Ejercicio Entregable 6

Gestión de tareas mediante Express.js

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

El ejercicio de esta hoja es entregable y puede realizarse en el laboratorio. Los ejercicios entregables realizados a lo largo del curso computan el 10 % de la nota final, pero su entrega no es obligatoria para aprobar la asignatura. La entrega se realiza por parejas.

Fecha límite de entrega: 30 de noviembre de 2017.

La entrega se realizará a través del Campus Virtual.

En este ejercicio integraremos los resultados de algunos de los ejercicios anteriores con el fin de desarrollar una aplicación web que gestione una lista de tareas. Aquí nos restringiremos, de momento, a la gestión de tareas en sí: mostrar las tareas disponibles, añadir nuevas tareas, eliminar las tareas completadas, y marcar tareas como completadas. La gestión de usuarios y sesiones se tratará en otro ejercicio entregable posterior.

Si no has completado alguna de las tareas entregables anteriores, no te preocupes. En el Campus Virtual se proporciona la implementación de todos los componentes (HTML, CSS y JS) necesarios para realizar este ejercicio[1]. No obstante, si has realizado los ejercicios anteriores puedes utilizar tus propios ficheros.

1 Preparación del proyecto

En el Campus Virtual encontrarás un proyecto Node.js que contiene, entre otros ficheros, los recursos mencionados anteriormente. Además encontrarás:

1. Un fichero `.sql` que genera la base de datos de tareas. Carga esta base de datos en MySQL si no lo has hecho en la entrega anterior. Para ello debes crear una base de datos vacía e importar el fichero `.sql` dentro de la misma.

2. Un fichero `config.js` que puedes modificar para indicar el nombre de la base de datos, la información de acceso a MySQL, y el puerto TCP en el que arrancar el servidor.

3. Un fichero `package.json` con las dependencias del proyecto. Utiliza `npm install` para descargar dichas dependencias.

4. Un fichero `main.js` que crea un servidor *Express.js* y lo arranca en el puerto indicado en `config.js`. Además crea un *pool* de conexiones a la base de datos de MySQL y crea una instancia de `DAOTasks`.

Antes de continuar, ejecuta el fichero `main.js` para comprobar que el servidor se inicia correctamente. En lugar de ejecutarlo mediante `node main.js`, puedes utilizar `nodemon main.js` para que el servidor

[1] Los ficheros `.js` están ofuscados, porque no quiero proporcionar la solución de ejercicios entregables anteriores.

se reinicie automáticamente cada vez que se realicen cambios en alguno de los ficheros. Si no tienes `nodemon` instalado, puedes instalarlo mediante el comando `npm install -g nodemon`.

2 Recursos estáticos

Añade el *middleware* `static` a la aplicación web, con el fin de que esta pueda proporcionar los recursos estáticos de la aplicación al cliente. Estos recursos se encuentran en el directorio `public`. Con el servidor arrancado intenta acceder a la dirección `http://localhost:3000/tasks.html` mediante el navegador web para verificar que el servidor puede acceder a estos recursos y devolvérselos al cliente.

3 Listado de tareas

A continuación vamos a transformar el fichero estático que contiene la lista de tareas en una plantilla EJS. Para ello mueve el fichero `tasks.html` a la carpeta `views` y renómbralo como `tasks.ejs`. Modifica este último fichero para que, al renderizar su contenido, se muestre la lista de tareas almacenada en variable `taskList`. Suponemos que esta variable contine un array de tareas en el que cada tarea es un objeto como los vistos entregas anteriores (es decir, un objeto con atributos `id`, `text`, `done`, y `tags`). De momento, preocúpate solamente del texto y las etiquetas de cada tarea, teniendo en cuenta que las tareas finalizadas tienen un estilo CSS distinto al de las no finalizadas. Puedes ignorar el botón `[Marcar finalizada]`.

A continuación modifica el fichero `main.js` e introduce un manejador para la ruta `/tasks`. Este ha de obtener, mediante el DAO de tareas, la lista de tareas correspondiente al usuario `usuario@ucm.es`[2], y mostrar la vista `tareas`, pasando dicha lista como modelo.

4 Añadir tareas

Al final de la lista de tareas tenemos un cuadro de texto y un botón `[Añadir]` que permite insertar una tarea en la base de datos. Al pulsar el botón se realiza una petición `POST` a la ruta `/addTask`. Implementa el manejador de esta ruta para que se añada la tarea a la base de datos y se redirija a la URL `/tasks`. La tarea recién creada debe estar asociada al usuario con identificador `usuario@ucm.es`. No te olvides de incluir el middleware `body-parser` para que puedas obtener la información del formulario a partir del cuerpo de la petición HTTP de tipo `POST`.

El texto introducido en el formulario puede contener etiquetas de la forma `@etiqueta`. Puedes utilizar la función que hiciste en el Ejercicio entregable 3 para obtener un objeto tarea a partir de este texto. Alternativamente, puedes utilizar la función `createTask` del módulo `task_utils.js` (ver Apéndice).

5 Marcar tarea como finalizada

Al lado de cada tarea no completada tenemos un botón `<input type="submit">` que permite marcar una tarea como finalizada. Introduce etiquetas `<form>` en la plantilla EJS para que cada uno de estos botones esté contenido dentro de su propio formulario. De este modo, al pulsar el botón, el navegador debería realizar una petición `POST` a la URL `/finish`.

El siguiente paso será crear, dentro de `main.js`, el manejador de ruta `/finish`. Sin embargo, al implementar este manejador te darás cuenta que nos falta información. Necesitamos saber el identificador

[2] En el siguiente ejercicio entregable modificaremos esto para que se obtengan las tareas del usuario actualmente identificado en el sistema.

de la tarea que debemos marcar como completada. Esta información debe provenir desde el formulario en el que se encuentra el botón pulsado. Para ello añade al formulario de cada tarea un elemento `<input>` de tipo `hidden` que contenga el identificador de la tarea correspondiente, para que así el manejador de ruta `/finish` pueda acceder al mismo y saber qué tarea debe marcarse. Una vez realizada la operación correspondiente a la base de datos, el manejador debe redirigir a la dirección `/tasks`.

6 Eliminar tareas completadas

Al pulsar el enlace de la parte inferior de la página `[Eliminar tareas completadas]`, el navegador salta a la dirección `/deleteCompleted`. Implementa al manejador de esta ruta para que se eliminen de la base de datos aquellas tareas que hayan sido marcadas como finalizadas y redirija a la URL `/tasks`.

7 Apéndice: módulos auxiliares

Fichero `dao_tasks.js`:

```
/**
 * Proporciona operaciones para la gestión de tareas
 * en la base de datos.
 */
class DAOTasks {
    /**
     * Inicializa el DAO de tareas.
     *
     * @param {Pool} pool Pool de conexiones MySQL. Todas las operaciones
     *                    sobre la BD se realizarán sobre este pool.
     */
    constructor(pool) { ... }


    /**
     * Devuelve todas las tareas de un determinado usuario.
     *
     * Este método devolverá (de manera asíncrona) un array
     * con las tareas de dicho usuario. Cada tarea debe tener cuatro
     * atributos: id, text, done y tags. El primero es numérico, el segundo
     * una cadena, el tercero un booleano, y el cuarto un array de cadenas.
     *
     * La función callback ha de tener dos parámetros: un objeto
     * de tipo Error (si se produce, o null en caso contrario), y
     * la lista de tareas (o undefined, en caso de error).
     *
     * @param {string} email Identificador del usuario.
     * @param {function} callback Función callback.
     */
    getAllTasks(email, callback) { ... }
```

```
    /**
     * Inserta una tarea asociada a un usuario.
     *
     * Se supone que la tarea a insertar es un objeto con, al menos,
     * dos atributos: text y tags. El primero de ellos es un string con
     * el texto de la tarea, y el segundo de ellos es un array de cadenas.
     *
     * Tras la inserción se llamará a la función callback, pasándole el objeto
     * Error, si se produjo alguno durante la inserción, o null en caso contrario.
     *
     * @param {string} email Identificador del usuario
     * @param {object} task Tarea a insertar
     * @param {function} callback Función callback que será llamada tras la inserción
     */
    insertTask(email, task, callback) { ... }

    /**
     * Marca la tarea indicada como realizada, estableciendo
     * la columna 'done' a 'true'.
     *
     * Tras la actualización se llamará a la función callback, pasándole el objeto
     * Error, si se produjo alguno durante la actualización, o null en caso contrario.
     *
     * @param {object} idTask Identificador de la tarea a modificar
    * @param {function} callback Función callback que será llamada tras la actualización
     */
    markTaskDone(idTask, callback) { ... }

    /**
     * Elimina todas las tareas asociadas a un usuario dado que tengan
     * el valor 'true' en la columna 'done'.
     *
     * Tras el borrado se llamará a la función callback, pasándole el objeto
     * Error, si se produjo alguno durante la actualización, o null en caso contrario.
     *
     * @param {string} email Identificador del usuario
     * @param {function} callback Función llamada tras el borrado
     */
    deleteCompleted(email, callback) { ... }
}

module.exports = {
    DAOTasks: DAOTasks
}
```

Fichero task_utils.js

```
/**
 * A partir de un texto seguido de una serie de etiquetas de la forma
 * @tag1, @tag2, etc., extrae las etiquetas, creando un objeto tarea
 * como resultado.
 *
 * El objeto devuelto contiene dos atributos:
 *   - tags: un array con las etiquetas de la tarea.
 *   - text: la cadena de texto de entrada sin las etiquetas.
 *
 * @param {string} text Texto de la tarea, incluyendo etiquetas
 */
function createTask(text) { ... }


module.exports = {
  createTask: createTask
}
```