Hoja de ejercicios Tema 5

Frameworks en el servidor: Express.js

▷ **1.** Implementa una aplicación web que realice una encuesta. Los usuarios de la aplicación deberán responder a una determinada pregunta seleccionando una de las opciones disponibles (Figura 1). Cuando el usuario haga clic en el botón `Enviar`, se mostrará una página con los votos recibidos por cada opción hasta el momento (Figura 2). Por simplicidad, puedes suponer que la pregunta a realizar y el número de votos de cada opción están guardados en variables globales de la aplicación, como se muestra a continuación:

```
var pregunta = "¿Cuál es tu color favorito?";

var opciones = [
    {
        texto: "Rojo",
        numeroVotos: 0
    },
    {
        texto: "Azul",
        numeroVotos: 0
    },
    {
        texto: "Verde",
        numeroVotos: 0
    },
    {
        texto: "Ninguno de los anteriores",
        numeroVotos: 0
    }
];
```

▷ **2.** Supongamos que tenemos una variable `usuarios` que contiene un array con la lista de usuarios de una determinada aplicación:

```
let usuarios = ["Javier Montoro", "Dolores Vega", "Beatriz Nito"];
```

En clase de teoría hemos visto cómo mostrar una página web que contenga un listado de usuarios como el mostrado en la Figura 3. Modifica este ejemplo para que, al lado de cada uno de los usuarios, se introduzca un enlace que permita eliminar al correspondiente usuario del listado. El navegador ha de realizar una petición al servidor indicando el índice, dentro del array, del elemento que se quiere eliminar.

(a). Realiza este ejercicio utilizando peticiones GET con URLs paramétricas (por ejemplo: `/borrar/1`, `/borrar/2`, etc.) para indicar el índice del array a eliminar.

(b). Modifica el apartado anterior para peticiones GET con parámetros en la *query string* de la URL (por ejemplo: `/borrar?id=1`, `/borrar?id=2`).

(c). Modifica el apartado anterior para realizar peticiones de tipo POST. En este caso, el índice del array ha de pasarse al servidor mediante formularios. Para ello sustituye el enlace para borrar un usuario por un

Figura 1: El usuario ha de contestar a una de las opciones.

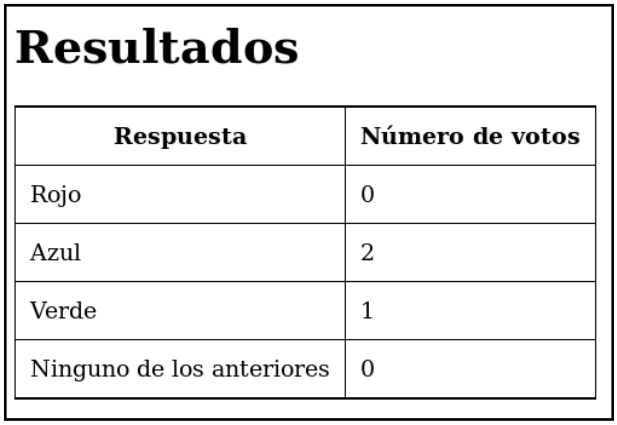

Figura 2: Resultados de la encuesta.

Figura 3: Listado de usuarios de una aplicación.

`<input>` de tipo `submit`, e incluye en el formulario el índice del array a borrar mediante un `<input>` de tipo `hidden`.

▷ **3.** Realiza una aplicación web para sumar dos números. La petición de información se hará en dos fases: en una página se pedirá el primer sumando, y en otra página distinta se pedirá el segundo sumando. El resultado de la suma se mostrará en una tercera página. En la Figura 4 se muestran estas tres páginas. Cuando el usuario hace clic en el botón `Enviar` de la primera página se guardará el número introducido en una *cookie* y se saltará a la segunda página de la Figura 4. Al hacer clic en el botón `Enviar`, se guardará el número en otra *cookie* y se saltará a la tercera página, donde se mostrará el resultado de sumar los números introducidos previamente.

Si el usuario introduce una entrada no válida en alguno de los cuadros de texto (por ejemplo, si introduce `"abc"`), el servidor volverá a la página en la que se introdujo la entrada incorrecta junto con un mensaje de error (ver Figura 5).

Cuando el usuario acaba en la última página de la aplicación (la que muestra la suma), se deberán borrar las cookies que contienen cada uno de los sumandos. Utiliza, para ello, el método `clearCookie` del objeto `response`.

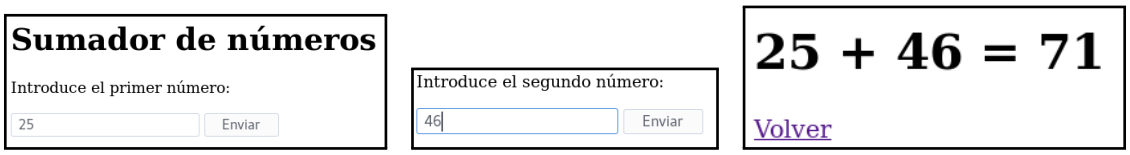

Figura 4: Páginas de las que se compone la aplicación.

Figura 5: Mensajes de error.

▷ **4. Examen septiembre 2017**

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). 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). 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`

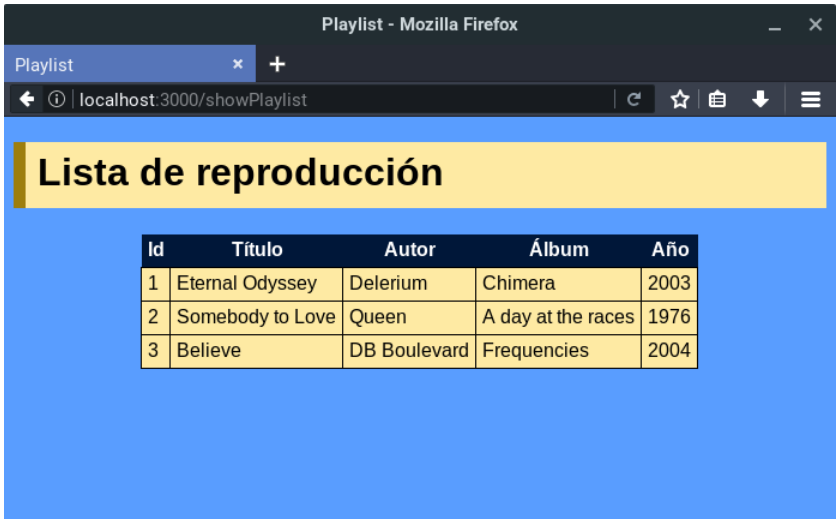

Figura 6: Visualización de la lista de reproducción.

pasándole un objeto error si se produce algún fallo, o `null` en caso contrario.

(c). 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 6).

(d). 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 7). 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) {
  // ...
});
```

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 8).

▷ 5. Examen septiembre 2017

Suponiendo una aplicación *Express.js*:

(a). 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.

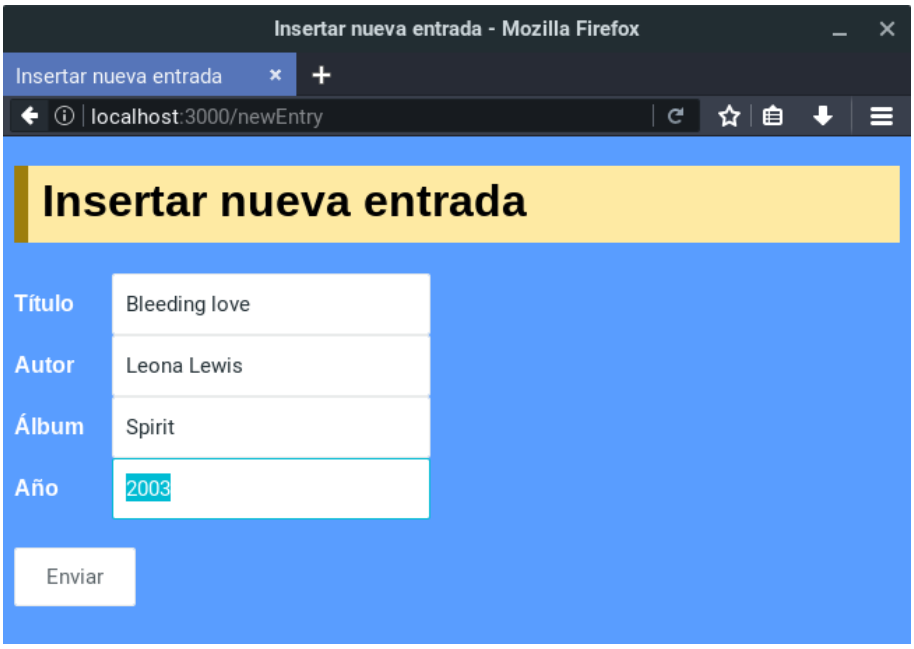

Figura 7: Inserción de una nueva entrada en lista de reproducción.

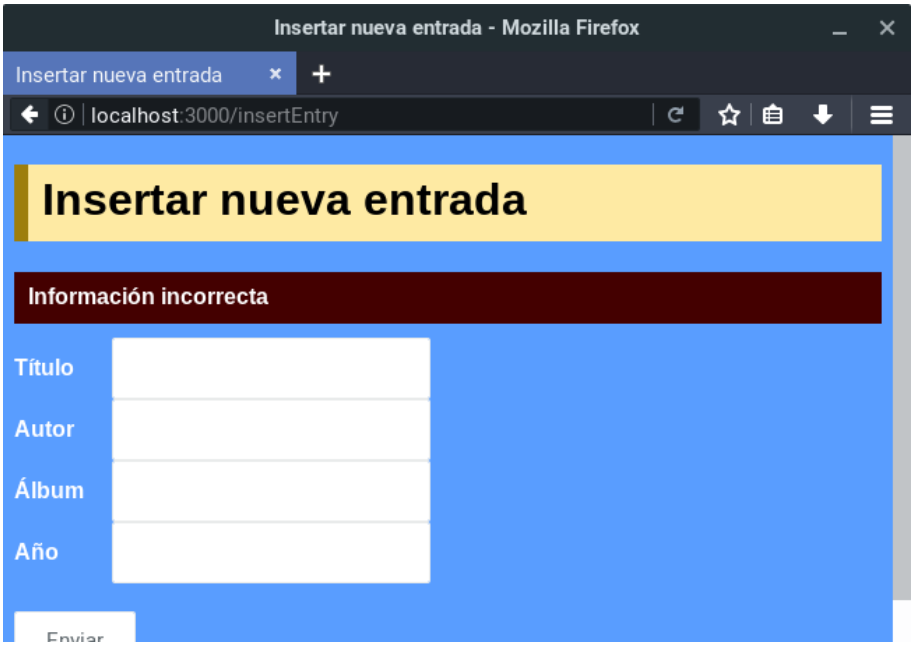

Figura 8: Mensaje de error al introducir información no válida.

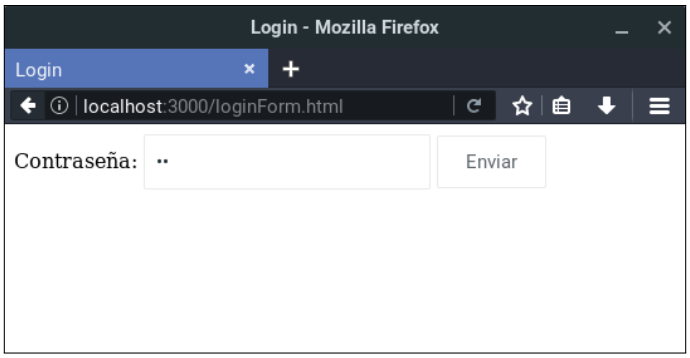

Figura 9: Introducción de contraseña.

(b). 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 9. 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.

(c). 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 9 para acceder al contador.

▷ 6. Examen febrero 2017

La constante matemática $e \approx 2{,}718281828459\ldots$ es un número irracional que puede aproximarse mediante la siguiente expresión (donde $n \geq 0$):

$$e \approx \sum_{i=0}^{n} \frac{1}{i!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

donde la notación $i!$ indica "factorial de i". La aproximación es más precisa cuanto mayor sea el valor de n. Implementa una aplicación web que solicite el valor de n al usuario y calcule una aproximación de e mediante la fórmula anterior (Figura 10).
Si el usuario no ha introducido un número, se volverá a mostrar el formulario junto con un mensaje de error (Figura 11).

▷ 7. Examen febrero 2017

El objetivo de este ejercicio es la realización de una aplicación web en la que el usuario pueda introducir una lista de tareas pendientes y marcar aquellas que ha finalizado (Figura 12). Las tareas han de guardarse **en la sesión del navegador** utilizando el *middleware* `express-session`.

(a). Implementa un manejador de ruta `/index.html` que muestre todas las tareas introducidas por el usuario, junto con un formulario invitando al usuario a introducir una nueva tarea. Implementa otro manejador para la ruta `/anyadirTarea` que recoja la información introducida en el formulario y añada la tarea introducida a la lista. De momento puedes ignorar la línea `Número de tareas pendientes` que se muestra en la Figura 12 y el enlace `Marcar como finalizada` mostrado a la derecha de cada enlace.

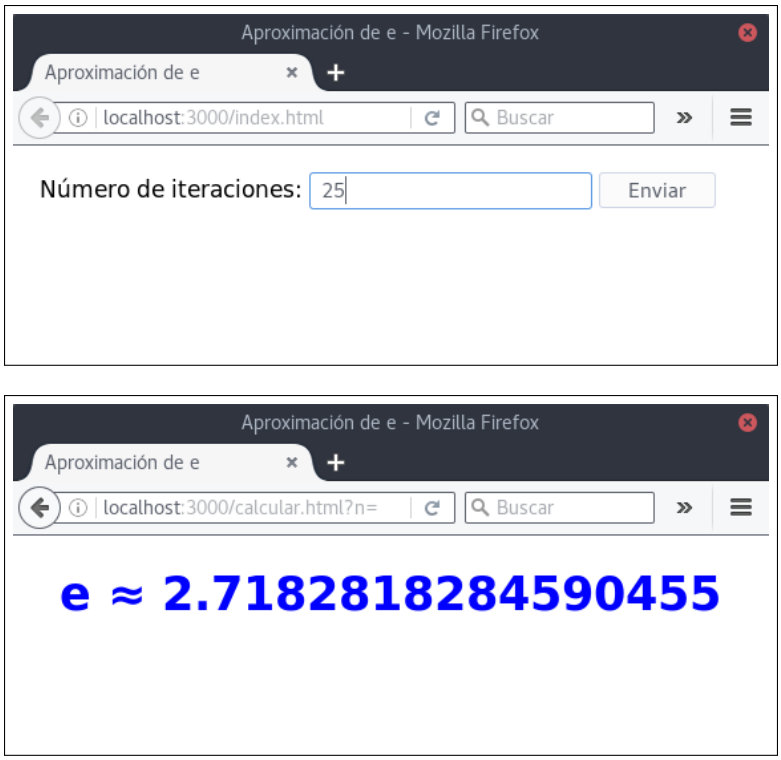

Figura 10: Funcionamiento de la aplicación para el cálculo de *e*

(b). Modifica la página anterior para que se muestre un enlace `[Marcar como finalizada]` a la derecha de cada tarea. Al hacer clic en ese enlace, la tarea se marcará como finalizada, y se mostrará de nuevo la página con la lista de tareas. Las tareas marcadas como finalizadas aparecerán tachadas y sin el enlace en el lado derecho (Figura 12).

(c). Añade un *middleware* a la aplicación anterior para que calcule el número de tareas de la lista que no hayan sido finalizadas y lo asigne a la variable `response.locals.numPendientes`. Modifica la página para que también muestre la lista de tareas pendientes.

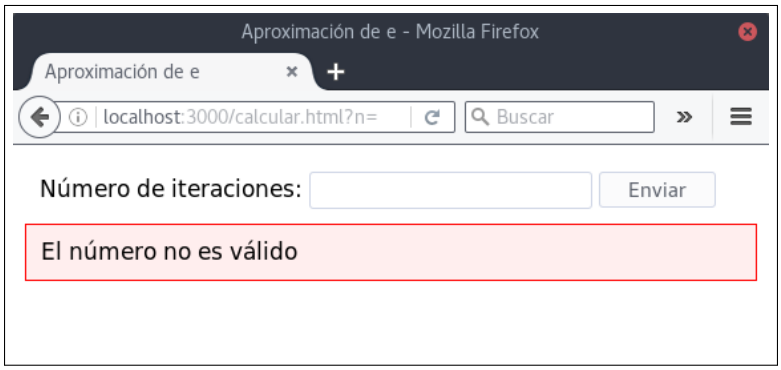

Figura 11: Mensaje de error en el formulario

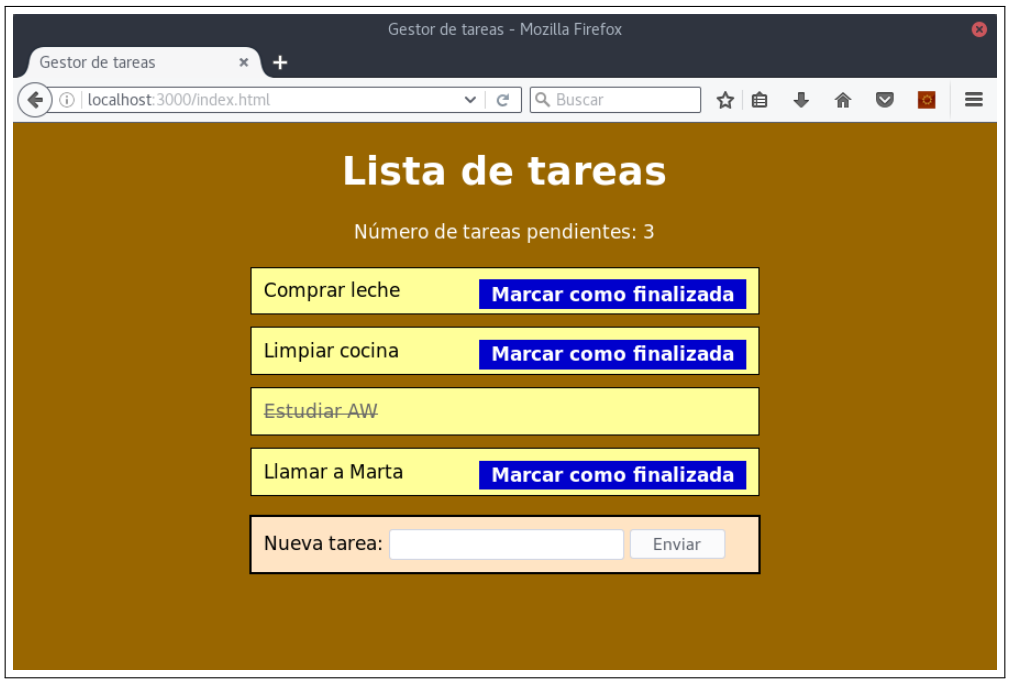

Figura 12: Aplicación de lista de tareas