Examen

Final septiembre 2017

Aplicaciones Web
Año 2016/2017
Grado en Ing. Software
Fdi - UCM

▷ 1. Acceso a BD mediante *Node* y *Express*

[4.5 pt] Suponemos una base de datos que almacena las canciones contenidas en una lista de reproducción (*playlist*). Esta base contiene una única tabla (`playlist`) cuyo contenido es el siguiente:

id	title	author	album	year
1	Eternal Odyssey	Delerium	Chimera	2003
2	Somebody to love	Queen	A day at the races	1976
3	Believe	DB Boulevard	Frequencies	2004

(a). *[0.75 pt]* Implementa una función `getPlayList(callback)` que obtenga todas las canciones de la base de datos. Recibe como parámetro una función `callback` con dos parámetros: El primero de ellos sirve para propagar el objeto error (si lo hay) y el segundo parámetro, en caso de éxito, debe contener una lista de objetos, cada uno de ellos con cinco atributos: `id`, `title`, `author`, `album`, `year`.

(b). *[0.75 pt]* Implementa una función `insertInPlaylist(title, author, album, year, callback)` que inserte una fila en la tabla `playlist` con la información pasada como parámetro, y después llame a la función `callback` pasándole un objeto error si se produce algún fallo, o `null` en caso contrario.

(c). *[1.5 pt]* Diseña una aplicación *Express.js* que utilice el método del apartado (a) para visualizar el contenido de la tabla `playlist`. Para ello implementa el siguiente router,

```
app.get("/showPlaylist", function(request, response) {
  // ...
});
```

en el que se muestre una vista que contenga una tabla HTML (`<table>`) con la información de la tabla de la base de datos (Figura 1).

(d). *[1.5 pt]* Supongamos que la ruta `/newEntry.html` muestra un formulario que contiene cuatro campos de texto para introducir el título, autor, álbum y año de una canción (Figura 2). Al pulsar el botón `Enviar` del formulario se realiza una petición POST a la ruta `/insertEntry`. Implementa el manejador de esta ruta para que se inserte la información introducida en el formulario en la base de datos:

```
app.post("/insertEntry", function(request, response) {
  // ...
});
```

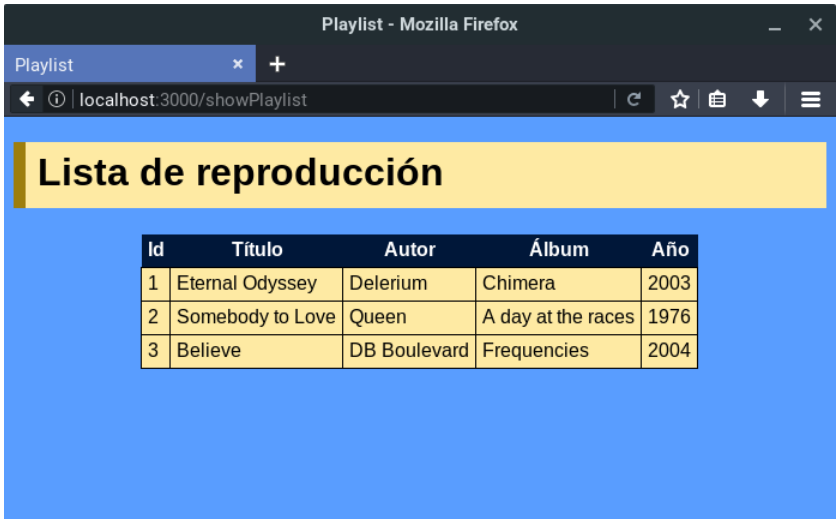

Figura 1: Visualización de la lista de reproducción.

En caso de éxito se debe redirigir a la ruta `/showPlaylist`. Si alguno de los campos del formulario está vacío, o el campo año no contiene un número, se debe volver a la página del formulario, introduciendo antes de este la cadena `Información incorrecta` (Figura 3).

Solución

(a). *[0.75 pt]* La sentencia SQL que debemos ejecutar es sencilla:

```
SELECT id, title, author, album, year FROM playlist
```

Suponemos la existencia de una función que crea una nueva conexion a la base de datos. Esta función será reutilizada en el apartado (b) de este ejercicio:

```
function createConnection() {
    return mysql.createConnection({
        host: "localhost",
        database: "examen_septiembre",
        user: "root",
        password: ""
    });
}
```

La función `getPlaylist` queda implementada del siguiente modo:

```
function getPlaylist(callback) {
    var connection = createConnection();
    connection.connect(function(err) {
        if (err) {
            callback(err);
```

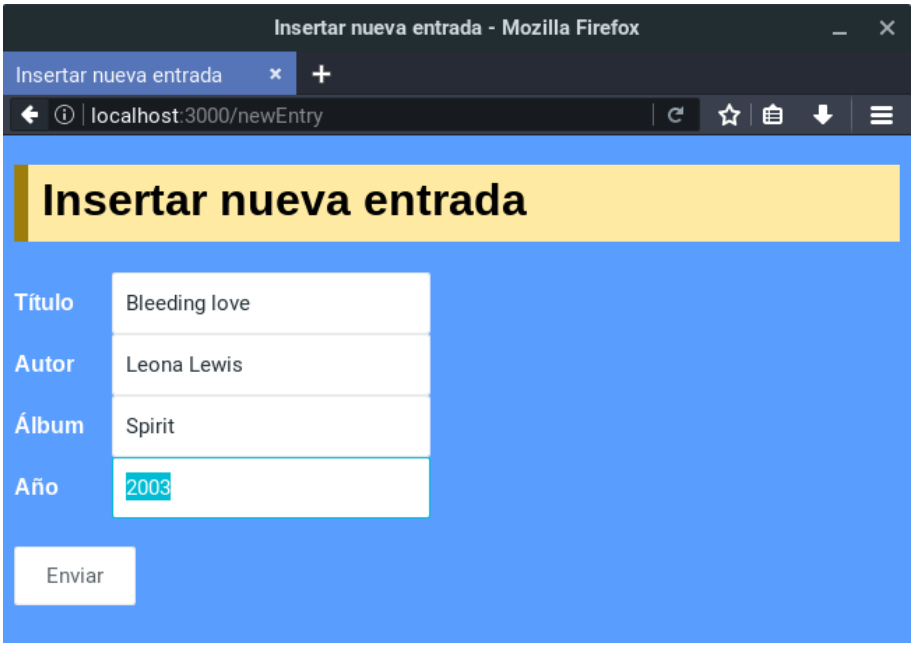

Figura 2: Inserción de una nueva entrada en lista de reproducción.

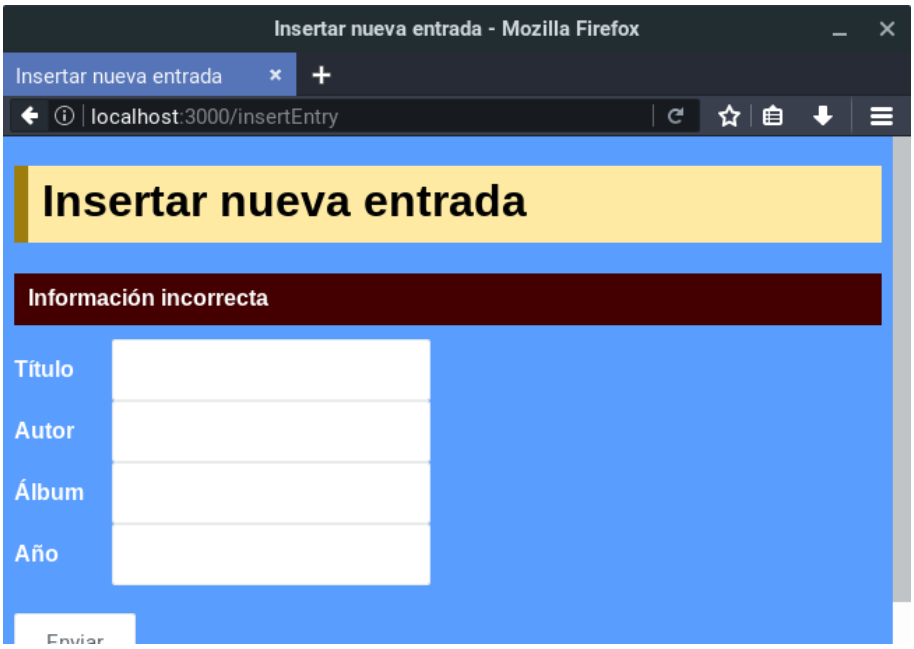

Figura 3: Mensaje de error al introducir información no válida.

```
        } else {
            connection.query(
              "SELECT id, title, author, album, year FROM playlist",
              function(err, result) {
                connection.end();
                if (err) {
                    callback(err);
                } else {
                    callback(null, result);
                }
            });
        }
    });
}
```

(b). *[0.75 pt]* De nuevo, la sentencia a ejecutar es la habitual:

```
INSERT INTO playlist(title, author, album, year) VALUES (?,?,?,?)
```

Se trata de una consulta paramétrica en la que los cuatro marcadores contendrán los datos de la entrada a insertar.

```
function insertInPlaylist(title, author, album, year, callback) {
    var connection = createConnection();
    connection.connect(function(err) {
        if (err) {
            callback(err);
        } else {
            connection.query(
              "INSERT INTO playlist(title, author, album, year) " +
              "VALUES (?,?,?,?)",
              [title, author, album, year],
              function(err) {
                  connection.end();
                  if (err) {
                      callback(err);
                  } else {
                      callback(null);
                  }
              });
        }

    });
}
```

(c). *[1.5 pt]* En este manejador de ruta haremos uso de la función implementada en el apartado (a):

```
app.get("/showPlaylist", function(request, response) {
    dao.getPlaylist(function(err, playlist) {
        if (err) {
            console.log("Error al acceder a la BD: " + err.message);
        } else {
            response.render("showPlaylist", { playlist: playlist });
        }
    });
});
```

Donde la vista showPlaylist viene dada por la plantilla EJS siguiente:

```
<!DOCTYPE html>
<html>
...
<table>
    <tr>
        <th>Id</th>
        <th>Título</th>
        <th>Autor</th>
        <th>Álbum</th>
        <th>Año</th>
    </tr>
    <% playlist.forEach(function(element) { %>
        <tr>
            <td><%= element.id %></td>
            <td><%= element.title %></td>
            <td><%= element.author %></td>
            <td><%= element.album %></td>
            <td><%= element.year %></td>
        </tr>
    <% }); %>
</table>
...
</html>
```

(d). *[1.5 pt]* La página web que muestra el formulario ha de ser dinámica, ya que contiene un mensaje de error (`Información incorrecta`) que puede, o no, aparecer en el resultado. Por este motivo creamos la plantilla EJS (`newEntry.ejs`) correspondiente a este formulario, que irá precedido por el mensaje de error en el caso de que una variable booleana `error` tenga el valor `true`.

```
<!DOCTYPE html>
<html>
...
<% if(error) { %>
<div class="error">
    Información incorrecta
```

```
</div>
<% } %>
<form action="/insertEntry" method="POST">
    <div class="formulario">
        <div class="fila_formulario">
            <div class="celda_enunciado">
                Título
            </div>
            <div class="cuadro_entrada">
                <input type="text" name="title">
            </div>
        </div>
        <div class="fila_formulario">
            <div class="celda_enunciado">
                Autor
            </div>
            <div class="cuadro_entrada">
                <input type="text" name="author">
            </div>
        </div>
        <div class="fila_formulario">
            <div class="celda_enunciado">
                Álbum
            </div>
            <div class="cuadro_entrada">
                <input type="text" name="album">
            </div>
        </div>
        <div class="fila_formulario">
            <div class="celda_enunciado">
                Año
            </div>
            <div class="cuadro_entrada">
                <input type="text" name="year">
            </div>
        </div>
    </div>
    <input type="submit" value="Enviar">
</form>
...
</html>
```

De este modo, al introducir la URL `/newEntry.html` en el navegador, deberá mostrarse esta vista sin el mensaje de error:

```
app.get("/newEntry.html", function(request, response) {
```

```
    response.render("newEntry", { error: false });
});
```

El formulario realiza una petición POST a la URL `/insertEntry` cuando el usuario hace clic en el botón `[Enviar]`. En el manejador de ruta de esta URL se comprueba la validez de la información introducida.

```
app.post("/insertEntry", function(request, response) {
    var title = request.body.title.trim();
    var author = request.body.author.trim();
    var album = request.body.album.trim();
    var year = Number(request.body.year.trim());

    if (title === "" || author === "" || album === "" || year === 0 || isNaN(
    year)) {
        response.render("newEntry", { error: true });
    } else {
        dao.insertInPlaylist(title, author, album, year, function(err) {
            if (err) {
                console.log("Error al acceder a la BD: " + err.message);
            }
            response.redirect("/showPlaylist");
        });
    }
});
```

Vemos que, en caso de que la información sea incorrecta, se indica de nuevo la vista del formulario con la variable `error` establecida a `true`, con lo que se mostrará el formulario precedido por el mensaje de error.

▷ 2. Manejo de *cookies* en *Express*

[2.5 pt] Suponiendo una aplicación *Express.js*:

(a). *[1 pt]* Escribe un manejador de ruta "/" que muestre una página con el texto "`Has visitado esta página XX veces`", donde `XX` es un contador que se incrementa cada vez que el usuario accede a dicha ruta. Utiliza *cookies* para almacenar el valor del contador y una plantilla EJS para mostrar este valor al usuario.

(b). *[1 pt]* Modifica la ruta anterior para que solo se permita acceder al valor del contador si el usuario ha introducido previamente la contraseña "`aw`" en un formulario como el de la Figura 4. Si la contraseña es correcta, se inicializará el contador de visitas a cero y se mostrará dicho contador. Si la contraseña no es correcta, se redirigirá a una página `wrongPassword.html` indicándolo. Si el usuario intenta acceder al contador mediante la ruta "/" sin haber introducido la contraseña previamente, se le redirigirá al formulario de introducción de contraseña. Para realizar este ejercicio implementa un *middleware* que compruebe si el usuario ha introducido la contraseña previamente, y redirija al formulario de introducción de contraseña en caso contrario.

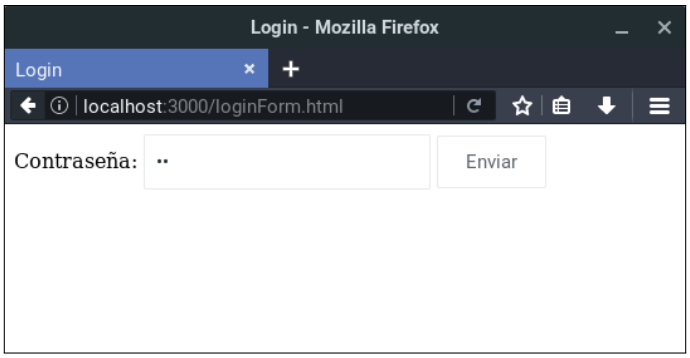

Figura 4: Introducción de contraseña.

(c). *[0.5 pt]* Añade a la página del contador un enlace `[Desconectar]` que, al ser pulsado, elimine la información sobre la contraseña introducida previamente por el usuario, de modo que este último tenga que volver al formulario de la Figura 4 para acceder al contador.

Solución

(a). *[1 pt]* Si hemos cargado previamente el middleware `cookie-parser`, tendremos disponibles en el atributo `request.cookies` los valores de las cookies que hayamos establecido. Para este ejercicio estableceremos una cookie llamada `contador` con el valor actual del contador. La primera vez que se accede a la página, el valor de `request.cookies.contador` será `undefined`, por lo que al convertir este valor a un número se obtendrá `NaN`, e inicializaremos el contador a cero. Tras obtener el valor actual (o inicial) del contador, guardamos la cookie mediante el método `response.cookie()` con el contador incrementado, y mostramos la vista con el valor actual del contador.

```
app.get("/", function(request, response) {
    var contador = Number(request.cookies.contador);
    if (isNaN(contador)) {
        contador = 0;
    }
    response.cookie("contador", contador + 1);
    response.render("showCounter", { num: contador });
});
```

Fichero `showCounter.ejs`:

```
...
<p class="visited">Has visitado esta página <%= num %> veces desde que te
    conectaste.</p>
...
```

(b). *[1 pt]* En este caso necesitamos una cookie adicional (`logged`) que contendrá la cadena "`yes`" en el caso en que el usuario haya introducido previamente la contraseña. El manejador de ruta POST del formulario se encarga de establecer esta cookie en caso de que la contraseña sea correcta:

```
app.post("/login", function(request, response) {
    var passwd = request.body.pass;
    if (passwd === "aw") {
        response.cookie("logged", "yes");
        response.cookie("contador", 0);
        response.redirect("/logged.html");
    } else {
        response.redirect("/wrongPassword.html");
    }
});
```

Además, el manejador de ruta `/login` se encargará de reinicializar el contador a cero.

El siguiente *middleware* se encarga de comprobar si existe la cookie `logged` con el valor `yes`. En caso contrario, redirige a la página de petición de contraseña:

```
function middlewareLogged(request, response, next) {
    var logged = request.cookies.logged;
    if (logged === "yes") {
        next();
    } else {
        response.redirect("/loginForm.html");
    }
}
```

Incorporamos este middleware al manejador de la ruta `/`, de modo que se compruebe la comprobación anterior antes de mostrar el valor del contador:

```
app.get("/", middlewareLogged, function(request, response) {
    // ... igual que antes ...
});
```

(c). *[0.5 pt]* Se trata, básicamente, de añadir una ruta `/logout` que, o bien elimine la cookie, o bien la establezca a un valor distinto de `yes`:

```
app.get("/logout", function(request, response) {
    response.cookie("logged", "no");
    response.redirect("/loggedOut.html");
});
```

▷ 3. AJAX y servicios web

[3 pt] Partimos de una página como la de la Figura 5. Esta página contiene dos elementos `<div>` que contienen, respectivamente, una lista de productos disponibles (izquierda) y un carro de la compra inicialmente vacío (derecha).

```
<div id="stockD">
    <ul id="stock">
        <li>Aceite</li>
```

Figura 5: Carro de la compra.

```
        <li>Huevos</li>
        ...
    </ul>
</div>
<div id="carroD">
    <ul id="carro">
    </ul>
</div>
```

(a). *[1.5 pt]* Utilizando *jQuery* añade el manejador de eventos necesario para que al hacer clic en uno de los `<li>` de la lista de productos disponibles, se elimine el elemento seleccionado de la lista de productos disponibles y se añada a la lista del carro de la compra.

(b). *[1.5 pt]* Modifica la página anterior para que la lista inicial de productos disponibles se obtenga mediante una petición AJAX al servidor mediante la ruta `/stock`. Implementa dicha ruta en el servidor suponiendo que los productos están almacenados en el siguiente array:

```
var stock = ["Aceite", "Galletas", "Pan", "Cebolla", "Cereales", "Huevos",
             "Tomate", "Mantequilla", "Leche", "Café"];
```

Tras realizar la petición AJAX, se debe rellenar el `<ul>` de la lista de productos disponibles a partir del resultado devuelto por el servidor.

Solución

(a). *[1.5 pt]* Mediante el manejador del evento `ready` del objeto `document` podemos realizar acciones que se ejecutarán cuando el documento esté cargado. En nuestro caso nos limitaremos a capturar los eventos de pulsación sobre los elementos de la lista de productos disponibles:

```
$(document).ready(function () {
    $("#stock␣li").on("click", moverAlCarro);
});
```

Donde la función `moverAlCarro` se implementa del siguiente modo:

```
function moverAlCarro() {
    $(this).remove();
    $("#carro").append($(this));
}
```

(b). *[1.5 pt]* Implementamos el manejador de ruta en el servidor:

```
app.get("/stock", function(request, response) {
    response.json(stock);
});
```

Por último, modificamos el manejador del evento `ready` para que realice una petición AJAX a la ruta anterior e inserte los elementos en la página web:

```
$(document).ready(function () {
    $.ajax({
        method: "GET",
        url: "/stock",
        success: function (data, statusText, jqXHR) {
            data.forEach(function(elemento) {
                var li = $("<li>").text(elemento);
                $("#stock").append(li);
            });
            $("#stock␣li").on("click", moverAlCarro);
        },
        error: function (jqXHR, statusText, errorThrown) {
            console.log("Error:␣" + errorThrown);
        }
    });
});
```