

| | |
|--------------|--------------------------------------|
| SD | Sistemas Distribuidos |
| 24/25 | Servicios web – API REST con Node.js |

Descripción

El objetivo de este tutorial es mostrar la forma de realizar un API REST básica usando Node.js. Para la realización de los ejemplos y la ejecución de comandos se va a utilizar la distribución “Ubuntu 16” proporcionada por la Escuela Politécnica Superior de la Universidad de Alicante. Se va a utilizar una base de datos MySQL para mostrar un sencillo ejemplo de uso y POSTMAN para comprobar el funcionamiento de la API REST.

1. Instalación de Node.js

Para la realización del API REST usando Node.js es necesaria la descarga e instalación del software Node.js y las dependencias necesarias.

1.1. Instalación de Node.js

Node.js es un entorno de ejecución para JavaScript. Se puede descargar el software en la página oficial <https://nodejs.org/es/download/>.

En la distribución de Ubuntu de la EPS ya se encuentra instalado este software, pero se puede instalar y actualizar mediante el siguiente comando:

```
$ sudo apt install nodejs
```

Tras instalar el software se puede validar la versión actualizada usando el siguiente comando:

```
$ node --version
```

1.2. Instalación de npm

npm es un sistema de gestión de paquetes para Node.js que permite gestionar las dependencias para las aplicaciones.

En la distribución de Ubuntu de la EPS ya se encuentra instalado este software, pero se puede instalar y actualizar mediante el siguiente comando:

```
$ sudo apt install npm
```

Tras instalar el software se puede validar la versión actualizada usando el siguiente comando:

```
$ npm --version
```

2. Creación de la aplicación

Tras instalar el software necesario, el siguiente paso consiste en crear una aplicación web. En este caso se va a crear una aplicación llamada “**REST_SD**” que implementará la API REST.

2.1. Crear la aplicación

Para crear una aplicación web llamada “**REST_SD**”, se crea un directorio y se inicia la aplicación dentro de ese directorio. Se solicita al usuario que introduzca por consola los datos de la

```
$ mkdir REST_SD  
$ cd REST_SD  
$ npm init
```

aplicación.

El programa crea un fichero llamado **package.json** con la configuración de la aplicación cuyo contenido será similar al siguiente:

```
{  
  "name": "rest_sd",  
  "version": "1.0.0",  
  "description": "Ejemplo de API REST para SD",  
  "main": "appSD.js",  
  "dependencies": {  
    "body-parser": "^1.19.0",  
    "express": "^4.17.1",  
    "mysql": "^2.18.1",  
    "request": "^2.88.2"  
  },  
  "devDependencies": {},  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [  
    "API",  
    "REST",  
    "SD"  
  ],  
  "author": "Jlalbentosa"  
  "license": "ISC"  
}
```

2.2. Instalar dependencias necesarias

Para este ejemplo se van a utilizar una serie de componentes que van a facilitar el desarrollo de la aplicación. Es necesario instalar los componentes correspondientes para poder utilizarlos en la aplicación posteriormente.

```
npm install express body-parser request mysql
```

2.3. Crear fichero de la aplicación

Se debe crear un fichero con la aplicación que será el que ejecute el servidor. En nuestro caso lo llamaremos **"appSD.js"**. Este fichero será el que se ejecutará cuando se llame desde la aplicación que hemos creado.

```
const express = require("express");

const app = express();

// Se define el puerto
const port=3000;

app.get("/", (req, res) => {

  res.json({message: 'Página de inicio de aplicación de ejemplo de SD'})
});

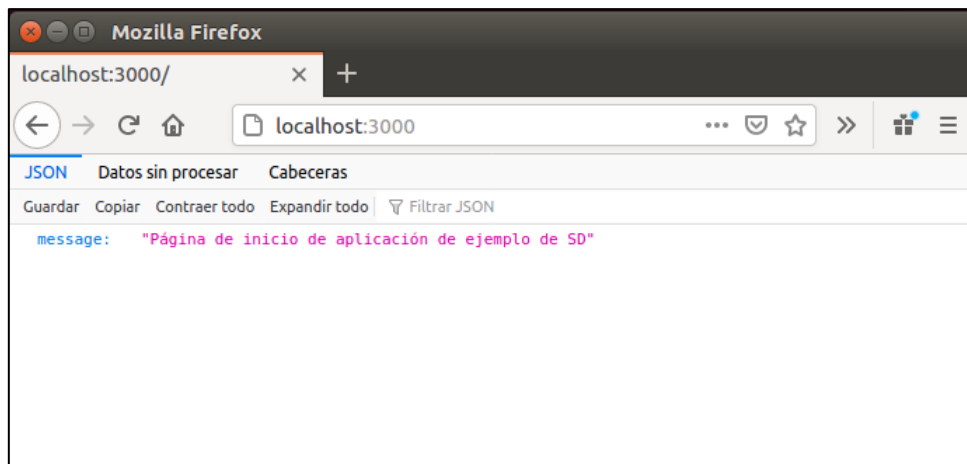
// Ejecutar la aplicacion
app.listen(port, () => {
  console.log(`Ejecutando la aplicación API REST de SD en el puerto ${port}`);
});
```

2.4. Arrancar la aplicación

Con el fichero anterior creado, se puede arrancar el servidor Node.js que se encargará de publicar la aplicación para que la puedan utilizar los usuarios:

```
$ node appSD.js
```

Se puede comprobar el resultado usando el navegador <http://localhost:3000>.



Otra forma de arrancar la aplicación consiste en la utilización del gestor de paquetes **npm**. Para ello es necesario modificar el fichero de configuración de la aplicación (package.json) e indicarle la acción a realizar en el arranque del servidor:

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node appSD.js"
}
```

De esta forma se puede arrancar el servidor con la aplicación de forma más sencilla usando la siguiente sintaxis:

```
npm start
```

3. Creación de la base de datos

En este apartado se crea una base de datos MySQL que se va a utilizar como ejemplo para realizar las consultas de la API REST a implementar.

Se va a utilizar una base de datos llamada “SD_MYSQL” que va a contener una tabla llamada “Usuarios”. Se utilizará el software “MySQL Workbench” para la conexión a MySQL, pero se puede utilizar cualquier otro software de conexión.

```
CREATE DATABASE SD_MYSQL;

USE SD_MYSQL;

CREATE TABLE Usuarios
(
  idUsuario INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
  nombre VARCHAR (25),
  ciudad VARCHAR (25),
  correo VARCHAR (25)
);

INSERT INTO Usuarios (nombre,ciudad,correo) VALUES ('María López','Alicante','mlopez@miempresa.es');
INSERT INTO Usuarios (nombre,ciudad,correo) VALUES ('Juan Fernández','Alicante','jfernandez@miempresa.es');
INSERT INTO Usuarios (nombre,ciudad,correo) VALUES ('Lucía Martínez','Madrid','lmartinez@miempresa.es');
INSERT INTO Usuarios (nombre,ciudad,correo) VALUES ('José Luis
Gutiérrez','Alicante','jgutierrez@miempresa.es');
INSERT INTO Usuarios (nombre,ciudad,correo) VALUES ('Francisco
García','Barcelona','fgarciaUsuarios@miempresa.es');

COMMIT;
```

El script para la creación de la base de datos es el siguiente:

Se puede ver el contenido de la tabla creada usando MySQL Workbench:

The screenshot shows a SQL client interface with a query editor and a results grid. The query editor contains the SQL statement: `SELECT * FROM SD_MYSQL.Usuarios;`. The results grid displays the following data:

| # | idUsuario | nombre | ciudad | correo |
|---|-----------|---------------------|-----------|-------------------------|
| 1 | 1 | María López | Alicante | mlopez@miempresa.es |
| 2 | 2 | Juan Fernández | Alicante | jfernandez@miempresa.es |
| 3 | 3 | Lucía Martínez | Madrid | lmartinez@miempresa.es |
| 4 | 4 | José Luis Gutiérrez | Alicante | jgutierrez@miempresa.es |
| 5 | 5 | Francisco García | Barcelona | fgarcia@miempresa.es |
| * | NULL | NULL | NULL | NULL |

4. Implementación de la API REST

Se va a crear una sencilla API REST de ejemplo que permita realizar las operaciones básicas sobre la tabla USUARIOS de la base de datos MySQL creada en el punto anterior. Se trata de las operaciones CRUD básicas de una base de datos relacional.

Los servicios que se van a generar en la API son los que se muestran en la siguiente tabla:

| VERBO HTTP | RUTA | DESCRIPCIÓN |
|------------|----------------|--|
| GET | /usuarios | Obtener el listado de todos los usuarios. |
| GET | /usuarios/{id} | Obtener los datos del usuario con identificador {id}. |
| POST | /usuarios | Añadir un nuevo usuario. |
| PUT | /usuarios/{id} | Modificar los datos del usuario con el identificador {id}. |
| DELETE | /usuarios | Borrar el usuario con identificador {id}. |

4.1. Configurar la conexión a la base de datos

El primer paso consiste en configurar la aplicación para poder acceder a la base de datos MySQL. Para ello es necesario añadir las siguientes instrucciones al fichero **appSD.js**:

```
const mysql = require ("mysql");
const bodyParser = require("body-parser");

// Configuración de la conexión a la base de datos MySQL
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'SD_MYSQL'
});

// Comprobar conexión a la base de datos
connection.connect(error=> {
  if (error) throw error;
  console.log('Conexión a la base de datos SD_MYSQL correcta');
});
```

Con el código anterior se configura el acceso y se valida el correcto acceso a la base de datos. Si todo funciona correctamente, al arrancar la aplicación se mostrará por consola el mensaje correspondiente:

```
$ npm start
> rest_sd@1.0.0 start /home/alu/REST_SD
> node appSD.js

Ejecutando la aplicación API REST de SD en el puerto 3000
Conexión a la base de datos SD_MYSQL correcta
```

4.2. Implementar la operación de listado de todos los usuarios

Se va a implementar la opción de consulta del listado de todos los usuarios de la tabla:

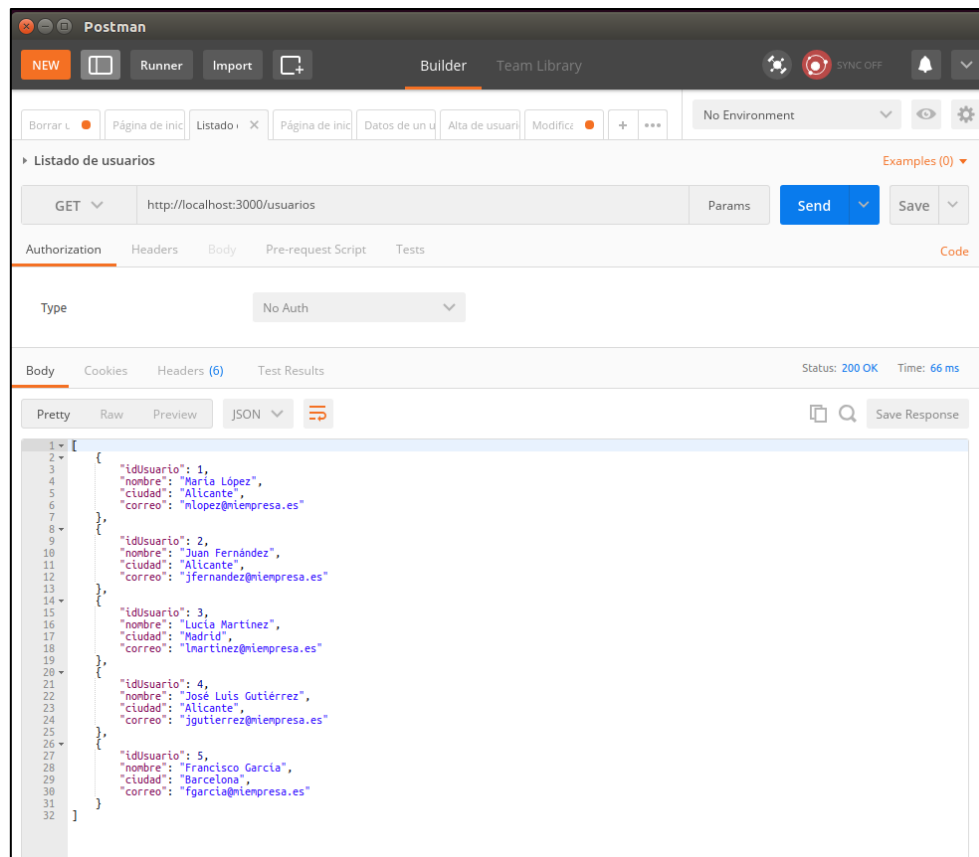
| VERBO HTTP | RUTA | DESCRIPCIÓN |
|------------|-----------|---|
| GET | /usuarios | Obtener el listado de todos los usuarios. |

Se debe crear la sentencia SQL correspondiente y ejecutarla sobre la conexión a la base de datos. El código para hacerlo es el siguiente:

```
// Listado de todos los usuarios
appSD.get("/usuarios", (request, response) => {
  console.log('Listado de todos los usuarios');

  const sql = 'SELECT * FROM Usuarios';
  connection.query(sql, (error, resultado) => {
    if (error) throw error;
    if (resultado.length > 0) {
      response.json(resultado);
    } else {
      response.send('No hay resultados');
    }
  });
});
```

Si la tabla está vacía se devolverá el mensaje de “No hay resultados”. Si la tabla contiene datos se mostrará el listado de todos los usuarios en formato JSON como se muestra a continuación:



4.3. Implementar la operación de obtener datos de un usuario

Se va a implementar la opción de obtener los datos de un usuario concreto de la tabla. Para ello se facilitará en la llamada el identificador del usuario del que se quiere obtener la información:

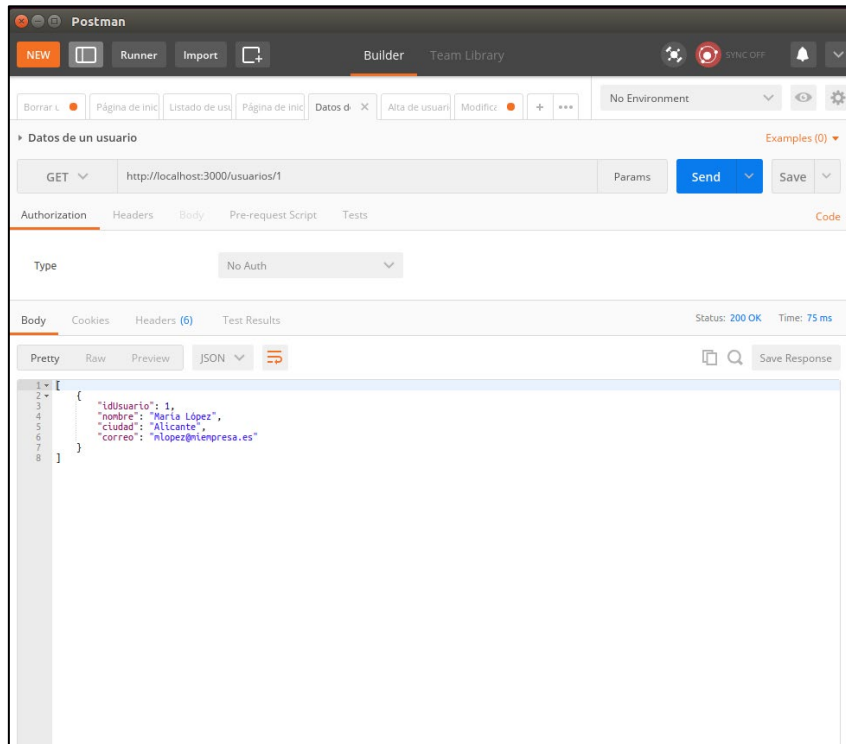
| VERBO HTTP | RUTA | DESCRIPCIÓN |
|------------|----------------|---|
| GET | /usuarios/{id} | Obtener los datos del usuario con identificador {id}. |

El primer paso consiste en obtener el identificador del usuario de los parámetros de la llamada. Luego se debe crear la sentencia SQL correspondiente y ejecutarla sobre la conexión a la base de datos. El código para hacerlo es el siguiente:

```
// Obtener datos de un usuario
appSD.get("/usuarios/:id", (request, response) => {
  console.log('Obtener datos de un usuario');

  const {id} = request.params;
  const sql = `SELECT * FROM Usuarios WHERE idUsuario = ${id}`;
  connection.query(sql, (error, resultado) => {
    if (error) throw error;
    if (resultado.length > 0) {
      response.json(resultado);
    } else {
      response.send('No hay resultados');
    }
  })
});
```

Si el identificador del usuario no existe se mostrará el texto “No hay resultados”. Si existe un usuario con el identificador proporcionado se mostrará toda su información como se muestra a continuación:



4.4. Implementar la operación de añadir un nuevo usuario

Se va a implementar la opción de añadir un nuevo usuario de la tabla. Para ello se facilitarán en el body de la llamada todos los datos del nuevo usuario:

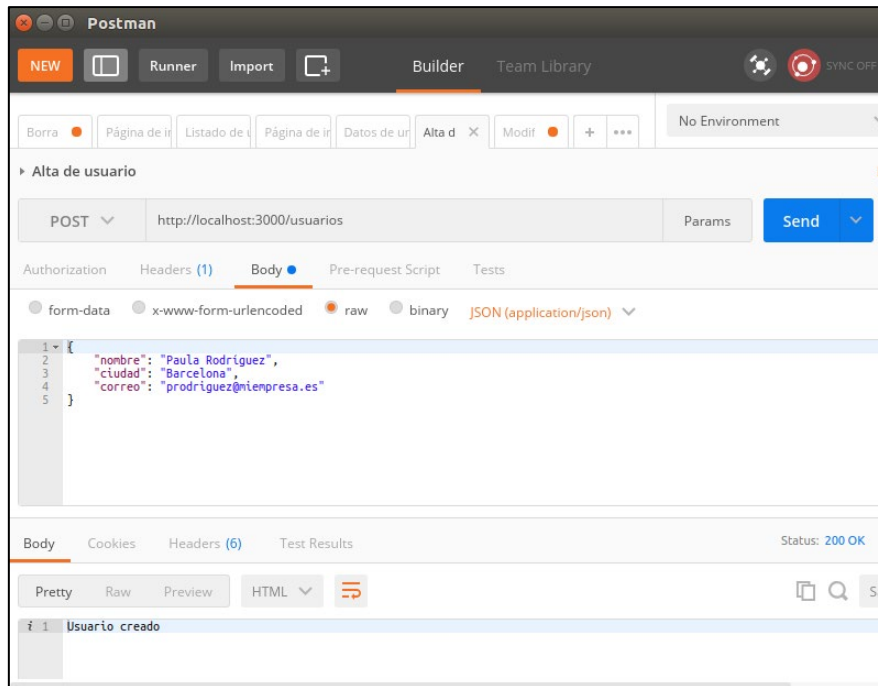
| VERBO HTTP | RUTA | DESCRIPCIÓN |
|------------|-----------|--------------------------|
| POST | /usuarios | Añadir un nuevo usuario. |

El primer paso consiste en obtener los datos del usuario del body de la llamada. Luego se debe crear la sentencia SQL correspondiente y ejecutarla sobre la conexión a la base de datos. El código para hacerlo es el siguiente:

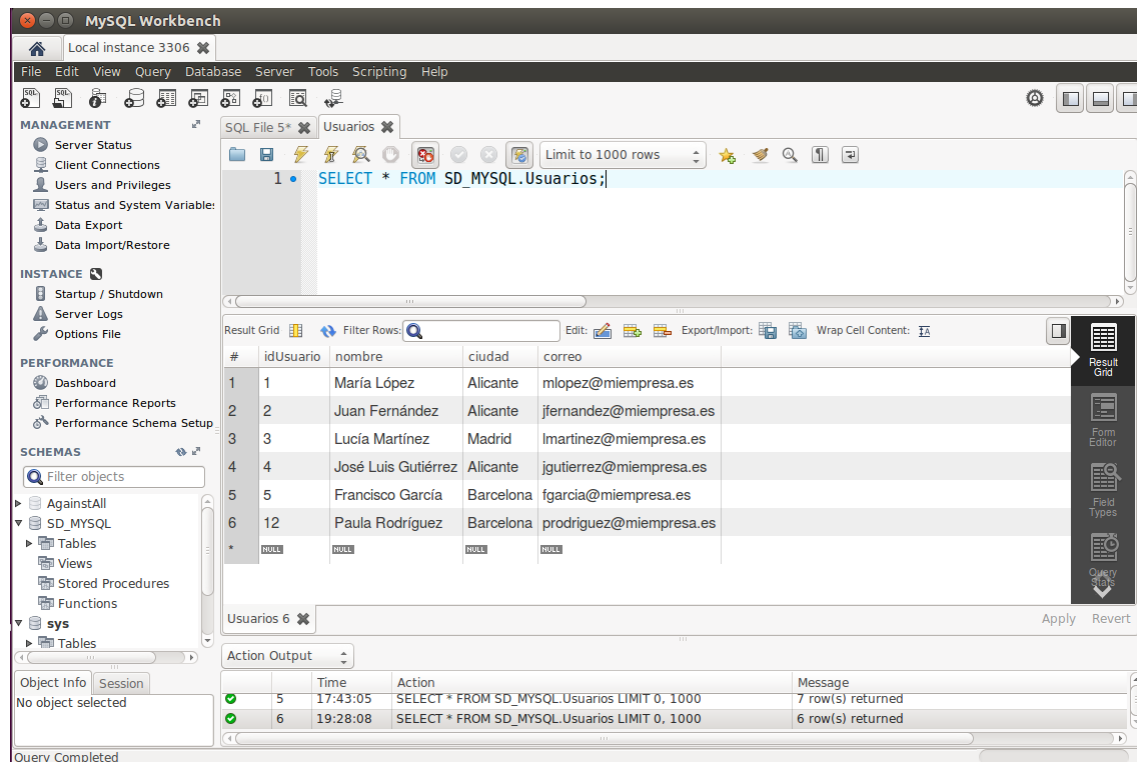
```
// Añadir un nuevo usuario
appSD.post("/usuarios", (request, response) => {
  console.log('Añadir nuevo usuario');
  const sql = 'INSERT INTO Usuarios SET ?';

  const usuarioObj = {
    nombre: request.body.nombre,
    ciudad: request.body.ciudad,
    correo: request.body.correo
  }
  connection.query(sql, usuarioObj, error => {
    if (error) throw error;
    response.send('Usuario creado');
  });
});
```


Los datos del nuevo usuario a dar de alta se deben proporcionar en el body de la llamada en formato JSON. Eso puede hacerse directamente dentro de la pestaña “body” de Postman. Si todo funciona correctamente se creará un nuevo registro en la base de datos como se muestra a continuación:



Se puede confirmar que hay un nuevo registro consultando directamente el contenido de la tabla en la base de datos MySQL:



4.5. Implementar la operación de modificación de un usuario

Se va a implementar la opción de modificar un usuario de la tabla. Para ello se facilitarán en la llamada el identificador del usuario a modificar y en el body de la llamada se incluirán todos los datos del usuario a modificar:

| VERBO HTTP | RUTA | DESCRIPCIÓN |
|------------|----------------|--|
| PUT | /usuarios/{id} | Modificar los datos del usuario con el identificador {id}. |

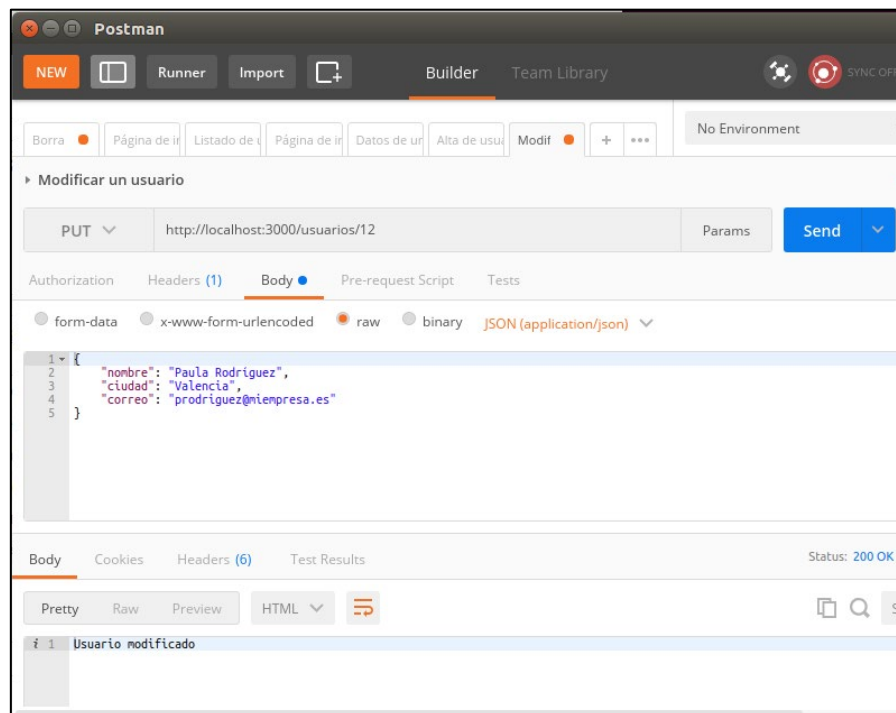
El primer paso consiste en obtener el identificador del usuario de los parámetros de la llamada y obtener los nuevos datos del usuario del body de la llamada. Luego se debe crear la sentencia SQL correspondiente y ejecutarla sobre la conexión a la base de datos. El código para hacerlo es el siguiente:

```
// Modificar un usuario
appSD.put("/usuarios/:id", (request, response) => {
  console.log('Modificar usuario');

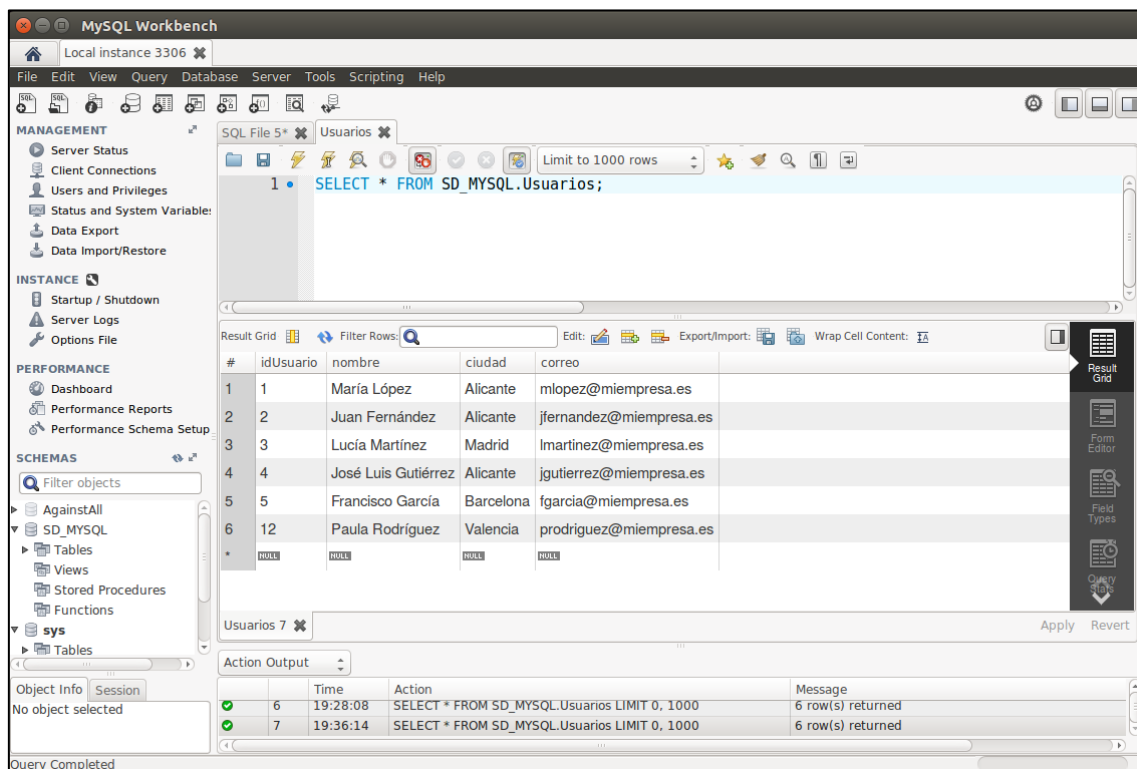
  const {id} = request.params;
  const {nombre, ciudad, correo} = request.body;
  const sql = `UPDATE Usuarios SET nombre='${nombre}', ciudad='${ciudad}',
correo='${correo}' WHERE idUsuario=${id}`;
  connection.query(sql, error => {
    if (error) throw error;
    response.send('Usuario modificado');
  });
});
```

Al igual que en el caso del alta, se deben introducir en el body de la llamada todos los valores de los campos del usuario a modificar en formato JSON. Si todo funciona correctamente se

modificarán los datos del usuario cuyo identificador se ha facilitado en los parámetros de la llamada como se muestra a continuación:



Se puede confirmar que los datos del usuario se han modificado correctamente consultando directamente el contenido de la tabla en la base de datos MySQL:



4.6. Implementar la operación de baja de un usuario

Se va a implementar la opción de dar de baja a un usuario de la tabla. Para ello se facilitará en la llamada el identificador del usuario a dar de baja:

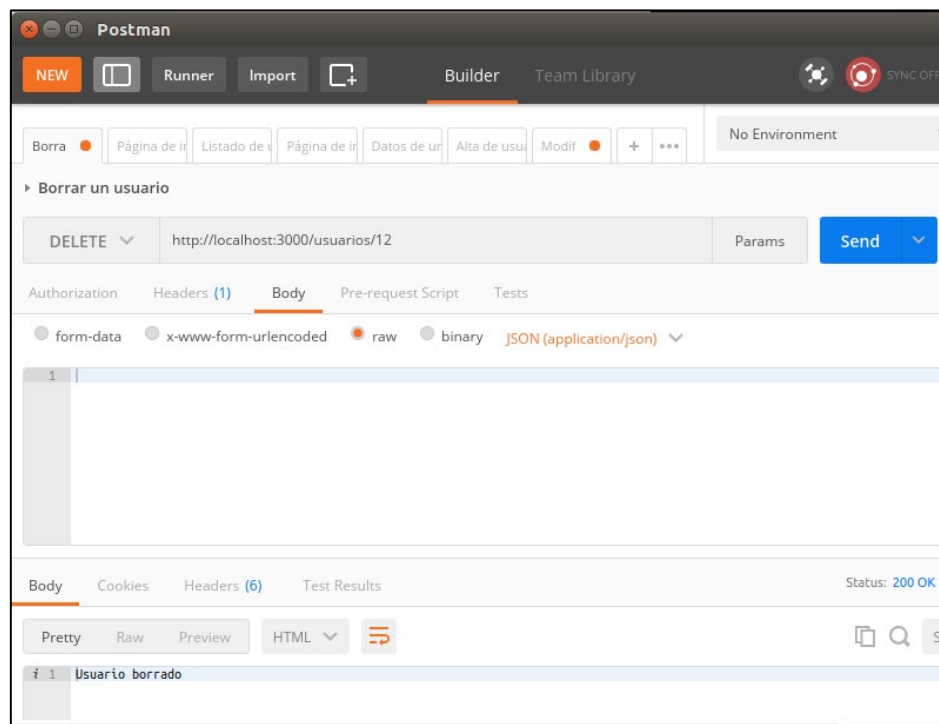
| VERBO HTTP | RUTA | DESCRIPCIÓN |
|---------------|-----------|---|
| DELETE | /usuarios | Borrar el usuario con identificador {id} . |

El primer paso consiste en obtener el identificador del usuario de los parámetros de la llamada. Luego se debe crear la sentencia SQL correspondiente y ejecutarla sobre la conexión a la base de datos. El código para hacerlo es el siguiente:

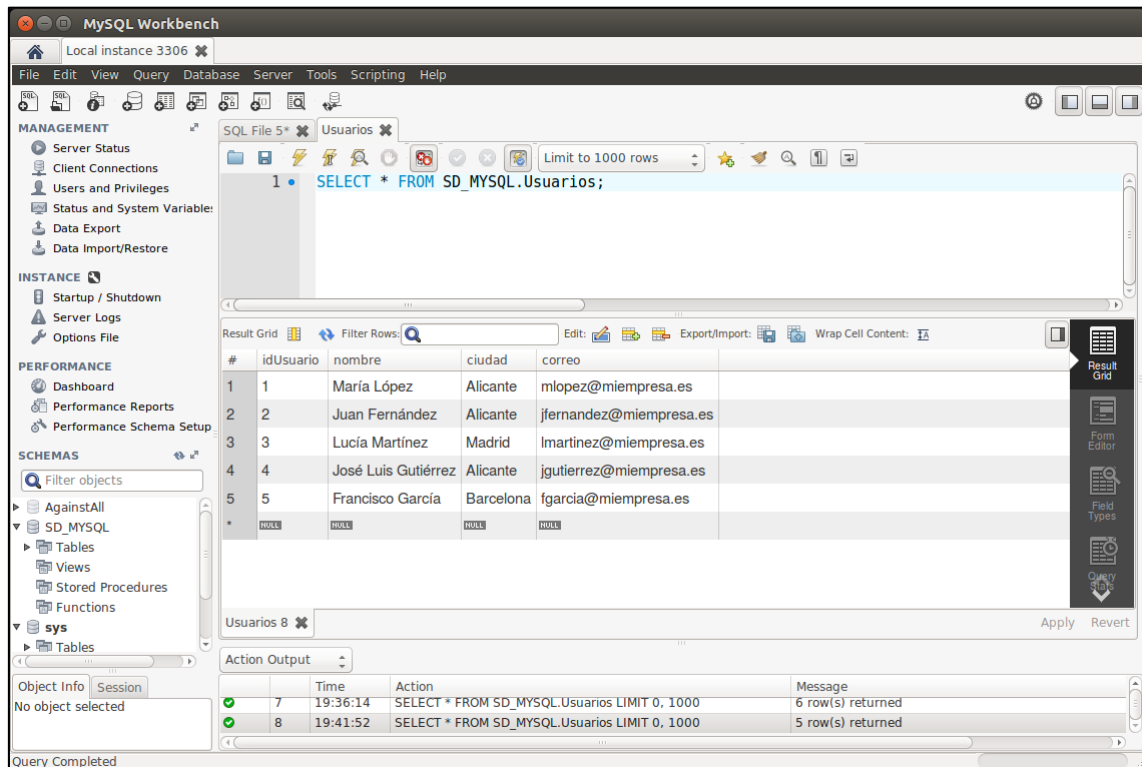
```
// Borrar un usuario
appSD.delete("/usuarios/:id", (request, response) => {
  console.log('Borrar usuario');

  const {id} = request.params;
  sql = `DELETE FROM Usuarios WHERE idUsuario= ${id}`;
  connection.query(sql, error => {
    if (error) throw error;
    response.send('Usuario borrado');
  });
});
```

Si todo funciona correctamente se eliminará de la tabla al usuario cuyo identificador se ha facilitado en los parámetros de la llamada como se muestra a continuación:



Se puede confirmar que el usuario se ha eliminado correctamente consultando directamente el contenido de la tabla en la base de datos MySQL:



The screenshot shows the MySQL Workbench interface. The SQL Editor contains the query: `SELECT * FROM SD_MYSQL.Usuarios;`. The Result Grid displays the following data:

| # | idUsuario | nombre | ciudad | correo |
|---|-----------|---------------------|-----------|-------------------------|
| 1 | 1 | María López | Alicante | mlopez@miempresa.es |
| 2 | 2 | Juan Fernández | Alicante | jfernandez@miempresa.es |
| 3 | 3 | Lucía Martínez | Madrid | lmartinez@miempresa.es |
| 4 | 4 | José Luis Gutiérrez | Alicante | jgutierrez@miempresa.es |
| 5 | 5 | Francisco García | Barcelona | fgarcia@miempresa.es |
| * | NULL | NULL | NULL | NULL |

The Action Output pane at the bottom shows the execution of the query:

| | Time | Action | Message |
|---|----------|---|-------------------|
| 7 | 19:36:14 | SELECT * FROM SD_MYSQL.Usuarios LIMIT 0, 1000 | 6 row(s) returned |
| 8 | 19:41:52 | SELECT * FROM SD_MYSQL.Usuarios LIMIT 0, 1000 | 5 row(s) returned |

The status bar at the bottom indicates "Query Completed".

5. Acceso a fichero de texto

Además de utilizar como fuente de datos una base de datos, también es posible utilizar un fichero de texto para almacenar la información. El API REST puede en este caso utilizar las funciones de acceso a fichero para obtener/modificar la información.

Se va a mostrar un ejemplo de uso que permite obtener un listado de usuarios a partir de la información existente en un fichero de texto en formato JSON.

5.1. Crear fichero JSON con los datos

Para seguir con el mismo ejemplo implementado en los apartados anteriores, se va a crear un fichero en formato JSON que contiene un listado de usuarios. El fichero se llamará “**usuarios.json**” y su contenido será el siguiente:

```
{
  "1": {
    "nombre": "María López",
    "ciudad": "Alicante",
    "correo": "mlopez@miempresa.es"
  },
  "2": {
    "nombre": "Juan Fernández",
    "ciudad": "Alicante",
    "correo": "jfernandez@miempresa.es"
  },
  "3": {
    "nombre": "Lucía Martínez",
    "ciudad": "Madrid",
    "correo": "lmartinez@miempresa.es"
  },
  "4": {
    "nombre": "José Luis Gutiérrez",
    "ciudad": "Alicante",
    "correo": "jgutierrez@miempresa.es"
  },
  "5": {
    "nombre": "Francisco García",
    "ciudad": "Barcelona",
    "correo": "fgarcia@miempresa.es"
  }
}
```

5.2. Implementar la operación de listado de todos los usuarios

Se va a implementar la opción de consulta del listado de todos los usuarios del fichero.

| VERBO HTTP | RUTA | DESCRIPCIÓN |
|------------|----------------|--|
| GET | /usuarios_json | Obtener el listado de todos los usuarios del fichero de texto en formato JSON. |

Se debe acceder al fichero de texto, leer su contenido y devolver el resultado. El código para hacerlo es el siguiente:

```
// Leer fichero JSON
appSD.get("/usuarios_json", (request, response) => {
  console.log('Leer datos de fichero en formato JSON');

  try {
    const datos = fs.readFileSync('usuarios.json', 'utf8');
    response.send(JSON.parse(datos));
  } catch (error) {
    console.log (error);
  }
});
```

Al hacer la llamada se mostrará el listado de todos los usuarios del fichero de texto en formato JSON como se muestra a continuación:

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/usuarios_json`. The response is a JSON array of 5 user objects, displayed in a pretty-printed format. The response status is 200 OK and the time taken is 82 ms.

```
[{"id": 1, "nombre": "María López", "ciudad": "Alicante", "correo": "mlopez@empresa.es"}, {"id": 2, "nombre": "Juan Fernández", "ciudad": "Alicante", "correo": "jfernandez@empresa.es"}, {"id": 3, "nombre": "Lucía Martínez", "ciudad": "Madrid", "correo": "lmartinez@empresa.es"}, {"id": 4, "nombre": "José Luis Gutiérrez", "ciudad": "Alicante", "correo": "jgutierrez@empresa.es"}, {"id": 5, "nombre": "Francisco García", "ciudad": "Barcelona", "correo": "fgarcia@empresa.es"}]
```