PRÁCTICA 2

Servicios Web Rest

La finalidad de esta práctica es introducirse en el empleo de los servicios web en concreto los servicios Rest. En concreto, se trabajará sobre el framework nodejs para el desarrollo de los servicios web. Este framework nos permitirá crear servicios web Rest, de formas sencilla y con un bajo consumo de recursos.

El objetivo de la práctica es familiarizarse con el uso de servicios web mediante el framework nodejs. Para ello primero se realizarán unos ejercicios básicos, comprobando de esta forma el funcionamiento de los servicios web y posteriormente se propone un ejercicio, con una parte básica y varios puntos adicionales.

Para la realización de esta, necesitaremos de nodejs, express, mysql y visual studio 2017 con las Visual Studio Tools para Apache Cordova.

Ejemplo 1. Nodejs Hola Mundo

Para la creación de la primera aplicación “Hola Mundo” vamos a realizar los siguientes pasos:

El primer paso será comprobar que tenemos instalado nodejs. Para ello abrimos un terminal de CMD y escribimos “node –v” para así comprobar que disponemos de nodejs y de la última versión, en caso de no disponer de nodejs o no tener la último versión, accederemos a: https://nodejs.org/es/ para descargar e instalar nodejs.

A continuación, mediante un editor de texto escribimos el siguiente código y lo guardamos como **Ejemplo1.js** (notepad++ por ejemplo):

```
var http = require('http');
var server = http.createServer();
function control(petic, resp) {
        resp.writeHead(200, {'content-type': 'text/plain'});
        resp.write('Hola, Mundo!');
        resp.end();
}
server.on('request', control);
server.listen(8080);
console.log('Servidor iniciado en puerto 8080…');
```

En la primera línea, importaremos los módulos necesarios, posteriormente asignamos a la variable server la inicialización del módulo http.

Tras esto, definiremos una función, que es la que se encargará de "manejar" y controlar las peticiones y respuestas hacia y desde el servidor. Esta función, tendrá dos parámetros, que serán "petic" (petición) y "resp" (respuesta).

Con el objeto de respuesta (resp), nos podremos comunicar con el cliente y, lo primero que haremos, será mandar la cabecera HTTP con una confirmación correcta (200) y con el tipo de contenido que vamos a mandar. Además, mandaremos un mensaje de "Hola, Mundo!" hacia el cliente, y finalizaremos la respuesta.

En la línea `server.on('request', control);` llamamos a la función control y posteriormente, inicializamos el servidor e indicamos que el servidor está lanzado en la consola.

Para comprobar el funcionamiento:

```
node Ejemplo1.js
```

Y probamos a acceder desde el navegador al recurso expuesto:

http://localhost:8080.

Ejemplo 2. Diseño API Rest

En este ejemplo vamos a diseñar un API Rest con varias operaciones típicas a emplear en una aplicación real.

Las operaciones a diseñar serán:

Operación	Función	Petición	Url	Parameters		
				URL	Query	Body
Read	GetAll	GET	./items			
	GetFiltered	GET	./items?filter=ABC		Filter	
	GetById	GET	./id	Id		
Create	Add	POST	./items			Item
Update	UpdateById	PUT	./items			Id, Item
Delete	DeleteById	DELETE	./items/id	Id		

Estas serán las típicas operaciones CRUD realizadas sobre una base de datos.

Crearemos una carpeta que llamaremos, ApiRestDISM. Ahora instalamos el módulo **Express.js**. Para ello, desde la consola de comandos, nos desplazamos al directorio que hemos creado previamente y escribimos:

```
npm install expres
```

Una vez finalizado la instalación de Express, creamos un nuevo fichero que llamaremos, Ejemplo2.js. Dentro de este fichero copiamos el siguiente contenido:

```
// Cargar modulos y crear nueva aplicacion
var express = require("express");
var app = express();
var bodyParser = require('body-parser');
app.use(bodyParser.json()); // soporte para bodies codificados en jsonsupport
app.use(bodyParser.urlencoded({ extended: true })); // soporte para bodies codificados

//Ejemplo: GET http://localhost:8080/items
app.get('/items', function(req, res, next) {
  if(req.query.filter) {
   next();
   return;
  }
  res.send('Get all');
});

//Ejemplo: GET http://localhost:8080/items?filter=ABC
app.get('/items', function(req, res) {
  var filter = req.query.filter;
  res.send('Get filter ' + filter);
});

//Ejemplo: GET http://localhost:8080/items/10
app.get('/items/:id', function(req, res, next) {
  var itemId = req.params.id;
  res.send('Get ' + req.params.id);
});

//Ejemplo: POST http://localhost:8080/items
app.post('/items', function(req, res) {
   var data = req.body.data;
   res.send('Add ' + data);
});

//Ejemplo: PUT http://localhost:8080/items
app.put('/items', function(req, res) {
   var itemId = req.body.id;
   var data = req.body.data;
   res.send('Update ' + itemId + ' with ' + data);
});

//Ejemplo: DELETE http://localhost:8080/items
app.delete('/items/:id', function(req, res) {
   var itemId = req.params.id;
   res.send('Delete ' + itemId);
});

var server = app.listen(8080, function () {
    console.log('Servidor iniciado en puerto 8080...');
});
```

En primer lugar, hemos cargado los módulos necesarios, y creado una aplicación de Express.

En el resto del código, estamos definiendo distintas rutas para nuestra aplicación, según la definición que hemos hecho en el apartado anterior.

En estas funciones, empleamos el objeto ‘request’ para obtener los parámetros que hemos pasado al API en la petición. En función del tipo de request, usamos la propiedad oportuna (params, query o body).

Por otro lado, empleamos el objeto ‘response’ para devolver los resultados al cliente.

En este ejemplo, simplemente estamos devolviendo un texto informando de que la petición correspondiente se ha recibido correctamente. En una aplicación real se realizarían consultas a una base de datos, operaciones con los datos, validaciones, etc... Finalmente, en el último bloque de código iniciamos la aplicación poniéndola a escuchar peticiones en localhost, puerto 8080.

Para comprobar el funcionamiento:

```
node Ejemplo2.js
```

Y probamos a acceder desde el navegador a cualquier recurso de los expuestos:

http://localhost:8080/items
http://localhost:8080/items?filter=ABC
http://localhost:8080/items/10

Ejemplo 3. Acceso a Base de Datos desde API Rest

A continuación, vamos a crear una API Rest, que accede a una base de datos MySQL. Lo primero que debemos de realizar es crear la base de datos, para ello emplearemos el script de BD adjunto, en el cual creamos una BD llamada DISM que contiene una tabla llamada Usuarios.

Ahora debemos de instalar el paquete *Express*, *Mysql* y *Cors* para nodejs mediante npm:

```
npm install mysql
```
https://www.npmjs.com/package/mysql
```
npm install cors
```
https://www.npmjs.com/package/cors
```
npm install express
```
https://www.npmjs.com/package/express

Y ahora procedemos a escribir el ejemplo y lo guardamos como **Ejemplo3.js**:

```
var express = require("express");
var mysql      = require('mysql');
var app = express();
var bp = require('body-parser');
const cors = require('cors');
app.use(cors());
app.options('*', cors());

app.use(bp.json());

var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : 'root',
  database : 'dism'
```

```
});

//Ejemplo: GET http://localhost:8080/usuarios
app.get('/usuarios', function(req, resp) {
        connection.query('select * from usuarios', function(err, rows) {
                if (err) {
                                console.log('Error en /usuarios '+err);
                                resp.status(500);
                                resp.send({message: "Error al obtener usuarios"});
                }
                else {
                        console.log('/usuarios');
            resp.status(200);
                        resp.send(rows);
                }
        })
});

var server = app.listen(8080, function () {
    console.log('Servidor iniciado en puerto 8080…');
});
```

Para comprobar el funcionamiento:

```
node Ejemplo3.js
```

Y probamos a acceder desde el navegador al recurso expuesto:

http://localhost:8080/usuarios

NOTA: Para MySQL 8.0, se debe de crear un usuario nuevo, en nuestro caso crearemos una llamado *usuario* con contraseña *clave* y le daremos permisos:

```
CREATE USER ‘usuario’@’localhost’ IDENTIFIED WITH mysql_native_password BY ‘clave’;
GRANT ALL ON dism.* TO ‘usuario’@’localhost’;
```

Eso lo realizaremos desde MySQL WorkBench.

Enunciado Práctica

Haciendo uso de los ejemplos guiados anteriores, vamos a proceder a crear un catálogo de API Rest, el cual será el siguiente:

http://localhost/municipios
http://localhost/estaciones
http://localhost/observacion

Las 3 APIs anteriores sustituirán a las de la web de AEMET empleadas en la práctica anterior, por la correspondiente en esta y estarán asociados a una tabla de base de datos de mysql, con los campos necesarios para que cada recurso expuesto sea igual al de AEMET (el alumno deberá de crear en mysql la estructura necesaria).

Como parte **adicional** dispondremos de una cuarta API, la cual será: http://localhost/introducirDatos y se encargará al invocarla, de traer los datos de AEMET e introducirlos en las diferentes tablas creadas en el apartado anterior en Mysql, para posteriormente poder consumirlas.

En caso de no realizar la parte adicional, introduciremos varios registros de prueba en cada tabla, para comprobar el correcto funcionamiento.

Finalmente se deberá modificar la aplicación creada en la Práctica 1 para que haga uso de la API Rest nuestra en lugar de la de AEMET.

Incluiremos un botón nuevo en la aplicación para llamar a la API /introducirDatos (en caso de realizar dicho apartado).

Se valorará:

- Escritura de "código fuente legible", modularizando, estructurando y detallando el código fuente de nodejs.
- El uso de una apikey propia para acceder a los diferentes recursos, validando está con la base de datos creada.
- Se valorará el trabajo diario del alumno, valorando su trabajo en clase, así como la capacidad de resolución de los problemas que se presentan en la práctica y no sean resueltos por el profesor.

9. Entrega y Corrección

- Se debe entregar, por el Campus Virtual, mediante una entrega de práctica, el Proyecto Visual Studio 2017 creado, así como los ficheros nodejs y scripts de base de datos, comprimido en ZIP, antes del 14-11-2018.
- Se realizará una demo con la aplicación desarrollada al profesor en el laboratorio, el 14-11-2018 y el 15-11-2018, en los turnos asignados al alumno, contestando correctamente a las preguntas realizadas por el profesor.
- La práctica es de carácter individual.
- Si se constata que el código desarrollado no es original (por tanto, es una copia), la práctica se calificará con un 0.