Creación de servicios API/REST en JAVA sin utilizar Spring Boot

Introducción

Cuando no queremos usar frameworks como Spring Boot, podemos utilizar el servidor HTTP básico incluido en Java para implementar servicios API/REST. Aunque no es tan completo ni eficiente como otros frameworks, es una buena opción para proyectos pequeños, aprendizaje o escenarios donde deseemos mantener las dependencias mínimas.

Pasos para Crear un Servicio API/REST

1. Configuración del Proyecto

1. **IDE**: Puedes usar cualquier IDE como IntelliJ IDEA, Eclipse o NetBeans.
2. **Requisitos**:
    - JDK 11 o superior (recomendado).

Estructura del Proyecto:

```
src/
├── main/
    ├── java/
        ├── com.miserviciorest.api/
            ├── Main.java
            ├── Usuario.java
            ├── UsuarioHandler.java
            ├── BaseHandler.java
```

3.

---

2. Crear la Clase Principal

La clase principal configurará el servidor HTTP y registrará las rutas.

Código: `Main.java`

```
package com.miserviciorest.api;
```

```
import com.sun.net.httpserver.HttpServer;

import java.io.IOException;
import java.net.InetSocketAddress;

public class Main {
    public static void main(String[] args) throws IOException {
        // Crear un servidor HTTP en el puerto 8080
        HttpServer server = HttpServer.create(new
InetSocketAddress(8080), 0);
        System.out.println("Servidor iniciado en
http://localhost:8080");

        // Registrar las rutas
        server.createContext("/usuarios", new UsuarioHandler());

        // Iniciar el servidor
        server.setExecutor(null); // Usa el executor por defecto
        server.start();
    }
}
```

---

3. Crear el Modelo

El modelo `Usuario` representa los datos que gestionará la API.

Código: `Usuario.java`

```
package com.miserviciorest.api;

public class Usuario {
    private int id;
    private String nombre;
    private String email;

    public Usuario() {}

    public Usuario(int id, String nombre, String email) {
        this.id = id;
        this.nombre = nombre;
```

```
        this.email = email;
    }

    // Getters y setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Usuario{" +
                "id=" + id +
                ", nombre='" + nombre + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

4. Crear el Handler

El `Handler` es el componente que gestiona las solicitudes HTTP. Aquí implementaremos las operaciones **GET**, **POST**, **PUT** y **DELETE**.

Código: `UsuarioHandler.java`

```
package com.miserviciorest.api;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;

import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.stream.Collectors;

public class UsuarioHandler implements HttpHandler {
    private static List<Usuario> usuarios = new ArrayList<>();
    private static int idCounter = 1;

    @Override
    public void handle(HttpExchange exchange) throws IOException {
        String metodo = exchange.getRequestMethod();
        String path = exchange.getRequestURI().getPath();

        // Ruta principal "/usuarios"
        if (path.equals("/usuarios")) {
            switch (metodo) {
                case "GET":
                    handleGetAll(exchange);
                    break;
                case "POST":
                    handlePost(exchange);
                    break;
                default:
                    sendResponse(exchange, 405, "Método no
permitido");
            }
        }
        // Ruta con ID "/usuarios/{id}"
        else if (path.matches("/usuarios/\\d+")) {
            int id = Integer.parseInt(path.split("/")[2]);
```

```
            switch (metodo) {
                case "GET":
                    handleGetById(exchange, id);
                    break;
                case "PUT":
                    handlePut(exchange, id);
                    break;
                case "DELETE":
                    handleDelete(exchange, id);
                    break;
                default:
                    sendResponse(exchange, 405, "Método no
permitido");
            }
        } else {
            sendResponse(exchange, 404, "Ruta no encontrada");
        }
    }

    // Manejar GET: obtener todos los usuarios
    private void handleGetAll(HttpExchange exchange) throws
IOException {
        String json = usuarios.stream()
                .map(u ->
String.format("{\"id\":%d,\"nombre\":\"%s\",\"email\":\"%s\"}",
u.getId(), u.getNombre(), u.getEmail()))
                .collect(Collectors.joining(",", "[", "]"));
        sendResponse(exchange, 200, json);
    }

    // Manejar GET: obtener un usuario por ID
    private void handleGetById(HttpExchange exchange, int id) throws
IOException {
        Usuario usuario = usuarios.stream().filter(u -> u.getId() ==
id).findFirst().orElse(null);
        if (usuario != null) {
            String json =
String.format("{\"id\":%d,\"nombre\":\"%s\",\"email\":\"%s\"}",
usuario.getId(), usuario.getNombre(), usuario.getEmail());
            sendResponse(exchange, 200, json);
        } else {
```

```
            sendResponse(exchange, 404, "Usuario no encontrado");
        }
    }

    // Manejar POST: crear un nuevo usuario
    private void handlePost(HttpExchange exchange) throws
IOException {
        InputStreamReader isr = new
InputStreamReader(exchange.getRequestBody(),
StandardCharsets.UTF_8);
        BufferedReader reader = new BufferedReader(isr);
        String body = reader.lines().collect(Collectors.joining());
        Map<String, String> datos = parseForm(body);

        Usuario nuevoUsuario = new Usuario(idCounter++,
datos.get("nombre"), datos.get("email"));
        usuarios.add(nuevoUsuario);
        sendResponse(exchange, 201, "Usuario creado con éxito");
    }

    // Manejar PUT: actualizar un usuario existente
    private void handlePut(HttpExchange exchange, int id) throws
IOException {
        Usuario usuario = usuarios.stream().filter(u -> u.getId() ==
id).findFirst().orElse(null);
        if (usuario != null) {
            InputStreamReader isr = new
InputStreamReader(exchange.getRequestBody(),
StandardCharsets.UTF_8);
            BufferedReader reader = new BufferedReader(isr);
            String body =
reader.lines().collect(Collectors.joining());
            Map<String, String> datos = parseForm(body);

            usuario.setNombre(datos.get("nombre"));
            usuario.setEmail(datos.get("email"));
            sendResponse(exchange, 200, "Usuario actualizado con
éxito");
        } else {
            sendResponse(exchange, 404, "Usuario no encontrado");
        }
```

```
    }

    // Manejar DELETE: eliminar un usuario
    private void handleDelete(HttpExchange exchange, int id) throws
IOException {
        if (usuarios.removeIf(u -> u.getId() == id)) {
            sendResponse(exchange, 200, "Usuario eliminado con
éxito");
        } else {
            sendResponse(exchange, 404, "Usuario no encontrado");
        }
    }

    // Responder con un mensaje
    private void sendResponse(HttpExchange exchange, int statusCode,
String mensaje) throws IOException {
        byte[] response = mensaje.getBytes(StandardCharsets.UTF_8);
        exchange.sendResponseHeaders(statusCode, response.length);
        OutputStream os = exchange.getResponseBody();
        os.write(response);
        os.close();
    }

    // Parsear datos del cuerpo en formato x-www-form-urlencoded
    private Map<String, String> parseForm(String body) {
        return Arrays.stream(body.split("&"))
                .map(param -> param.split("="))
                .collect(Collectors.toMap(pair -> pair[0], pair ->
pair[1]));
    }
}
```

Conclusión

Este servidor API/REST básico utiliza únicamente las clases estándar de Java para gestionar rutas y métodos HTTP. Es ideal para proyectos pequeños y para comprender cómo funcionan las capas de comunicación en la web sin frameworks adicionales.

Cliente para Consumir el API/REST en Java

Introducción

El cliente es una aplicación que interactúa con la API/REST utilizando HTTP para realizar operaciones como obtener, crear, actualizar y eliminar recursos. En este caso, implementaremos un cliente que consuma los endpoints del servidor que acabamos de crear.

Para la comunicación HTTP, utilizaremos la clase estándar `HttpURLConnection`, que forma parte del JDK.

---

1. Configuración de la Clase Cliente

Crearemos una clase `ClienteAPI` que incluirá métodos para consumir los endpoints del servidor.

Código: `ClienteAPI.java`

```
package com.miserviciorest.api;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

public class ClienteAPI {

    private static final String BASE_URL =
"http://localhost:8080/usuarios";

    // Método para realizar solicitudes GET
    public static void obtenerTodosLosUsuarios() {
        try {
            URL url = new URL(BASE_URL);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("GET");

            int status = connection.getResponseCode();
            if (status == 200) {
                BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
```

```
                String linea;
                while ((linea = reader.readLine()) != null) {
                    System.out.println(linea);
                }
                reader.close();
            } else {
                System.out.println("Error al obtener usuarios:
Código " + status);
            }
            connection.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Método para realizar solicitudes GET con ID
    public static void obtenerUsuarioPorId(int id) {
        try {
            URL url = new URL(BASE_URL + "/" + id);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("GET");

            int status = connection.getResponseCode();
            if (status == 200) {
                BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                String linea;
                while ((linea = reader.readLine()) != null) {
                    System.out.println(linea);
                }
                reader.close();
            } else {
                System.out.println("Error al obtener usuario: Código
" + status);
            }
            connection.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    // Método para realizar solicitudes POST
    public static void crearUsuario(String nombre, String email) {
        try {
            URL url = new URL(BASE_URL);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("POST");
            connection.setDoOutput(true);
            connection.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");

            String parametros = "nombre=" + nombre + "&email=" +
email;
            OutputStream os = connection.getOutputStream();
            os.write(parametros.getBytes());
            os.flush();
            os.close();

            int status = connection.getResponseCode();
            if (status == 201) {
                System.out.println("Usuario creado con éxito.");
            } else {
                System.out.println("Error al crear usuario: Código "
+ status);
            }
            connection.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Método para realizar solicitudes PUT
    public static void actualizarUsuario(int id, String nombre,
String email) {
        try {
            URL url = new URL(BASE_URL + "/" + id);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("PUT");
            connection.setDoOutput(true);
```

```
            connection.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");

            String parametros = "nombre=" + nombre + "&email=" +
email;
            OutputStream os = connection.getOutputStream();
            os.write(parametros.getBytes());
            os.flush();
            os.close();

            int status = connection.getResponseCode();
            if (status == 200) {
                System.out.println("Usuario actualizado con
éxito.");
            } else {
                System.out.println("Error al actualizar usuario:
Código " + status);
            }
            connection.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Método para realizar solicitudes DELETE
    public static void eliminarUsuario(int id) {
        try {
            URL url = new URL(BASE_URL + "/" + id);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("DELETE");

            int status = connection.getResponseCode();
            if (status == 200) {
                System.out.println("Usuario eliminado con éxito.");
            } else {
                System.out.println("Error al eliminar usuario:
Código " + status);
            }
            connection.disconnect();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

2. Clase Principal para Probar el Cliente

Crearemos una clase `TestClienteAPI` para probar las operaciones del cliente.

Código: `TestClienteAPI.java`

```
package com.miserviciorest.api;

public class TestClienteAPI {
    public static void main(String[] args) {
        // Obtener todos los usuarios
        System.out.println("=== Obtener todos los usuarios ===");
        ClienteAPI.obtenerTodosLosUsuarios();

        // Crear un nuevo usuario
        System.out.println("\n=== Crear un nuevo usuario ===");
        ClienteAPI.crearUsuario("Luis Fernández",
"luis@ejemplo.com");

        // Obtener usuario por ID
        System.out.println("\n=== Obtener usuario por ID ===");
        ClienteAPI.obtenerUsuarioPorId(1);

        // Actualizar un usuario
        System.out.println("\n=== Actualizar usuario ===");
        ClienteAPI.actualizarUsuario(1, "Luis Actualizado",
"luis_actualizado@ejemplo.com");

        // Eliminar un usuario
        System.out.println("\n=== Eliminar usuario ===");
        ClienteAPI.eliminarUsuario(1);

        // Verificar que se eliminó correctamente
        System.out.println("\n=== Verificar eliminación ===");
```

```
        ClienteAPI.obtenerTodosLosUsuarios();
    }
}
```

Funcionamiento del Cliente

1. **Operaciones Soportadas**:
    - Obtener todos los usuarios (`GET` a `/usuarios`).
    - Obtener un usuario específico (`GET` a `/usuarios/{id}`).
    - Crear un usuario (`POST` a `/usuarios`).
    - Actualizar un usuario (`PUT` a `/usuarios/{id}`).
    - Eliminar un usuario (`DELETE` a `/usuarios/{id}`).
2. **Ejecutar el Cliente**:
    - Ejecuta primero el servidor (`Main.java`).
    - Luego, ejecuta `TestClienteAPI.java` para realizar las operaciones.

Conclusión

El cliente implementado permite consumir las operaciones de la API/REST usando solo las herramientas estándar de Java. Aunque es básico, es funcional y ofrece una base sólida para entender cómo interactuar con servicios REST. Si deseas añadir autenticación o mejorar la gestión de errores, avísame, y lo desarrollaremos.