Hoja de ejercicios Tema 4

Javascript en el servidor: Node.js

Aplicaciones Web
Año 2017/2018
Grado en Ing. Software
Fdi - UCM

▷ **1.** Supongamos un fichero que contiene una secuencia de números separados por espacios o caracteres de fin de línea. Por ejemplo:

```
3 5 6 10 21 9
```

Partimos de una clase `Lector` contenida en un módulo `lectorFichero.js`, cuya implementación encontrarás en el Campus Virtual. El constructor de esta clase recibe el nombre del fichero a leer, y tiene un método `siguienteEntero(callback)` para leer sucesivamente los números que componen el fichero. Cada llamada a este método lee el siguiente número del fichero y lo pasa a la función `callback`. La función callback recibe dos parámetros: el objeto `Error` (si lo hubiere) y el número leído. Podemos obtener los distintos números del fichero mediante sucesivas llamadas a `siguienteEntero`:

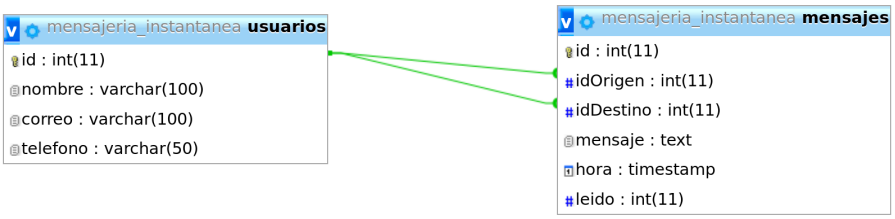
```
let l = new Lector("fichero.txt");

l.siguienteEntero(function(err, num)
    console.log("Primer número: " + num);
    l.siguienteEntero(function(err, num)
        console.log(Segundo número: " + num);
    );
);
```

(a). Escribe una función `sumaDosNumeros(nombreFichero, callback)` que, utilizando la clase `Lector`, lea los dos primeros números del fichero pasado como primer parámetro y llame a `callback` pasándole la suma de ambos.

(b). Escribe una función `sumaTodosLosNumeros(nombreFichero, callback)` que, utilizando la clase `Lector`, calcule la suma de los números del fichero pasado como parámetro y pase el resultado a la función `callback`. Utiliza las funciones auxiliares que necesites.

Indicación: Las funciones callback de este apartado y del anterior reciben dos parámetros: el objeto de tipo `Error` (si se produce un error, o `undefined` en otro caso) y el resultado del cálculo.

▷ **2.** Supongamos el siguiente esquema relacional de un sistema de mensajería instantánea:



Tenemos una tabla de usuarios y una tabla de mensajes. Ambas tablas tienen un atributo `id` que es incrementado automáticamente por el SGBD. Para cada mensaje se indica el usuario emisor, el usuario receptor, el mensaje a enviar, y la fecha/hora en la que se envió. También se incluye un campo (que puede tomar el valor 0 o 1) para indicar si el mensaje ha sido leído por el receptor o no.

Diseña un módulo `dao.js` que exporte una clase `DAO` con cuatro métodos:

- `insertarUsuario(usuario, callback)`: Inserta un usuario en la BD. El `usuario` pasado como parámetro es un objeto con tres atributos: `nombre`, `correo`, `telefono`. El método debe añadir un atributo nuevo `id` al objeto `usuario` recibido como parámetro. El valor de este atributo es el identificador con el que se ha insertado la fila en la tabla correspondiente de la BD.
- `enviarMensaje(usuarioOrigen, usuarioDestino, mensaje, callback)`: Inserta un mensaje en la BD. Los parámetros `usuarioOrigen` y `usuarioDestino` son objetos como los descritos en `insertarUsuario`, y en los que se supone la existencia de un atributo `id`.
- `bandejaEntrada(usuario, callback)`: Recupera los mensajes no leídos del usuario pasado como parámetro. En caso de éxito, la función `callback` recibirá un array con dichos mensajes, cada uno de ellos representado como un objeto con los atributos `nombre`, `mensaje` y `hora`, siendo `nombre` el nombre del usuario que ha enviado el mensaje.
- `buscarUsuario(str, callback)`: Recupera los usuarios cuyo nombre contenga la cadena pasada como parámetro. La función `callback` recibirá un array con dichos usuarios.

El constructor de la clase `DAO` recibe cuatro parámetros: el *host* en el que se encuentra la base de datos, el nombre de usuario y contraseña con el que se realizarán las conexiones a la BD, y el nombre de la BD. Cada uno de los cuatro métodos mencionados anteriormente deberá crear una conexión y finalizarla tras realizar la operación correspondiente.

▷ **3.** Utilizando el módulo creado en el apartado anterior, realiza una pequeña aplicación que añada usuarios a la base de datos. Para ello haz uso del módulo `http`. El servidor web ha de poder atender dos tipos de peticiones, ambas con el método GET:

- `/form.html`. Devuelve al cliente un formulario en formato HTML para que pueda introducir los datos del nuevo usuario de la BD. Para ello se supone que existe en el servidor un fichero llamado `form.html`, cuyo contenido se muestra en la Figura 1.
- `/nuevo_usuario`. Sirve para procesar la información del formulario mostrado anteriormente. Se saltará a esta URL cuando el usuario haya hecho clic en el botón `Enviar` del formulario. Al ser una petición de tipo GET, el contenido del formulario estará contenido dentro de la URL:

  `/nuevo_usuario?nombre=` nombre `&correo=` dirección de correo `&telefono=` núm. telefono

  Indicación: Examina la documentación del módulo `url`. Este módulo tiene una función `parse` que te permitirá "trocear" la URL en sus distintas componentes, así como obtener la información del formulario mediante el atributo `query`.

▷ **4. Examen febrero 2017**

En la Figura 2 se muestra el diseño relacional de una base de datos que almacena artículos de revista. Cada artículo tiene asociado un identificador numérico, un título, una fecha de publicación y una lista de palabras clave, que se encuentran en una tabla separada (`palabrasclave`).

Implementa una función `leerArticulos(callback)` que obtenga todos los artículos de la base de datos. Esta función debe construir un **array de objetos**, cada uno de ellos representando la información de un artículo mediante cuatro atributos: `id` (númerico), `titulo` (cadena de texto), `fecha` (objeto `Date`) y `palabrasClave` (array de cadenas). Por ejemplo:

```
{
  id: 1,
  titulo: "An inference algorithm for guaranteeing Safe destruction",
  fecha: 2008-07-19,  // ⟵ como objeto de la clase Date
  palabrasClave: [ 'formal', 'inference', 'memory' ]
}
```

Fichero index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ejercicio 2</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="index.css">
    </head>

    <body>
        <form method="GET" action="/nuevo_usuario">
            <div>Nombre:</div>
            <div>
                <input type="text" name="nombre">
            </div>
            <div>Correo:</div>
            <div>
                <input type="text" name="correo">
            </div>
            <div>Telefono:</div>
            <div>
                <input type="text" name="telefono">
            </div>
            <div></div>
            <div>
                <input type="submit" value="Enviar">
            </div>
        </form>
    </body>
</html>
```

Fichero index.css

```
form {
    display: grid;
    grid-template-columns: auto 1fr;
}

form > div {
    padding: 10px;
}
```

Figura 1: Formulario de inserción de usuarios y su hoja de estilo

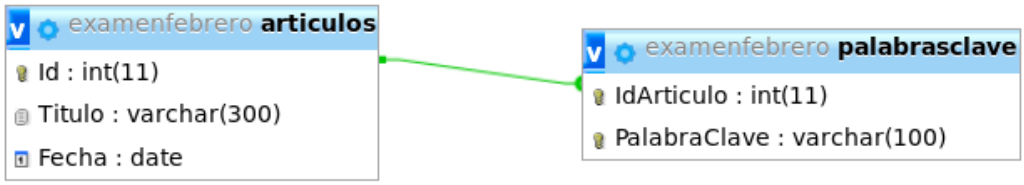


Figura 2: Diseño de la base de datos de artículos y palabras clave

La función `callback` pasada como parámetro a `leerArticulos` funciona de igual modo que las vistas en clase. Recibe dos parámetros: un objeto con información de error (en caso de producirse), y la lista con los artículos recuperados de la base de datos.