obtener respuestas de promises por separado  
•   } catch(error){  
•     console.log(error)  
•   }  
• }  
• }  
→ app();
```

265. Async Await hacia una API con Fetch

- Como utilizar async await con fetch api, en un llamado hacia un api <https://picsum.photos/list>
 - Como consumir json de esta api, utilizando asyc await y fetch api
- `const url ='https://picsum3.photos/list';`
- `document.addEventListener('DOMContentLoaded', obtenerDatos);`
- `// con promises`
- `/*function obtenerDatos(){`
 - `fetch(url)`
 - `.then(respuesta => respuesta.json())`
 - `.then(resultado => console.log(resultado))`
 - `.catch(error => console.log(error));`
- `*/`
- `//con async await(es mas corto) el de arriba si E un erro si me dice donde y este no porque este aun no tiene el catch`
- `/*async function obtenerDatos(){`
 - `const respuesta= await fetch(url); // esta linea bloquea la siguiente porque en base a esa respuesta es lo que va a ejecutar la siguiente linea`
 - `const resultado= await respuesta.json();`
 - `console.log(resultado);`
- `*/`
- `//agregando try catch`
- `async function obtenerDatos(){`
 - `try{`
 - `const respuesta= await fetch(url); // esta linea bloquea la siguiente porque en base a esa respuesta es lo que va a ejecutar la siguiente linea`
 - `const resultado= await respuesta.json();`
 - `console.log(resultado);`
 - `}`
 - `}catch(error){`
 - `console.log(error);`
 - `}`
- `}// ahora si este y el fetch son equivalentes y el de fetch es menos código, pero esta es una sintaxis mas clara de leer`
-

Sección 42: PROYECTO : Moviendo el Proyecto de Pixabay a Async Await

266. Moviendo el Proyecto a Async Await

- Mover proyecto pixabay de promises hacia fetch api con async await

→ *async function* buscarImágenes() {

- - `const termino = document.querySelector('#termino').value;`
 -
 - `const key = '20407877-bff5313c9288c8ff3739dec5';`
 - `const url = `https://pixabay.com/api/?key=${key}&q=${termino}&per_page=${registrosPorPagina}&page=${paginaActual}`;`
 -
 - `// fetch(url)`
 - `// .then(respuesta => respuesta.json())`
 - `// .then(resultado => {`
 - `// totalPaginas = calcularPaginas(resultado.totalHits);`
 - `// mostrarImágenes(resultado.hits);`
 - `// });`
 -
 - `// lo mismo que el código comentado`
 - `try{`
 - `const respuesta = await fetch(url);`
 - `const resultado = await respuesta.json();`
 - `totalPaginas = calcularPaginas(resultado.totalHits);`
 - `mostrarImágenes(resultado.hits);`
 - `}catch(error){`
 - `console.log(error);`
 - `}`
- }

Sección 43: PROYECTO: Moviendo el Proyecto de Criptomonedas a Asyn...

267. Moviendo el Proyecto

- Migrar el proyecto de criptomonedas
 - Para criptomonedas
- *async function* consultarCriptomonedas(){
- *const* url='https://minapi.cryptocompare.com/data/top/mktcapfull?limit=10&tsym=USD'; //url de criptomonedas populares (limit=10 //)
 - // a reemplazar
 // fetch(url)
 - // .then(respuesta => respuesta.json())
 - // .then(resultado => obtenerCriptomonedas(resultado.Data))
 - // .then(criptomonedas => selectCriptomonedas(criptomonedas))
 - // reemplazando
 try {
 - *const* respuesta= await fetch(url); // await porque requieren que se allan completado los anteriores
 - *const* resultado= await respuesta.json(); // es igual a la constante de arriba
 - *const* criptomonedas= await obtenerCriptomonedas(resultado.Data);
 - selectCriptomonedas(criptomonedas); // solo 3 awaits como 3 then
 -
 - } catch (error) {
 - console.log(error);
 -
 - }
- > }

Sección 44: PROYECTO: CRM Crud con REST y Async Await

Ligas:

<https://github.com/typicode/json-server>

<https://nodejs.org/es/>

268. Creando una API con JSON Server

- Crear proyecto desde cero crud
- JavaScript es un lenguaje que te permite crear aplicaciones fullStack (probablemente es el único lenguaje) , funciona tanto en el cliente (navegador), como en un servidor
- Creando un servidor utilizando una dependencia que se llama JSON server
- Las apis que hemos estado creando no sabemos en que backend están ejemplo la de clima no sabemos si esta hecha en java, en csharp o en php, eso pasa porque una forma de conectar diferentes tecnologías es por medio de un api que nos de una respuesta de tipo json, de esa forma puedes conectar un backend hecho en java, csharp, con un frontend hecho en una tecnología diferente, en este caso JavaScript, y eso es lo que te permite hacer json server, crear una api muy rápido porque no requieres programar nada, solamente creas un archivo y ya tienes una api lista, que va a responder a los diferentes verbos , ya sea actualizar un registro, listarlo, eliminarlo o mostrarlos
- Como agregar json server.
- Instalando node.js:
 - Cmd
 - → node -v //para ver la version de node que se instalo
- Npm: node package manager: una forma que te permite instalar dependencias o paquetes de codigo que alguien mas escribio, y los pone disponibles , a manera de dependencia, basicamente son librerias
 - → npm -v
- Instalando json server
 - Cmd:
 - npm i -g json-server
 - para que se instale de forma global (en cualquier carpeta de mi equipo, puedo correr el comando de json // // server
- A veces cuando instalas un paquete normalmente, te marca errores, en mac y en linux se soluciona asi:
 - Cmd:
 - sudo npm i -g json-server // instalar como administrador

- Crear api
- Llegar a la carpeta donde esta mi proyecto en cmd
- 44-PROYECTO-CRM-CRUD-REST
- cls // para limpiar pantalla cmd

- Cuando instalas una dependencia de forma global, tienes acceso a un comando que ejecute esa dependencia

- En archivo db.json esta la estructura que requieres para crear un api utilizando json server, pero todo eso es un json

```
→ {
  •   "clientes": [
  •     {
  •       "nombre": "Juan NUEVO!",
  •       "email": "correo2@correo.com",
  •       "telefono": "12",
  •       "empresa": "12212",
  •       "id": 26
  •     }
  •   ]
}
```

→} // este archivo va a actuar como la BD de nuestro proyecto

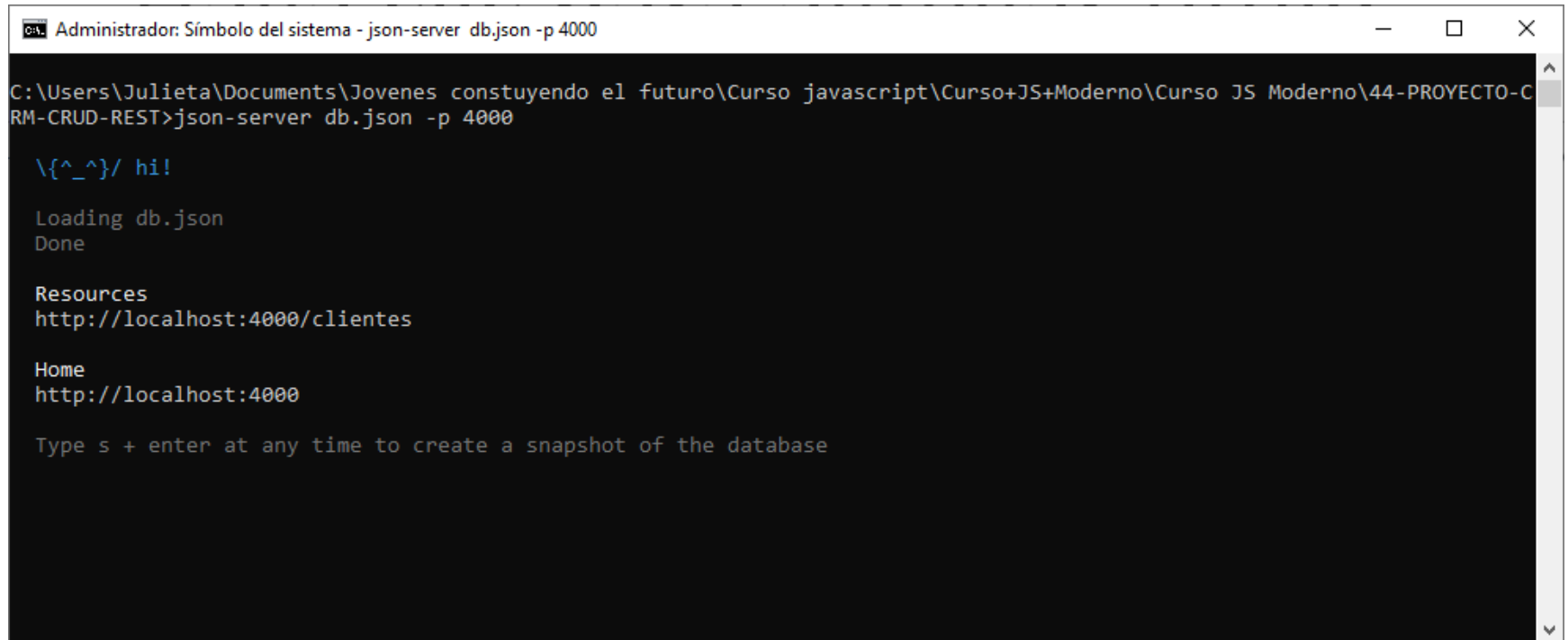
- Json server si te permite simular la creación de una api en 5 min o menos, pero no es algo que puedas utilizar en un proyecto muy grande por ejemplo no puedes crear el próximo twitter usando esto, es mas para practicar, e ir creando los diferentes end points de una api

- Ejecutando el comando en cmd en la carpeta del proyecto:

→ json-server db.json -p 4000

- -----// archivo de la BD
- -----para un puerto, y le asigno el puerto 4000

Aparece que cargo correctamente la BD y en resources, esta la ruta de nuestra API, que si la abrimos en el navegador, nos sale nuestra info de db.json



```
Administrador: Símbolo del sistema - json-server db.json -p 4000

C:\Users\Julieta\Documents\Jovenes constuyendo el futuro\Curso javascript\Curso+JS+Moderno\Curso JS Moderno\44-PROYECTO-CRM-CRUD-REST>json-server db.json -p 4000

\{^_^}/ hi!

Loading db.json
Done

Resources
http://localhost:4000/clientes

Home
http://localhost:4000

Type s + enter at any time to create a snapshot of the database
```

- Ctrl+c para detener la ejecución en cmd
- La idea es que los registros se generen desde la interface web, por lo tanto eliminamos el contenido de lo que hay dentro de los corchetes de "clientes":[] en db.json
- Va a ser muy sencillo ir creando nuevos registros, e ir listando los elementos, nos va a permitir simular un BD, también nos va a permitir entregar unos resultados como json
- API.js tiene todas las interacciones con la API
- Editarcliente.js y nuevocliente.js : tienen la validación del formulario, leen los campos etc
- funciones.js : funciones que podemos reutilizar en diferentes archivos
- Comenzamos con la parte de crear nuevos clientes para poder listarlos, editarlos y eliminarlos

→ //otra forma de validar

- `const cliente={`
- `nombre,`
- `email,`
- `telefono,`
- `empresa`
- `}`
- `console.log(Object.values(cliente).every(input=> input != ''));`

→ }

- Se recomienda manejar asi, ya en una aplicación mas grande

- Ya el código como quedo:
- `//otra forma de validar`
- `const cliente={`
- `nombre,`
- `email,`
- `telefono,`
- `empresa`
- `}`
-
- `if(validar(cliente)){`
- `//Mostrar mensaje`
- `console.log('Todos los campos son obligatorios');`
- `return;`
- `}`
-
- `console.log('Si se paso la validacion');`
- `}`
-
- `function validarCliente(obj){`
- `return !Object.values(cliente).every(input => input !== ""); // sale true si almenos uno esta vacio`
- `}`
- `})();`

270. Mostrar un mensaje si no se pasa la validación

- Como mostrar alerta, que colocaremos en funciones porque es de uso mas general
- En html
- (max-w-lg) // tamaño máximo del tamaño grande
- (mx-auto) para centrar
- (mt-6) // separado arriba
- ```
export function mostrarAlerta(mensaje){
 • const alerta= document.querySelector('.bg-red-100');
 •
 if(!alerta){
 • const alerta=document.createElement('p');
 •
 alerta.classList.add('bg-red-100', 'border-red-400', 'text-red-700', 'px-4', 'py-3', 'rounded', 'max-w-lg', 'mx-auto', 'mt-6', 'text-center');
 •
 alerta.innerHTML=`
 • <strong class="font-bold">Error!
 • ${mensaje}
 • `;
 •
 const formulario=document.querySelector('#formulario');
 • formulario.appendChild(alerta);
 •
 setTimeout(() => {
 • alerta.remove();
 • }, 3000);
 •
 }
 }
→ }
```



# 271. Enviar una Petición POST al servidor con el nuevo cliente

- Insertar cliente en nuestra api
- Como agregar un nuevo cliente, para hacer eso es muy útil utilizar una rest api (forma organizada de realizar las cosas y no le agrega complejidad que si tiene grapcool)
- Nuestra api dice /clientes . json server ya soporta todos los verbos de http ej:
  - get: obtener registros
  - post: para enviar registros al servidor nuevos
  - put o patch: para actualizar
  - delete: para eliminar
- Json server automaticamente agrega todos estos verbos hacia /clientes, es por eso que es una excelente herramienta para aprender rest apis
- GET /posts/1
- -----id, cada uno de los registros conforme se vayan guardando en la api
- Insertar cliente en nuestra api (nuevo registro):
- Usar async await con try catch para intentar insertar un nuevo cliente en BD y puede que falle porque BD no esta conectada o el servidor se callo etc, puede que pasen muchas cosas en ese intervalo en el cual mandamos info al servidor
- Fetch por lo general ejecuta el verbo de get, es decir obtener datos del servidor, en este caso estamos creando un nuevo cliente, y cuando creas un nuevo cliente, usualmente se utiliza el verbo post, por lo tanto siguiendo los principios de rest (recomendación para mantener tus urls en un buen orden) , nuevo cliente tomaría url, creamos un objeto de configuración y decirle que vamos a utilizar el metodo de post( siempre que vayamos a crear un nuevo registro), y eso es independiente de cualquier lenguaje de backend
- En cmd nos dice que hemos enviado un post
- Sea agrega lo que envíe a db.json e incluso le agrega un id, y si vemos nuestra API (pagina 4000) ya tienen un registro agregado

```

→ const url= 'http://localhost:4000/clientes';
•
export const nuevoCliente= async cliente => {
• try {
• await fetch(url, {
• method:'POST',// para crear un nuevo registro
• body: JSON.stringify(cliente), //contenido de peticion post hacia la url de /clientes, se envia de 2 formas, co
//mo string o como objeto
• //convertir de objeto a string (mandando cliente como metodo post a la url)
• headers:{
• 'Content-Type':'application/json'
• } // informacion de que tipo de datos estamos enviando, varia de acuerdo a lo que estés enviando al servid/
//or, si subes archivos es otro
•
• });
• window.location.href='index.html'; //nos lleva hacia pagina index una vez que se complete acción anterior cor
//rectamente, si detecta un error se sale y cae en el catch
• } catch (error) {
• console.log(error);
• }
→ }

```

# 272. Listar los clientes de la API

- Listar los diferentes elementos que tenemos en nuestra api
- En app.js
- Cuando queremos obtener todos los registros se envia un get, fetch envia un get por default
- Dice promise pending porque cuando el documento carga, yo mando llamar una funcion que aun no se ha completado ( completa y descarga todo el listado) aun no se cumple el promise de fetch para descargar todos los clientes por lo tanto detenemos ejecución hasta que se complete con un await

→ **async function** mostrarClientes(){

- `const clientes= await obtenerClientes();`
- 

`console.log(clientes);`

→ }

- El elemento **tr** representa a una fila de una tabla ( table ). En **HTML**
- <https://gist.github.com/juanpablogdl/35989dd9fdf45557e82e0544e8661e3f>

- En api.js

```
→ //obtiene todos los clientes // lee los datos de la API, por eso no toma nada
export const obtenerClientes = async () => {
 try {
 const resultado= await fetch(url);
 const clientes=await resultado.json();
 return clientes;
 } catch (error) {
 console.log(error);
 }
}
```

- En app.js

```
→ import{obtenerClientes} from './API.js';
•
(function(){
 const listado=document.querySelector('#listado-clientes');
 •
 document.addEventListener('DOMContentLoaded', mostrarClientes);
 •
 async function mostrarClientes(){
 const clientes= await obtenerClientes();
 •
 clientes.forEach(cliente => {
 const{nombre, email, telefono, empresa, id}= cliente;
 •
 const row=document.createElement('tr');
 •
 row.innerHTML +=`
 •
 <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200">
 •
 <p class="text-sm leading-5 font-medium text-gray-700 text-lg font-bold"> ${nombre} </p>
 •
 <p class="text-sm leading-10 text-gray-700"> ${email} </p>
 •
 </td>
 •
 <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 ">
 •
 <p class="text-gray-700">${telefono}</p>
 •
 </td>
 •
 <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 leading-5 text-gray-700">
 •
 <p class="text-gray-600">${empresa}</p>
 •
 </td>
 •
 <td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 text-sm leading-5">
 •
 Editar
 •
 Eliminar
 •
 </td>
 •
 `;
 •
 listado.appendChild(row);
 •
 });
 •
 }
}
→})();
```

# 273. Eliminar Clientes de la API

- Eliminar un registro recuerda que estamos usando modulos
  - Es importante saber que id quieres eliminar porque siguiendo los principios de rest cuando quieres eliminar, tienes que especificar el id
  - En app.js
- ```
→ function confirmarEliminar(e){  
•     if(e.target.classList.contains('eliminar')){  
•         const clienteld=parseInt(e.target.dataset.cliente); // leer el valor del id para saber a que le di click , convierte a entero//s  
•  
•         const confirmar=confirm('¿Deseas eliminar este registro?');// nativo de javaScript  
•  
•         if(confirmar){  
•             eliminarCliente(clienteld);  
•         }  
•     }  
→ }
```
- En API.js
- ```
→ // Eliminar un cliente... en la api y por lo tanto en el html
• export const eliminarCliente=async id =>{
• try {
• await fetch(`${url}/${id}`, { // inyecta el id en la url
• method:'DELETE'
• });
• } catch (error) {
• console.log(error);
• }
→ }
```

# 274. Obtener un Cliente para Editar

- Editar( extraer la info de un cliente en especifico(get a un id especifico) y después un put o un patch)
- Requiere dos cosas, extraer info del cliente a editar y guardar los cambios
- Al dar click en editar se llenen los campos(obtener registro de nuestra rest api)

En consola:

- Identificar que registro estamos visitando para hacer la consulta a la API y traernos los resultados  
→ `const parametrosURL = new URLSearchParams(window.location.search);`  
→ `parametrosURL.get('id');`
- Editar cliente:

```

→ import {obtenerCliente} from './API.js';
•
• (function(){
•
• //campos del formulario
• const nombreInput=document.querySelector('#nombre');
• const emailInput=document.querySelector('#email');
• const empresaInput=document.querySelector('#empresa');
• const telefonInput=document.querySelector('#telefono');
• const idInput=document.querySelector('#id'); // este no esta en el formulario pero esta oculta en el html y tambien lo tenemos que inyectar
•
•
• document.addEventListener('DOMContentLoaded', async () =>{
• const parametrosURL=new URLSearchParams(window.location.search);
•
• const idCliente= parseInt(parametrosURL.get('id')); //Identificar que registro estamos visitando para hacer la consulta a la API y traernos los resultados
•
• const cliente= await obtenerCliente(idCliente);
•
• mostrarCliente(cliente);
• });
•
• //llenar las partes del formulario
• function mostrarCliente(cliente){
• const {nombre, empresa, email, telefono, id}=cliente;
•
• nombreInput.value=nombre;
• empresaInput.value=empresa;
• emailInput.value=email;
• telefonInput.value=telefono;
• idInput.value=id;
• }
→ })();

```

- En el html editarcliente tenemos:
- →<input type="hidden" name="id" id="id" value="">

- API.js:
- //Obtiene un cliente por su ID(toma el id e imprime el cliente(toda la info))
- export *const* obtenerCliente= async *id* => {
  - try {
  - *const* resultado= await fetch(`\${url}/\${id}`);
  - *const* cliente= await resultado.json();
  - return cliente;
  - }
  - } catch (error) {
  - console.log(error);
  - }
- }



# 275. Guardar Cambios del Cliente a Editar

- Como guardar los cambios y reescribir el registro
- Escuchar para cuando demos click en guardar cambios, entonces primero validar, y luego reescribir el registro
- Cuando actualizas un registro, los principios de rest nos dicen que tienen que ser muy similar a eliminar, es decir tienes que poner la url / id que deseas actualizar en este caso
- Put es completo, patch es parcial
- Json server si es una api real, si reacciona a los diferentes verbos , pero sirve mas para probar y avanzar en el desarrollo de un api real, en lo que backend termina de realizar su trabajo

- En Api:

→ //actualizar un registro

```
• export const editarCliente= async cliente => {
• try{
• await fetch(`${url}/${cliente.id}`, {
• method: 'PUT',
• body: JSON.stringify(cliente),
• headers: {
• 'Content-Type': 'application/json'
• }
• });
• window.location.href='index.html'; // nos lleva a esta pagina
•
• }catch(error){
• console.log(error);
• }
→ }
```

- En funciones:

→ export function validar (obj){

```
• return !Object.values(obj).every(input => input !== ""); // sale true si almenos uno esta vacio
→ }
```

- En editar cliente:
- *function* validarCliente(e){
- e.preventDefault();
- 
- const cliente={
- nombre: nombreInput.value,
- email:emailInput.value,
- telefono:telefonoInput.value,
- empresa:empresaInput.value,
- id: parseInt(idInput.value)
- }
- 
- 
- if(validar(cliente)){
- //mostrar mensaje
- mostrarAlerta('Todos los campos son obligatorios');
- return;
- }
- 
- //reescribe el objeto
- editarCliente(cliente);
- }
-

# **Sección 45: Functional JavaScript - Qué es, Ventajas y Como Escribirlo**

# 276. Qué es Functional JS

- JavaScript funcional: es crear tu código utilizando funciones, pero no es como lo hemos hecho anteriormente ya que se siguen ciertas reglas
- Las funciones deben tomar una entrada y una salida de datos
- No se permite la modificación de los datos (por ejemplo si tienes un arreglo y quieres filtrar ciertos registros, lo ideal es crear un arreglo nuevo y no modificar el arreglo existente)
- Tiene una sintaxis más enfocada a la matemáticas
- Conceptos clave de la programación funcional:
- Inmutabilidad: los datos no deben modificarse (utilizar `const` casi siempre), no puedes reasignar una variable, no puedes modificar un valor
- Se separan funciones de datos (los array methods se utilizan mucho en la programación funcional de JavaScript)
- Se utilizan muchas funciones que retornan un nuevo dato o Array Methods, de esa forma tendremos funciones que entregan un resultado nuevo pero nunca modifican los datos (el arreglo u objeto original)
- **First-class functions** (veremos porque JavaScript es un excelente lenguaje para la programación funcional)
- **es poder crear funciones que parezcan cualquier variable como lo es function expression ej:**
- - `const suma = function(a,b){`
  - `return a+b;`
  - `}`
  - `const resultado = suma; //first-class function ya que suma es una funcion(sintaxis)`

Se tienen que cumplir los 3 principios: inmutabilidad, separar datos de funciones y first-class functions

# 277. First Class Functions

- Existen lenguajes funcionales, JavaScript no es un lenguaje funcional, sin embargo sus características permiten actuar como uno
- Cuando un lenguaje puede asignar una función como si fuera un string, un número o un booleano, quiere decir que ese lenguaje soporta first-class functions

→ `const suma=function(a,b){`

- `return a+b;`
  - `}`
  - `const resultado=suma; // si puedes asignar una función de esta forma, el lenguaje soporta first class function`
- `console.log(resultado(10, 20));`

# 278. Funciones como Argumentos

- Como pasar funciones como argumentos, característica de la programación funcional
- `const suma=(a,b) => a+b;`
- `const multiplicar = (a, b) => a*b;`
- `const sumarOMultiplicar= fn => fn(10,20); //funcion intermedia que toma una funcion o la otra`
- `console.log(sumarOMultiplicar(suma)); // funcion como argumento`
- `console.log(sumarOMultiplicar(multiplicar));`

# 279. Separar los Datos de las funciones

- Similares a array methods
  - Higher order functions : función que toma o retorna una función como argumento y básicamente la mayoría de los array methods son higher order functions
- 
- ```
const carrito = [  
  { nombre: 'Monitor 20 Pulgadas', precio: 500},  
  { nombre: 'Televisión 50 Pulgadas', precio: 700},  
  { nombre: 'Tablet', precio: 300},  
  { nombre: 'Audifonos', precio: 200},  
  { nombre: 'Teclado', precio: 50},  
  { nombre: 'Celular', precio: 500},  
  { nombre: 'Bocinas', precio: 300},  
  { nombre: 'Laptop', precio: 800},  
];  
  
// obtener productos con precio mayor a 400  
// const resultado=carrito.filter(producto=> {  
//   return producto.precio>400;  
// });  
  
// console.log(resultado);  
  
// llevando lo anterior a higher order functions  
const mayor400=producto=> {  
  return producto.precio > 400;  
}  
  
const resultado=carrito.filter(mayor400); //funcion que tomara una funcion (higher order functions)(creamos nuevo arreglo, no estamos modificando el arreglo original)  
console.log(carrito);  
→ console.log(resultado);
```


280. .map es muy utilizado en Functional JS

- Array methods son buenos candidatos para utilizarlos en la programación funcional
- .map
- Crear un arreglo que solo nos liste los nombres de los productos
- Permite crear nuevos arreglos, nuevos datos, sin modificar los existentes

```
→ const carrito = [  
  • { nombre: 'Monitor 20 Pulgadas', precio: 500},  
  • { nombre: 'Televisión 50 Pulgadas', precio: 700},  
  • { nombre: 'Tablet', precio: 300},  
  • { nombre: 'Audifonos', precio: 200},  
  • { nombre: 'Teclado', precio: 50},  
  • { nombre: 'Celular', precio: 500},  
  • { nombre: 'Bocinas', precio: 300},  
  • { nombre: 'Laptop', precio: 800},  
  • ];  
  •  
  • //aquí tampoco se modifica el arreglo original  
  • const obtenerNombres= producto => {  
  •   return producto.nombre;  
  • }  
  •  
  • const resultado=carrito.map(obtenerNombres);  
  • console.log(resultado);  
→ console.log(carrito);
```

281. Menos cantidad de código en tus Funciones

```
→ const carrito = [  
  • { nombre: 'Monitor 20 Pulgadas', precio: 500},  
  • { nombre: 'Televisión 50 Pulgadas', precio: 700},  
  • { nombre: 'Tablet', precio: 300},  
  • { nombre: 'Audifonos', precio: 200},  
  • { nombre: 'Teclado', precio: 50},  
  • { nombre: 'Celular', precio: 500},  
  • { nombre: 'Bocinas', precio: 300},  
  • { nombre: 'Laptop', precio: 800},  
  • ];  
  •  
  const obtenerNombres= p => p.nombre;  
  • const resultado=carrito.map(obtenerNombres);  
  • console.log(resultado);  
  •  
  const mayor400=p => p.precio > 400;  
  • const resultado2=carrito.filter(mayor400);  
→ console.log(resultado2);
```

- Pure functions, se utilizan mucho en react:
- Son funciones que retornan un dato, pero no modifican los valores de las variables, es decir si hay una variable global, o una funcion global, no van a modificar ese valor, si no que retornan un dato nuevo.
- Con una entrada de datos, es decir toman un parametro, deben de retornar la misma cantidad de datos que recibe la entrada, usualmente el resultado debe ser una nueva variable ya con el nuevo valor

→ //funciones puras o pure functions

- `const duplicar= numero => numero*2 ; // toma un numero y retorna un numero`

•

`// no modifican un valor original`

- `const numero1=20;`
- `const resultado=duplicar(numero1); //generamos un nuevo valor`

•

`console.log(resultado);`

→ `console.log(numero1);`

283. Funciones que Retornan funciones

- Funciones que retornan una función
 - Usualmente en librerías y en algunas documentaciones
- // función que retorna una función (una función que tiene dentro otra función)
- `const obtenerCliente = () => () => console.log('Juan Pablo');`
 -
- `const fn = obtenerCliente();` // asignamos función
- `fn();` // mandamos ejecutar la función

284. Closures

- Closures: van acompañados muchas veces de el scope

→ //scope, la funcion no sabe que la variable de afuer existe

- `const cliente='Juan';`
-
- `function mostrarCliente(){`
- `const cliente='Pablo';`
-
- `console.log(cliente);`
- `}`

→ `mostrarCliente();`

- Closure: Muchas veces quieres que un valor que esta dentro de una función, este disponible por fuera de una función, los closures son creados cada vez que se crea una funcion, y es una forma de acceder a una función o valor desde el exterior

→ //closure

- `const obtenerCLiente=() => {`
- `const nombre="Juan";`
-
- `function muestraNombre(){`
- `console.log(nombre);`
- `}`
-
- `return muestraNombre;`
- `}`
-

→ `const cliente=obtenerCLiente();`
`cliente()`

285. Partials y Currying

- Currying en wordpress(bloques gutember(editor), react para crear sitios web dinámicos , antes era php):
 - Dividir una función que toma mas de un parámetro, en argumentos de forma parcial
- `const suma=(a,b,c)=> a+b+c;`
- //en tres partes
- `const parcial =a =>b => c => suma(a,b,c);`
 -
 - `const primerNumero=parcial(5);`
 - `const segundoNumero=primerNumero(4);`
 - `const resultado=segundoNumero(3);`
- `console.log(resultado);`
-
- Con menos código:
- `const parcial =a =>b => c => suma(a,b,c);`
-
- `const resultadoParcial=parcial(5)(4)(3);`// ¿que hace? aplica currying y partials dividiendo función en pequeñas partes
- `console.log(resultadoParcial);`
-
- Casi no se ve en producción, sirve para entrevistas

286. Composition

- Composition: una alternativa a las clases , a ganado mucha popularidad sobre las clases:
- Es escribir muchas funciones, e ir utilizando en tus objetos lo que creas que vas a necesitar
- Se utiliza cuando tienes funciones que se pueden compartir entre objetos.
- Escribes una función que puedes utilizar en diferentes objetos y se los vas asignando para que puedas utilizarla
- En lugar de crear una clase y heredar, vas creando funciones que tu mismo vas armando y defines que funciones son necesarias para cada objeto
- Escribir funciones de uso general e irlas utilizando en tus objetos
- Muy comun hoy en dia los proyectos con composition, el manager(persona encargada del proyecto) define si se utilizan clases o composition

Sección 46: Dominando JavaScript

287. Scope

- Preguntas para obtener un trabajo:
- Es el alcance de una variable, es cuando creas una variable y puede ser vista, ya sea por una función, o por un bloque de código.
- Existen 2 tipos de scope:
- Scope global
- Scope en una función o en un bloque de código
- tiene prioridad la variable que este en el scope donde se esta llamando
- ```
////////////////////
// const cliente='Juan';
.
// function mostrarCLiente(){
// console.log(cliente);
// }
// mostrarCLiente(); // se muestra juan porque es una variable global
.
////////////////////////////////////
//si la variable esta adentro y el console.log se da por fuera
// function mostrarCLiente(){
// const cliente='Juan'; // error porque variable solo E entre estas llaves (scope de una función o bloque de código)
// }
// console.log(cliente);
.
// mostrarCLiente();
.
////////////////////////////////////
```

```

• // const cliente='Juan';
• // const cliente='Pedro'; // error porque la variable ya ha sido definida
•
• // function mostrarCliente(){
• // console.log(cliente);
• // }
•
• // mostrarCliente();
•
• //////////////////////////////////////
• // const cliente='Juan';// esta en el scope global
•
• // function mostrarCliente(){
• // const cliente='Pedro'; // Aqui ya se detectan como 2 variables diferentes, y te detecta pedro, si quito esta linea, muestra Juan, esto pasa porque tiene prioridad en el scope
• // console.log(cliente);
• // }
•
• // mostrarCliente();
•
• //////////////////////////////////////
• // const login=true;
•
• // function clienteLoagueado(){
• // const cliente='Juan';
• // console.log(cliente); //scope por bloque
•
• // if(login){
• // const cliente='Admin';
• // console.log(cliente); //scope por bloque
• // }
• // }
•
• //clienteLoagueado();
•
• //////////////////////////////////////

```

```
→ const login=true;
•
 function clienteLoagueado(){
• const cliente='Juan';
• console.log(cliente); //scope por bloque
•
 if(login){
• console.log(cliente); // antes de variable, por lo tanto toma juan // (también toma las de mayor jerarquía)
• const cliente='Admin';
•
• }
• }
→ clienteLoagueado();
•
 // puedes crear variables con el mismo nombre siempre que estén en diferente scope
```

# 288. Hoisting

- Se refiere a como funcionan los contextos de ejecución de JavaScript
- Existen 2 fases, una de creación y una de ejecución, en la primera se crean todas las variables, se registran, y en la segunda se ejecutan. En la primera registra las funciones, y en la segunda las manda llamar
- ¿Cuál es la diferencia entre function expression y function declaration?
- Cuando utilizas function declaration, primero puedes utilizar la función y después declararla, y no marca error

```
→ obteneCliente('Juan');
• function obteneCliente(nombre){
• console.log(`EL nombre del cliente es ${nombre}`); // aqui primero registra la function y despues la manda llamar
→ }
•
```

- Con function expression como tiene una sintaxis como si fuera una variable no se puede utilizar la función antes de declararla porque marca error

```
• //////////////////////////////////////
```

```
→ obteneCliente2('Pablo');
• const obteneCliente2 = function(nombre){
• console.log(`EL nombre del cliente es ${nombre}`)
→ }
•
```

```
 // es como si tuvieramos esto
→ obteneCliente2('Pablo');
• const obteneCliente2; // por eso marca error, porque esto esta como undefined
•
 obtenerCliente2=function(nombre){
→ console.log(`EL nombre del cliente es ${nombre}`)
```

# 289. Coercion

- Conversión automática explícita o implícita de valores de un tipo dado hacia otro
- `// coercion implícita: se esta forzando a que javaScript lo haga y lo modifique`
  - `const numero1=20;`
  - `const numero2="40";`
  - `console.log(numero1+numero2);` `// javaScript lo modifica de forma implícita imprime 2040 lo convierte a string`
  - `// coercion explícita, esta usualmente requiere utilizar una funcion`
- `console.log(Number(numero2));`
  - `console.log(numero1.toString());`
  - `const pedido=[1,2,3,4];`
  - `console.log(pedido.toString());`
  - `console.log(JSON.stringify(pedido));` `// de arreglo a string`

# 290. Implicit Binding

- Palabra reservada this, E diferentes tipos de this, se les conoce como bindings
- E implicit binding, new binding etc.
- Implicit binding:

→ `const usuario = {`

- `nombre: 'Juan',`
- `edad: 20,`
- `informacion(){`
- `console.log(`Mi nombre es ${this.nombre} y mi edad es ${this.edad}`); // que se encuentran en este objeto`
- `},`
- `mascota:{`
- `nombre:'Hook',`
- `edad: 1,`
- `informacion(){// metodo`
- `console.log(`Mi ombre es ${this.nombre} y mi edad es ${this.edad}`)// no se mezcla con lo de arriba porque`
- `busca en este objeto de mascota`
- `}`
- `}`
- `}`
- 

`usuario.informacion();`

→ `usuario.mascota.informacion();`

# 291. Explicit Binding

- Se utilizan 3 funciones: call, bind, apply
- ¿Cuál es la diferencia entre call, apply y bind?
- Call: existe en todas las funciones de JavaScript, incluso en las que tu creas y puedes pasarle un objeto o un arreglo dentro de esa función .
- Cuando utilizas .call y vas a pasarle un arreglo, tienes que pasarle cada elemento del arreglo de forma individual con su posición en el arreglo, Es como tener una función y unirle valores externos
- Apply: Si puede tomar todo el arreglo
- Bind: es igual a call, pero te crea una nueva función

```
→ //Explicit binding ...
•
function persona(el1,el2){
 console.log(`Mi nombre es ${this.nombre} y Escucho ${el1} y ${el2}`);
}
•
//-----
const informacion={
 nombre: 'Juan'
}
•
const musicaFavorita=['Heavy Metal', 'Rock '];
•
//-----
•
persona.call(informacion, musicaFavorita[0], musicaFavorita[1]);
•
persona.apply(informacion, musicaFavorita)
•
const nuevaFn=persona.bind(informacion, musicaFavorita[0], musicaFavorita[1]);
→ nuevaFn();
```

# 292. new Binding

- Puedes crear un nuevo objeto con el object constructor y vas a tener acceso a la palabra reservada this
- Es lo que se conoce como programación orientada a objetos en versiones anteriores de JavaScript
- Window binding: Cuando mandas llamar una variable JavaScript la manda buscar a la ventana global si no la encuentra en variables del programa

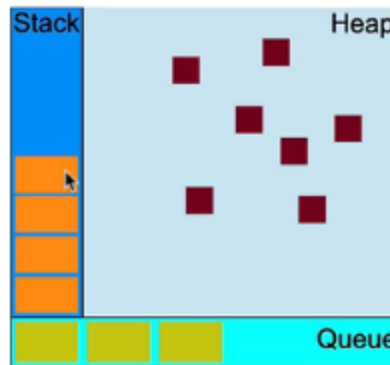
```
→ // New binding
•
function Auto(modelo, color){
 this.modelo=modelo; // este es el new binding, cada vez que creas un nuevo objeto, tienes acceso a este binding
 this.color=color;
}
•
const auto=new Auto('Camaro', 'Negro');
console.log(auto);
•
////////////////////////////////////
•
// otro tipo de binding
//window binding
window.color='negro'; //asignar una variable a la ventana global
•
function hola(){
 console.log(color);
}
→ hola();
```



# 293. Event Loop o Loop de Eventos en JS

- En español modelo de concurrencia y loop de eventos: es como se va a ejecutar el código en JavaScript, es decir que tiene más prioridad?, una función, una variable, un promise etc
- El código de JavaScript, es como se dice, de un solo hilo, es decir se ejecuta en una línea, solamente sucede una cosa a la vez y cada vez que se va completando una, se manda llamar la siguiente línea, y eso se le conoce como el loop de eventos, es decir va ejecutando tareas mientras halla algo disponible
- <https://www.youtube.com/watch?v=8aGhZQkoFbQ&t=119s>
- En JavaScript hay eventos que tienen más prioridad que otros, eso se conoce como el loop de eventos

- Las Funciones y el código se colocan en el stack (pila) dependiendo de la naturaleza de cada función, y otras se van colocando en el Queue
- Console log son los primeros que se ejecutan, por lo tanto se van directamente al stack, en lo que se ejecutan, las demás se terminan yendo al queue (setTimeout y promise) <https://developer.mozilla.org/es/docs/Web/JavaScript/EventLoop>

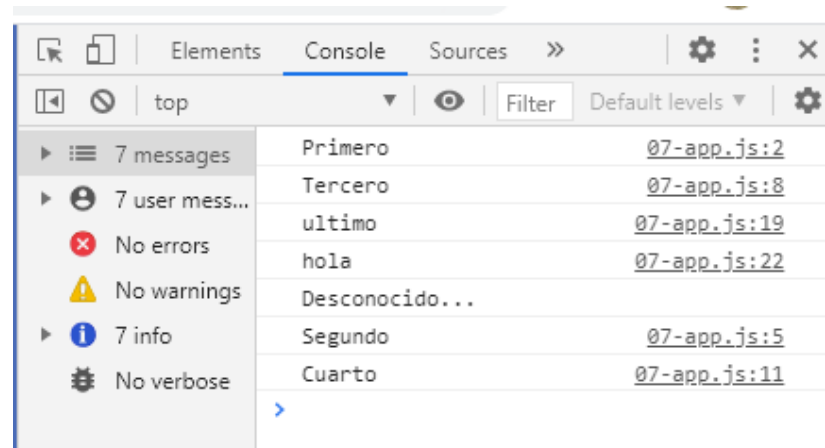


- Después de que se despachan los console log, JavaScript saca del queue (cola) y coloca en el stack lo siguiente que va a ejecutar, promise tiene más prioridad que setTimeout
- Funciones tienen una mayor prioridad que el promise

```

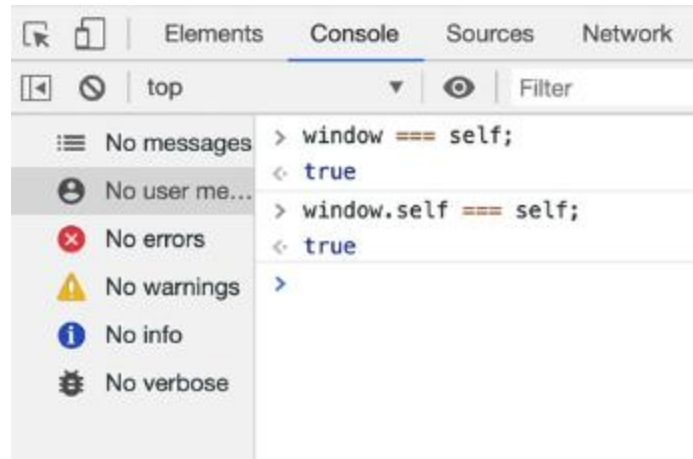
→ console.log('Primero');
•
• setTimeout(() => {
• console.log('Segundo');
• }, 0);
•
• console.log('Tercero');
•
• setTimeout(() => {
• console.log('Cuarto');
• }, 0); // el setTimeout no empieza en cero, es algo así como 30
•
• new Promise(function(resolve){
• resolve('Desconocido...')
• }).then(console.log)
•
• console.log('ultimo')
•
• function hola(){
• console.log('hola')
• }
→ hola();
•
• Respuesta:

```



# 294. Qué es self?

- Self existe en service workers o progressive web applications
- Self se refiere a la ventana global y es igual a window, sin embargo self se utiliza mucho en los service workers y en los web workers y en un service worker no esta disponible la palabra window, por lo tanto se utiliza self ( básicamente se refiere a la ventana global)



```

→ // similar al DOMContentLoaded
• self.onload={() => {
• console.log('Ventana Lista');
→ }
•
 ///otra forma de usar self

→ const producto={
• nombre:'Monitor 20 pulgadas',
• precio:30,
• disponible:true,
• mostrarInfo:function(){
• const self=this;
• return `El producto: ${self.nombre} tiene un precio de ${self.precio}`
• }
• }
→ console.log(producto.mostrarInfo());
•
 //otra forma de usar self
→ window.nombre='Monitor 20 pulgadas';
•
 const producto={
•
• precio:30,
• disponible:true,
• mostrarInfo:function(){
• return `El producto: ${self.nombre}` // se refiere a la ventana global
• }
• }
→ console.log(producto.mostrarInfo());

```

# **Sección 47: Service Workers y Progressive web apps (PWA)**

# 295. Que son los Service Workers

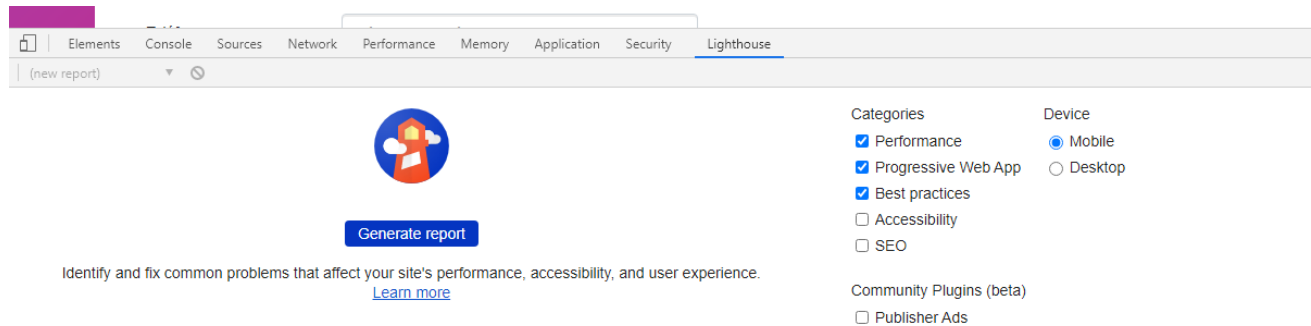
- **Progressive web Apps(aplicaciones web progresivas) (PWA) y funcionan con alguno llamado service workers**
- Características:
- cargan toda la informacion en menos de 5 segundos
- Instalable: se pueden navegar o instalar en tu navegador o telefono movil con una aplicación nativa
- Soporte offline: pueden funcionar incluso sin conexión a internet
- **Service workers:** es la base de una PWA. Son scripts que estan corriendo todo el tiempo detrás de escenas, por lo tanto el service worker va a estar escuchando cuando hay conexión, cuando no hay conexión etc, y va a estar cargando o mostrando cierta información
- Funcionan offline
- No tienen acceso al DOM: no funciona sobre el DOM
- Cargan de forma instantanea: cuando lo registras en el codigo, automaticamente se manda llamar el service worker
- Pueden sincronizar datos detrás de escena o sin interferir en la navegacion
- Supongamos que estas creando un proyecto que almacena cierta información de un formulario, pero por algo se pierde la conexión a internet, puedes guardar la informacion en cache, y un avez que vuelva la conexión a internet, sincronizar esos datos con los de un servidor, similar a uber ya que la app funciona sin conexión a internet de forma limitada, pero te dice que tienes que habilitar la conexión a internet para poder tener una buena experiencia de usuario

- **Funciones no disponibles en service workers**
- No soporta window, utiliza self
- No utiliza document, utiliza caches
- No utiliza localStorage, utiliza fetch, puede ser muy similar al fetch que ya tenemos, sin embargo tiene ciertas funciones específicas para una PWA



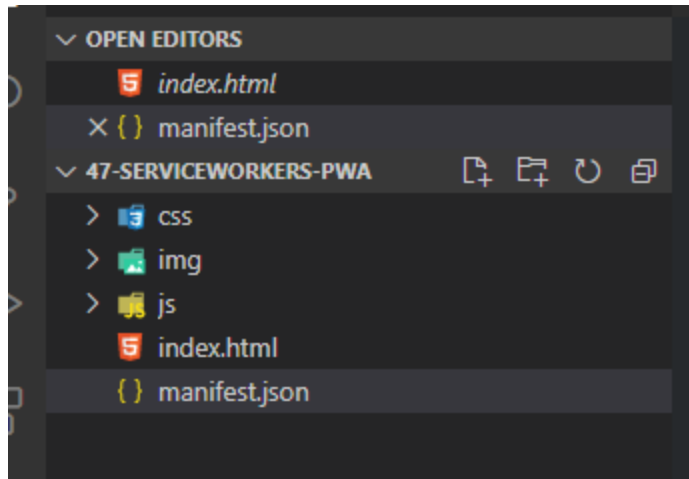
# 296. Generando un Reporte de Lighthouse

- Vamos a crear nuestra primera PWA
- Inspeccionar > lighthouse > categoria quito Seo y accesibility y generamos reporte



- El reporte nos va a decir que tenemos que mejorar para que sea una PWA
- Para que sea PWA tiene que estar servida en un dominio con https o también en local host
- Apple-touch-icon en html:
  - `<link rel="apple-touch-icon" href="img/icons/icon-96x96.png">`
  - `<meta name="apple-mobile-web-app-capable" content="yes">`
  - `<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">`
  - `<meta name="theme-color" content="black">`

- Creamos un nuevo archivo manifest.json, es obligatorio para crear una PWA



- Display// como se va a ver interface al usar PWA ¿aplicación nativa? Podemos hacer que se vea mas nativa(sin barra o mantener barra)

| Display Mode | Description                                                                                                                                                                                                                                                                                                           | Fallback Display Mode |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| fullscreen   | Se utiliza toda la pantalla disponible no se muestran elementos del user agent chrome.                                                                                                                                                                                                                                | standalone            |
| standalone   | La aplicación se mostrará como una app independiente. Así la aplicación puede tener su propia ventana, su propio icono en el lanzador de aplicaciones, etc. En este modo, the user agent excluirá los elementos de interfaz para controlar la navegación, pero puede incluir otros elementos como la barra de estado. | minimal-ui            |
| minimal-ui   | La aplicación se mostrará como una app independiente, pero tendrá un mínimo de elementos de interfaz para controlar la navegación. Estos elementos podrán variar según navegador.                                                                                                                                     | browser               |
| browser      | La aplicación se abrirá en una pestaña nueva del navegador o una ventana nueva, dependiendo del navegador y plataforma. Esto es por defecto.                                                                                                                                                                          | (None)                |

- En manifest.json:
- ```

→ {
  • "name": "APV", //administrador de pacientes de veterinaria
  • "short_name": "APV",
  • "start_url": "/index.html", //pagina de inicio ¿app nativa?
  • "display": "standalone", // no tenga la barra de navegacion que tienen todos los navegadores móviles
  • "background_color": "#D41872",
  • "theme_color": "#D41872",
  • "orientation": "portrait", // esta app va a funcionar unicamente con el celular de pie

  • //todas las versiones de los iconos
  • "icons": [//definir serie de iconos que se van a utilizar dependiendo de la densidad de pixeles que tenga esa pantalla, no todas son lo mismo
    {
      • "src": "img/icons/icon-72", //donde se va a encontrar este icono
      • "type": "image/png",
      • "sizes": "72x72"
    }
  ]
}
→

```

- En img vienen iconos de diferentes tamaños (los iconos los genero con una pagina web, no tienes que generar todos los iconos de diferentes tamaños, hay una pagina web, donde subes un icono y te genera todas las diferentes versiones(es decir lo hace en automático)
- Una vez que tenemos nuestro manifest, hay que registrarlo en el html en cualquier lugar, mientras este dentro del head
→ `<link rel="manifest" href="manifest.json">`
- Volvemos a generar el reporte
- Sigue marcando que no tenemos un PWA porque no hemos registrado el que se conoce como service worker, pero dice que nuestro manifest ya cumple los criterios para ser instalado, nos hacen falta otras cosas por ejemplo que funcione offline, tener un service worker etc, pero ya aparecen mas cosas en color verde
- Si abrimos inspeccionar> application>manifest encontramos toda la info del manifest

- Parte de service workers como registrarla
- <https://gist.github.com/juanpablogdl/72f048d5a2b4d70bb6b720bc1bf15a9c>

297. Detectar el soporte de Service Workers

- Apv tiene el código que hicimos en capítulos anteriores
- En App vamos a revisar si nuestro navegador soporta el service worker y si si, lo vamos a registrar <https://caniuse.com/?search=service%20workers> pagina de si puedo usar service worker
- Crear nuevo archivo sw.js en la raíz del proyecto(fuera de js)
→ // verificar si soporta el service workers
- ```
if('serviceWorker' in navigator){
```
- ```
  //registrando service worker
```
- ```
 navigator.serviceWorker.register('./sw.js')// retorna un promise
```
- ```
    .then(registrado => console.log('Se instalo correctamente...', registrado))
```
- ```
 .catch(error => console.log('Fallo la instalacion...', error));
```
- ```
  }else{
```
- ```
 console.log('Service Workers no soportados');
```
- ```
  }
```
- En Application>service workers> source sw.js ya aparece y esta activo y ejecutándose correctamente

298. Instalar y Activar un Service Worker

- Diferentes metodos que existen en el service workers
- Como instalar y activar el service worker
- Application> service worker> Unregister elimina el service worker y si recargo, se vuelve a instalar

→

//cuando se instala el service worker, se ejecuta una vez cuando el service worker se instala, si recargamos //ya no se ejecuta de nuevo similar a index db cuando se creba la BD

- self.addEventListener('install', e => {
- console.log('Instalado el service worker');
- console.log(e);
- });

//Activar el service worker, no aparece esto en consola, pero en Application>service workers> skipWaiting //se activa, entonces se ejecuta y ya aparece en consola

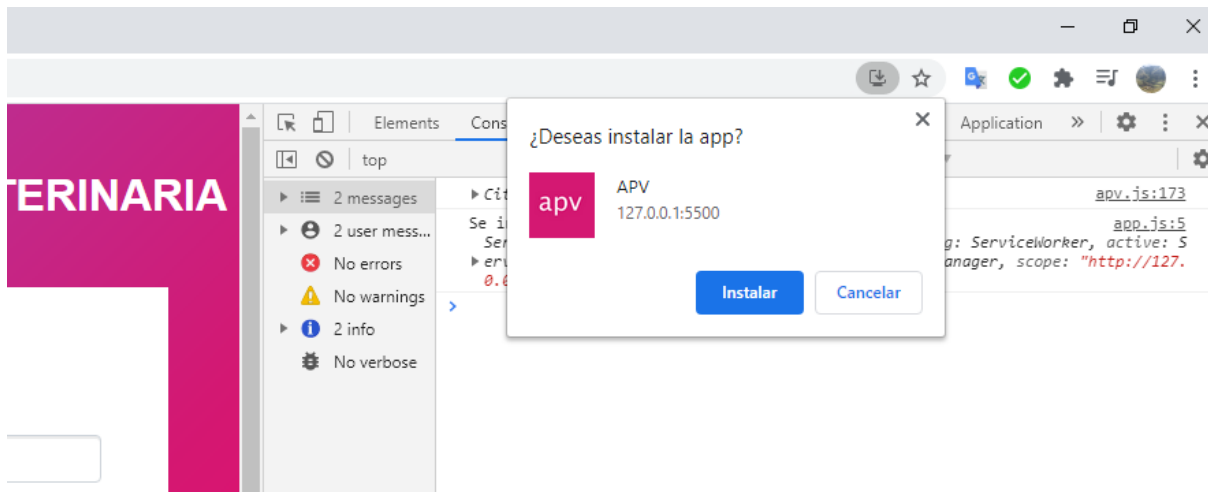
- self.addEventListener('activate', e =>{
- console.log('Service worker Activado');
- console.log(e);
- })

- Cada uno tiene ciertas funciones, por ejemplo cuando se instala es un buen lugar para cachear ciertos archivos, cuando se activa es un buen lugar para nuevas versiones de nuestra PWA

299. Hacer una PWA Instalable

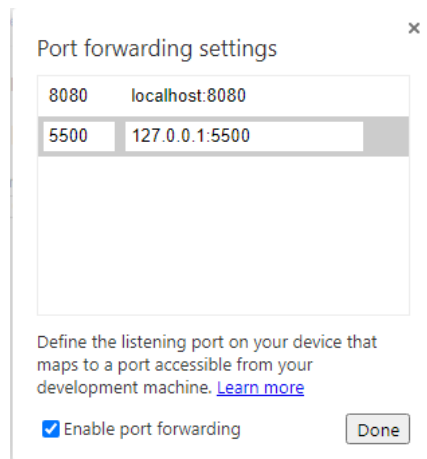
- Como instalar nuestra PWA
- Una característica de la PWA es que se pueden instalar incluso en un navegador de escritorio como este, o también en un teléfono móvil, eso le va a dar la apariencia de una aplicación nativa
- Para que una PWA se pueda instalar tienes que tener 3 cosas:
- Un **manifest** válido
- Tener un dominio https o ser un **local host**
- Tienes que tener registrado el eventListener de fetch(falta)
- En sw.js:
 - //registrar el fetch
 - //Evento fetch para descargar archivos estáticos
 - self.addEventListener('fetch', e =>{
 - console.log('fetch...', e);
 - })

- Instalamos la aplicación como una aplicación de chrome, vemos como desaparece la barra de navegación en la parte superior, y aparece como si fuera una app de escritorio



- El local host de un emulador es diferente al local host de una computadora, porque emulador es una maquina virtual
- Para instalarlo en el emulador de teléfono, tenemos que servir el sitio web desde local host: (yo uso teléfono porque no pude usar android studio y en teléfono desbloqueamos opciones de desarrollador: ajustes>acerca del teléfono>version de software 7 veces> conectamos a compu por usb> activamos depurador usb en: ajustes> opciones de desarrollador> depuración USB
- Inspeccionar>...>more tools> remove devices> aparece link en la parte de abajo> damos click en open> copiamos url de nuestra pagina web en cuadro> y vemos que no se abre la pestaña, por lo tanto la cerramos > port forwarding> port 5500

- Ponemos esto, y done



- Y en la parte inferior del teléfono aparece agregar APV al home screen, esto nos permite instalar nuestra PWA.
- Si volvemos a sacar el reporte ya aparece PWA y aparece todo verde en instalable

300. Cachear Archivos

- Como hacer que nuestra PWA sea mas rápida y funcione sin conexión a internet
- Para detener el servidor: al detenerlo se quita la conexión en la pagina web(no funciona pagina web sin una conexión a internet



- Para añadir soporte de nuestra aplicación, y en sw.js, en install es un buen lugar para hacer el catching de los archivos de nuestro proyecto
- En sw.js:

```

→ //nombre del cache
• const nombreCache='apv-v1';
•
const archivos=[ //archivos que vamos a estar cacheando, y en caso de que no tengamos conexion a internet va a obtener una copia de ese cache o leer e
//sos datos de cache, y va a mostrarlos, eso es lo que hace una PWA muy rapida
• //tambien en caso de que halla conexion a internet y ya visitamos el sitio web, entonces va a cargar los datos de cache
• // si tenemos 2 paginas, a y b , visitamos pag a eso descarga el cache, si vas a la pag b, tambien descarga el cache, si por algo el visitante solo visita una p
//agina, pero despues sin conexion intenta visitar la pagina b no va a funcionar, podrias agregar esa
• //segunda pag al cache, pero si agregas todo el sitio web a cache, va a ser algo lento ese catching y una PWA debe ser rapida
•
• '/', // cacheamos la pagina principal
• 'index.html', //cacheando index.html
• // puedes agregar mas paginas pero no agregar demasiadas porque lo hace mas lento
•
// para apariencia cacheamos:
• 'css/bootstrap.css',
• 'css/styles.css', //tambien podrias cachear imagenes si tuvieras, puedes cachear fuentes, cualquier archivo en realidad que sea necesario para que tu a
//pp funcione sin conexion a internet
•
• 'js/app.js',
• 'js/apv.js'
• ]

//cuando se instala el service worker, se ejecuta una vez cuando el service worker se instala, si recargamos ya no se ejecuta de nuevo similar a index db cu
//ando se creba la BD
• self.addEventListener('install', e => {
• console.log('Instalado el service worker');
•
//espera hasta que se hallan descargado todos los archivos
• e.waitUntil(
• caches.open(nombreCache)
• .then(cache =>{
• console.log('cacheando'); //los vamos cacheando
• cache.addAll(archivos)//los agregamos al cache //addAll para agregar un arreglo de archivos, si solo fuera un archivo solo se le pone add
• })
• )
→ });

```

- Se guarda en application> cache storage> ahí estan los archivos cacheados

301. Agregar soporte Offline

- Como traernos la copia de cache para que sea mas rápido ya que esto solo los cachea, aun no mejora el performance de nuestra app, tenemos que decirle que utilice la copia de cache
 - Como mostrar los archivos que tenemos en cache para que sea mas rapido la descarga de nuestra app
 - En sw.js
- ```
→ //registrar el fetch
```
- //Evento fetch para descargar archivos estáticos
  - self.addEventListener('fetch', e =>{
  - console.log('fetch...', e);
  - 
  - e.respondWith(//dale esta respuesta, una vez que estemos haciendo este fetch
  - caches.match(e.request) // identificar el request que se esta haciendo
  - .then( *respuestaCache* => {
  - return *respuestaCache* //en caso de que sea igual a lo que tenemos en cache, entonces cargamos el cache
  - })
  - 
  - )
- ```
→ })
```


- Si vamos a network vemos que nuestros archivos vienen del service worker
- Si vamos a application> service worker> offline, sigue funcionando correctamente
- Volvemos, quitamos offline y generamos reporte y aparece que es rápida

302. Agregar una página de error cuando no hay conexión

- Cuando visitas una pagina que no E, muestre una pagina de error
- Como agregar una pagina de error si alguien intenta navegar sin conexión a internet decirle que necesita internet
- Creamos un archivo error.html

- Error.html:
- <!DOCTYPE html>
- <html lang="en">
- <head>
- <meta charset="UTF-8">
- <meta name="viewport" content="width=device-width, initial-scale=1.0">
- <meta http-equiv="X-UA-Compatible" content="ie=edge">
- <meta name="Description" content="PWA JS.">
- <title>Administrador de Pacientes</title>
- <link rel="manifest" href="manifest.json">
- <link rel="stylesheet" href="css/bootstrap.css">
- <link rel="stylesheet" href="css/styles.css">
- <link rel="apple-touch-icon" href="img/icons/icon-96x96.png">
- <meta name="apple-mobile-web-app-capable" content="yes">
- <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
- <meta name="theme-color" content="black">
- </head>
- <body>
- <h2 class="text-center my-5 titulo">Admin de Pacientes de Veterinaria</h2>
- <div class="container mt-5 p-5">
- <h1 class="text-center">Hubo un error, esta pagina no se puede mostrar</h1>
- </div><!--.container-->
- <script src="js/app.js"></script>
- </body>
- </html>

- Como mostrar esta pagina:
- En sw.js
- ```
self.addEventListener('fetch', e =>{
 • console.log('fetch...', e);
 •
 • e.respondWith(//dale esta respuesta, una vez que estemos haciendo este fetch
 • caches
 • .match(e.request) // indentificar el request que se esta haciendo
 • .then(respuestaCache => (respuestaCache ? respuestaCache : caches.match('error.html'))))
 •)
 → })
```
- Y agregamos **'/error.html'** al arreglo de archivos que se van a cachear
- Si ya no reconoce el manifest podemos clear storage, quitamos el modo offline del service worker, vamos al boton recargar del navegador y seleccionamos la opción 3
- **ERRORES EN CONSOLA!!!!!!!!!!!!!!**
- En firefox la parte de cache esta en almacenamiento>almacenamiento en cache>ap-v1

# 303. Como Realizar nuevas secciones del PWA

- Como crear una nueva versión, porque si hacemos un cambio, y ya hay archivos en cache, las personas no van a poder ver esos cambios
- Cuando editamos el html no lee los cambios aunque nos cambiemos a online, si limpiamos el cache y recargamos ya muestra los cambios, pero esto los usuarios no lo van a hacer, yo soy la responsable de actualizar la PWA
- Para crear nuevas versiones que se van a guardar en nuevos archivos de cache solo tenemos que cambiarle el nombre en sw.js

→ `const nombreCache='apv-v3';`

- Cuando lancemos una nueva versión, tenemos que decirles que utilicen la ultima, pero tenemos que eliminar las otras 2, un buen lugar es en activate sw.js:

→

```
//Activar el service worker, no aparece esto en consola, pero en Application>service workers> skipWaiting se activa, entonces se ejecuta y ya aparece en consola
self.addEventListener('activate', e =>{
 console.log('Service worker Activado');
 .
 .
 e.waitUntil(
 caches.keys()//hasta que se cumpla la parte de obtener los archivos cache que aparecen en inspeccionar
 .then(keys => {
 //console.log(keys);
 .
 return Promise.all(
 keys.filter(key => key !== nombreCache) // solo traiga el que tiene mismo nombre que esta en este archivo
 .map(key => caches.delete(key))// elimine los que sobran
)
 })
)
 → });
```

- Renombramos archivo y recargamos y ya solo aparece la ultima versión
- PWA te pueden permitir acceder a la ubicación del dispositivo, a la cámara, incluso realizar pagos de forma segura, es un tema bastante grande hoy en día

- Para eliminar otro problema en el reporte, en el manifest:

→ {

- "src": "img/icons/icon-196.png",
- "type": "image/png",
- "sizes": "196x196",
- "purpose": "any maskable"

→ },

# **Sección 48: Design Patterns o Patrones de Diseño en JavaScript**

# 304. Qué son los Design Patterns?

- Patrones de diseño
- Son soluciones típicas a problemas comunes en desarrollo de software, E en todos los lenguajes de programación(soluciones universales), y cada patrón es como un plano que se puede personalizar para resolver un problema de diseño en el código.
- Beneficios:
- Son soluciones a problemas de diseño de código(si te encuentras con el problema de que estas repitiendo mucho código, seguramente al adoptar un patrón de diseño, todos esos problemas se van a resolver)
- Son soluciones probadas
- Son soluciones conocidas por todos, y evitan la forma de escribir código como cada quien entiende
- Categorías de patrones de diseño:
- De creación: permiten crear objetos y permiten la reutilización del código
- Estructura: explican como deben de comunicarse los objetos y classes en grandes proyectos
- De comportamiento: se encargan de cómo se comportan y comunican los objetos
- Hay muchos, Aprenderemos los básicos para cualquier desarrollador de software



# 305. Class Pattern

- Si no funciona eliminar el service worker del capitulo anterior en application
- Class patern: utilizar clases para la creacion de los objetos, patron de diseño de creacion , va a definir como deben de crearse los objetos, y en mucho trabajos te van a decir, hay que utilizr clases para la creacion de todos los objetos

→ // Class Pattern

- ```
class Persona{  
  constructor(nombre, email){  
    this.nombre=nombre;  
    this.email=email;  
  }  
}
```
- ```
const persona=new Persona('Juan', 'correo@correo.com');
```
- ```
console.log(persona);
```

→ // puedes crear muchos objetos siguiendo este plano

306. Constructor Pattern

- Se utiliza una clase base que sirve como plano para que las demás clases hereden sobre esta, en otros lenguajes se les conoce como clases abstractas, también se le conoce como un antipatron.
- Usualmente cuando utilizas este patron de diseño es porque el lenguaje soporta clases abstractas, es decir esas clases no se pueden instanciar, solamente se pueden extender o heredar, JavaScript no soporta algo como eso, pero en otros lenguajes si (las clases son algo nuevo en JavaScript)

→ //constructor pattern

```
•  
class Persona{  
•   constructor(nombre, email){  
•       this.nombre=nombre;  
•       this.email=email;  
•   }  
• }  
•  
class Cliente extends Persona{  
•   constructor(nombre, email, empresa){  
•       super(nombre, email); //hace referencia a los atributos de la clase padre  
•       this.empresa=empresa; // nueva propiedad (utilizamos buena parte del constructor padre y agregamos nuevos atributos al constructor hijo)  
•   }  
• }  
•  
//tambien puedes instanciar persona:  
• const persona=new Persona('Juan', 'correo@correo.com');  
• console.log(persona);  
•  
const cliente=new Cliente ('Miguel', 'cliente@cliente.com', 'codigo con juan');  
→ console.log(cliente);
```

307. Singleton

- El mas común de todos los patrones de diseño de todos los lenguajes de programación
- No te permite crear multiples instancias de una misma clase, en cambio siempre te va a retornar el objeto instanciado, llega a ser bastante útil tener un objeto con toda la información para no estar creando múltiples instancias

→ // Singleton

```

let instancia=null;

class Persona{
  constructor(nombre,email){
    if(!instancia){
      this.nombre=nombre;
      this.email=email;
      instancia=this;
    }else{
      return instancia;
    }
  }
}

//instanciando
const persona= new Persona('Juan', 'correo@correo.com');
console.log(persona);

const persona2= new Persona('Juan2www', 'correo@correo.com'); // si vuelvo a crear una instancia retorna la primera instancia asignada
→ console.log(persona)

```

308. Factory

- Forma de crear objetos basados en cierta condición, es decir van a compartir algunos atributos, pero en base a esas condiciones, esos atributos, algunos se reutilizan, pero otros son diferentes
- Soluciona problemas de código, si por ej creas un constructor visual como wix o como gutember, un factory es una excelente forma de hacerlo, porque basado en ciertas condiciones va a ir instanciando diferente classes
- No utilizarlo para todo, solo en ciertos casos
- Recordar que Dessing pattern the dice como organizar el código en un proyecto mas grande

```

→ //Factory- Crea objetos basados en ciertas condiciones
•
class InputHTML {
•   constructor(type, nombre){
•       this.type=type;
•       this.nombre= nombre;
•   }
•
•   //metodo:
•   crearInput(){
•       return`<input type="${this.type}" name="${this.nombre}" id="${this.nombre}">`; //basado en los datos de arriba va a crear diferentes inputs
•   }
• }
•
//segunda clase
• class HTMLFactory{
•   crearElemento(tipo, nombre){
•       switch(tipo){ // vamos a evaluar
•           case 'text':
•               return new InputHTML('text', nombre)
•           case 'tel':
•               return new InputHTML('tel', nombre)
•           case 'email':
•               return new InputHTML('email', nombre)
•
•           default:
•               return;
•       }
•   }
• }
•
// como utilizar HTMLFactory para que cree las instancias de InputHTML y nos traiga el crearInput
•
const elemento= new HTMLFactory();
const inputText= elemento.crearElemento('text', 'nombre-cliente');
console.log(inputText.crearInput()); // todo esto solo para crear un input de tipo text
•
const elemento2= new HTMLFactory();
const inputText2= elemento2.crearElemento('tel', 'telefono-cliente');
console.log(inputText2.crearInput());
•
//estamos instanciando la misma clase, pero cada objeto va a ser diferente basado en ciertas condiciones
•
const elemento3= new HTMLFactory();
const inputText3= elemento3.crearElemento('email', 'email-cliente');
console.log(inputText3.crearInput()); // todo esto solo para crear un input de tipo text
→ //asi podemos crear diferentes tipos de objetos basados en diferentes tipos de condiciones

```

309. Module

- Como organizar código, existen diferentes formas, el mas común en JavaScript es module, ese cambio hace poco
- Patrones de diseño para organizar código, los primero 4 fueron de creación, ahora veremos organizar código

→ //Module Pattern

- • `const mostrarCliente = nombre => {`
 - `console.log(nombre);`
 - `}`
- `export default mostrarCliente;`

→ //module pattern anteriormente, hoy se recomienda export default o export const para lograr lo mismo 95% de los proyectos previos a ES6 utilizan esta forma

- • `const modulo1=function(){`
 - `const nombre='Juan';`
 - • `function hola(){`
 - `console.log('hola')`
 - `}`
 - • `return {`
 - `nombre,`
 - `hola`
 - `}`
 - `})();`

- En otro archivo podemos llamar lo que esta en el anterior

→ `console.log(modulo1.nombre);`

→ `modulo1.hola();`

310. Mixin Pattern

- Forma de agregar funciones a una clase una vez que ha sido creada
- Crear un objeto con múltiples funciones, tener alguna clase, y si llegas a tener otra clase, puedas añadirle las funciones que tienes en el objeto para que se convinen
- La ventaja es que puedes tener un objeto con funciones y crear una segunda clase y copiar las funciones del objeto en la nueva clase


```

→ class Persona{
•   constructor(nombre, email){
•       this.nombre=nombre;
•       this.email=email;
•   }
• }
•
• class Cliente{
•   constructor(nombre, email){
•       this.nombre=nombre;
•       this.email=email;
•   }
• }
•
• //tenemos funciones externas que vamos a compartir entre diferentes clases
• //creando objeto
• const funcionesPersona={
•   mostrarInformacion(){
•       console.log(`Nombre Persona: ${this.nombre} Email:${this.email}`)
•   },
•   mostrarNombre(){
•       console.log(`Mi nombre es ${this.nombre}`)
•   }
• }
•
• //Añadir funcionesPersona a la clase de Persona
• Object.assign(Persona.prototype, funcionesPersona);
• // -----copiar lo que tenemos en el prototype de persona
• // -----Agregamos funciones de persona
• //ya en el prototype tendremos mostrarInformacion
• Object.assign(Cliente.prototype, funcionesPersona);
•
• const cliente= new Persona('Juan', 'correo@correo.com');
• console.log(cliente);
• cliente.mostrarInformacion();
• cliente.mostrarNombre();
•
• const cliente2= new Persona('Cliente', 'cliente@cliente.com');
• console.log(cliente2);
• cliente2.mostrarInformacion();
→ cliente2.mostrarNombre();

```

311. Namespace

- Design pattern para la organización del código
- Ayuda a evitar colisión con nombres en el scope global de JavaScript, útil en algunas apps que tienden a crecer bastante
- La idea del namespace es crear un objeto global alrededor de tu app y agregar todas las funciones dentro, en lugar de crear múltiples funciones y objetos que se puedan acceder de forma global
- Siempre inicia como un objeto, sobre el cual colocamos todos los datos, todas las funciones y estaremos realizando todas las operaciones, de esa forma se evita el choque de nombres de funciones en el scope global
- Muy común en la organización de código de JavaScript
- Se puede acceder a los valores en la ventana global

```

→ const restaurantApp={};
•
• //le definimos una nueva propiedad al objeto (todo queda en namespace de restaurant app)
• restaurantApp.platillos =[
• {
•   platillo:'Pizza',
•   precio:25
• },
• {
•   platillo:'Hamburguesa',
•   precio:20
• },
• {
•   platillo:'Hot Dog',
•   precio: 20
• }
• ];
•
• //agregandole funciones
• restaurantApp.funciones={
•   mostrarMenu: platillos =>{
•     console.log('Bienvenidos a nuestro menu');
•
•     platillos.forEach((platillo, index) => {
•       console.log(`${index}: ${platillo.platillo} $$${platillo.precio}`)
•     });
•   },
•   ordenar: id =>{
•     console.log(`Tu Platillo: ${restaurantApp.platillos[id].platillo} se esta preparando`);
•   },
•   agregarPlatillo:(platillo,precio) => {
•     const nuevo={
•       platillo,
•       precio
•     };
•     restaurantApp.platillos.push(nuevo);
•   }
• }
•
• restaurantApp.funciones.ordenar(1);
• restaurantApp.funciones.agregarPlatillo('Taco', 20);
• const{platillos} = restaurantApp;
• //para acceder: muy dificilmente va a chocar con otro objeto o funcion que se llame igual
→ restaurantApp.funciones.mostrarMenu(platillos);

```

312. Mediator o intermediario

- Patrón de diseño que se comunica con diferentes objetos a la vez
- El mediador define objetos ya localizados para objetivos específicos
- Este mediador va a requerir diferentes objetos
- E mas de 30 patrones de diseño, en trabajo te dicen que patrón de diseño utilizar

```

→ function Vendedor(nombre){
  •   this.nombre=nombre;
  •   this.sala=null;
  • }
  •
  Vendedor.prototype={
  •   oferta:(articulo, precio) => {
  •     console.log(`Tenemos el siguiente articulo ${articulo}, iniciamos con un precio de ${precio}`);
  •   },
  •   vendido:Comprador=>{
  •     console.log(`Vendido a ${Comprador}`);
  •   }
  • }
  •
  function Comprador(nombre){
  •   this.nombre=nombre;
  •   this.sala=null; // una vez que sea creada la subasta, se van a llenar esto en comprador y vendedor, y van a estar en una misma sala
  •
  • }
  •
  Comprador.prototype={
  •   oferta: (cantidad, comprador) => {
  •     console.log(`${comprador.nombre} : ${cantidad}`)
  •   }
  • }
  •
  function Subasta(){ //aqui se agrupan para comunicarse uno con otro
  •   // colocar al vendedor y al comprador en la misma sala
  •   let compradores={};
  •
  •   return{
  •     registrar: usuario => {
  •       compradores[usuario.nombre]= usuario; // creando una propiedad nueva (saber la info del usuario )
  •       usuario.sala=this;
  •     }
  •   }
  • }
  •
  //Crear objetos (forma de comunicar todos estos objetos al mismo tiempo)
  • const juan=new Comprador('Juan');
  • const pablo=new Comprador('Pablo');
  • const vendedor=new Vendedor('Vendedor de Autos');
  • const subasta=new Subasta();
  •
  • // antes de que halla ofertas , la subasta tiene que registrarlos
  • //subasta es el mediador, el que comunica tanto vendedor con compradores(comunica diferentes objetos)
  • subasta.registrar(juan);
  • subasta.registrar(pablo);
  • subasta.registrar(vendedor);
  • //esta de arriba es la sala, para participar en la compra
  •
  vendedor.oferta('Mustang 66', 300);
  juan.oferta(350, juan);
  • pablo.oferta(450,pablo);
  • juan.oferta(500, juan);
  • pablo.oferta(700, pablo);
  vendedor.vendido('Pablo');

```

Sección 49: Debug, Medir Performance y Seguridad

313. performance.now() para conocer cuanto tarda en ejecutarse el código

- Como mejorar el performance de tus aplicaciones y como medirlo, también veremos el debugger de chrome y consejos sobre seguridad
- Para consultar o verificar el performance de nuestras apps muy común:
 - inspeccionar> network> recargar> descripción de que es lo que pasa y cuanto tiempo tarda en ejecutarse tus diferentes scripts, cuanto tiempo tardan en descargarse, pero si quieres medir cuanto tiempo tarda en hacer consulta, también aparece en network en la lista, si lo abrimos viene mucha info, y el tiempo que tarda en ms
- Si vez que esta corriendo lento , te permite identificar en que parte esta lenta la consulta, de esa forma sabes si es la API o si es algun archivo muy pesado
- Cualquier milisegundo que ahorres es ganancia en un proyecto con millones de visitas
- ```
function selectCriptomonedas(criptomonedas) {
 .
 //funcion que puedes utilizar para saber cuanto tiempo tarda en generarse todas estas opciones por medio de scripting
 // conocer el tiempo de ejecucion de pieza de codigo y compararla
 .
 .
 const inicio=performance.now();
 .
 // criptomonedas.forEach(cripto => {
 // const { FullName, Name } = cripto.CoinInfo;
 // const option = document.createElement('option');
 // option.value = Name;
 // option.textContent = FullName;
 // // insertar el HTML
 // criptomonedasSelect.appendChild(option);
 // });
 .
 for(let i=0; i<criptomonedas.length; i++){// este es mas rápido que el anterior
 // const { FullName, Name } = criptomonedas[i].CoinInfo;
 // const option = document.createElement('option');
 // option.value = Name;
 // option.textContent = FullName;
 // // insertar el HTML
 // criptomonedasSelect.appendChild(option);
 }
 .
 .
 const fin=performance.now();
 console.log(fin-inicio);
→ }
```

- Para medir tiempo de apis

→ *function* consultarAPI() {

- 

- ***const* inicio=performance.now();**

- 

- *const* { moneda, criptomoneda} = objBusqueda;

- 

- *const* url = `https://min-api.cryptocompare.com/data/pricemultifull?fsyms=\${criptomoneda}&tsyms=\${moneda}`;

- 

- mostrarSpinner();

- 

- fetch(url)

- - .then(*respuesta* => respuesta.json())

- - .then(*cotizacion* => {

- - mostrarCotizacionHTML(cotizacion.DISPLAY[criptomoneda][moneda]);

- - });

- ***const* fin=performance.now();**

- ***console.log*(fin-inicio);**

→ }



# 314. async o defer? cual utilizar

- Una forma de cargar mas rapido nuestros scripts
- Como cargar nuestros scripts para ganar algo de performance y las diferente formas de hacerlo
- En html en el script le puedes pasar algunos atributos E 2 async y defer
- Con asyn se completa al final (network), no se muestran las opciones porque esl escript se va a cargar y ejecutar inmediatamente una vez que ha sido descargado, html empiesa a descargar todo de arriba abajo y cuando llega a script, este se manda llamar y ejecuta las funciones que tiene, puede ser que a esas alturas aun no tengamos la respuesta de nuestra api, aun no se hallan creado las opciones y ya se esta mandando llamar el script, por eso no funciona
- Se usa cuando tienes codigo que no modifica la apariencia o el html inicial
- Asyn descarga y ejecuta el codigo inmediatamente que ha sido descargado

→ `<script src="js/app.js" async></script>`

- Defer no va a ejecutar el codigo del script hasta que el html este listo, gana un poco en performance, y todo funciona
- Descarga pero ejecuta hasta que el html este listo

→ `<script src="js/app.js" defer></script>`

# 315. Como utilizar debugger;

- En chrome
- Detener la ejecución y mostrarte todas las variables a esas alturas que tienes disponibles y sus valores, pones play para continuar con la ejecución, las que aparecen undefined es porque se definen después del debugger, es conforme se va ejecutando el código
- → `debugger; //` en `app.js`
- Conocer que funciones hay disponibles a esas alturas y que valores hay disponibles en las variables
- Para saber en que parte se esta consumiendo mas memoria > `memory> take snapshot`

# 316. Opciones para Ofuscar el código y ocultarlo

- Medidas de seguridad que tienes que tomar en cuenta
- Cualquiera puede ver el código de javascript sources> js> app.js
- Por lo tanto ofuscar el código, si tenemos una variable, la hace mas complicada de leer, pero no lo utiliza mucho porque así como hay herramientas para ofuscar el código, también hay técnicas para quitarle el ofuscamiento al código
- Pagina para ofuscar: <http://javascript2img.com/obfuscating.php>
- Podemos verlo en source, formatearlo {} pero será muy complicado de leer
- Pagina para quitar la ofuscacion: <https://beautifier.io/>
- No hay tantas medidas de seguridad del código de JavaScript porque corre en el cliente

# 317. Otras Medidas de seguridad

- Medidas para asegurar tus apps en JavaScript
- No almacenes passwords en localStorage, tampoco en indexDB , no imágenes, tarjetas de crédito, porque funcionan en el cliente y cualquiera podría acceder a ellos en cualquier momento
- El DOM Scripting ya escapa los datos( algunas vulnerabilidades que obtienes al usar innerHTML son evitadas utilizando el DOM Scripting) y evita riesgos de seguridad, por lo tanto utiliza textContent la mayoría de las veces
- Utiliza innerHTML solo cuando la fuente de los datos es segura, es decir estas consumiendo tu propia API

# formularios

- Valida en el cliente con JavaScript, para darle retroalimentación en tiempo real a tus visitantes(no confíes mucho en pasar esa validación), pero también valida en el servidor(se hace con node o con cualquier framework o lenguaje que funcione en el backend)
- Si deseas crear apps con autenticación de usuarios puedes utilizar json web token (JWT), técnica más utilizada, o si no quieres agregar tanto código, si no quieres programar, puedes utilizar el sitio web o la herramienta de Auth0

# Otras consideraciones

- Cuando trabajes con dependencias, utiliza una herramienta para verificar vulnerabilidades como snyk.io, en ella puedes agregar tu proyecto, y te va a decir si hay vulnerabilidades en algunas versiones o en otras
- Ofuscar el código si lo consideras necesario
- Hashea información sensible con la librería de bcrypt (si vas a crear un proyecto que utilice node para conectarse a una BD y autenticar usuarios , los passwords tendrán que ser hasheados utilizando bcrypt, de esa forma no van a ser visibles para nadie, incluso si te roban tu BD no va a verse comprometida la info de tus usuarios )

# **Sección 50: TESTING - Creando un Mini Framework**

# 318. Introducción al Testing

- Mejorar la calidad de tu software, evitando bugs
- Probar todos los diferentes escenarios puede ser complicado o tardado, pero hay herramientas que automatizan las pruebas de nuestros proyectos
- Liberar nuevas versiones sin las preocupaciones de que algo salga mal
- CONSIDERACIONES CON EL TESTING
- ¿Cuántas veces has agregado nuevas funciones a un proyecto existente pero desconoces si funciona bien con lo existente? (pero desconoces si se comunica bien con otras partes o si simplemente el proyecto sigue funcionando bien ), probablemente hagas la función como te la pidieron, pero no sabes en realidad si rompió algo mas (se previene con testing de tus apps web)
- También tener pruebas hará que una persona que no ha mantenido un proyecto (que no esta familiarizado con el proyecto conozca que es lo que hace cada parte)
- Importante tener el testing incluso como parte de la documentación para que otras personas sepan que es lo que hace cada parte de nuestro proyecto y como se integra con otras secciones
- no harás pruebas de todo, mas bien se recomienda hacer testing de cómo se integran las diferentes partes de tu aplicación (mas recomendada, mas utilizada en todos los proyectos web)



# Diferentes tipos de testing

- El mas común, el mas utilizado es:
- End to End- es el mas interactivo, simula algunos clicks, llenar formularios y asegurarse de que lo que se muestre en pantalla es lo que se desea ver. Se utiliza mucho cypress
- Integración: revisa que múltiples partes de nuestro proyecto funcionen bien, este lo podemos hacer con jest, al igual que las pruebas unitarias
- Unit: Revisa que cada parte por si sola funcione bien
- Static: revisar por errores en el código mientras vas escribiendo, también jest puede ser utilizado en esto, incluso typeScript puede ser utilizado en este tipo de testing, ya que por ejemplo en typeScript si defines una interface que acepta una forma de un objeto, es decir un nombre de un cliente como string, un balance que tiene que ser un numero, y si por algo en la programación le pasas, en lugar de un numero un string te va a marcar errores, entonces ya desde ahí typeScript te va a decir que hay error, que en esta parte definiste la interface que debe de ser un numero y le estas pasando un string
- HERRAMIENTAS PARA TESTING
- Cada tecnología tiene sus herramientas para testing, pero una muy popular es Jest, hay versiones para vueJS, Angular, TypeScript, Node, React, etc. Es sin duda la mas popular de todas y es necesario tener instalado Node.js
- Otra opción es cypress que es una herramienta un poco mas enfocada para hacer testings End to end (simula algunos clicks, simula que es lo que se ve en pantalla)

# 319. Creando un Mini Framework para Testing

- Como hacer testing sin herramientas y en realidad como funcionan estas herramientas de testing
  - Como hacer testing sin herramientas ya para poderlo ver con jest y sus beneficios
  - Hacerlo con console log
- ```
→ //Proobar 2 valores
```
- ```
function suma(a,b){
 return a+b;
}
```
  - ```
let resultado=suma(2,2);  
let esperado=3; // lo que se espera obtener
```
 - ```
if(resultado !== esperado){
 console.error(`El ${resultado} es diferente a lo esperado; la prueba no paso`); // o prueba mal implementada o
 funcion no hace lo que debe
}else {
 console.log('La prueba paso correctamente ');
}
```
- ```
→
```
- Pruebas de testing tienen funciones ya implementadas, y estas validaciones, tu puedes crear el tuyo
 - Jest te dice si la función esta mal o la implementación

320. Más Funciones a nuestro Framework

- Al final es un if lo que tendrían en jest
- Jest también tiene una función llamada test, o también llamada it, son lo mismo

```
→ //Probar 2 valores
•
• function suma(a,b){
•   return a+b;
• }
•
• function restar (a, b){
•   return a-b;
• }
•
• let resultado=suma(1,2);
• let esperado=3; // lo que se espera obtener
•
•
• //similar a lo que hacen jest
• expected(esperado).toBe(resultado);
•
• resultado=restar(10,5);
• esperado=5;
•
• expected(esperado).toBe(resultado);
•
• expected(esperado).toEqual(resultado);
•
• // en jest funciones tambien se llaman asi
• function expected( esperado ){
•   return {
•     toBe(resultado){
•       if(resultado !== esperado){
•         console.error('El ${resultado} es diferente a lo esperado; la prueba no paso'); // o prueba mal implementada o funcion no hace lo que debe
•       }else {
•         console.log('La prueba paso correctamente ');
•       }
•     },
•     toEqual(resultado){
•       if(resultado !== esperado){
•         console.error('El ${resultado} no es igual a lo esperado; la prueba no paso'); // o prueba mal implementada o funcion no hace lo que debe
•       }else {
•         console.log('La prueba paso correctamente ');
•       }
•     }
•   }
• }
•
• }
```

```
→ }
```

321. Probando Código Asincrono

- Agregando funciones asíncronas para tener cuando se cumple y cuando no se cumple
- Como crear la función de test y crear promises para en caso de que se cumpla o no la condición y aceptar o rechazar la prueba que estemos realizando
- En jest no te preocupas por escribir las funciones, solo te preocupas por utilizarlas

```

• //Proobar 2 valores
•
• function suma(a,b){
•   return a+b;
• }
•
• function restar (a, b){
•   return a-b;
• }
•
• //creando promise para poder utilizar try catch
• async function sumaAsync (a, b){
•   return Promise.resolve(suma(a,b));
• }
•
•
•
• let resultado=suma(1,2);
• let esperado=3; // lo que se espera obtener
•
•
• //similar a lo que hacen jest
• expected(esperado).toBe(resultado);
•
•
• resultado=restar(10,5);
• esperado=5;
•
• expected(esperado).toBe(resultado);
•
• expected(esperado).toEqual(resultado);
• //-----
• // en jest funciones tambien se llaman asi
•
• test ('Suma 10 + 20 y el resultado debe ser 30', async()=>{
•   const resultado=await sumaAsync(10,20);
•   const esperado=30;
•   expected(esperado).toBe(resultado);
• })
•
•
• async function test(mensaje, callback){
•   try{
•     await callback();
•     console.log('El Test: ${mensaje} se ejecuto correctamente');
•
•   }catch(error){
•     console.error('Error:');
•     console.error(error);
•   }
• }
•

```

```

function expected( esperado ){
  return {
    toBe(resultado){
      if(resultado !== esperado){
        console.error('El ${resultado} es diferente a lo esperado; la prueba no paso'); // o prueba/
        //a mal implementada o funcion no hace lo que debe
      }else {
        console.log('La prueba paso correctamente ');
      }
    },
    toEqual(resultado){
      if(resultado !== esperado){
        console.error('El ${resultado} no es igual a lo esperado; la prueba no paso'); // o prueba
        //mal implementada o funcion no hace lo que debe
      }else {
        console.log('La prueba paso correctamente ');
      }
    }
  }
}

```

Sección 51: JEST - Introducción al Testing en JavaScript con Jest

322. Primeros pasos con Jest

- Creamos un archivo llamado package.json
- Jest funciona con node.js, por lo tanto requiere ese archivo, en el cual podemos agregar algunos scripts, y también algunas dependencias
- Click derecho en carpeta 51> abrirlo con la terminal integrada >npm init> description> aprendiendo jest>author>julieta> yes // esto crea el archivo package.json(desde aquí podemos agregar algunos scripts, instalar software, instalar algunas dependencias, y mandarlas llamar desde este archivo que funciona con node)
- Vamos a instalar jest como una dependencia, E 2 tipos de dependencias, unas de desarrollo y E otras dependencias del proyecto, jest es una dependencia de desarrollo, porque hacemos el testing en desarrollo, no en producción
- Para que lo guarde como dependencia de desarrollo:
- En terminal:
 - npm i --save-dev jest // instala jest y crea una carpeta llamada node modules(es muy grande y tiene muchos archivos porque jest es una dependencia, pero tiene otras dependencias) y otro archivo llamado package.lock(nunca debes de modificarlo, guarda la referencia de las dependencias sobre lo que vas instalando) y en el package.json, como dependencias de desarrollo ya tenemos a jest
- Carpeta node modules> bin (serie de binarios, archivos que puedes ejecutar>jest
- En package.json:
 - {
 - "name": "51-testing-jest",
 - "version": "1.0.0",
 - "description": "Aprendiendo jest",
 - "main": "index.js",
 - "scripts": {
 - "test": "jest" // cuando mande llamar scripts va a ejecutar el binario de jest
 - },
 - "author": "Julieta Rosales",
 - "license": "ISC",
 - "devDependencies": {
 - "jest": "^26.6.3"
 - }
 - }
 -

- Para mandar llamar script de jest en terminal:
 - `npm run test`
 - `npm test`
 - `npm t` // los 3 hacen lo mismo
 - Crear tus archivos de testing (archivos que jest pueda entender):
 - En testing jest crear nueva carpeta `__tests__` (nombre que jest puede comprender, va a buscar los archivos de testing ahí)
 - Crear nuevo archivo en esa carpeta `holamundo.js` que seria nuestra primer prueba, y cada prueba utiliza la función llamada `test`
- `//test('Hola Mundo en Jest', () => {});` //en llaves tendríamos que poner el cuerpo de nuestra prueba
- `//mandar llamar en terminal npm t que ejecuta el script que tenemos en el package .json que se llama test, y manda llamar a jest`
 - `// lo busca en __tests__ y nos dice que paso la prueba`
- `//it('Hola Mundo en Jest', () => {});` //hace lo mismo que anterior
- Si no quieres tener una carpeta dedicada a tests, puedes mover `holamundo` a `js` y renombrarla como `holamundo.test.js`, si lo vuelves a llamar en terminal, lo encuentra y ejecuta la prueba, (hace lo mismo que anterior párrafo)
 - El lo regresa a `test` porque para el es mas cómodo
 - No vas a crear un archivo para cada prueba, puedes agruparlas dentro de otra función que se llama `describe`
 - Es buena practica que las pruebas que estén relacionadas, estén en un solo archivo, las vas a ir agrupando en `describe` y la mandas llamar con `npm t` y te dice que paso las 2 pruebas
- `describe('Grupo de pruebas', () => {`
- `test('Hola Mundo en Jest', () => {});`
 - `test('Otro hola mundo', () => {});`
- `})`

323. Probando Strings

- Como hacer testing a un string
- Describe y test son funciones que E en jest, E muchas otras
- Si quieres validar que una cadena de texto tenga cierta cantidad de caracteres, jest ya tiene ciertos métodos, si quieres ver cuantos elementos hay en un arreglo, jest también tiene ciertos métodos

→ // una variable tenga un string asociado

-
- //metodos para strings
- `const password="123456";`
-
- `describe('Valida que el password no este vacio y tenga una estension de 6 caracteres', () => {`
- `test('Que el password tenga 6 caracteres', () =>{`
- `expect(password).toHaveLength(6);`
- `// -----lo que vamos a validar`
- `// -----cuantos caracteres quieres comprobar que tenga`
- `});`
-
- `test('Password no vacio', () => {`
- `expect(password).nottoHaveLength(0);`
- `})`
-
- `}) //agrupar diferentes pruebas en un mismo archivo`

324. Probando Arrays

- Como validar un arreglo
- Crear un nuevo archivo ejemplo-arrays.js en `__tests__`

→ `const carrito=['Producto 1', 'Producto 2', 'Producto 3'];`

- - `describe('Testing al carrito de compras', () => {`
 - `test('Probar que el array tenga 3 elementos', () => {`
 - `expect(carrito).toHaveLength(3);`
 - `});`
 - `test('Verificar que el carrito no este vacio', () => {`
 - `expect(carrito).nottoHaveLength(0);`
 - `})`
- `})`

325. Probando Objetos

- Como hacer testing a un objeto
- Crear un nuevo archivo ejemplo-objetos.js en `__tests__`

```
→ const cliente={
  nombre:'Juan Pablo',
  balance: 500
};

// si el cliente tiene mas de 400 entonces pertenece a la categoría de premium
describe('Testing al cliente', () =>{
  test('El cliente es premium', () => {
    expect(cliente.balance).toBeGreaterThan(400);
  });

  test('Es Juan Pablo', () => {
    expect(cliente.nombre).toBe('Juan Pablo');
    // ---- que sea igual a juan pablo
  });

  //para comprobar que no sea un valor para evitar falsos positivos
  test('No es otro cliente', () => {
    expect(cliente.nombre).not.toBe('Pedro');
  });

  test('No tiene 500', () => {
    expect(cliente.balance).not.toBe(400);
  });
});
→ })
```

326. Probando Funciones

- Testing a una función
- Lo mas común es que escribas testing hacia funciones porque las funciones te permiten integrar datos con otras partes de tu aplicación, es muy común comprobar que tus funciones hagan lo que deben de hacer
- Crear archivo ejemplo-funciones.js en __tests__
- En lugar de enviar a la consola y llenando los datos manualmente, jest te simplifica, puedes probar diferentes funciones al mismo tiempo muy rapido

→ //probar que funciones hagan lo que uno espera

- `function suma(a,b){`
- `return a+b;`
- `}`
- `function restar (a, b){`
- `return a-b;`
- `}`
- `describe('Testing a las funciones de suma y resta', () => {`
- `test('Suma de 20 y 30', () => {`
- `expect(suma(20,30)).toBe(50);`
- `});`
- `test('Resta de 10-5', ()=>{`
- `expect (restar(10, 5)).toBe(5);`
- `});`
- `// no tenga ese valor`
- `test('Que la suma 10 y 10, no sea 10', () => {`
- `expect(suma(10,10)).not.toBe(10);`
- `});`
- `test('Que la resta de 10-5 no sea otro valor', () => {`
- `expect(restar(10,5)).not.toBe(2);`
- `})`
- `});`
- `// con esto verificamos que las funciones trabajan bien`

327. Introducción a Snapshots

- Comprobar un objeto de una forma mas sencilla
- Crear archivo ejemplo-snapshots.js en __tests__
- Como probar el objeto completo , una seria convertir objeto en string con JSON.stringify, y comparar el objeto con la cadena de texto de stringify, pero jest nos ofrece los snapshots
- Snapshots son datos que se almacenan en un string, se crean en una carpeta aparte y sobre ella podemos comparar si es el mismo dato, es decir si es el mismo cliente, o si es otro cliente
- Si quiero cambiar para un nuevo cliente, puedo entrar al snapshot y cambiar el nombre del cliente, o eliminar todos los snapshots, otra opcion es si quieres actualizar el snapshot, en terminal le pones(forma mas recomendable o eliminar snapshot para generar otro)
→ `npm t -- -u` (actualiza el snapshot con lo que tengas en la prueba(reescribe))
- Te permiten almacenar algo en un BD por así decirlo y te permiten compararlo
- En lugar de probar cada una de las partes de un objeto, puedes utilizar esta sintaxis para probar todo el objeto
→

```
const cliente={  
  nombre: 'Juan 2', //si cambio esto, marca error porque detecta que es diferente al snapshot que hay almacenado, son diferentes, no pasa la prueba  
  balance: 500,  
  tipo: 'Premium'  
};  
  
//como probar el objeto completo  
  
describe('Testing al cliente', () => {  
  test('Es Juan pablo', () => {  
    expect(cliente).toMatchSnapshot();  
  }); // si ejecutamos prueba crea una nueva carpeta llamada snapshots y si la abrimos tiene un string con la forma del objeto, estos archivos solo jest d/  
  //ebe de modificarlos  
});
```

328. Agregar Babel para realizar pruebas de funciones en otros archivos

- funciones que ya tenemos en un proyecto Existente y no queremos reescribirlas, como podemos importarlas?
- Creamos archivo ejemplo-modulos.js en __tests__, donde queremos importar una función de la carpeta de funciones de js
- Habilitar babel, permite que puedas escribir código, el mas nuevo, llevarlo a una versión compilada(transpilada) de una versión anterior, entonces puedes utilizar la nueva sintaxis aunque no este soportada, porque la versión transpilada, si soporta ese código(version que internet explorer si pueda leer), también sirve si tienes que dar soporte a un navegador muy viejo
- Habilitar babel crear un archivo .babelrc en carpeta principal
- Desde terminal instalar una dependencia babelpreset
- `npm i --save-dev @babel/preset-env`
-----dependencia de desarrollo
- Ahora en carpeta .babelrc podemos decirle la configuración que debe tener este proyecto
- `//targets, que versiones vamos a soportar , permite habilitarle imports a un proyecto de node`
- ```
{
 "presets": [
 [
 "@babel/preset-env", {
 "targets": {
 "node": "current"
 }
]
]
]
}
```
- El constructo de la clase, en el archivo UI da problemas
- Correr prueba en terminal
- `npm t`

→ En ejemplo-modulos.js:

- `import {suma} from '../js/funciones.js';`
- `describe('Suma de 2 numeros', () => {`
- `test('Sumar 10 y 20, debe dar como resultado 30', () => {`
- `expect(suma(10,20)).toBe(30)`
- `})`

→ `});`

En funciones.js

→ `export function suma(a,b){`

- `return a+b;`

→ `}`

# 329. Testing al Proyecto de Citas - Agregar Cita

- Como agregar babel y ieslint y webpack para tener tu propio framework para crear apps modernas en JavaScript y gracias a webpack tener un buen performance, a ieslint seguir buenas practicas con tu código y gracias a babel poderlo transpilar a versiones anteriores y utilizar el código mas nuevo
- Probar clase de Citas por medio de un snapshot
- Crear archivo en `__tests__ citas.js`, y gracias a babel ya podemos importar clase de citas dentro de archivo de pruebas

```
→ import Citas from '../js/classes/Citas';
•
• describe('Probar la clase de Citas', () => {
•
• const citas=new Citas(); //instanciando
•
• test('Agregar una nueva cita', () => {
• const citaObj = {
• mascota: 'Laika',
• propietario: 'Julieta',
• telefono: '123456',
• fecha: '10-12-2020',
• hora: '10:30',
• sintomas: 'solo duerme'
• };
•
• citaObj.id=Date.now();
•
• citas.agregarCita(citaObj); // se usa la instanciada
•
• //prueba
• expect(citas).toMatchSnapshot();
• // -----
• verificar que info que estamos pasando a la clase, se este guardando correctamente, escribe un snapshot con el arreglo y la info
•
• });
• });
→ });
• Prueba en terminal npm t
```



## 330. Testing al Proyecto de Citas - Actualizar y Eliminar

- Como trabajar con la edición de citas
- Limpiar snapshot o actualizarlo en terminal:  
→ `npm t -- -u //` también hace las pruebas
- Ahora en el snapshot tendremos los 2 snapshots de las 2 pruebas para compararlos

```

→ import Citas from './js/classes/Citas';
•
• describe('Probar la clase de Citas', () => {
•
• const citas=new Citas(); //instanciando
• const id=Date.now(); //global para que sepa que id es, y sea el mismo id para ambas pruebas
•
• test('Agregar una nueva cita', () => {
• const citaObj = {
• id,
• mascota: 'Laika',
• propietario: 'Julieta',
• telefono: '123456',
• fecha: '10-12-2020',
• hora:'10:30',
• sintomas: 'solo duerme'
• }; //parametros que le paso estan aqui porque no los importamos
•
•
• citas.agregarCita(citaObj); // se usa la instanciada
•
• //prueba
• expect(citas).toMatchSnapshot();
• // -----verificar que info que estamos pasando a la clase, se este guardando correctamente, escribe un snapshot con el arreglo y la info
•
• });
•
• test('actualizar cita', () => {
• const citaActualizada={
• id,
• mascota: 'Nuevo Nombre',
• propietario:'Julieta',
• telefono:'19999393',
• fecha:'10-12-2020',
• hora:'10:30',
• sintomas:'Solo duerme'
• };
•
• citas.editarCita(citaActualizada);
•
• expect(citas).toMatchSnapshot(); // es correcto porque esta actualizando basado en el id
• });
•
• test('Eliminar Cita', () => {
• citas.eliminarCita(id);
•
• expect(citas).toMatchSnapshot();
• })
• });
•
• npm t -- -u

```

- Por lo tanto, métodos bien implementados
- Cypress ofrece herramientas de pruebas mas robustas, como se integran diferentes elementos ya en la pantalla
- Ya sea que desees implementar jest con react, con angular o con vuejs, serán escenarios específicos para cada plataforma, la base es escribir algo como lo que tenemos en los ejemplos

# **Sección 52: CYPRESS - Introducción al Testing con Cypress**

# 331. Qué es Cypress?

- Te ofrece pruebas end to end testing
- Es una herramienta para hacer testing, se le conoce como una herramienta de la siguiente generacion ya que ofrece pruebas que antes no se podian hacer, se parece a otra herramienta para hacer pruebas llamada selenium, pero cypres es mejor
- Con cypres es posible saber que esta viendo el usuario en pantalla
- Cuando escribes pruebas son de codigo, pero no sabes que es lo que el usuario experimenta
- [cypress.io](https://cypress.io)
- Mucho mas enfocado a lo que el usuario ve en pantalla
- PRUEBAS EN CYPRESS
- Es considerado end to end pero tambien ofrece pruebas unitarias y de integracion (como se integra lo que tenemos en un archivo con otro)
- Funciona en el navegador, por lo tanto puedes probar con cypress proyectos hechos en java, C#, python, php, Node, React o VueJS siempre y cuando se vea en un navegador
- INSTALANDO POR MEDIO DE NPM
- Terminal> npm init (para crear el archivo de package.json> npm i --save-dev cypress>npx cypress verify> npx cypress open
- **npx (que no instala toda la dependencia completa) open (abre la aplicación de escritorio) Y run( realiza las pruebas en el cli)**
- **Npm start correria los scripts de package.json**
- Se abre cypress y nos crea una carpeta llamada cypress y las carpetas que estan en la app son las mismas que estan en integration
- Las carpetas que vienen en la app son ejemplos de cómo utilizar cypress

# 332. Instalando y Primeros Pasos con Cypress

- En carpeta Integration es donde vas a colocar todas tus pruebas
- En plugins > index > realizar algunas configuraciones del comportamiento de cypress, algunos eventos y cosas similares
- Support > commands> puedes registrar tus propios comandos o reescribir los que vienen en cypress, se pueden instalar plugins que irian en la carpeta de support
- En carpeta cypress.json podemos cambiar la configuracion por default de cypress, ej modificar el tamaño del navegador

```
→ {
 • "viewportHeight": 1500,
 • "viewportWidth": 1200
→ }
```

- En terminal ctrl+c para detenerlo y vuelve a ejecutar npx cypress open para que vuelva a abrir la app de escritorio
-

- CREAR UNA NUEVA PRUEBA
  - Integration>nuevo archivo primer\_test.spec.js
  - ----- para que lo tome cypress automaticamente
  - Probar que se muestre administrador de pacientes de veterinaria
  - Cypres funciona unicamente en el navegador, no puede probar una conexión a una BD, solo muestra lo que se ve en pantalla, lo que el usuario ve en pantalla
  - Cypres trata de buscar que se cumpla la prueba 4 segundos, no funciona porque se tiene que conectar al proyecto, porque no sabe de su E
- `/// <reference types="cypress" />`
- `// tener un buen autocompletado`
  - 
  - `describe('Carga la pagina principal', () =>{`
  - `it('Carga la pagina principal', () => {`
  - 
  - `cy.visit("http://127.0.0.1:5500/52-Testing-Cypress/index.html");`
  - `// -----abrir url primero`
  - `cy.contains('h1', 'Administrador de Pacientes de Veterinaria');`
  - `// -----revisar si un elemento en el DOM E`
  - `cy.get('h1').should('exist');`
  - `});`
- `});`
- Pruebas enfocadas mas a lo que ve el usuario

# 333. Los Atributos en Cypress

- Buenas practicas para escribir selectores
- Agregar atributo data-cy y ponerle un nombre que pueda ser único en el proyecto, en el html, incluso un elemento como seleccionar por id, estas dependiendo de que pueda ser eliminado del DOM

→ `<h1 data-cy="titulo-proyecto" class="text-center my-5 titulo">Administrador de Pacientes de Veterinaria</h1>`

- Diferencia mayusculas de minusculas, también no pasa la prueba si le dejas un espacio
- Pruebas rigurosas y exactas de lo que el usuario debe ver en pantalla, o de la experiencia que deben de tener
- Get es como el querySelector del cypress
- Herramienta robusta para hacer testing moderno hoy en dia



```

→ /// <reference types="cypress" />
• // tener un buen autocompletado
•
describe('Carga la pagina principal', () =>{
• it('Carga la pagina principal', () => {
•
• cy.visit("http://127.0.0.1:5500/52-Testing-Cypress/index.html");
• // -----abrir url primero
•
• //verificar el elemento y su texto
• cy.contains('[data-cy="titulo-proyecto"]', 'Administrador de Pacientes de Veterinaria');
• // -----revisar si un elemento en el DOM E
•
• //cy.get('h1').should('exist'); // no es recomendable porque puede E muchos h1 y no sabe cual es el que debe probar
• // -----para obtener elementos del DOM(botones, imagenes, h1, parrafos)
• // -----comprobar si tiene algun texto o no
•
• //verificar que exista
• cy.get('[data-cy="titulo-proyecto"]').should('exist');// siempre utilizar el atributo para selectores
•
• //verificar que exists el elemento y contenga un texto en especifico
• cy.get('[data-cy="titulo-proyecto"]')
• .invoke('text')
• .should('equal', 'Administrador de Pacientes de Veterinaria');
•
• //Verificar el texto de las citas
• cy.get('[data-cy=citas-heading')
• .invoke('text')
• .should('equal', 'No hay Citas, comienza creando una');
•
• // que no sea igual
• cy.get('[data-cy=citas-heading]')
• .invoke('text')
• .should('not.equal', 'Julieta Rosales');
• });
→ });

→ <h2 data-cy="citas-heading" id="administra" class="mb-4"></h2> // (en html)

```

# 334. Validación de Formularios

- Crear nuevo archivo .spec.js
- `<form data-cy="formulario" id="nueva-cita">`
- También podemos hacerle una prueba a esa alerta
- Para eso en classes UI (para generar el atributo con código de JavaScript)
- `//Agregar data-cy`
- `divMensaje.dataset.cy='alerta'; //nombramos el atributo`
- `/// <reference types="cypress" />`
- describe('Valida el formulario', () => {  
 it('Submit al formulario y mostrar la alerta de error', () => {  
 cy.visit("http://127.0.0.1:5500/52-Testing-Cypress/index.html");  
 •  
 •  
 **cy.get('[data-cy="formulario"]')**  
 .submit(); // presiona en el boton de crear cita  
 •  
 •  
 // Seleccionar la alerta  
 **cy.get('[data-cy=alerta]')**  
 .invoke('text')  
 .should('equal', 'Todos los campos son Obligatorios')  
 •  
 •  
 //verificar si tiene una clase  
 cy.get('[data-cy=alerta]')  
 .should('have.class', 'alert-danger')  
 •  
 •  
 })  
 })  
→ });

```
// seleccionar la alerta
cy.get('[data-cy=alerta]')
 .invoke('text')
 .should('equal', 'Todos los campos son Obligatorios')
 .should('have.class', 'alert-danger')
```

Esto arca error  
porque se estaria  
probando el text  
por el invoke

```
// seleccionar la alerta
cy.get('[data-cy=alerta]')
 .should('have.class', 'alert-danger')
 .invoke('text')
 .should('equal', 'Todos los campos son Obligatorios')
```

Esto si pasa, pero es  
mas limpio separlo  
como el anterior

# 335. Escribir en Formularios y Crear Nueva Cita

- Como cypress maneja llenar estos formularios
- Crear nuevo archivo crear\_cita.spec.js
- En cypress.json se puede definir una url de base,

```
→ {
 • "viewportHeight": 1500,
 • "viewportWidth": 1200,
 • "baseUrl": "http://127.0.0.1:5500/52-Testing-Cypress"
 •
→ }
```

- como hemos hecho cambios, detengo el servidor cc^C y lo vuelvo a ejecutar npx cypress open> y en archivo spec.js solo poner:

```
→ cy.visit('/index.html'); // ya no tenemos que escribir toda la url
```

- ```

En html:
→ input data-cy="mascota-input" type="text" id="mascota" name="mascota" class="form-control" placeholder="Nombre Mascota">
• Asi con todas las partes del formulario en el html

• También puedes agregar atributo al boton para seleccionarlo

→ /// <reference types="cypress" />
• describe('Llena los campos para una nueva cita y la muestra', () => {
•   it('campos nueva cita', () => {
•
•     cy.visit('/index.html');
•
•     cy.get('[data-cy=mascota-input]')
•     // simular o hacer que cypres escriba en inputs es:
•     .type('Hook');
•     // -----lo que quieres escribir
•
•     cy.get('[data-cy=propietario-input]')
•     .type('Julieta');
•
•     cy.get('[data-cy=telefono-input]')
•     .type('1234566');
•
•     cy.get('[data-cy=fecha-input]') // si detecta que es tipo date y le paso numeros cypres te dice error, y el formato que se tiene que seguir
•     .type('2020-12-03');
•
•     cy.get('[data-cy=hora-input]') // formato especifico
•     .type('20:30');
•
•     cy.get('[data-cy=sintomas-textarea]')
•     .type('el gato solo come y duerme');
•
•     cy.get('[data-cy=submit-cita]')
•     .click(); //para click en el boton button
•
•     cy.get('[data-cy=citas-heading]')
•     .invoke('text')
•     .should('equal', 'Administra tus Citas');
•
•     // seleccionar la alerta
•     cy.get('[data-cy=alerta]')
•     .invoke('text')
•     .should('equal', 'Se agregó correctamente')
•
•     cy.get('[data-cy=alerta]')
•     .should('have.class', 'alert-success')
•
•   })
→ });

```

336. Editar Citas

- Al hacer la prueba vemos que dos alertas se juntan, por lo tanto en UI:

→ `btnEditar.dataset.cy='btn-editar';`

→ `const alertaPrevia=document.querySelector('.alert');`

•

`if(alertaPrevia){`

- `alertaPrevia.remove();`

→ `}`

- Luego hacemos la prueba en `editar_cita.spec.js`

```

→ /// <reference types="cypress" />
•
describe('llena los campos para una nueva cita y la edita', () => {
•
  it('campos nueva cita', () => {
•
•
    cy.visit('/index.html');
•
•
    cy.get('[data-cy=mascota-input]')
    // simular o hacer que cypres escriba en inputs es:
    .type('Hook');
    // -----lo que quieres escribir
•
•
    cy.get('[data-cy=propietario-input]')
    .type('Julieta');
•
•
    cy.get('[data-cy=telefono-input]')
    .type('1234566');
•
•
    cy.get('[data-cy=fecha-input]')// si detecta que es tipo date y le paso numeros cypres te dice error, y el formato que se tiene que seguir
    .type('2020-12-03');
•
•
    cy.get('[data-cy=hora-input]')// formato especifico
    .type('20:30');
•
•
    cy.get('[data-cy=sintomas-textarea]')
    .type('el gato solo come y duerme');
•
•
    cy.get('[data-cy=submit-cita]')
    .click(); //para click en el boton button
•
•
    cy.get('[data-cy=citas-heading]')
    .invoke('text')
    .should('equal', 'Administra tus Citas');
•
•
    // seleccionar la alerta
    cy.get('[data-cy=alerta]')// desde UI document
    .invoke('text')
    .should('equal', 'Se agregó correctamente')
•
•
    cy.get('[data-cy=alerta]')
    .should('have.class', 'alert-success')
•
  });

```

- it('Editar la cita', () => {
- cy.get('[data-cy="btn-editar"]') // desde archivo UI
- .click(); // se llenan todos los campos para editarlos
-
- cy.get('[data-cy=mascota-input]')
- .clear() // limpia el input y reescribe después para que no se escriban ambas cosas en el input
- .type('NUEVO HOOK');
-
- cy.get('[data-cy=submit-cita]')
- .click();
-
- cy.get('[data-cy=alerta]')
- .invoke('text')
- .should('equal', 'Guardado Correctamente');
-
- cy.get('[data-cy=alerta]')
- .should('have.class', 'alert-success')
- })
- });

337. Eliminar Citas

- Como eliminar
- Creamos un nuevo archivo eliminar_cita.spec.js

→

```
/// <reference types="cypress" />

describe('llena los campos para una nueva cita y la elimina', () => {
  it('campos nueva cita', () => {
    cy.visit('/index.html');

    cy.get('[data-cy=mascota-input]')
    // simular o hacer que cypress escriba en inputs es:
    .type('Hook');
    // -----lo que quieres escribir

    cy.get('[data-cy=propietario-input]')
    .type('Julieta');

    cy.get('[data-cy=telefono-input]')
    .type('1234566');

    cy.get('[data-cy=fecha-input]') // si detecta que es tipo date y le paso numeros cypress te dice error, y el formato que se tiene que seguir
    .type('2020-12-03');

    cy.get('[data-cy=hora-input]') // formato especifico
    .type('20:30');

    cy.get('[data-cy=sintomas-textarea]')
    .type('el gato solo come y duerme');

    cy.get('[data-cy=submit-cita]')
    .click(); //para click en el boton button

    cy.get('[data-cy=citas-heading]')
    .invoke('text')
    .should('equal', 'Administra tus Citas');

    // seleccionar la alerta
    cy.get('[data-cy=alerta]') // desde UI document
    .invoke('text')
    .should('equal', 'Se agregó correctamente')

    cy.get('[data-cy=alerta]')
    .should('have.class', 'alert-success')
  });
});
```

- `it('Eliminar una cita', () => {`
- `cy.get('[data-cy=btn-eliminar]')// desde UI`
- `.click();`
-
- `cy.get('[data-cy=citas-heading')`
- `.invoke('text')`
- `.should('equal', 'No hay Citas, comienza creando una');`
-
- `cy.get('[data-cy=alerta]')// desde IU`
- `.invoke('text')`
- `.should('equal', 'Se elimino correctamente');`
-
- `cy.get('[data-cy=alerta]')`
- `.should('have.class', 'alert-success');`
-
- `})`
- `});`

338. Que más Ofrece Cypress?

- Viene en 2 versiones(en terminal)
 - `npx cypress open`, esto te abre la app de escritorio basada en electron y tiene las pruebas, das click y te abre el navegador, corre tus pruebas y muestra lo que el usuario vería
 - `npx cypress run`, ejecuta cypress en el modo cli, es decir desde la terminal, lo ejecuta en electron en modo headless(muchos navegadores tienen un modo headless(no tienen interface, solamente funcionan desde una terminal)), dice que pudo encontrar 5 pruebas, y las comienza a ejecutar cada una
- Dice de video, en la carpeta de cypress hay videos que simulan ej la creación de una cita desde la interface web
- Para aumentar la calidad de los videos en `cypress.json`
- 0-51, 0 la mejor calidad que E
- ```
{
 • "viewportHeight": 1500,
 • "viewportWidth": 1200,
 • "baseUrl": "http://127.0.0.1:5500/52-Testing-Cypress",
 • "videoCompression": 0
 •
→ }
```
- Al volver a correr elimina los videos y vuelve a crearlos
- Queda evidencia de que la app funciona correctamente
- En `eliminar_cita.spec.js`
  - `cy.screenshot();` //toma un screenshot una vez que llega a esta parte
- Se encuentran en carpetas `> cypress> screenshots`

- Cypress funciona en proyectos para java, para c#
- Las pruebas se escriben en JavaScript, pero puedes probar un frontend en cualquier lenguaje de programación,
- Para software de mejor calidad

# **Sección 53: WEBPACK, ESLINT y BABEL - Creando un Frontend Profecional**

# 339. Introducción a Webpack, ESLint y Babel

- Como crear un front end profesional con webpack, Eslint y Babel
- **ESlint**
  - Con eslint podemos seguir buenas practicas a la hora de escribir código(solucionar errores en el código)
  - Permite solucionar problemas de código mientras lo vas escribiendo, ej si escribes const pero no lo utilizas, eslint te marca error
  - Permite instalar una guía de estilos (forma de escribir código en la que muchas personas se ponen de acuerdo, en seguir esa forma de escribir código) como Airbnb
- **Babel**
  - Un arrow function jamas va a funcionar en internet explorer, tampoco las classes
  - Permite comenzar a escribir codigo de la ultima generacion en navegadores antiguos
  - Permite añadir funciones que aun no son parte de JavaScript, pero si las puedes utilizar ( ESNext) y las lleva a versiones anteriores para que puedas utilizarlas
- **Webpack**
  - Compila assets o archivos estáticos en un bundle(caja grande que contiene todos los archivos que se requieren en el proyecto, puedes agregar archivos de saas o archivos de css o también crear un bundle grande con todos los archivos que conforman un proyecto
  - Ej el administrador de citas tiene diferentes archivos que estamos utilizando con exports e imports, el bundle genera un solo archivo para todos ellos optimizado por parte de webpack
  - Va a ser un bundle grande, o sea un archivo bastante grande, pero gracias a webpack queda optimizado para que puedas utilizarlo en producción y no te preocupes tanto por el performance
- Webpack se integra bastante bien con eslint y babel
- Webpack te permite crear un servidor de desarrollo y te permite habilitar Hot Reload para los cambios(algo que agrego react que estas haciendo un proyecto, guardas cambios y tienes que recargar ) webpack lo lleva a otro nivel

# 340. Transpilando el Código con Babel

- npm init para crear package.json, donde agregaremos algunas dependencias
- Estaremos instalando como dependencias, babel, eslint, webpack y cada uno de ellos también tiene otras dependencias
- INSTALANDO DEPENDENCIAS
- En terminal:
  - npm i --save-dev @babel/cli @babel/core @babel/preset-env
- (dependencias de desarrollo: una vez que yo lleve este proyecto a producción no las necesito)
- Cli=comand line interface
- Presets: tu puedes compilar una version en especifico ej ES2020 lo puedes compilar utilizando un preset, pero ya todo esta en un solo preset llamado env preset (pasar de un código a otro equivalente en otro navegador viejito)

- Creamos archivo entrada.js en carpeta

```
→ class Cliente{
• constructor(nombre, apellido){
• this.nombre=nombre;
• this.apellido=apellido;
• }
•
• }
• const hola= () => {
• console.log('Hola mundo en babel');
→ }
```

- Para transpilar código de entrada.js, hacia la salida, es decir hacia otro archivo

- npx(sin instalar un paquete global, solo localmente)
- Babel(porque en node.modules>bin >babel (ejecutables))

- En terminal:

```
→ npx babel entrada.js --out-file salida.js --presets =@babel/preset-env // crea un archivo nuevo llamado salida.js donde aparece el código transpilado
```

- Para no escribir todo en la terminal en package.json

```
→ "scripts": {
• "babel": "babel entrada.js --out-file salida.js --presets=@babel/preset-env"
→ },
• Y ahora solo pongo en terminal:
→ npm run babel
```



En babel pagina: <https://babeljs.io/>

Settings panel from Babel Playground:

- SETTINGS**
  - ☐ Evaluate
  - ☒ Line Wrap
  - ☐ Prettify
  - ☐ File Size
  - ☐ Time Travel
- Source Type
  - Module
- TARGETS**
  - ie 6
- PRESETS**
  - ☐ react
  - ☐ flow
  - ☐ typescript
  - ☐ stage-3
  - ☐ stage-2
  - ☐ stage-1
  - ☐ stage-0

Environment Preset panel from Babel Playground:

**ENV PRESET**

- ☒ Enabled
- ELECTRON** 1.8 ☐
- NODE** 10.13 ☐
- BUILT-INS** core-js 3.6 Usage ☐
- SPEC** ☐
- LOOSE** ☐
- BUG FIXES** ☒
- SHIPPED PROPOSALS** ☐
- FORCE ALL TRANSFORMS** ☐
- PLUGINS**

Docs Setup Try it out Videos Blog Search Donate Team GitHub

```
1 class Cliente{
2 constructor(nombre, apellido){
3 this.nombre=nombre;
4 this.apellido=apellido;
5 }
6
7 }
8 const hola= () => {
9 console.log('Hola mundo en babel');
10 }
11
```

```
1 "use strict";
2
3 function _classCallCheck(instance, Constructor) { if (!(instance
instanceof Constructor)) { throw new TypeError("Cannot call a
class as a function"); } }
4
5 var Cliente = function Cliente(nombre, apellido) {
6 _classCallCheck(this, Cliente);
7
8 this.nombre = nombre;
9 this.apellido = apellido;
10 };
11
12 var hola = function hola() {
13 console.log('Hola mundo en babel');
14 };
```

Se transpila el código hacia un código que funciona correctamente en internet explorer 6 por ejemplo

# 341. Agregando Webpack

- Instalar webpack
- Lo que hace webpack es tomar como entrada diferentes archivos y los compila dentro de archivos estáticos
- Puedes tener muchos archivos de JavaScript y al final te crea uno, muchos archivos de CSS y al final te crea un archivo de CSS, etc.
- Forma moderna de integrar diferentes tecnologías y es una alternativa a gulp
- Lo que hace poderoso a webpack son los loaders (permite añadir soporte a ciertas tecnologías, desde aquí puedes agregar soporte a archivos de CSS, o también a Babel, etc.)
- VAMOS A INSTALAR WEBPACK, Y DESDE AHÍ ESTAREMOS INSTALANDO BABEL
- Instalaremos 2 paquetes, webpack y webpack cli (para tener acceso a los comandos de webpack)
- En terminal:
  - `npm i --save-dev webpack webpack-cli`
  - En `package.json`, en dependencia vamos a tener a webpack y a webpack cli
  - Para poder darle una configuración a webpack creamos un archivo que se llama `webpack.config.js` (archivo que webpack entiende bien)
  - Webpack utiliza la sintaxis de `command.js`
- En `webpack.config.js`:
  - `const path = require('path');` // importamos la librería de path, con la cual vamos a saber en qué carpeta nos encontramos
  - `module.exports = {`
  - `entry: './js/app.js',` // archivo principal de este proyecto
  - `output: {`
  - `path: path.resolve(__dirname, 'public/js'),` // para no escribir toda la ubicación de la carpeta de origen y donde desee colocar el proyecto, (// librería de node)
  - `// -----Para que tomo info de ubicación`
  - `// -----nombre de carpeta`
  - `filename: 'bundle.js'` // como se va a llamar archivo
  - `}`
  - `}`
  - `pwd`: para saber en qué carpeta estoy // en terminal

- En package.json:
- `"scripts": {`
- `"babel": "babel entrada.js --out-file salida.js --presets=@babel/preset-env",`
- `"webpack": "webpack" //ejecuta webpack en bin, toma la configuración de webpack.config.js y va a compilarlo`
- `},`
- Ahora corremos webpack en terminal:
- `npm run webpack // compila archivos y nos crea el bundle`
- Aparece una advertencia, E 2 modos uno desarrollo y el otro producción , el de default es producción , para decirle que queremos utilizar webpack en modo desarrollo en package.json:
- `"scripts": {`
- `"babel": "babel entrada.js --out-file salida.js --presets=@babel/preset-env",`
- `"webpack": "webpack --mode=development"`
- `},`
- Volvemos a ejecutar en terminal:
- `npm run webpack`
- Otra forma es eliminar lo que esta en negritas y en webpack.config.js poner:
- `const path= require('path'); // importamos la libreria de path, con la cual vamos a saber en que carpeta nos encontramos`
- 
- `module.exports={`
- `entry: './js/app.js', // archivo principal de este proyecto`
- `output: {`
- `path: path.resolve(__dirname, 'public/js'), // no escribir toda la ubicación de la carpeta de origen y donde desee colocar el proyecto, libreria de node`
- `// -----Para que tomo info de ubicacion`
- `// -----nombre de carpeta`
- `filename: 'bundle.js' // como se va a llamar archivo`
- `},`
- `mode: 'development' // también se puede crear una variable de entorno en node y leer en local development, o si esta en un servidor que sea produccion`
- `}`

- Ahora en index.html ya podemos cambiar el app.js por el bundle que es una versión compilada de todo lo que tenemos con un mejor soporte gracias a webpack
- `<script src="public/js/bundle.js" type="module"></script>`
- Si abrimos en live server, sigue funcionando pero ahora tenemos mayor soporte y algunas mejoras de performance
- Si quieres darle soporte a internet explorer 6 u 8 etc. Tienes que habilitar un loader, el de babel

# 342. Agregando Babel por medio de Webpack

<https://webpack.js.org>

- Como agregar babel al proyecto en webpack
- Forma en que agregas tecnologías externas a webpack es por medio de algo llamado loader( es similar a un plugin, realiza cierta funcionalidad especifica E muchos loaders)
- En pagina> loaders>babel-loaders> toda la documentación

• Instalando loader:  
→ `npm i --save-dev babel-loader` // es como instalar el plugin(funcionalidad extra, permite agregar cierto soporte) de babel para webpack, se les conoce como loaders

- En node modules> baidings> hay muchos archivos js que no queremos que sean parte del bundle, por lo tanto excluimos esos

• En `webpack.config.js`:

→ `const path= require('path');` // importamos la libreria de path, con la cual vamos a saber en que carpeta nos encontramos

```
module.exports={
 entry: './js/app.js', // archivo principal de este proyecto
 output: {
 path: path.resolve(__dirname, 'public/js'), // no escribir toda la ubicacion de la carpeta de origen y donde desee colocar el proyecto, libreria de node
 // -----Para que tomo info de ubicacion
 // -----nombre de carpeta
 filename: 'bundle.js' // como se va a llamar archivo
 },
 mode: 'development', // también se puede crear una variable de entorno en node y leer en local development, o si esta en un servidor que sea produccion
 module: {
 rules: [// arreglo porque podemos tener diferentes loaders y cada uno se configura en un objeto
 {
 test: /\.js$/, // test permite identificar diferentes archivos(/\.js$/ busca todos los archivos js en carpeta principal)
 exclude: /(node_modules)/,
 use: {
 loader: 'babel-loader', // loader que bamos a utilizar
 options: {
 presets: ['@babel/preset-env'] // compila utilizando el preset que hemos instalado previamente
 }
 }
 }
]
 }
}
```

- Eliminamos entrada.js y salida.js porque no los requerimos en el bundle

• Ahora en terminal:

→ `npm run webpack`

- Nos crea nuestro bundle y nos va a funcionar en un navegador anterior

# 343. Creando un Servidor con webpack dev server

- Webpack te permite crear un servidor e ir recargando conforme vayamos haciendo cambios
- Webpack dev server es una opción mas robusta que live server porque funciona en cualquier editor y live server solo funciona en visual studio code
- En terminal para crear servidor webpack:
  - `npm i --save-dev webpack-dev-server`
  - Se configura en `webpack.config.js`
  - `devServer: { // pasarle la configuración`
    - `contentBase: path.join(__dirname, '/'), // en que carpeta va a encontrar el contenido, en este caso`  
`// carpeta principal, porque ahí esta index`
    - `compress: true,`
    - `port: 9000 // puesto en el que se va a ejecutar`
  - `},`
  - Despues en `package.json` en `scripts`
  - `"dev": "webpack-dev-server" // esta en bin pero no funciona y funciona con:`
  - `"dev": "webpack serve"`
- Ahora en terminal
  - `npm run dev // crea servidor de webpack`
- Donde corre el proyecto: <http://localhost:9000/>

- Posible solución a error



Jorge — Profesor asistente

hace 5 meses

0 ↑ ⋮

Asegurate de tener estas versiones en tu package.json:

```
1 | "webpack": "^4.44.2",
2 | "webpack-cli": "^3.3.12",
3 | "webpack-dev-server": "^3.11.0"
```



Julio Genaro Esquivel

hace 5 meses

0 ↑ ⋮

tengo estas:

```
1 | "webpack": "^5.1.0",
2 | "webpack-cli": "^4.0.0",
3 | "webpack-dev-server": "^3.11.0"
4 |
5 | como instalo una version en especifico??
```



Jorge — Profesor asistente ★Respuesta

hace 5 meses

1 ↑ ⋮

Ah simplemente borra `node_modules` cambia los numeros de las versiones, escribes `npm i` en tu terminal y listo



- Otra solucion:



Ernesto Israel

hace 5 meses

33 ↑ ⋮

Para trabajar con las versiones actualizadas de webpack 5.4.0, webpack-cli 4.2.0, y webpack-dev-server 3.11.0, sin que les aparezca el mensaje de error mencionado. En el archivo package.json dentro del arreglo de "scripts" agreguen la siguiente línea

```
"dev": "webpack serve"
```

Posteriormente ejecuten: `npm run dev`

- Si abrimos app.js y mandamos imprimir en consola y guardamos cambios, en terminal dice que se compilo correctamente, pero si abro la consola, no aparece nada, si recargo tampoco, tendría que abrir otra terminal y poner:
  - `npm run webpack//` para que lo vuelva a compilar, entonces si recargamos en pagina ahora si dice en consola
  - Mucho trabajo porque tendríamos que tener 2 terminales, una para el servidor, otra para compilar cada vez que hago cambios

# 344. Agregando Live Reload

- Por lo tanto hacer esto mas rápido con un watch en scripts:
  - `"watch": "webpack --w" // no funciona por lo tanto`
  - `"watch": "webpack -w"`
- y ejecuto en terminal
  - `npm run watch` // de esa forma cada vez que yo haga cambios lo compila solo, y solo tengo que recargar en la pagina
- Para que la pagina se recargue automáticamente
- De los scripts solo puedes tener uno a la vez, es decir tendrías que tener multiples terminales
- Instalamos una dependencia
- Detenemos el servidor y en terminal:
  - `npm i --save-dev concurrently` // para poder correr 2 o mas scripts al mismo tiempo
- Luego en package.json cambiamos en scripts el dev:
  - `"dev": "concurrently --kill-others \"npm run watch\" \"webpack serve\""` // comillas dobles para escapar // no sirve con instalar el concurrently basta segun yo y dejarlo como estaba
- Luego en webpack.config.js:
  - `devServer: { // pasarle la configuracion`
  - `contentBase: path.join(__dirname, '/'), // en que carpeta va a encontrar el contenido, en este caso carpeta principal, porque ahi esta index`
  - `compress: true,`
  - `port: 9000, // puesto en el que se va a ejecutar`
  - `publicPath: "/public/js/", // lo que vamos a estar compilando ubicacion`
  - `watchContentBase: true`
  - `},`
  - Ahora en terminal ponemos:
  - `npm run dev` // porque nuestro codigo tiene concurrently
- Esto es mas rápido para no estar recargando todo el tiempo, ya conectado al watch de webpack y al servidor de webpack dev server

# 345. Añadiendo ESLint a nuestro código

- Como agregar Eslint a este proyecto
- Instalar Eslint para encontrar y solucionar errores en el código de JavaScript
- Instalar la extensión de Eslint
- Cuadritos barra izquierda visual > eslint > allow global
- Agregamos eslint a nuestro proyecto
- En terminal:
  - `npm i --save-dev eslint`
  - `npm install eslint --save-dev // yo use esta otra forma`
- Creamos un archivo de eslint, en terminal:
  - `npx eslint --init`
- Nos pregunta para que vamos a utilizar eslint
- Revisar sintaxis, encontrar problemas y añadir una guía de código de estilo // tercera opción
- Elegimos la segunda opción : Revisar sintaxis, encontrar problemas
- Elegimos que soporta los módulos de JavaScript
- Le damos que ninguno, ni react ni vuejs
- No utiliza TypeScript
- Este proyecto corre en el navegador
- En que formato queremos crear el archivo de configuración de eslint > javascript
- Y todo eso nos crea `eslint.config.js` que sigue la sintaxis de JavaScript y tiene la configuración

- Si abres el webpack.config, marca algunos errores, tendríamos que decirle que ignore este archivo porque corre en node:
- Ctrl+shif+p>json> open settings> tiene lo que tienes instalado en tu editor, y ponemos:  
→`"eslint.options":{`
- `"ignorePattern":"webpack.config.js"`
- `}//` para ya no tener error en archivo , puedes deshabilitar algunos archivos y mantener otros
- Creamos archivo nuevo
- prueba.js // te marca los errores y si guardas cambios y en terminal:  
→`npx eslint prueba.js//` nos dice los errores en la terminal

# 346. Agregando la Guía de Estilos de Airbnb

- Como agregar Airbnb como una guía de estilo para el código, que es y como utilizarlo
- Le dice a muchas personas como se recomienda escribir el código de forma que todo el mundo lo pueda entender
- <https://github.com/airbnb/javascript//> buenas practicas para escribir código
- Le da recomendaciones a las personas para que puedan seguir buenas practicas a la hora de escribir codigo
- E diferentes guias de estilo, la mas comun es Airbnb, google tambien tiene la suya, no es obligatorio que la implementes en todos los proyectos, pero es una buena forma de tener un orden adecuado en tu codigo
- Agregando guia de estilos de Airbnb
- En terminal:
  - `npx eslint --init` seleccionamos, checar sintaxis, encontrar problemas y forzar una guia de codigo > modules > none of these > no > browser > usar una guia popular > airbnb > javascript > yes
- Gracias a webpack ya no requieres la extensión js
- Guia de estilos, dar un orden mas claro al codigo
- En app.js ya con errores corregidos:
  - `import App from './classes/App';`
  - `// eslint-disable-nextline`
  - `App();`
- Vamos corrigiendo los errores que nos marca
- Guía de estilos de Airbnb dice que tienen que ser 2 espacios
- Para deshabilitar para una parte, raton sobre error quick fix > disable for this line
- Linter Airbnb

# 347. Solucionar errores con ESLint y Airbnb StyleGuide

- Soluciones al abrir el archivo, como Agregar el loader de eslint y que lo haga webpack
- Como corregir errores en eslint automáticamente con webpack
- Para dejar de ver tanto errores desactivamos la extensión eslint
- Requerimos un loader de eslint en terminal:  
→ `npm i --save-dev eslint-loader`
- En `webpack.config`:

```

→ module:{
• rules: [// arreglo porque podemos tener diferentes loaders y cada uno se configura en un objeto
• {
• test:/\.js$/, // test permite identificar diferentes archivos(/\.js$/ busca todos los archivos js en carpeta principal)
• exclude:/(node_modules)/,
• use:{
• loader:'babel-loader',// loader que vamos a utilizar
• options:{
• presets:['@babel/preset-env']// compila utilizando el preset que hemos instalado previamente
• }
• }
• },
• //Agregando segundo loader
• {
• // antes de que haga algo, ejecute esta parte E pos que es despues de
• enforce:'pre',
• test:/\.js$/,
• exclude:/(node_modules)/,
• loader:'eslint-loader' // hay que instalar este loader
• }
•]
→ }

```

- En terminal:

```
→ npm i --save-dev eslint-loader
```

- Ahora en terminal ejecutamos el comando de dev

```
→ npm run dev
```

- Salen los errores que podemos ir corrigiendo manualmente o automaticamente poniendo lo siguiente en webpack.config.js



```

→ {
 • enforce:'pre', // antes de que haga algo, ejecute esta parte E pos que es después de
 • test: /\.js$/,
 • exclude: /(node_modules)/,
 • loader: 'eslint-loader', // hay que instalar este loader
 • options: {
 • fix: true // arregla los errores
 • }
→ }

```

- Detenemos la ejecución en terminal y volvemos a correr

→ npm run dev // soluciona muchos problemas

Y hay que ir solucionando los que faltan hasta que el bundle compile correctamente, nos apoyamos habilitando la extensión de eslint, en otros casos utilizamos la terminal para verlo de una mejor forma

En terminal:

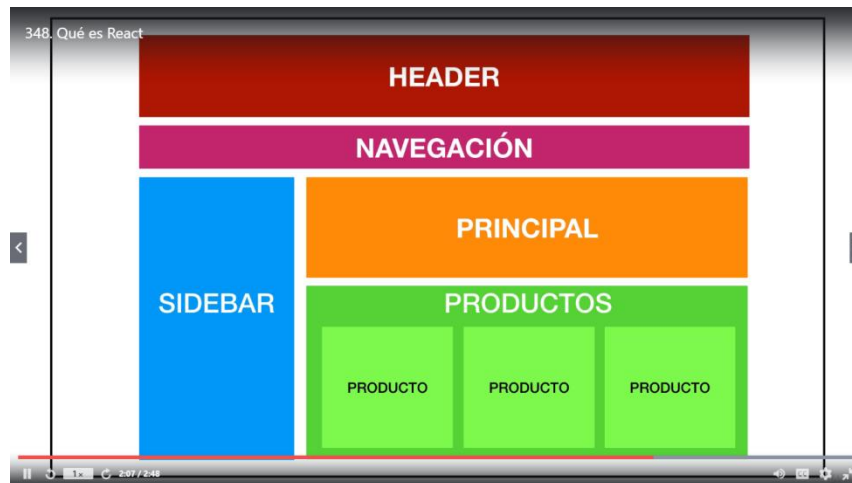
→ npm run dev // ya todo funciona correctamente

- Código más ordenado, más fácil de leer para más personas
- Buena práctica implementarlo
- Se recomienda cuando son muchas personas en un proyecto

# **Sección 54: REACT: Aprende React Creando un Proyecto**

# 348. Qué es React

- React o ReactJS: es una librería de JavaScript para crear interfaces de usuario para aplicaciones web, es decir únicamente lo que es visible en un sitio web es lo que es manejado por parte de react
- Es desarrollada por facebook(herramienta bien soportada y que no va a desaparecer de un día para otro, popularidad bastante alta, mucho mas que otras librerías similares o frameworks como angular o vuejs)
- Corre en el cliente, (no necesita una respuesta de un servidor)
- Te permite ordenar tu proyecto mejor por medio de componentes(piezas de código, en este caso funciones que son reutilizables, pero react tiene su propia forma de crearlas)
- Una de las grandes ventajas de react es que hay una gran cantidad de herramientas y librerías así como soporte bastante bueno, si tienes algún problema en un proyecto y te aparece un mensaje de error, puedes buscar en google y encontraras la respuesta
- React esta a favor de la separación, es decir tendríamos un componente para:
- E algunos componentes que tienen dentro otros componentes



- ¿Quién utiliza react
  - Udemy
  - Paginas web como Airbnb
  - Facebook
  - Twitter
  - Microsoft
  - Spotify
  - Asana
  - Dropbox
- 
- Tenemos que tener instalado node

# 349. Lo que vamos a construir

- Todo lo que requerimos y crear el primer proyecto en react
- Todo lo que aprendiste en este curso ya practico en una librería bastante popular (react) que es la que tiene mas crecimiento hoy en día sobre angular y vuejs
- Cotizador de prestamos con ultima version de react
- Todo lo que tienes que saber en lo basico de react
- Mientras mas dinero pidas el interes es un poco mas bajo, pero mientras mas tardes en pagar el interes es mas alto
- Spinner
- resultado

# 350. Herramientas para trabajar con React

- Tener instalado node y npm
- Comprobar su instalacion en cmd:  
→ node -v  
→ npm -v // si aparecen las versiones si esta instalado
- Utilizamos herramienta create react app, de esta forma no nos preocupamos por las dependencias de react, no nos preocupamos por el archivo de webpack, ya viene todo listo para comenzar un proyecto de react en 5 min o menos
- <https://github.com/facebook/create-react-app>
- Por lo tanto en cmd:  
→ npx create-react-app prestamos  
-----nombre del proyecto
- // no aparece el service worker con el comando de arriba, por lo tanto:  
→ npx create-react-app prestamos --template cra-template-pwa
- Instalamos una extensión que es muy util cuando trabajas con react
- En extensiones de chrome> react developer tools> add to chrome
- En terminal:  
→ Cd prestamos/  
→ npm start
- Abrimos la carpeta que se creo de prestamos
- Crea un servidor y lo abrimos en el navegador
- Local: <http://localhost:3000> // para navegador
- On Your Network: <http://192.168.64.1:3000> // ip para ver tu proyecto en telefono en misma red wifi, o en otro dispositivo una compu ejemplo
- Ahora instalamos un par de dependencias
- En cuadritos de la izquierda de visual > Es7 react/redux/graphql7/react-native snippets
- Y simple react snippets
- En react E 2 carpetas public y src, usualmente se escribe el codigo en src, aqui escribes tus componentes, despues react se encarga de crear una version compilada bundle y es lo que se muestra en public

- Usualmente public solo lo modifica para abrir el index y cargar algunas hojas de estilo que se van a utilizar de forma global en todo el proyecto, vamos a agregar 2 , porque vamos a agregar un framework css llamado skeleton
- <https://cdnjs.com/> // lo busco en esta pagina> copiamos con el link tag> y lo pegamos en cualquier lugar en el index, en este caso lo pegamos antes de la etiqueta title, eso agrega skeleton a todo el proyecto completo
- Skeleton requiere normalize(reset), hara que los elementos se vean un poco mas consistentes a lo largo de diferentes navegadores, y es una dependencia de skeleton, otros frameworks como bustrap no requieren skeleton porque tienen su propio reset, pero skeleton es mucho mas ligero que bustrap
- Buscamos normalize en la pag> copiamos el link tag, y pegamos antes de skeleton
- En react puedes utilizar cualquier framework de frontend( skeleton, bustrap, materialicss, tailwind
- En Create react app una vez que guardamos cambios la pagina se actualiza sola
- AHORA VEREMOS LA PARTE DE SRC
- Index importa el archivo app, se encarga de registrar el serviceWorker, es decir que tu app pueda funcionar incluso sin internet en algunos casos, o darle al usuario un mensaje un poco mejor , también importa una hoja de estilo
- App es el archivo principal que contiene o va a contener todos los demás componentes, importa un logo, import aun css,
- Eliminamos de las carpetas app.css/ app.js/ logo / app.test / / porque no los vamos a ocupar

- Eliminamos una parte del archivo de App.js y queda así:

→ *function* App() {

- return (
  - <div className="App">
  - <h1>Hola Mundo</h1>
  - </div>
- );

→ }

- export default App;
-



# 351. Comenzando nuestro primer proyecto

- <https://gist.github.com/juanpablogdl/194c15d376391a67c46dd08817bf9ae4> copiar en index.css lo que viene en este enlace (cambia apariencia a nuestro proyecto)
- Creamos nuestro primer componente, los componentes de react su tamaño varia E componentes sencillos de pocas líneas y E componentes con cientos de líneas
- Creamos carpeta componentes en src y creamos dentro un archivo
- Los componentes deben iniciar con mayúscula, sin espacios usualmente
- Header.js
- Si utilizo simple react snippets  
→ `import // importa react`
- Un componente siempre es una función

- En Header.js:

```
→ // Este es mi primer componente, ahora tengo que mandarlo llamar con un import en App.js que es el principal para los demás componentes de la app
```

- 
- `import React from 'react';`
- 
- `function Header(){`
- `return(// de react (lo que se va a mostrar en pantalla es lo que se coloque dentro del return)`
- `<h1>Hola Mundo</h1>`
- 
- `)`
- `}`

```
→ export default Header;
```

En App.js:

```
→ import React from 'react';
• import Header from './componentes/Header';
•
function App() {
• return (
• <div className="App">
• <Header />//para utilizar Header
• // -----nombre del componente
• </div>
•);
• }
→ export default App;
```

- Ventajas de componente:
- Son reutilizables , puedo pegarlo múltiples veces y se manda llamar esa cantidad de veces( en este caso se imprime en pantalla 5 veces por ejemplo)

- Ej si tienes un listado de productos, creas un componente que muestra imagen, titulo y precio, lo puedes llamar múltiples veces con un `forEach` o con un `map`, en react, `map` es el recomendado cuando iteras y quieres mostrar múltiples veces un mismo componente pero con diferente info
- Cuando tienes un `return`, tienes que retornar un elemento que contenga los demás elementos, si no marca error
- Si no quieres crear código extra en react importamos un `Fragment` que es como crear un `div` en el `return` para retornar todos los elementos, pero no se va a mostrar en el frontend, de esa forma no vamos a crear contenido o html extra
- En `App.js`:
  - `import React, {Fragment} from 'react';`
  - `import Header from './componentes/Header';`
  - 
  - `function App() {`
  - `return (`
  - `<Fragment>`
  - `<Header />`
  - 
  - `</Fragment>`
  - `);`
  - `}`
  - `export default App;`

# 352. Pasar datos entre components

- Otra ventaja que tienen los componentes es que le puedes pasar info desde el componente principal hacia el otro, eso se hace con props
- En react los datos fluyen del componente principal hacia los componentes hijos

• En App.js:

→ import Header from './componentes/Header';

```
•
function App() {
• return (
• <Fragment>
• <Header
• titulo="Cotizador de prestamos"
• //-----propiedad
• // -----puede ser funcion, numeros (cualquier tipo de dato de JavaScript)
• // como voy a leer este dentro del Header
• />
•
• </Fragment>
•);
• }
→ export default App;
•
```

- En Header.js:
- // Este es mi primer componente, ahora tengo que mandarlo llamar con un import en App.js que es el principal para los demás componentes de la app
- 
- import React from 'react';
- 
- *function* Header(**props**){
- // -----para leer lo del otro archivo aqui
- // aqui puedes colocar codigo estandar de JavaScript
- **console.log(props);**
- 
- return(// de react (lo que se va a mostrar en pantalla es lo que se coloque dentro del return)
- 
- <h1>Hola Mundo</h1>
- 
- )
- }
- export default Header;
- 
- Con esto mandamos imprimir a la consola , pero para eso instalamos react developer tools
- Para ver los props en consola> components> header> props(ver todos los props que se le han pasado a un componente en especifico)
- Te permite ver el arbol de componentes de tu app
- 
- Ahora para mostrar los props
- React utiliza una sintaxis jsx que te permite mezclar html con JavaScript
- Como le hago para acceder a la info de javascript dentro de esa parte del html con jsx:
- Lo que esta entre {} interpretalo como codigo de JavaScript, eso ya imprime en el DOM el prop
- Si le pones 1+1 dentro de {} realiza la operación y si le quitas las llaves lo interpreta como html

→ En Header.js:

// Este es mi primer componente, ahora tengo que mandarlo llamar con un import en App.js que es el principal para los demas componentes de la app

•

• import React, { Fragment } from 'react';

•

*function* Header(*props*){

• // -----para leer lo del otro archivo aquí

• // aqui puedes colocar codigo estandar de JavaScript

• console.log(props);

•

return(// de react (lo que se va a mostrar en pantalla es lo que se coloque dentro del return)

• **<Fragment>**

• **<h1>{props.titulo}</h1>**

• **<p>{props.descripcion}</p>**

• // -----nombre del prop

• **</Fragment>**

• )

• }

→ export default Header;

- En App.js
- import React, {Fragment} from 'react';
- import Header from './componentes/Header';
- 
- *function* App() {
- return (
- **<Fragment>**
- **<Header**
- **titulo="Cotizador de prestamos"**
- //-----propiedad
- //         -----puede ser funcion, numeros (cualquier tipo de dato de JavaScript)
- // como voy a leer este dentro del Header
- **descripcion="Utiliza el formulario y obten una cotizacion"**
- **/>**
- 
- **</Fragment>**
- );
- }
- export default App;

- En lugar de props puedes aplicar destructuring para que cree la variable, y extraiga su valor
- // Este es mi primer componente, ahora tengo que mandarlo llamar con un import en App.js que es el principal para los demás componentes de la app
- 
- import React, { Fragment } from 'react';
- 
- *function* Header(**{{titulo}}**){
- //                   -----para leer lo del otro archivo aquí
- 
- 
- return(// de react (lo que se va a mostrar en pantalla es lo que se coloque dentro del return)
- <Fragment>
- <h1>**{titulo}**</h1>
- 
- 
- </Fragment>
- )
- }
- export default Header;
-



## 353. Segunda forma de escribir componentes funcionales

- Otra forma en que puedes crear componentes y ventajas de su sintaxis
- Los componentes son funciones, y lo puedes crear utilizando function declaration o function expresion
- Utilizando snippets  
→ `sfc` // crea el cuerpo

- Características: le puedes dar por implícito el return
- Código mas simplificado, quitamos Fragment porque no lo estamos requiriendo
- Si das por implícito el return no puedes usar la parte de colocar código estándar

```
1 import React from 'react'
2
3 const Header = ({titulo}) => (
4 <h1>{ titulo }</h1>
5);
6
7 export default Header;
```

# 354. Creando un Formulario

- Agregar el formulario con los diferentes campos que se requieren para realizar una cotización
- Crear un segundo componente donde colocaremos el formulario, para que los usuarios puedan elegir la cantidad que desean pedir prestada y el plazo para pagar
- Creamos en componentes archivo Formulario.js:

→ `import React from 'react';`

```
const Formulario = () => {
 return (
 <h1>Formulario Aqui</h1>
);
}
```

→ `export default Formulario;`

- En App.js:

→ `import React, {Fragment} from 'react';`  
`import Header from './componentes/Header';`  
`import Formulario from './componentes/Formulario';`

```
function App() {
 return (
 <Fragment>
 <Header
 titulo="Cotizador de prestamos"
 //-----propiedad
 // -----puede ser funcion, numeros (cualquier tipo de dato de JavaScript)
 // como voy a leer este dentro del Header
 />

 <div class="container"> // mostrar el formulario (container es clase de skeleton)
 <Formulario/>
 </div>
 </Fragment>
);
}
```

→ `export default App;`

- En consola nos marca un error, dice que no es valido un class si no className porque es regla de jsX
- En la parte de formulario aquí ponemos el gist que nos proporciono en pagina y ya tenemos el formulario con boton para calcular, ya tenemos la interface

# 355. Leer valores de un formulario

- Leer lo que el usuario escribe y escribir una función de JavaScript para que realice el calculo
- Como acceder a lo que el usuario escribe en los diferentes inputs
- State: lo que hace que las app de react sean muy rapidas
- Como implementar state en nuestra app
- En formulario lo importo y lo defino
- Usualmente cada pieza de tu app va a tener un state, todo lo que sea interactivo, en este caso en el formulario, la cantidad y el plazo
- Definimos uno para la cantidad
- useState una vez que lo utilizas te retorna 2 valores
- Como le hago para que cuando haga cambios en formulario valor impreso del state cambie también, para eso es la funcion guardarCantidad, y lo vamos a hacer por medio de un evento
- React utiliza eventos y en lugar de que en JavaScript le pongas un id o una clase y escribas un query selector y un addEventListener, react cuenta una gran cantidad de eventos, usualmente los eventos tienen un on al inicio, y después el nombre del evento

```

→ import React, {useState} from 'react';
•
•
const Formulario = () => {
•
 //Definir el state
• const [cantidad, guardarCantidad] = useState(0); // en useState le puedes dar un valor inicial ej 0, arranca en cero y cuando utilicemos funcion de guardar cantidad, es que se va a ir ree
scribiendo
• // -----
usando array destructuring, primero una variable que contiene el valor del state y el otro es una funcion utilizada para interactuar y guardar el state que estamos creando, para guardar o
modificar el state
• const leerCantidad=(e) => {
• guardarCantidad(parseInt(e.target.value)); //cada vez que hacemos cambios cambia el numero impreso arriba
• }
• return (
• <form>
• {cantidad}
• <div className="row">
• <div>
• <label>Cantidad Prestamo</label>
• <input
• className="u-full-width"
• type="number"
• placeholder="Ejemplo: 3000"
• onChange={leerCantidad} // evento que se va a ejecutar cada vez que el usuario comience a escribir algo o la cambie en primer dese
• // ----- funcion que se va a ejecutar cada vez que cambie
• />
• </div>
• <div>
• <label>Plazo para Pagar</label>
• <select
• className="u-full-width"
• >
• <option value="">Seleccionar</option>
• <option value="3">3 meses</option>
• <option value="6">6 meses</option>
• <option value="12">12 meses</option>
• <option value="24">24 meses</option>
• </select>
• </div>
• <div>
• <input
• type="submit"
• value="Calcular"
• className="button-primary u-full-width"
• />
• </div>
• </div>
• </form>
•);
• }
•
→ export default Formulario;

```

- Cuando el state va cambiando, se va actualizando la interface por eso react es tan rapido
- En consola> componentes> formulario> tenemos props y state, de esta forma puedes ver los valores

```

import React, {useState} from 'react';

const Formulario = () => {

 //Definir el state
 const [cantidad, guardarCantidad] = useState(0); // en useState le puedes dar un valor inicial ej 0, arranca en cero y cuando utilicemos funcion de guardar cantidad, es que se va a ir reescribiendo
 // -----
 usando array destructuring, primero una variable que contiene el valor del state y el otro es una funcion utilizada para interactuar y guardar el state que estamos creando, para guardar o modificar el state

 return (
 <form>
 {cantidad} //valor del state
 <div className="row">
 <div>
 <label>Cantidad Prestamo</label>
 <input
 className="u-full-width"
 type="number"
 placeholder="Ejemplo: 3000"
 onChange={e => guardarCantidad(parseInt(e.target.value))} // para no definir la función arriba y escribir mas código
 // ---- funcion que se va a ejecutar cada vez que cambie
 />
 </div>
 <div>
 <label>Plazo para Pagar</label>
 <select
 className="u-full-width"
 >
 <option value="">Seleccionar</option>
 <option value="3">3 meses</option>
 <option value="6">6 meses</option>
 <option value="12">12 meses</option>
 <option value="24">24 meses</option>
 </select>
 </div>
 <div>
 <input
 type="submit"
 value="Calcular"
 className="button-primary u-full-width"
 />
 </div>
 </div>
 </form>
);
}

export default Formulario;

```



- El state lo quiero pasar por diferentes componentes
- En react los valores fluyen del padre al hijo, pero nunca del hijo al padre, por lo tanto dejamos de definir el estado en el formulario, y lo definimos en la App.js y lo importo también
- Y aplico destructuring en formulario para mandarlo llamar
- Ahora ya lo podemos comunicar desde la app principal con diferentes componentes, ya que si lo tienes en el hijo en este caso formulario, no se puede sacar se ahí

- En formulario:
- `import React from 'react';`
- 
- `const Formulario = ({cantidad, guardarCantidad}) => {`
- 
- `return (`
- `<form>`
- `{cantidad}`
- `<div className="row">`
- `<div>`
- `<label>Cantidad Prestamo</label>`
- `<input`
- `className="u-full-width"`
- `type="number"`
- `placeholder="Ejemplo: 3000"`
- `onChange={e => guardarCantidad(parseInt(e.target.value)) } // para no definir la funcion arriba y escribir mas codigo`
- `// ---- funcion que se va a ejecutar cada vez que cambie`
- `/>`
- `</div>`
- `<div>`
- `<label>Plazo para Pagar</label>`
- `<select`
- `className="u-full-width"`
- `>`
- `<option value="">Seleccionar</option>`
- `<option value="3">3 meses</option>`
- `<option value="6">6 meses</option>`
- `<option value="12">12 meses</option>`
- `<option value="24">24 meses</option>`
- `</select>`
- `</div>`
- `<div>`
- `<input`
- `type="submit"`
- `value="Calcular"`
- `className="button-primary u-full-width"`
- `/>`
- `</div>`
- `</div>`
- `</form>`
- `);`
- `}`
-

- En App.js:
- import React, {Fragment, **useState**} from 'react';
- import Header from './componentes/Header';
- import Formulario from './componentes/Formulario';
- 
- function App() {*
- **// definir el state**
- **const [cantidad, guardarCantidad]=useState(0); //para que fluya del padre al hijo**
- 
- return (*
- *<Fragment>*
- *<Header*
- *titulo="Cotizador de prestamos"*
- *//-----propiedad*
- *// -----puede ser funcion, numeros (cualquier tipo de dato de JavaScript)*
- *// como voy a leer este dentro del Header*
- 
- */>*
- 
- <div className="container">*
- *<Formulario*
- *cantidad={cantidad}*
- *guardarCantidad={guardarCantidad}//prop*
- */>*
- 
- *</div>*
- 
- </Fragment>*
- *);*
- *}*
- export default App;
-

# 356. Leer los datos de un Select

- Segunda pieza de state para el plazo
- En App.js:
- import React, {Fragment, **useState**} from 'react';
- import Header from './componentes/Header';
- import Formulario from './componentes/Formulario';
- 
- *function* App() {
- // definir el state
- **const** [cantidad, guardarCantidad]=**useState**(0); //para que fluya del padre al hijo
- **const** [plazo, guardarPlazo]=**useState**(0);// para el select segundo campo del formulario
- 
- return (
- **<Fragment>**
- **<Header**
- titulo="Cotizador de prestamos"
- //-----propiedad
- // -----puede ser funcion, numeros (cualquier tipo de dato de JavaScript)
- // como voy a leer este dentro del Header
- 
- **>**
- 
- **<div** className="container">
- **<Formulario**
- cantidad={cantidad}
- guardarCantidad={guardarCantidad}//prop
- plazo={plazo}
- guardarPlazo={guardarPlazo} // los nombra igual para no pensar en muchos nombres
- **>**
- 
- **</div>**
- 
- **</Fragment>**
- );
- }
- export default App;
-

- En Formulario.js:

```

→ import React from 'react';
•
• const Formulario = ({cantidad, guardarCantidad, plazo, guardarPlazo}) => {
•
• return (
• <form>
•
• <div className="row">
• <div>
• <label>Cantidad Prestamo</label>
• <input
• className="u-full-width"
• type="number"
• placeholder="Ejemplo: 3000"
• onChange={e => guardarCantidad(parseInt(e.target.value))} // para no definir la funcion arriba y escribir mas codigo
• // ---- funcion que se va a ejecutar cada vez que cambie
• />
• </div>
• <div>
• <label>Plazo para Pagar</label>
• <select
• className="u-full-width"
• onChange={e => guardarPlazo(parseInt(e.target.value))}
• >
• <option value="">Seleccionar</option>
• <option value="3">3 meses</option>
• <option value="6">6 meses</option>
• <option value="12">12 meses</option>
• <option value="24">24 meses</option>
• </select>
• </div>
• <div>
• <input
• type="submit"
• value="Calcular"
• className="button-primary u-full-width"
• />
• </div>
• </div>
• </form>
•);
• }
•
→ export default Formulario;

```

- Se guarda la cantidad conforme la cambia el usuario ( YA que se imprimía la nueva cantidad en pantalla )

# 357. Leer un submit y validar un formulario

- Como leer cuando el usuario halla seleccionado tanto una cantidad, como un plazo y presione el botón (también se hace con un evento)
- Leer cuando el usuario haga submit en este formulario y hacer validación, para ello react nos da eventos
- Cuando quieres revisar un formulario, o hacer algo una vez que el usuario ha presionado submit, haces un `addEventListener` a submit, de la misma forma aquí
- App.js cambiamos a string vacío:  
→ `const [plazo, guardarPlazo]=useState("");` // para el select segundo campo del formulario
- En formulario:
- Importamos `useState`:  
→ `import React, {useState} from 'react';`
- //Definimo el state  
→ `const [error, guardarError] = useState(false);` // como vamos a tener un mensaje de error inicia en false porque no hay error hasta que el usuario haga algo mal, entonces cambia a true
- Definimos la función:  
→ //Cuando el usuario hace submit
- `const calcularPrestamo = e => {`
- `e.preventDefault();`
- `//validar`
- `if(cantidad===0 || plazo===""){`
- `guardarError(true);`
- `}`
- `//Realizar la cotizacion`  
→ `}`
- Después del return, mandamos llamar la función a través de un `addEventListener`:  
→ `return (`  
→  `<form onSubmit={calcularPrestamo}>`

# 358. Mostrar mensaje de error

- Muestre un mensaje de error en caso de que el error sea true
- Cuando cambia el state a true mostrar mensaje de error en pag

- En formulario agregamos esto:

```
→ //validar
• if(cantidad===0 || plazo===""){
• guardarError(true);
• return; // para que se detenga la ejecucion
• }
•
• // puede ser que el usuario tenga un error y despues lo corrija
• //Eliminar el error previo
• guardarError(false);
→ //Realizar la cotizacion
```

- En return ponemos:

```
• return (
→ <Fragment>
```

```
 {(error)? <p className="error">Todos los campos son obligatorios</p> : null } // si error es true entonces..... Caso contrario nada, null
→ </Fragment>
```

```
•
```

# 359. Escribiendo el código de Cotizar la cantidad a pagar

- Realizar la cotización
- Como escribir una función cuando comienzas a tener una gran cantidad de código, lo mejor es escribir un helper
- Si el usuario pide una cantidad mas alta, el porcentaje de intereses es menor, pero si pide una cantidad mas alta de meses, el % de intereses seria un poco mayor, vamos a escribir alrededor de 50 líneas, pero el lugar de llenar componentes con código, por orden se recomienda generar un archivo de helpers, o funciones que puedas importar y utilizar
- En src creo archivo helpers.js (funciones que puedo reutilizar muchas veces y permiten no cargar tanto componentes con mucho código)
- En react si tus componentes son de mas de 100 líneas, lo mas seguro es que puedas cortarlo creando diferentes componentes



- En helpers:

```
→ export function calcularTotal(cantidad, plazo){
 • // Cantidades
 • //0-1000=25% hacia lo que esta pidiendo
 • //1001-5000=20%
 • //5001-10000=15%
 • //10000+=10%
 •
 • let totalCantidad;
 • if(cantidad <= 1000){
 • totalCantidad=cantidad*.25;
 •
 • }else if(cantidad >1000 && cantidad <= 5000){
 • totalCantidad=cantidad*.20;
 • }else if(cantidad >5000 && cantidad <= 10000){
 • totalCantidad=cantidad*.15;
 • }else {
 • totalCantidad=cantidad*.10;
 • }
 •
 • console.log(totalCantidad);
→ }
```

- En formulario importamos los helpers:

```
→ import {calcularTotal} from '../helpers';
```

- Mandamos llamar la función:

```
→ //Realizar la cotización
→ calcularTotal(cantidad, plazo);
```

# 360. Escribiendo el código para cotizar según el plazo

- En helper puedes escribir código estándar de JavaScript sin ningún problema

- En helpers:

→ `// Calcular el plazo`

- `// 3 = 5%`

- `// 6 = 10%`

- `// 12 = 15%`

- `// 24 = 20%`

- 

`let totalPlazo=0;`

- 

`switch(plazo){`

- `case 3:`

`totalPlazo=cantidad*.05;`

- `break;`

- `case 6:`

`totalPlazo=cantidad*.10;`

- `break;`

- `case 12:`

`totalPlazo=cantidad*.15;`

- `break;`

- `case 24:`

`totalPlazo=cantidad*.20;`

- `break;`

- 

`default:`

- `break;`

- 

`}`

- 

`//console.log(totalPlazo);`

- `return totalPlazo + totalCantidad + cantidad; // total a pagar`

→ `}`

- En formulario.js:
  - Importamos calcular total:
- `import {calcularTotal} from '../helpers';`

→//Realizar la cotización

- `const total= calcularTotal(cantidad, plazo);`
- `console.log(total);`

# 361. Agregando el Componente para la cantidad a pagar

- Mostrar el total a pagar en la parte inferior el resumen de cuanto el usuario va a pagar
- En App.js:
  - `const [total, guardarTotal]= useState(0);`
- Luego :
  - `<Formulario`
  - `cantidad={cantidad}`
  - `guardarCantidad={guardarCantidad}//prop`
  - `plazo={plazo}`
  - `guardarPlazo={guardarPlazo} // los nombra igual para no pensar en muchos nombres`
  - `total={total}`
  - `guardarTotal={guardarTotal}`
  - `/>`
  - `<p>Total a pagar: ${total}</p> // imprime en pagina`
- En formulario:
- Cambiamos esta parte:
  - `const Formulario = (props) => {`
  - `const {cantidad, guardarCantidad, plazo, guardarPlazo, total, guardarTotal}= props;`
- Luego en calcularPrestamo :
  - `//Realizar la cotizacion`
  - `const total= calcularTotal(cantidad, plazo);`
  - `// Una vez calculado, guardarTotal`
  - `guardarTotal(total); // esta función guarda automáticamente sin declararla normal, si no con lo del state`
- Si la cantidad a pagar es 0, que no muestre nada porque usuario ni siquiera a hecho uso de la app, pero en caso contrario darle indicaciones de cómo utilizar este proyecto

# 362. Carga Condicional de Componentes

- Como cargar un componente de forma condicional
- La cantidad a pagar la vamos a poner dentro de su propio componente
- Creamos un archivo en componentes, Resultado.js
- Creamos otro componente llamado Mensaje.js que le va a decir al usuario como utilizar este proyecto
- En Resultado.js:
  - `import React from 'react';`
  - 
  - `const Resultado = () => {`
  - `return (`
  - `<h1>Desde resultado</h1>`
  - `);`
  - `}`
  - 
  - `export default Resultado;`
- En Mensaje.js:
  - `import React from 'react';`
  - 
  - `const Mensaje = () => (`
  - `<p>Agrega una cantidad y plazo a pagar para cotizar</p>`
  - `);`
  - 
  - `export default Mensaje;`
- Carga condicional de componentes: si colocamos una cantidad y un plazo el mensaje con indicaciones debe desaparecer y mostrar el total, en algunos casos quieres que se cargue uno, en otros quieres que se cargue otro

- En App.js:
- Importamos Mensaje y Resultado
- `import Mensaje from './componentes/Mensaje';`
- `import Resultado from './componentes/Resultado';`

Dentro de function App:

- `// carga condicional de componentes en react`
- 
- `let componente;`
- 
- `if(total===0){`
- `componente=<Mensaje />`
- 
- `}else{`
- `componente=<Resultado />`
- `}`
- 
- Dentro del return:
- En className container
- `<div className="mensajes"> //de la hoja de estilos`
- `{componente} // variable`
- `</div>`
- 
- `// <p>Total a pagar: ${total}</p>`
- `// imprime en pagina`
- 
- `</div>`

# 363. Mostrando el Resultado

- En App.js:
  - `let componente;`
  - 
  - `if(total===0){`
    - `componente=<Mensaje />`
    - 
    - `}else{`
      - `componente=<Resultado`
        - `total={total}`
        - `plazo={plazo}`
        - `cantidad={cantidad}`
        - `/>`
    - `}`
  - En resultado.js:
    - `const Resultado = ({total, plazo, cantidad,}) => (`
      - `<div className="u-full-width resultado">`
        - `<h2>Resumen</h2>`
        - `<p>La cantidad solicitada es:${cantidad}</p>`
        - `<p>A pagar en:{plazo} Meses</p>`
        - `<p>Su pago mensual es de:${ (total/plazo).toFixed(2) } </p>`
          - `// -----cantidad de centavos`
        - `<p>Total a pagar: ${((total).toFixed(2))}</p>`
        - `</div>`
      - `);`
      -

# 364. Mostrando un Spinner de Carga

- <https://tobiasahlin.com/spinkit/>
- Creamos un archivo en componentes llamado Spinner.js:
- import React from 'react';
- import './Spinner.css'; // incluir css dentro del componente

```
→ const Spinner = () => {
• return (// colocamos el codigo html
• <div className="sk-chase">
• <div className="sk-chase-dot"></div>
• <div className="sk-chase-dot"></div>
• <div className="sk-chase-dot"></div>
• <div className="sk-chase-dot"></div>
• <div className="sk-chase-dot"></div>
• <div className="sk-chase-dot"></div>
• </div>
•);
• }
•
→ export default Spinner;
```

- Creo archivo en componentes Spinner.css donde pongo el css de la pagina
- Como create react app, viene con webpack es que puedes importar el spinner

```
• En Spinner.css :
→ .sk-chase {
• width: 40px;
• height: 40px;
• position: relative;
• animation: sk-chase 2.5s infinite linear both;
• margin: 0 auto; /* centrar spinner */
→ }
```

```
• En:
→ .sk-chase-dot:before {
• content: "";
• display: block;
• width: 25%;
• height: 25%;
• background-color: #33C3F0; /* // cambia color del spinner */
• border-radius: 100%;
• animation: sk-chase-dot-before 2.0s infinite ease-in-out both;
→ }
```



- En App.js:

```
→import Spinner from './componentes/Spinner';
```

- En *function* App() {

```
→const [cargando, guardarCargando]=useState(false); //para spinner, false para que no cargue al inicio
```

```
→ let componente;
```

- 
- if(cargando){
- componente=<Spinner/>
- }else if (total===0){
- componente=<Mensaje />
- 
- }else{
- componente=<Resultado
- total={total}
- plazo={plazo}
- cantidad={cantidad}
- />

```
→ }
```

- En return:

```
→ <Formulario
```

- cantidad={cantidad}
- guardarCantidad={guardarCantidad} //prop
- plazo={plazo}
- guardarPlazo={guardarPlazo} // los nombra igual para no pensar en muchos nombres
- total={total}
- guardarTotal={guardarTotal}
- **guardarCargando={guardarCargando}**

```
→ />
```

- En formulario.js:
- En `const Formulario = (props) => {`  
 → `const {cantidad, guardarCantidad, plazo, guardarPlazo, total, guardarTotal, guardarCargando} = props;`
- En `const calcularPrestamo = e => {`  
 → `// Habilitar el spinner`  
 • `guardarCargando(true);`  
 •  
 • `setTimeout(() => {`  
 •  
 • `//Realizar la cotizacion`  
 • `const total= calcularTotal(cantidad, plazo);`  
 •  
 • `// Una vez calculado, guardarTotal`  
 • `guardarTotal(total); // esta funcion guarda automaticamente sin declararla normal, si no con lo`  
 del state  
 • `//Desabilitar el spinner`  
 • `guardarCargando(false);`  
 → `}, 3000 );`

# 365. Deployment del Proyecto

- Subir proyecto a un servidor
- Como subir proyecto a un sitio web para que lo puedas compartir con alguien
- Netlify
- Detenemos el servidor y :  
→ `npm run build` // crea un paquete optimizado para produccion

Y en carpeta del proyecto tenemos carpeta build, esa es la carpeta qu subirias en netlify

Te da un dominio y asi es como publicas un proyecto en react

- Si tienes un hosting de esos tradicionales de ftp en build> index>buscas /static los que son js les tienes que quitar la diagonal si decides subirlo a un ftp

# **Sección 55: Introducción a NodeJS - Creando Un Proyecto c/ Express Sequelize Pug y Bootstrap**

# 366. Lo que vamos a Construir en este capitulo (Sitio Web con Node y MVC)

- Introduccio a node.js en la creacion de sitios web, crearemos un proyecto utilizando node.js, Express como nuestro servidor y sequelize para consumir y agregar datos a una BD
- Proximos viajes vienen desde un BD, como conectar tus aplicaciones de node.js y Express a una BD SQL utilizando Sequelize
- Como validar formularios con express y con node.js
- Agrega los testimonios a la BD y lo muestra
- Sequelize para insertar los datos que estamos escribiendo en nuestras apps o en nuestros sitios web de node.js y express dentro de una BD para que se almacenen de forma permanente siempre se van a almacenar en la BD
- Objetivo: Lo que has aprendido de JavaScript, lo puedas utilizar también en el backend con node.js y express, ademas de sequelize
- Crear app web o sitios web completos con JavaScript (todo lo que has aprendido conectado)

# 367. Qué es Node.js?

- Creando un sitio web
  - Es un entorno de código abierto, multiplataforma, que permite crear aplicaciones del lado del servidor en JavaScript, JavaScript, nació como el lenguaje para la web pero ha evolucionado mucho que ya te permite crear apps en el servidor
  - Permite crear APIs, apps web e incluso acceder a archivos y leer sus contenidos, puedes realizar pagos de forma segura, puedes acceder a una BD, autenticar usuarios, subir archivos y mucho más
  - VENTAJAS DE NODE:
  - Velocidad y rendimiento, node fue pensado para solucionar estos problemas, además de que es excelente para apps en tiempo real
  - JavaScript, si ya tienes experiencia en JavaScript, ahora podrás crear aplicaciones fullstack con node sin necesidad de compiladores ni lenguajes extras
  - npm: node package manager, es una gran cantidad de paquetes y librerías de código abierto ya listos para ser utilizados, entre ellos encontrarás algunos que te van a permitir subir archivos, autenticar usuarios, enviar emails, template engines, seguridad y mucho más
  - INSTALAR PAQUETES DE NPM
- npm install express
- nombre del paquete
- Ó → npm i // nombre del paquete
- Node funciona en windows, mac y linux
  - Una gran cantidad de librerías, soporte (porque es muy popular hoy en día), documentación, ejemplos y una gran comunidad

# 368. Qué es Express

- Introduccion al framework de node
- Todo el proyecto que vamos a realizar estara hecho sobre este framework
- Es el framework mas popular para node, y es la herramienta sobre la que estan desarrollados otros frameworks como mean, sails, loopback, graphHQL, yoga y otros
- CARACTERISTICAS DE EXPRESS
- Soporta los diferentes verbos http que son los mismos que tendria una api como son post, get, put, patch, delete en las rutas [routes]
- Soporta vistas que son los datos mostrados en la pantalla mediante determinadas respuestas (MVC ( model view controler) ver video)
- Permite la creacion de aplicaciones MVC que permiten tener una separacion de codigo y un mejor orden
- Soporta a middleware, mediante peticiones que se ejecutan en la tuberia de la peticion(puedes ejecutar una funcion, y una vez que finaliza que ejecute esta otra, esta otra, esta otra, y eso se le conoce como el middleware en express
- El middleware se utiliza bastante en express, permite acceder a archivos, revisar si un usuario esta autenticado o no, entre otras
- Ejemplo:
  - ```
Router.get('/administracion', // visitar la pagina de administracion
  authController.usuarioAutenticado, // un controlador que revisa si el usuario esta autenticado y otro que muestre el panel de
  administracion
  adminController.panelAdministracion
  → );
```

Tienes una funcion, despues otra, y vas agregando mas y mas funciones

- El middleware se ejecuta en pila, por lo tanto se ejecuta uno, y despues el otro.
- En algunos caso el middleware lo creas tu (similar a crear tus propias funciones) o tambien esta disponible en express una gran cantidad de middlewares(express tiene muchos middlewares)
- DESVENTAJAS DE EXPRESS
- A diferencia de otros frameworks como django, laravel o rubionrails, express no viene con baterias incluidas, ese termino se le da cuando un framework ya tiene autenticación, roles, ORM o generador de modelos, estas piezas usualmente se instalan y se configuran via npm y con tu codigo , de esta forma solo instalas lo que necesitas en tu app(lo cual puede ser una ventaja)

369. Instala la última versión de Node para utilizar Imports y Exports

- Instalar node, en windows tienes que instalar chocolatey y después tienes que instalar node y python
- Utiliza la última versión de código
- Ver versiones en cmd:

→ npm -v

→ node -v

- Creamos una carpeta en el escritorio agenciaViajesNode
- En cmd:

→ Cd Desktop

→ cd agenciaViajesNode // y aquí estaremos instalando express y creando nuestro proyecto

- Lo primero que vas a hacer siempre en todos los proyectos de node es crear un archivo que se llama package.json, este archivo tiene 2 fines, primero tener todas tus dependencias de proyecto(para mostrar algo en pantalla, para crear el servidor, para hacer consultas a una BD vas a instalar dependencias, esas dependencias tienen una version y si estas en un equipo de trabajo puedes compartir el archivo package.json y todos van a estar trabajando con las mismas versiones, todos van a poder tener el mismo código

En cmd:

→ npm init

→ Contesta la preguntas

- En carpeta agenciaViajesNode ya esta el package.json, abrimos carpeta en visual estudio

- Instalamos nuestra primer dependencia (Express) porque esa va a ser la plataforma que vamos a utilizar para crear nuestro proyecto,
- Como dependencia de producción(si se requiere en el servidor donde vamos a publicar nuestro proyecto)
- En cmd:
- npm i express
- Ahora tenemos express en dependencias en package.json
- Crea un package-lock.json este archivo usualmente nunca lo debes de modificar
- Crea carpeta node modules donde se instala todo lo que express requiere o todo lo que requieren las demás dependencias que vamos a ir instalando

- Instalamos otra dependencia de desarrollo en cmd:

→ npm install --save-dev nodemon

Como vamos a estar haciendo muchos cambios en desarrollo, es decir guardando cambios y agregando mas funciones nodemon detecta cambios en esos archivos y reinicia el servidor, de otra forma tendríamos que venir a la terminal, detener el servidor y volverlo a arrancar

- Creamos un archivo index.js archivo sobre el cual vamos a configurar express
- Vamos a crear nuestro servidor de express sobre el cual vamos a tener nuestra app
- En versiones anteriores de node utilizabas la sintaxis command js :
- En index.js:
- `const express = require('express');` // del paquete de express que hemos instalado extraemos express asignandolo a una variable
- `const app=express();` // ejecutamos funcion para ejecutar express
- `//Definir puerto`
- `const port = process.env.PORT || 4000;` // o si no E 4000
- `//` -----variable de entorno
- `//` puedes crear un archivo y manejar algunas variables del entorno local y una vez que hagas el deployment tendrías otras variables ya para el entorno de produccion (node maneja esa parte automaticamente por ti)
- `//` esa variable de entorno con el puerto nos la va a asignar usualmente donde hagamos el deployment porque no sabemos que puerto va a haber disponible, lo va asignar automaticamente
- `//` como estamos en local la parte PORT no va a E por lo tanto se va a imprimir este 4000 abajo, pero una vez que hagamos el deployment, la variable PORT ya va a E y ya no va a aparecer 4000, va a aparecer otro numero
- `//` que es muy dificil saber cual te va a tocar
- `//` Arrancando el servidor con .listen
- `app.listen(port, () => {`
- `//` -----puerto sobre el cual quieres ejecutar
- `console.log(`El servidor esta funcionando en el puerto ${port}`);` // si esta funcionando correctamente
- `//` -----
- `porque una vez que hagamos el deployment aerocu esta variable tiene que estar creada`
- `})`

- Como ejecutar index.js para arrancar el servidor, segunda utilidad del package.json
- Primera es manejar dependencias, segunda arrancar archivos
- Creamos nuestro primer script llamado dev(arrancar en desarrollo)
- En package.json:

```

-→ {
  •   "name": "agenciaviajesnode",
  •   "version": "1.0.0",
  •   "description": "Primer proyecto con Node.js",
  •   "main": "index.js",
  •   "scripts": {
  •     "dev": "nodemon index.js"
  •   },
  •   "author": "Julieta Rosales",
  •   "license": "ISC",
  •   "dependencies": {
  •     "express": "^4.17.1"
  •   },
  •   "devDependencies": {
  •     "nodemon": "^2.0.7"
  •   }
→ }

```

- Para mandar llamar script, desde cmd:

```

→ npm run dev

```

- -----ejecuta cualquier script en package
- -----nombre del script
- Para poder ver nuestro proyecto en navegador:
- Localhost:4000

- Al hacerlo dice que no puede obtener la url, no hemos definido el routing, no hemos definido las vistas pero ya está funcionando

370. Habilitando la Sintaxis de Imports y Exports

- Como utilizar la nueva sintaxis de imports
- La sintaxis `commonjs` no es propia de JavaScript, JavaScript solamente soporta los módulos, pero no soporta `commonjs` de forma nativa, `express` y `node` la adoptaron porque eso era lo que había antes porque no había módulos y esta era una forma de separar entre diferentes componentes, entre diferentes paquetes.
- Después salieron los módulos de JavaScript, los nativos, pero ya `node` y `express` y toda la comunidad de `node.js` había adoptado de esta forma todos los componentes que había, entonces comenzaron a reescribir, y hasta hace muy poco ya tiene soporte nativo para los imports y los módulos de JavaScript
- No puedes utilizar un import fuera de un módulo, en `express` y en `node` tienes que decirle que quieres utilizar módulos de JavaScript en el `package.json` (también puedes habilitar `commonjs` en el mismo lugar, `common` el default por ahora)

```
→ {  
  •   "name": "agenciaviajesnode",  
  •   "version": "1.0.0",  
  •   "description": "Primer proyecto con Node.js",  
  •   "main": "index.js",  
  •   "scripts": {  
  •     "dev": "nodemon index.js"  
  •   },  
  •   "type": "module",  
  •   "author": "Julieta Rosales",  
  •   "license": "ISC",  
  •   "dependencies": {  
  •     "express": "^4.17.1"  
  •   },  
  •   "devDependencies": {  
  •     "nodemon": "^2.0.7"  
  •   }  
→ }
```

```

→ import express, { response } from 'express';
•
• const app=express(); // ejecutamos funcion para ejecutar express
•
• //Definir puerto
• const port = process.env.PORT || 4000; // o si no E 4000
• //-----variable de entorno
• // puedes crear un archivo y manejar algunas variables del entorno local y una vez que hagas el deployment tendrias otras variables ya para el entorno de produccion (node maneja esa parte automaticamente por ti)
• // esa variable de entorno con el puerto nos la va a asignar usualmente donde hagamos el deployment porque no sabemos que puerto va a haber disponible lo va asignar automaticamente
• // como estamos en local la parte blanca no va a E por lo tanto se va a imprimir este 4000 abajo, pero una vez que hagamos el deployment, la variable blanca ya va a E y ya no va a aparecer 4000, va a aparecer otro numero
• // que es muy dificil saber cual te va a tocar
•
• app.get('/', (req, res) => { // peticion hacia un url, en este caso la pagina principal (get: cuando yo visito pagina)
• // -----Express utiliza el request y el response y el next
• // ----request es lo que tu envias
• // ----response es lo que express te envia por parte de node, pero tambien puedes crear tu propia respuesta:
• res.send('Inicio'); // send es metodo utilizado para mostrar algo en pantalla, tambien E .json que retorna respuesta de json( un objeto)
• // tambien E .render, para mostrar una vista
• });
•
• app.get('/nosotros', (req, res) => { // creamos una segunda pagina
• res.send('Nosotros');
• });
•
• app.get('/contacto', (req, res) => { // creamos una tercer pagina que cuando la consultamos en el navegador aparece lo que esta en el send
• res.send('contacto');
• });
•
• // Arrancando el servidor con .listen
• app.listen(port, () => {
• // -----puerto sobre el cual quieres ejecutar
• console.log(`El servidor esta funcionando en el puerto ${port}`) // si esta funcionando correctamente
• // -----
• porque una vez que hagamos el deployment aerocu esta variable tiene que estar creada
→ });

```

- Si tienes 30 url se esta cargando mucho el archivo principal, podemos separarlos, es una de las grandes ventajas del model view controller, veamos como llevar nuestras rutas hacia su propio archivo para un mejor orden

371. Routing en Express

- .render para mostrar un html completo
- Llevar vistas hacia su propio archivo
- Creamos carpeta routes y dentro archivo index.js donde colocaremos todo lo relacionado a las rutas
- Routes en index.js:
 - import express from 'express';
 -
 - `const router = express.Router();` // instancia de express
 -
 - `router.get('/', (req, res) => {`
 - `res.send('Inicio');`
 - `});`
 -
 - `router.get('/nosotros', (req, res) => {` // creamos una segunda pagina
 - `res.send('Nosostros');`
 - `});`
 -
 -
 - `router.get('/contacto', (req, res) => {` // creamos una tercer pagina que cuando la consultamos en el navegador aparece lo que esta en el send
 - `res.send('contacto');`
 - `});`
 -
 - `// se colocan rutas , dentro del router de express para agregarlo a app de express juntandolas, solo tienes que tener una instancia de express o se reinicia el servidor y no va estar conectada una parte con la otra`
 - `export default router;`
 - Enviar datos hacia una url .post
 - Eliminar un recurso .delete
 - Uno que abarca todos los verbos .use

- en index.js:
- `import express, { response } from 'express';`
- **`import router from './routes/index.js';`** // en esta version si se tiene que colocar la extension
-
- `const app=express();` // ejecutamos funcion para ejecutar express
-
- `//Definir puerto`
- `const port = process.env.PORT || 4000; // o si no E 4000`
-
-
- `//agregar Router a app`
- **`app.use('/', router);`** // desde la pagina principal agrega router que agrega todas las paginas
- `//quitamos de aquí las paginas y las pasamos al archivo de router`
- `// Arrancando el servidor con .listen`
- `app.listen(port, () => {`
- `console.log(`El servidor esta funcionando en el puerto ${port}`)// si esta funcionando correctam`
`ente`
- `})`

372. Que es un Template Engine? y Habilitando Pug

- Template engine y como habilitar uno
- Si queremos mostrar una respuesta mas completa o que el usuario vea algo de html tenemos que agregar una vista
- TEMPLATE ENGINE O MOTORES DE PLANTILLA
- Son la V(vista) del Model view controler MVC
- Permiten mostrar la parte visual(html) en una app de express, debido a que el modelo retorna un objeto o arreglo de datos, un template engine permite acceder a ese resultado y mostrarlo en pantalla
- Ej: el resultado de una consulta de una BD se puede pasar también a la vista y desde ahí mostrarlos
- CARACTERISTICAS DE UN TEMPLATE ENGINE
- Hay una gran variedad y cada uno tiene su propia sintaxis
- usualmente puedes escribir codigo JavaScript dentro del codigo html ej para iterar sobre un arreglo puedes hacerlo muy facilmente en el tamplate engine
- Si tienes experiencia en angular, react o vue, usualmente reemplazan estos templates engines en una app
- LOS MAS COMUNES EN NODE Y EXPRESS
- Pug- antes jade (el que vamos a ocupar en este proyecto)
- Ejs- embedded javascript
- Hbs- handlebars.js (mustache.js)
- React igual a sintaxis de react js
- TEMPLATE ENGINE SE INSTALA VIA NPM
- `npm install pug`
- Y se habilita en el archivo principal colocando:
- `//habilitar pug`
- `app.set('view engine','pug');`

- EJEMPLO PUG

→ If tareas.length

each tarea in tareas // each en lugar de forEach

li.tarea(data-tarea=`\${tarea.id}`)

→ p=tarea.tarea

// se utiliza indentacion que permite ir creando bloques de codigo

- EJEMPLO EJS

```
<div class="informacion-usuario">
  <div class="imagen">
    <% if(usuario.imagen ) { %>
      
    <% } %>
  </div>
  <div class="texto">
    <%- usuario.descripcion %>
  </div>
</div>
```

- HABILITANDO PUG en nuestro proyecto
- Detenemos el servidor
- npm install pug (si lo requerimos en producción para que muestre la info)
- Ya aparece en las dependencias de package.json
- Ejecuto:
- npm run dev // para volver a arrancar servidor
- Volvemos al archivo principal index.js donde hacemos todas las configuraciones de nuestro proyecto
- //habilitar PUG
- app.set('view engine', 'pug');
- Creamos una carpeta llamada views en raíz y dentro un archivo nosotros.pug
- //- etiqueta, espacio y lo que deseas imprimir
- h1 hola
- p Desde Pug

- En index.js de routes
- `router.get('/nosotros', (req, res) => { // creamos una segunda pagina`
- `res.render('nosotros');`
- `//` ----- nombre de una vista, busca archivo y lo muestra en navegador
- `});`
- Ya se ve lo que esta en erchivo pug en la pagina en el navegador , de esa forma ya hemos habilitado un template engine

373. Pasar variables hacia las vistas

- Como pasarle valores hacia la vista
 - Usualmente se hará en un controlador o desde routes index.js donde hacemos una consulta a una BD y esos resultados los queremos mostrar, o una variable, mostrarla de forma condicional si un usuario esta autenticado o no, o simplemente el valor de una variable
- `router.get('/nosotros', (req, res) => { // creamos una segunda pagina`
- - `const viajes= 'Viaje a Alemania'; //¿como lo paso a la vista?`
 -
 - `res.render('nosotros', { // dentro de estas llaves toda la info que requieras mandar hacia la vista nosotros, es un objeto`
 - `// ----- nombre de una vista, busca archivo y lo muestra en navegador`
 - `viajes`
 -
 - `});`
- `});`
- En nosotros.pug:
- `// - etiqueta, espacio y lo que deseas imprimir`
- `h1 hola`
 -
 - `// - p= viajes`
 - `// - si le pones un igual a la etiqueta, espera una variable de JavaScript`
 -
 - `// - otra forma:`
- `p #{viajes}`

- Crear una segunda vista o sea archivo inicio.pug en views
- En routers index.js:

```
→ router.get('/', (req, res) => {  
•   res.render('inicio');  
→ });
```

- Si tenemos multiples vistas, digamos 20 o 30 tendríamos que estar repitiendo el código y si nos dicen cambia la agencia de viajes porque la empresa ya se llama diferente, tendríamos que hacer 30 cambios
- En inicio.pug:
→ h1 Agencia De Viajes
→ p Inicio
- En nosotros.pug:
→ h1 Agencia De Viajes
→ p Nosotros
- Express y pug soportan el master page o pagina maestra, veamos como crear una pagina maestra que sirva como contenedor para las demás paginas

374. Creando un Layout Principal

- Crear una pagina maestra que nos evite estar repitiendo codigo
 - Creamos la carpeta layout en views, todo lo relacionado a layout: header, footer etc, todo iria en esta carpeta
 - Y creamos archivos index.pug (archivo principal de esa carpeta)
 - Todo proyecto web debe iniciar con el doctype
- 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`
 - `body`
 - **h1 Agencia De Viajes Nueva**
 - `//`- lo movemos hacia aqui lo de los otros pug y como es el layout principal, esto va a estar en todas las diferentes paginas
 - **block contenido**
- `//`-el contenido de los pug se va a agregar automaticamente y va a mantener todo lo demás
- Si
- `!+tab`, crea estructura basica siguiendo los lineamientos de pug, porque pug tiene apertura, pero no tiene cierre y respeta mucho indentacion

Cambios se van a reflejar donde estemos utilizando es layout, asi podemos agregar una hoja de estilos o algunos scripts y se van a agregar en los diferentes archivos, o ir definiendo secciones como el header o el footer

- En inicio.pug:
- `//`- utilizar block contenido en este inicio
- `//`- decirle que archivo quieres utilizar , no necesitas la extension porque es un archivo de pug
 - **extends ./layout/index**
 -
- `//`-agregar este contenido en el block contenido (identacion)
- **block contenido**
- **p Inicio**

375. Agregar Hojas de Estilos y Archivos estaticos


- Como cargar hojas de estilo bustrap, fuentes de google fonts
- Como agregar dos hojas de estilo a nuestro proyecto
- Las imágenes y las hojas de estilo usualmente se van a colocar en una carpeta llamada public
- Creo carpeta public en agenciaViajesNode, que va a ser a lo que el usuario tiene acceso sin tener acceso a las demas rutas
- Copiamos css e img hacia la carpeta public de los archivos que descargué en este video
- Vamos a decirle a express que public va a ser la carpeta publica, en esa puede tomar hojas de estilo o algunas imágenes
- Abrimos el index.js de la raiz, (archivo principal) y definimos la carpeta publica :
 - `// Definir la carpeta publica`
 - `app.use(express.static('public'));` // agregamos la carpeta publica como los archivos estaticos de express, de esta forma tenemos acceso a sus archivos
- Cargamos las hojas de estilo en archivo principal de pug index.pug
 - Link pone solo la sintaxis para cargar hojas de estilo
- En nosotros.pug:
 - `extends ./layout/index`
 - `block` contenido
 - `p Nosotros`
 - `img(src="img/cupon.jpg")`
 - `//` muestra la imagen en navegador por lo tanto tenemos acceso a archivos public en este caso img
- Ahora agregamos fuentes de google fonts, ya vimos como agregar hojas de estilo de forma local, ahora desde internet <https://fonts.google.com/>
- 2 fuentes staatliches> seleccionar este estilo
- Covered by your grace>seleccionar este estilo> embed da la sintaxis que se utilizaria en html tradicional, pero puedo utilizar una herramienta que convierta de html a pug <https://html-to-pug.com/>

- 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
- **link(href='https://fonts.googleapis.com/css2?family=Covered+By+Your+Grace&family=Staatl
iches&display=swap' rel='stylesheet')**
- **link(rel="stylesheet", href="/css/bootstrap.css")**
- //- primero tenemos que cargar bootstrap
- **link(rel="stylesheet", href="/css/style.css")**
- body
- h1 Agencia De Viajes Nueva
- //-
lo movemos hacia aqui lo de los otros pug y como es el layout principal, esto va a estar en todas las diferentes paginas
- *block* contenido
- //-el contenido de los pug se va a agregar automaticamente y va a mantener todo lo demas

376. Creando un Header y Navegación Principal

- Agregar 2 layout mas uno para header y otro para footer que estaremos agregando en este layout principal
 - Pug te permite agregar archivos parciales
 - Tenemos que tener un header con mas info en lugar de un h1, por lo tanto creamos archivo header.pug y otro llamado footer.pug , ambos en layout
 - La mayoría de las paginas el header y el footer son iguales
 - Movemos h1 hacia header
 - Como incluir header dentro de index.pug
- include header
- En index.pug:
- body
- - ***include header***
 - *block* contenido
 - *//*-el contenido de los pug se va a agregar automaticamente y va a mantener todo lo demas
- ***include footer***
- *//*- div(class="navegacion")
 - *//*- de hoja style
 - *//*- le puedes poner:
 - *//*- div.navegacion y te crea sintaxis de arriba lo mismo pasa con .navegacion
 -
 - *//*-agregando clases que pertenecen a bootstrap(compila el codigo normal conocido)
 -
 - *//*- creamos 2 columnas del lado izquierdo el logotipo del lado derecho la navegación
 - *//* container es una clase de bootstrap
- En header.pug:

→ header

- .navegacion
- .container
- .row.justify-content-center.justify-content-md-between.align-items-center.py-4
- **.col-md-4.col-6**
- [a\(href="/"\)](#) //- lleva a pagina principal
-  (src="/img/logo.svg", alt="sitio logo")
- **.col-md-8**
- nav.mt-5.mt-md-0.nav.justify-content-center.justify-content-md-end
- [a.nav-link\(href="/nosotros"\)](#) Nosotros
- [a.nav-link\(href="/viajes"\)](#) Viajes
- [a.nav-link\(href="/testimoniales"\)](#) Testimoniales
-

- El logotipo generalmente siempre debe llevar a la pagina principal
- Creo enlace
 - a+tab // completa el href para decile que cuando de click en enlace me lleve a pagina principal
- En col-md-8 creamos la navegacion y todas las navegaciones deben de ir dentro de la etiqueta nav, si no te crea un div
- Para poner 3 enlaces a navegaci3n:
 - a.nav-link*3 // agregamos texto con un espacio y despu3s el nombre

377. Middleware en Express

- Ahora en footer
- Creamos un footer
- py-5 es clase de bustrap
- .row para tener acceso al grid
- .col toma todo lo que halla disponible que nos deje col-md-6
- En footer.pug:
 - footer.container.py-5
 - .row
 - .col-md-6
 - nav.nav.footer.justify-content-center.justify-content-md-start
 - a.nav-link(href="/nosotros") Nosotros
 - a.nav-link(href="/viajes") Viajes
 - a.nav-link(href="/testimoniales") Testimoniales
 - .col
 - p.copyright.text-center.text-md-right
 - | Todos los Derechos Reservados
 - span=actualYear

- Obtener la fecha con JavaScript e imprimirla, eso por medio del middleware
- Abrimos index.js de raíz
- Creamos nuestro propio middleware, aquí ya hay un middleware de express con app. Donde utilizamos las funciones que ellos definen

• En index.js de raíz:

→ `//creando propio middleware`

- `//obtener el año actual (lo vamos a pasar por variables internas de express)cuando visitas pag siempre tienes req y res`
- `app.use((req,res,next) => {`
- `// console.log(res); // da mucha info en cmd`
- `// res.locals.unaVariable='Una Nueva Variable' // escribimos sobre objeto res`
- `// console.log(res.locals) // forma de compartir valores ej de este archivo hacia una vista`
- `const year= new Date();`
- `res.locals.actualYear=year.getFullYear();`
- `next();`

→ `});`

- Al correr nuevamente el servidor no puedo ver sitio web porque E next que te manda hacia el siguiente middleware, si no se lo pones, no va hacia el siguiente middleware

- Si lo pones ya carga la pagina

→ `Return next` //por si no se quiere ir al siguiente middleware (forzarlo)

- Como pasar valores de un archivo hacia una vista o de un archivo hacia otro, express utiliza locals que son como variables internas de express

- Aparece multiples veces porque .use se ejecuta en todos los diferentes metodos, puede ser un get, un post, etc

- Variables en locals faciles de leer en las vistas solo pones su nombre

→ `span= unaVariable` //forma de imprimir variables, sin el igual imprime el texto

378. Creando todas las vistas

- Crear el resto de paginas y que el routing funcione correctamente
- Trabajamos en la pagina de inicio.pug:
 - `extends ./layout/index`
 - `block contenido`
 - `main.container.mt-5`
 - `.row`
 - `h1=pagina`
- Creamos archivo viajes.pug y testimoniales.pug en raiz que tienen el mismo contenido de arriba
- En cada una de las paginas agregamos el texto que se va a mostrar en la pagina en azulito cuando des click en los enlaces a la pagina
- en index.js de routes:
 - `router.get('/nosotros', (req, res) => { // creamos una segunda pagina`
 - `res.render('nosotros', {`
 - `pagina:'Nosotros'`
 - `});`
 - `});`
- En index.js de la raiz creamos una nueva variable:
 - `app.use((req,res, next) => {`
 - `const year= new Date();`
 - `res.locals.actualYear=year.getFullYear();`
 - `res.locals.nombresitio="Agencia de Viajes";`
 - `next();`
 - `});`
- Ya tengo el routing completo, ya puedo visitar todas las paginas

- En index.pug para que aparezca el nombre en la pestaña
- En:
 - Head
 - `title #{nombresitio} | #{pagina}`

379. Creando la página de Nosotros

- Nosotros no tiene interacciones con la BD
- `class="img-fluid" // para que imagen no se encime con texto`
- <https://es.lipsum.com/> generar texto random
- Gracias a bootstrap es adaptable a los diferentes tamaños de pantalla
- En nosotros.pug:
→ `extends ./layout/index`
- `//-agregar este contenido en el block contenido (identacion)`
`block contenido`
 - `main.container.mt-5`
 - `.row`
 - `.col-md-5`
`img(src="img/nosotros.jpg", alt="imagen sobre nosotros", class="img-fluid")`
 - `.col-md-7`
`h1=pagina`
 - **blockquote** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque felis risus, volutpat nec sagittis vitae, ornare malesuada nibh. Sed ut enim vitae dui fringilla
 - **p** Sed et rhoncus arcu. Donec posuere neque a efficitur consectetur. Suspendisse at volutpat massa. Etiam non erat sagittis leo euismod egestas. Pellentesque laoreet at nisi quis
 - **p** Fusce eget sapien id leo vehicula tempor. Nullam fermentum elementum varius. Phasellus aliquam ipsum in nulla scelerisque pellentesque. Integer eu ante lacus. Morbi
 - .listado-iconos
 - `.container.mt-5.py-5.text-center`
 - `.row`
 - `.col-md-4`
`img(src="img/icono_seguridad.svg", alt="icono seguridad", class="img-fluid mb-4")`
h2.mb-4 Seguridad
p Etiam gravida erat ut volutpat pretium. Curabitur sed ullamcorper nulla. Morbi eget tellus turpis. Ut at gravida metus.
 - `.col-md-4`
`img(src="img/icono_destinos.svg", alt="icono destinos", class="img-fluid mb-4")`
h2.mb-4 Mejores Destinos
p Etiam gravida erat ut volutpat pretium. Curabitur sed ullamcorper nulla. Morbi eget tellus turpis. Ut at gravida metus.
 - `.col-md-4`
`img(src="img/icono_precios.svg", alt="icono precios", class="img-fluid mb-4")`
h2.mb-4 Los Mejores Precios
 - **p** Etiam gravida erat ut volutpat pretium. Curabitur sed ullamcorper nulla. Morbi eget tellus turpis. Ut at gravida metus.

380. Conectando a una base de datos MySQL

- Trabajando con la sección de viajes, toda info de aquí viene desde una BD
- Como conectar una BD hacia nuestro proyecto
- Corro wamp server
- Abro MySQL WorkBench> conexión local>create new schema> nombre agenciaviajes>apply>finish
- Selecciono BD agenciaviajes>server>dataimport>import from self-contained file> selecciono dump> default schema to be imported> agenciaviajes> ok > actualizar y ya podemos ver la tabla, en tabla en iconitos de derecha con el mouse
- Ahora instalamos dependencias que permiten conectar a la BD, pero tambien interactuarla, hacerle algunas consultas
- Abro una nueva pestaña en cmd, me aseguro de estar en agenciaViajesNode
- Dependencias de produccion es decir las requerimos tambien en el servidor
- npm install mysql2 sequelize
-----este seria un ORM
- Una vez instalado todo vamos a index.js de raiz
- Creo carpeta config en raiz que va a contener la configuracion y dentro creamos archivo db.js que va a tener la configuracion de nuestra BD
- Al guardar cambios en cmd dice BD conectada

- En index.js raiz:

```

→ import db from './config/db.js'; // extension es importante en esta nueva version
• const app=express(); // ejecutamos funcion para ejecutar express
•
• //conectar la base de datos
• db.authenticate()
• .then( ()=> console.log('Base de datos conectada'))
→ .catch( error=> console.log(error));

```

- En db.js:

```

→ import Sequelize from 'sequelize';
•
• const db=new Sequelize('agenciaviajes', 'root', '', {
• // ----- nombre BD a la que te quieres conectar
• // -----nombre del usuario
• // -----password
• // -----serie de configuraciones
• host: '127.0.0.1',
• port: '3306', // vienen en workbench en la conexion
• dialect: 'mysql', // porque sequelize es un ORM que tambien soporta posgress
• define:{
• timestamps: false // porque tiende a agregar un par de columnas cuando fue creado y cuando fue actualizado un registro
• },
• //resto configuracion de sequelize
• pool:{
• max:5,
• min:0,
• acquire:30000,
• idle:10000
• },
• operatorAliases: false
•
• });
→ export default db;

```

381. Introducción a Model View Controller y Creando un Modelo

- Creando el modelo para viajes y presentación del model view controller
- Creando el primer modelo
- MVC o model view controller: es un patrón de diseño de software que permite la separación de obligaciones de cada pieza de tu código en tu app web
- Permite tener un mejor orden, y cada parte de tu proyecto se va a dedicar a algo, el modelo hace una cosa, la vista hace otra cosa, el routing hace otra cosa y el controlador hace algo más
- Enfatiza la separación de la lógica de programación y lo que se muestra en pantalla, esto lo hace principalmente en el controlador con los datos que se le pasan hacia la vista

-

MODEL = MODELO

VIEW = VISTA

CONTROLLER = CONTROLADOR

MODEL

ENCARGADO DE LOS DATOS (DESDE UNA BASE DE DATOS) Y DE LA LÓGICA PARA MOSTRAR ESOS DATOS.

- Datos vienen usualmente desde una BD

EJEMPLO:

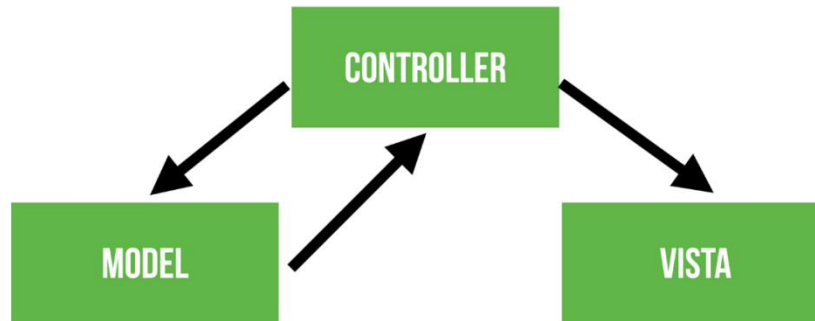
UN USUARIO QUIERE VER LA SECCIÓN DE PRODUCTOS, EL MODELO SE ENCARGARÁ DE REALIZAR ESA CONSULTA EN LA BASE DE DATOS.

- Con esos productos , tal vez solo quieres saber el nombre, código y cuantos quedan en existencia, entonces el modelo solo muestra estos campos, o muestra los 10 mas vendidos

- View: se encarga de lo que se ve en pantalla(html)
- Ej: si el modelo hace la consulta a la BD para los productos, es la vista la que muestra esos resultados (siguiendo nuestro ejemplo sera en pug donde se estaran mostrando esos resultados)
- Controller: es el que se comunica entre el modelo y la vista antes de que el modelo consulte a la BD es el encargado de mandarlo llamar, de saber que modelo es el que requerimos, y tambien una vez que el modelo tiene los resultados de la consulta es el que se encarga de pasarlos hacia la vista
- Router: encargado de registrar todas las urls o endpoints que nuestra app soporta
- Si el usuario accede a /productos, entonces el router va a llamar a un controlador que se va a comunicar con el modelo para obtener los datos, y estos seran pasados hacia la vista para ser mostrados
- DIAGRAMA: visitamos /productos, esta ruta tiene atado un controlador, este controlador tiene diferentes funciones , el controlador se comunica con el modelo, le pide los 10 productos mas vendidos, le pide los resultados de una busqueda y el modelo se los regresa al controlador, el controlador se los pasa hacia la vista y la vista muestra todo en el html

381. Introducción a Model View Controller y Creando un Modelo

/PRODUCTOS



3:00 / 6:31

Creando nuestro primer modelo

- Creamos carpeta models en raiz
- Se recomienda en archivo utilizar la primer letra como mayuscula, dentro creamos Viaje.js
- De momento nuestro routing index.js routes sirve como routing, pero tambien sirve como controlador, porque definimos la url y estamos diciendo que queremos que haga, lo que queremos que haga es algo que debe definir el controlador

- En Viajes.js:
- import Sequelize from 'sequelize';
- //importamos archivo de config porque tiene una instancia de sequelize llamado db
- import db from '../config/db.js';
-
- //definimos nuestro primer modelo como lo exporto, lo podria importar en el controlador para hacer la consulta
- export const Viaje=db.define('viajes', { // objeto de configuracion para definir cada una de las tablas(en este caso la BD ya tiene una tabla que ya tiene contenido), pero usualmente colocarias lo que planeaste para tu proyecto, el id no es necesario, se da por hecho que E
- // ---nombre de la tabla en BD
-
- titulo:{ // que tipo de dato va a tener y cuantos caracteres podemos utilizar (viene en documentacion de sequelize en que caso cada uno de ellos)
- type: Sequelize.STRING
- },
- precio:{
- type: Sequelize.STRING
- },
- fecha_ida:{
- type: Sequelize.DATE
- },
- fecha_vuelta:{
- type: Sequelize.DATE
- },
- imagen:{
- type: Sequelize.STRING
- },
- descripcion:{
- type: Sequelize.STRING
- },
- disponibles:{
- type: Sequelize.STRING
- },
- slug:{
- type: Sequelize.STRING
- },
- });

- En index.js de routes resaltamos:
- ```
→ router.get('/', (req, res) => { //definimos url
```
- **res.render('inicio',{ //decimos que queremos que haga**
  - **pagina:'Inicio'**
  - **});**
- ```
→ });
```

382. Creando un Controller

- Creando nuestro primer controlador
- Como realizar controladores en lugar de tener `res.render` en index de routes, porque seria el controlador quien se debe de encargar de que info se va a mostrar y que vista es la que se va a cargar
- Creamos carpeta controllers y dentro archivo `paginasController.js`

- En pagina controllers.js:
- // se va a encargar de mostrar las diferentes paginas // cortamos codigo de index.js routes
- `const paginaInicio=(req, res) => {`
- `res.render('inicio',{`
- `pagina:'Inicio'`
- `});`
- `}`
-
- `const paginaNosotros=(req, res) => { // creamos una segunda pagina`
- `res.render('nosotros',{`
- `pagina:'Nosotros'`
- `});`
- `}`
-
- `const paginaViajes=(req, res) => { // creamos una segunda pagina`
- `res.render('viajes',{`
- `pagina:'Viajes'`
- `});`
- `}`
-
- `const paginaTestimoniales=(req, res) => { // creamos una segunda pagina`
- `res.render('testimoniales',{`
- `pagina:'Testimoniales'`
- `});`
- `}`
-
- `export {`
- `paginaInicio,`
- `paginaNosotros,`
- `paginaViajes,`
- `paginaTestimoniales`
- `}`

- En index.js routes despues de cortar codigo e importar, queda:
 - import express from 'express';
 - **import {paginaInicio, paginaNosotros, paginaTestimoniales, paginaViajes} from '../controllers/paginasController.js';**
 - *const* router = express.Router(); // instancia de express
 - router.get('/', **paginaInicio**); // paginaInicio es el controlador
 - router.get('/nosotros', **paginaNosotros**);
 - router.get('/viajes', **paginaViajes**);
 - router.get('/testimoniales', **paginaTestimoniales**);
 - // se colocan rutas , dentro del router de express para agregarlo a app de express juntandolas, solo tienes que tener una instancia de express o se reinicia el servidor y no va estar conectada una parte con la otra
 - export default router;

383. Consultar la Base de datos

- Antes del render tendríamos que consultar la BD para pasarle el resultado hacia la vista
 - Un ORM como sequelize o mongus (mas populares en node) tiene los diferentes metodos para realizar consultas en una BD
 - <https://sequelize.org/master/manual/model-querying-basics.html#simple-select-queries>
 - findAll se trae todos los resultados
 - Where: traerte todos con una condicion
 - Traer con id con findByprimarykey
 - Vienen operadores ej tienda virtual buscar entre rango de precios
 - Antes se utilizaban promises, ahora hay ejemplos con async await, que es una mejor forma de hacerlo, mas corta sintaxis
 - COMO consultar viajes en Bd para pasarlos hacia la vista
 - revisar que se pase correctamente info hacia la vista en viajes.pug
 - Cuando le pones= espera codigo de JavaScript
- `p= JSON.stringify(viajes) // saber que se esta pasando todo, una forma de debugiar`
- Viajes es un arreglo y pug tiene diferentes metodos para poder iterar sobre ese arreglo
 - Para formatear fecha se puede usar una librería como date fns o moment
 - Colocar un icono <https://tablericons.com/> > calendar > se puede cambiar el color >click en icono y pegamos pero como es codigo html tenemos que convertirlo a pug

- En paginasController.js
- // importamos modelo porque tiene todos los campos de la tabla y gracias al modelo tenemos acceso a los metodos
- import {Viaje} from '../models/Viaje.js'
- `const paginaViajes= async(req, res) => { // creamos una segunda pagina`
- `//consultar BD`
 - `const viajes= await Viaje.findAll(); // se trae todos los resultados que halla en esa tabla`
 - `// ---modelo`
 - `console.log(viajes); //aparece en cmd , es un arreglo []`
 - `res.render('viajes', {`
 - `pagina:'Próximos Viajes',`
 - `viajes, //pasar hacia la vista`
 - `});`
 -
- `}`

- En viajes.pug:

```

→ extends ./layout/index
•
//agregar este contenido en el block contenido (identacion)
block contenido
•
  h1(class="mt-5 text-center") #{pagina}
•
  main.container.mt-5
•
    .row.proximos-viajes
•
      each viaje in viajes // iterador
•
        .col-md-6.col-lg-4.mb-4
•
          .card
•
            img.card-img-top(src=`img/destinos_${viaje.imagen}.jpg`)
•
            .card-body
•
              h2= viaje.titulo // titulo de la tabla de BD
•
              p
•
                svg.icon.icon-tabler.icon-tabler-calendar-event(xmlns=`http://www.w3.org/2000/svg` width='44' height='44' viewBox='0 0 24 24' stroke-
width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')
•
                  path(stroke='none' d='M0 0h24v24H0z' fill='none')
•
                  rect(x='4' y='5' width='16' height='16' rx='2')
•
                  line(x1='16' y1='3' x2='16' y2='7')
•
                  line(x1='8' y1='3' x2='8' y2='7')
•
                  line(x1='4' y1='11' x2='20' y2='11')
•
                  rect(x='8' y='15' width='2' height='2')
•
                | #{viaje.fecha_ida}-#{viaje.fecha_vuelta} // de BD
•
•
              p
•
                svg.icon.icon-tabler.icon-tabler-users(xmlns=`http://www.w3.org/2000/svg` width='44' height='44' viewBox='0 0 24 24' stroke-
width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')
•
                  path(stroke='none' d='M0 0h24v24H0z' fill='none')
•
                  circle(cx='9' cy='7' r='4')
•
                  path(d='M3 21v-2a4 4 0 1 4 -4h4a4 4 0 1 4 4v2')
•
                  path(d='M16 3.13a4 4 0 0 1 0 7.75')
•
                  path(d='M21 21v-2a4 4 0 0 1 -3.85') // iconos
•
                | #{viaje.disponibles} Disponibles // de BD
•
•
              p= viaje.descripcion.substr(0,100) + '...' // cortar string a 100 y luego muestrab ...
→ a(href=`viajes/${viaje.slug}`, class="btn btn-success btn-block") Más Información // agrega boton con texto mas info

```


384. Consultar la BD de cada viaje según la URL visitada

- Registrar nuevas rutas y controlador que se encarga de registrar
- Como crear la pagina con el detalle de cada viaje
- Si doy click en el boton en pagina, en cmd vienen los params: {viaje: 'viaje-inglaterra'} // seria la url sobre la cual estamos visitando
- Creamos vista viaje.pug
- Agregamos icono price

- En index.js routes:

→ import express from 'express';

- import {
 - paginaInicio,
 - paginaNosotros,
 - paginaTestimoniales,
 - paginaViajes,
 - **paginaDetalleViaje**

→ } from '../controllers/paginasController.js';

→ router.get('/viajes', paginaViajes);

→

router.get('/viajes/:slug', **paginaDetalleViaje**); // usando un comodin que puedes nombrar como quieras, en lugar de crear una ruta para el 1, una para el 2 y que todos nos muestren lo mismo, solamente que con info de cada viaje, ese comodin carga un metodo del controlador que va a ser igual, y la unica diferencia va a ser esa variable sobre el viaje en el cual estamos tratando de ver

- En paginasController.js:
- //Muestra un viaje por su slug
- **const paginaDetalleViaje= async (req, res) => {**
- // console.log(req.params.viaje); // params se asocia mucho con el comodin que tenemos en index.js
- // ---- para acceder al String, nombre del comodin
- **const{ slug }= req.params;** // para extraer y crear la variable del comodin
-
- // en caso de que no pueda hacer la consulta a la BD app no va a fallar
- try {
- **const viaje= await** Viaje.findOne({where:{ slug }}); // objeto slug:slug se trae uno solo findOne, ejemplo al que tenga el slug de viaje a rio de janeiro
-
- res.render('viaje',{
- **pagina: 'Informacion Viaje',**
- **viaje**
- })
- } catch (error) {
- console.log(error);
- }
- }
-
- export {
- paginaInicio,
- paginaNosotros,
- paginaViajes,
- paginaTestimoniales,
- **paginaDetalleViaje**
- }

- En viaje.pug:

→ `extends ./layout/index`

-
- `// -agregar este contenido en el block contenido (identacion)`
- `block contenido`
- `main.container.mt-5`
- `.row`
- `.col-md-5`
- `img(src=`/img/destinos_${viaje.imagen}_ln.jpg`, alt ="imagen viaje", class="img-fluid")`
- `// - trae cada una de las imagenes asociadas al viaje`
- `.col-md-7`
- `h1 #{viaje.titulo}`
- `p`
- `svg.icon.icon-tabler.icon-tabler-calendar-event(xmlns='http://www.w3.org/2000/svg' width='44' height='44' viewBox='0 0 24 24' stroke-width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')`
- `path(stroke='none' d='M0 0h24v24H0z' fill='none')`
- `rect(x='4' y='5' width='16' height='16' rx='2')`
- `line(x1='16' y1='3' x2='16' y2='7')`
- `line(x1='8' y1='3' x2='8' y2='7')`
- `line(x1='4' y1='11' x2='20' y2='11')`
- `rect(x='8' y='15' width='2' height='2')`
- `| #{viaje.fecha_ida}-#{viaje.fecha_vuelta}`
- `p`
- `svg.icon.icon-tabler.icon-tabler-coin(xmlns='http://www.w3.org/2000/svg' width='44' height='44' viewBox='0 0 24 24' stroke-width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')`
- `path(stroke='none' d='M0 0h24v24H0z' fill='none')`
- `circle(cx='12' cy='12' r='9')`
- `path(d='M14.8 9a2 2 0 0 0 -1.8 -1h-2a2 2 0 0 0 4h2a2 2 0 0 1 0 4h-2a2 2 0 0 1 -1.8 -1')`
- `path(d='M12 6v2m0 8v2')`
- `| #{viaje.precio} Dolares`
- `p`
- `svg.icon.icon-tabler.icon-tabler-users(xmlns='http://www.w3.org/2000/svg' width='44' height='44' viewBox='0 0 24 24' stroke-width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')`
- `path(stroke='none' d='M0 0h24v24H0z' fill='none')`
- `circle(cx='9' cy='7' r='4')`
- `path(d='M3 21v-2a4 4 0 0 1 4 -4h4a4 4 0 0 1 4 4v2')`
- `path(d='M16 3.13a4 4 0 0 1 0 7.75')`
- `path(d='M21 21v-2a4 4 0 0 0 -3 -3.85')`
- `| #{viaje.disponibles} Disponibles`
- `p= viaje.descripcion`

→

385. Creando un Formulario para Testimoniales

- Tener un formulario
- Como enviar datos a un formulario, como validarlos y como generar registros desde un formulario
- Formulario donde usuarios pueden enviar testimonial, tendremos sección para mostrar testimoniales
- Crear formularios, validar formularios y agregarla a una BD
- Siempre que llenas un formulario, tiene que ser de metodo POST
- Form-group // clase de bootstrap para dar estilos
- Para definir id en pug es con #nombre
- name="" nos permite leer lo que el usuario escriba en input y guardarlo en la BD
- For lo que hace es que cuando presionas en el label, activa el input, por eso for y id deben ser iguales
- Col para que tome todo el espacio disponible
- <label> representa una etiqueta para un elemento en una interfaz de usuario
- Cuando presionamos enviar, se mandan los datos al action, y se van a enviar por medio de un metodo POST
- El método POST no tiene **límite de cantidad de información** a enviar.
- La información proporcionada no es visible, por lo que se puede enviar **información sensible**.

- En testimoniales.pug:
- `extends ./layout/index`
-
- `//-agregar este contenido en el block contenido (identacion)`
- `block contenido`
- `main.container.mt-5`
- `h1.text-center.mt-5 #{pagina}`
- `.row`
- `//- formulario`
- `//- col tome todo el espacio disponible`
- `h2.d-block.w-100.text-center Agrega un Testimonial`
-
- `.row.justify-content-center.col`
- `.col-md-8`
- `form(action="" method="POST")`
- `.form-group`
- `label(for="nombre") Nombre`
- `input#nombre(type="text", placeholder="Tu Nombre", name="nombre", class="form-control")`
- `.form-group`
- `label(for="correo") Correo Electronico`
- `input#correo(type="text", placeholder="Tu Correo", name="correo", class="form-control")`
- `.form-group`
- `label(for="mensaje") Mensaje`
- `textarea#mensaje(class="form-control", name="mensaje", placeholder="Tu Mensaje", rows=3)`
-
- `input.btn.btn-success.btn-block(type="submit")`

386. Leer datos escritos en el Formulario

- Leer info que el usuario coloque en el formulario y el tipo de endpoint que vamos a requerir, de momento en routing todos son get , pero aquí enviamos algo por medio de metodo post
- Get se usa para cuando visitas una url
- POST forma en que envias datos hacia un servidor
- En action tenemos que definir una url sobre la cual vamos a escribir, en este caso enviare los datos hacia la misma url `action="/testimoniales"`
- En `testimoniales.pug`:
→ `form(action="/testimoniales" method="POST")`
- Como nuestro controlador es de paginas, creamos segundo controlador por lo tanto
- En archivo `controllers> testimonialController.js`
- `req.body` es lo que el usuario coloque en el formulario
- Get visitar la pag
- Post enviar datos hacia pagina

- En testimonialControllers.js
- `const guardarTestimonial=(req, res) => {`
-
- `// Validar...`
- `const {nombre, correo, mensaje}= req.body; // destructuring , crear variable y extraer valor`
-
- `const errores=[];`
-
- `if(nombre.trim() === ''){ // trim quita espacios en blanco al inicio y al final`
- `errores.push({mensaje: 'El nombre esta vacio'});`
-
- `}`
-
- `if(correo.trim() === ''){ // trim quita espacios en blanco al inicio y al final`
- `errores.push({mensaje: 'El correo esta vacio'});`
-
- `}`
-
- `if(mensaje.trim() === ''){ // trim quita espacios en blanco al inicio y al final`
- `errores.push({mensaje: 'El mensaje esta vacio'});`
-
- `}`
- `console.log(errores)// en cmd aparece el arreglo , podriamos iterar sobre este arreglo e ir mostrando cada uno de los mensajes`
-
- `// console.log(req.body); // poder ver lo que se escribe en formulario`
- `}`
-
- `export{`
- `guardarTestimonial`
- `}`

- En index de routes:

→import{

- guardarTestimonial

→} from '../controllers/testimonialController.js';

→router.get('/testimoniales', paginaTestimoniales);

→**router.post('/testimoniales', guardarTestimonial);** // hacia la pagina de testimoniales

En index de raiz:

→// Agregar body parser para leer los datos de formulario(a veces se tiene que instalar una dependencia)

→app.use(express.urlencoded({extended:true})); // ya aparece en cmd lo que escribe usuario

387. Validación de Formularios

- Tenemos arreglo con errores, tendríamos que pasarlo a la lista de testimoniales
- E una dependencia, llamada Express validator pero ahorita no
- Si yo mando el formulario vacio, me muestre de nuevo la vista de testimoniales, pero con los mensajes de error
- alert-danger clase de bustrap
- Ideal que los formularios mantengan el ultimo valor que se les escribio, ej escribi un review espectacular y se borra
- Value=mensaje
- Pasamos variables desde controlador y las estamos mostrando en la vista

EN testimonialController.js en guardarTestimonial:

```
→ if (errores.length>0){ // almenos hay un error
•   //mostrar la vista con errores
•   res.render('testimoniales',{
•     pagina: 'Testimoniales',
•     errores,
•     nombre, //para que no se borre lo que ya escribi en el formulario
•     correo,
•     mensaje
•   })
•
•   } else{
•     //Almacenarlo en la BD
→   }
```

- En testimoniales.pug:
- *extends ./layout/index*
-
- *//*-agregar este contenido en el block contenido (identacion)
- *block* contenido
- main.container.mt-5
- h1.text-center.mt-5 #{pagina}
- .row
-
- .col-md-12
- *//*- formulario
- *//*- col tome todo el espacio disponible
- h2.d-block.w-100.text-center Agrega un Testimonial
- *//*-----
- **if**(errores)
- *each* error in errores
- .alert.col.alert-danger.text-center=error.mensaje
-
- .row.justify-content-center.col
- .col-md-8
- form(action="/testimoniales" method="POST")
- .form-group
- label(for="nombre") Nombre
- input#nombre(type="text", placeholder="Tu Nombre", name="nombre", class="form-control", **value=nombre**)
- .form-group
- label(for="correo") Correo Electronico
- input#correo(type="text", placeholder="Tu Correo", name="correo", class="form-control", **value=correo**)
- .form-group
- label(for="mensaje") Mensaje
- textarea#mensaje(class="form-control", name="mensaje", placeholder="Tu Mensaje", rows=3)= **mensaje**
-
- input.btn.btn-success.btn-block(type="submit")

388. Almacenar los datos en la base de datos

- Estamos creando testimoniales, y la idea es tener los testimoniales aquí, pero no hemos creado esa tabla en la BD
- Tenemos que crear la tabla en la BD donde se van a guardar los testimoniales, y también tenemos que crear el modelo
- La mayoría de frameworks hoy en día tienen ese scaffolding que tú defines el modelo, que tablas vas a requerir, y cuando le das migrar, entonces las crea.
- Sequelize también los tiene, pero tiene el detalle de que cada vez que ejecutas ese comando limpia todas las tablas aunque tengan info
- Por lo tanto en workbench creamos las tablas
- New table> nombre: testimoniales, en el primero en id Auto increment
- Ahora creamos el modelo, en la carpeta models creamos
- Testimoniales.js:
→

```
import Sequelize from 'sequelize';  
import db from '../config/db.js';  
  
// export para poderlo importar en el controlador, y ahí crear los registros  
export const Testimonial=db.define('testimoniales',{ // objeto de configuración para definir cada una de las tablas( en este caso la BD ya tiene una tabla que ya tiene contenido), pero usualmente colocarías lo que planeaste para tu proyecto, el id no es necesario, porque el ORM da por hecho que E  
// ---nombre de la tabla en BD  
//ahora tenemos acceso al modelo y a todos esos métodos que tiene sequelize crear registros o listarlos  
  
  nombre:{ // que tipo de dato va a tener y cuantos caracteres podemos utilizar (viene en documentación de sequelize en que caso cada uno de ellos  
  }  
  type: Sequelize.STRING  
  },  
  correo:{  
    type: Sequelize.STRING  
  },  
  mensaje:{  
    type: Sequelize.STRING  
  },  
});
```

- En testimonialController.js:
- //importamos modelo en controlador
- import{Testimonial} from '../models/Testimoniales.js';

→const guardarTestimonial= **async** (req, res) => {

- Async await si tarda un poco en realizar la insercion a la BD
- En guardarTestimonial:

```
→ if (errores.length>0){ // almenos hay un error
•   //mostrar la vista con errores
•   res.render('testimoniales',{
•   pagina: 'Testimoniales',
•   errores,
•   nombre, //para que no se borre lo que ya escribi en el formulario
•   correo,
•   mensaje
•   })
•
•   } else{
•   //Almacenarlo en la BD
•   try {
•   await Testimonial.create({
•   nombre,
•   correo,
•   mensaje
•   });
•
•   res.redirect('/testimoniales');// lo lleva a testimoniales, y como tenemos un get a testimoniales, va a mostrar la pagina de testimoniales(terminamos la operación)
•   } catch (error) {
•   console.log(error)
•   }
→ }
```

- Si enviamos el formulario ya aparece en workbench

389. Consultar los Testimoniales de la base de datos

- Se muestren algunos testimoniales arriba para leer los testimoniales que los usuarios nos han dejado
- Abrimos el controlador de las paginas
- En paginasController.js:
→ `import {Testimonial} from '../models/Testimoniales.js'`

```
→ const paginaTestimoniales= async (req, res) => { // creamos una segunda pagina
•
  try {
•    const testimoniales=await Testimonial.findAll(); //traer todos los testimoniales que van a estar en variable que podemos pasar hacia la vista
•
•    res.render('testimoniales', {
•      pagina:'Testimoniales',
•      testimoniales // pasandolos hacia la vista
•    });
•  } catch (error) {
•    console.log(error);
•
•  }
•
→ }
```

En testimoniales.pug

```
→ block contenido
•   main.container.mt-5
•     h1.text-center.mt-5 #{pagina}
→     p=JSON.stringify(testimoniales)
```

390. Mostrar los Testimoniales en la Vista

- Iterar sobre arreglo de testimoniales para mostrarlo
 - findAll siempre retorna como arreglo
 - Footer se limita a la parte inferior de un elemento
 - el elemento `blockquote` define "una sección [dentro de un documento] que se cita de otra fuente".
 - El **footer** es la parte inferior de una estructura web en la que generalmente se incluyen links de navegación, enlaces de interés, copyright o botones a las redes
-
- Código que nos permite mostrar los testimoniales que vienen desde la BD
 - En `testimoniales.pug`:
 - - `blockquote.text-center` Lee sobre nuestros clientes y sus experiencias
 - `.row.testimoniales`
 - `each testimonial in testimoniales`
 - `.col-md-6.col-lg-4.mb-4`
 - `.card`
 - `.card-body`
 - `blockquote.blockquote`
 - `p.mb-0`
 - `|#{testimonial.mensaje}`
 - `footer.blockquote-footer`
 - `|#{testimonial.nombre}`
 -

- Si le doy en el boton enviar, marca error En testimonialController le estoy pasando los errores, pero no le estoy pasando los testimoniales , mientras que en paginasController si
- En testimonialController.js:
 - En guardar testimonial :
 - if (errores.length>0){ // almenos hay un error
 -
 - **//Consultar Testimoniales Existentes**
 - **const testimoniales=await Testimonial.findAll();**
 - //mostrar la vista con errores
 - res.render('testimoniales',{
 - pagina: 'Testimoniales',
 - errores,
 - nombre, //para que no se borre lo que ya escribi en el formulario
 - correo,
 - mensaje,
 - **testimoniales**
 - })

391. Primeros pasos con el Index

- Pagina de inicio, incluye 2 consultas viajes y testimoniales
- En layout, index, definir un nuevo bloque slider, donde yo lo requiera lo mando llamar, pero si no lo requiero no pasa nada, se queda vacio y no se ejecuta, slider es de hoja de estilos y hace que se vea asi
- `class="img-fluid"` // clase de bustrap para las imágenes

- En inicio.pug:
 - *extends ./layout/index*
 - **block slider**
 - **.slider**
 - **h1 Rio de Janeiro**
 - **em aventura**
 - *// - agregar este contenido en el block contenido (identacion)*
 - **block contenido**
 - **main.container.mt-5**
 -
 - **h2.text-center.mb-5 Sobre Nosotros**
 - **.row.mb-5**
 - **.col-md-6**
 - **p** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut sagittis tincidunt pellentesque. Nullam efficitur interdum diam, ac luctus arcu molestie eu. Vivamus sed porta quam, in mattis ligula. Ut id eros non erat ultrices consequat quis eget ex. Nunc mollis et lectus sit amet maximus. Sed vel mattis velit.
 - **p** Donec placerat purus erat. Phasellus orci eros, pellentesque quis consectetur et, gravida et orci. Vestibulum maximus tortor vel massa feugiat mattis. Proin imperdiet, augue sit amet tristique interdum, tortor urna condimentum metus, et auctor metus mi quis diam.
 - **.col-md-6**
 - **img(src="/img/inicio_nosotros.jpg", alt="imagen nosotros", class="img-fluid")**
 -
 - En layout index.pug creamos el block de slider
 - **body(class=clase) // imagen completa en el menu de inicio**
 -
 - *include header*
 -
 - **block slider**

- En paginasController.js
 - `const paginalInicio = (req, res) => {`
 - `res.render('inicio', {`
 - `pagina: 'Inicio',`
 - **clase: 'home'** // viene en la hoja de estilos para poner toda la imagen completa en el menu de inicio
 - `});`
 - `}`

392. Reutilizando el código de viajes

- Como consultar los viajes y mostrarlos
- Consultar la BD para mostrar 3 vajes en la parte inferior
- En paginaController, tenemos que consultar el modelo, y pasar los resultados hacia la vista
- En paginasController:
 - `const paginalInicio= async (req, res) => {`
 - - `// Consultar 3 viajes del modelo Viaje`
 - `try {`
 - `const viajes=await Viaje.findAll({limit: 3 }); //ORM te ofrece todo tipo de consultas`
 - - `res.render('inicio',{`
 - `pagina:'Inicio',`
 - `clase:'home', // viene en la hoja de estilos`
 - `viajes`
 - `});`
 - `} catch (error) {`
 - `console.log(error);`
 - `}`
- Copiamos lo que esta en viajes.pug y pegarlo en inicio.pug para mostrar los 3 viajes
- Y asi tenemos los viajes con el routing incluido
- Pero si me piden un cambio, lo tengo que hacer en ambos archivos, lo podemos simplificar con los parciales o con los includes de express
- Por lo tanto en el layout creo archivo bloque_viajes.pug, donde coloco el codigo que copie y pegue y lo quitamos de viajes.pug y ponemos un include, lo mismo en inicio
- Y ya tenemos un solo archivo

- En bloque_viajes.pug:
- .row.proximos-viajes
-
- *each* viaje in viajes
- .col-md-6.col-lg-4.mb-4
- .card
-  (src=`img/destinos_{\$viaje.imagen}.jpg`)
- .card-body
- h2= viaje.titulo
- p
- 

```

event(xmlns='http://www.w3.org/2000/svg' width='44' height='44' viewBox='0 0 24 24' stroke-width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')
    path(stroke='none' d='M0 0h24v24H0z' fill='none')
    rect(x='4' y='5' width='16' height='16' rx='2')
    line(x1='16' y1='3' x2='16' y2='7')
    line(x1='8' y1='3' x2='8' y2='7')
    line(x1='4' y1='11' x2='20' y2='11')
    rect(x='8' y='15' width='2' height='2')
    | #{viaje.fecha_ida}-#{viaje.fecha_vuelta}

```
- p
- 

```

svg.icon.icon-tabler.icon-tabler-users(xmlns='http://www.w3.org/2000/svg' width='44' height='44' viewBox='0 0 24 24' stroke-width='1.5' stroke='#dc135f' fill='none' stroke-linecap='round' stroke-linejoin='round')
    path(stroke='none' d='M0 0h24v24H0z' fill='none')
    circle(cx='9' cy='7' r='4')
    path(d='M3 21v-2a4 4 0 0 1 4 -4h4a4 4 0 0 1 4 4v2')
    path(d='M16 3.13a4 4 0 0 1 0 7.75')
    path(d='M21 21v-2a4 4 0 0 0 -3 -3.85')

```
- | #{viaje.disponibles} Disponibles
-
- p= viaje.descripcion.substr(0,100) + '...'
- a(href=`viajes/{\$viaje.slug}`, class="btn btn-success btn-block") Más Información

- Y en inicio.pug y en viajes.pug:
- *include* ./layout/bloque_viajes

393. Creando la sección de Descuento

- Mostrar un cupon de descuento
- En inicio.pug:
 - *include* ./layout/bloque_viajes
 - .descuento
 - .container
 - .row
 - .col-md-6
 - .contenido
 - h3 5% de Descuento
 - p.titulo Viaje a Canada
 - p.fecha 20 de Marzo de 2022-28 de Marzo de 2022
 - p.precio \$2,500 USD
 - .container
 - h2.mt-5.text-center Testimoniales

394.Como realizar múltiples Consultas

- Como mostrar los testimoniales en la pagina de inicio
- Creamos un bloque
- En layout archivo nuevo bloque_testimoniales.pug
- Use **rows** to create horizontal groups of columns.
- Cuando lo pongo en la pagina de inicio marca error, porque no hemos consultado la BD

- En bloque_testimoniales:
 - .row.testimoniales
 - *each* testimonial in testimoniales
 - .col-md-6.col-lg-4.mb-4
 - .card
 - .card-body
 - blockquote.blockquote
 - p.mb-0
 - |#{testimonial.mensaje}
 - footer.blockquote-footer
 - |#{testimonial.nombre}
- Es testimoniales.pug de donde corte el codigo
- En block contenido:
 - blockquote.text-center Lee sobre nuestros clientes y sus experiencias
 - include ./layout/bloque_testimoniales*
 - **.row**
- En inicio.pug donde tambien necesito el codigo
- En block contenido:
 - .container
 - h2.mt-5.text-center Testimoniales
 - ***include ./layout/bloque_testimoniales***
- Pero lo de inicio no funciona porque debemos consultar la BD

- En paginasController.js
- `// Consultar 3 viajes del modelo Viaje`
- `try {`
- `const viajes=await Viaje.findAll({limit: 3 }); //ORM te ofrece todo tipo de consultas`
- `const testimoniales=await Testimonial.findAll({limit: 3 }); // se debe llamar igual que en el each del bloque in testimoniales`
- `res.render('inicio',{`
- `pagina:'Inicio',`
- `clase:'home', // viene en la hoja de estilos`
- `viajes,`
- `testimoniales`
- `});`
- `} catch (error) {`
- `console.log(error);`
- `}`

Si funciona, pero... async await bloquea la ejecucion del codigo hasta que linea await se ejecute

Ejecuto el primer await, pero el segundo no esta haciendo nada hasta que el primero finaliza, luego comienza a ejecutarse, y pero lo que sigue no hace nada hasta que no termine, por lo tanto es muy lento lo ideal es que las dos lineas del await se ejecuten y arranquen al mismo tiempo, por lo tanto un promise (mejora muy grande en performance), tarda lo que tardan en ejecutarse ambas al mismo tiempo

Esta bien bloquear una, pero solo si depende del resultado de la otra linea, por lo tanto:

- En paginasController.js:

```
→ const paginalInicio= async (req, res) => {
•
  // Consultar 3 viajes del modelo Viaje
•  const promiseDB=[];
•
•  promiseDB.push(Viaje.findAll({limit: 3 }));
•  promiseDB.push(Testimonial.findAll({limit: 3 })); // los agrega en este orden al arreglo
•  try {
•    const resultado=await Promise.all(promiseDB); // arranca ambas consultas al mismo tiempo
•
•    res.render('inicio',{
•      pagina:'Inicio',
•      clase:'home', // viene en la hoja de estilos
•      viajes: resultado[0],
•      testimoniales: resultado[1]
•    });
•  } catch (error) {
•    console.log(error);
•  }
•
→ }
```

395. Preparando el proyecto para deployment - variables de entorno (BD)

- Realizar el deployment a un servidor
- Utilizando heroku
- Detenemos el servidor e instalamos:
 - `npm install --save-dev dotenv` // dependencia que nos permite generar variables de entorno, es decir vamos a tener algunas variables para desarrollo local y unas otras variables una vez que estemos ya en el servidor de heroku, va a ser necesario para el deployment, porque los datos de la BD, localmente son unos, y en el servidor van a ser otros, de esa forma vamos a tener un entorno local y uno de producción
- Creamos archivo en raíz `variables.env` en donde vamos a agregar algunas variables
- En `variables.env`: los datos vienen de `config db.js`
 - `BD_NOMBRE=agenciaviajes`
 - `BD_USER=root`
 - `BD_PASS=`
 - `BD_HOST=127.0.0.1`
 - `BD_PORT=3306`
 - - # estas son variables locales en heroku podemos generar variables que van a ser un poco diferentes y de esa forma vamos a tener un ambiente local (este caso o también llamado desarrollo) y el que es producción que va a ser el servidor donde lo vamos a publicar
- Instalamos extensión DotENV

- En db.js:
- `import Sequelize from 'sequelize';`
- `// leer variables con DotENV y tener acceso a info del archivo variables.env`
- `//require('dotenv').config({path:'variables.env'}) no funciona por lo tanto:`
- `import dotenv from 'dotenv';`
- `dotenv.config({path:'variables.env'});`
-
- `console.log(process.env.BD_HOST); // para ver si accedemos correctamente a las variables`
-
- `const db=new Sequelize(process.env.BD_NOMBRE, process.env.BD_USER, process.env.BD_PASS, {`
- `//`
- `//`
- `//`
- `//`
- `//`
- `host: process.env.BD_HOST,`
- `port: process.env.BD_PORT, // vienen en workbench en la conexion`
- `dialect: 'mysql', // porque sequelize es un ORM que tambien soporta postgres`
- `define:{`
- `timestamps: false // porque tiende a agregar un par de columnas cuando fue creado y cuando fue actualizado un registro`
- `},`
- Guardamos cambios y en terminal dice BD conectada y aparece el host
- Estamos accediendo correctamente a las variables, y a las lee desde aquí

396. Preparando el proyecto para deployment - variables de entorno (Servidor)

- Preparando mas para el deployment
- Utilizar la configuración de variables, tambien en el index.js raiz
- Ajustes a app para decirle que tiene que correr en determinado puerto, en determinado host, pero tiene que ser un poco dinamico, porque una vez que hagamos el deployment, heroku nos asigna tanto el host como el puerto
- En index.js:
 - `import dotenv from 'dotenv';`
 - `dotenv.config({path:'variables.env'});` // porque vamos a generar variable extra en variables.env
 - `/**Puerto y host para la app */`
 - `const host=process.env.HOST || '0.0.0.0';`
 - `// -----toma lo de variables.env (local)`
 - `// -----dir IP no valida, pero heroku le asigna una`
 - `//Definir puerto`
 - `const port = process.env.PORT || 4000; // o si no E 4000`
 - `// --esta parte la asigna heroku o si es local toma 4000`
 - `//-----variable de entorno`
 - `// Arrancando el servidor con .listen`
 - `app.listen(port, host, () => { // con valores ya sea locales o de heroku`
 - `// -----puerto sobre el cual quieres ejecutar`
 - `console.log('El servidor esta funcionando')// si esta funcionando correctamente`
 - `// -----porque una vez que hagamos el deployment heroku esta variable tiene que estar creada`
 - `}`

En variables. Env:

- `HOST=localhost`
`//#para tener una variable de entorno local, cuando trabajemos en local este archivo E y corre en local host, una vez que subamos a produccion, heroku nos asigna el servidor`

- No hay errores en cmd
- El sig paso para produccion
- En package.json
 - `"scripts": {`
 - `"dev": "nodemon index.js" // esta bien para desarrollo, pero para produccion, tenemos que crear un script de node`
- Lo cambiamos por
 - `"scripts": {`
 - `"dev": "nodemon index.js",`
 - `"start": "node index.js" // heroku corre npm start automaticamente`
 - `// ---- carpeta donde este index para que lo detecte o directamente el archivo`
 - `},`
- Guardamos cambios
- Detenemos el servidor y
 - `npm start // arranca desde el servidor de node`

397. Creando un repositorio de Git

- Crear un repositorio, tenemos que subirlo por medio de git y de ahí replicarlo a heroku
- Subir el proyecto a un repositorio de gitHub
- Tener instalado git y tener una cuenta de github
- Instalo github para windows // software para compu
- <https://git-scm.com/download/win> tambien instale solo git
- En github> repositories> new > nombre: agencia_deployment > descripcion: Deployment para la agencia de viajes
>crear repositorio> viene una serie de pasos que tenemos que hacer
- 1. decir en visual, que el proyecto en el que estamos va a ser un repositorio
- En cmd detengo servidor
- git init // hace que el proyecto en el que estamos trabajando sea un repositorio de git
- En visual creamos archivo .gitignore en raiz, permite quitar ciertos archivos para que no sean parte del commit una vez que lo subamos a un repositorio
- No requieres en repositorio node_modules lo que pongas en este archivo, se va a quitar
- Tampoco variables.env // porque son variables de entorno y solo van a estar disponibles localmente, tienen info de BD, no quiero que cualquiera pueda verla
- En .gitignore:
 - node_modules
 - variables.env
 - .DS_Store

- Commit : Una confirmación, o "revisión", es un cambio individual en un archivo (o conjunto de archivos)
- Vamos a agregar instrucciones al commit
- Agregar todos los archivos que han registrado un cambio . Para agregar todos
- En cmd:
 - `git add .` // agrega todos los archivos como parte del sig commit sin tenerlos que escribir uno por uno
- Darle un nombre al commit:
 - `git commit -m "primer commit"` // agrega todos los archivos como parte del commit
- Decirle que este repositorio que creamos, una vez que hagamos el push, es decir una vez que subamos los cambios, se tiene que subir aquí en git
 - `git remote add origin https://github.com/yrosalesd/agencia_deployment.git` // son diferentes copia y pega de git
 - // dice que este repositorio que tenemos, su url donde va a subir los cambios es esta

- Finalmente hacemos el push
→ `git push -u origin main` // pero no funcionaba, por lo tanto

Es posible que no funcionara porque teníamos que autorizar una app
A first-party GitHub OAuth application (Git Credential Manager)
with gist, repo, and workflow scopes was recently authorized to
access your account.

- `git push --set-upstream origin master` // esto lo va a subir a github,
o sea ya al servidor
- Si recargamos en la pagina de github , ya aparecen nuestros
archivos en github

398. Deployment a Heroku

- Crear la app de heroku
- Crear cuenta de heroku, espacio hasta de 500 Megs y cierta cantidad de trafico, si empiezas a tener miles de visitas, lo mas seguro es que te quieran cobrar, puedes hospedar dif apps node, php, python rubion rails etc
- <https://www.heroku.com/>
- Iniciamos sesión y creamos nuestra primera app
- Instalar heroku cli <https://devcenter.heroku.com/articles/heroku-cli>
- Ahora en cmd
- heroku // si responde, el cli se instalo correctamente
- Creamos nuestra app, en cmd:
- heroku create --remote production // crea nuestra primera app de heroku
- Y si recargo en la pagina en navegador, ya tengo mi primera app
- Luego subimos lo que tenemos como codigo git hacia app
- git push production master // cambia el repositorio de git al de heroku y subimos nuestro repositorio ya hacia heroku , detecta que es una app de node, sube todas las dependencias, instala las dependencias y corre los scripts
- Ya que termina de subir la app, la podemos abrir en la pagina de heroku> more > view logs> y puedes ver cierta info de tu app
- Vemos que marca un error, y si damos click en abrir app marca error, porque cuando agregamos variables de entorno a db.js, esas variables no E aquí porque ignoramos el archivo de variables.env, no se agrego como parte del commit, pero ahora tenemos que agregar esas variables en heroku, pero ahora con las variables de producción, con una BD de producción

399. Migrando la Base de datos

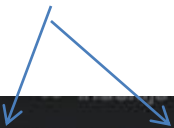
- Nos hace falta subir la BD y agregar las credenciales de la BD
- Heroku soporta BD MySQL, tenemos que habilitarla, así que damos click en resources> Add-ons> mysql> clearDB MySQL, eso agrega esta BD en nuestro proyecto> provision
- Registrar tarjeta de crédito, esto nos crea la BD, pero donde voy a encontrar las credenciales, para pasarla como variable de entorno:
- Settings>reveal config vars> viene la url de la BD de CLEAR_DATABASE_URL y en la otra pestaña viene mysql y toda la info, copiamos la parte de mysql y pegarlo en un archivo nuevo en visual untitled-1
- Tenemos que migrar nuestra BD a lo que es este servidor
- En cmd
- `mysqldump -u root -p agenciaviages > agenciaviages.sql`
- -----Nos genera un respaldo y va a exportar toda la BD completa
- ----- Para pasarle el usuario
- -----nombre BD
- -----como quieres que se nombre el archivo
- Nos pide contraseña
- Si abrimos la carpeta del proyecto, ya viene agenciaviages.sql (todo el archivo con el script para generar la BD)
- El siguiente paso es subirla al servidor
- En cmd:
- `heroku config | grep CLEARDB_DATABASE_URL // esto nos retorna la info de la BD`
- -----nombre de var en la pag de heroku
- Ahora para conectarnos:
- En cmd:
- `mysql -u nombre de usuario -h houting -p pass word (vacío) nombre de la BD a la que quieres reescribir < agenciaviages.sql`
- nos pide contraseña // parte azul

```
iMac-de-Juan:introNodeJS juandelatorre$ heroku config | grep CLEARDB_DATABASE_URL
CLEARDB_DATABASE_URL: mysql://b89535dc1ad184:7250ef2c@us-cdbr-iron-east-02.cleardb.net/heroku_ef894c4dfc30765?reconnect=true
iMac-de-Juan:introNodeJS juandelatorre$ mysql -u b89535dc1ad184 -h us-cdbr-iron-east-02.cleardb.net -p heroku_ef894c4dfc30765 < agenciaviages.sql
Enter password: ?
```

- Usuario:

```
mysql://b89535dc1ad184:7250ef2c@us-cdb-iron-east-02.cleardb.net/  
heroku_ef894c4dfc30765?reconnect=true
```

- Password



```
mysql://b89535dc1ad184:7250ef2c@us-cdb-iron-east-02.cleardb.net/  
heroku_ef894c4dfc30765?reconnect=true
```

- Servidor:

```
1 mysql://b89535dc1ad184:7250ef2c@us-cdb-iron-east-02.cleardb.net/  
heroku_ef894c4dfc30765?reconnect=true
```

- Nombre BD

```
mysql://b89535dc1ad184:7250ef2c@us-cdbr-iron-east-02.cleardb.net/  
heroku_ef894c4dfc30765?reconnect=true
```

- Generando variables.env ya en heroku
- En cmd:

```
iMac-de-Juan:introNodeJS juandelatorre$ heroku config:set BD_NOMBRE=heroku_ef894c4dfc30765
```

- `heroku config:set` `---`variable `----`nombre BD
- Esto agrega la variable de entorno
- Ahora en heroku en settings> reveal config var> ya aparece BD_NOMBRE con nombre BD
- Puedes subir todas las variables asi
- Otra forma:













- Agregar directamente en la pag de heroku

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Config Vars

Hide Config Vars

BD_NOMBRE	heroku_ef894c4dfc30765		
CLEARDB_DATABASE_URL	mysql://b89535dc1ad184:7250ef2c@us-cd		
BD_USER	b89535dc1ad184		
BD_PASS	7250ef2c		
BD_HOST	us-cdbr-iron-east-02.cleardb.net/		
BD_PORT	3306		
KEY	VALUE		

- {
- En more view logs encuentras mas info sobre los errores

- Error no puede encontrar el modulo dotenv porque es de desarrollo no de producción, por lo tanto reinstalamos
- Cmd:
- npm install --save dotenv
- Se elimina de dependencias de desarrollo para instalarlo en dependencias, la idea es que todas la dependencias se instalen en el servidor

- El package json va a ser diferente al que subimos y esto va a aplicarse siempre que hagamos cambios en un archivo del servidor, tal vez realicemos el deployment, después te pidan más cambios, haces los cambios, y esta es la forma en que vas a reescribir esos cambios
- Agregaremos los archivos al commit
- En cmd:
 - `git add .`
 - `git commit -m "solucionar errores en el package.json" // mensaje de lo que hicimos`
 - `git push -u origin master // subir los cambios en el repositorio`
- Lo replicamos en heroku
 - `git push production master // se va a subir nuestro código nuevo, incluso se va a subir todo el proyecto a heroku, el deployment se ha subido`
- Al abrir la app ya no salen errores, ya se carga la info desde la BD
- Siempre que hagamos cambios en tu código, tienes que realizar estos 4 pasos que vimos
- Cambios que realices localmente, y después los vas a poder subir (integración continua) tienes que utilizar git y heroku, este te lo va a facilitar mucho, para estar realizando muchos cambios en tu proyecto
- Heroku es una excelente opción para almacenar apps de node

Sección 56: FullStack JavaScript: API en Express, Node y MongoDB

401. Lo que vamos a construir

- Proyecto administrador de pacientes, lo estaremos integrando con diferentes tecnologías
- Tendremos una api hecha con express y con node, y la BD estara conectada a mongo
- Y este servidor de express y node va a administrar o va a manejar todas las diferentes peticiones, ya sea un delete, un post o un get para obtener los diferentes pacientes
- Tendremos una respuesta json, de esta forma podremos leer sus resultados en lo que viene siendo el frontend y tambien lo que es la app de escritorio
- Tendremos el frontend hecho en react
- Pacientes corre en un puerto diferente, dominio totalmente diferente, tiene separado el backend del front end
- Formulario, validacion, al crear cita nos lleva hacia el home, en el backend tenemos un registro nuevo, y si abrimos BD tambien tenemos un registro nuevo
- Tendremos un backend, una api de node, express y mongo, y el frontend con react
- Tambien veremos como crear una app de escritorio
- Utilizaremos electron para crear app de escritorio que si se hace algo se consume en front end y en back
- Elaboraremos partes de un crud
- Electron te permite integrar tecnologías web que ya conoces para crear este tipo de apps, y crea los instalables (archivos .exe zip o dmg) para distribuir app

402. Servidor en Express - Instalando Dependencias

- Crear carpeta en el escritorio, llamada fullStackJS y la abrimos en visual y creamos carpeta API que va a contener todo el backend, dentro creamos el archivo index.js que va a ser el archivo principal de nuestro servidor(estaremos creando el servidor con express)
 - En api abrimos terminal
 - Generamos un package.json que va a manejar todas nuestras dependencias y los scripts para ejecutar express y npm
- npm init

- Contestamos preguntas

```
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (api) fullstack-api
version: (1.0.0) Backend en Mongo y Express
Invalid version: "Backend en Mongo y Express"
version: (1.0.0)
description: Backend en Mongo y Express
entry point: (index.js)
test command:
git repository:
keywords:
author: Julieta Rosales
license: (ISC)
About to write to C:\Users\Julieta\Desktop\FullStackJS\API\package.json:

{
  "name": "fullstack-api",
  "version": "1.0.0",
  "description": "Backend en Mongo y Express",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Julieta Rosales",
  "license": "ISC"
}

Is this OK? (yes) yes
```

- Instalamos un par de dependencias para lo que es el proyecto, y una de desarrollo

En terminal:

→ `npm install express mongoose`

Mongoose es un ORM, una forma de consultar una BD de mongo de una forma muy sencilla

Aparecen en el package y se crea carpeta `node_modules`

- Dependencia que solo requerimos mientras desarrollamos nuestro sitio web:

→ `npm install --save-dev nodemon` // Como vamos a hacer muchos cambios en el código de nuestro servidor, cada vez que E cambios reinicia el servidor y toma esos últimos cambios, útil en desarrollo

403. Creando un servidor para la API

- Comenzar a agregar nuestro servidor, vamos a importar express y correr el servidor
- index.js archivo principal de configuracion de servidor (backend de proyecto)
- En index.js:
 - `// importamos dependencia de express (importacion de modulos en versiones anteriores) node esta adoptando imports y exports, disponible en ultimas versiones, y aun no funciona como se espera`
 - `const express = require ('express');`
 - `// Creamos el servidor`
 - `const app= express();`
 - `// agregamos lo que requerimos`
 - `// lo mas basico para crear un servidor, definimos un puerto y arrancamos el servidor`
 - `app.listen(4000, () => { // puerto donde va a correr porque app en react va a correr en 3000`
 - `console.log('Servidor funcionando')`
 - `});`
- Comenzamos a crear el servidor
- Instalamos express(forma muy sencilla de crear un servidor) no tiene nada incluido, tu agregas lo que requieres, incluso la creacion del servidor
- Despues tenemos que correr index.js en package.json ya que guarda dependencias pero tambien algunos scripts para comunicarnos con npm
- En package.json:
 - `"scripts": {`
 - `"dev": "nodemon ./index.js"`
 - `},`
- En terminal para correr script
 - `npm run dev`
 - nombre script

Con esto esta listo nuestro servidor de express

404. Conectando a MongoDB

- Como conectar nuestro proyecto, con una BD de mongo
- Instalamos mongoDB community server > custom> desmarcamos instalar mongo como servicio para que no se ejecute siempre, solo cuando lo inicio yo
- Ahora vamos a disco c, y creamos carpeta llamada data y dentro otra carpeta que se llame db donde se van a guardar las config de las BD etc.
- Arrancamos mongo db
- Disco c> archivos de programa> mongoDB> server>4.0>bin> ejecutamos mongod.exe(deja mongoddb en segundo plano) y arranco mongo.exe(console de mongo para poder trabajar en la BD)
- Por ahora solo arrancamos mongod
- Abrimos cmd
- y nos aseguramos de estar corriendo el servidor de mongo DB que tiene un ORM mongoose que permite interactuar con BD de forma sencilla

- Como conectar nuestra app con mongo
- En index.js:
 - `// conectar nuestra app con mongo`
 - `const mongoose=require('mongoose');`
 - `//conectar con mongodb`
 - `mongoose.Promise=global.Promise;`
 - `mongoose.connect('mongodb://localhost/veterinaria', { // si lo tienes corriendo en local la url siempre va a ser esta (definimos la conexión)`
 - `useNewUrlParser:true, // -----nombre BD`
 - `useUnifiedTopology:true,`
 - `useFindAndModify:false`
 - `});`
- Abrimos compass y creamos nueva BD , primero le damos en conectar y luego creamos BD veterinaria con collection Name pacientes
- De esta forma vamos a estar conectados con la BD,
- lo que va a interactuar directamente con la BD son los modelos sig video

405. Definiendo un Modelo

- Creamos el modelo que va a definir la estructura que va a tener cada registro en la BD
- Modelo, interactúa con la BD y con eso podemos insertar nuevos registros, editarlos, traerlos o eliminarlos
- Creamos una carpeta llamada models en API y dentro creamos Paciente.js
- En Paciente.js:
 - // vamos a hacer uso de algunos metodos
 - `const mongoose = require('mongoose')`
 - `const Schema=mongoose.Schema; // crea la estructura en la BD`
 -
 - `//definimos nuevo squema (nueva tabla con diferentes cambios)`
 - `const pacientesSchema = new Schema ({`
 - `nombre:{`
 - `type:String,`
 - `trim: true, // elimina espacios de mas`
 - `},`
 - `propietario: {`
 - `type: String,`
 - `trim: true`
 - `},`
 - `fecha: {`
 - `type: String, // date también te agrega los segundos`
 - `trim: true`
 - `},`
 - `hora:{`
 - `type: String,`
 - `trim: true`
 - `},`
 - `sintomas:{`
 - `type: String,`
 - `trim: true`
 - `}`
 -
 - `});`
 -
 - `// hacemos disponible este modelo porque lo requerimos en los controlers, es ahi donde vamos a insertar nuevos registros, y tambien traer nuevos registros`
 - `module.exports=mongoose.model('Paciente', pacientesSchema)// se asocia un nuevo modelo, mongoose todo lo tiene en una sola instancia, por lo tanto podemos hacer referencia a Pacientes a lo largo de toda nuestra app`
 - `// modelo llamado Paciente con forma de pacienteSchema`
 - Ahora tenemos listo nuestro modelo, solo sera 1, porque app es pequeña

406. Creando un Controller

- Comenzamos a trabajar con el controlador, vamos a crear una carpeta, un archivo nuevo
- Vamos a crear nuestro controlador
- Creamos nueva carpeta dentro de API Controllers y dentro archivo pacienteControllers.js
- Dentro del controlador vas a tener diferentes funciones que se van a asociar en cierta forma al modelo y en cierta forma con el routing
- En pacienteControllers.js:
 - `// cuando se crean un nuevo cliente (primer método dentro del controller)`
 - `exports.nuevoCliente=(req, res, next) => {`
 - `// TODO: Insertar en la BD, pero para mandarlo llamar tenemos que registrar una url en el routing, una url puede estar agregada o relacionada con uno o multiples metodos de un controlador`
 - `// siempre tienes que enviar una respuesta hacia la API`
 - `res.json({ mensaje: 'El cliente se agrego correctamente'}); //tiene parámetros que se pasan en el arrow de arriba`
 - `}`

407. Habilitando Routing

- Crear el routing para conectar nuestra app con el routing y el controlador
- Creamos el routing para conectar nuestra app (servidor funcionando), con cada uno de los controladores
- Creamos nueva carpeta routes en API y dentro index.js
- En index.js de routes :
 - `// importamos express para usar el routing`
 - `const express= require('express');`
 - `const router= express.Router();` // todos los metodos de router van a estar dentro de esta variable
 - `// importamos nuestro controlador:`
 - `const pacienteController = require('../controllers/pacienteControllers');`
 -
 - `// vamos a importar esto, en el archivo principal`
 - `module.exports=function(){`
 -
 - `//Agrega nuevos pacientes via POST (cuando mandas datos al servidor de registros nuevos usualmente se hace por medio de post)`
 - `router.post('/pacientes', // res para mejor orden nos da lineamientos, dice que si vas a crear un listado de pacientes, para agregar un nuevo paciente tienes que enviar petición post hacia url de pacientes, si quieres tener el registro de todos los pacientes, es a la misma url pero es get`
 - `// ejecutando el controlador`
 - `pacienteController.nuevoCliente`
 - `)`
 - `return router;` // para que estén disponibles en index.js de API, todas las diferentes rutas que vamos a registrar
 - `}`
- No le hemos dicho a nuestra API que queremos utilizar ese routing, abrimos index de api e importamos routing

- En index.js de api:
 - `const routes = require('./routes')` // Java Script busca el archivo index , por lo tanto solo nombre de
 - la carpeta
 - // habilitar routing
 - `app.use('/', routes())` // cuando visite sitio web ejecuta routes
 - `//---nombre de app`
 - `// ---middleware de express`
- El `module.exports` es un objeto especial que se incluye en cada archivo JavaScript en la aplicación Node.js de forma predeterminada. El `module` es una variable que representa el módulo actual, y `exports` es un objeto que será expuesto como un módulo. Por lo tanto, todo lo que asigne a `module.exports` se pondrá como un módulo.

408. Enviando una petición hacia el servidor

- Enviar nuestra petición, y ver que todo este conectado, utilizaremos una herramienta llamada postment
- Vamos a enviar una petición y usualmente lo vamos a hacer utilizando postman, ya que estamos creando una rest API <https://www.postman.com/downloads/>
- Enviamos nuestra primera petición
- No tenemos una url principal, y esa es la idea de rest, tener diferentes endpoints unicamente
- En postman signo de mas > post > en url: <http://localhost:4000/pacientes> > send , y en la respuesta tenemos el cliente se agrego correctamente
- O sea que nuestro router y nuestro controller esta funcionando correctamente
- Controller, Como insertar en la BD :
- Nos vamos a carpeta models > Paciente, tenemos que enviar datos con esta forma del schema
- Nos vamos a postman > body > x-www-form-urlencoded > key / value
- Insertamos los valores (en desarrollo bastante util)
- Fecha mongo la maneja como año, mes y día
- Como click en send, se envia una petición al servidor, pero en nuestro controlador tenemos que leer esos datos, usamos bodyparser

ns
uests,
run.

http://localhost:4000/pacientes

Save

</

POST

http://localhost:4000/pacientes

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	KEY	VALUE	DESCRIPTION		Bulk Edit
<input checked="" type="checkbox"/>	nombre	Hook			
<input checked="" type="checkbox"/>	propietario	Julieta			
<input checked="" type="checkbox"/>	fecha	2019-12-10			
<input checked="" type="checkbox"/>	hora	10:30			
<input checked="" type="checkbox"/>	sintomas	Aqui algunos sintomas			

Body

Cookies

Headers (7)

Test Results

200 OK

262 ms

283 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

"mensaje": "El cliente se agrego correctamente"

409. Leyendo los contenidos de una petición con body-parser

- Habilitar el bodyparse y ver como vamos a leer, lo que el usuario esta enviando
- Para poder leer lo que pusimos en key/value, tenemos que habilitar el body parser, esta forma va a ser exactamente lo mismo si creas un sitio web, porque esa es la forma en que vas a leer los campos que vienen desde un formulario o diferentes inputs
- En index.js de API:
 - `const bodyParser= require('body-parser'); // importamos bodyparser`
 - `//habilitar el body-parser`
 - `app.use(bodyParser.json());`
 - `app.use(bodyParser.urlencoded({extended:true})); // extraer lo que el usuario envia, o una peticion que se envia a l servidor, dentro de una variable que vamos a tener en el controller automaticamen`
 - `te`
- En pacienteControllers.js:
 - Dentro de nuevoCliente:
 - `// coloca todo el request de lo que el usuario envia dentro de variable req.body (lo que el usuario esta enviando)`
 - `console.log(req.body);`
- abrimos postman > send y en la terminal de visual aparece lo que el usuario ha enviado o lo que esta en postman . en terminal porque es un console.log en express, por lo tanto se muestra en el servidor

410. Insertando un registro en la BD

- Traer el modelo para hacer la inserción en la BD , como guardar los registros en la BD
- Como importar nuestro modelo que va a estar interactuando con la BD y lo que es inserción o traer registros, van a estar a cargo del modelo siempre
- En pacienteController.js :
 - `// importamos el modelo`
 - `const Paciente = require('../models/Paciente');`
 - `// cuando se crean un nuevo cliente (primer metodo dentro del controller)`
 - `exports.nuevoCliente= async (req, res, next) => {`
 - `//realizando la insercion`
 - `//crear objeto de paciente con datos de req.body`
 - `const paciente = new Paciente(req.body);`
 - `try { // porque puede ser que en el momento que envíe los datos, el servidor se caiga`
 - `//intentar guardarlo en la BD`
 - `await paciente.save(); // save de mongoose para guardar en la BD`
 - `// siempre tienes que enviar una respuesta hacia la API`
 - `res.json({ mensaje: 'El cliente se agrego correctamente'}); //tiene parametros que se pasan en el arrow de arriba`
 - `} catch (error) {`
 - `console.log(error);`
 - `next(); // baya hacia la sig funcion`
 - `}`
 - `}`
- Guardamos cambios, y en postman > send , eso inserta el registro y en la BD de compass, abro la BD, pacientes y veo mi primer registro con la misma info que coloque en postman
- Del send se va al routing, hacia url, va hacia el controller que le dice que se tiene que ejecutar la función de nuevoCliente, donde tenemos importado el modelo, en el modelo insertamos todos los datos, se llena el modelo, ya tenemos una instancia en paciente, guardamos cambios, de esa forma es como se comunican y se mandan llamar routing, controller, modelo y finalmente todo con la BD

411. Obtener todos los registros

- Seguimos trabajando con los demás endpoints que nos recomienda rest, el siguiente seria como traernos todos los registros, es decir vamos a crear una nueva ruta, que en este caso va a ser get, y una nueva función en el controller que nos traiga todos los registros
 - Como traernos los registros, via rest
 - Vamos a crear un segundo endpoint que vamos a poder probar desde navegador
 - En el index.js de routes en `module.exports=function(){:`
 - `//Obtiene todos los registros de pacientes en la BD`
 - `router.get('/pacientes',`
 - `pacienteController.obtenerPacientes`
 - `);`
 - En pacienteControllers.js
 - `/**Obtiene todos los pacientes */`
 - `exports.obtenerPacientes= async (req,res,next) => {`
 - `try {`
 - `const pacientes= await Paciente.find({});` // consultamos la BD .find para traernos todos los registros
 - `res.json(pacientes);` //respuesta para que la pueda leer react en json
 - `} catch (error) {`
 - `console.log(error);`
 - `next();`
 - `}`
 - `}`
 - `}`
- Guardamos cambios, abrimos /pacientes en el navegador <http://localhost:4000/pacientes>, y vemos los registros que tengo en la BD
 - Y si en postman cambiamos a get > enviar , tenemos nuestros 2 pacientes en la respuesta del software postman

412. Obtener un registro por ID

- Como obtener un paciente en especifico, tenemos que pasarle un id, como hacerlo en express y seguir los lineamientos de rest
- Como escribir nuestro siguiente route para traernos un paciente en especifico
- En index.js de routes en `module.exports=function(){`:
 - `//Obtiene un paciente en especifico (ID)`
 - `router.get('/pacientes/:id', // como no se que id el paciente va a buscar o visitar por lo tanto se usa el comodin y asi cualquier url ej: /pacientes/1 responde a este routing`
 - `pacienteController.obtenerPaciente`
 - `)`
 -
 - En pacienteControllers.js:
 - `/** Obtiene un paciente en especifico por su ID */`
 - `exports.obtenerPaciente= async(req, res, next) =>{`
 - `try {`
 - `const paciente= await Paciente.findById(req.params.id)`
 - `//cuando enviamos nuestra peticion va a ser ej /pacientes/1`
 - `//` ----- los params es lo que va estar en donde se encuentra el 1
 - `//` -----como puse el nombre en el comodin
 - `res.json(paciente)// si E lo retornamos como JSON`
 - `} catch (error) {`
 - `console.log(error)`
 - `next();`
 - `}`
 - `}`
 -
- Si en el navegador ponemos el id, solo nos trae ese registro <http://localhost:4000/pacientes/607f4aeabb1d8a32c8db5826> , lo mismo pasa si pongo esto es postman

413. Actualizando un registro

- Como actualizar un registro
- En index.js de routes, en `module.exports=function(){`:
 - `// Actualizar un registro con un id especifico`
 - `router.put('/pacientes/:id',`
 - `pacienteController.actualizarPaciente`
 - `);`
- En pacienteControllers.js
 - `/**Actualiza un registro por su ID */`
 - `exports.actualizarPaciente=async(req, res, next) =>{`
 - `try {`
 - `const paciente=await Paciente.findOneAndUpdate({_id : req.params.id}, req.body, {`
 - `//` ----busca un registro y lo actualizarlo pasandole ciertos valores, lo actualiza automaticamente
 - `//` -----como aparece en la BD Compass
 - `//` ---usuario que tiene ese id es el que queremos actualizar
 - `//` ----lo voy a actualizar con el req.body
 - `new:true // que nos traiga el resultado, el nuevo documento, porque si no te trae el primero, antes de la actualización`
 - `});`
 - `res.json(paciente);`
 -
 - `} catch (error) {`
 - `console.log(error);`
 - `next();`
 -
 - `}`
 - `}`
- Para probarlo, en postman> put> cambiamos algun key/value> send, y en la BD ya deben estar los cambios si recargamos

414. Eliminando un registro

- Eliminar un registro
- En index.js routes en `module.exports=function(){`:
 - `//Eliminar un paciente por su ID`
 - `router.delete('/pacientes/:id',`
 - `pacienteController.eliminarPaciente`
 - `);`
- En paciente controllers:
 - `/** Elimina un paciente por su ID */`
 - `exports.eliminarPaciente=async(req, res, next) => {`
 - `try {`
 - `await Paciente.findOneAndDelete({_id : req.params.id})// no es necesario asignarlo a una variable porque lo vamos a eliminar, a menos que quie`
`ras ver lo que estas eliminando`
 - `res.json({mensaje: 'El paciente fue eliminado'})`
 - `} catch (error) {`
 - `console.log(error);`
 - `next();`
 - `}`
 - `}`
- Para probarlo, en postman> delete> send y como respuesta tenemos el paciente fue eliminado
- En compass recargamos y solo queda un registro
- Ya tenemos las 4 operaciones del crud mas el detalle de un registro
- Luego veremos como integrar todo esto con react, y luego como integrarlo con electrón

Sección 57: FullStack JavaScript - Creando el Front End con React

415. Creando la App y Primeros Pasos

- Ya tenemos el backend, que viene siendo una API utilizando rest
- Abrimos una nueva terminal, es importante tener corriendo mongo, también tu API
- Creamos un nuevo proyecto
- Abrimos la carpeta del proyecto en cmd
- Lo que quiero es tener la API como el frontend en la misma carpeta
- En cmd:
 - `npm create-react-app frontend`
 - Como ya tenemos arrancada una terminal en visual en el servidor del backend, doy click derecho en la carpeta del frontend en visual y abrimos la terminal , y en esta:
 - `npm start //` de esa forma tengo 2 terminales, una con la API y otra con el frontend
- En cmd abro la carpeta frontend porque vamos a instalar algunas dependencias
- Vamos a agregar algunos archivos frontend> public> index.html pegamos después del titulo de react
- <https://gist.github.com/juanpablogdl/9f75be22c9fa50b6f0d7ccb63e03408c>
 - `<title>React App</title>`
 - `<link href="https://fonts.googleapis.com/css?family=Roboto|Staatliches&display=swap" rel="stylesheet">`
 - `<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">`
 - `<link rel="stylesheet" href="css/app.css">`
- En public creamos una nueva carpeta css, y dentro archivo app.css y pegamos lo de la pagina
- Abrimos src > App.js > eliminamos el contenido:
 - `import React from 'react';`
 - `function App() {`
 - `return (`
 - `<h1>Pacientes</h1>`
 - `);`
 - `}`
 - `export default App;`
- Eliminamos App.css / App.test.js / index.css
- En index.js elimino referencias de index.css
- Guardamos cambios y recargamos

416. Agregando Routing y Componentes Principales

- Este proyecto va a tener diferentes paginas, se hace con react router dom, veamos como agregarlo al proyecto en el siguiente video
- Vamos a instalar react router dom, de esta forma podemos ir a diferentes paginas y cargar diferentes componentes, hacer como si fuera un sitio web en el que puedes ir a diferentes lugares, es una dependencia que no viene en react

- En cmd:

→ `npm install react-router-dom`

- En App.js

→ `import React from 'react';`

- `// vamos a importar lo que requerimos de routing`
- `import {BrowserRouter as Router, Route, Switch} from 'react-router-dom';`
- `// importamos el componente:`
- `import Pacientes from './components/Pacientes';`
- `import NuevaCita from './components/NuevaCita';`
- `import Cita from './components/Cita';`

- `function App() {`
- `return (`
- `<Router>`
- `<Switch>`
- `<Route //definimos cada ruta:`
- `exact`
- `path="/" // url que el usuario va a visitar`
- `component={Pacientes}`
- `/>`
- `<Route //agregando mas rutas`
- `exact`
- `path="/nueva" // url que el usuario va a visitar`
- `component={NuevaCita}`
- `/>`
- `<Route //agregando mas rutas`
- `exact`
- `path="/cita/:id" // tambien E comodin para IDs`
- `component={Cita}`
- `/>`
- `</Switch>`
- `</Router>`
- `);`
- `}`

→ `export default App;`

- En carpeta components en Cita.js :

→ `import React from 'react';`

- `const Cita = () => {`
- `return (<h1>Desde Cita</h1>);`
- `}`
- `export default Cita;`

- Contienen lo mismo los otros componentes

- Creamos un componente llamado pacientes en src> nueva carpeta> components, dentro Pacientes.js
- En pag nueva va a ir el formulario que nos va a permitir enviar las peticiones en lugar de hacerlo desde postman, ya va a ser desde una interface web
- Al probar en el la pagina /cita/id (un numero), vemos que pierde los estilos, esto es porque falta una diagonal en public> index.html
→ <link rel="stylesheet" href="/css/app.css">
- Con esto tenemos listo nuestro routing, ya nuestra app soporta diferentes pag

417. Agregando una base para las consultas a la API con Axios

- Comenzamos con el primer componente “/”
- Listado de todos los pacientes
-
- Vamos a consumir una api externa hecha por nosotros
- Json es un formato que se le conoce como transporte, por lo tanto puedes consumir de un servidor y mostrarlo en otro, algunos proyectos fueron hechos con fetch y otros con axios
-
- Axios simplifica mucho mas lo que es el desarrollo, lo vamos a utilizar, y como lo vamos a usar en multiples componentes, vamos a ver como crear un cliente axios
- La ventaja principal de este cliente, es que podemos tener una conexión base que apunte a un dominio en especifico y una vez que hagamos el deployment, solamente cambiamos un valor, y todos los diferentes endpoints van a verse actualizados
-
- Abrimos cmd desde carpeta frontend
- npm install axios
- En src creo nueva carpeta config, dentro creo archivo axios.js donde voy a definir mi cliente de axios
-
- React soporta las variables de entorno, en el mismo nivel del package.json creamos nuevo archivo .env.development estas cargan automaticamente cuando estes en desarrollo, una vez que creas el build de produccion, este archivo simplemente es ignorado
-
- En axios.js:
→ import axios from 'axios';
- `const clienteAxios= axios.create({`
- `// definimos url base, para que cuando lo llevemos podamos cambiar la base que es localhost:4000 y seguimos manteniendo las rutas ej /paciente`
`s/123344`
- `// creamos cliente:`
- `baseUrl: process.env.REACT_APP_BACKEND_URL`
- `});`
- `export default clienteAxios;` // de esta forma puedo utilizar este cliente axios que va a tener una url base a lo largo de los diferentes componentes (proyectos una vez que los lleve a deployment sean mas faciles de mantener, no tengo que hacer tantos cambios)
-

- En .env.development
- # forma de crear estas variables de entorno
- REACT_APP_BACKEND_URL= http://localhost:4000 #tenemos una variable de entorno, que una vez que lo llevo a produccion, creo un archivo .env.production y ahí puedo colocar la misma variable pero ya con el dominio donde vaya a quedar mi servidor
- # -----TU DEFINES EL NOMBRE
- # esto se va a propagar a lo largo de todos los componentes es una gran ventaja para producción
- Para probar que funcione la var de entorno en App.js:
- ```
function App() {
```
- ```
// esta comentado porque al final no lo ocupo
```
- ```
// leer var de entorno
```
- ```
//console.log(process.env.REACT_APP_BACKEND_URL)// CUANDO VISITE PAGINA PRINCIPAL EN CONSOLA DEBO VER ESE VALOR
```
- Siempre que agregas variables de entorno de este tipo, tienes que detener tu servidor del frontend
- ctrl+c
- npm start
- Ya que estemos en produccion podemos crear una var de entorno con el mismo nombre ej heroku te permite crear esas variables desde la interface y netlify tambien te permite en el caso del frontend
- La proxima vez que importemos nuestro cliente de axios ya va a tener localhost: 4000 y podemos comenzar a utilizar las diferentes rutas, sin tener que escribir el dominio

418. Consultando la API con Axios

- Como consultar nuestra API en el componente principal
 - Nuestra API ya tiene una respuesta como json que es un buen formato para consumir y mostrar sus resultados en react, de esa forma tienes tu backend con cierta info, la consulto a la BD, entregas una respuesta en json y lo puedes leer en react, ya sea con fetch o con axios
 - Como integrar tecnologías de backend con el frontend y crear todo el proyecto
-
- Como consumir los registros de la BD dentro de nuestra app de react
 - Voy a pasar los datos desde App.js hacia los demás componentes
-
- En App.js:
 - import React, {**useEffect**, **useState**} from 'react';
 - // importamos nuestro cliente de axios
 - import clienteAxios from './config/axios';
 - *function* App() {
 - - // colocamos State de la app
 - *const* [citas, guardarCitas]= *useState*([]) // todas las citas van a guardarse como un arreglo y así podemos utilizar alguno de los metodos de arreglo para revisar si E una cita al menos
 -
 - *useEffect*(() => {
 - //console.log('desde *useEffect*'); // sin llamar función se ejecuta automáticamente cuando la interface de react realice algún cambio, o cuando cargue por primera vez (en consola), es un buen lugar para consumir una API externa
 - // haciendo consulta desde *UseEffect*:
 - *const* consultarAPI=() => {
 - // usamos cliente Axios que importa url base
 - clienteAxios.get('/pacientes')// que equivale a localhost:4000/pacientes
 - //Axios es cliente de promises |
 - .then(*respuesta* =>{
 - console.log(*respuesta*)
 - })
 - .catch(*error* =>{
 - console.log(*error*)
 - })
 - }
 - consultarAPI();
 - }, []);
 - // --dependencias alguna pieza de state que si cambia se vuelve a ejecutar nuestro *useEffect*
 - Como estamos en react podemos ver los resultados desde la consola, tenemos error que dice que estamos tratando de consumir una API que no soporta las consultas y fue bloqueado por cors
 - Tenemos que habilitar cors para que los recursos puedan ser intercambiados de un servidor a otro
 - Muchas de nuestras APIs que hemos utilizado, pidieron un API Key , esa es una forma de hacer que los recursos sean compartidos por medio de esa API Key

419. Habilitando CORS

- Vamos a mantenerlo mas simple y Solo vamos a habilitar cors y algunas otras medidas
- Como habilitar cors en nuestro proyecto en el servidor de node, por lo tanto en visual abrimos terminal de API e instalamos cors
 - `npm install cors`
- Abro API > `index.js` de raiz
- Ya tenemos una respuesta, ya no marca error