UNIVERSIDAD COMPLUTENSE MADRID

APLICACIONES WEB
2019-2020
Grado en Ingeniería del Software
Facultad de Informática

Práctica guiada - Parte 4

Fecha de entrega: 28-nov-2019

El objetivo de esta parte de la práctica guiada es integrar los resultados de algunas partes previas. Esta integración se refiere únicamente a la **gestión de tareas: mostrar las tareas disponibles, añadir nuevas tareas, marcar tareas como completadas y eliminar las tareas completadas**. La gestión de usuarios y sesiones se tratará en la siguiente parte.

Creación del proyecto

Antes de comenzar a desarrollar el servidor, se debe crear la estructura del proyecto de la siguiente manera:

1. Crear un directorio para el proyecto.
2. Crear un proyecto node con el comando **npm init** indicando como punto de entrada el fichero **app.js**. En el directorio del proyecto se creará automáticamente el fichero **package.json**.
3. Instalar en el proyecto los módulos **mysql**, **express**, **ejs** y **body-parser**. Al terminar la instalación de los módulos se habrá creado el directorio **node_modules** dentro del directorio del proyecto.
4. Dentro del directorio del proyecto crear un directorio **views** para almacenar las plantillas.
5. Dentro del directorio del proyecto crear un directorio **public**, y dentro de él los directorios **css** e **img**. En el directorio **public** se guardarán los ficheros html, en **css** las hojas de estilo y en **img** las imágenes.
6. Crear en el directorio del proyecto el fichero **config.js** similar al de la parte 3 de la práctica guiada pero añadiendo también el puerto en el que arranca el servidor. Este fichero podría tener el siguiente contenido:

```
// config.js
"use strict";
module.exports = {
  mysqlConfig: {
        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: "tareas"     // Nombre de la base de datos
  },
    port: 3000                 // Puerto en el que escucha el servidor
}
```

7. Copiar en el directorio del proyecto el módulo **DAOTasks.js** desarrollado en la parte 3 de la práctica guiada.
8. Recuperar de la parte 1 de esta práctica guiada todos los ficheros (html, css, png, etc) ubicándolos en sus correspondientes directorios.
9. Recuperar de la parte 2 de esta práctica guiada la funciones implementadas y agruparlas en un módulo que las exporte. Dicho módulo se llamará **utils.js** y se ubicará en el directorio del proyecto.
10. Dentro del directorio del proyecto crear el fichero **app.js** que debe realizar las siguientes acciones:
    - Crear un servidor Express.js
    - Crear un pool de conexiones a la base de datos de MySQL.
    - Crear una instancia de **DAOTasks**.
    - Arrancar el servidor en el puerto indicado en **config.js**.

En la figura se muestra el aspecto de este fichero.

```
// app.js
const config = require("./config");
const DAOTasks = require("./DAOTasks");
const utils = require("./utils");
const path = require("path");
const mysql = require("mysql");
const express = require("express");
const bodyParser = require("body-parser");
const fs = require("fs");

// Crear un servidor Express.js
const app = express();

// Crear un pool de conexiones a la base de datos de MySQL
const pool = mysql.createPool(config.mysqlConfig);

// Crear una instancia de DAOTasks
const daoT = new DAOTasks(pool);

// Arrancar el servidor
app.listen(config.port, function(err) {
  if (err) {
     console.log("ERROR al iniciar el servidor");
  }
  else {
     console.log(`Servidor arrancado en el puerto ${config.port}`);
  }
});
```

11. Crear la base de datos del proyecto (en el CV hay disponible un script .sql para generar el esquema pero no añade ningún registro, sería necesario añadir registros, es decir, tareas con sus tags, para un hipotético usuario cuyo email sea "usuario@ucm.es").

Antes de continuar, ejecutar el fichero **app.js** para comprobar que el servidor se inicia correctamente. Si se prefiere, en lugar de ejecutarlo mediante **node app.js** se puede utilizar **nodemon app.js** para que el servidor se reinicie automáticamente cada vez que se realicen cambios en alguno de los ficheros. Se puede instalar **nodemon** con el comando **npm install -g nodemon.**

Recursos estáticos

Añadir el middleware **static** a la aplicación web, con el fin de que ésta pueda proporcionar los recursos estáticos de la aplicación al cliente. Los recursos estáticos se encuentran en el directorio **public**. Con el servidor arrancado intentar acceder a la dirección **http://localhost:3000/tasks.html** mediante el navegador web para verificar que el servidor puede acceder a estos recursos y devolverlos al cliente (**tasks.html** es la página web desarrollada en la parte 1 de esta práctica guiada. Se debe situar en el directorio **public**, los ficheros **css** en el directorio **css** y las imágenes en el directorio **img**).

Listado de tareas

El objetivo de este apartado es transformar el fichero estático que contiene la lista de tareas en una plantilla EJS. Para ello, en primer lugar se debe mover el fichero **tasks.html** a la carpeta **views** y renombrarlo como **tasks.ejs.** Posteriormente modificar este último fichero para que, al renderizar su contenido, se muestre la lista de tareas almacenada en la variable **taskList.** Se supone que esta variable contiene un array de tareas en el que cada tarea es un objeto como los vistos en las partes anteriores de esta práctica guiada (es decir, un objeto con atributos **id**, **text**, **done**, y **tags**). En este apartado, sólo es necesario mostrar correctamente el texto y las etiquetas de cada tarea, teniendo en cuenta que las tareas finalizadas tienen un estilo CSS distinto al de las no finalizadas. Por el momento, se puede ignorar el botón **[Marcar finalizada]**.
A continuación modificar el fichero **app.js** e introducir 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 y mostrar la vista **tasks.ejs** pasando dicha lista como modelo.
En la siguiente parte de esta práctica guiada se modificará el manejador para la ruta **/tasks** para que se obtengan las tareas del usuario actualmente identificado en el sistema (en lugar del usuario fijo "usuario@ucm.es" que se utiliza en esta parte).

Añadir tareas

La página web tiene un enlace **[Añadir]** que permite insertar una tarea nueva en la base de datos. Este enlace está situado al lado de un campo de texto para la descripción de la tarea. En primer lugar, editar el fichero **tasks.ejs** para sustituir dicho enlace y el campo de texto asociado por un formulario con las siguientes características:

- tiene un elemento de tipo "text" para la descripción de la tarea

- tiene un elemento de tipo “submit” que sustituye al enlace **[Añadir]**
- **method = ”POST”**
- **action = “/addTask”**

La creación del nuevo formulario probablemente implique algún ajuste en el fichero css de la página web.

Al pulsar en el botón **Añadir** se realiza una petición **POST** a la ruta **/addTask**. Implementar el manejador de esta ruta para que se añada la tarea a la base de datos y posteriormente se redirija a la URL **/tasks**. La tarea recién creada debe estar asociada al usuario con identificador **usuario@ucm.es**. El middleware **body-parser** instalado previamente permite 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**. Se puede utilizar la función **createTask()** codificada en la parte 2 de esta práctica guiada para obtener un objeto tarea a partir de este texto.

Marcar tarea como finalizada

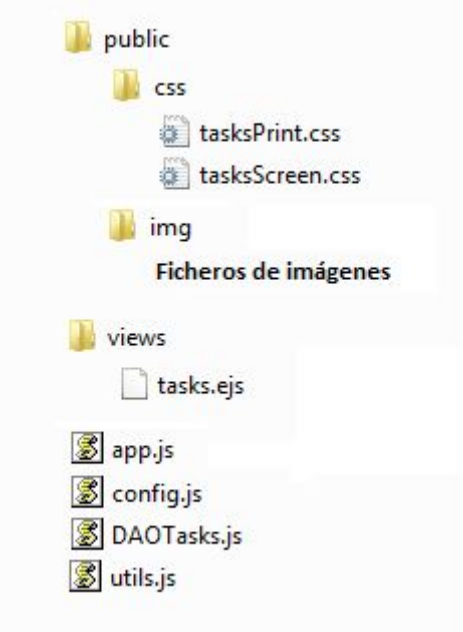
Al lado de cada tarea no completada hay un enlace **[Marcar finalizada]** que permite marcar una tarea como finalizada. Al pulsar el enlace, el navegador debería saltar a la dirección del enlace, por ejemplo saltar a **/finish**. Por lo tanto, el siguiente paso sería crear, dentro de **app.js** el manejador de ruta **/finish**. Sin embargo, al implementar este manejador se observa que falta información, en concreto el identificador de la tarea que se quiere marcar como completada. Si se hace uso de las **rutas paramétricas**, la solución es sencilla. En primer lugar, se debe modificar la plantilla **tasks.ejs** para que en cada tarea no finalizada, el campo **href** del enlace **[Marcar finalizada]** contenga el id de la tarea, es decir **href =” /finish/taskId”**. En segundo lugar se añade dentro de **app.js** el manejador de ruta **/finish/:taskId**.
Una vez realizada la operación correspondiente a la base de datos, el manejador debe redirigir a la dirección **/tasks**.

Eliminar tareas completadas

Al pulsar el enlace de la página **[Eliminar tareas completadas]**, el navegador salta a la dirección **/deleteCompleted**. Implementar el 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**.

Entrega

La fecha límite para la entrega de esta práctica es el jueves 28 de noviembre de 2019 (16:00). Se entregará a través del CV en un fichero **gxx.zip** (siendo xx el número de grupo de laboratorio) que contenga todos los ficheros necesarios para reconstruir el proyecto organizados en los directorios adecuados. En la siguiente imagen se muestra qué estructura de directorios y ficheros deben ser entregados.



Como se puede ver en la imagen, el directorio **node_modules** NO es necesario entregarlo.

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