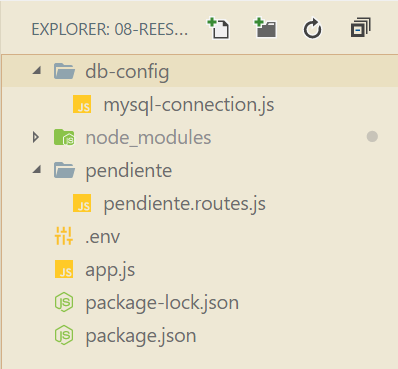
**Restructurar rutas y conexión a MySQL**

Después de haber incluido MySQL y variables de entorno a nuestro proyecto, ya viene siendo hora de estructurar nuestra aplicación. A lo largo de los anteriores ejercicios, siempre hemos tenido toda la lógica en el archivo ***app.js***, de hecho, es el único archivo de JavaScript con el que hemos trabajado. Sin embargo, esto sólo es viable con proyecto muy pequeños. Si tuviéramos un proyecto muy grande con miles y miles de líneas de código, podemos estar seguros de que tener toda la lógica en un solo archivo no es la mejor idea.

Por lo tanto, debemos ver cuál es la estructura que seguiremos a la hora de reestructurar nuestra aplicación. Hay muchas formas de estructurar un proyecto, pero la que más recomendamos es la siguiente:



Bien, empezaremos por el archivo *app.js,* el cual ha sufrido cambios importantes en el buen sentido:

*app.js*

"use strict";

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

**const** app = express();

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

app.use(bodyParser.json());

**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 apreciar, el contenido de nuestro archivo *app.js*, se ha reducido muchísimo. Removimos toda la lógica de nuestras rutas y la conexión de la base de datos, y lo único que añadimos después de eso fue importar el archivo pendiente.routes.js, el cual explicaremos un poco más adelante.

¿Pero cuál es la diferencia entre requerir archivos personalizados y requerir módulos de nuestra carpeta *node\_modules?* La diferencia está en el parámetro que le pasamos a require('');

Por ejemplo, cuando requerimos un módulo que está en la carpeta de *node\_modules* o que viene integrado por defecto en Node.js, lo hacemos de la siguiente manera (como con *Express*):

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

Si observamos la *string* que tiene como parámetro “require('express');”, es tan sólo el nombre de *express*. Al importar un módulo de esta manera, lo que *require* hará es buscar el módulo en los módulos de núcleo Node.js, y si no lo encuentra, procederá a buscar el módulo en la carpeta de *node\_modules.*

Pero en el caso de requerir módulos nuestros, la sintaxis cambia ligeramente:

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

Al requerir un módulo nuestro, se lo especificamos a *require* con un “./” antes de indicar el *path* o directorio de nuestro archivo “*.js”.* En nuestro caso, como tenemos un archivo llamado pendiente.routes en una carpeta llamada pendiente, le especificamos el directorio relativo que tiene que seguir para llegar al archivo pendiente.routes desde el archivo actual (*app.js*)*.*

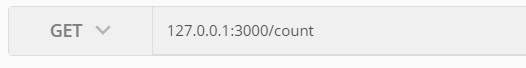
También habrás notado que al importar el archivo pendiente.routes.js, no incluimos la extensión ***.js*** en el *require*. Y esto es debido a que *require* implícitamente agrega la extensión de ***.js*** al leer el archivo.

Después de requerir nuestro archivo pendiente.routes, llegamos a la siguiente línea:

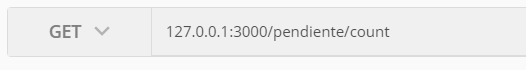
app.use('/pendiente', pendiente);

Esta línea le indica a la instancia de *Express* que todas las rutas que establezcamos en el archivo pendiente.routes, tendrán que ser accedidas con '/pendiente'.

Por ejemplo, anteriormente accedíamos a la ruta /count con el siguiente formato:



Pero de ahora en adelante, accederemos a la ruta con /count con '/pendiente' antepuesta, es decir:



Esto se hace con el fin de separar la lógica de cada módulo que agreguemos a nuestra aplicación. De esta manera, si creáramos más rutas, por ejemplo, para *contactos*, en lugar de tener todas nuestras rutas combinadas, las tendríamos separadas por módulo.

Es decir, tendríamos una ruta de *count* para *pendientes:* /pendiente/count

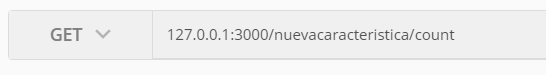
Y tendríamos una ruta de *count* para *contactos:* /contacto/count

Y como puedes ver, de esta manera podemos modularizar y aumentar la escalabilidad de nuestra aplicación. Así que, si deseas poner rutas para una nueva característica, lo que tendrías que hacer, es poner sus rutas en un archivo llamado ***nuevacaracteristica.routes.js*** dentro de una carpeta con el nombre de ***nuevacaracteristica*** y finalmente, tendrías que importar las rutas en el archivo *app.js* de la siguiente manera:

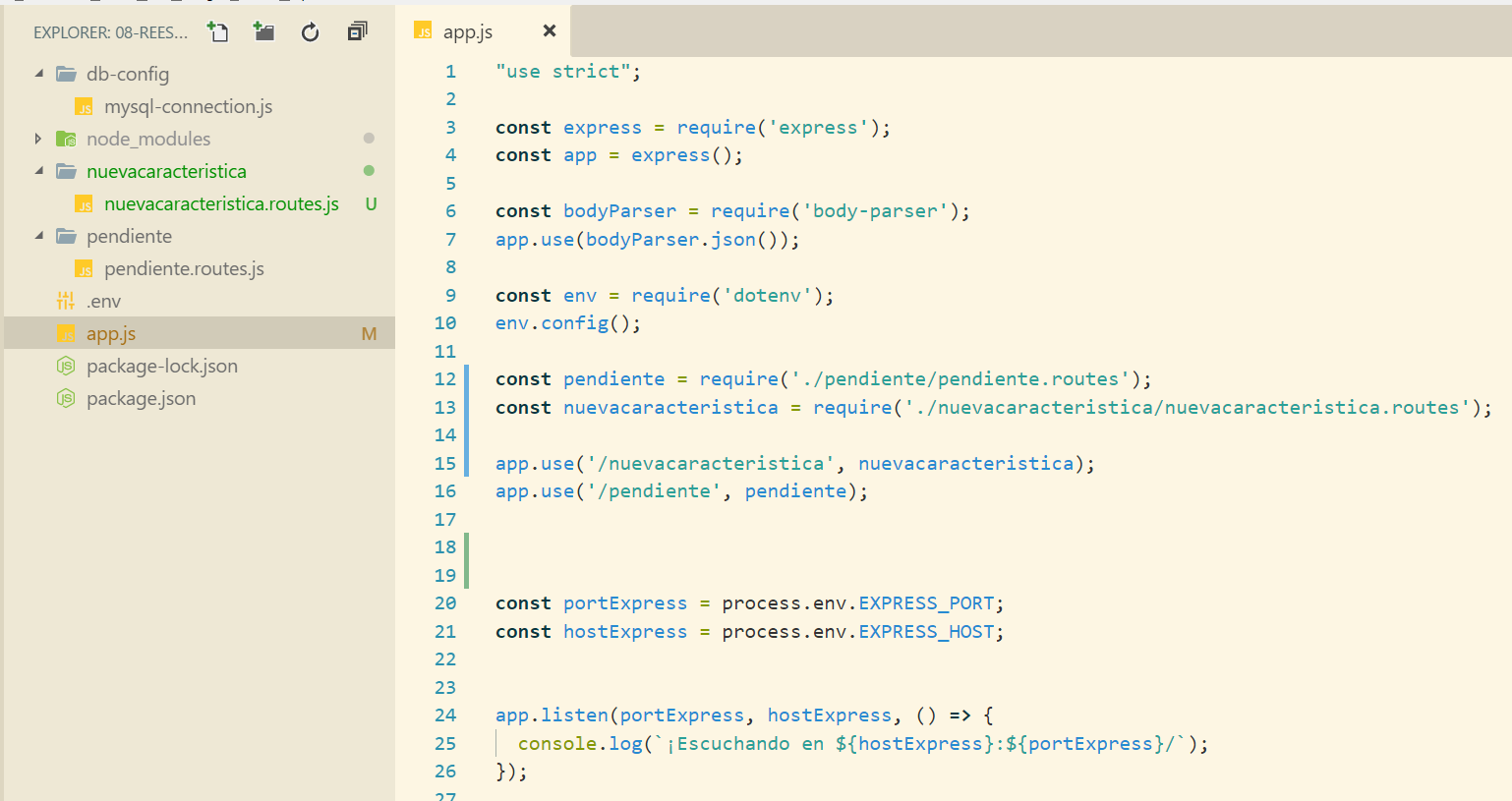
**const** nuevacaracteristica = require('./nuevacaracteristica/nuevacaracteristica.routes');

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

Accederías a las rutas de la siguiente manera:



Y la estructura de tu aplicación sería la siguiente:



Y antes de que lo olvidemos, el archivo *“.env”* contiene:

*.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**

Buen, después de haber explicado cómo funciona la importación de nuestros propios módulos, es hora explicar cómo exportamos código desde nuestro archivo ***pendiente.routes.js***

*pendiente/pendiente.routes.js*

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

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

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

**function** responderAlCliente( error, res, datos ) {

if ( error )

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

else

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

}

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

router

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

if ( conexion ) {

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

if (error)

return responderAlCliente( error, res );

return responderAlCliente( null, res, resultados );

})

} else

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

})

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

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

if ( conexion ) {

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

if (error){

return responderAlCliente( error, res );

}

return responderAlCliente( null, res, contador );

})

} else

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

})

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

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

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 responderAlCliente( error, res );

return responderAlCliente( null, res, siguienteId );

})

} else

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

})

*//Consultar un pendiente por id.*

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

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

if ( conexion ) {

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

FROM pendientes

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

if (error)

return responderAlCliente( error, res );

return responderAlCliente( null, res, pendiente );

})

} else

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

})

*//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

}

if ( conexion ) {

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

if ( error )

return responderAlCliente( error, res );

return responderAlCliente( null, res, respuesta )

});

} else

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

})

*//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

}

if ( conexion ) {

conexion.query(`UPDATE pendientes

SET descripcion = ?, estado = ?

WHERE pendiente\_id = ?`,

[pendientePorEditar.descripcion,

pendientePorEditar.estado,

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

if ( error )

return responderAlCliente( error, res );

return responderAlCliente( null, res, respuesta )

});

} else

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

})

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

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

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

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 responderAlCliente( error, res );

return responderAlCliente( null, res, respuesta );

})

} else

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

})

module.exports = router;

Ahora vemos que todas las rutas que teníamos establecidas anteriormente en el archivo *app.js* están en este archivo dedicado a las rutas. ¿Pero qué cambio?

Lo único que cambias de las rutas fue que removimos algunos comentarios para ahorrar espacio. Lo que sí cambio y realmente es importante, es lo siguiente:

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

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

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

Al inicio del archivo ***pendiente.routes.js*** importamos *express*, pero en lugar de instanciarlo en una variables *app* como lo hemos hecho en los anteriores ejercicios, en este caso solamente requerimos de la funcionalidad de “express.Router();”. Dicha funcionalidad la guardaremos en una constante llamada “router”, esta constante nos permitirá crear un *middleware* exclusivo para las rutas de *pendientes* y en lugar de “*app.get(…)”* o “*app.post(…)”,* lo que necesitaremos hacer es “*router.get(…)”* , “*router.post(…)*”, etcétera:

router

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

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

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

*//Etcétera.*

Pero nos falta explicar la línea clave para exportar nuestro *router,* y esa línea es:

module.exports = router;

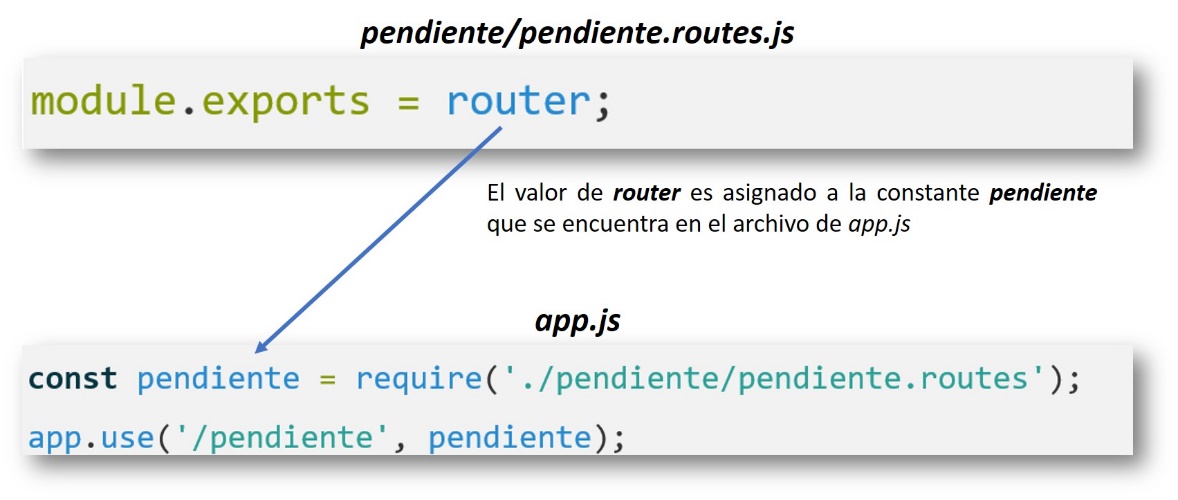
Con esta línea, le indicamos a Node.js que queremos que la constante router pueda ser usada fuera del archivo ***pendiente.routes.js***. Si no especificáramos que queremos exportar router con “module.exports”, cuando importemos este archivo desde ***app.js***, no podríamos usar las rutas de *pendientes*. Así que básicamente, si queremos compartir alguna característica de uno de nuestros módulos, tenemos que especificarlo con “module.exports”.

Ahora, regresemos un poco a la manera en la que importamos y usamos el archivo ***pendiente.routes.js*** desde ***app.js***:

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

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

Al importar o requerir “require('./pendiente/pendiente.routes');” y asignarlo a “pendiente”, lo que estamos haciendo es importar el router que exportamos en el archivo de ***pendiente.routes.js***, es decir:



Y por lo tanto, en el archivo ***app.js*,** podemos indicarle a la instancia de *Express (app)* que use como *middleware* las rutas que exportamos en el router con “module.exports = router;”, y esto se lo indicamos con:

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

Por último, nos falta explicar cómo importamos (requerimos) la conexión de MySQL al principio del archivo de ***pendiente.routes.js***:

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

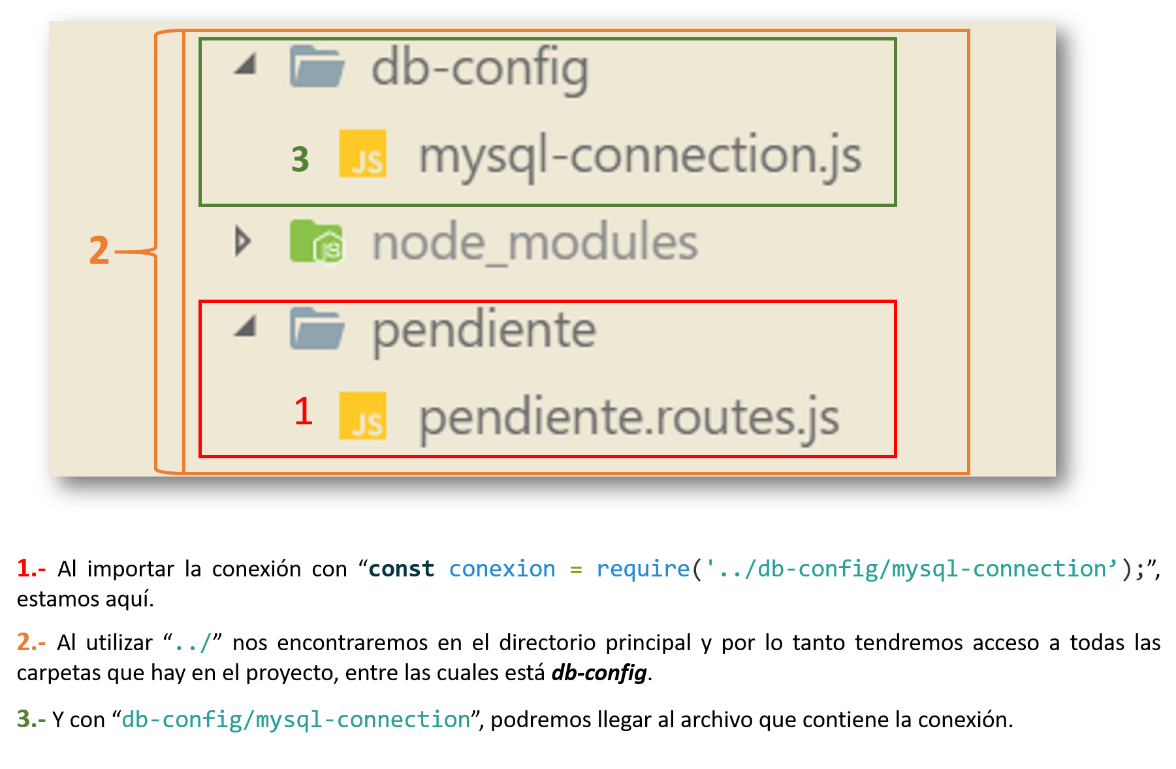
Y básicamente es lo mismo que cuando requerimos ***pendiente.routes.js*** desde el ***app.js***. Simplemente debemos de especificar la ruta relativa en la que se encuentre el archivo que contenga la conexión. En este caso, nuestro el archivo en el que estamos es:

/pendiente/pendiente.routes

Y el archivo que queremos importar se encuentra en:

/db-config/mysql-connection

Por lo que para importar la conexión tendremos que retroceder una carpeta con “../”. Lo que significaría que ahora estaríamos en el directorio principal de nuestro proyecto, es decir “/”, así que para llegar a la conexión solamente nos faltaría “db-config/mysql-connection”. Puede sonar un poco confuso, así que miremos la siguiente imagen:



Por último, sólo nos falta ver que hay en el archivo ***mysql-connection.js***:

*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:")

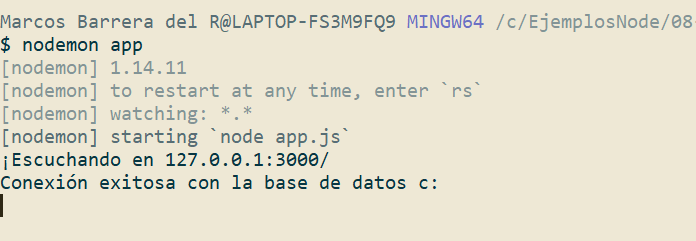
});

module.exports = conexion;

Y como podrás ver, la conexión a MySQL sigue siendo la misma, lo único nuevo que hacemos es exportar la conexión para que puede ser utilizada en cualquier otra parte del proyecto:

module.exports = conexion;

Y al correr el proyecto con ***nodemon app***, todo sigue funcionando correctamente:



La única diferencia es que ahora nuestras peticiones para *pendientes* tendrán que ir dirigidas a “***127.0.0.1:3000/pendiente/***”

