**Implementación de modelos con Express Cross-Origin Resource Sharing (CORS)**

En este ejercicio implementaremos CORS a nuestro servidor *CORS* es un tema de seguridad que trata características relativamente avanzadas, así que no te preocupes si no entiendes completamente su funcionamiento en la primera vez que leas esto. ¿Pero qué es *CORS?*

*Cross-Origin Resource Sharing (CORS)* es un mecanismo que implementa *Headers* adicionales a las peticiones *HTTP* para permitir a un cliente (como un navegador web) ganar permisos para acceder a ciertos recursos de un servidor de diferente origen (dominio) que en que se encuentra actualmente.

Por ejemplo, supongamos que ya leímos los ejercicios de Angular de esta guía y queremos conectarnos desde Angular a la API que hemos estado desarrollando en estos últimos ejercicios de Node con *Express*. Básicamente tendremos dos servidores instalados y corriendo en nuestra computadora, el servidor de Angular en la URL “*127.0.0.1:4200*” que se ejecuta con ***ng serve*** y el servidor de Node.js en la URL “*127.0.0.1:3000*” el cual se ejecuta con ***nodemon app***.

Ahora supongamos que queremos utilizar nuestro navegador web para acceder al servidor de *Angular* con la URL “*127.0.0.1:4200*”. Al haber hecho esto, nuestro navegador se encontrará en el servidor de *Angular*. Y si seguiste la sección de *Angular* de esta guía, te habrás dado cuenta de que, en cierto punto, se empezó a utilizar la “*API de pendientes*” para acceder a los distintos *endpoints* que creamos y consumirlos desde *Angular.* Pero al estar en la URL “*127.0.0.1:4200*” el origen (dominio) actual es el del servidor local de *Angular*. Así que si tú desde tu navegador web accedes al servidor de *Angular* estarás en ese origen, pero como en lo servicios de *Angular* hay código que intenta conectarse a nuestra API, la cual está en otro servidor local con el origen (dominio) de “*127.0.0.1:3000*” habrá conflictos a la hora de intentar compartir los recursos entre *Angular* y nuestra *API de Express.*

Lo sabemos, *CORS* puede llegar a ser confuso y enredoso, así que no te preocupes, sólo ten en cuenta que para poder conectar a *Angular* con nuestra *API* de *Express* tenemos que habilitar *CORS* desde el código de nuestra API.

Si quiere saber más sobre cómo funciona *CORS*, te invitamos a leer el siguiente artículo:

<https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS>

Ahora tenemos que activar *CORS* en nuestra API y el proceso de activación es realmente sencillo, así que veamos el código final una vez que se implementa *CORS*

Primero tendremos que instalar un paquete llamado *cors* (muy creativo el nombre), y eso lo haremos ejecutando el siguiente comando:

“***npm install cors@2.8.4 --save***”

Y lo implementaremos en el archivo *app.js*

*app.js*

"use strict";

**const** express = require('express');

**const** app = express();

**const** bodyParser = require('body-parser');

app.use(bodyParser.json());

**const** cors = require('cors');

app.use(cors());

**const** env = require('dotenv');

env.config();

**const** pendiente = require('./pendiente/pendiente.routes');

app.use('/pendiente', pendiente);

**const** portExpress = process.env.EXPRESS\_PORT;

**const** hostExpress = process.env.EXPRESS\_HOST;

app.listen(portExpress, hostExpress, () **=>** {

console.log(`¡Escuchando en ${hostExpress}:${portExpress}/`);

});

Como podemos ver, la implementación de *CORS* en *Express* tan sólo require de dos líneas de código:

**const** cors = require('cors');

app.use(cors());

Primero requerimos el paquete que recién acabamos de instalar y guardamos una referencia a él en la constante cors. Después utilizamos “app.use(cors())” para añadir a “cors()” como *middleware* en nuestra API de *pendientes*. Y eso es todo, hemos terminado. Sencillo, ¿no?

Obviamente la configuración por defecto del paquete *cors* de Node.js es una configuración estándar y puede que quieras personalizarla para que cumpla con tus necesidades, sin embargo, por fines prácticos sólo implementaremos la configuración estándar. Si te interesa saber cómo configurar más a fondo *cors,* visita la página de GitHub oficial del paquete en:

<https://github.com/expressjs/cors>

Por último, aunque los archivos restantes no requieren ningún cambio, queremos agregarlos para que los tengas como referencia:

*.env*

**#Las líneas que empiezan con un '#', son tratadas**

**#como comentario y por lo tanto, son ignoradas.**

**#NO SE RECOMIENDA SUBIR EL ARCHIVO .env A REPOSITORIOS**

**#DE GIT, PERO POR CUESTIONES PRÁCTICAS, LO SUBIREMOS ESTA VEZ**

**#Configuración de MySQL**

**DB\_HOST=127.0.0.1**

**DB\_USER=root**

**DB\_PASSWORD=qwerty1234**

**DB\_PORT=3306**

**DB\_DATABASE=db\_pendientes**

**#Configuración de nuestro servidor**

**EXPRESS\_PORT=3000**

**EXPRESS\_HOST=127.0.0.1**

*pendiente/pendiente.model.js*

**const** conexion = require('../db-config/mysql-connection');

**let** Pendiente = { };

Pendiente.obtenerTodos = ( res, cb ) **=>** {

if ( conexion ) {

conexion.query('SELECT pendiente\_id, descripcion, estado FROM pendientes', (error, resultados) **=>** {

if (error)

return cb( error, res );

return cb( null, res, resultados );

})

} else

return cb('Hubo un error con la conexión a MySQL :(', res);

}

Pendiente.contador = ( res, cb ) **=>** {

if ( conexion ) {

conexion.query('SELECT COUNT (pendiente\_id) AS numero\_de\_registros FROM pendientes', (error, contador) **=>** {

if ( error )

return cb( error, res );

return cb( null, res, contador );

})

} else

return cb('Hubo un error con la conexión a MySQL :(', res);

}

Pendiente.siguienteId = ( res, cb ) **=>** {

if ( conexion ) {

conexion.query(`SELECT \`AUTO\_INCREMENT\` AS siguiente\_id\_autoincrementable

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_SCHEMA = '${process.env.DB\_DATABASE}'

AND TABLE\_NAME = 'pendientes';`, (error, siguienteId) **=>** {

if (error)

return cb( error, res );

return cb( null, res, siguienteId );

})

} else

return cb('Hubo con error con la conexión a MySQL :(', res);

}

Pendiente.encontrarPorId = ( idPendiente, res, cb ) **=>** {

if ( conexion ) {

conexion.query(`SELECT pendiente\_id, descripcion, estado

FROM pendientes

WHERE pendiente\_id = ?`, [idPendiente], (error, pendiente) **=>** {

if (error)

return cb( error, res );

return cb( null, res, pendiente );

})

} else

return cb('Hubo con error con la conexión a MySQL :(', res);

}

Pendiente.insertar = ( nuevoPendiente, res, cb ) **=>** {

if ( conexion ) {

conexion.query('INSERT INTO pendientes SET ?', [nuevoPendiente], (error, respuesta) **=>** {

if ( error )

return cb( error, res );

return cb( null, res, respuesta )

});

} else

return cb('Hubo con error con la conexión a MySQL :(', res);

}

Pendiente.actualizar = ( pendientePorEditar, res, cb ) **=>** {

if ( conexion ) {

conexion.query(`UPDATE pendientes

SET descripcion = ?, estado = ?

WHERE pendiente\_id = ?`,

[pendientePorEditar.descripcion,

pendientePorEditar.estado,

pendientePorEditar.pendiente\_id], (error, respuesta) **=>** {

if ( error )

return cb( error, res );

return cb( null, res, respuesta )

});

} else

return cb('Hubo con error con la conexión a MySQL :(', res)

}

Pendiente.eliminar = ( idPendiente, res, cb ) **=>** {

if ( conexion ) {

conexion.query(`DELETE FROM pendientes

WHERE pendiente\_id = ?`, [idPendiente], (error, respuesta) **=>** {

*//Si hay un error, le respondemos al cliente con el error.*

if (error)

return cb( error, res );

return cb( null, res, respuesta );

})

} else

return cb('Hubo un error con la conexión a MySQL :(', res);

}

Pendiente.responderAlCliente = ( error, res, datos ) **=>** {

if ( error )

res.status(500).json(error);

else

res.status(200).json(datos);

}

module.exports = Pendiente;

*db-config/mysql-connection.js*

**const** mysql = require('mysql');

**const** conexion = mysql.createConnection({

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

port: process.env.DB\_PORT,

database: process.env.DB\_DATABASE

});

conexion.connect(err **=>** {

if (err) {

console.log('Error trying to connect with Data Base: ' + err.stack);

throw err;

}

console.log("Conexión exitosa con la base de datos c:")

});

module.exports = conexion;

*pendiente/pendientes.routes.js*

**const** express = require('express');

**const** router = express.Router();

*//Importamos el modelo de Pendiente.*

**const** Pendiente = require('./pendiente.model');

*//Consulta general de todos los pendientes en la base de datos.*

router

.get('/', (req, res) **=>** {

return Pendiente.obtenerTodos( res, Pendiente.responderAlCliente );

})

*//Consultar cuando registros hay en la tabla de pendientes.*

.get('/count', (req, res) **=>** {

return Pendiente.contador( res, Pendiente.responderAlCliente );

})

*//Consultar cuando registros hay en la tabla de pendientes.*

.get('/siguienteIdAutoIncrementable', (req, res) **=>** {

return Pendiente.siguienteId( res, Pendiente.responderAlCliente );

})

*//Consultar un pendiente por id.*

.get('/:idPendiente', (req, res) **=>** {

**const** idPendiente = req.params.idPendiente;

return Pendiente.encontrarPorId( idPendiente, res, Pendiente.responderAlCliente );

})

*//Agregar un nuevo pendiente a la base de datos.*

.post('/', (req, res) **=>** {

**const** nuevoPendiente = {

pendiente\_id: null,

descripcion: req.body.descripcion,

estado: req.body.estado

}

return Pendiente.insertar( nuevoPendiente, res, Pendiente.responderAlCliente );

})

*//Modifica un pendiente existente en la base de datos.*

.put('/:idPendiente', (req, res) **=>** {

**const** idPendiente = req.params.idPendiente;

**const** pendientePorEditar = {

pendiente\_id: idPendiente,

descripcion: req.body.descripcion,

estado: req.body.estado

}

return Pendiente.actualizar( pendientePorEditar, res, Pendiente.responderAlCliente );

})

*//Elimina un pendiente existente en la base de datos.*

.delete('/:idPendiente', (req, res) **=>** {

**const** idPendiente = req.params.idPendiente;

return Pendiente.eliminar( idPendiente, res, Pendiente.responderAlCliente );

})

module.exports = router;