Sistemas Distribuidos e Internet
Desarrollo de aplicaciones Web con NodeJS
Sesión - 9
Curso 2017/ 2018

Introducción

En esta sesión de prácticas se finalizarán las operaciones restantes relativas a la gestión de canciones y se implementará un sistema de paginación.

Eliminar canciones

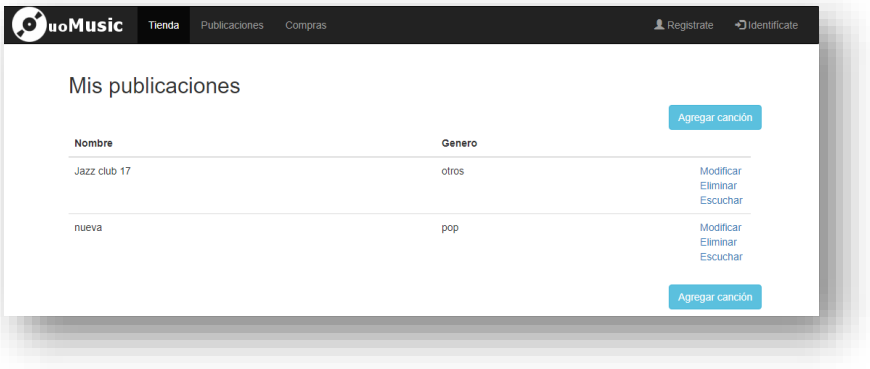
Implementamos la función **GET /cancion/eliminar/:id** en el controlador **rcanciones.js**. Obtendremos la id de la canción a eliminar e invocaremos a **gestorBD.eliminarCancion().**

```
app.get('/cancion/eliminar/:id', function (req, res) {
	var criterio = { "_id" : gestorBD.mongo.ObjectID(req.params.id)  };

	gestorBD.eliminarCancion(criterio,function(canciones){
		if ( canciones == null ){
			res.send(respuesta);
		} else {
			res.redirect("/publicaciones");
		}
	});
})
```

Implementamos la función **eliminarCanciones** en el **gestorBD.js**.

```
module.exports = {
	mongo : null,
	app : null,
	init : function(app, mongo) {
		this.mongo = mongo;
		this.app = app;
	},
	eliminarCancion : function(criterio, funcionCallback) {
		this.mongo.MongoClient.connect(this.app.get('db'), function(err, db) {
			if (err) {
				funcionCallback(null);
			} else {
				var collection = db.collection('canciones');
				collection.remove(criterio, function(err, result) {
					if (err) {
						funcionCallback(null);
					} else {
						funcionCallback(result);
					}
					db.close();
				});
			}
		});
	},
```

Sí ejecutamos la aplicación podemos observar que las canciones se eliminan correctamente.



No obstante, tenemos un problema de seguridad. Solo debería tener a acceso a **GET / POST /cancion/modificar/:id** y **GET /cancion/eliminar** un usuario identificado y que además sea el autor de la canción.

Abrimos el fichero principal **app.js,** vamos a añadir un nuevo Router **routerUsuarioAutor** . Este router obtiene el parámetro id de la URL y busca a que canción corresponde, después verifica si es el usuario que hay en sesión es su autor.

Aplicamos este router a **/cancion/modificar** y **/cancion/eliminar .** Para obtener el parámetro id de la URL no podemos utilizar **req.params.id** , no funciona dentro del router si los parámetros están en la URL sin clave asociada (si que funcionaria para un parámetro de otro tipo **?id=valor**)

```
// routerUsuarioSession
var routerUsuarioSession = express.Router();
routerUsuarioSession.use(function(req, res, next) {
         console.log("routerUsuarioSession");
          if ( req.session.usuario ) {
            // dejamos correr la petición
             next();
          } else {
             req.session.destino = req.originalUrl;
             console.log("va a : "+req.session.destino)
             res.redirect("/identificarse");
          }
});

//Aplicar routerUsuarioSession
app.use("/canciones/agregar",routerUsuarioSession);
app.use("/publicaciones",routerUsuarioSession);

//routerUsuarioAutor
var routerUsuarioAutor = express.Router();
routerUsuarioAutor.use(function(req, res, next) {
         console.log("routerUsuarioAutor");
         var path = require('path');
         var id = path.basename(req.originalUrl);
         // Cuidado porque req.params no funciona
         // en el router si los params van en la URL.

         gestorBD.obtenerCanciones(
                         { _id : mongo.ObjectID(id) }, function (canciones) {
                 console.log(canciones[0]);
                 if(canciones[0].autor == req.session.usuario ){
                         next();
                 } else {
                    res.redirect("/tienda");
                 }
```

```
        })

});

//Aplicar routerUsuarioAutor
app.use("/cancion/modificar",routerUsuarioAutor);
app.use("/cancion/eliminar",routerUsuarioAutor);
```

Con esta comprobación solo el propio autor podrá **modificar** y **eliminar** canciones. Otra opción también podría ser hacer las comprobaciones de autoría en el propio modificar y eliminar, pero usar un router puede simplificar el desarrollo y mejorar la arquitectura.

Algunas aplicaciones definen paths específicos en las URLS para agruparlas en función del nivel de autorización requerido, por ejemplo
/usr/canciones/agregar
/usr/aut/cancion/modificar
Solo es necesario ser un usuario identificado **/usr/** para agregar una canción, sin embargo, hay que ser el autor para modificarla. Utilizando esta técnica se simplifica el uso del enrutador ya que cada uno se aplica a un conjunto de subURLs, por ejemplo

```
app.use("/usr/aut/",routerUsuarioAutor);
```

Redirecciones

Por el momento no tenemos especificadas demasiadas redirecciones, hay un gran número de redirecciones que podríamos aplicar para mejorar la navegabilidad.

Por ejemplo, en **app.js** podríamos redireccionar la URL principal **GET /** a la **/tienda .** Como se trata de la página de inicio es buena idea especificarlo en el propio **app.js.**

```
app.get('/', function (req, res) {
        res.redirect('/tienda');
})

// lanzar el servidor
app.listen(app.get('port'), function() {
        console.log("Servidor activo");
});
```

Después de identificarse **POST /identificarse** (controlador **rusuarios.js**) sería buena idea re direccionar a **/publicaciones** .

```
app.post("/identificarse", function(req, res) {
        var seguro = app.get("crypto").createHmac('sha256', app.get('clave'))
                        .update(req.body.password).digest('hex');

        var criterio = {
                email : req.body.email,
                password : seguro
        }

        gestorBD.obtenerUsuarios(criterio, function(usuarios) {
                if (usuarios == null || usuarios.length == 0) {
                        req.session.usuario = null;
```

```
                res.send("No identificado: ");
            } else {
                req.session.usuario = usuarios[0].email;
                res.send("identificado");
                res.redirect("/publicaciones");
            }

        });
});
```

COMPLETAR: Todavía hay algunas redirecciones más que se pueden incluir, por ejemplo:

- Cuando se termina de agregar una canción desde **Post /cancion** ir a **/publicaciones**
- Cando se termina de modificar una canción **Post /cancion/modificar** ir a **/publicaciones**
- Cuando se termina de registrar un nuevo usuario **Post /usuario** ir a **/identificarse**

Mensajes de información y alertas

Debemos intentar eliminar las respuestas de texto simple de la aplicación **res.send("texto") ,** no obstante en muchos casos nos interesa enviar mensajes claros al usuario, por ejemplo:

- **/identificarse –** el usuario introduce sus datos incorrectamente.
- **/cancion/eliminar –** se ha eliminar la canción con éxito
- **/cancion/modificar –** se ha modificado la canción con éxito
- En muchos otros casos, para indicar el resultado de acciones o estado de la aplicación

Como este tipo de mensajes van a ser comunes a toda la aplicación los vamos a incluir en la plantilla de las vistas **base.html.** Incluimos un pequeño script jQuery que obtenga los parámetros **mensaje** y **tipoMensaje** de la URL.

En caso de que cualquier URL contenga esos parámetros (**mensaje** y **tipoMensaje**) vamos a incluir un alert https://getbootstrap.com/docs/3.3/components/#alerts en el div principal del contenido ** **

La variable **tipoMensaje** será opcional (si no recibe ninguna mostrará mensajes de alert-info) , así será capaz de mostrar mensajes de diferentes tipos **alert-success, alert-info, alert-danger , etc.** Modificamos **/views/base.html :**

```
<div class="container">
        <script>
                var mensaje = getUrlParameter('mensaje');
                var tipoMensaje = getUrlParameter('tipoMensaje');

                if ( mensaje != ""){
                        if (tipoMensaje == "" ){
                                tipoMensaje = 'alert-info';
                        }
                        $( ".container" )
                                .append("<div class='alert "+tipoMensaje+"'>"+mensaje+" </div>");
                }

                function getUrlParameter(name) {
                    name = name.replace(/[\[]/, '\\[').replace(/[\]]/, '\\]');
                    var regex = new RegExp('[\\?&]' + name + '=([^&#]*)');
                    var results = regex.exec(location.search);
                    return results === null ? '' :
                        decodeURIComponent(results[1].replace(/\+/g, ' '));
```

```
                };
        </script>

        <!-- Contenido -->
        {% block contenido_principal %}
        <!-- Posible contenido por defecto -->
        {% endblock %}

</div>
```

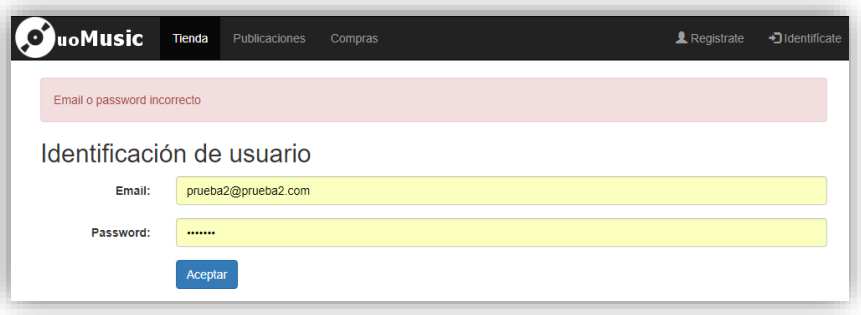
Nota: Este código JavaScript no tiene nada que ver con el del resto de la aplicación **Node**, se **ejecuta en el Cliente** (navegador del usuario).

El objeto **URLSearchParams** de Javascript permite obtener los parámetros Get, pero lamentablemente no funciona de forma correcta en algunos navegadores.
En este caso optamos por implementar "manualmente" la función **getUrlParameter(name).**

Abrimos el controlador **rusuarios.js** y modificamos la respuesta que se envían desde **Post /identificarse** cuando las credenciales de los usuarios no son correctas.

```
app.post("/identificarse", function(req, res) {
        var seguro = app.get("crypto").createHmac('sha256', app.get('clave'))
                        .update(req.body.password).digest('hex');

        var criterio = {
                email : req.body.email,
                password : seguro
        }

        gestorBD.obtenerUsuarios(criterio, function(usuarios) {
                if (usuarios == null || usuarios.length == 0) {
                        req.session.usuario = null;
                        res.send("No identificado: ");
                        res.redirect("/identificarse" +
                                                "?mensaje=Email o password incorrecto"+
                                                "&tipoMensaje=alert-danger ");
                } else {
                        req.session.usuario = usuarios[0].email;
                        res.redirect("/publicaciones");
                }

        });
});
```

Guardamos los cambios y ejecutamos la aplicación, comprobamos que el mensaje se muestra.



Siguiendo este mismo enfoque sustituimos los mensaje planos por redirecciones a URLS + mensajes . Modificamos las respuestas de **Post /usuario.**

```
app.post('/usuario', function(req, res) {
    var seguro = app.get("crypto").createHmac('sha256', app.get('clave'))
        .update(req.body.password).digest('hex');

        var usuario = {
                email : req.body.email,
                password : seguro
        }

        gestorBD.insertarUsuario(usuario, function(id) {
                if (id == null){
                        res.send("Error al insertar ");
                        res.redirect("/registrarse?mensaje=Error al registrar usuario")
                } else {
                        res.send('Usuario Insertado ' + id);
                        res.redirect("/identificarse?mensaje=Nuevo usuario registrado");
                }
        });

})
```

Podríamos aplicar los mensajes en muchas otras partes de la aplicación, es bueno darle al usuario feedback de las acciones que está realizando.

Comprar canción

La vista de detalles de canción **/views/bcancion.html** tiene un botón donde aparece el precio, sustituimos el botón por un enlace a **/cancion/comprar/:id**

```
{% block contenido_principal %}
 <div class="row">
        <div class="media col-xs-10">
          <div class="media-left media-middle">
                <img class="media-object" src="/portadas/{{ cancion._id.toString() }}.png" >
          </div>
          <div class="media-body">
                <h2>{{ cancion.nombre }}</h2>
                <p>{{ cancion.autor }}</p>
                <p>{{ cancion.genero }}</p>
```

```
                <button type="button" class="btn btn-primary pull-right">{{ cancion.precio }}
€</button>
                <a class="btn btn-primary pull-right"
                        href="/cancion/comprar/{{ cancion._id.toString() }}">{{
cancion.precio }} €</a>
                <!-- Cambiar el precio por "reproducir" si ya está comprada -->
          </div>
        </div>
```

Agregamos una nueva función **insertarCompra()** en **gestorBD.js** para agregar una compra que relaciona al usuario con la canción comprada. Cada documento **compra** registrará el **email** del usuario y la **_id** de la canción comprada.

En la función insertamos un nuevo objeto dentro de la colección **compras**.

```
module.exports = {
        mongo : null,
        app : null,
        init : function(app, mongo) {
                this.mongo = mongo;
                this.app = app;
        },
        insertarCompra: function(compra, funcionCallback) {
                this.mongo.MongoClient.connect(this.app.get('db'), function(err, db) {
                        if (err) {
                                funcionCallback(null);
                        } else {
                                var collection = db.collection('compras');
                                collection.insert(compra, function(err, result) {
                                        if (err) {
                                                funcionCallback(null);
                                        } else {
                                                funcionCallback(result.ops[0]._id);
                                        }
                                        db.close();
                                });
                        }
                });
        },
```

En el controlador **rCanciones.js** agregamos la respuesta para **GET /cancion/comprar/:id.** Una vez agregada la compra redirigimos la navegación a **/compras** (aunque todavía no la hemos implementado).

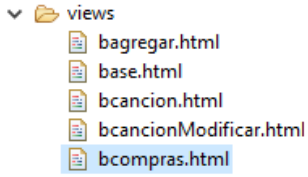
```
app.get('/cancion/comprar/:id', function (req, res) {
        var cancionId = gestorBD.mongo.ObjectID(req.params.id);
        var compra = {
                usuario : req.session.usuario,
                cancionId : cancionId
        }

        gestorBD.insertarCompra(compra ,function(idCompra){
                if ( idCompra == null ){
                        res.send(respuesta);
                } else {
                        res.redirect("/compras");
                }
        });
})
```

Implementamos **GET /compras** , como respuesta a esta URL se mostrarán todas las canciones que el usuario tenga en su lista **compras**.

Primero debemos implementar la función **obtenerCompras ()** en **gestorBD.js** Esta función va a retornar la lista de compras en función de un criterio que le será enviado como parámetro.

```
module.exports = {
        mongo : null,
        app : null,
        init : function(app, mongo) {
                this.mongo = mongo;
                this.app = app;
        },
        obtenerCompras : function(criterio,funcionCallback){
                this.mongo.MongoClient.connect(this.app.get('db'), function(err, db) {
                        if (err) {
                                funcionCallback(null);
                        } else {

                                var collection = db.collection('compras');
                                collection.find(criterio).toArray(function(err, usuarios) {
                                        if (err) {
                                                funcionCallback(null);
                                        } else {
                                                funcionCallback(usuarios);
                                        }
                                        db.close();
                                });
                        }
                });
        },
```

Movemos la vista **/public/bcompras.html** al directorio **/views/** . La vista recibe la lista de canciones y las muestra, es muy similar a **bpublicaciones.html**, pero sin los botones de modificar y eliminar.



Solo falta implementar **GET /compras**,

1. Obtendremos primero todas las compras realizadas por el usuario que hay en sesión. Cada compra tiene el Id de la canción comprada, podemos utilizar las ids para recuperar toda la información de la canción.
2. Guardamos en nuevo array **cancionesCompradasIds** las ids de todas las canciones compradas por el usuario. Llamamos a la función obtenerCanciones especificando como criterio que la id de la canción este en ese array: **{ "_id" : { $in: cancionesCompradasIds } }**

Finalmente, las canciones compradas por el usuario y las mostrará en la vista **bcompras.html.**

```
app.get('/compras', function (req, res) {
```

```
        var criterio = { "usuario" : req.session.usuario };

        gestorBD.obtenerCompras(criterio ,function(compras){
                if (compras == null) {
                        res.send("Error al listar ");
                } else {

                        var cancionesCompradasIds = [];
                        for(i=0; i <  compras.length; i++){
                                cancionesCompradasIds.push( compras[i].cancionId );
                        }

                        var criterio = { "_id" : { $in: cancionesCompradasIds } }
                        gestorBD.obtenerCanciones(criterio ,function(canciones){
                                var respuesta = swig.renderFile('views/bcompras.html',
                                {
                                        canciones : canciones
                                });
                                res.send(respuesta);
                        });
                }
        });
})
```

Revisamos los Routers declarados en **app.js** para ver si es conveniente incluir restricciones en las nuevas URLs

- **/cancion/comprar/:id** y **/compras** : comprobar que el usuario que accede está identificado en sesión.

```
//Aplicar routerUsuarioSession
app.use("/canciones/agregar",routerUsuarioSession);
app.use("/publicaciones",routerUsuarioSession);
app.use("/cancion/comprar",routerUsuarioSession);
app.use("/compras",routerUsuarioSession);
```

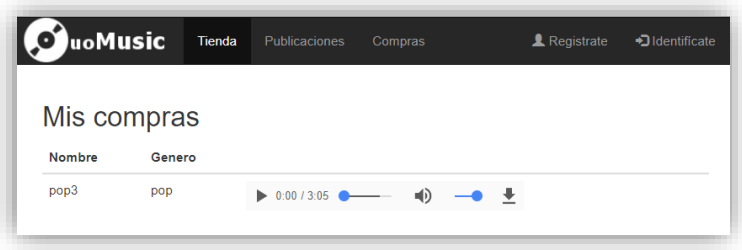
Ampliamos la lógica del **routerAudios** para que puedan acceder al fichero de audio también los usuarios que han comprado la canción (no solo los autores). Obtenemos el usuario y comprobamos su colección de compras para ver si el ID de la canción se encuentra en él.

```
//routerAudios
var routerAudios = express.Router();
routerAudios.use(function(req, res, next) {
     console.log("routerAudios");
     var path = require('path');
     var idCancion = path.basename(req.originalUrl, '.mp3');

     gestorBD.obtenerCanciones(
             {  id : mongo.ObjectID(idCancion) }, function (canciones) {

         if( canciones[0].autor == req.session.usuario ){
             next();
         } else {
             //res.redirect("/tienda");
             var criterio = {
                usuario : req.session.usuario,
                cancionId : mongo.ObjectID(idCancion)
             };

             gestorBD.obtenerCompras(criterio ,function(compras){
                 if (compras != null && compras.length > 0  ){
                     next();
                 } else {
```

```
                    res.redirect("/tienda");
                }
            });

        }
    })

});
```

Nota: Después de un **next()** no podemos volver enviar a otra respuesta o se producirá un error, una buena estrategia es incluir **return** después de los **next(),**

Guardamos los cambios, ejecutamos la aplicación y probamos a comprar una canción.



Sí quisiéramos mejorar el funcionamiento podríamos incluir una comprobación en **/cancion/comprar** para que no deje compra canciones que ya hemos comprado, o si somos el autor de las mismas.

También podríamos mejorar la página de detalles de canción **/cancion/id** para que si ya hemos comprado la canción se muestre el botón de reproducir canción en lugar de comprar.

Sistema de paginación

No suele ser adecuado utilizar listados de muchos elementos en una sola página, sería recomendable utilizar paginación. Existen varios módulos específicos para crea un sistema de paginación, como **express-paginate** (https://github.com/expressjs/express-paginate). No obstante implementarla desde cero no es complejo y ayuda a comprender su funcionamiento.

En **gestorBD.js** implementamos la función **obtenerCancionesPg(pag),** no devolverá todas las canciones sino las correspondientes a una página concreta. Hemos decidido que va a haber por 4 canciones por página. La página 1 tendrá las canciones 1 – 4, la página 2 las canciones 5 – 8, etc.

Obtenemos la colección de canciones, contamos cuantas canciones hay dentro (**.count**), después solicitamos las canciones situadas en ciertas posiciones utilizando la función **skipt()** , pondremos siempre un límite siempre de 4 resultados utilizando la función **limit()** .

A la función de callback le retornamos:

1. La lista de canciones que deberían ir en la página indicada como parámetro.

2. El número total de canciones (se utilizará para calcular cuantas páginas debe mostrar el sistema de paginación)

```
module.exports = {
mongo : null,
app : null,
init : function(app, mongo) {
        this.mongo = mongo;
        this.app = app;
},
obtenerCancionesPg : function(criterio,pg,funcionCallback){
        this.mongo.MongoClient.connect(this.app.get('db'), function(err, db) {
                if (err) {
                        funcionCallback(null);
                } else {

                        var collection = db.collection('canciones');
                        collection.count(function(err, count){

                                collection.find(criterio).skip( (pg-1)*4 ).limit( 4 )
                                        .toArray(function(err, canciones) {

                                        if (err) {
                                                funcionCallback(null);
                                        } else {
                                                funcionCallback(canciones, count);
                                        }
                                        db.close();
                                });

                        });
                }
        });
},
```

En el controlador modificamos el contenido de la función **GET /tienda**, vamos a admitir que pueda recibir un parámetro get con nombre **pg** que indicara la página a mostrar, por ejemplo **tienda?pg=2**.

Debemos ser cuidadosos ya que el valor de los parámetros es siempre un string, hay que transformarlo a entero **parseInt**(), también hay que tener en cuenta que se trata de un parámetro opcional , si no se recibe el parámetro le asignamos valor 1 (ir por defecto a la página 1)

A la vista **bTienda.html** le vamos a enviar la última página necesaria para mostrar todas las canciones **pgUltima** (canciones totales / canciones por página: redondear el resultado hacia arriba, si el resultado es que la última página es 5,2 debería de ser la 6), la lista de **canciones** (como antes) y la pagina actual **pgActual** .

```
app.get("/tienda", function(req, res) {
    var criterio = {};
    if( req.query.busqueda != null ){
        criterio = { "nombre" : {$regex : ".*"+req.query.busqueda+".*"}  };
    }

    var pg = parseInt(req.query.pg); // Es String !!!
    if ( req.query.pg == null){ // Puede no venir el param
        pg = 1;
    }

    gestorBD.obtenerCancionesPg(criterio, pg , function(canciones, total ) {
        if (canciones == null) {
            res.send("Error al listar ");
```

```
        } else {

            var pgUltima = total/4;
            if (total % 4 > 0 ){ // Sobran decimales
                pgUltima = pgUltima+1;
            }

            var respuesta = swig.renderFile('views/btienda.html',
            {
                canciones : canciones,
                pgActual : pg,
                pgUltima : pgUltima
            });
            res.send(respuesta);
        }
    });

});
```

Modificaremos la parte final de la vista **bTienda.html,** evaluando los casos, páginas a mostrar:

- La **primera página** 1 **siempre**
- La **anterior a la actual** pgActual – 1. Si es que existe (pgActual – 1 >= 1)
- La **actual** pgActual **siempre**
- La **siguiente a la actual** pgActual +1. Si es que existe (pgActual + 1 <= pgUltima)
- La **última** pgUltima **siempre**

```
<div class="row text-center">
    <ul class="pagination">
        <!-- Primera -->
        <li class="page-item" >
            <a class="page-link" href="/tienda?pg=1" >Primera</a>
        </li>

        <!-- Anterior (si la hay ) -->
        {% if pgActual-1 >= 1 %}
        <li class="page-item">
            <a class="page-link" href="/tienda?pg={{ pgActual -1 }}" >{{ pgActual -1 }}</a>
        </li>
        {% endif %}

        <!-- Actual  -->
        <li class="page-item active" >
            <a class="page-link" href="/tienda?pg={{ pgActual }}">{{ pgActual }}</a>
        </li>

        <!-- Siguiente (si la hay)  -->
        {% if pgActual+1 <= pgUltima %}
        <li class="page-item">
            <a class="page-link" href="/tienda?pg={{ pgActual+1 }}" >{{ pgActual+1 }}</a>
        </li>
        {% endif %}

        <!-- Última -->
        <li class="page-item" >
            <a class="page-link" href="/tienda?pg={{ pgUltima }}" >Última</a>
        </li>
    </ul>
</div>
```

Guardamos los cambios, ejecutamos la aplicación y comprobamos que la paginación funciona en todos los casos.

Errores en la aplicación

Por defecto y entorno de desarrollo se suele dejar que la aplicación lance excepciones. Cada vez que se produce una excepción se muestra una traza (aunque la aplicación no debería detenerse).

No es nada recomendable que en un entorno de producción se muestren trazas de excepción. Vamos a probar a acceder a acceder a los detalles de una canción con una ID en formato invalido http://localhost:8081/cancion/RRRRRRRRRRRRRRRRRRRRR se produce una excepción y un retorno con **Status 500** por parte del sitio web.

```
Error: Argument passed in must be a single String of 12 bytes or a string of 24 hex characters
    at new ObjectID (C:\Users\jordansoy\work\TiendaMusica\node_modules\bson\lib\bson\objectid.js:50:11)
    at Function.ObjectID (C:\Users\jordansoy\work\TiendaMusica\node_modules\bson\lib\bson\objectid.js:31:42)
```

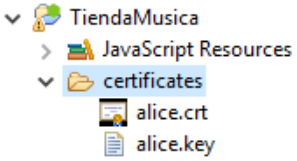
Una forma de detener las excepciones antes de que lleguen al cliente es interceptándolas, las podríamos almacenar en un log y dar una respuesta distinta. Añadimos la función básica de manejo de errores en el fichero **app.js** (Los objetos err o err.stack nos da información sobre el error) .

```
app.use( function (err, req, res, next ) {
    console.log("Error producido: " + err); //we log the error in our db
    if (! res.headersSent) {
        res.status(400);
        res.send("Recurso no disponible");
    }
});

app.listen(app.get('port'), function() {
        console.log("Servidor activo");
});
```

Accedemos de nuevo a la URL anterior para ver que el problema se ha solucionado.

(Opcional) Https

Descargamos unos certificados SSL del campus virtual **alice.crt** y **alice.key** creamos la carpeta **certificates** y copiamos dentro los certificados.



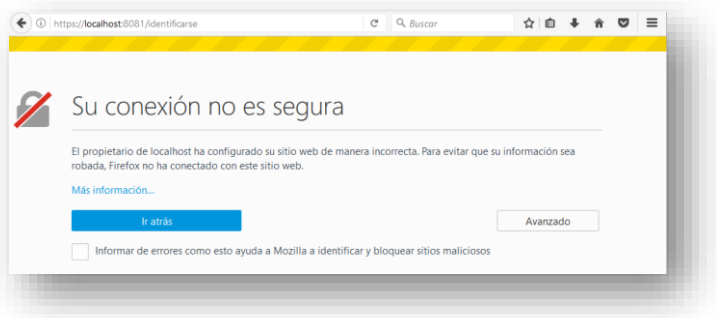
Abrimos el fichero principal **app.js** e incluimos dos nuevos módulos **fs (file system) y https.** No es necesario descargar esto módulos puesto que están incluidos en el core de Node.

```
// Módulos
var express = require('express');
var app = express();

var fs = require('fs');
var https = require('https');
```

Modificamos la creación del servidor para utilizar **https**, indicándole donde está la clave y el certificado.

```
app.listen(app.get('port'),·function()·{
        console.log("Servidor activo");
});

https.createServer({
    key: fs.readFileSync('certificates/alice.key'),
    cert: fs.readFileSync('certificates/alice.crt')
}, app).listen(app.get('port'), function() {
        console.log("Servidor activo");
});
```

A partir de ahora debemos acceder a nuestra aplicación con https://localhost:8080 . El navegador nos advertirá que el certificado de la página no es "confiable" y debemos confirmar una excepción de seguridad para este certificado, aunque son certificados validos no los ha generado una entidad confiable y esto alerta al navegador.



Una vez probado volver a la versión anterior sin http