Creación de servicios API/REST en JAVA con Spring Boot

Introducción

Los servicios **API/REST** permiten la interacción entre aplicaciones a través de protocolos HTTP, utilizando un enfoque basado en recursos. REST (Representational State Transfer) es un estilo de arquitectura que facilita la comunicación escalable, eficiente y sencilla entre cliente y servidor.

Con Java, podemos crear servicios API/REST utilizando frameworks como **Spring Boot**, que simplifica enormemente el proceso de desarrollo.

Características de los Servicios REST

1. **Protocolos de Comunicación**: Utilizan HTTP como medio principal de comunicación.
2. **Estilo Basado en Recursos**: Cada recurso (entidad) se representa mediante una URL.
    - Ejemplo: `https://api.miapp.com/usuarios/1`
3. **Operaciones CRUD mediante Verbos HTTP**:
    - `GET`: Leer recursos.
    - `POST`: Crear recursos.
    - `PUT`: Actualizar recursos.
    - `DELETE`: Eliminar recursos.
4. **Formato de Datos**:
    - Usualmente JSON o XML para la transferencia de información.

Creación de un Servicio REST con Spring Boot

Paso 1: Configurar un Proyecto Spring Boot

1. Ir a Spring Initializr.
2. Seleccionar:
    - **Maven Project**
    - **Lenguaje**: Java.
    - **Dependencias**: Spring Web.
3. Descargar el proyecto y abrirlo en un IDE (como IntelliJ IDEA o Eclipse).

Paso 2: Estructura del Proyecto

Supongamos que queremos gestionar un recurso llamado `Usuario`. La estructura básica será:

```
src/
└── main/
    ├── java/
    │   └── com.miapp.api/
    │       ├── ApiApplication.java
    │       ├── controlador/
    │       │   └── UsuarioControlador.java
    │       ├── modelo/
    │       │   └── Usuario.java
    │       └── servicio/
    │           └── UsuarioServicio.java
```

Paso 3: Crear un Modelo (Entidad)

```
package com.miapp.api.modelo;

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

    // Constructores
    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;
    }
}
```

Paso 4: Crear un Servicio

```
package com.miapp.api.servicio;

import com.miapp.api.modelo.Usuario;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class UsuarioServicio {
    private final List<Usuario> usuarios = new ArrayList<>();

    public UsuarioServicio() {
        usuarios.add(new Usuario(1, "Juan Pérez",
"juan@ejemplo.com"));
```

```
        usuarios.add(new Usuario(2, "María López",
"maria@ejemplo.com"));
    }

    public List<Usuario> obtenerTodos() {
        return usuarios;
    }

    public Usuario obtenerPorId(int id) {
        return usuarios.stream().filter(u -> u.getId() ==
id).findFirst().orElse(null);
    }

    public Usuario agregar(Usuario usuario) {
        usuarios.add(usuario);
        return usuario;
    }

    public Usuario actualizar(int id, Usuario usuarioActualizado) {
        Usuario usuario = obtenerPorId(id);
        if (usuario != null) {
            usuario.setNombre(usuarioActualizado.getNombre());
            usuario.setEmail(usuarioActualizado.getEmail());
        }
        return usuario;
    }

    public boolean eliminar(int id) {
        return usuarios.removeIf(u -> u.getId() == id);
    }
}
```

Paso 5: Crear un Controlador

```
package com.miapp.api.controlador;

import com.miapp.api.modelo.Usuario;
import com.miapp.api.servicio.UsuarioServicio;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/usuarios")
public class UsuarioControlador {

    @Autowired
    private UsuarioServicio servicio;

    @GetMapping
    public List<Usuario> obtenerTodos() {
        return servicio.obtenerTodos();
    }

    @GetMapping("/{id}")
    public Usuario obtenerPorId(@PathVariable int id) {
        return servicio.obtenerPorId(id);
    }

    @PostMapping
    public Usuario agregar(@RequestBody Usuario usuario) {
        return servicio.agregar(usuario);
    }

    @PutMapping("/{id}")
    public Usuario actualizar(@PathVariable int id, @RequestBody
Usuario usuario) {
        return servicio.actualizar(id, usuario);
    }

    @DeleteMapping("/{id}")
    public String eliminar(@PathVariable int id) {
        if (servicio.eliminar(id)) {
            return "Usuario eliminado con éxito.";
        } else {
            return "Usuario no encontrado.";
        }
    }
}
```

Paso 6: Ejecutar la Aplicación
Ejecutar la clase principal:

```
package com.miapp.api;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiApplication.class, args);
    }
}
```

1. Acceder a las rutas:
    - GET Todos los usuarios: `http://localhost:8080/api/usuarios`
    - GET Usuario por ID: `http://localhost:8080/api/usuarios/1`
    - POST Crear usuario: Enviar un JSON con nombre y email.
    - PUT Actualizar usuario: Enviar un JSON con los datos actualizados.
    - DELETE Eliminar usuario: `http://localhost:8080/api/usuarios/1`

Conclusión

Con Spring Boot, puedes crear servicios escalables y seguros de forma rápida, utilizando herramientas que simplifican la implementación de rutas, controladores y servicios.

Cliente en Java para Consumir la API/REST desarrollada con Spring Boot

Introducción

Cuando se desarrolla una API/REST con Spring Boot, podemos consumirla desde otra aplicación Java utilizando librerías estándar como `HttpURLConnection` o herramientas más avanzadas como **Apache HttpClient** o **Spring WebClient**. Aquí, crearemos un cliente utilizando dos enfoques:

1. **Con `HttpURLConnection` (sin dependencias externas).**
2. **Con `RestTemplate` (si puedes usar Spring Framework en el cliente).**

---

1. Cliente con `HttpURLConnection`

Este cliente se comunica directamente con la API mediante HTTP utilizando las clases estándar de Java.

Código: `ClienteSpringBootAPI.java`

```
package com.miapp.cliente;

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

public class ClienteSpringBootAPI {

    private static final String BASE_URL =
"http://localhost:8080/api/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;
                System.out.println("Usuarios:");
                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 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/json");

            String json =
String.format("{\"nombre\":\"%s\",\"email\":\"%s\"}", nombre,
email);

            OutputStream os = connection.getOutputStream();
            os.write(json.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/json");

            String json =
String.format("{\"nombre\":\"%s\",\"email\":\"%s\"}", nombre,
email);

            OutputStream os = connection.getOutputStream();
            os.write(json.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. Cliente con `RestTemplate`

Este enfoque utiliza **Spring Framework** en el cliente, lo que simplifica la interacción con la API.

Agregar Dependencia al Proyecto

En el archivo `pom.xml`, añade la dependencia de Spring Web:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Código: `RestTemplateCliente.java`

```
package com.miapp.cliente;

import org.springframework.web.client.RestTemplate;

import java.util.HashMap;
import java.util.Map;

public class RestTemplateCliente {

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

    // Obtener todos los usuarios
    public static void obtenerTodosLosUsuarios() {
        RestTemplate restTemplate = new RestTemplate();
        String respuesta = restTemplate.getForObject(BASE_URL,
String.class);
        System.out.println("Usuarios:");
        System.out.println(respuesta);
    }

    // Crear un usuario
    public static void crearUsuario(String nombre, String email) {
        RestTemplate restTemplate = new RestTemplate();
        Map<String, String> usuario = new HashMap<>();
        usuario.put("nombre", nombre);
        usuario.put("email", email);

        String respuesta = restTemplate.postForObject(BASE_URL,
usuario, String.class);
        System.out.println("Respuesta del servidor: " + respuesta);
    }

    // Actualizar un usuario
    public static void actualizarUsuario(int id, String nombre,
String email) {
        RestTemplate restTemplate = new RestTemplate();
        Map<String, String> usuario = new HashMap<>();
        usuario.put("nombre", nombre);
```

```
        usuario.put("email", email);

        restTemplate.put(BASE_URL + "/" + id, usuario);
        System.out.println("Usuario actualizado con éxito.");
    }

    // Eliminar un usuario
    public static void eliminarUsuario(int id) {
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.delete(BASE_URL + "/" + id);
        System.out.println("Usuario eliminado con éxito.");
    }
}
```

---

3. Clase Principal para Probar el Cliente

Código: `TestClientesAPI.java`

```
package com.miapp.cliente;

public class TestClientesAPI {
    public static void main(String[] args) {
        // Usando HttpURLConnection
        System.out.println("=== Cliente con HttpURLConnection ===");
        ClienteSpringBootAPI.obtenerTodosLosUsuarios();
        ClienteSpringBootAPI.crearUsuario("Ana García",
"ana@ejemplo.com");
        ClienteSpringBootAPI.actualizarUsuario(1, "Ana Actualizada",
"ana_actualizada@ejemplo.com");
        ClienteSpringBootAPI.eliminarUsuario(1);

        // Usando RestTemplate
        System.out.println("\n=== Cliente con RestTemplate ===");
        RestTemplateCliente.obtenerTodosLosUsuarios();
        RestTemplateCliente.crearUsuario("Carlos Pérez",
"carlos@ejemplo.com");
        RestTemplateCliente.actualizarUsuario(2, "Carlos
Actualizado", "carlos_actualizado@ejemplo.com");
        RestTemplateCliente.eliminarUsuario(2);
```

```
    }
}
```

Funcionamiento del Cliente

1. **Operaciones Soportadas**:
    - Obtener todos los usuarios (`GET`).
    - Crear un nuevo usuario (`POST`).
    - Actualizar un usuario existente (`PUT`).
    - Eliminar un usuario (`DELETE`).
2. **Ejecución**:
    - Primero, inicia el servidor Spring Boot.
    - Luego, ejecuta `TestClientesAPI` para probar ambos clientes.

Conclusión

Hemos desarrollado dos enfoques para consumir la API de Spring Boot:

1. **Con `HttpURLConnection`**: Para mantener la simplicidad y evitar dependencias externas.
2. **Con `RestTemplate`**: Para aprovechar las capacidades avanzadas de Spring Framework.

Cliente en Java para Consumir la API/REST con Spring WebClient

Introducción

Spring WebClient es una herramienta reactiva y no bloqueante que permite realizar llamadas HTTP de manera eficiente. Es ideal para aplicaciones modernas que necesitan manejar grandes volúmenes de solicitudes o interactuar con servicios externos de forma reactiva.

Configuración del Proyecto

1. Agregar Dependencias

En el archivo `pom.xml`, agrega las dependencias necesarias para **Spring WebFlux**:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

2. Estructura del Proyecto

css
CopiarEditar

```
src/
└── main/
    ├── java/
    │   └── com.miapp.cliente/
    │       ├── WebClientCliente.java
    │       └── TestWebClientAPI.java
```

Cliente con WebClient

El cliente se implementará utilizando `WebClient` para realizar las operaciones **GET**, **POST**, **PUT**, y **DELETE**.

Código: `WebClientCliente.java`

```
package com.miapp.cliente;

import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;

public class WebClientCliente {

    private final WebClient webClient;

    // Constructor
    public WebClientCliente(String baseUrl) {
        this.webClient = WebClient.builder()
                .baseUrl(baseUrl)
                .build();
    }

    // Obtener todos los usuarios
    public void obtenerTodosLosUsuarios() {
```

```
        webClient.get()
                .uri("/usuarios")
                .retrieve()
                .bodyToMono(String.class)
                .doOnError(e -> System.out.println("Error: " +
e.getMessage()))
                .subscribe(respuesta -> {
                    System.out.println("Usuarios:");
                    System.out.println(respuesta);
                });
    }

    // Crear un usuario
    public void crearUsuario(String nombre, String email) {
        Usuario usuario = new Usuario(nombre, email);

        webClient.post()
                .uri("/usuarios")
                .bodyValue(usuario)
                .retrieve()
                .bodyToMono(String.class)
                .doOnError(e -> System.out.println("Error: " +
e.getMessage()))
                .subscribe(respuesta -> System.out.println("Usuario
creado: " + respuesta));
    }

    // Actualizar un usuario
    public void actualizarUsuario(int id, String nombre, String
email) {
        Usuario usuario = new Usuario(nombre, email);

        webClient.put()
                .uri("/usuarios/{id}", id)
                .bodyValue(usuario)
                .retrieve()
                .bodyToMono(String.class)
                .doOnError(e -> System.out.println("Error: " +
e.getMessage()))
                .subscribe(respuesta -> System.out.println("Usuario
actualizado: " + respuesta));
```

```
    }

    // Eliminar un usuario
    public void eliminarUsuario(int id) {
        webClient.delete()
                .uri("/usuarios/{id}", id)
                .retrieve()
                .bodyToMono(String.class)
                .doOnError(e -> System.out.println("Error: " +
e.getMessage()))
                .subscribe(respuesta -> System.out.println("Usuario
eliminado: " + respuesta));
    }

    // Clase interna para representar un Usuario
    static class Usuario {
        private String nombre;
        private String email;

        public Usuario(String nombre, String email) {
            this.nombre = nombre;
            this.email = email;
        }

        public String getNombre() {
            return nombre;
        }

        public String getEmail() {
            return email;
        }
    }
}
```

Clase Principal para Probar el Cliente

Código: `TestWebClientAPI.java`

```
package com.miapp.cliente;
```

```
public class TestWebClientAPI {
    public static void main(String[] args) {
        // Crear una instancia del cliente
        WebClientCliente cliente = new
WebClientCliente("http://localhost:8080/api");

        // Probar las operaciones
        System.out.println("=== Obtener todos los usuarios ===");
        cliente.obtenerTodosLosUsuarios();

        System.out.println("\n=== Crear un nuevo usuario ===");
        cliente.crearUsuario("Lucía Martínez", "lucia@ejemplo.com");

        System.out.println("\n=== Actualizar un usuario ===");
        cliente.actualizarUsuario(1, "Lucía Actualizada",
"lucia_actualizada@ejemplo.com");

        System.out.println("\n=== Eliminar un usuario ===");
        cliente.eliminarUsuario(1);
    }
}
```

Funcionamiento del Cliente

1. **Operaciones Soportadas**:
    - Obtener todos los usuarios (`GET` a `/usuarios`).
    - Crear un nuevo usuario (`POST` a `/usuarios`).
    - Actualizar un usuario existente (`PUT` a `/usuarios/{id}`).
    - Eliminar un usuario (`DELETE` a `/usuarios/{id}`).
2. **Ejecución**:
    - Inicia el servidor Spring Boot (`RestApplication`).
    - Ejecuta `TestWebClientAPI` para probar las funcionalidades del cliente.
3. **Salida Esperada**:
    - Los resultados de cada operación se imprimirán en la consola.

Ventajas de WebClient

- **No bloqueante**: Ideal para aplicaciones que necesitan manejar múltiples solicitudes simultáneamente.
- **Reactivo**: Utiliza `Mono` y `Flux` para gestionar datos de forma eficiente.
- **Fácil Configuración**: Ofrece una API fluida para definir solicitudes HTTP.

Conclusión

Con **Spring WebClient**, puedes consumir APIs de manera eficiente y reactiva, lo que lo hace ideal para aplicaciones modernas. Es más flexible y avanzado que `RestTemplate`, aunque requiere una curva de aprendizaje para entender los conceptos reactivos.