El futuro digital es de todos

MinTIC

UNIVERSIDAD EL BOSQUE

MisiónTIC 2022

Recurso 6

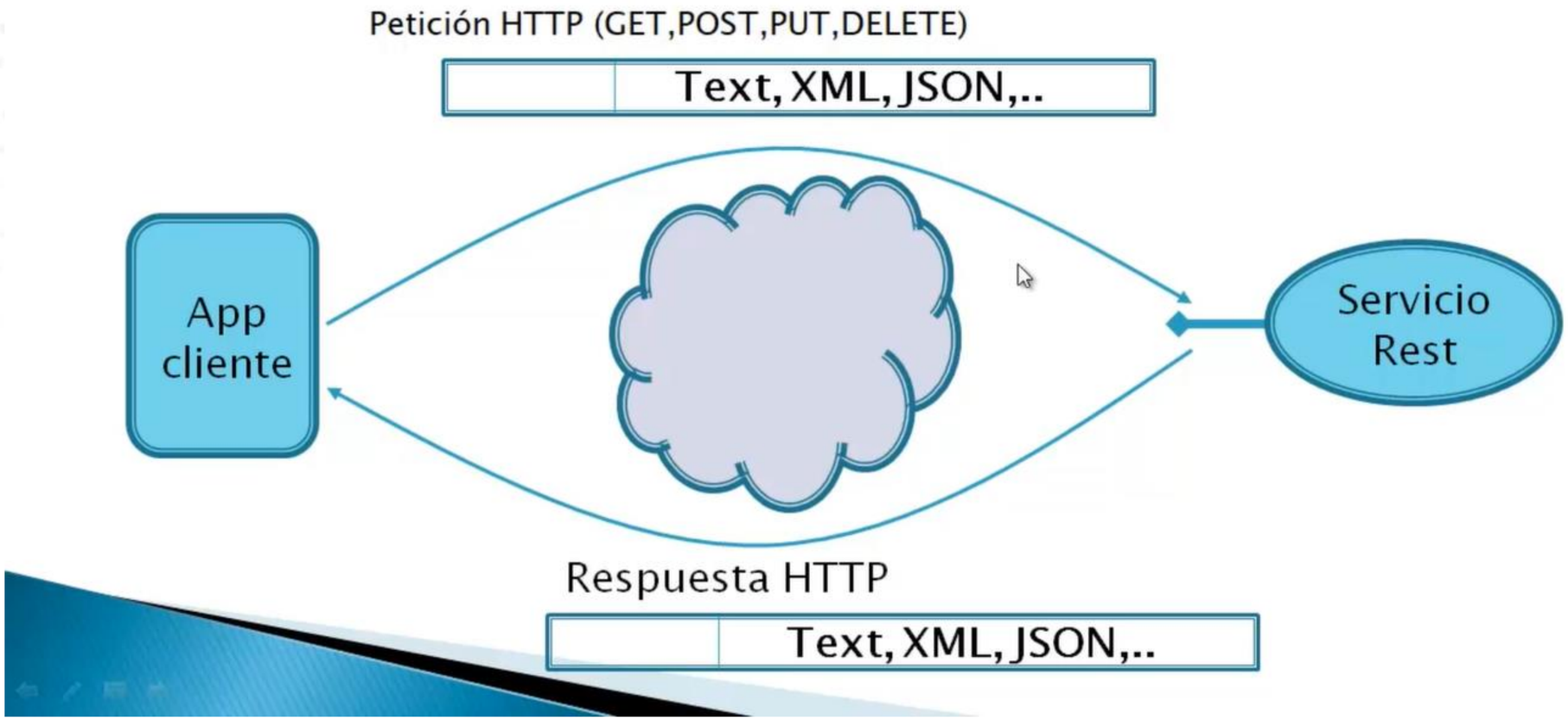
Servicios REST

Mag. Carlos Adolfo Beltrán Castro

Fundamentos REST

- Los servicios REST (Representational State Transfer), exponen a través de Internet una serie de recursos a los que se puede acceder con peticiones HTTP sencillas (tipo GET, PUT, POST, DELETE, etc.)
- Los recursos consisten en datos que se ofrecen (pedidos, pedido de un determinado identificador) u operaciones sobre esos datos (modificar un pedido, añadir un pedido,..)
- Cada recurso se identifica por: URL, método HTTP, parámetros, tipo de respuesta, tipo consumido

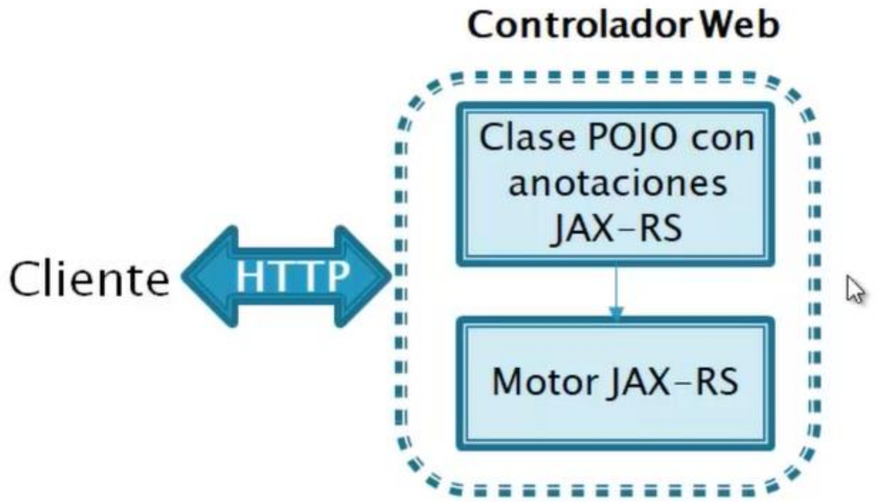
Formato de intercambio de datos



El futuro digital es de todos | MinTIC | UNIVERSIDAD EL BOSQUE | MisiónTIC 2022

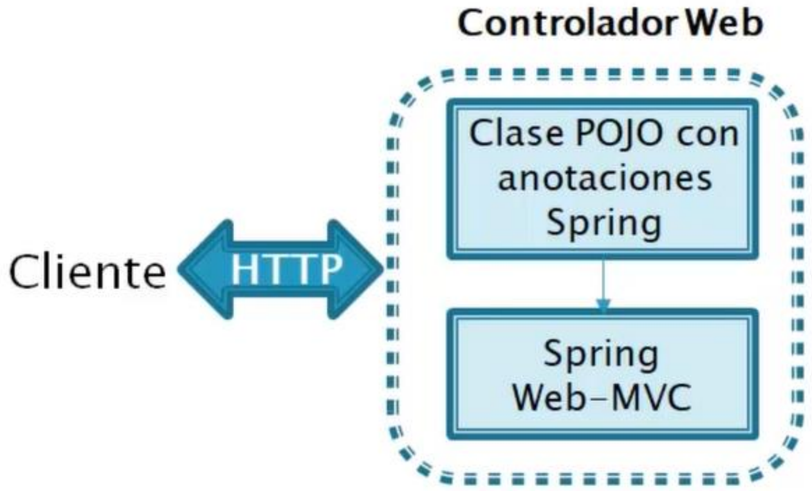
El API JAX-RS

- Especificación de Java EE que simplifica la creación del adaptador Web en servicios REST Java
- Incluye anotaciones para delegar tareas en el motor JAX-RS



Servicios REST con Spring

- El módulo Web-MVC proporciona el soporte necesario para la creación de servicios REST con Spring
- Incluye anotaciones específicas de Spring



Estructura controlador REST

- Clase POJO con anotaciones spring MVC

```
@RestController
public class ClaseServicio{
        @GetMapping(..)
        public ..metodo(..){..}

        @PostMapping(..)
        public .. metodo2(..){..}
        :
}
```

Principales anotaciones Spring

- **@RestController**. Indica que la clase es un controlador REST.

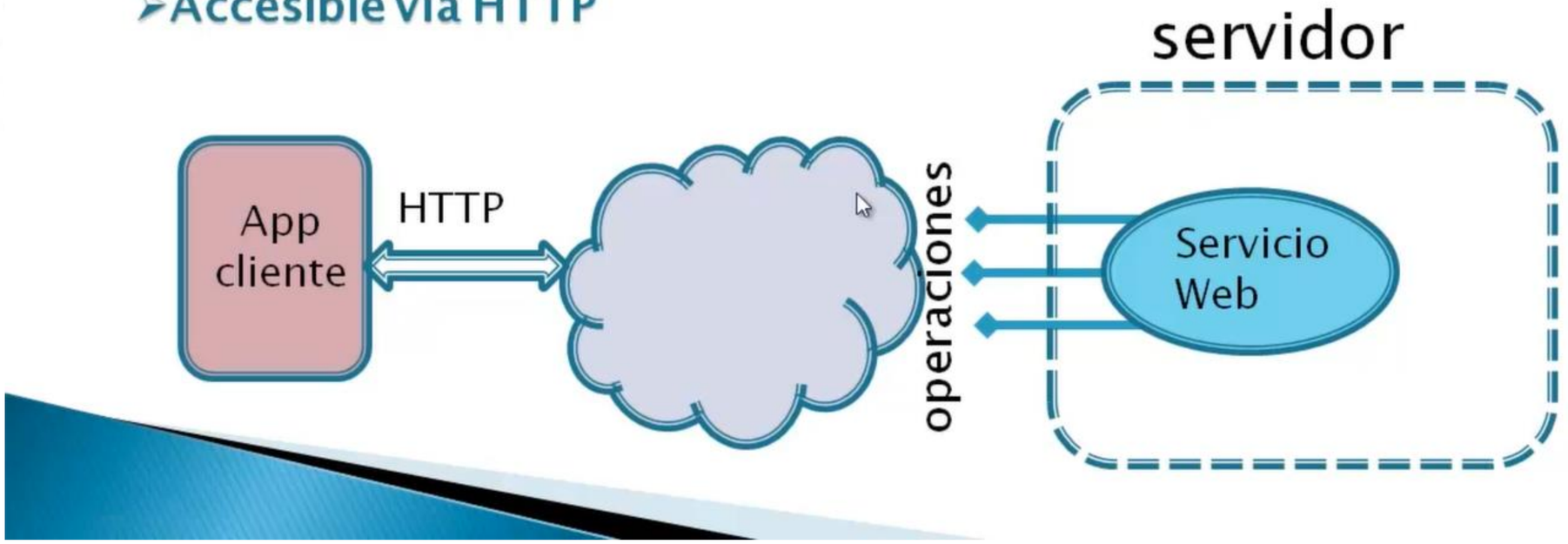
- **@GetMapping, @PostMapping, @PutMapping @DeleteMapping**. Asocian a los métodos del servicio un determinado método HTTP. A través de su atributo *value*, se indica también la url a la que se asociará el método.

- **@PathVariable**. Asocia una variable de la URL a un parámetro de método.

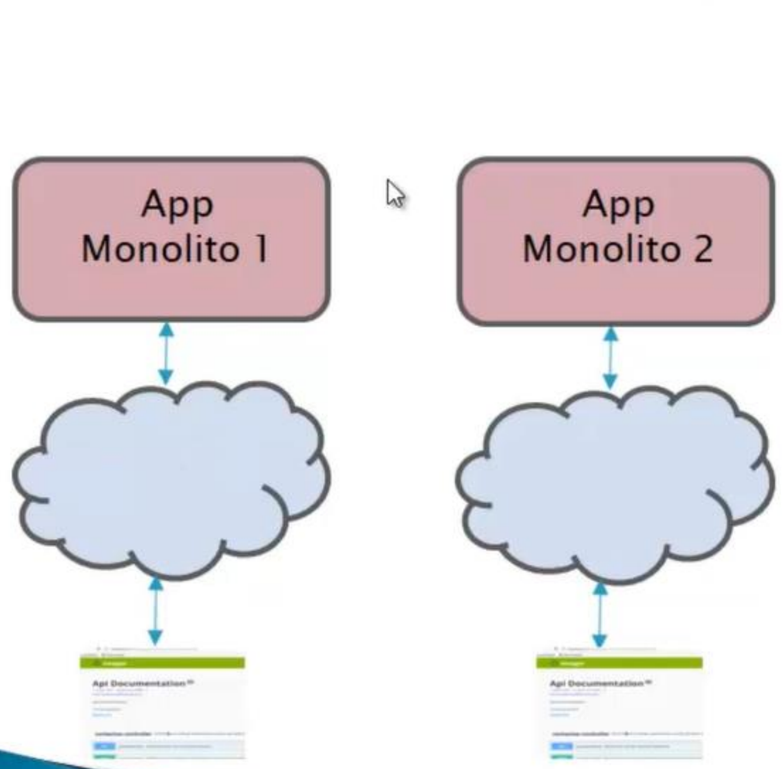
- **@RequestBody**. Asocia el contenido del cuerpo de la petición a un parámetro objeto, dentro del método de respuesta.

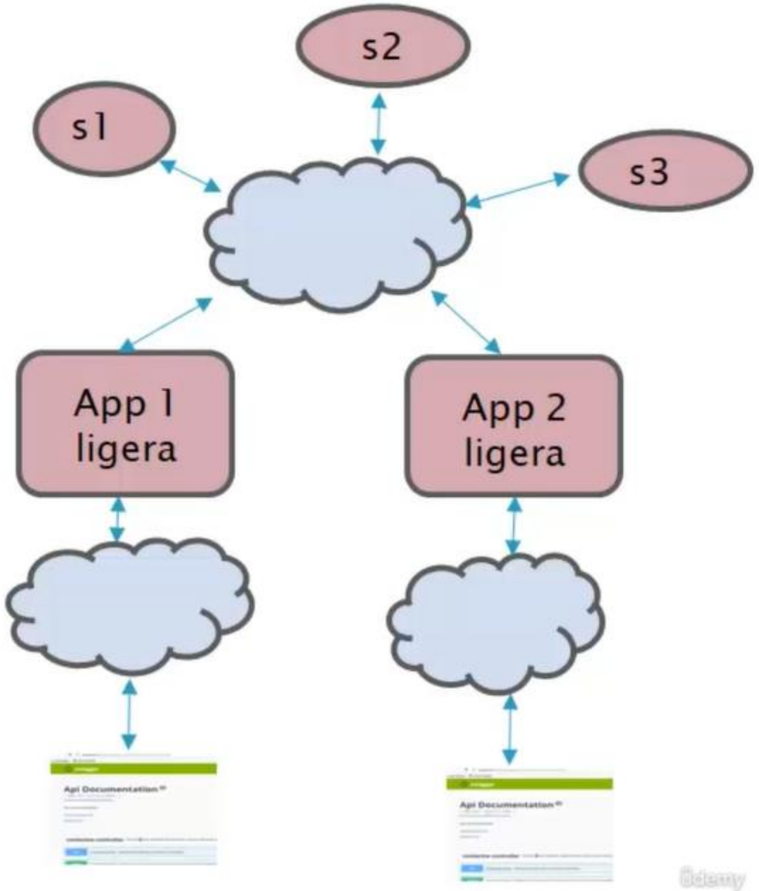
¿Qué es un servicio Web?

- Componente que expone operaciones a otros programas a través de la Web
- Accesible vía HTTP



Servicios y aplicaciones clásicas





Ventajas servicios Web

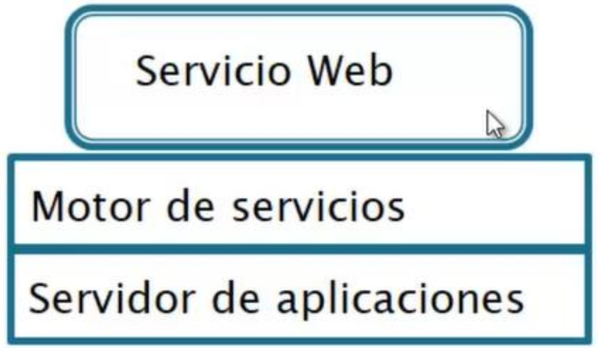
- Reutilización de código.
- Independencia de la plataforma
- Escalabilidad
- Mejora de rendimiento

Estructura de un servicio Web

- **Lógica de negocio**. Implementación de las operaciones
- **Controlador Web**. Interacción con el exterior: Mapeo de datos nativos a formato de intercambio (XML, JSON,..), gestión de conexiones, etc.

Entorno de ejecución

- El entorno de ejecución de un servicio Web lo proporciona:
  - **Motor de servicios.** Librería que proporciona utilidades e implementaciones de objetos para simplificar el trabajo del programador.
  - **Servidor de aplicaciones.** Como cualquier aplicación Web, se ejecuta sobre un servidor de aplicaciones



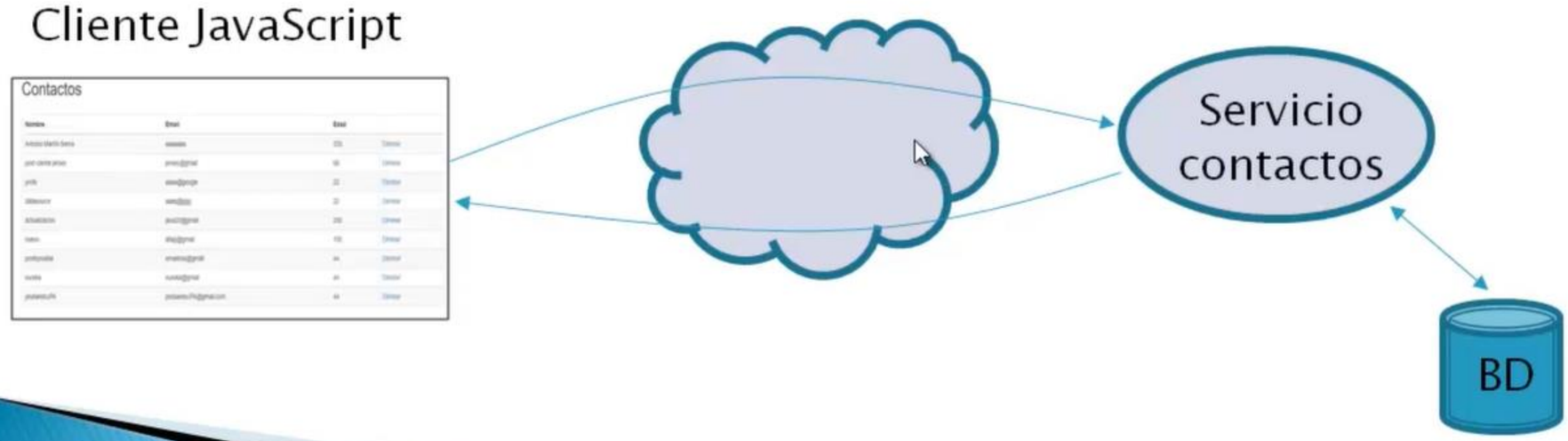
Aproximaciones

- *Servicios Web XML*. Se basan en el intercambio de documentos XML entre el servicio y su aplicación cliente.

- *Servicios Rest*. Los servicios web exponen una serie de recursos a los que se accede simplemente con una petición HTTP, sin necesidad de enviar ningún documento XML adicional. Los servicios REST pueden aceptar datos y devolver resultados en múltiples formatos

Descripción

- Servicio: Aplicación Web con Spring MVC ejecutándose sobre servidor de aplicaciones Tomcat
- Cliente: Script ejecutándose en página HTML



¿Qué es un microservicio?

- Un microservicio es una unidad de software que implementa una funcionalidad concreta y se ejecuta de forma autónoma e independiente
- Incluye todo lo necesario para su ejecución, sin depender de ningún software adicional
- Despliegue automático e independiente

Microservicio vs servicio

App servicio Rest estándar

App microservicio

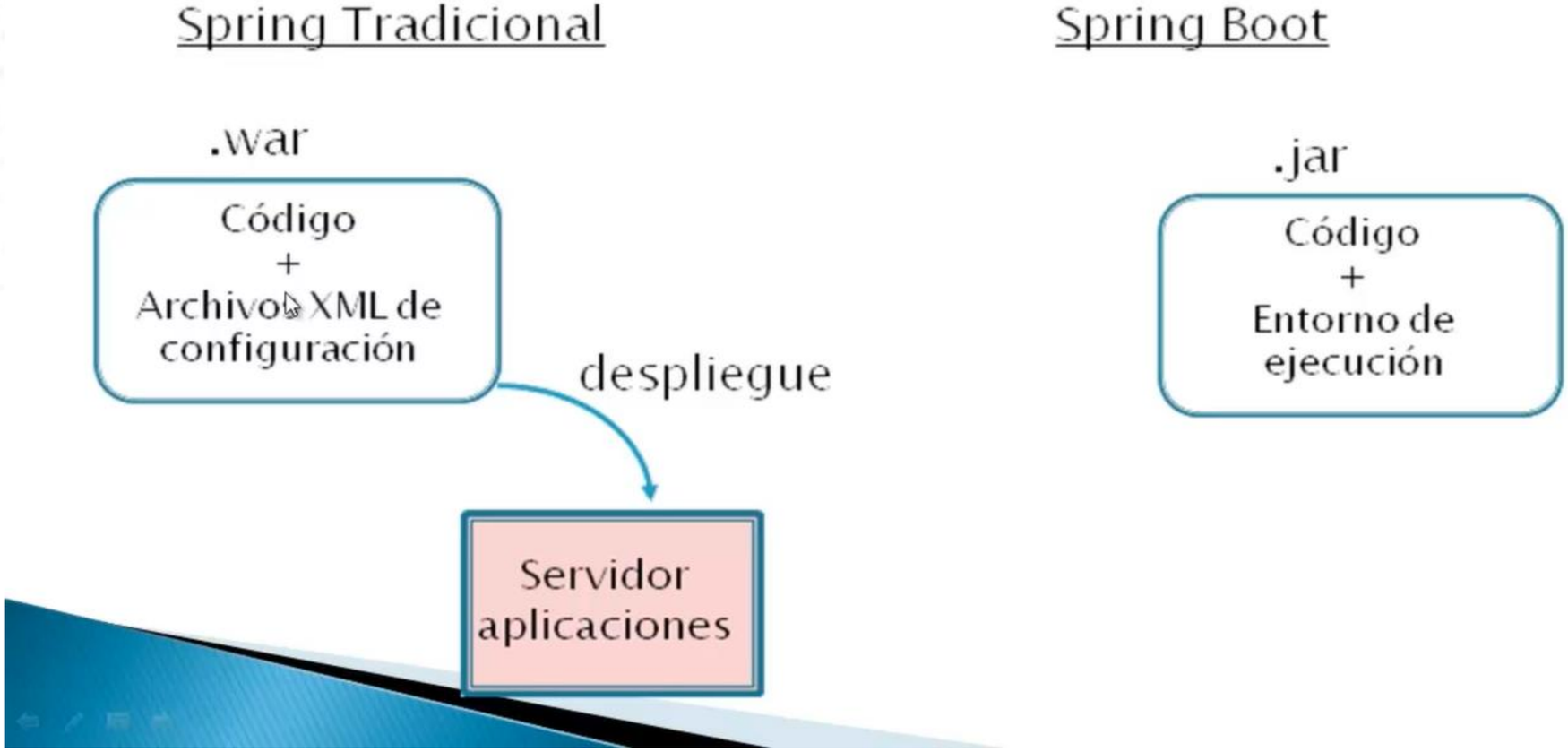
Ventajas uso de microservicios

- Facilita el escalado de la aplicación
- Mantenimiento amigable y sencillo
- Reutilización
- Tolerancia a errores

Spring Boot

- Simplifica el proceso de configuración de aplicaciones
- Permite integrar entorno de ejecución en la aplicación
- Aplicaciones Java estándar (.jar)
- Ideal para creación de microservicios

Microservicios con Spring Boot



Mag. Carlos Adolfo Beltrán Castro

Dependencias Maven

- La inclusión de dependencias en una aplicación Spring boot se simplifica mediante los starters
- Un starter incluye un conjunto de dependencias maven para desarrollar un tipo de aplicación
- Ejemplo:

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

Configuración de aplicaciones

- Se eliminan los archivos de configuración .xml
- Se asumen una serie de configuraciones por defecto
- Para indicar parámetros de configuración adicionales:
  - application.properties
  - application.yml

El futuro digital es de todos | MinTIC

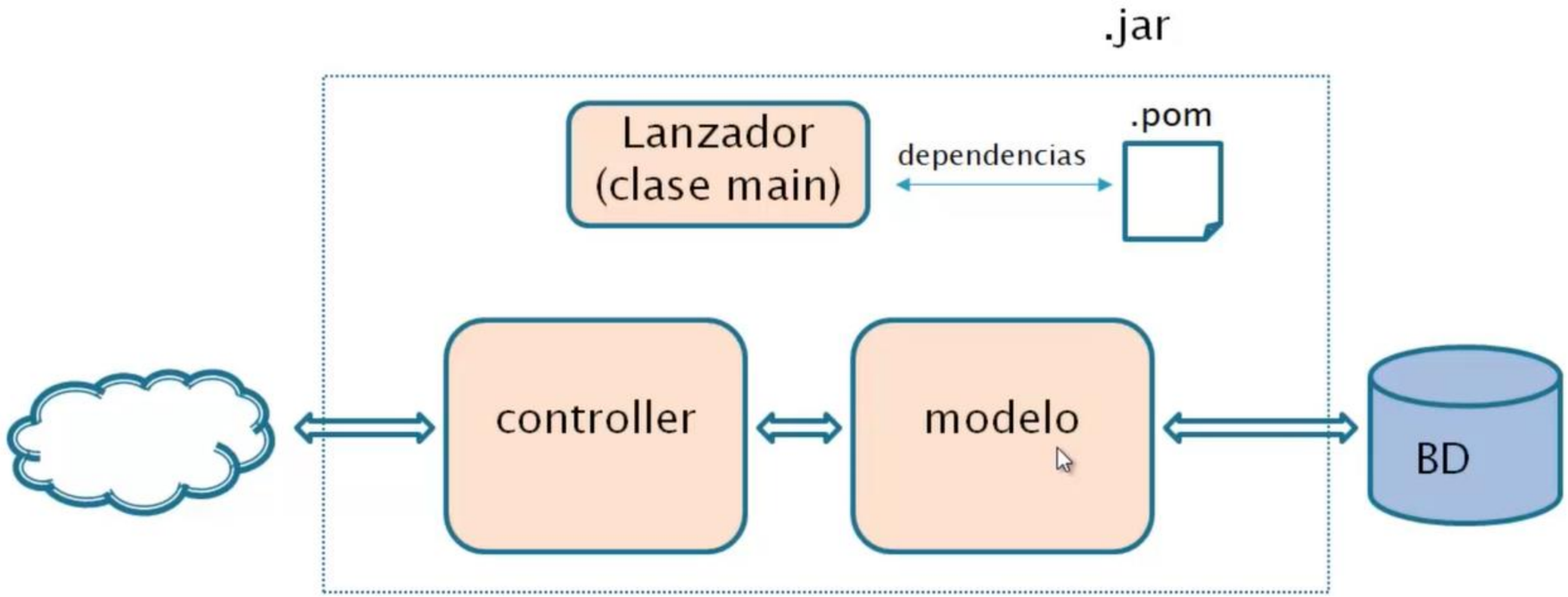
UNIVERSIDAD EL BOSQUE

MisiónTIC 2022

Estructura de un microservicio con Spring Boot

Mag. Carlos Adolfo Beltrán Castro

Estructura de aplicación



Mag. Carlos Adolfo Beltrán Castro

La clase main

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

- @SpringBootApplication:
  - @Configuration
  - @EnableAutoConfiguration
  - @ComponentScan

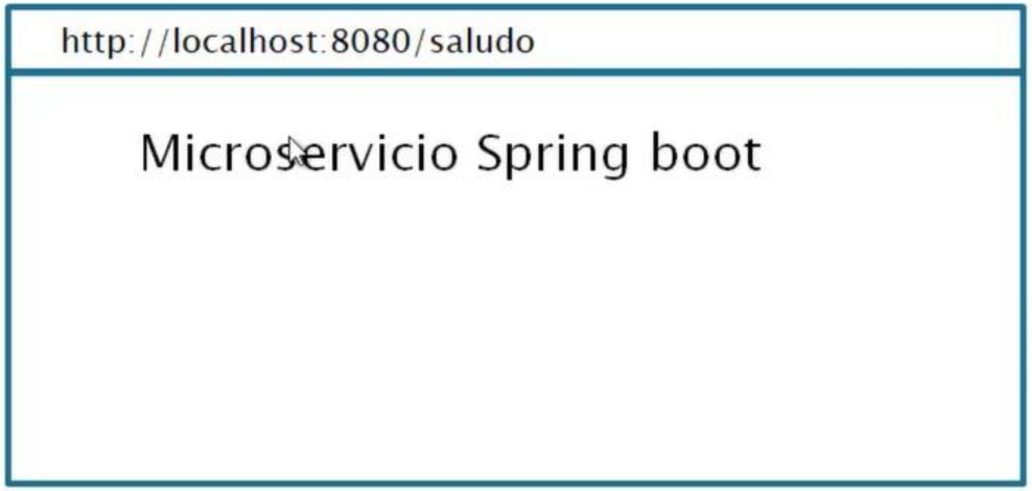
Controlador Rest

- Clase POJO que define los métodos para atender las peticiones al servicio

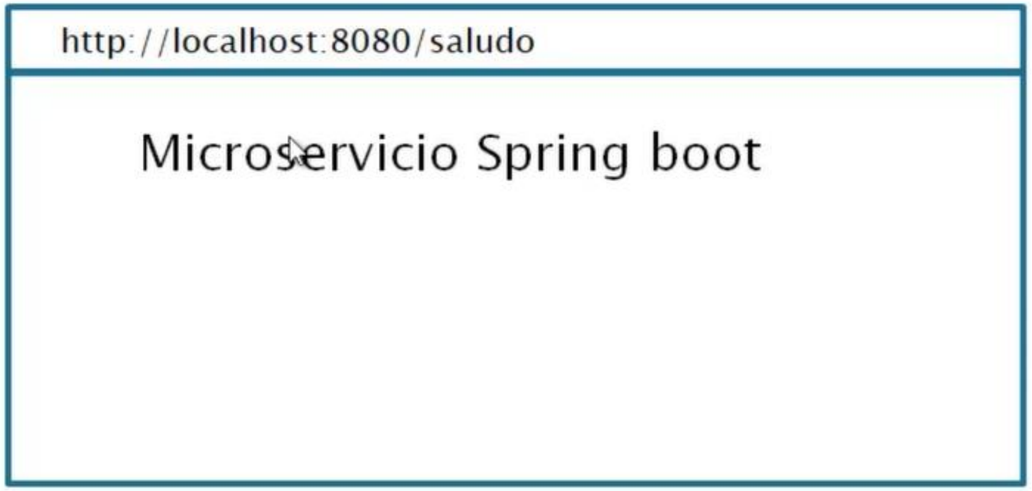
url asociada al recurso

Tipo de devolución

```
@RestController
public class Controller {
        @GetMapping(value="prueba", produces="MediaType.TEXT_PLAIN_VALUE")
        public String metodo(){
                return "Bienvenido a mi servicio";
        }
}
```



VER EJEMPLO

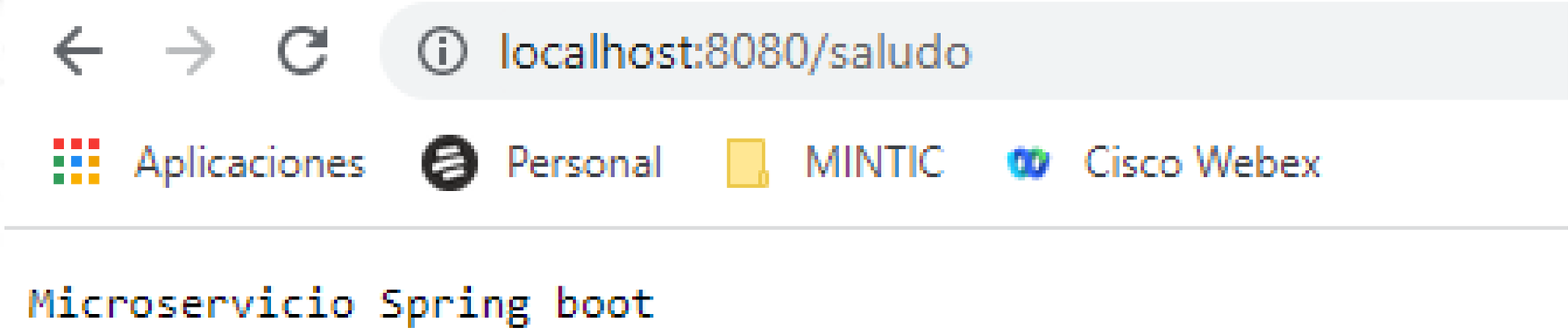


VER EJEMPLO

- Crear Proyecto Spring Boot con solo dependencia web
- Agregar controlador con el siguiente contenido:

```java
package com.boot.ejemplo;

import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SaludoController {
    @GetMapping(value = "saludo",produces=MediaType.TEXT_PLAIN_VALUE)
    public String saludo() {
        return "Microservicio Spring boot";
    }
}

```

- Probamos el proyecto en un navegador con localhost:8080/saludo
- El resultado debe ser el siguiente:



Propiedades de configuración

Mag. Carlos Adolfo Beltrán Castro

application.properties

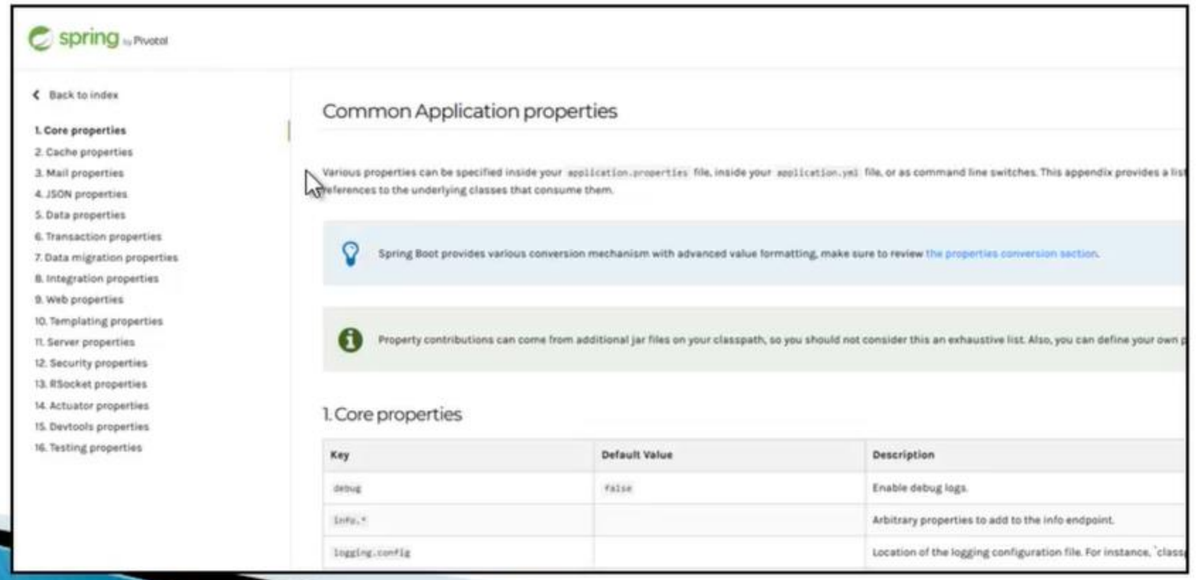
- Define propiedades de configuración de la aplicación Spring Boot como parejas nombre=valor

```
server.port=8000
server.servlet.context-path=/personasdev
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/libros
spring.datasource.username=root
spring.datasource.password=root
```

Información sobre propiedades

- Podemos encontrar información sobre las propiedades más importantes en:

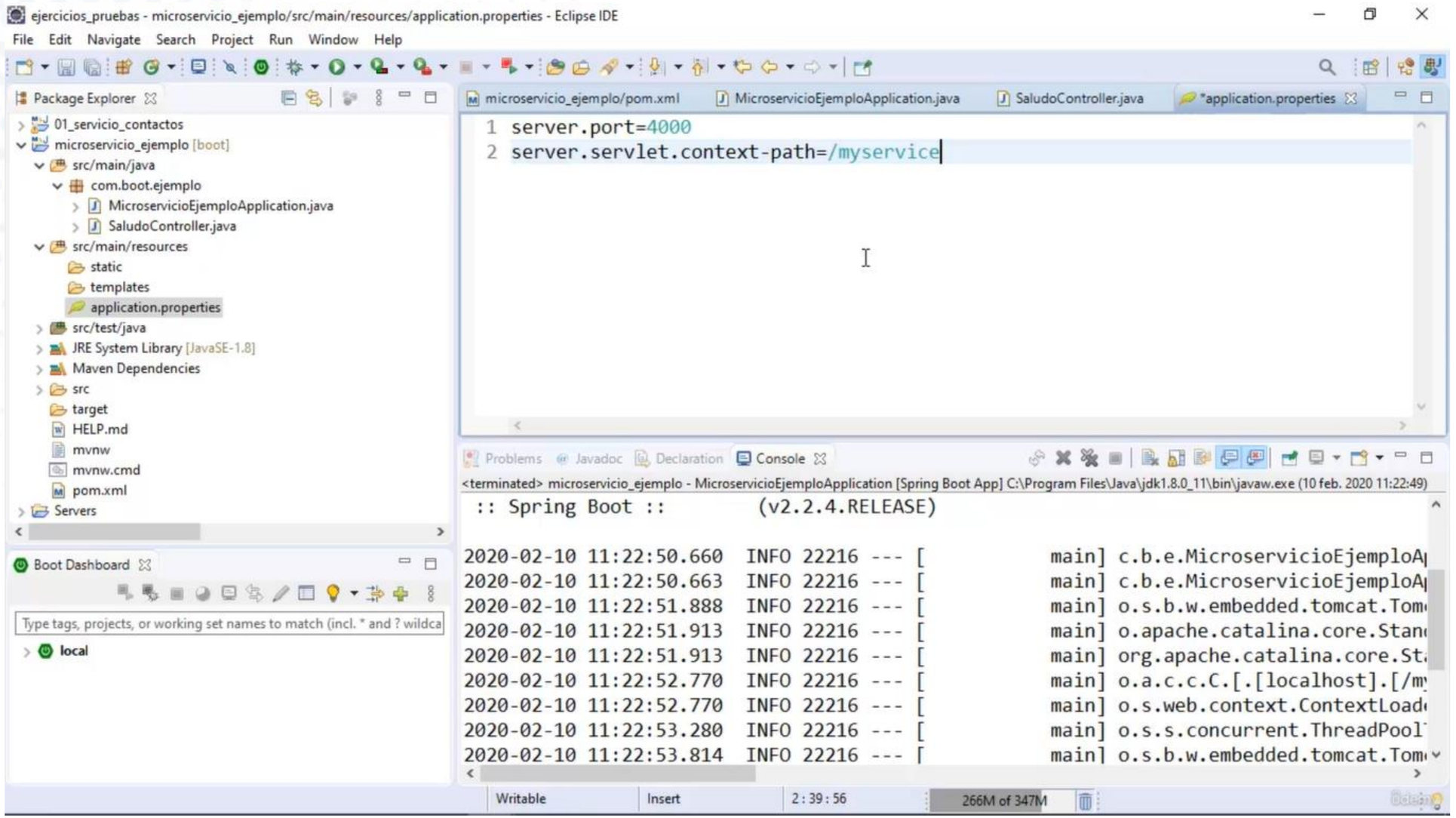
https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html



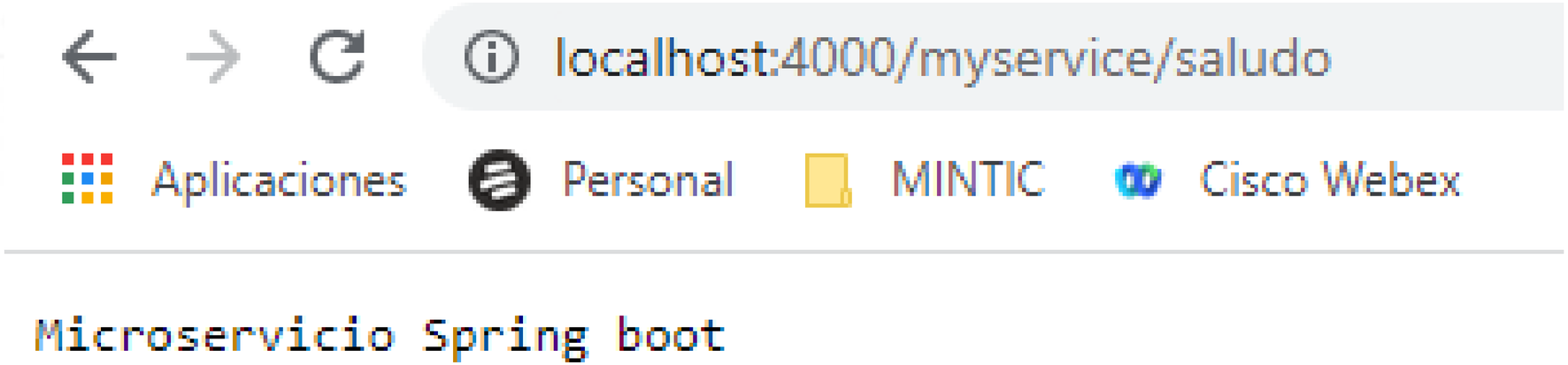
Mag. Carlos Adolfo Beltrán Castro

Algunas propiedades habituales

- server.port. Puerto de escucha utilizado por el servidor embebido.
- server.servlet.context-path. Context path o dirección raíz de la aplicación.
- spring.config.name. Nombre del archivo de configuración, por defecto application.


ejercicios_pruebas - microservicio_ejemplo/src/main/resources/application.properties - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Package Explorer
01_servicio_contactos
microservicio_ejemplo [boot]
src/main/java
com.boot.ejemplo
MicroservicioEjemploApplication.java
SaludoController.java
src/main/resources
static
templates
application.properties
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
Servers
Boot Dashboard
Type tags, projects, or working set names to match (incl. * and ? wildca
local
microservicio_ejemplo/pom.xml
MicroservicioEjemploApplication.java
SaludoController.java
*application.properties
1 server.port=4000
2 server.servlet.context-path=/myservice
Problems Javadoc Declaration Console
<terminated> microservicio_ejemplo - MicroservicioEjemploApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_11\bin\javaw.exe (10 feb. 2020 11:22:49)
:: Spring Boot :: (v2.2.4.RELEASE)
2020-02-10 11:22:50.660 INFO 22216 --- [main] c.b.e.MicroservicioEjemploA
2020-02-10 11:22:50.663 INFO 22216 --- [main] c.b.e.MicroservicioEjemploA
2020-02-10 11:22:51.888 INFO 22216 --- [main] o.s.b.w.embedded.tomcat.Tom
2020-02-10 11:22:51.913 INFO 22216 --- [main] o.apache.catalina.core.Stan
2020-02-10 11:22:51.913 INFO 22216 --- [main] org.apache.catalina.core.St
2020-02-10 11:22:52.770 INFO 22216 --- [main] o.a.c.c.C.[.[localhost].[/m
2020-02-10 11:22:52.770 INFO 22216 --- [main] o.s.web.context.ContextLoad
2020-02-10 11:22:53.280 INFO 22216 --- [main] o.s.s.concurrent.ThreadPool
2020-02-10 11:22:53.814 INFO 22216 --- [main] o.s.b.w.embedded.tomcat.Tom
Writable Insert 2 : 39 : 56 266M of 347M

- Probamos el proyecto en un navegador con localhost:4000/myservice/saludo
- El resultado debe ser el siguiente:



Envío de datos

Mag. Carlos Adolfo Beltrán Castro

Variables en URL

- Datos que se envían como parte de la URL, incluyéndolos a continuación de la dirección del servicio

Recogida de variables

- Los variables de la URL se deben mapear a los parámetros del método que procesa la petición mediante @PathVariable

```
@RestController
public class TestService{
        @GetMapping(value="saludo/{x}/{y}",
                produces(MediaType.TEXT_PLAIN_VALUE)
        public String saludar(@PathVariable("x")  String a,
                              @PathVariable("y")  int b){
                :
        }
}
```

Parámetros en QueryString

- Los parámetros se envían en parejas nombre=valor, separados de la dirección por una ?

Recogida de parámetros

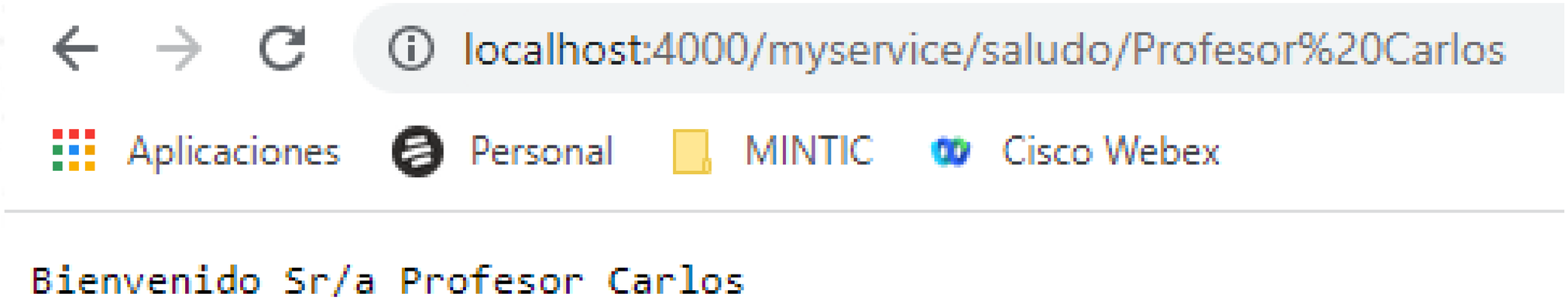
- Los parámetros querystring se mapean a parámetros del método mediante @RequestParam

```
@RestController
public class TestService{
        @GetMapping(value="saludo",
                produces(MediaType.TEXT_PLAIN_VALUE)
        public String saludar(@RequestParam("x")  String a,
                        @ RequestParam("y")  int b){
                :
        }
}
```

- Modificamos el controlador agregando un nuevo método pero asociado a la misma url pero agregamos una variable:

```
15    @GetMapping(value = "saludo/{name}",produces=MediaType.TEXT_PLAIN_VALUE)
16    public String saludo(@PathVariable("name") String n) {
17        return "Bienvenido Sr/a "+n;
18    }
```

- Probamos el proyecto en un navegador con localhost:4000/myservice/saludo/profesor Carlos
- El resultado debe ser el siguiente:



Mapeado de objetos

Mag. Carlos Adolfo Beltrán Castro

Transformación a JSON

- A través de la librería Jackson, Spring mapea automáticamente un JavaBean a JSON y viceversa

```
public class Persona{
        private String nombre;
        private String email;
        private int edad;

        public Persona() {}
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
:

}
```

⇨

```
{"nombre":"profe",
"email":"aaa@gmail.com",
"edad":23}
```

Implementación del servicio

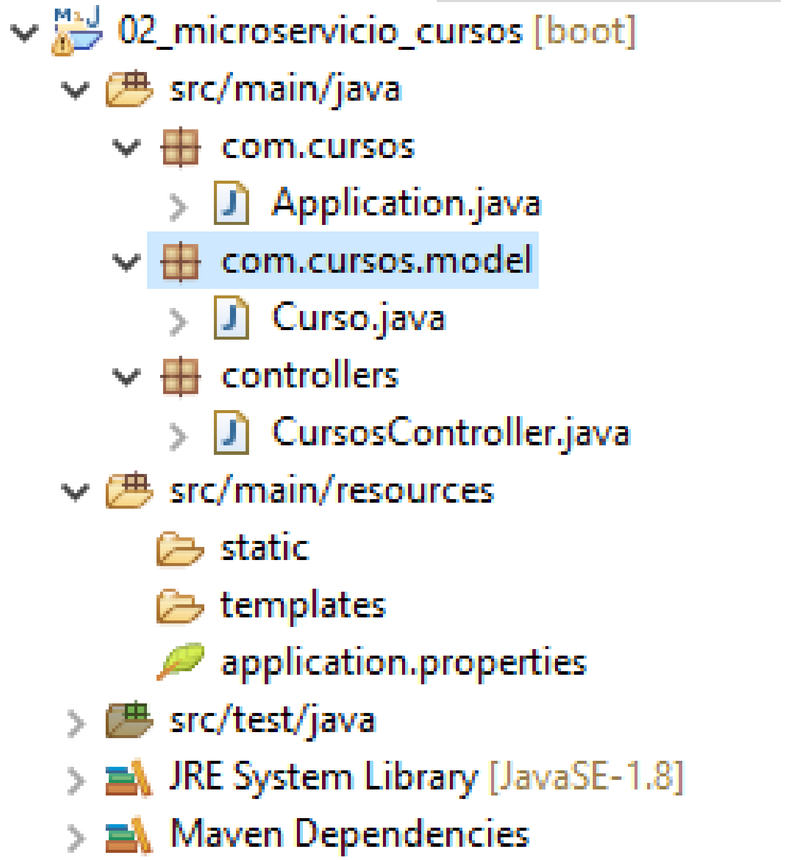
- A través del atributo produces, le indicamos el formato al que tiene que trasformar el objeto en la respuesta
- El tipo de devolución en el método es el tipo JavaBean

```
public class TestService{
        @GetMapping( value=.. produces=MediaType.APPLICATION_JSON_VALUE)
        public Persona datosPersona(){
                :
        }
}
```

Descripción

- Expondrá tres recursos para búsqueda de cursos
  - Devuelve un curso (nombre, duración y horario) cualquiera
  - Devuelve la lista de cursos existentes
  - Devuelve los cursos de un determinado nombre

- Creamos un nuevo Proyecto web con Spring Boot básico, la estructura se puede apreciar en la siguiente figura:



- Clase Curso

```java
package com.cursos.model;

public class Curso {

    // Atributos
    private String nombre;
    private int duracion;
    private String horario;

    // Constructores
    public Curso(String nombre, int duracion, String horario) {
        super();
        this.nombre = nombre;
        this.duracion = duracion;
        this.horario = horario;
    }
    public Curso() {

    }

    // Setter y Getter
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public int getDuracion() {
        return duracion;
    }
    public void setDuracion(int duracion) {
        this.duracion = duracion;
    }
    public String getHorario() {
        return horario;
    }
    public void setHorario(String horario) {
        this.horario = horario;
    }

}
```

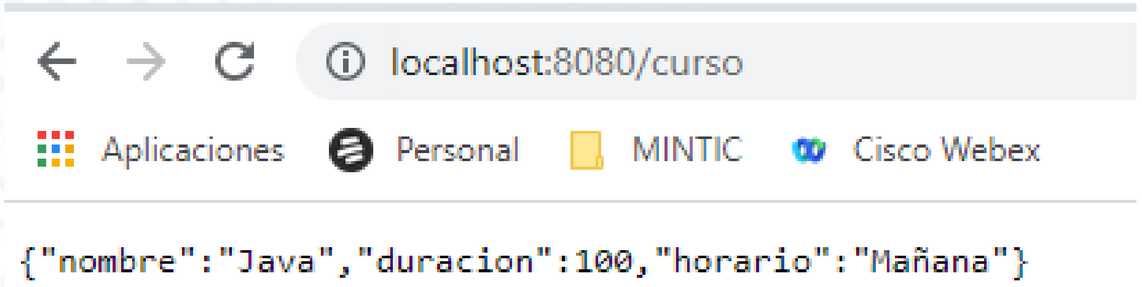
- Controlador

```java
package controllers;

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

import javax.annotation.PostConstruct;

import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import com.cursos.model.Curso;

@RestController
public class CursosController {

    // Lista de todos los cursos
    private List<Curso> cursos;
    @PostConstruct
    public void init() {
        cursos=new ArrayList<>();
        cursos.add(new Curso("Spring",25,"tarde"));
        cursos.add(new Curso("Spring boot",20,"tarde"));
        cursos.add(new Curso("Python",30,"tarde"));
        cursos.add(new Curso("Java EE",50,"fin de semana"));
        cursos.add(new Curso("Java básico",30,"mañana"));
    }

    @GetMapping(value="cursos",produces=MediaType.APPLICATION_JSON_VALUE)
    public List<Curso> getCursos(){
        return cursos;
    }

    // Listar el Curso Java
    @GetMapping(value="curso",produces=MediaType.APPLICATION_JSON_VALUE)
    public Curso getCurso() {
        return new Curso("Java",100,"Mañana");
    }

    // Mostrar los datos de cierto curso
    @GetMapping(value="cursos/{name}",produces=MediaType.APPLICATION_JSON_VALUE)
    public List<Curso> buscarCursos(@PathVariable("name") String nombre){
        List<Curso> aux=new ArrayList<>();
        for(Curso c:cursos) {
            if(c.getNombre().contains(nombre)) {
                aux.add(c);
            }
        }
        return aux;
    }
}
```

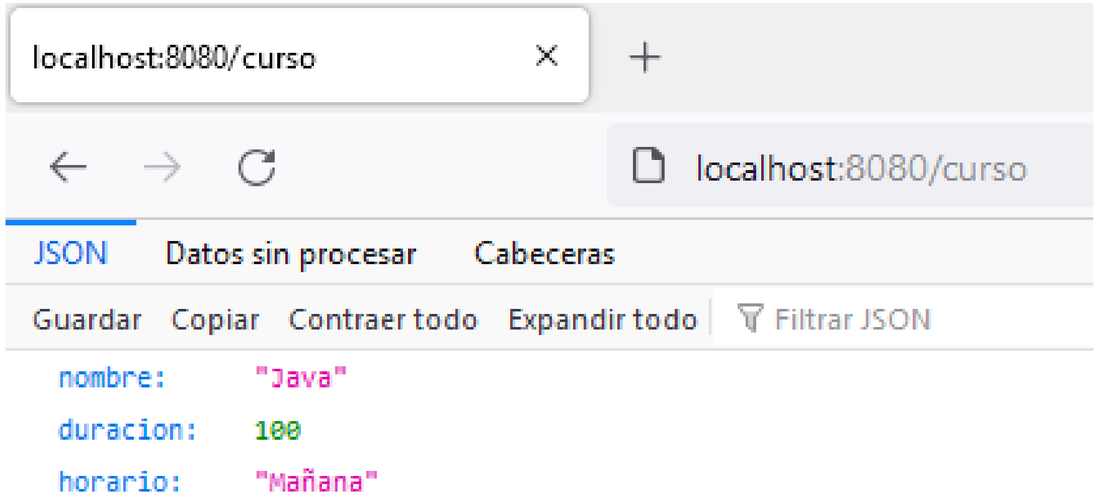
- Cambio en el archivo de ejecución porque la url no es reconocida debido que está por fuera del package de la clase principal.

```java
package com.cursos;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication
@ComponentScan(basePackages = {"controllers"})
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

- Probar el proyecto:

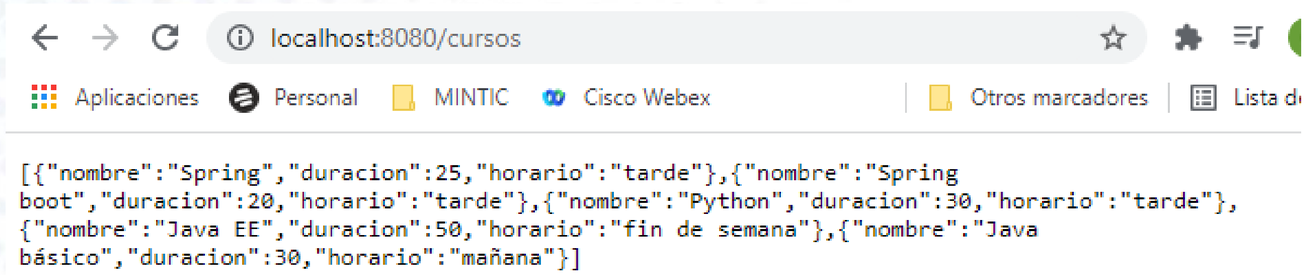


Google Chrome



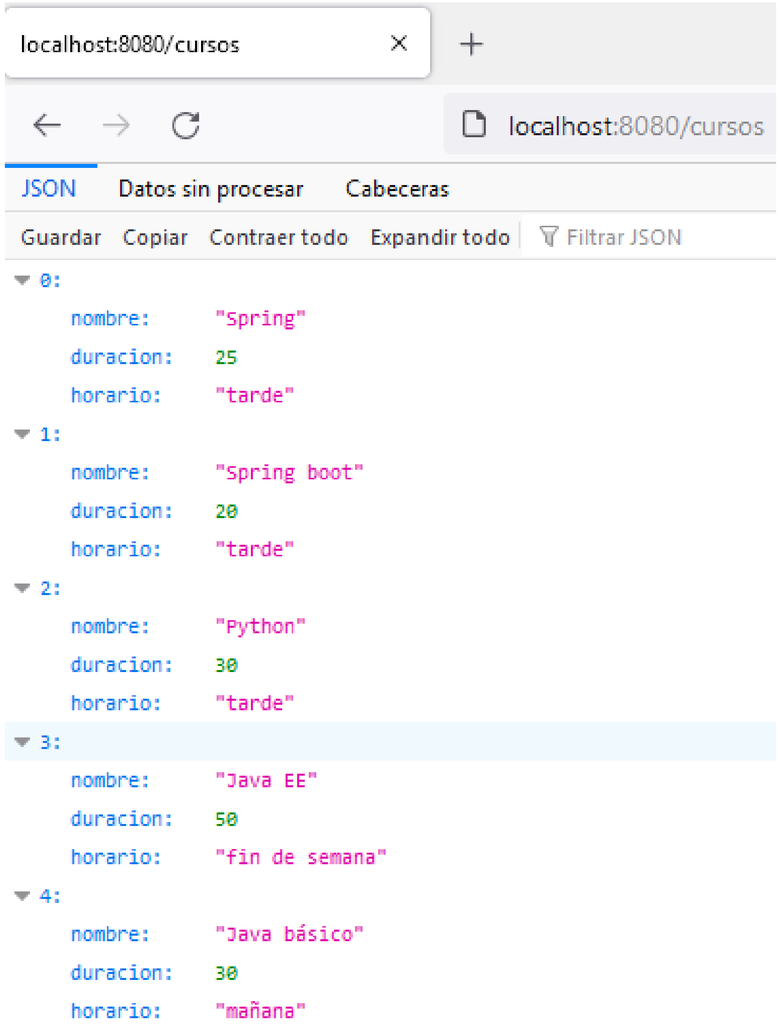
Mozzilla Firefox

- Probar el proyecto:



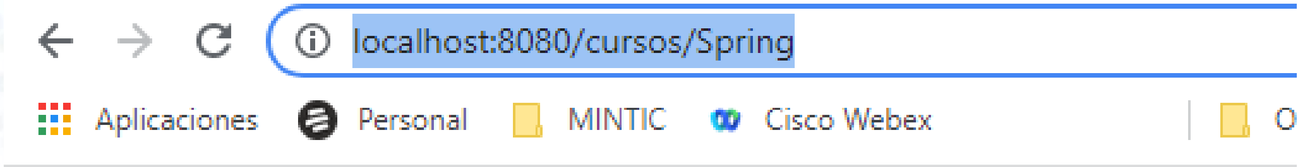

```
[{"nombre":"Spring","duracion":25,"horario":"tarde"},{"nombre":"Spring
boot","duracion":20,"horario":"tarde"},{"nombre":"Python","duracion":30,"horario":"tarde"},
{"nombre":"Java EE","duracion":50,"horario":"fin de semana"},{"nombre":"Java
básico","duracion":30,"horario":"mañana"}]
```


Google Chrome



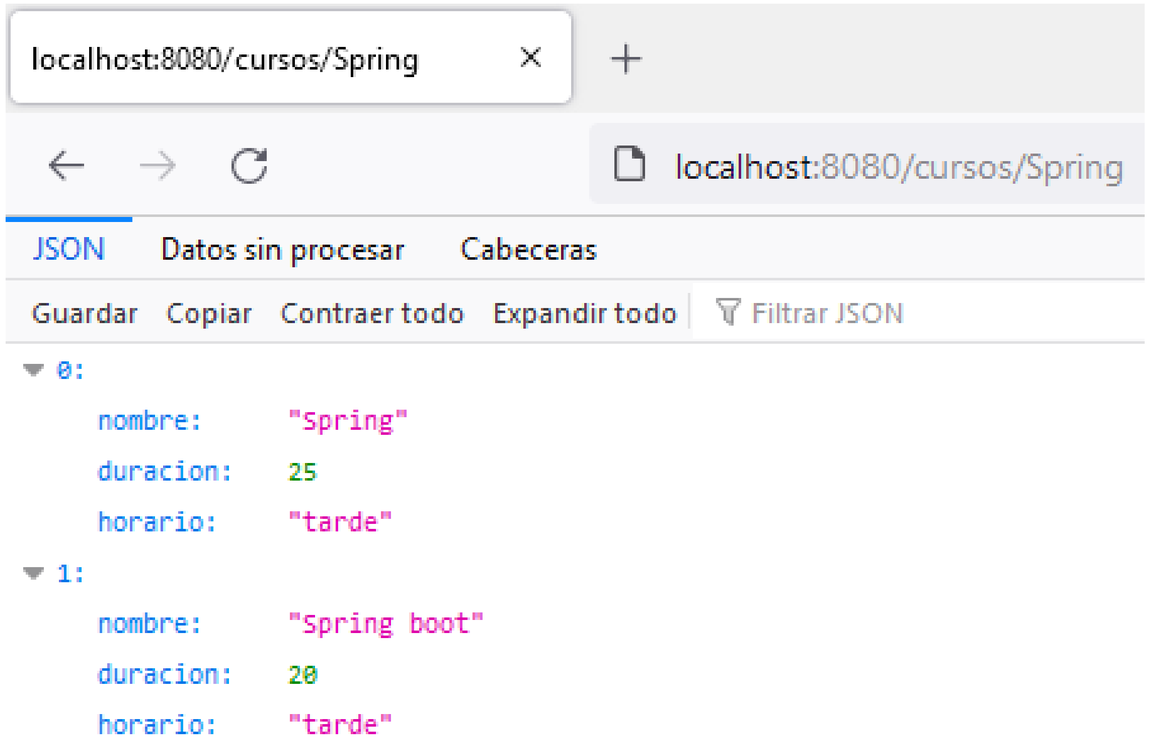
Mozzilla Firefox

- Probar el proyecto:




```
[{"nombre":"Spring","duracion":25,"horario":"tarde"},{"nombre":"Spring boot","duracion":20,"horario":"tarde"}]
```


Google Chrome



Mozzilla Firefox

Mapeado a XML

- La clase bean se anota con @XmlRootElement

```
@XmlRootElement
public class Persona{
    :
}
```

```
public class TestService{
        @GetMapping( value=..  produces=MediaType.APPLICATION_XML_VALUE)
        public Persona datosPersona(){
                :
        }
}
```

El futuro digital es de todos

MinTIC

UNIVERSIDAD EL BOSQUE

MisiónTIC 2022

Respuesta a diferentes métodos HTTP

Mag. Carlos Adolfo Beltrán Castro

DELETE

- Normalmente asociado a operaciones de eliminación de recursos
- Se asocia al método a través de @DeleteMapping

```
@DeleteMapping(value=..  produces=..)
public tipo metodoDelete(){
        //operación de eliminación
}
```

POST

- Normalmente asociado a operaciones de inserción de recursos
- En el cuerpo de la petición se envía en formato JSON o XML el objeto a insertar. Mediante el atributo consumes de @PostMapping se indica el tipo.
- El cuerpo se mapea a un bean mediante @RequestBody

```
@PostMapping( value=..  consumes=MediaType.APPLICATION_JSON_VALUE)
public void metodoPost (@RequestBody Persona p){
        //operación de inserción del objeto Persona
}
```

El tipo de devolución puede ser void, o de un tipo específico, en cuyo caso se indicará en ***produces***

PUT

- Normalmente asociado a operaciones de actualización de recursos
- Como en POST, en el cuerpo de la petición se envía en formato JSON o XML el objeto a insertar.

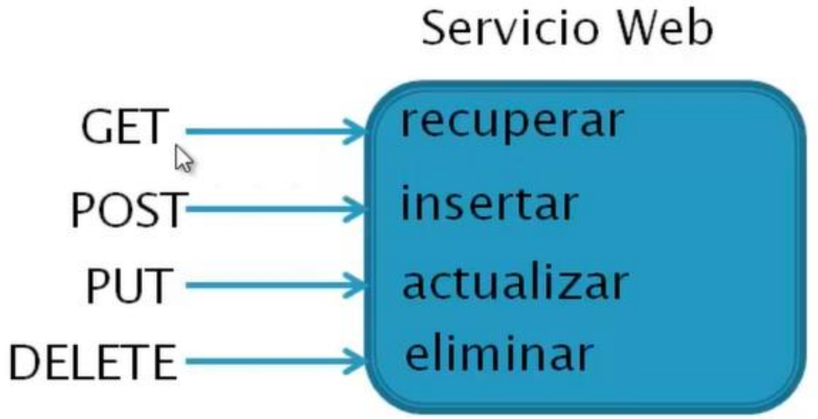
```
@PutMapping( value=..  consumes=MediaType.APPLICATION_JSON_VALUE)
public void metodoPut(@RequestBody Persona p){
        //operación de actualización del objeto Persona
}
```

Creación de un servicio completo

Mag. Carlos Adolfo Beltrán Castro

Operaciones CRUD

- El servicio ofrecerá la posibilidad de Crear, Recuperar, Actualizar y Eliminar objetos Curso a través de los métodos HTTP POST, GET, PUT y DELETE



- Cambios en el Controlador:

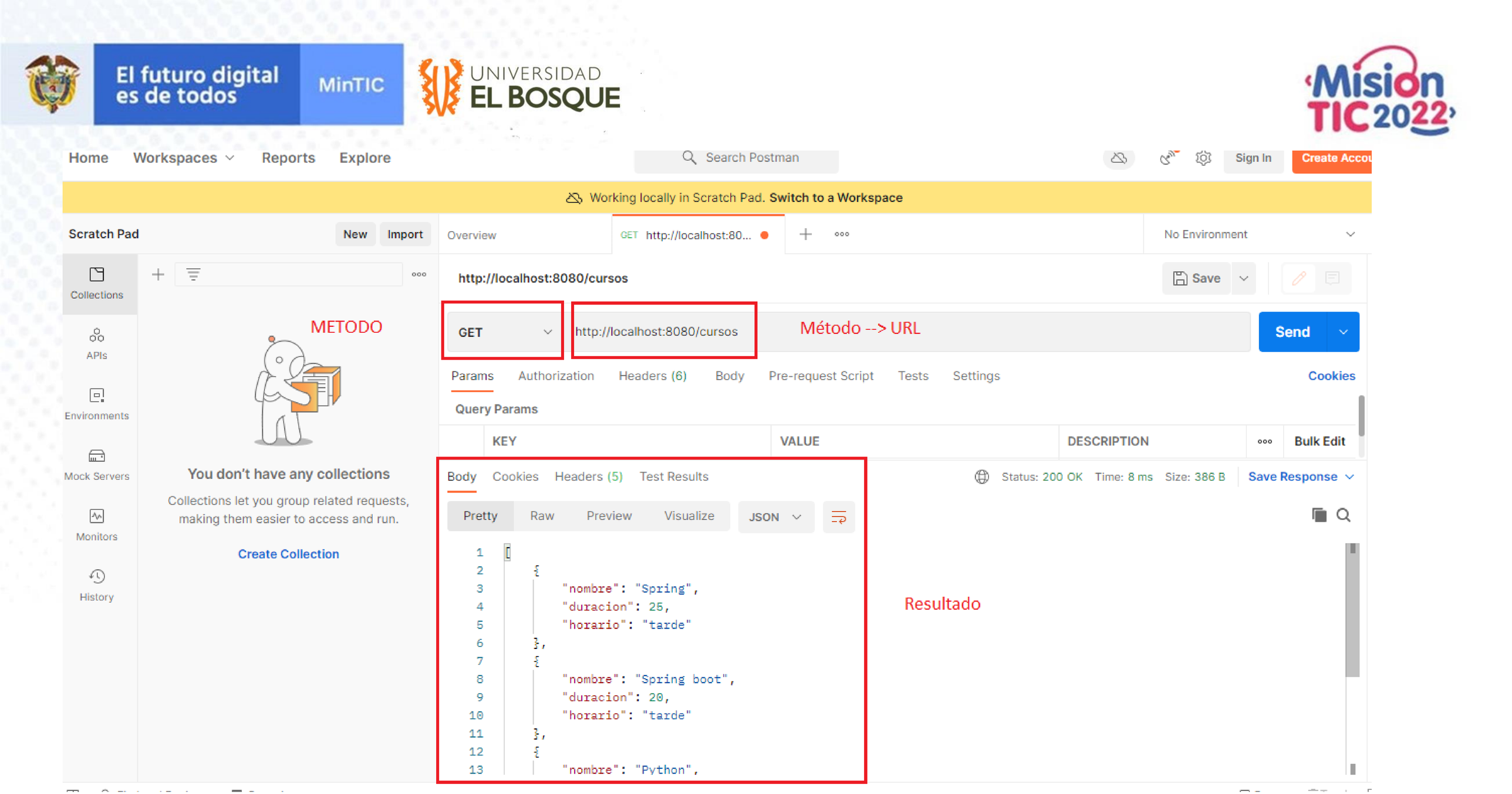
```
51      // Método para eliminar
52      @DeleteMapping(value="curso/{name}")
53      public void eliminarCurso(@PathVariable("name") String nombre) {
54          //elimina de la colección los elementos
55          //que cumplen la condición
56          cursos.removeIf(c->c.getNombre().equals(nombre));
57      }
58
59      // Agregar un curso
60      @PostMapping(value="curso",consumes=MediaType.APPLICATION_JSON_VALUE,produces=MediaType.APPLICATION_JSON_VALUE )
61      public List<Curso> altaCurso(@RequestBody Curso curso){
62          cursos.add(curso);
63          return cursos;
64      }
65
66      // Modificar un curso
67      @PutMapping(value="curso",consumes=MediaType.APPLICATION_JSON_VALUE,produces=MediaType.APPLICATION_JSON_VALUE )
68      public List<Curso> actualizaCurso(@RequestBody Curso curso){
69          //recorre los cursos y sustituye aquel que
70          //coincida con el nombre
71          for(int i=0;i<cursos.size();i++) {
72              if(cursos.get(i).getNombre().equals(curso.getNombre())) {
73                  cursos.set(i, curso);
74              }
75          }
76          return cursos;
77      }
```

Utilización de Postman

- Herramienta para lanzar peticiones HTTP y visualizar resultados
- Podemos descargar Postman desde la dirección:

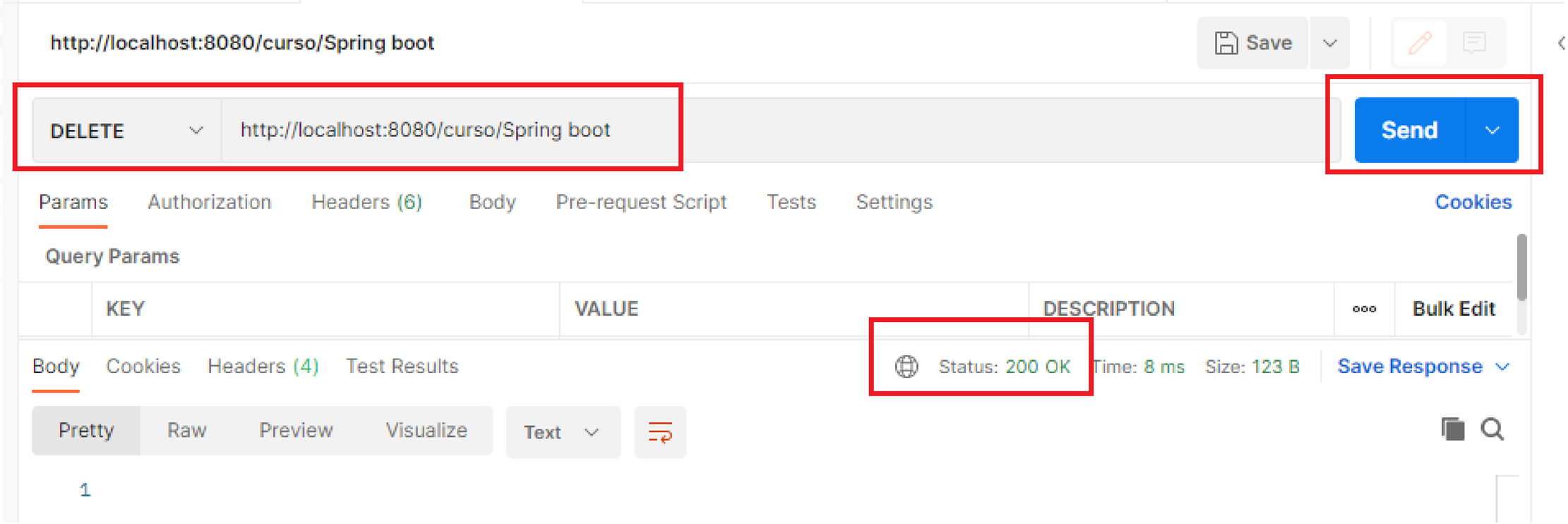
https://www.getpostman.com/downloads/

Mag. Carlos Adolfo Beltrán Castro

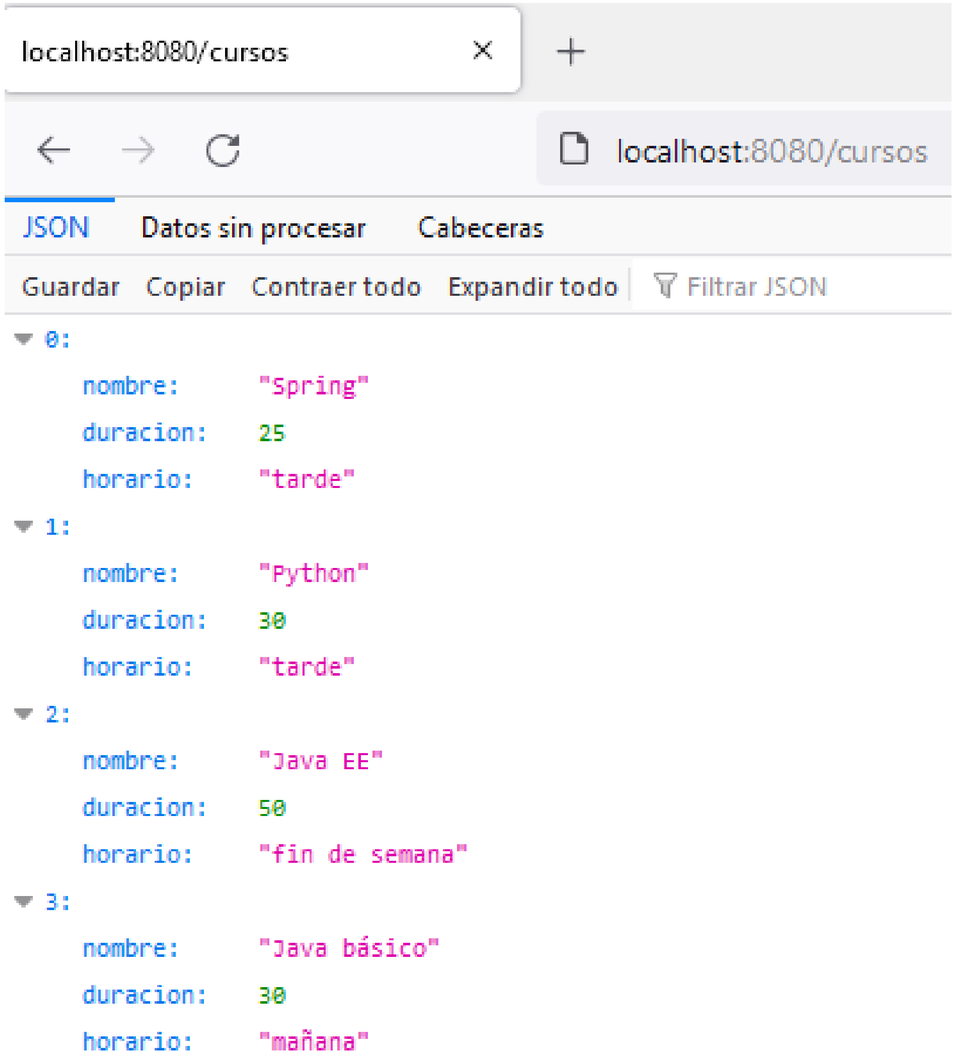


Mag. Carlos Adolfo Beltrán Castro

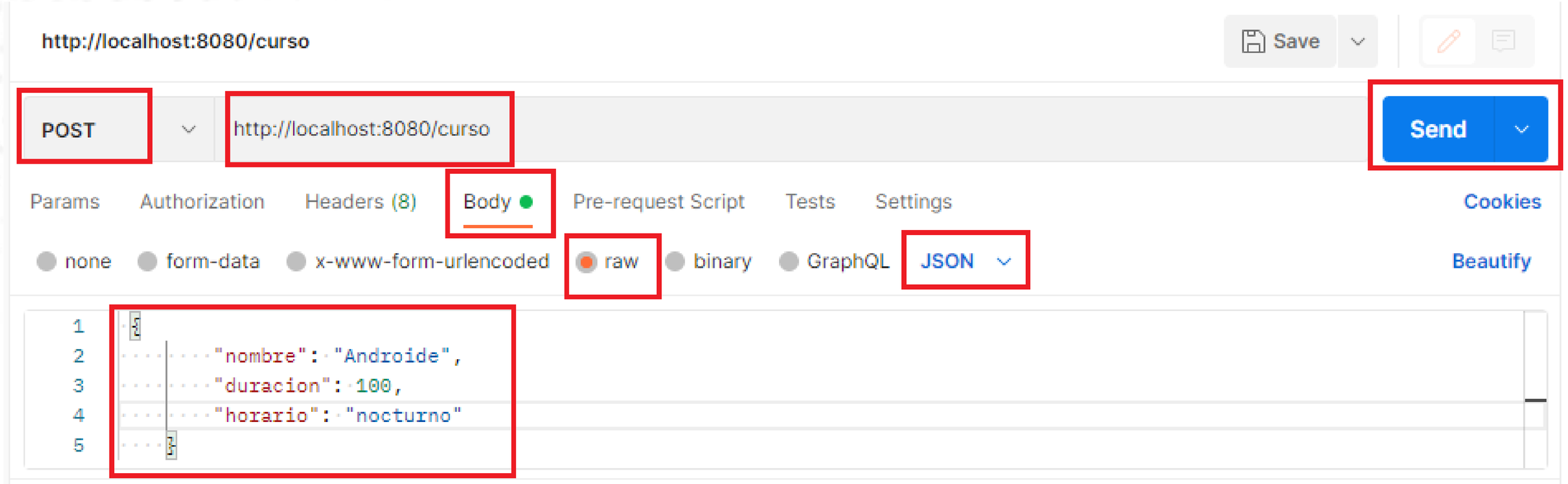
- Probando Eliminar un Objeto:



- Probando el resultado de eliminar:



- Probando el método post agregar:



- Revisando el resultado del método post agregar:

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
19          "duracion": 30,
20          "horario": "mañana"
21      },
22      {
23          "nombre": "Androide",
24          "duracion": 100,
25          "horario": "nocturno"
26      }
27  ]
```

- Probando el método put modificar: