



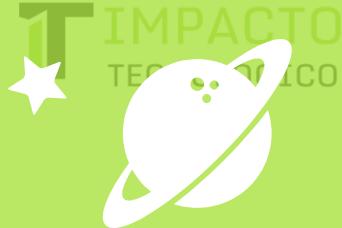


Spring Boot

Configuración

Entorno e Instalación





¿Qué es Spring Boot?



Spring Boot

Spring Boot es un sub-proyecto de Spring, que nos permite facilitar la creación de proyectos con el framework Spring eliminando la necesidad de crear largos archivos de configuración xml.

Spring Boot provee configuraciones por defecto para Spring y otra gran cantidad de librerías, además provee un modelo de programación parecido a las aplicaciones java tradicionales.

Spring Boot

Sus características principales son que provee out-of-the-box una serie de elementos que nos permiten desarrollar diferentes tipos de aplicaciones de forma casi inmediata.

Algunas de estas características son:

- ◆ Servidores de aplicaciones embebidos (Tomcat, Jetty, Undertow)
- ◆ POMs con dependencias y plug-ins para Maven
- ◆ Uso extensivo de anotaciones que realizan funciones de configuración, inyección, etc.

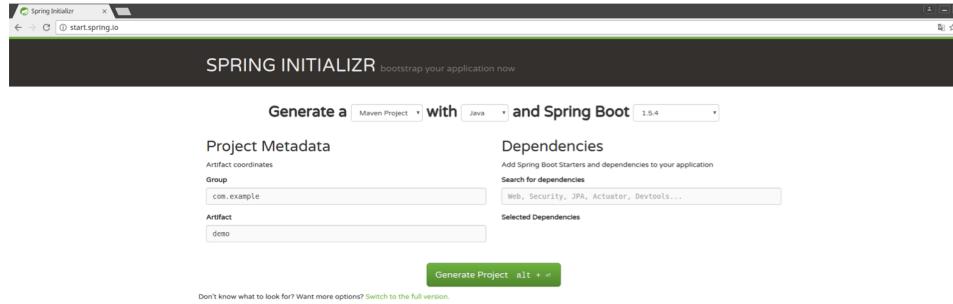
Spring Boot

Asistente Spring Boot (**Spring Initializr**)

Spring provee un servicio que nos permite generar nuestro proyecto de una manera muy fácil. Brinda una interfaz simple en la cual podemos elegir el nombre del proyecto (Artifact) el grupo (Group) elegimos el tipo de gestor de dependencias (Maven o Gradle) por último agregamos las dependencias a utilizar.

Spring Boot

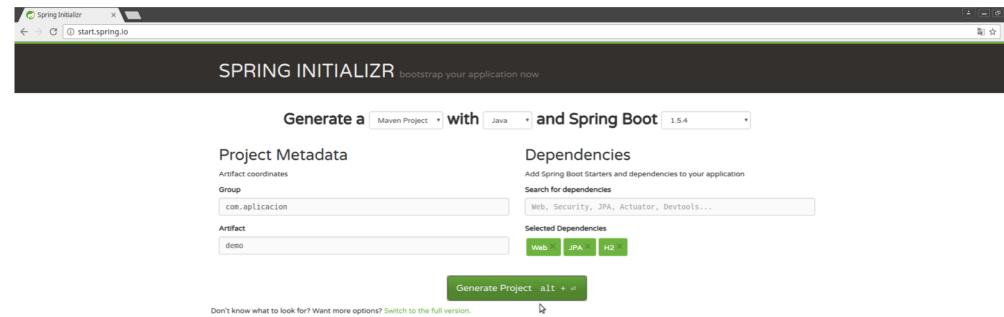
Puedes acceder a Spring Initializr con el siguiente enlace:
<http://start.spring.io/>



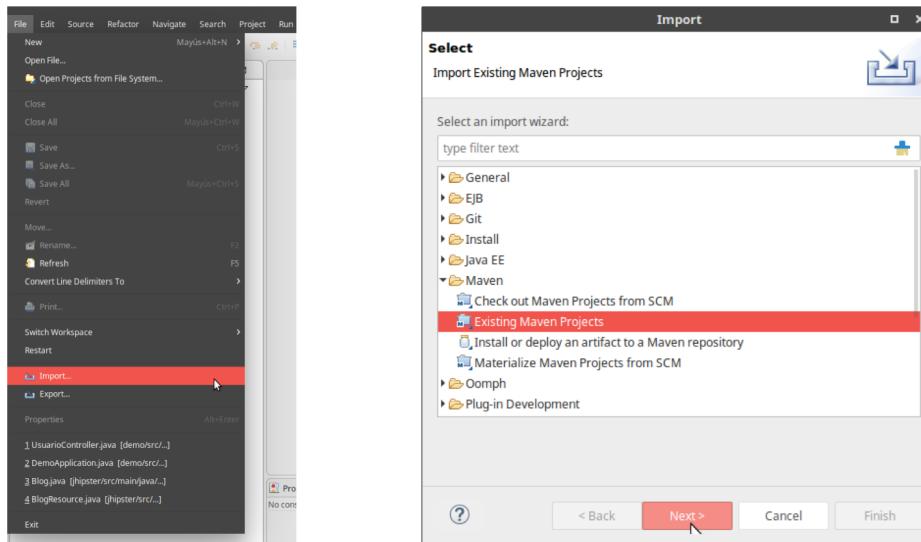
Crear mi primera aplicación



Para la creación de la aplicación, se utilizará el asistente **Spring Initializr**. En este caso se va a construir una aplicación Spring MVC y se seleccionara las dependencias web, JPA y H2, así como se muestra en la imagen



Luego de generar el proyecto se descomprime y se va importar en eclipse: **File -> Import**, saldrá una ventana y buscan la carpeta que dice **Maven -> Existing Maven Projects** para importar el proyecto



Si revisamos el archivo pom.xml, se puede ver que tiene agregada las dependencias que se seleccionaron con el asistente Web (spring-boot-starter-web), JPA (spring-boot-starter-data-jpa) y H2.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5
6   <groupId>com.aplicacion</groupId>
7   <artifactId>demo</artifactId>
8   <version>0.1-SNAPSHOT</version>
9   <packaging>jar</packaging>
10
11  <name>demo</name>
12  <description>Demo project for Spring Boot</description>
13
14<parent>
15   <groupId>org.springframework.boot</groupId>
16   <artifactId>spring-boot-starter-parent</artifactId>
17   <version>1.5.4.RELEASE</version>
18   <relativePath/> <!-- lookup parent from repository -->
19 </parent>
20
21<properties>
22   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23   <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24   <java.version>1.8</java.version>
25 </properties>
26
27<dependencies>
28   <dependency>
29     <groupId>org.springframework.boot</groupId>
30     <artifactId>spring-boot-starter-data-jpa</artifactId>
31   </dependency>
32   <dependency>
33     <groupId>org.springframework.boot</groupId>
34     <artifactId>spring-boot-starter-web</artifactId>
35   </dependency>
36
37   <dependency>
38     <groupId>com.h2database</groupId>
39     <artifactId>h2</artifactId>
40     <scope>runtime</scope>
41   </dependency>
42   <dependency>
43     <groupId>org.springframework.boot</groupId>
44     <artifactId>spring-boot-starter-test</artifactId>
45     <scope>test</scope>
46   </dependency>
47 </dependencies>
48
49<build>
50   <plugins>
51     <plugin>
52       <groupId>org.springframework.boot</groupId>
53       <artifactId>spring-boot-maven-plugin</artifactId>
54     </plugin>
55   </plugins>
56 </build>
57</project>
```

Ahora vamos a realizar con este proyecto un servicio Api Rest con la siguiente estructura:

- ◆ Modelo
- ◆ JPA Repository
- ◆ Servicios
- ◆ Controlador



En Eclipse se van a crear 4 paquetes para separar los archivos mencionados en el punto anterior

- ◆ com.aplicacion.demo.model
- ◆ com.aplicacion.demo.repository
- ◆ com.aplicacion.demo.service
- ◆ com.aplicacion.demo.controller



Crear Modelo

En el paquete model se va crear una nueva clase llamada “Blog” que será nuestra entidad; sus variables son:

- ◆ id
- ◆ titulo
- ◆ contenido



Ahora que tenemos la clase “Blog” agregada, colocamos el siguiente código que contiene las variables de la entidad y se usan las siguientes anotaciones @Entity para identificar que la clase es una Entidad, @Id y @GeneratedValue para identificar que la variables es un primary_key y auto_increment



```
Blog.java ✘
1 package com.aplicacion.demo.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Blog {
9
10     @Id
11     @GeneratedValue
12     private long id;
13
14     private String titulo;
15
16     private String contenido;
17
18     public Blog(){
19
20     }
21
22     public Blog(String titulo, String contenido){
23         this.titulo = titulo;
24         this.contenido = contenido;
25     }
26
27     public long getId() {
28         return id;
29     }
30
31     public void setId(long id) {
32         this.id = id;
33     }
34
35     public String getTitulo() {
36         return titulo;
37     }
38
39     public void setTitulo(String titulo) {
40         this.titulo = titulo;
41     }
42
43     public String getContenido() {
44         return contenido;
45     }
46
47     public void setContenido(String contenido) {
48         this.contenido = contenido;
49     }
50 }
```

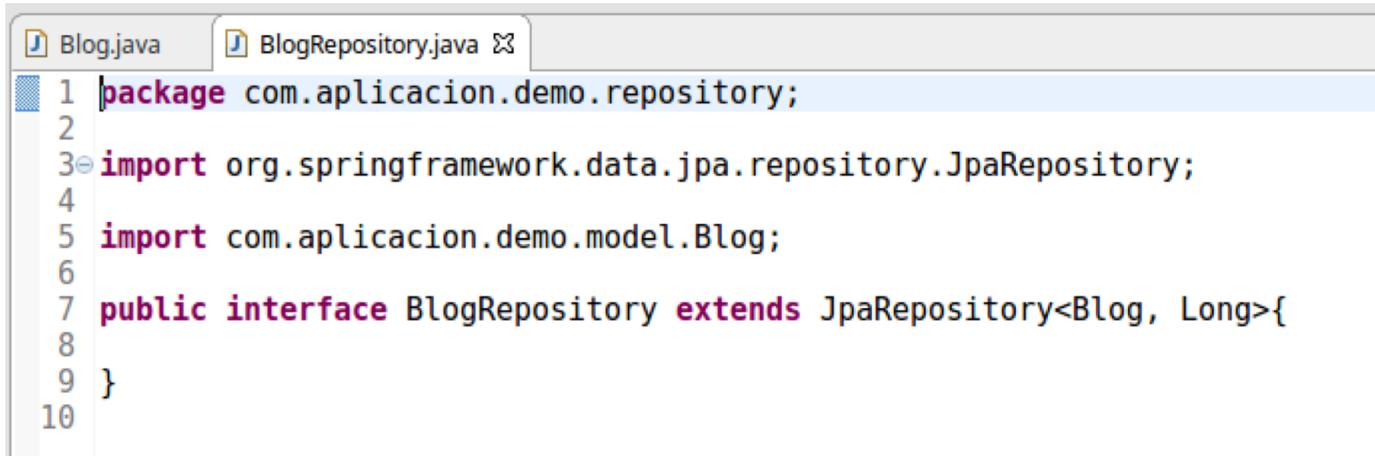
Crear JPA Repository

En este punto se va crear un Interface, el cual heredará de JpaRepository. Con esta interfaz se puede tener acceso a todos los métodos existentes de Jpa para realizar un CRUD rápidamente.

Para crear el archivo nos dirigimos nuevamente a eclipse y se selecciona el paquete repository, se da click derecho New -> Interface, se muestra una ventana para colocar el nombre el interface, este archivo se llamará “BlogRepository”.



El código que tendrá el archivo “BlogRepository”, quedará de la siguiente forma:



The screenshot shows a Java code editor with two tabs at the top: "Blog.java" and "BlogRepository.java". The "BlogRepository.java" tab is selected, showing the following code:

```
1 package com.aplicacion.demo.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.aplicacion.demo.model.Blog;
6
7 public interface BlogRepository extends JpaRepository<Blog, Long>{
8
9 }
10
```



Crear Servicio

Se llamará “BlogService”.

En este archivo la clase se identificará con la anotación @Service. Además se inyecta la clase “BlogRepository” con la anotación @Autowired para obtener los métodos que se usarán para el CRUD. El código queda de la siguiente forma:





```
Blog.java  BlogRepository.java  BlogService.java ✘
1 package com.aplicacion.demo.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import com.aplicacion.demo.model.Blog;
9 import com.aplicacion.demo.repository.BlogRepository;
10
11 @Service
12 public class BlogService {
13
14     @Autowired
15     private BlogRepository blogRepository;
16
17     public List<Blog> getAllBlog(){
18         return blogRepository.findAll();
19     }
20
21     public Blog getBlogById(long id){
22         return blogRepository.findOne(id);
23     }
24
25     public Blog saveBlog(Blog blog){
26         return blogRepository.save(blog);
27     }
28
29     public void removeBlog(Blog blog){
30         blogRepository.delete(blog);
31     }
32 }
```

Crear Controlador

Se llamará “BlogController”.

El controlador es el que se carga de manipular las rutas y retorna los valores según las peticiones que se hagan.



La clase controlador se identifica por medio de la notación @RestController. Se inyecta la clase “BlogService” para manipular los métodos creados del CRUD.

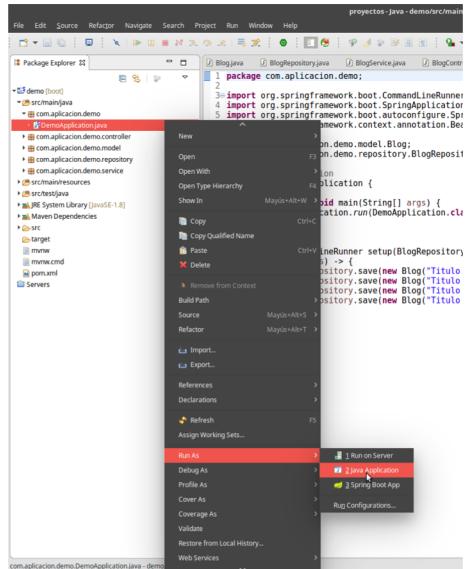


```
13  
14 import com.aplicacion.demo.model.Blog;  
15 import com.aplicacion.demo.service.BlogService;  
16  
17 @RestController  
18 public class BlogController {  
19  
20     @Autowired  
21     private BlogService blogService;  
22  
23     @RequestMapping(value = "/blog", method=RequestMethod.GET)  
24     public ResponseEntity<List<Blog>> getAllBlog(){  
25         return new ResponseEntity<List<Blog>>(blogService.getAllBlog(), HttpStatus.OK);  
26     }  
27  
28     @RequestMapping(value = "/blog/{id}", method = RequestMethod.GET)  
29     public ResponseEntity<Blog> getToDoById(@PathVariable("id") long id) {  
30         return new ResponseEntity<Blog>(blogService.getBlogById(id), HttpStatus.OK);  
31     }  
32  
33     @RequestMapping(value = "/blog", method = RequestMethod.POST)  
34     public ResponseEntity<Blog> saveToDo(@RequestBody Blog blog) {  
35         return new ResponseEntity<Blog>(blogService.saveBlog(blog), HttpStatus.OK);  
36     }  
37  
38     @RequestMapping(value = "/blog", method = RequestMethod.PUT)  
39     public ResponseEntity<Blog> updateToDo(@RequestBody Blog blog) {  
40         return new ResponseEntity<Blog>(blogService.saveBlog(blog), HttpStatus.OK);  
41     }  
42  
43     @RequestMapping(value = "/blog/{id}", method = RequestMethod.DELETE)  
44     public ResponseEntity<String> removeToDoById(@PathVariable("id") long id){  
45         Blog blog = blogService.getBlogById(id);  
46         blogService.removeBlog(blog);  
47         return new ResponseEntity<String>("Blog eliminado", HttpStatus.OK);  
48     }  
49 }
```



Ejecutar la aplicación

Seleccionar el archivo DemoApplication.java, hacer click derecho y seleccionar Run As -> Java Application



Realizar pruebas de llamadas http

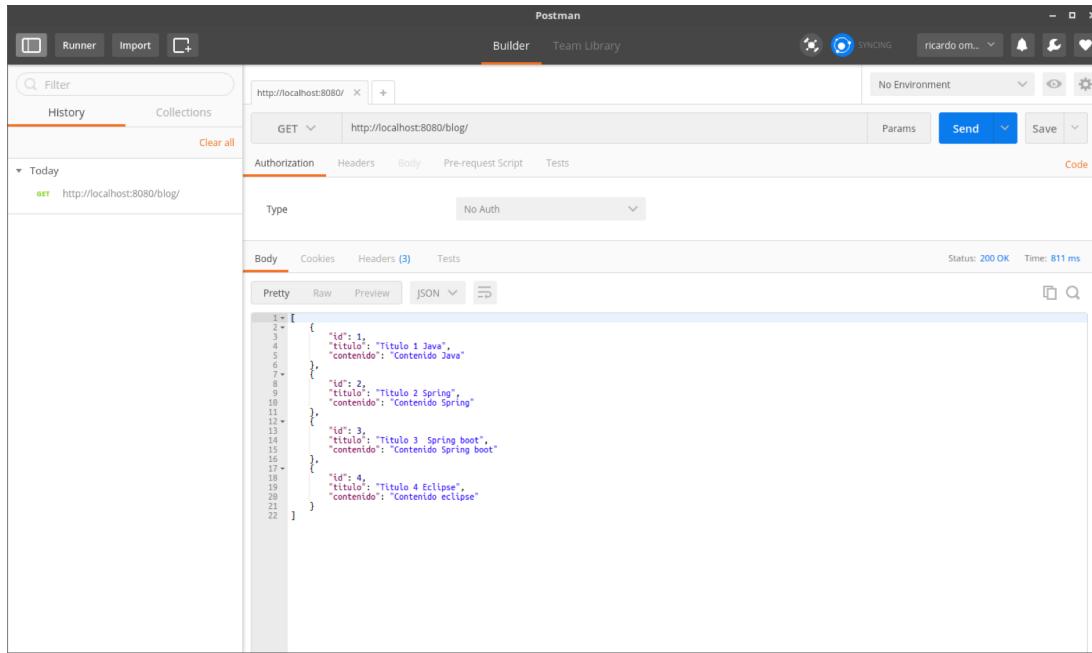
Para estas pruebas se utilizará la herramienta llamada Postman.

getpostman.com



Get All Blog

GET Request -> <http://localhost:8080/blog/>



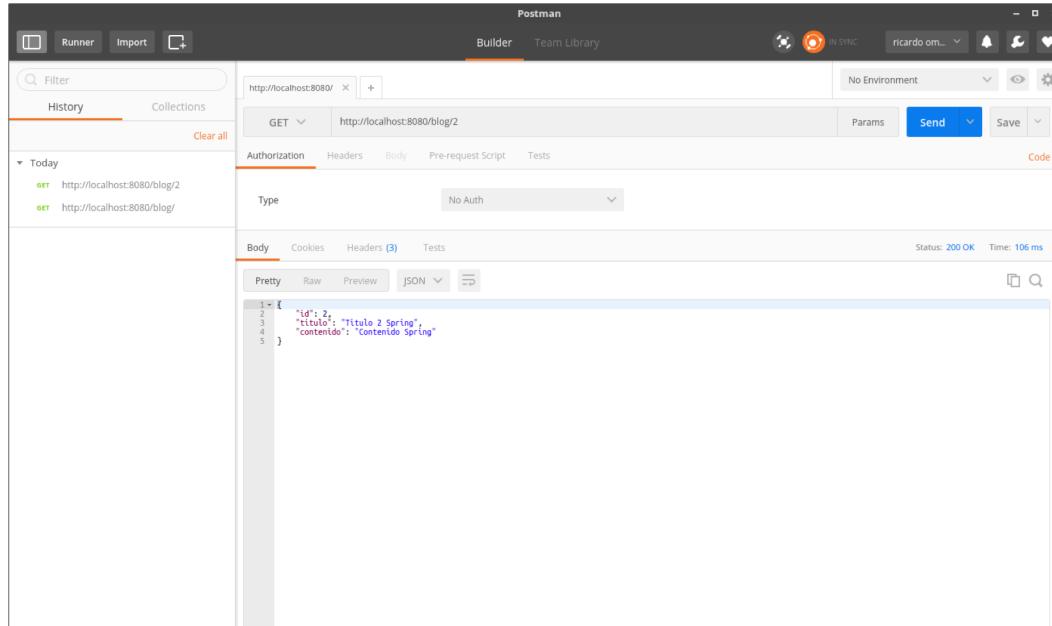
The screenshot shows the Postman application interface. In the top navigation bar, 'Builder' is selected. The main area displays a GET request to 'http://localhost:8080/blog/'. The response status is '200 OK' with a time of '811 ms'. The response body is a JSON array containing four blog entries:

```
[{"id": 1, "titulo": "Titulo 1 Java", "contenido": "Contenido Java"}, {"id": 2, "titulo": "Titulo 2 Spring", "contenido": "Contenido Spring"}, {"id": 3, "titulo": "Titulo 3 Spring boot", "contenido": "Contenido Spring boot"}, {"id": 4, "titulo": "Titulo 4 Eclipse", "contenido": "Contenido eclipse"}]
```



Get Blog by Id

GET Request -> <http://localhost:8080/blog/2>



The screenshot shows the Postman application interface. In the top bar, the URL `http://localhost:8080/` is selected. Below it, the specific endpoint `http://localhost:8080/blog/2` is entered. The "Send" button is highlighted in blue. The "Body" tab is active, showing a JSON response:

```
1: {  
2:   "id": 2,  
3:   "titulo": "Titulo 2 Spring",  
4:   "contenido": "Contenido Spring"  
5: }
```

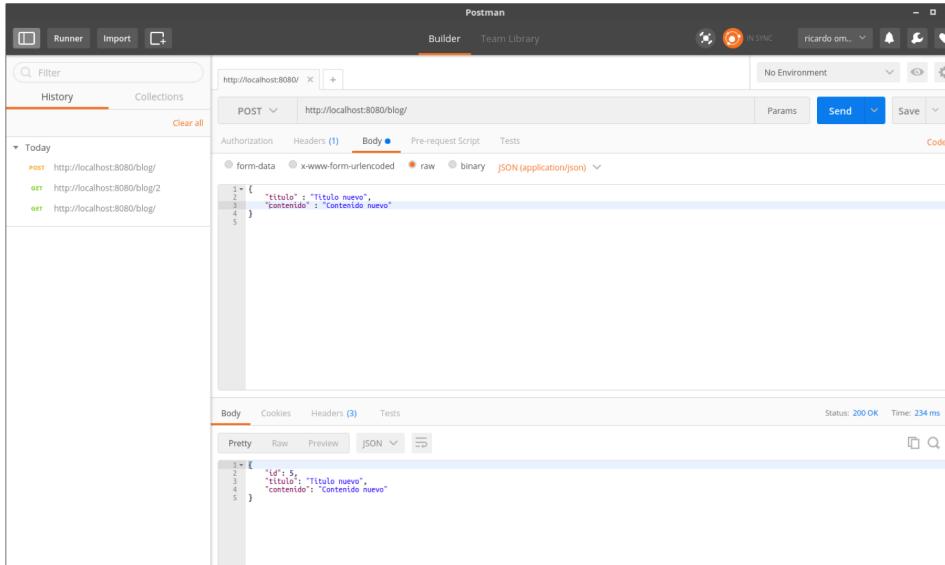
The status bar at the bottom indicates a `200 OK` response with a time of `106 ms`.



Create Blog

POST Request -> `http://localhost:8080/blog/`

Body -> `{"titulo": "Titulo nuevo", "contenido": "Contenido nuevo"}`



The screenshot shows the Postman application interface. In the left sidebar, under 'History', there are three entries: a successful POST request to `http://localhost:8080/blog/`, and two failed GET requests to `http://localhost:8080/blog/2`. The main workspace displays a POST request to `http://localhost:8080/blog/`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "titulo": "Titulo nuevo",  
3   "contenido": "Contenido nuevo"  
4 }
```

Below the request, the 'Body' tab is active, showing the raw JSON response received from the server:

```
1 {  
2   "id": 5,  
3   "titulo": "Titulo nuevo",  
4   "contenido": "Contenido nuevo"  
5 }
```

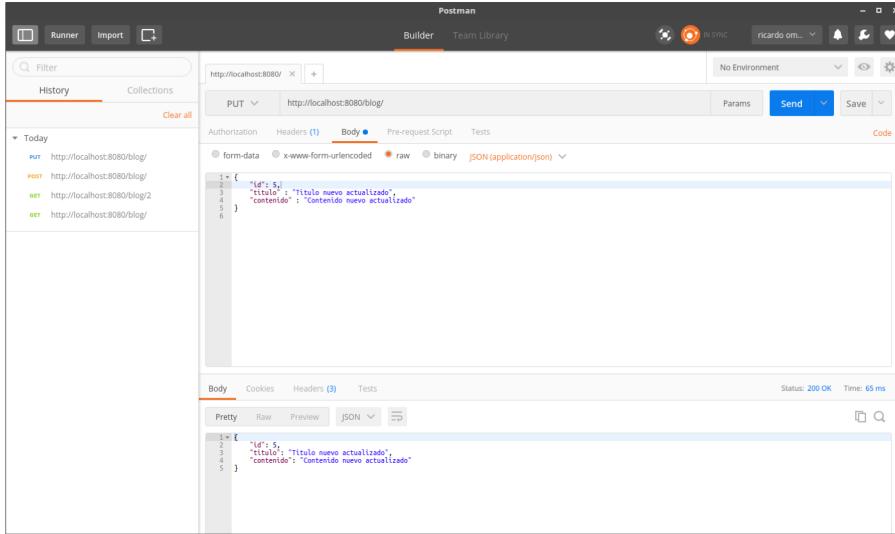
The status bar at the bottom indicates a `Status: 200 OK` and a `Time: 234 ms`.



Update Blog

PUT Request -> `http://localhost:8080/blog/`

Body -> `{"id": 5,"titulo" : "Titulo nuevo actualizado", "contenido" : "Contenido nuevo actualizado"}`



The screenshot shows the Postman application interface. The URL is set to `http://localhost:8080/blog/` and the method is `PUT`. The `Body` tab is selected, showing the JSON payload:

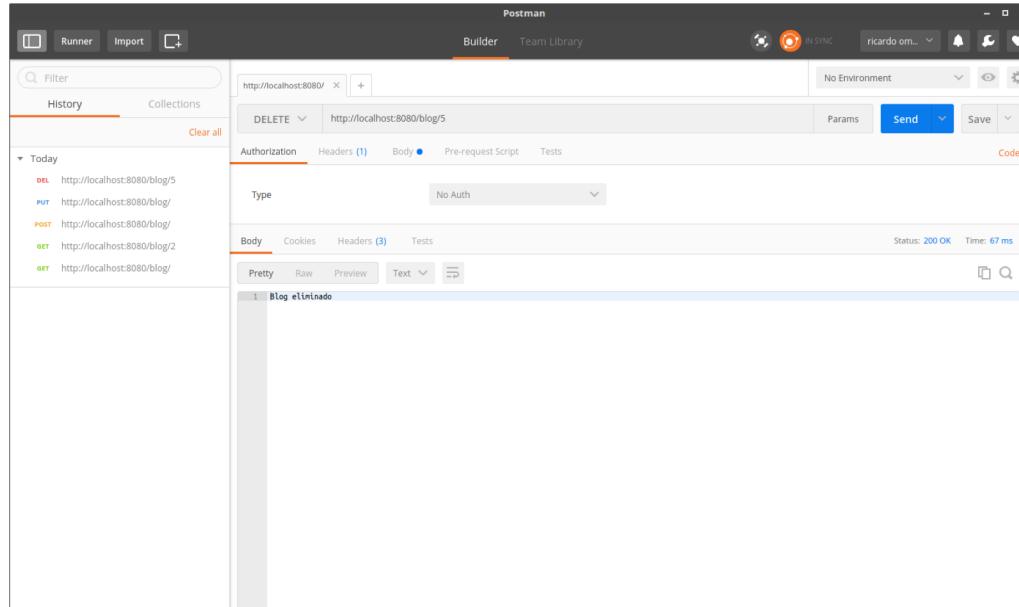
```
[{"id": 5, "titulo": "Titulo nuevo actualizado", "contenido": "Contenido nuevo actualizado"}]
```

The response panel shows a `200 OK` status with a `Time: 65 ms`. The response body is identical to the sent body.



Delete Blog

DELETE Request -> http://localhost:8080/blog/5



The screenshot shows the Postman application interface. In the left sidebar, under 'History', there is a list of requests made today, including a successful DELETE request to `http://localhost:8080/blog/5`. The main workspace displays a DELETE request to `http://localhost:8080/blog/5` in the 'Builder' tab. The 'Body' tab shows the response body, which contains the message `Blog eliminado`.



Conclusión del ejemplo

Como podemos ver en el ejemplo, Spring Boot nos facilita la creación de desarrollar aplicaciones rápidamente con todas las configuraciones necesarias, dejándonos más tiempo libre para que nos dediquemos en desarrollar la funcionalidad de la aplicación sin tener que complicarnos con la gestión de las librerías.



Entre las ventajas se puede apreciar lo siguiente:

- ◆ El contenedor de Spring y la utilización de inyección de dependencias permite obtener un código más desacoplado, permitiendo añadir y quitar módulos fácilmente.
- ◆ Los módulos que ofrece son completamente configurables y compatibles entre sí.

- ◆ Añadir módulos externos es posible gracias al contenedor de Spring.
- ◆ Spring MVC permite aplicar un modelo de diseño que permite desarrollar de forma ordenada y rápida.



Spring nos permite desarrollar aplicaciones de microservicios o REST, y esto nos permite extender el proyecto más allá de una aplicación web.

Cada vez es más común extender las aplicaciones a móviles y esto nos permite utilizar la misma lógica de servidor para persistir los datos y mostrarlos utilizando distintas tecnologías.



Gracias!

Alguna pregunta?

Encuentranos en www.impactotecnologico.net



FIN

