«Talento Tech»

Back-End

Java

Clase 11

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase 11: Desarrollo de la API REST con Spring Boot

Índice

1. Bienvenida a TechLab
2. Objetivos de la Clase
3. Situación Inicial en TechLab
4. Concepto de API REST y Métodos HTTP
    - Endpoints y Recursos
    - Métodos HTTP: GET, POST, PUT, DELETE
5. Creación de Controladores (@RestController)
    - Anotaciones clave: @RestController, @RequestMapping, @GetMapping, etc.
    - Inyección de Dependencias en Controladores
6. Manejo de Rutas y Endpoints Básicos
    - Definir un endpoint para listar productos
    - Filtrar y consultar productos por ID
7. Envío y Recepción de Datos en Formato JSON
    - Jackson y Serialización/Deserialización Automática
    - Pruebas con herramientas como cURL o Postman
8. Implementación de Operaciones CRUD en Memoria
    - GET /productos, POST /productos, PUT /productos/{id}, DELETE /productos/{id}
    - Almacenamiento temporal en una lista en memoria
9. Instalación y Configuración de MySQL y MySQL Workbench
    - Descarga e instalación de MySQL Community Server
    - Creación de una base de datos y conexión futura desde Spring Boot
10. Ejercicios Prácticos
11. Materiales y Recursos Adicionales
12. Preguntas para Reflexionar
13. Próximos Pasos

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Objetivos de la Clase

- Comprender el concepto de **API REST**, sus principios y la utilización de métodos HTTP para distintas operaciones.
- Crear controladores en Spring Boot utilizando `@RestController` para exponer endpoints.
- Configurar rutas para operaciones básicas (listar, obtener por ID, crear, actualizar y eliminar productos).
- Trabajar con datos en formato **JSON**, aprovechando la serialización automática de Spring Boot.
- Implementar operaciones CRUD en memoria como un primer paso, antes de persistir datos en una base real.
- Instalar y configurar **MySQL** y **MySQL Workbench**, preparando el terreno para la persistencia en la próxima clase.

Bienvenida a TechLab

Talento Lab

Ya hemos creado proyectos con Spring Boot, entendido su ecosistema y sentado las bases para un backend robusto. Ahora, llegó el momento de exponer nuestra lógica a través de endpoints que podrán ser consumidos por el frontend del e-commerce u otros clientes, convirtiendo nuestro backend en una **API REST** funcional.

La API REST nos permitirá entregar datos en formato JSON, un estándar de la industria para la comunicación entre servicios. Así, las aplicaciones que consuman la API podrán listar, crear, actualizar y eliminar productos con simples solicitudes HTTP.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Concepto de API REST y Métodos HTTP

Una **API REST** (Representational State Transfer) es un estilo arquitectónico para diseñar servicios web. Los recursos (en este caso, productos) se identifican por URLs, y las operaciones (listar, crear, actualizar, eliminar) se asocian a métodos HTTP:

- **GET:** Obtener recursos.
- **POST:** Crear recursos.
- **PUT:** Actualizar recursos.
- **DELETE:** Eliminar recursos.

Estos métodos son semánticos, lo que hace la API más intuitiva.

Endpoints y Recursos

Un endpoint es una URL que representa un recurso o una acción. Por ejemplo, `GET /productos` podría devolver la lista completa de productos. `GET /productos/1` devolvería el producto con ID 1. Esto facilita la interacción entre cliente y servidor.

Creación de Controladores (@RestController)

En Spring Boot, un controlador REST se define con `@RestController`. Esto indica que los métodos de la clase devolverán el resultado directamente en el cuerpo de la respuesta, usualmente en formato JSON.

Anotaciones clave: @RestController, @RequestMapping, @GetMapping, etc.

- `@RestController`: Indica que la clase es un controlador REST.
- `@RequestMapping`: Define la ruta base del controlador.
- `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`: Definen rutas específicas y métodos HTTP asociados.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Inyección de Dependencias en Controladores

Podés inyectar servicios, repositorios u otras dependencias en el controlador usando `@Autowired` en el constructor o atributos, manteniendo el código modular y limpio.

Manejo de Rutas y Endpoints Básicos

Por ejemplo, para listar productos:

```
@RestController

@RequestMapping("/productos")

public class ProductoController {

    private final ProductoService productoService;

    @Autowired

    public ProductoController(ProductoService productoService) {

        this.productoService = productoService;

    }

    @GetMapping

    public List<Producto> listarProductos() {

        return productoService.listarTodos();

    }

}
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

`GET /productos` devolverá la lista completa. Luego, `GET /productos/{id}` retornará uno en particular. Podemos añadir `@PathVariable` para capturar el ID de la URL:

```
@GetMapping("/{id}")
public Producto obtenerProducto(@PathVariable int id) {
    return productoService.obtenerPorId(id);
}
```

Envío y Recepción de Datos en Formato JSON

Spring Boot utiliza Jackson para serializar y deserializar JSON automáticamente. Si `Producto` es una clase con getters y setters, la respuesta se convertirá en JSON automáticamente sin configuración extra.

Para enviar datos con `POST /productos`, el cliente envía un JSON, y Spring Boot lo convierte en un objeto `Producto` gracias a `@RequestBody`:

java
Copy code

```
@PostMapping
public Producto crearProducto(@RequestBody Producto nuevoProducto) {
    return productoService.guardar(nuevoProducto);
}
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Implementación de Operaciones CRUD en Memoria

Como primer paso, almacenaremos los productos en una lista en memoria dentro de un `ProductoService`. Más adelante, reemplazaremos esta lista con una base de datos real.

```
@Service
public class ProductoService {
    private List<Producto> productos = new ArrayList<>();
    private int contadorId = 1;

    public List<Producto> listarTodos() {
        return productos;
    }

    public Producto obtenerPorId(int id) {
        return productos.stream()
                .filter(p -> p.getId() == id)
                .findFirst()
                .orElse(null);
    }

    public Producto guardar(Producto p) {
        p.setId(contadorId++);
        productos.add(p);
        return p;
    }

    public Producto actualizar(int id, Producto datos) {
        Producto p = obtenerPorId(id);
        if (p != null) {
            p.setNombre(datos.getNombre());
            p.setPrecio(datos.getPrecio());
            p.setCantidadEnStock(datos.getCantidadEnStock());
        }
```

```
        return p;
    }

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

Podemos así exponer `PUT /productos/{id}` y `DELETE /productos/{id}` para actualizar y eliminar, respectivamente.

Instalación y Configuración de MySQL y MySQL Workbench

Para dar el siguiente paso hacia la persistencia real, necesitamos una base de datos. **MySQL** es una opción común y robusta.

1. **Instalar MySQL Community Server:**
   Visitar https://dev.mysql.com/downloads/ y descargar la versión correspondiente.
   Seguir las instrucciones del instalador.
2. **Instalar MySQL Workbench:**
   Una herramienta visual para administrar MySQL. También se descarga desde el mismo sitio.
3. **Configurar la Base de Datos:**
   Una vez instalado, iniciar el servidor MySQL.
   Con MySQL Workbench, conectarse al servidor local, crear una base de datos (ej: `CREATE DATABASE techlab;`).

En la próxima clase, veremos cómo conectar Spring Boot con MySQL a través de Spring Data JPA, mapear entidades, y persistir los productos en la base.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Situación Inicial en TechLab

Silvia, la Product Owner, quiere que el frontend de la tienda en línea obtenga la lista de productos desde el backend, mostrando así datos reales. Matías y Sabrina preparan un controlador Spring Boot que exponga estos datos, primero desde una lista en memoria, y luego conectarán con una base de datos. Antes de dar el paso a la persistencia, es importante sentar las bases de la API y confirmar que la comunicación funciona correctamente.

Ejercicios Prácticos

1. **Endpoints CRUD en Memoria:**
   - Implementá `GET /productos`, `GET /productos/{id}`, `POST /productos`, `PUT /productos/{id}`, `DELETE /productos/{id}` en tu controlador.
   - Probá estos endpoints con Postman o cURL.
2. **Prueba de JSON:**
   - Envía una solicitud `POST /productos` con un cuerpo JSON.
   - Asegurate que el producto se guarde en la lista y que `GET /productos` lo muestre.

Materiales y Recursos Adicionales

- Documentación Oficial de Spring Web:
- Documentación de Spring Boot sobre REST:
- Videos recomendados en YouTube sobre creación de APIs REST con Spring Boot.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Preguntas para Reflexionar

- ¿Cómo facilita Spring Boot la creación de endpoints REST en comparación con una configuración manual?
- ¿Por qué es conveniente trabajar primero con datos en memoria antes de conectar la base de datos real?
- ¿Qué ventajas aporta el manejo de JSON como formato estándar de intercambio de datos con el frontend?

Próximos Pasos

En la próxima clase, profundizaremos en el acceso a datos con MySQL y Spring Data JPA, persistiendo los productos en una base de datos real. Esto permitirá que las modificaciones se mantengan entre reinicios del servidor, dando un paso más hacia una aplicación completa y lista para producción.

¡Felicitaciones por llegar hasta aquí! Con la API REST en funcionamiento, TechLab está más cerca de ofrecer un servicio web robusto, escalable y preparado para la interacción con clientes externos.

Buenos Aires Ciudad