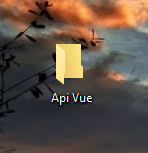
**Guía de como consumir tu api con vue.js**

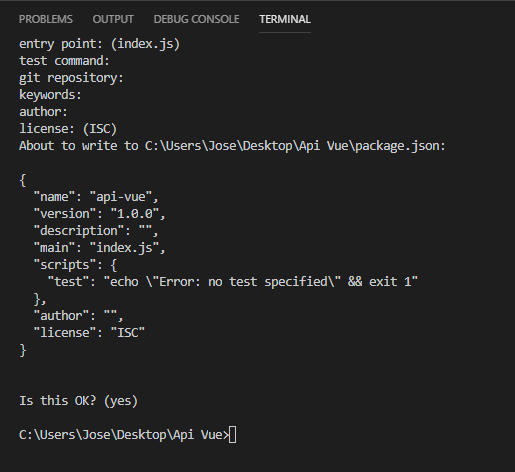
Creamos nuestra carpeta donde tendremos nuestro servidor y consumiremos nuestro api



Luego abriremos nuestra carpeta en nuestro editor de texto preferido (visual studio code)

Una vez abierto presionaremos ctrl+ñ para abrir nuestra terminar

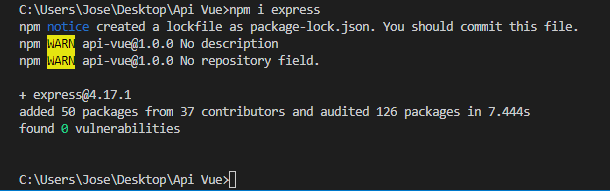
Y escribiremos el comando npm init



Daremos enter hasta terminar la instalación

Luego escribiremos el siguiente comando

Npm i express



Ahora crearemos un nuevo archivo llamado index.html y otro llamado server.js

En server.js escribimos la siguiente configuración

const express = require('express');

const server = express();

const port = 3000;//puerto

server.get("/", (req, res) => {

res.sendFile(\_\_dirname + '/index.html');//directorio que se ejecutara

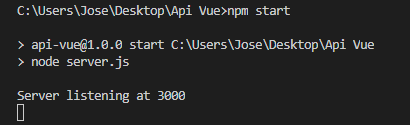
});

server.listen(port, () => {

console.log(`Server listening at ${port}`);

});

Luego con el código npm start iniciaremos nuestro servidor en el puerto que indicamos



Abriremos en nuestro navegador <http://localhost:3000> y se nos mostrara lo que hay en index.html

Ahora empezaremos a trabajar en nuestro index.html

Primero agregaremos las cdn de bootstrap desde <https://getbootstrap.com/docs/4.3/getting-started/introduction/>

Luego copiaremos el cdn de Vue.js desde <https://vuejs.org/v2/guide/> y los pegaremos en la parte baja de nuestro archivo index.html

Luego en nuestro body agregaremos un div con un id=”app”

<body class="bg-info">

<div id="app">

</div>

Y abajo agregamos un script y agregamos una url (del api) como const para utilizarla

const url = 'http://localhost/Tokens\_con\_CodeIgniter/index.php/api/Genero';

y abajo agregamos una constante de vue y instanciamos un objeto y agregamos una variable para utilizar de encabezado

const app = new Vue({

el: '#app',

data: {

Nombre: 'Api-Vue JS ',

},

})

Luego en nuestro body agregamos clases de bootstrap como un container, columnas y filas

<div id="app">

<div class="container ">

<h1 class="text-center">{{Nombre}}</h1>

<div class="row">

<div class="col s4">

</div>

<div class="col s8">

</div>

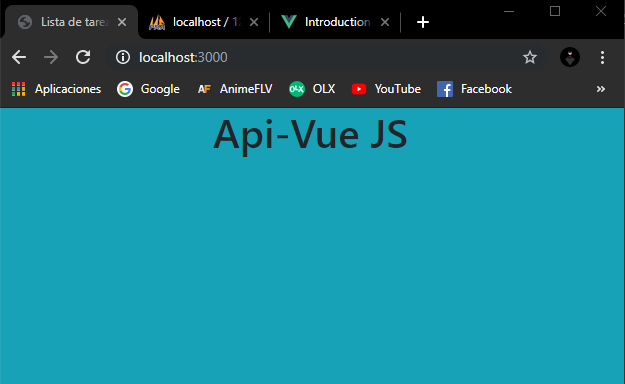
</div>

</div>

</div>

Y agregamos con las llaves dobles el nombre de nuestra variable declarada en el script

Y en nuestro navegador tendría q aparecer así:



Luego seguiremos trabajando en nuestro script y agregaremos un data

const app = new Vue({

el: '#app',

data: {

Generos: [], //array para almacenar los datos y poderlos mostrar

Nombre: 'Api-Vue JS ', //encabezador

titulo: '', //para los valores de nuestro form

id: '', //para los valores de nuestro form

Editar: false //para saber si editar o no

},

})

Luego seguiremos con los métodos y agregaremos los métodos después de la data y agregaremos un método para obtener los datos de nuestro api

methods: {

//metodo Get

//async se utiliza para trabajar de manera más cómoda con las promesas

async Get\_Generos() {

const response = await fetch(url) //La palabra clave await hace que JavaScript espere hasta que se cumpla esa promesa y devuelva su resultado.

const myJson = await response.json();

this.Generos = myJson; //se manda la informacion obtenida de nuestro api hacia el array

},

}

Luego agregamos el método mounted q sirve como su nombre lo indica para montar o cargar los datos en nuestra aplicación

//para que en el inicio de nuestra app cargen los datos o se monten

mounted() {

this.Get\_Generos()//para cargar los datos al iniciar la app

},

Ahora en una columna creada con bootstrap agregaremos una tabla

<div class="col s8">

<table class="table table-hover">

<thead class="thead-dark">

<tr>

<th>ID</th>

<th>Genero</th>

<th>Acciones</th>

</tr>

</thead>

<tbody>

<tr v-for="Genero of Generos"> <!--el v-form nos ayuda a recorrer el array -->

<td>{{Genero.id}}</td><!--el v-form nos ayuda a recorrer el array -->

<td>{{Genero.titulo}}</td>

<td class="btn-group "><!--el @click nos ayuda a escuchar eventos DOM y ejecutar JavaScript cuando se

activen -->

<button class="btn btn-danger" @click="Delete\_Generos(Genero.id)">Eliminar</button>

<button class="btn btn-warning" @click="Update\_Generos(Genero.id)">Editar</button>

</td>

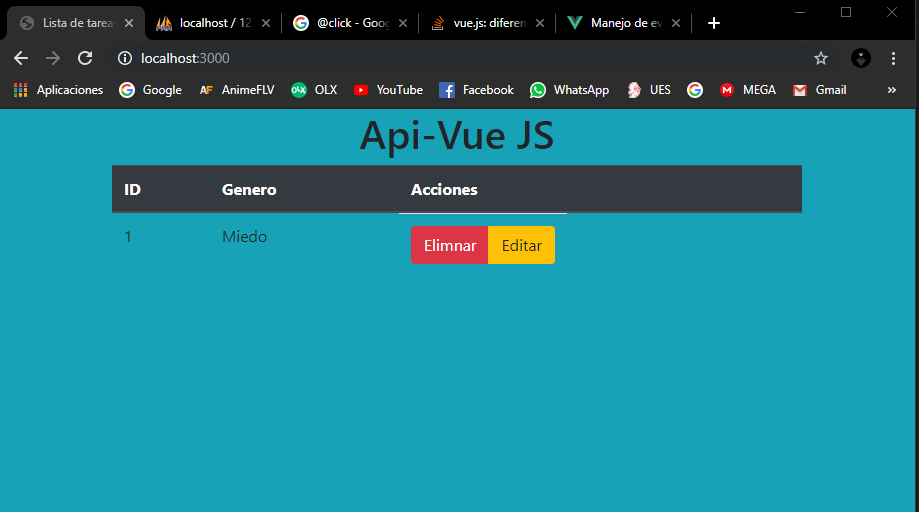
</tr>

</tbody>

</table>

</div>

Aquí nos ayudaremos del **v-for** q no ayudara a recorrer el array para mostrar los datos en nuestra tabla y usaremos **@click** para activar los métodos respectivos

Nuestra aplicación se verá así

Ahora agregaremos un formulario a nuestro body para es agregaremos otra columna

<div class="col s4">

<div class="card">

<div class="card-body">

<h5 class="card-title text-center">Agregar Genero</h5>

<div class="card-text text-center">

<form>

<div class="input-group mb-3">

<div class="input-group-prepend">

<span class="input-group-text" id="basic-addon1">Titulo</span>

</div>

<input type="number" ref="id" class="form-control" hidden><!--ref es la clave para saber el valor de el input -->

<input type="text" ref="titulo" class="form-control" placeholder="Genero" required><!--ref es la clave para saber el valor de el input -->

</div>

<div class="btn-group"><!--el @click nos ayuda a escuchar eventos DOM y ejecutar JavaScript cuando se activen -->

<button class="btn btn-success" @click="Post\_Genero()">Enviar</button>

<button type="reset" class="btn btn-danger">Reset</button>

</div>

</form>

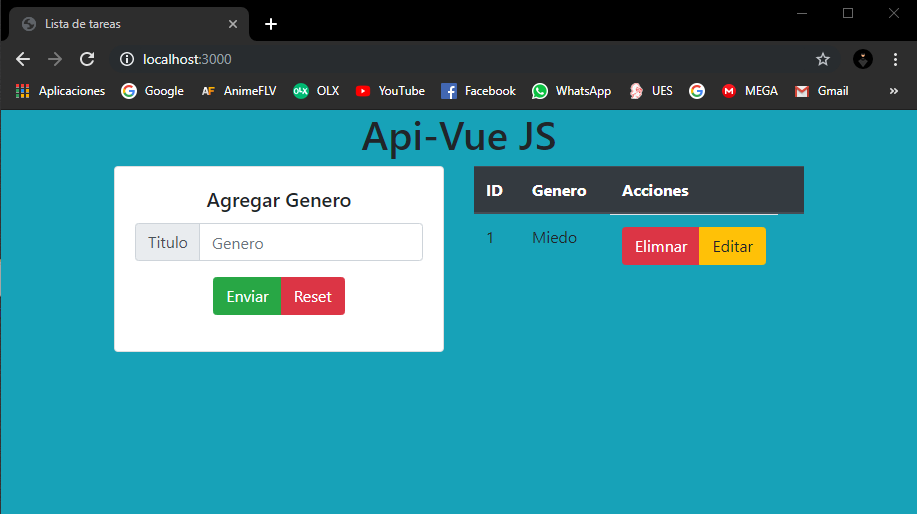
</div>

</div>

</div>

</div>

Aquí nos ayudaremos de los ref para identificar los valores de los inputs y de un @click para mandarlos datos al método post y En nuestro navegador se verá así



Ahora aremos el método Post

//metodo Post

async Post\_Genero(e) {

event.preventDefault(); //evito la recarga al enviar

//para mandar por form data a la api

const data = new FormData();

data.append('titulo', this.$refs.titulo.value); //agrego la clave valor y el valor

const response = await fetch(url, {

method: 'POST',

body: data

})

//para no recargar

this.Get\_Generos();

//this.Limpiar();

},

Primero hacemos un preventDefault para evitar que el sitio se recargue luego instanciamos un objeto Form data para mandar los datos

Agregamos los datos a nuestro objeto con append() pasando la clave valor y el valor

Luego hacemos nuestra consulta con fect y mandamos el método y el cuerpo de nuetsra peticon y por ultimo llamamos a Get\_genero() para no recargar la página y luego la función limpiar q nos ayuda a limpiar los valores de nuestro formulario

Ahora aremos el método Delete

//metodo Delete

async Delete\_Generos(id) {

const response = await fetch(url + "/" + id, {

method: 'DELETE'

});

this.Get\_Generos();

},

Mandaremos a través de la url nuestro id que obtenemos de nuestro botón eliminar en la tabla

<button class="btn btn-danger" @click="Delete\_Generos(Genero.id)">

Eliminar</button>

Ahora aremos el método Delete

//metodo GetByid

async Update\_Generos(id) {

const response = await fetch(url + "/" + id)

const myJson = await response.json();

this.$refs.titulo.value = myJson.titulo;

this.$refs.id.value = myJson.id;

this.Editar = true;

},

Mandaremos a través de la url nuestro id que obtenemos de nuestro botón editar en la tabla

<button class="btn btn-warning" @click="Update\_Generos(Genero.id)">

Editar</button>

Y los datos obtenidos los guardaremos en una variable y luego los utilizaremos para setear los values con las ref y su clave valor **this.$refs.titulo.value = myJson.titulo** y al final cambiaremos el estado de la variable editar a true **this.Editar=true** ahora podremos con el mismo formulario agregar datos y editarlos solo nos falta la función que nos permita saber si tiene que editar o agregar

Nuestro método post quedaría asi

//metodo post y put

//metodo Post

async Post\_Genero(e) {

event.preventDefault(); //evito la recarga al enviar

if (this.Editar === false) {

//para mandar por form data a la api

const data = new FormData();

data.append('titulo', this.$refs.titulo.value); //agrego la clave valor y el valor

const response = await fetch(url, {

method: 'POST',

body: data

})

//metodo Put

} else if (this.Editar === true) {

//para mandar por json a nuestra api

var data = JSON.stringify({

"id": this.$refs.id.value,

"titulo": this.$refs.titulo.value

})

console.log(data)

const response = await fetch(url, {

method: 'PUT',

body: data,

headers: {

'Content-Type': 'application/json'

},

});

}

this.Editar = false;

//para no recargar

this.Get\_Generos();

//this.Limpiar();

},

Agregaremos un condicionante **if** que nos evalué el estado de nuestra variable editar si es **true o false.**

Si es **false** nuestra app ara un post que ya hicimos en el ejercicio, pero si es **true** entrera en el método para editar que lo aremos mandando los datos como **JSON**

} else if (this.Editar === true) {

//para mandar por json a nuestra api

var data = JSON.stringify({"id": this.$refs.id.value,"titulo": this.$refs.titulo.value})//aqui convertimos un objeto a una cadena JSON

const response = await fetch(url, {

method: 'PUT',

body: data,

headers: {

'Content-Type': 'application/json'

},

});

}

Al final ejecutaremos **this.Get\_Generos()** para no recargar **y this.Limpiar**  para limpiar los imput y además de eso devolveremos el estado de Editar a false **this.Editar= false.**

Por ultimo agregaremos el método limpiar para setear los valores de los campos a vacios

**FIN.**