



El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE



# Recurso 6

## Servicios REST





# 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

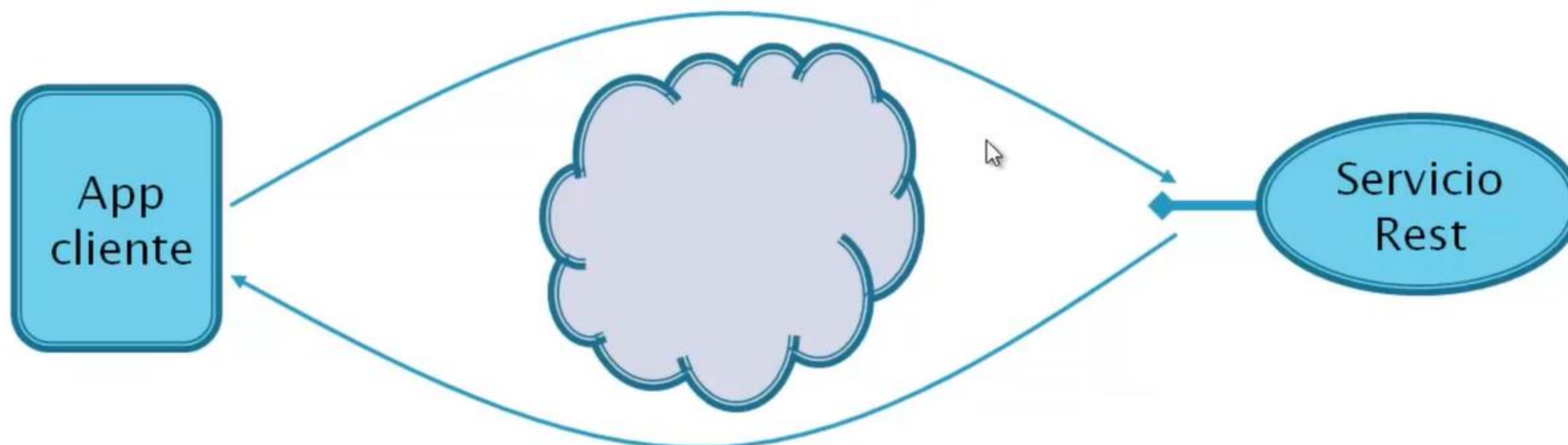




# Formato de intercambio de datos

Petición HTTP (GET,POST,PUT,DELETE)

Text, XML, JSON,..



Respuesta HTTP

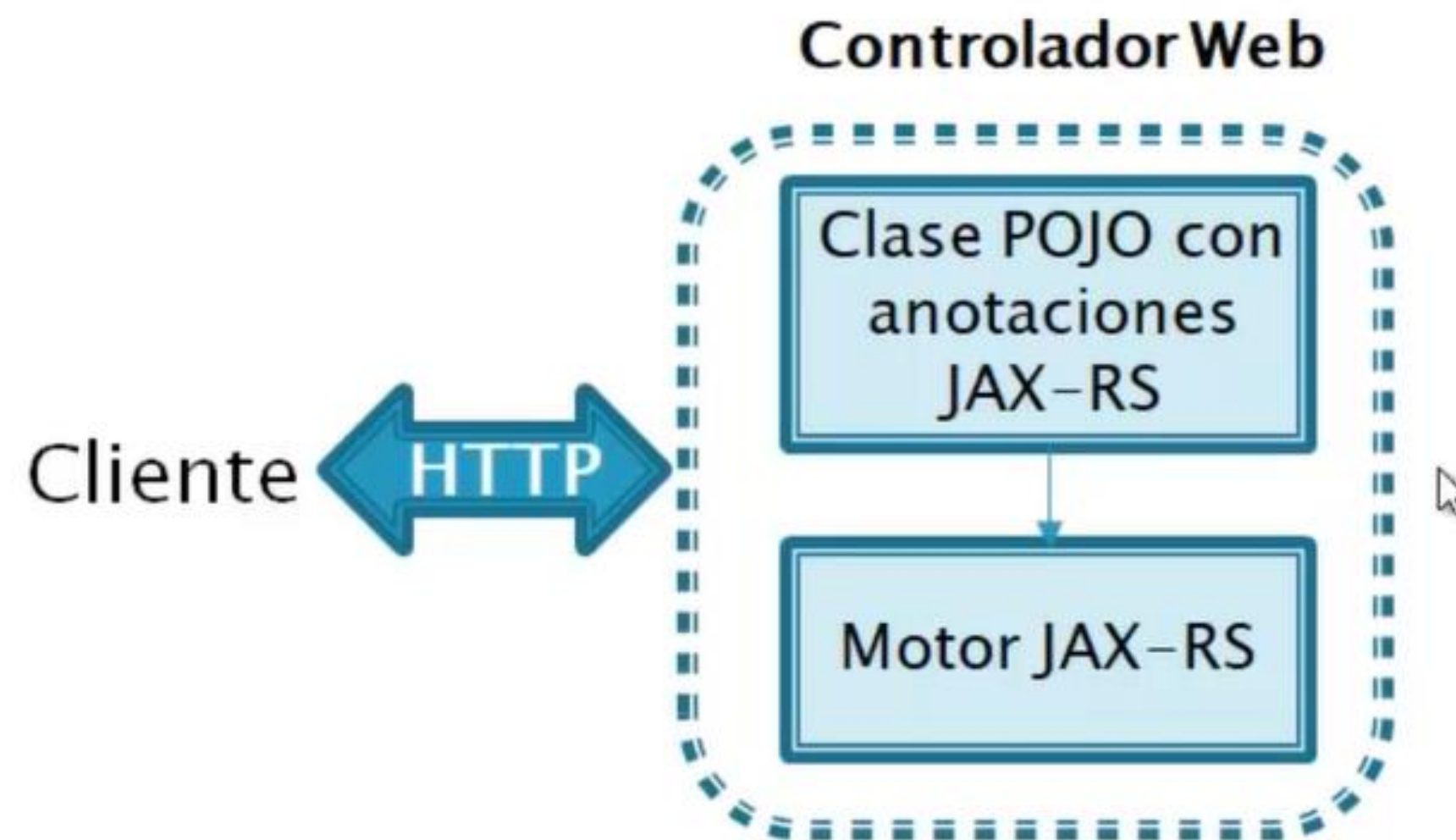
Text, XML, JSON,..





# 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

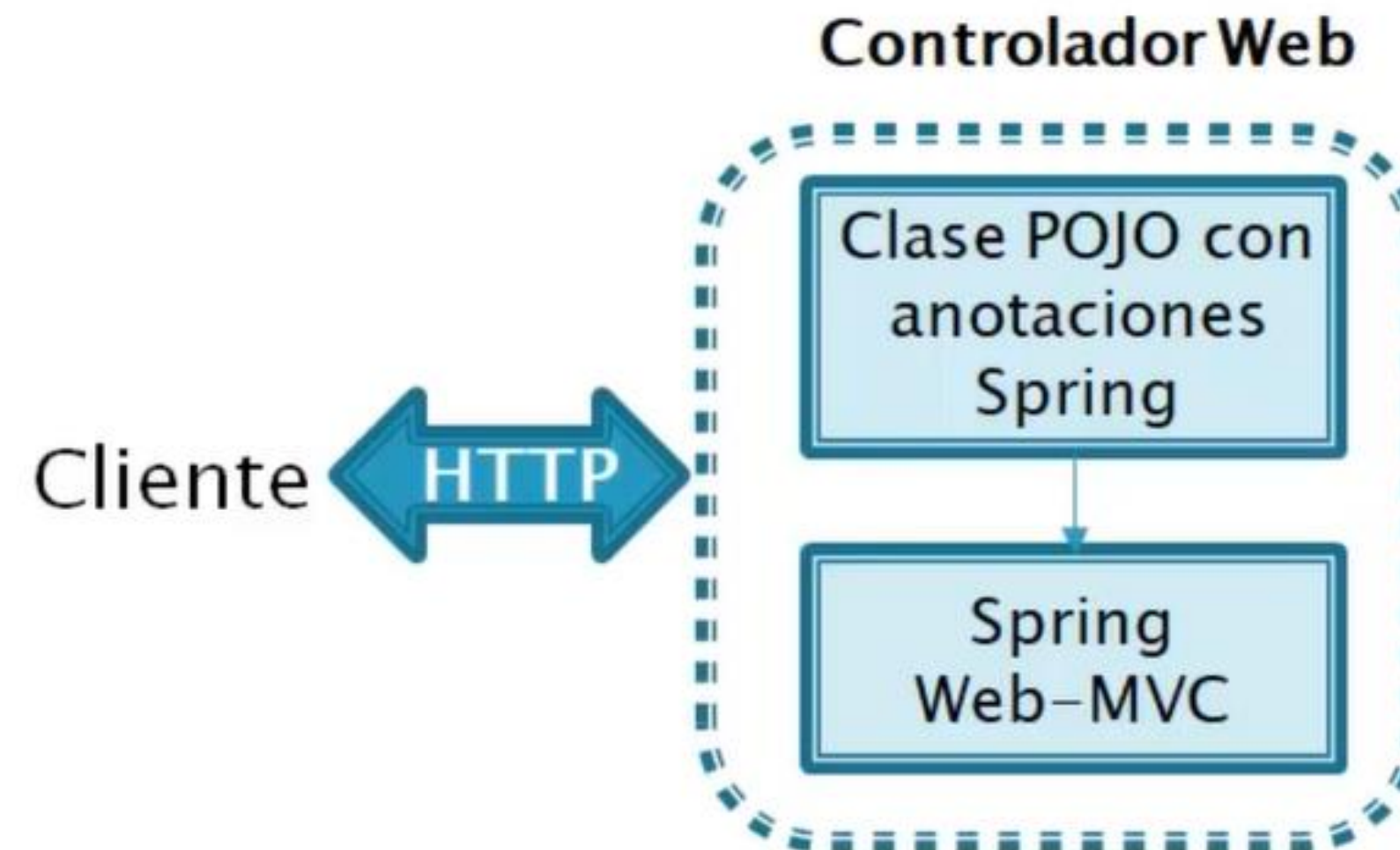






# Servicios REST con Spring

- 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.

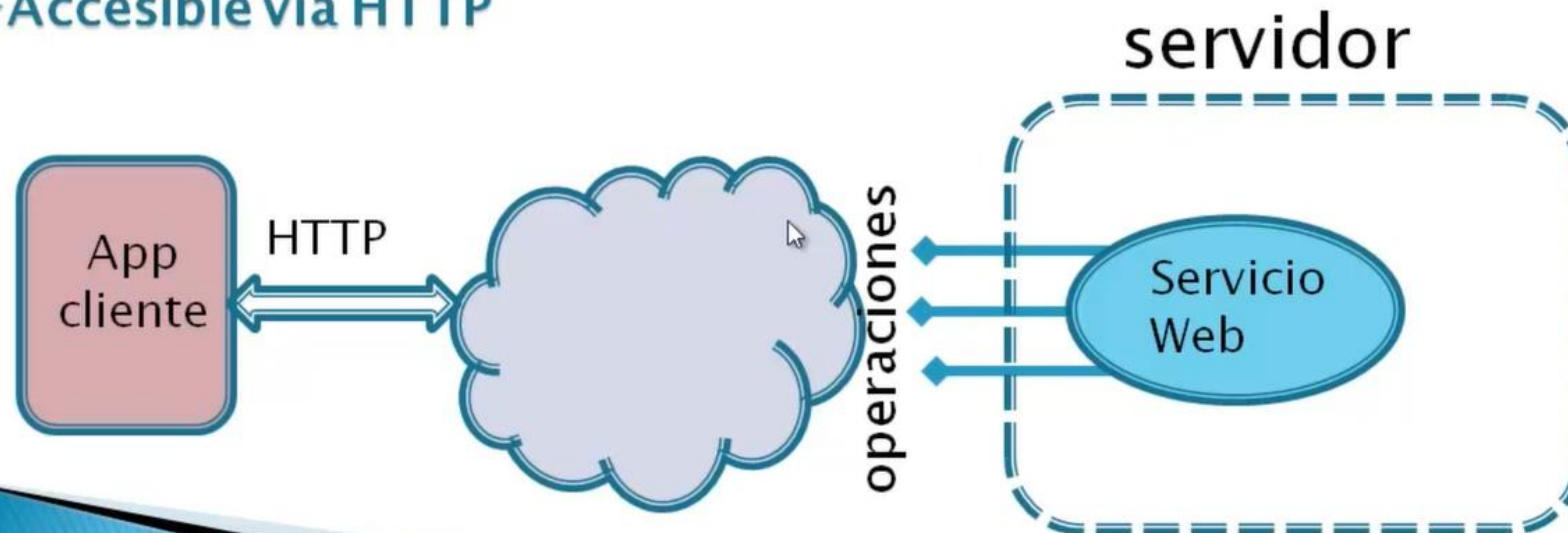






# ¿Qué es un servicio Web?

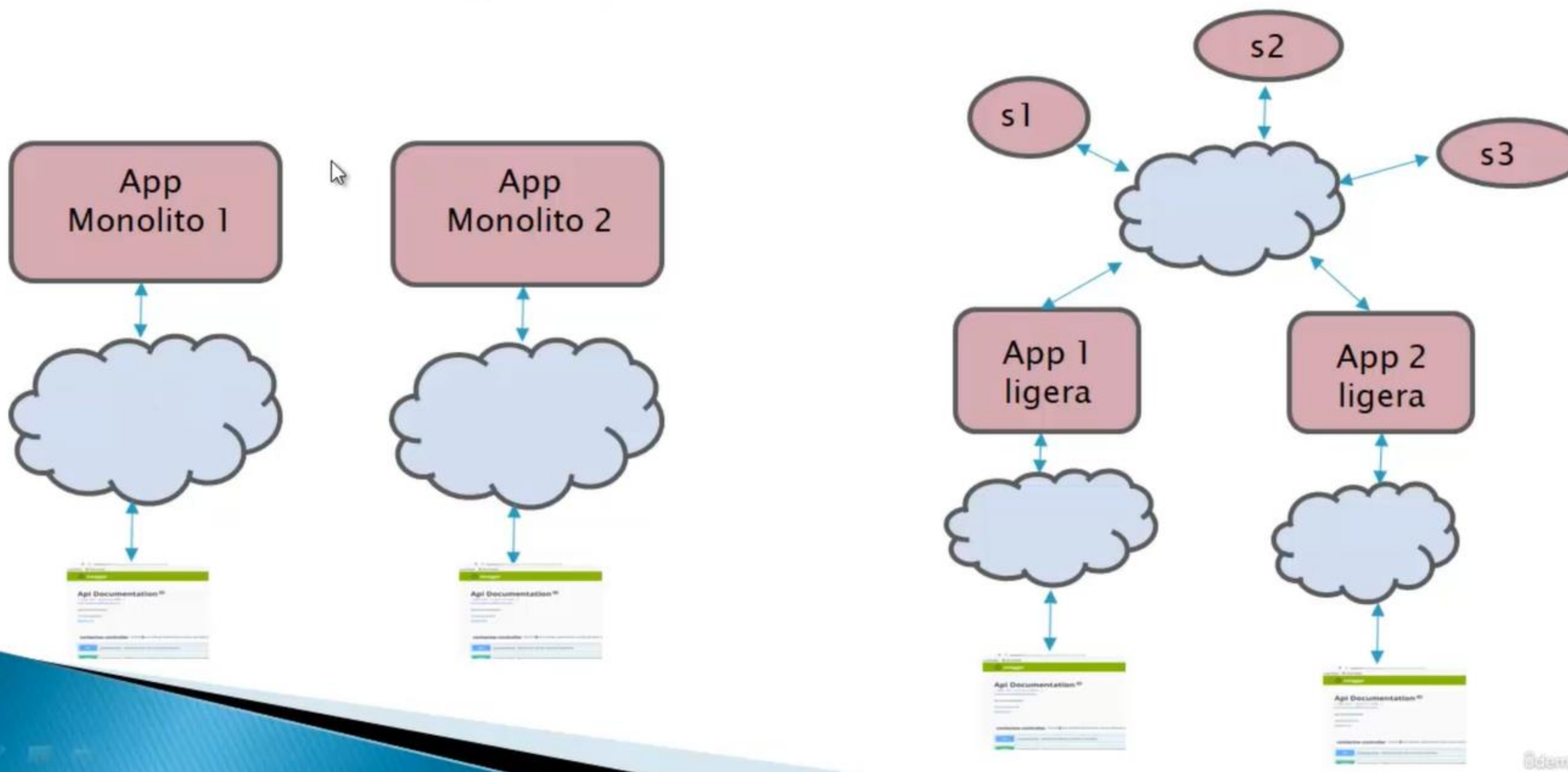
- Componente que expone operaciones a otros programas a través de la Web
- Accesible vía HTTP







# Servicios y aplicaciones clásicas







El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE



# Ventajas servicios Web

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



Odemy





# 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.



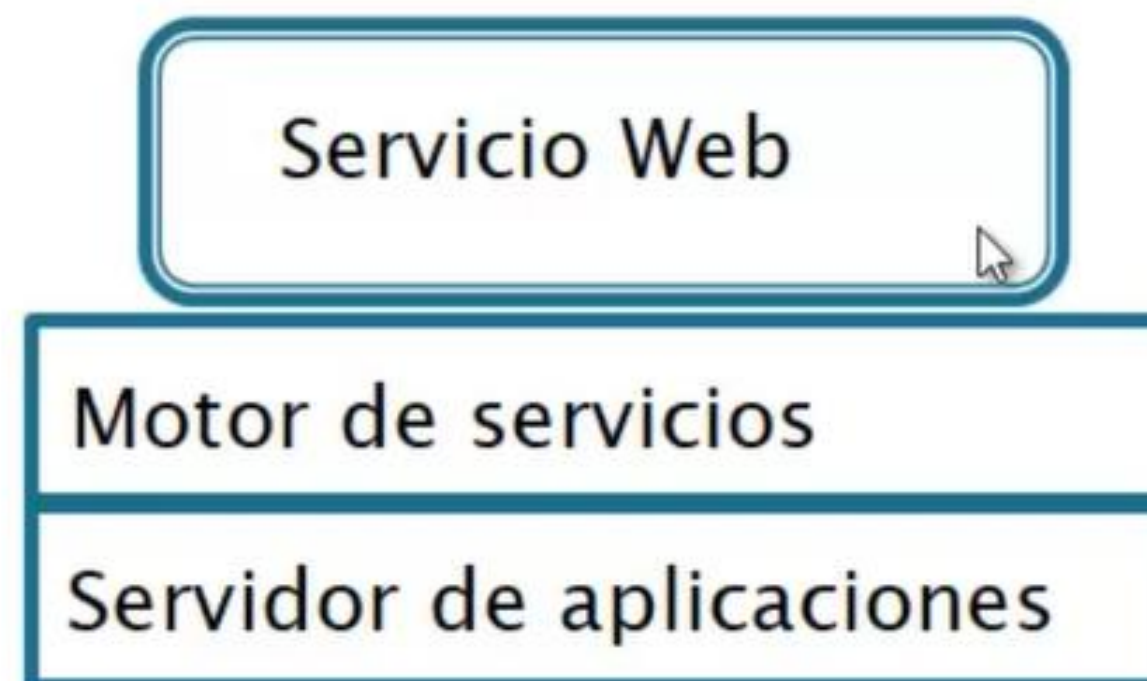




# Entorno de ejecución

➤ 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





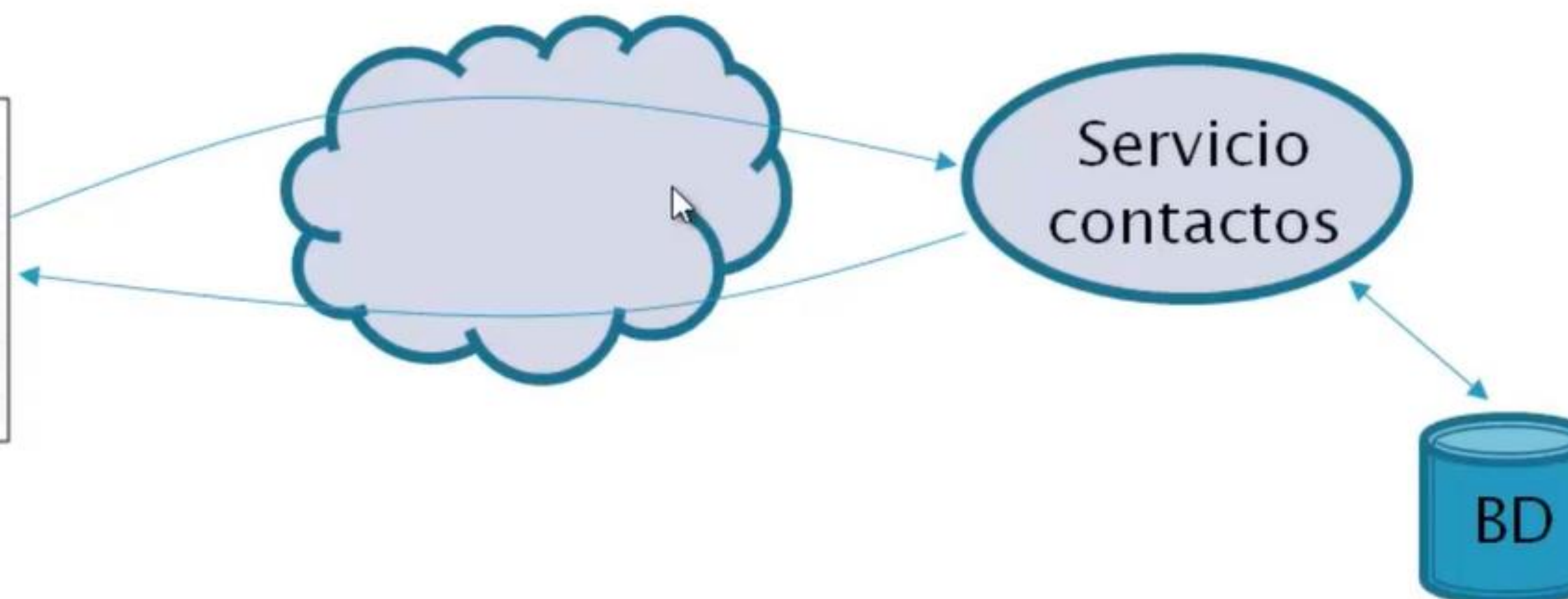


# Descripción

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

## Cliente JavaScript

Contactos			
Nombre	Email	Edad	Genero
Andrés Mario Soto	asoto	33	hombre
Juan Carlos Pérez	jperez@gmail	36	hombre
Julián	juand@gmail	33	hombre
William	william	33	hombre
Andrés	andres@gmail	33	hombre
Diego	diego@gmail	33	hombre
Andrés	andres@gmail	33	hombre
Diego	diego@gmail	33	hombre
Diego	diego@gmail	33	hombre
Diego	diego@gmail	33	hombre







# ¿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

Implementación  
servicio



Motor servicios

Servidor  
aplicaciones

App microservicio

Implementación  
+  
entorno de  
ejecución





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE



# Ventajas uso de microservicios

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

@clany





# 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

Spring Tradicional

.war

Código  
+  
Archivo XML de  
configuración

despliegue

Servidor  
aplicaciones

Spring Boot

.jar

Código  
+  
Entorno de  
ejecución





# 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

Mision  
TIC 2022

# Estructura de un microservicio con Spring Boot

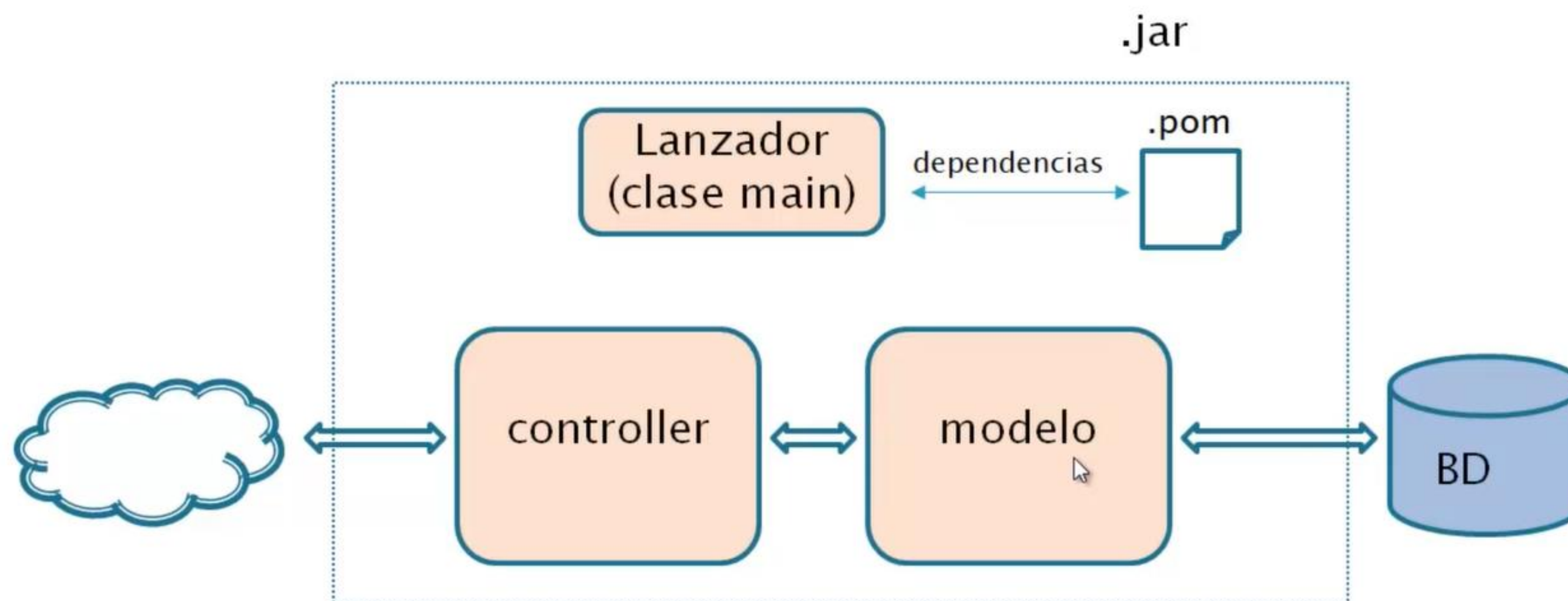


Udemy





# Estructura de aplicación







# 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";
    }
}
```





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
**EL BOSQUE**

Mision  
TIC 2022

`http://localhost:8080/saludo`

Microservicio Spring boot

VER EJEMPLO



Odemy





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
**EL BOSQUE**

Mision  
TIC 2022

`http://localhost:8080/saludo`

Microservicio Spring boot

VER EJEMPLO



Odemy





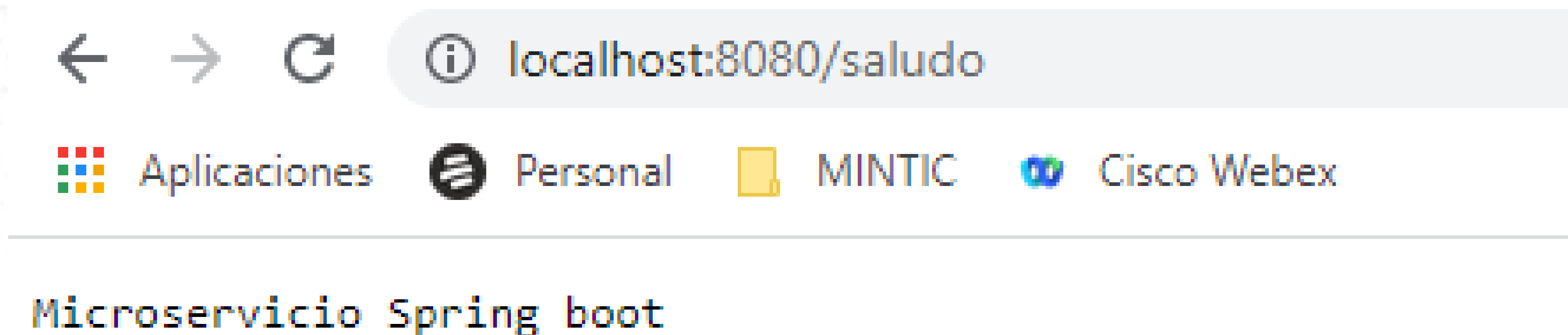
- Crear Proyecto Spring Boot con solo dependencia web
- Agregar controlador con el siguiente contenido:

```
1 package com.boot.ejemplo;
2
3 import org.springframework.http.MediaType;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PathVariable;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController
9 public class SaludoController {
10     @GetMapping(value = "saludo", produces=MediaType.TEXT_PLAIN_VALUE)
11     public String saludo() {
12         return "Microservicio Spring boot";
13     }
14 }
15
```





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







El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC 2022

# Propiedades de configuración



Udemy





# 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>

The screenshot shows the 'Common Application properties' section of the Spring Boot documentation. It includes a table of core properties and two informational boxes.

**Common Application properties**

Various properties can be specified inside your `application.properties` file, inside your `application.yml` file, or as command line switches. This appendix provides a list of references to the underlying classes that consume them.

**1. Core properties**

Key	Default Value	Description
<code>debug</code>	<code>false</code>	Enable debug logs.
<code>info.*</code>		Arbitrary properties to add to the info endpoint.
<code>logging.config</code>		Location of the logging configuration file. For instance, <code>classpath:logging.config</code> .

**Informational boxes:**

- Lightbulb icon:** Spring Boot provides various conversion mechanism with advanced value formatting. make sure to review [the properties conversion section](#).
- Information icon:** Property contributions can come from additional jar files on your classpath, so you should not consider this an exhaustive list. Also, you can define your own p...





# 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`.





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC 2022

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

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].[/my
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
```

Boot Dashboard

Type tags, projects, or working set names to match (incl. \* and ? wildca)

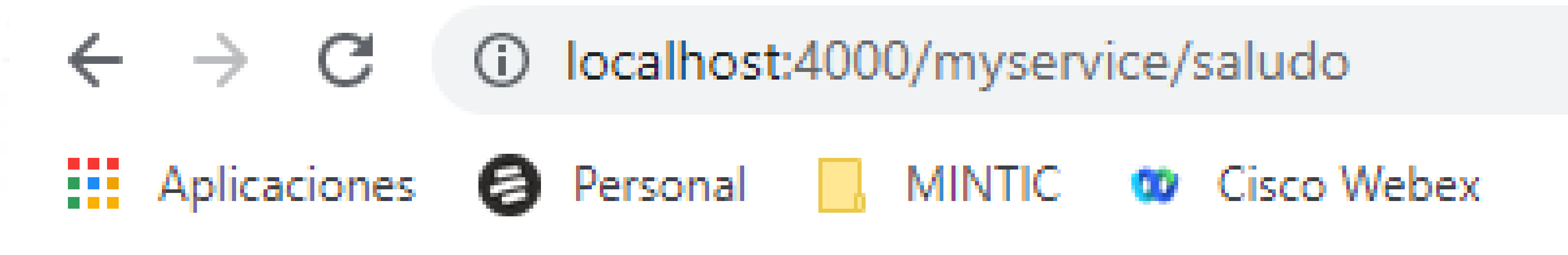
> local

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:



Microservicio Spring boot





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC 2022

# Envío de datos



Udemy





# Variables en URL

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

Context path      Dirección servicio      variables

`http://servidor:8080/app/servicio/p1/p2`





# 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 ?

Parámetros

`http://servidor:8080/app/rest/servicio?x=valor1&y=valor2`





# 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 }
```





El futuro digital  
es de todos

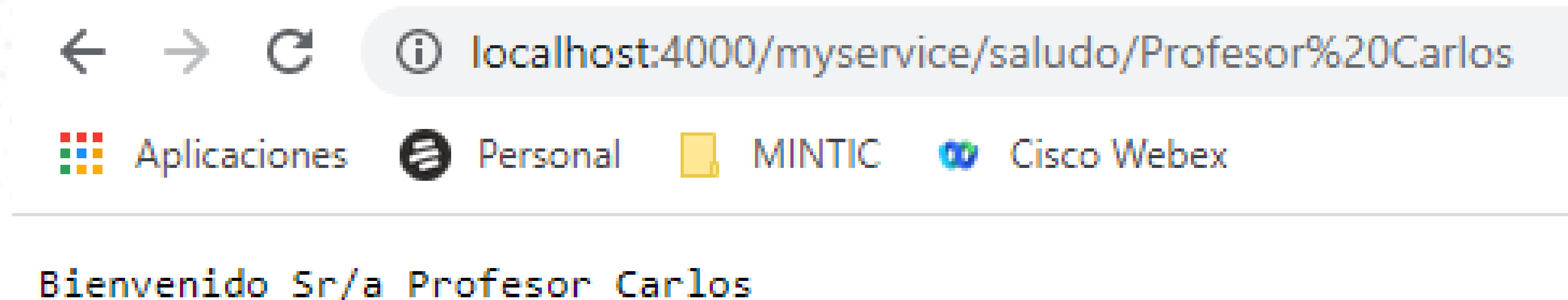
MinTIC



UNIVERSIDAD  
EL BOSQUE



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







El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC 2022

# Mapeado de objetos



Udemy

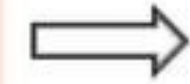




# Transformación a JSON

➤ 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 transformar 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

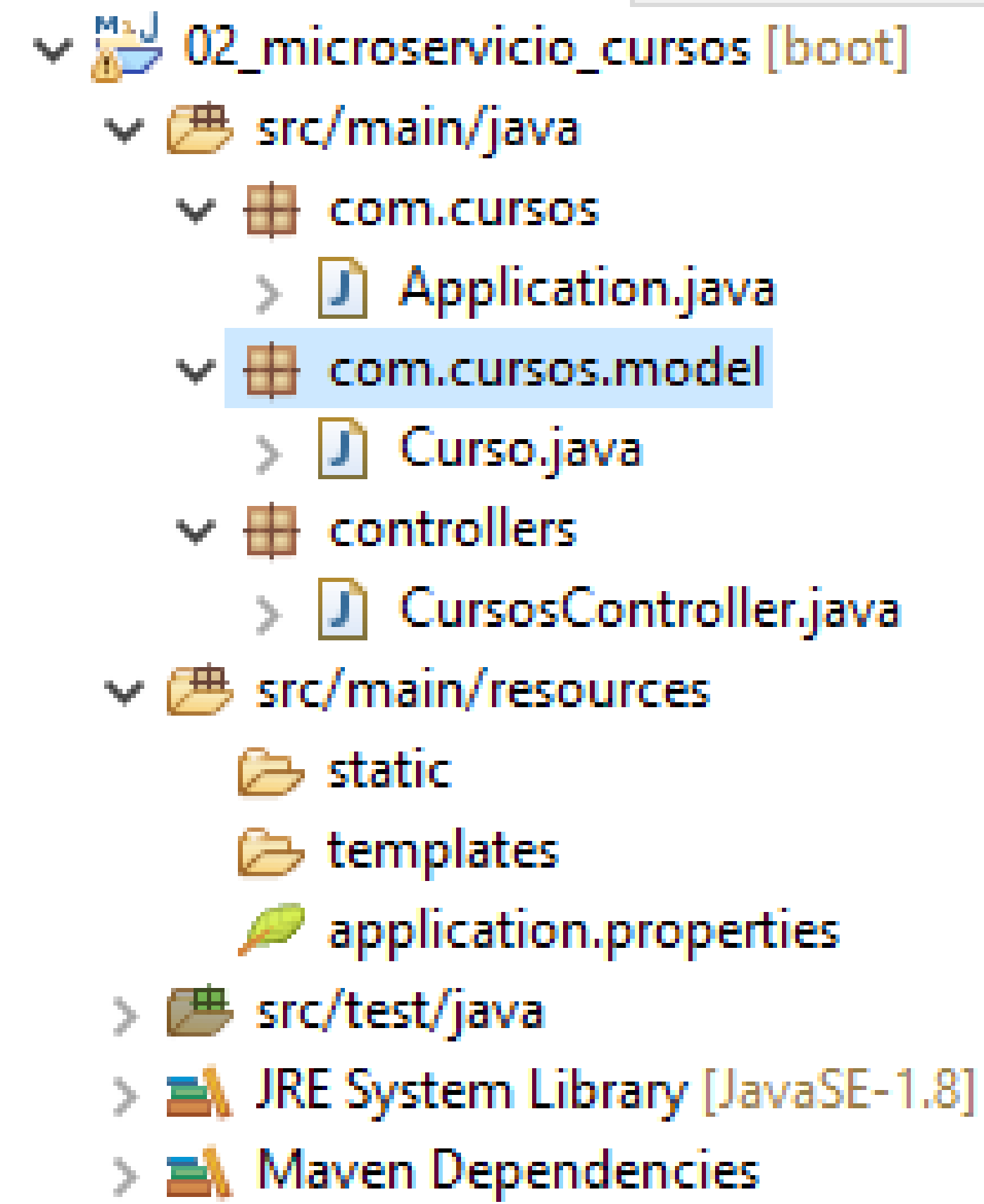
- Expondrá tres recursos para búsqueda de cursos
  - Devuelve un curso (nombre, duración y horario) cualquiera
  - Devuelve la lista de cursos existentes
  - Devuelve los cursos de un determinado nombre







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







- Clase Curso

```
1 package com.cursos.model;
2
3 public class Curso {
4
5     // Atributos
6     private String nombre;
7     private int duracion;
8     private String horario;
9
10    // Constructores
11    public Curso(String nombre, int duracion, String horario) {
12        super();
13        this.nombre = nombre;
14        this.duracion = duracion;
15        this.horario = horario;
16    }
17    public Curso() {
18    }
19
20
21    // Setter y Getter
22    public String getNombre() {
23        return nombre;
24    }
25    public void setNombre(String nombre) {
26        this.nombre = nombre;
27    }
```

```
28    public int getDuracion() {
29        return duracion;
30    }
31    public void setDuracion(int duracion) {
32        this.duracion = duracion;
33    }
34    public String getHorario() {
35        return horario;
36    }
37    public void setHorario(String horario) {
38        this.horario = horario;
39    }
40
41 }
```





## • Controlador

```
1 package controllers;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.annotation.PostConstruct;
7
8 import org.springframework.http.MediaType;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.RestController;
12
13 import com.cursos.modelCurso;
14
15 @RestController
16 public class CursosController {
17
18     // Lista de todos los cursos
19     private List<Curso> cursos;
20
21     @PostConstruct
22     public void init() {
23         cursos=new ArrayList<>();
24         cursos.add(new Curso("Spring",25,"tarde"));
25         cursos.add(new Curso("Spring boot",20,"tarde"));
26         cursos.add(new Curso("Python",30,"tarde"));
27         cursos.add(new Curso("Java EE",50,"fin de semana"));
28         cursos.add(new Curso("Java básico",30,"mañana"));
29     }
```

```
30 @GetMapping(value="cursos",produces=MediaType.APPLICATION_JSON_VALUE)
31 public List<Curso> getCursos(){
32     return cursos;
33 }
34
35 // Listar el Curso Java
36 @GetMapping(value="curso",produces=MediaType.APPLICATION_JSON_VALUE)
37 public Curso getCurso() {
38     return new Curso("Java",100,"Mañana");
39 }
40
41 // Mostrar los datos de cierto curso
42 @GetMapping(value="cursos/{name}",produces=MediaType.APPLICATION_JSON_VALUE)
43 public List<Curso> buscarCursos(@PathVariable("name") String nombre){
44     List<Curso> aux=new ArrayList<>();
45     for(Curso c:cursos) {
46         if(c.getNombre().contains(nombre)) {
47             aux.add(c);
48         }
49     }
50     return aux;
51 }
52 }
```





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

```
1 package com.cursos;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.annotation.ComponentScan;
6
7 @SpringBootApplication
8 @ComponentScan(basePackages = {"controllers"})
9 public class Application {
10
11     public static void main(String[] args) {
12         SpringApplication.run(Application.class, args);
13     }
14
15 }
```





El futuro digital  
es de todos

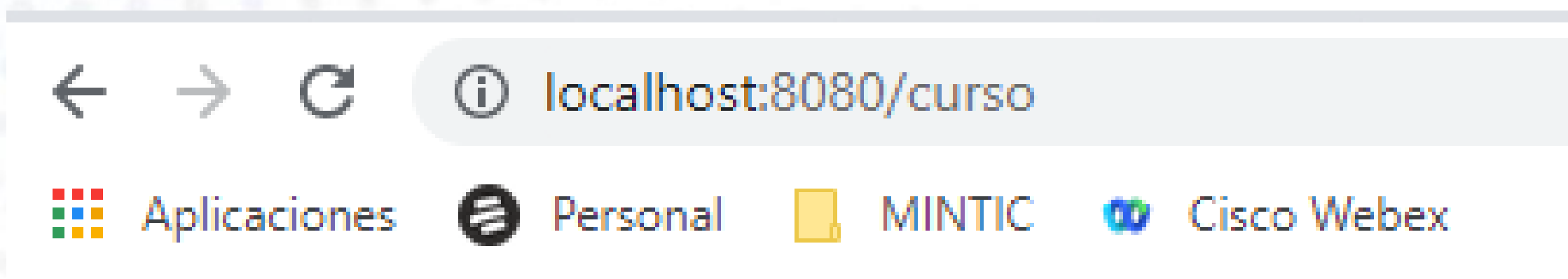
MinTIC



UNIVERSIDAD  
EL BOSQUE

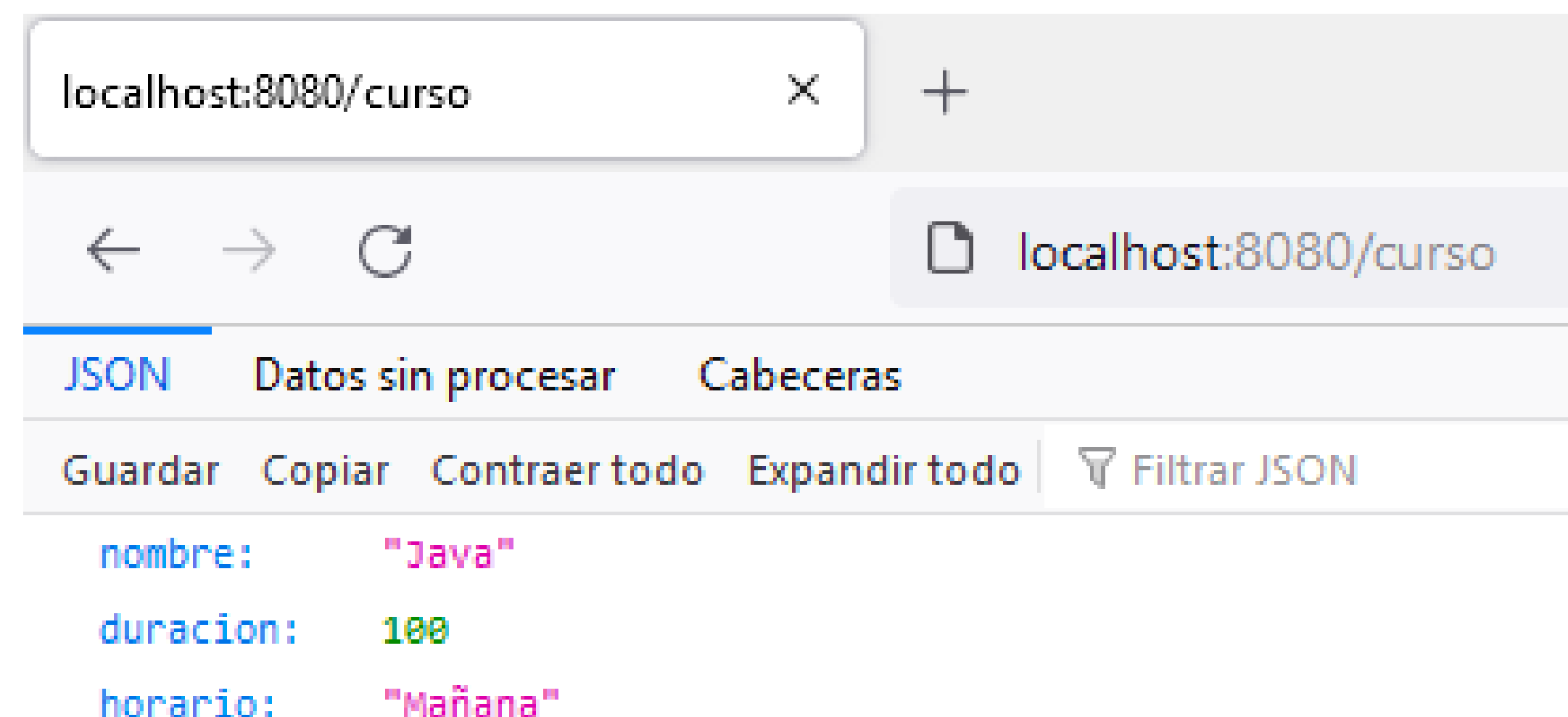


- Probar el proyecto:



```
{"nombre":"Java","duracion":100,"horario":"Mañana"}
```

**Google Chrome**



**Mozilla Firefox**





El futuro digital  
es de todos

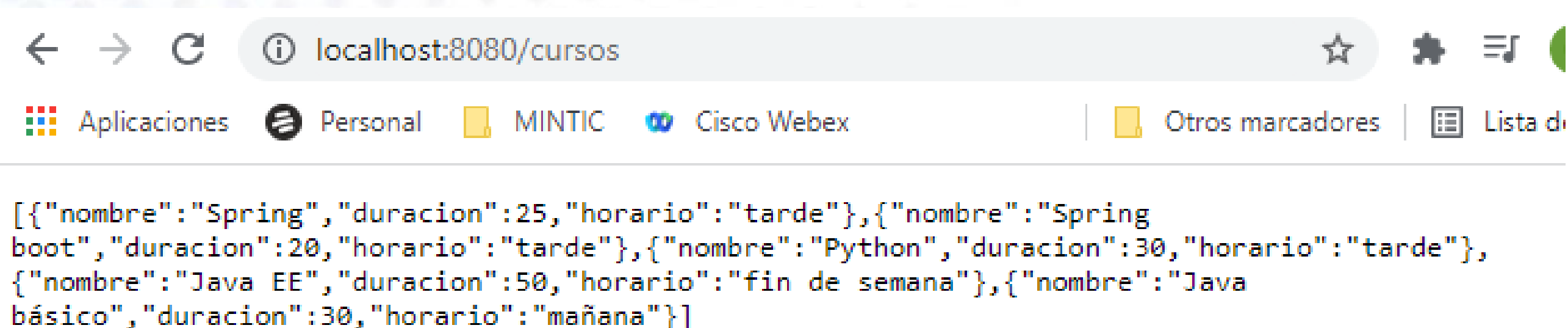
MinTIC



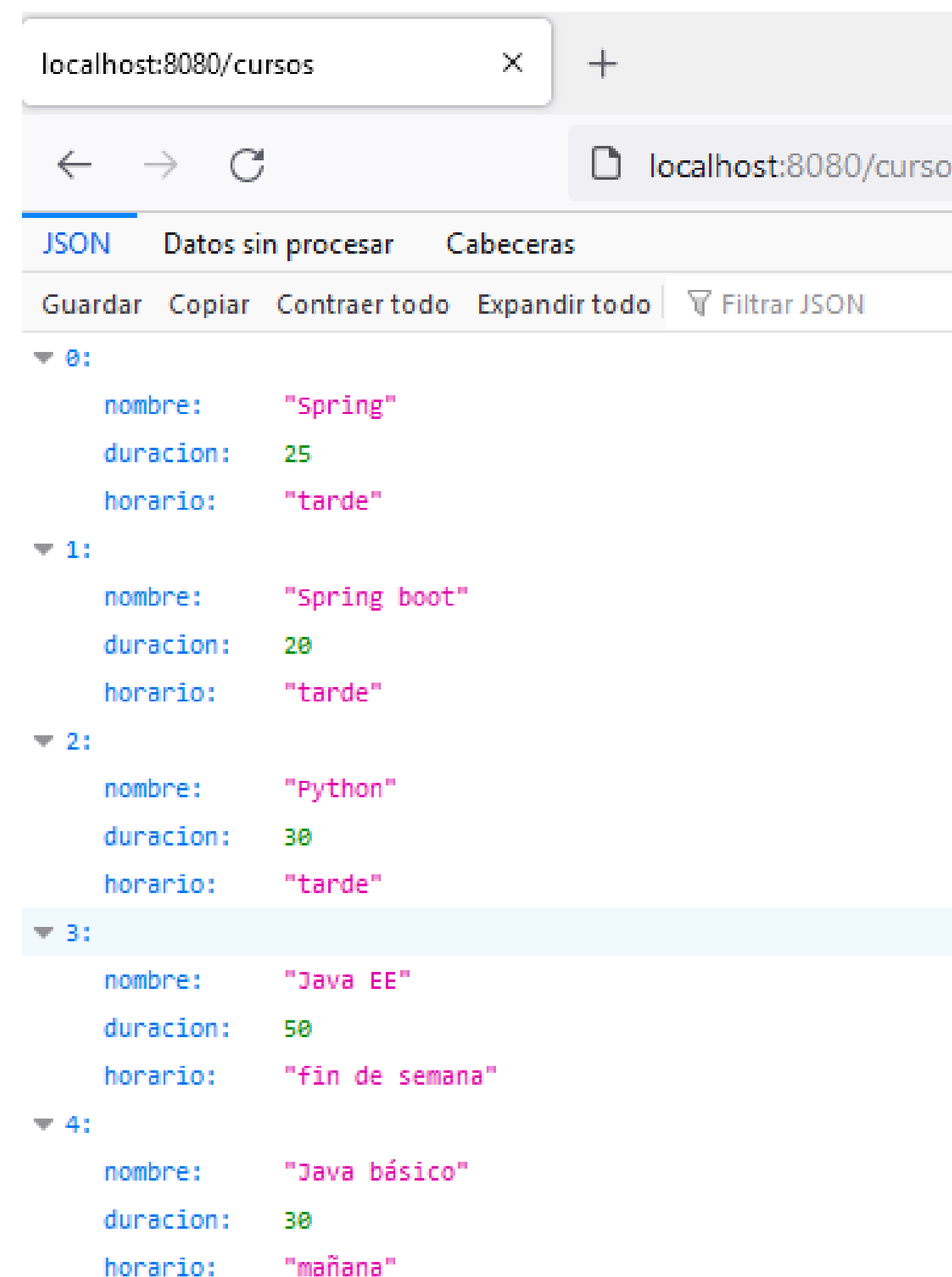
UNIVERSIDAD  
EL BOSQUE



- Probar el proyecto:



**Google Chrome**



**Mozilla Firefox**





El futuro digital  
es de todos

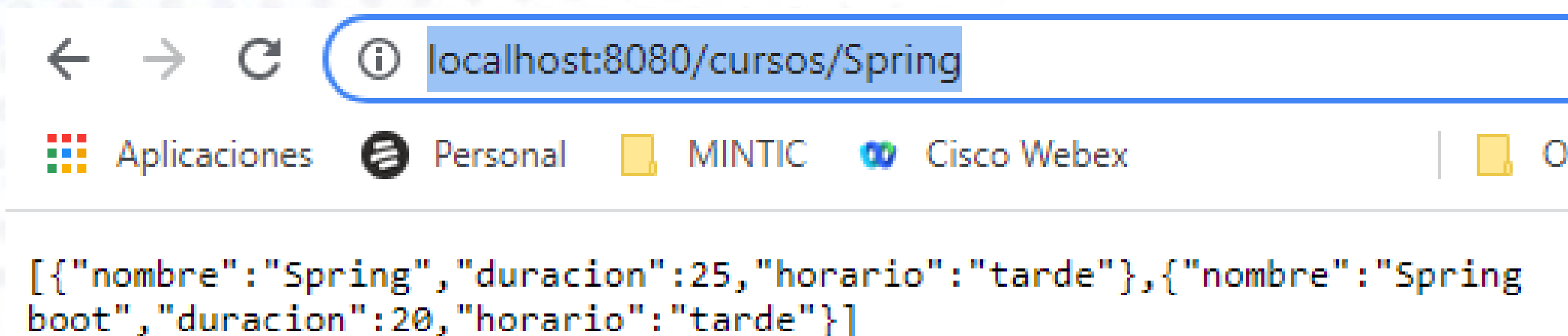
MinTIC



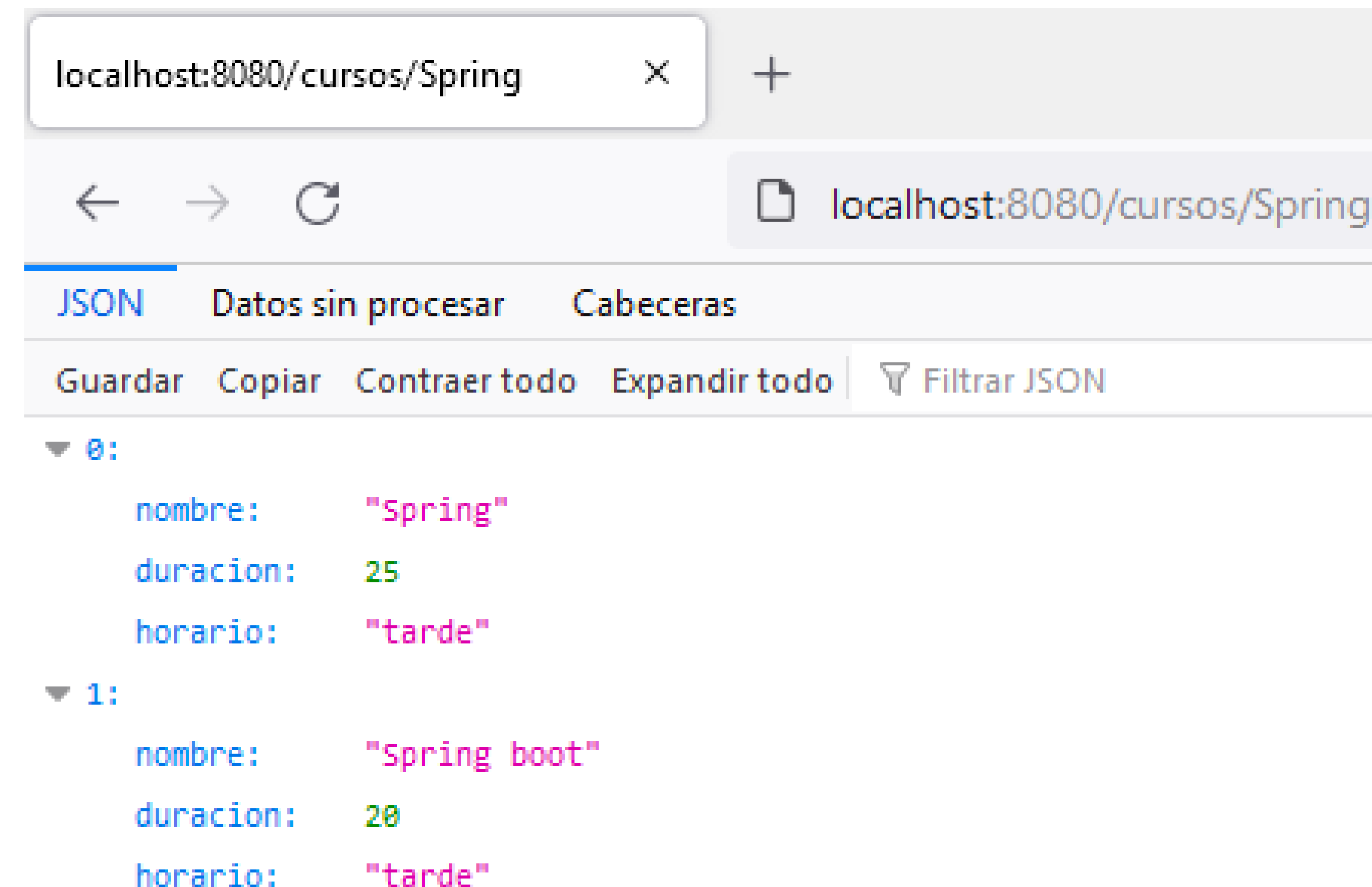
UNIVERSIDAD  
EL BOSQUE



- Probar el proyecto:



**Google Chrome**



**Mozilla Firefox**





# Mapeado a XML

➤ La clase bean se anota con `@XmlElement`

```
@XmlElement  
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

Mision  
TIC 2022

UNIVERSIDAD EL BOSQUE

# Respuesta a diferentes métodos HTTP



Udemy





# 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  
}
```





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Misión  
TIC 2022

# Creación de un servicio completo



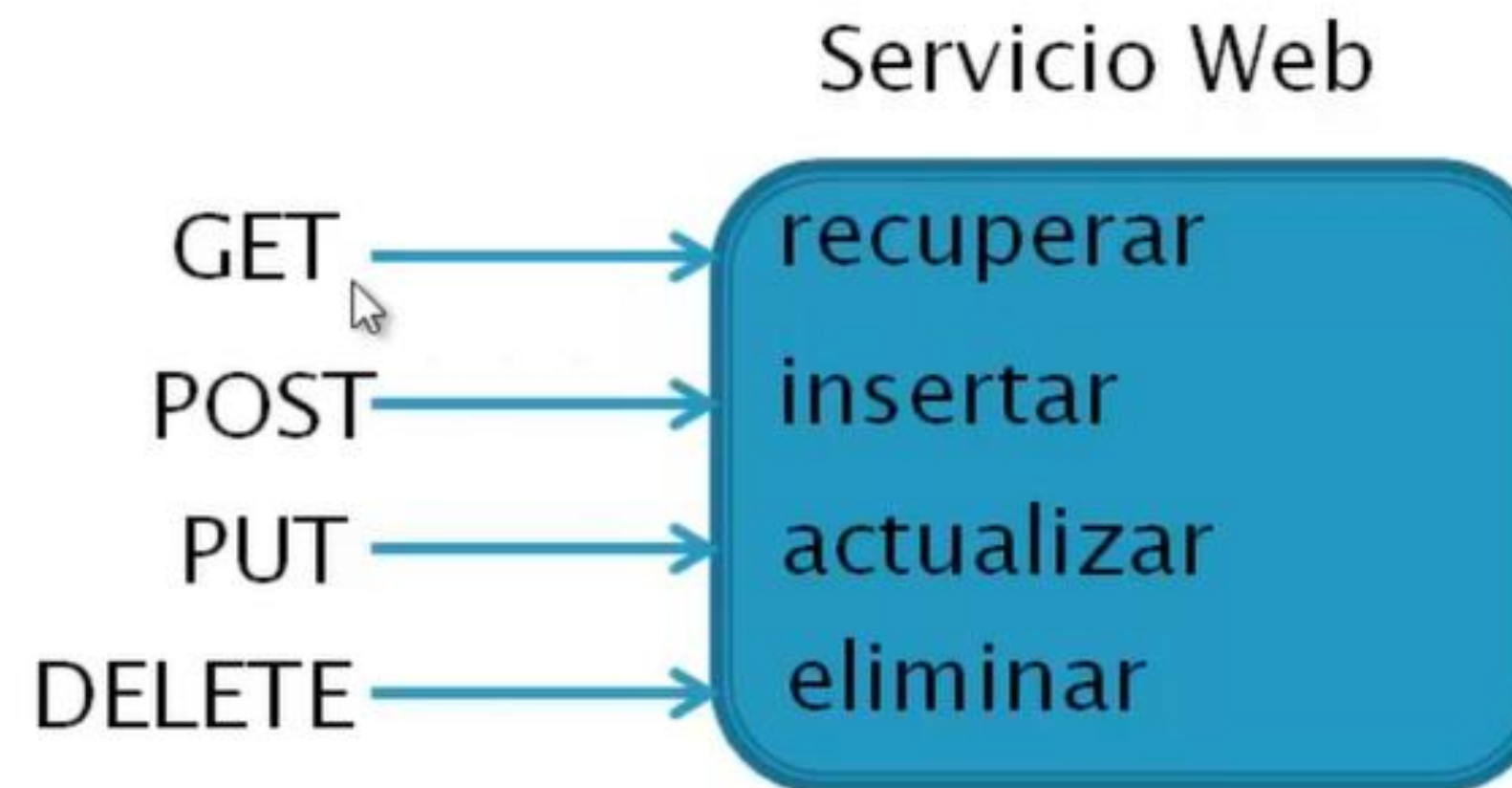
Udemy





# 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 }
```

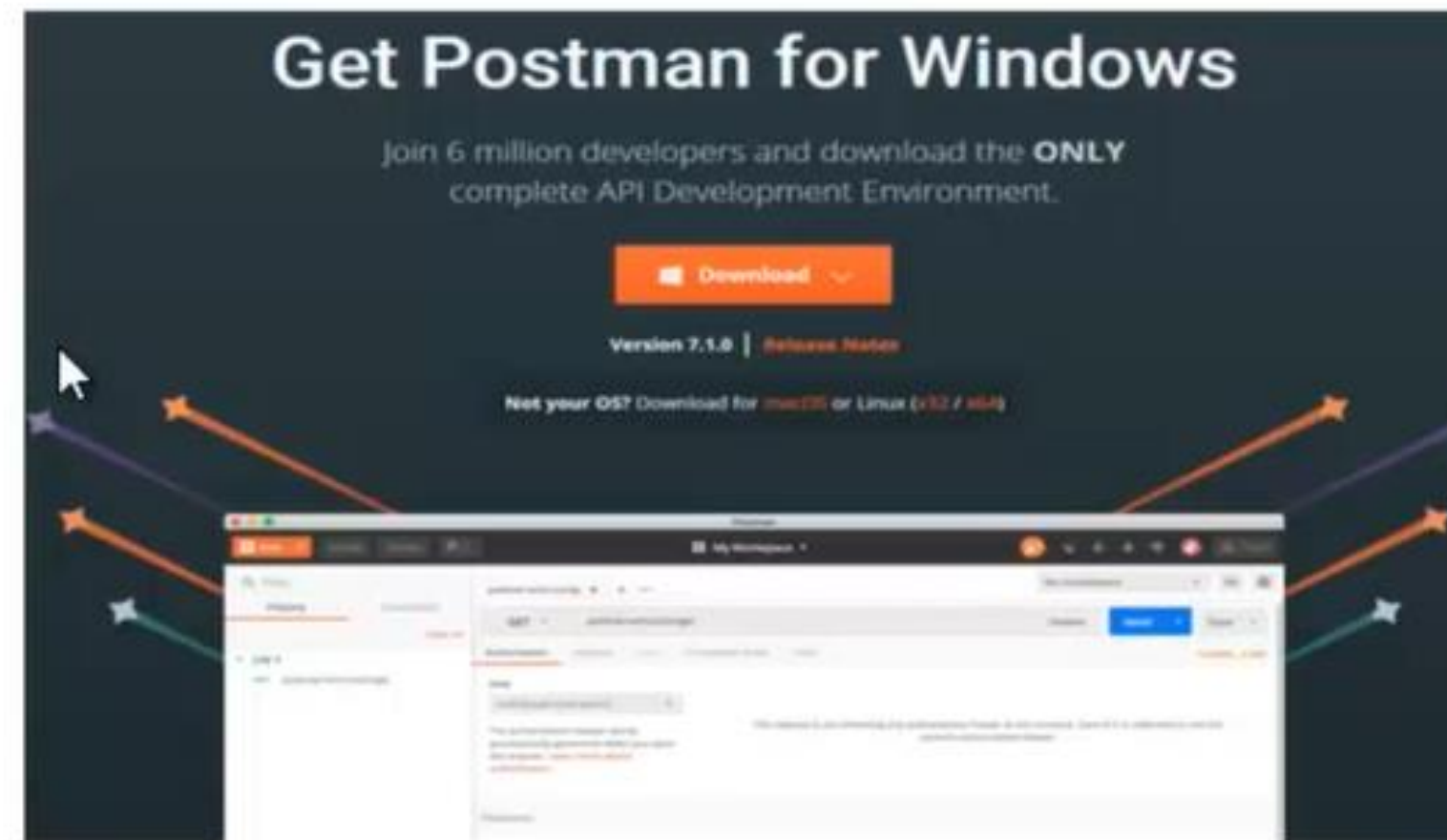




# Utilización de Postman

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

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







El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC2022

Home Workspaces Reports Explore

Search Postman



Sign In

Create Account

Working locally in Scratch Pad. Switch to a Workspace

Scratch Pad

New

Import

Overview

GET http://localhost:80...

+ ...

No Environment

Collections

APIs

Environments

Mock Servers

Monitors

History



METODO

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Create Collection

http://localhost:8080/cursos

Save

Send

GET

http://localhost:8080/cursos

Método --> URL

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

KEY

VALUE

DESCRIPTION

Bulk Edit

Body

Cookies

Headers (5)

Test Results



Status: 200 OK

Time: 8 ms

Size: 386 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

...

```
1 {
2   {
3     "nombre": "Spring",
4     "duracion": 25,
5     "horario": "tarde"
6   },
7   {
8     "nombre": "Spring boot",
9     "duracion": 20,
10    "horario": "tarde"
11  },
12  {
13    "nombre": "Python",
```

Resultado





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC 2022

- Probando Eliminar un Objeto:

http://localhost:8080/curso/Spring boot

Save

DELETE http://localhost:8080/curso/Spring boot

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
--	-----	-------	-------------	-----	-----------

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 8 ms Size: 123 B Save Response

Pretty Raw Preview Visualize Text

1





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

- Probando el resultado de eliminar:

localhost:8080/cursos

← → ↻

localhost:8080/cursos

JSON Datos sin procesar Cabeceras

Guardar Copiar Contraer todo Expandir todo Filtrar JSON

```
▼ 0:
  nombre: "Spring"
  duracion: 25
  horario: "tarde"
▼ 1:
  nombre: "Python"
  duracion: 30
  horario: "tarde"
▼ 2:
  nombre: "Java EE"
  duracion: 50
  horario: "fin de semana"
▼ 3:
  nombre: "Java básico"
  duracion: 30
  horario: "mañana"
```







El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE

Mision  
TIC2022

- Probando el método post agregar:

http://localhost:8080/curso

Save

POST

http://localhost:8080/curso

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nombre": "Androide",
3   "duracion": 100,
4   "horario": "nocturno"
5 }
```





- 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  ]
```





El futuro digital  
es de todos

MinTIC



UNIVERSIDAD  
EL BOSQUE



- Probando el método put modificar:

The screenshot shows the Chrome DevTools Network tab with an HTTP PUT request to `http://localhost:8080/curso`. The request is highlighted with a red box. The request body is a JSON object: `{ "nombre": "Spring", "duracion": 60, "horario": "mañana" }`, also highlighted with a red box. The response status is 200 OK, and the response body is a JSON object: `{ "nombre": "Spring", "duracion": 60, "horario": "mañana" }`, also highlighted with a red box.

PUT `http://localhost:8080/curso` Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nombre": "Spring",
3   "duracion": 60,
4   "horario": "mañana"
5 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 74 ms Size: 450 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "nombre": "Spring",
3   "duracion": 60,
4   "horario": "mañana"
5 },
6 }
```