UNIVERSIDAD COMPLUTENSE
MADRID

APLICACIONES WEB
2018-2019
Grado en Ingeniería del Software
Facultad de Informática

Node.js

Ejercicios

Ejercicio 1

a. Escribir un programa **ej1a.js** que lea un fichero concreto y sustituya cualquier grupo de uno o más espacios en blanco por un único blanco. Se deben utilizar las funciones asíncronas **readFile** y **writeFile** del módulo **fs**.

b. Utilizando parte del código del apartado a., escribir un módulo **ejnode.js** que exporte la función **freplace(fichero, buscar, sustituir, callback)** que permite buscar en **fichero** las cadenas que describe la expresión regular **buscar** y sustituirlas por la cadena **sustituir**. La función callback recibe un único parámetro que vale **null** si no ha ocurrido ningún error y en caso contrario un objeto **Error** que describe el error ocurrido.
El módulo **ejnode.js** tiene que estar preparado para exportar más elementos además de la función **freplace**.
Escribir un programa **ej1b.js** para probar la función **freplace**.
Si el **fichero1.txt** tiene el siguiente contenido**:**

```
ej1b.js: Prueba número 1 de los módulos de node en octubre de 2018.
```

La llamada **freplace("fichero1.txt",/[0-9]+/g, '{numero}, callback)** transformaría el fichero en:

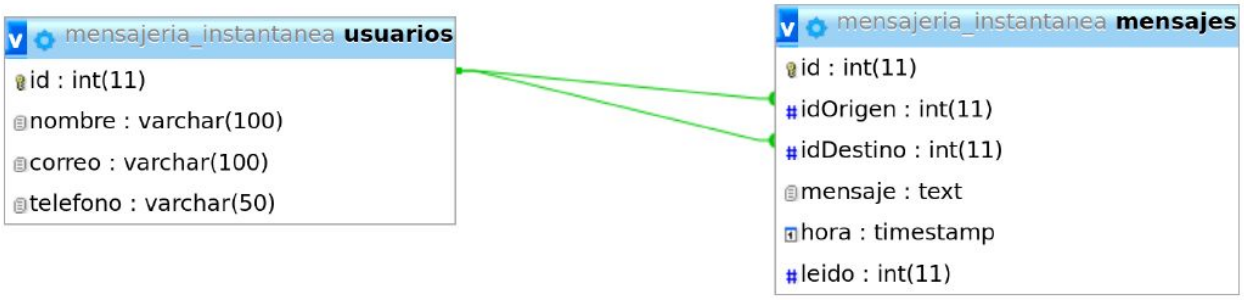
```
ej{numero}b.js: Prueba número {numero} de los módulos de node en octubre de {numero}.
```

Utilizando la expresión regular **/\b[0-9]+\b/g** , se transformaría en:

```
ej1b.js: Prueba número {numero} de los módulos de node en octubre de {numero}.
```

Ejercicio 2

En la figura se muestra el esquema relacional de un sistema de mensajería instantánea.



Como se puede ver, existen dos tablas, una de **usuarios** y otra de **mensajes**. Ambas tablas tienen como clave primaria 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 el campo **leido** (que puede tomar el valor 1 o 0) para indicar si el mensaje ha sido leído por el receptor o no.

El objetivo de este ejercicio es diseñar el módulo **dao.js** como capa de acceso a los datos. Este módulo exporta una clase **DAO** con los métodos que se describirán posteriormente. Todos ellos son asíncronos, lo que implica que recibirán como último parámetro una función callback.

En cuanto al 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.

El modelo de conexión con la base de datos será un pool de conexiones. La creación de dicho pool se puede hacer dentro del constructor llamando al método síncrono **createPool()**. Los métodos de la clase **DAO**, para acceder a la BD debe obtener previamente una conexión del pool utilizando el método **getConnection()** y liberar dicha conexión al terminar llamando al método **release()**.

Inserción de usuarios en la base de datos

El método **insertarUsuario(usuario, callback)** inserta un usuario en la BD. El **usuario** pasado como parámetro es un objeto con tres atributos: **nombre**, **correo** y **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.

La función **callback** recibirá un único parámetro con el objeto **Error** en el caso en que se produzca el mismo, o **null** si no se produce. Un posible esquema de esta función sería el siguiente:

```
function cb_insertarUsuario(err){
  if (err) {
    console.log("ERROR EN LA INSERCIÓN DE USUARIO");
```

```
    }
    else {
        console.log("USUARIO INSERTADO CORRECTAMENTE");
    }
};
```

Envío de mensajes

El método **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**.

La función **callback** recibirá un único parámetro con el objeto **Error** en el caso en que se produzca el mismo, o **null** si no se produce.

Bandeja de entrada de un usuario

El método **bandejaEntrada(usuario, callback)** recupera los mensajes no leídos del **usuario** pasado como parámetro.
La función **callback** recibirá dos argumentos. El primero es un objeto de la clase **Error** que contendrá el error si es que se ha producido, o null en caso de no producirse ningún error. El segundo argumento es un array con los 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. Un posible esquema de esta función sería el siguiente:

```
function cb_bandejaEntrada(err, mensajes){
    if (err) {
        console.log(“ERROR EN EL ACCESO A LA BANDEJA DE ENTRADA”);
    }
    else {
        // mostrar por consola el contenido del array mensajes
    }
};
```

Búsqueda de usuarios

El método **buscarUsuario(str, callback)** recupera los usuarios cuyo nombre contenga la cadena **str** pasada como parámetro.
La función **callback** recibirá dos argumentos. El primero es un objeto de la clase **Error** y el segundo un array con los usuarios que cumplen la condición de búsqueda.

Cierre del pool de conexiones

Debido a la naturaleza asíncrona del modelo, no se puede saber cuándo han terminado de ejecutarse todas las funciones callback y por lo tanto no se puede determinar el momento en el que es posible cerrar el pool de conexiones. No obstante se puede implementar en la clase **DAO** el método **terminarConexion(callback)** que cierre el pool de conexiones mediante una llamada al método **end()**. La función **callback** recibirá un único parámetro con el objeto **Error** en el caso en que se produzca el mismo, o **null** si no se produce.

Para probar el módulo **DAO** se debe programar otro modulo **main.js** que realice las llamadas a los métodos de la clase implementada. El aspecto sería algo parecido a lo siguiente:

```
const DAO = require("./dao");

const daoMensajeria = new DAO("localhost", "root", "", "mensajeria_instantanea");

// Creación de usuarios
let usuario1 = {
   nombre: “.....",
   correo: ”.....",
   telefono: "....."
};
let usuario2 = {
   nombre: “.....",
   correo: ”.....",
   telefono: "....."
};
// Definición de las funciones callback
// LLamadas a los métodos de DAO para insertar usuarios, enviar mensajes, etc
```

El módulo **dao.js** tiene la siguiente estructura:

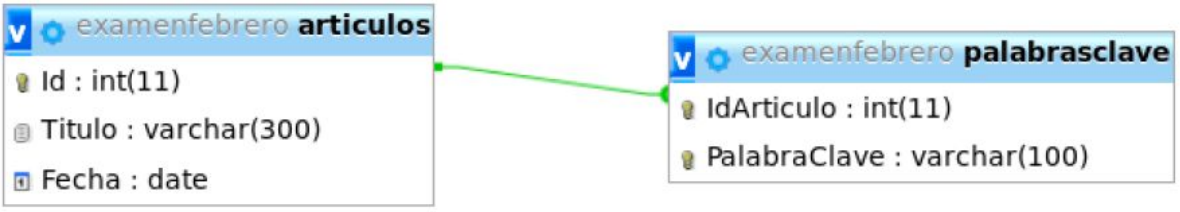
```
const mysql = require("mysql");
class DAO {
   constructor(host, user, password, database) { ... }
   insertarUsuario(usuario, callback) { ... }
   enviarMensaje(usuarioOrigen, usuarioDestino, mensaje, callback){ ... }
   bandejaEntrada(usuario, callback) { ... }
   buscarUsuario(str, callback){ ... }

   terminarConexion() { ... }
}
module.exports = DAO;
```

Ejercicio 3

Partiendo del ejercicio anterior implementar un nuevo módulo **dao_single.js** que exporte la misma funcionalidad que **dao.js** pero utilizando conexiones independientes en lugar de un pool de conexiones.

Ejercicio 4 (examen febrero 2017)

En la figura 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**).



Se pide implementar 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' ]
}
```

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.

Ejercicio 5

Utilizando el módulo implementado en el ejercicio 2, realizar una pequeña aplicación que añada usuarios a la base de datos. Para ello se utilizará el módulo **http**. El servidor web ha de poder atender dos tipos de peticiones, ambas con el método GET:

- **/index.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 index.html cuyo contenido se muestra posteriormente.
- **/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=Juan+Calvo&correo=juan.calvo%40ucm.es&telefono=678678678

Sugerencias

El módulo **url** dispone de la función **parse** que devuelve en forma de objeto el atributo **url** (que es un string) del objeto **request**. Se puede acceder a los atributos del objeto devuelto por la función **parse**. El particular, el atributo **pathname** almacena la cadena correspondiente a la url sin el nombre del servidor. Además, pasando el valor **true** al segundo parámetro de **parse** se consigue que la **query** de la petición http sea también un objeto y se pueda acceder fácilmente a sus atributos.
Por ejemplo, para la petición anterior:

/nuevo_usuario?nombre=Juan+Calvo&correo=juan.calvo%40ucm.es&telefono=678678678

dentro de la callback del servidor se pueden obtener los atributos descritos:

```
const http=require("http");
const url=require("url");
....

const servidor=http.createServer(function(request,response) {

   let method = request.method;
   console.log(method);
   →    GET
   let requestUrl = request.url;
   console.log(requestUrl);
                                                                    →
/nuevo_usuario?nombre=Juan+Calvo&correo=juan.calvo%40ucm.es&telefono=678678678
   let objetoUrl=url.parse(requestUrl,true);
   let pathname=objetoUrl.pathname;
   console.log(pathname);
   →    /nuevo_usuario
   let query = objetoUrl.query;
   console.log(query);
   →    { nombre: 'Juan Calvo',
        correo: 'juan.calvo@ucm.es',
        telefono: '678678678' }
   .....
});
```

Dentro de la callback del servidor, se comprobarían los valores de **method** y **pathname**, y en función de ellos se elaboraría la respuesta al cliente. Por ejemplo:

```
if (method === "GET" && pathname === "/index.html") {
    fs.readFile("."+ pathname, function(err, content) {
      if(err) {
        response.statusCode = 500;
        response.setHeader("Content-Type", "text/html");
        response.write("ERROR INTERNO");
        response.end();
      }
      else {
        response.statusCode = 200;
        response.setHeader("Content-Type", "text/html");
        response.write(content);
        response.end();
      }
```

Fichero **index.html**:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Ejercicio 5</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;
}
```