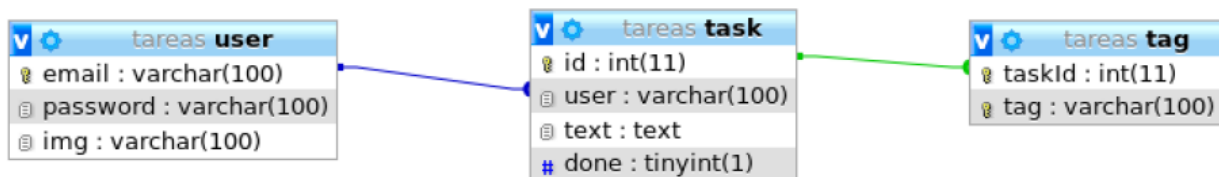


Prácticas voluntarias - Práctica 3

Fecha de entrega: 11-nov-2021 (09:00)

Esta práctica es la tercera parte de un proyecto completo consistente en una aplicación web de gestión de tareas. El objetivo de esta parte es implementar el acceso a la base de datos relacional que almacenará la información.

En la figura se muestra el esquema relacional de la base de datos.



Por un lado, la tabla **user** almacena la información de los usuarios. Para cada usuario se guarda su dirección de correo electrónico en el campo **email** (que es su clave primaria), su contraseña en **password**, y un atributo **img** con el nombre del fichero que contendrá la imagen de perfil, o **NULL** en caso de no tener imagen. Los detalles del almacenamiento de la imagen se resolverán en el futuro, de momento solo es necesario conocer que existe una columna **img** dentro de la tabla de usuarios.

Cada usuario tiene asociado un conjunto de tareas, que se almacenan en la tabla **task**. Cada tarea dispone de un texto (**text**), un bit (**done**) que indica si la tarea ha sido realizada y una lista de tags que, al ser un atributo multivaluado, se almacena en una tabla aparte (**tag**). La clave primaria de la tarea se almacena en el campo **id** que es autoincremental.

La tabla **tag** tiene dos columnas, la clave primaria autoincremental (**taskid**) y el propio texto del tag (**tag**).

Se pide implementar dos clases JavaScript: **DAOUsers** y **DAOTasks** en dos módulos **DAOUsers.js** y **DAOTasks.js**. La primera clase implementa la funcionalidad relacionada con la gestión de usuarios en la BD, y la segunda con la gestión de tareas. Se proporciona uno de los métodos de la clase DAOUsers para que sirva como ejemplo para la implementación de los demás métodos.

Los constructores de ambas clases reciben un pool de conexiones y lo almacenan en un atributo. Todas las operaciones deben **obtener una conexión del pool, ejecutar una o varias consultas paramétricas y devolver la conexión al pool.**

Los métodos de ambas clases son asíncronos como consecuencia de su implementación (ver código del método `isUserCorrect`). Esto implica que reciben como último parámetro una función callback a la que pasan los argumentos adecuados en cada caso. Como norma general, el primer argumento es un objeto de la clase **Error** que contiene el error producido al ejecutar la consulta o modificación en la BD, o **null** en caso de no producirse ningún error. El segundo parámetro, no presente en todas las funciones callback, es el resultado de la operación realizada.

La clase **DAOUsers** tiene dos métodos, uno para comprobar si un usuario es correcto y otro para obtener el nombre del fichero que contiene la imagen de perfil de un usuario.

Usuario correcto

El método **`isUserCorrect(email, password, callback)`** comprueba si en la base de datos existe un usuario cuyo identificador es **email** y su password coincide con **password**.

La función **callback** tiene dos argumentos. El primero es un objeto de la clase **Error** que contiene el error si se ha producido, o **null** en caso contrario. El segundo es un booleano indicando el resultado de la operación (**true**: el usuario existe con la contraseña pasada por parámetro, **false**: el usuario no existe o la contraseña es incorrecta). Los errores que puede recibir la función callback son los siguientes:

- "Error de conexión a la base de datos": falla la solicitud de conexión al pool.
- "Error de acceso a la base de datos": falla la consulta a la base de datos.

El código del método **`isUserCorrect`** sería el siguiente:

```
isUserCorrect(email, password, callback) {

  this.pool.getConnection(function(err, connection) {
    if (err) {
      callback(new Error("Error de conexión a la base de datos"));
    }
    else {
      connection.query("SELECT * FROM user WHERE email = ? AND password = ?",
        [email,password],
        function(err, rows) {
          connection.release(); // devolver al pool la conexión
          if (err) {
            callback(new Error("Error de acceso a la base de datos"));
          }
          else {
            if (rows.length === 0) {
```

```

        callback(null, false); //no está el usuario con el password proporcionado
    }
    else {
        callback(null, true);
    }
}
});
}
}
);
}
}

```

Un ejemplo de uso es el siguiente:

```

daoUser.isUserCorrect("usuario@ucm.es", "mipass", cb_isUserCorrect);

function cb_isUserCorrect(err, result){
    if (err) {
        console.log(err.message);
    } else if (result) {
        console.log("Usuario y contraseña correctos");
    } else {
        console.log("Usuario y/o contraseña incorrectos");
    }
}
}

```

Imagen de perfil de usuario

El método **getUserImageName(email, callback)** obtiene el nombre de fichero que contiene la imagen de perfil de un usuario cuyo identificador en la base de datos es **email**.

La función **callback** tiene dos argumentos. El primero es un objeto de la clase **Error** que contiene el error si se ha producido, o **null** en caso contrario. El segundo es una cadena con el nombre del fichero que contiene la imagen de perfil. Los errores que puede recibir la función callback son los siguientes:

- "Error de conexión a la base de datos": falla la solicitud de conexión al pool.
- "Error de acceso a la base de datos": falla la consulta a la base de datos.
- "No existe el usuario": el resultado de la consulta está vacío.

De acuerdo a la descripción anterior, el módulo **DAOUsers.js** tendría la siguiente estructura:

```
"use strict";

class DAOUsers {
    constructor(pool) { ... }
    isUserCorrect(email, password, callback) { ... }
    getUserImageName(email, callback) { ... }
}
module.exports = DAOUsers;
```

La clase **DAOTasks** tiene los cuatro métodos que se describen a continuación.

Tareas de usuario

El método **getAllTasks(email, callback)** devuelve todas las tareas asociadas a un determinado usuario de la BD junto con los tags asociados a cada una de ellas. En la implementación de este método se debe utilizar una única consulta que relacione las tablas tasks y tag, y reconstruir el resultado a partir de esta consulta. El resultado de esta operación es un array de tareas, siendo cada una de ellas un objeto con cuatro propiedades: **id**, **text**, **done** y **tags**. La última de ellas, **tags**, es un array con las etiquetas asociadas a la tarea.

La función **callback** tiene dos argumentos. El primero es un objeto de la clase **Error** que contiene el error si se ha producido, o **null** en caso contrario. El segundo es el array de tareas. Los errores que puede recibir la función callback son los siguientes:

- "Error de conexión a la base de datos": falla la solicitud de conexión al pool.
- "Error de acceso a la base de datos": falla la consulta a la base de datos.

Nuevas tareas

El método **insertTask(email, task, callback)** inserta en la BD la tarea **task** asociándola al usuario cuyo identificador es **email**. La tarea recibida como parámetro es un objeto que contiene tres atributos: **text**, **done** y **tags**. El último de ellos, **tags**, es un array con las etiquetas asociadas a la tarea.

Este método se puede implementar mediante dos consultas en la BD: una para insertar en la tabla **task**, y otra para insertar las etiquetas en la tabla **tag**. Para construir esta última consulta se puede utilizar la inserción simultánea de varias filas en la BD mediante una única sentencia INSERT:

```
INSERT INTO tag (taskId, tag) VALUES (?, ?), (?, ?), (?, ?), ..
```

La función callback recibirá en este caso, un único parámetro con el objeto **Error**, en caso de producirse. Los errores que pueden producirse son los siguientes:

- "Error de conexión a la base de datos": falla la solicitud de conexión al pool.
- "Error de acceso a la base de datos": falla la consulta a la base de datos.

Tareas finalizadas

El método **markTaskDone(idTask, callback)** marca la tarea **idTask** como realizada actualizando en la base de datos la columna **done** a **true**. El parámetro **idTask** es el identificador de la tarea dentro de la base de datos. La función callback recibirá un único parámetro con el objeto **Error**, en caso de producirse. Los errores que pueden producirse son los siguientes:

- "Error de conexión a la base de datos": falla la solicitud de conexión al pool.
- "Error de acceso a la base de datos": falla la consulta a la base de datos.

Borrar tareas

El método **deleteCompleted(email, callback)** elimina todas las tareas asociadas al usuario cuyo correo es **email** y que tengan el valor **true** en la columna **done**. La función callback recibirá un único parámetro con el objeto **Error**, en caso de producirse. Los errores que pueden producirse son los siguientes:

- "Error de conexión a la base de datos": falla la solicitud de conexión al pool.
- "Error de acceso a la base de datos": falla la consulta a la base de datos.

De acuerdo a la descripción anterior, el módulo **DAOTasks.js** tendría la siguiente estructura:

```
"use strict";

class DAOTasks {
  constructor(pool) { ... }
  getAllTasks(email, callback) { ... }
  insertTask(email, task, callback) { ... }
  markTaskDone(idTask, callback) { ... }
  deleteCompleted(email, callback) { ... }
```

```
}  
module.exports = DAOTasks;
```

La configuración del acceso a la base de datos se puede encapsular en un módulo **config.js** que tendría la siguiente estructura:

```
"use strict";  
  
module.exports = {  
  host: "localhost", // Ordenador que ejecuta el SGBD  
  user: "root", // Usuario que accede a la BD  
  password: "", // Contraseña con la que se accede a la BD  
  database: "tarefas" // Nombre de la base de datos  
}
```

Por último, para la realización de pruebas es necesario programar un módulo **main.js** que podría tener la siguiente estructura:

```
"use strict";  
  
const mysql = require("mysql");  
const config = require("./config");  
const DAOUsers = require("./DAOUsers");  
const DAOTasks = require("./DAOTasks");  
  
// Crear el pool de conexiones  
const pool = mysql.createPool({  
  host: config.host,  
  user: config.user,  
  password: config.password,  
  database: config.database  
});  
  
let daoUser = new DAOUsers(pool);  
let daoTask = new DAOTasks(pool);  
  
// Definición de las funciones callback  
// Uso de los métodos de las clases DAOUsers y DAOTasks
```

Entrega

La fecha límite para entregar el trabajo es el 11 de noviembre de 2020 (el enlace para la entrega en el campus virtual se cierra a las **09:00**).

En el campus se entregará un fichero **pv3_gxx.zip** (xx es el número de grupo de laboratorio) que contenga los módulos **main.js**, **DAOUsers.js** y **DAOTasks.js**.

Es imprescindible que se respeten los nombres de los módulos, métodos, etc, descritos en este enunciado. Las entregas que no lo hagan se considerarán como NO ENTREGADAS.