



DIPLOMADO

Desarrollo de sistemas con tecnología Java

Módulo 10

API RESTful con Spring Boot

M. en C. Jesús Hernández Cabrera



Convenios

- Tolerancia de inicio de clase 15 min
- 20 minutos de receso

Evaluación

- Prácticas 30%
- Participación 5%
- Ejercicios 40%
- Avance de proyecto final 25%

Objetivo

- El participante será capaz de desarrollar, configurar y manejar servicios REST utilizando Spring Boot, para crear soluciones eficientes y escalables en el mundo moderno de la tecnología de la información.

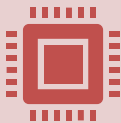
Temas

1. Introducción a los servicios REST
2. Principios REST
3. Anotaciones en SpringBoot
4. Desarrollo de servicios web con SpringBoot
5. Manejo de las respuestas a los métodos y solicitudes HTTP
6. Extracción de los parámetros de la solicitud
7. Configuración de la aplicación para exponer servicios REST con SpringBoot
8. Manejo de errores en servicios REST con SpringBoot
9. Consulta de Servicios REST

¿Qué es REST?



REST (**Representational State Transfer**), es una arquitectura de diseño de servicios web que utiliza métodos estándar de HTTP para facilitar la comunicación entre sistemas.



Permite la interacción entre aplicaciones independientemente de su lenguaje de programación, como un servicio Java intercambiando datos con una aplicación Python. Es decir, poder hacer intercomunicación entre aplicaciones escritas en diferentes lenguajes.



En REST, los recursos —entidades accesibles a través de URLs— se manejan mediante operaciones definidas por métodos HTTP. La arquitectura REST asegura la escalabilidad y simplicidad, denominándose implementaciones como APIs RESTful

Ventajas de REST

- **Visibilidad.**- El servidor solo requiere saber la información propia de las peticiones(request) durante los intercambios de datos.
- **Confiabilidad.**- Esta arquitectura permite la recuperación de fallas parciales durante el intercambio de información. Y a que no depende del estado de una sesión previa o almacenamiento de estado.
- **Escalabilidad.**- Dado que el servidor no debe almacenar el estado de las conversaciones con el cliente, este puede liberar recursos rápidamente, específicamente esto pasa cuando una petición ha sido respondida por completo

REST

Independencia de
plataforma.

Independencia de
lenguaje de
programación.

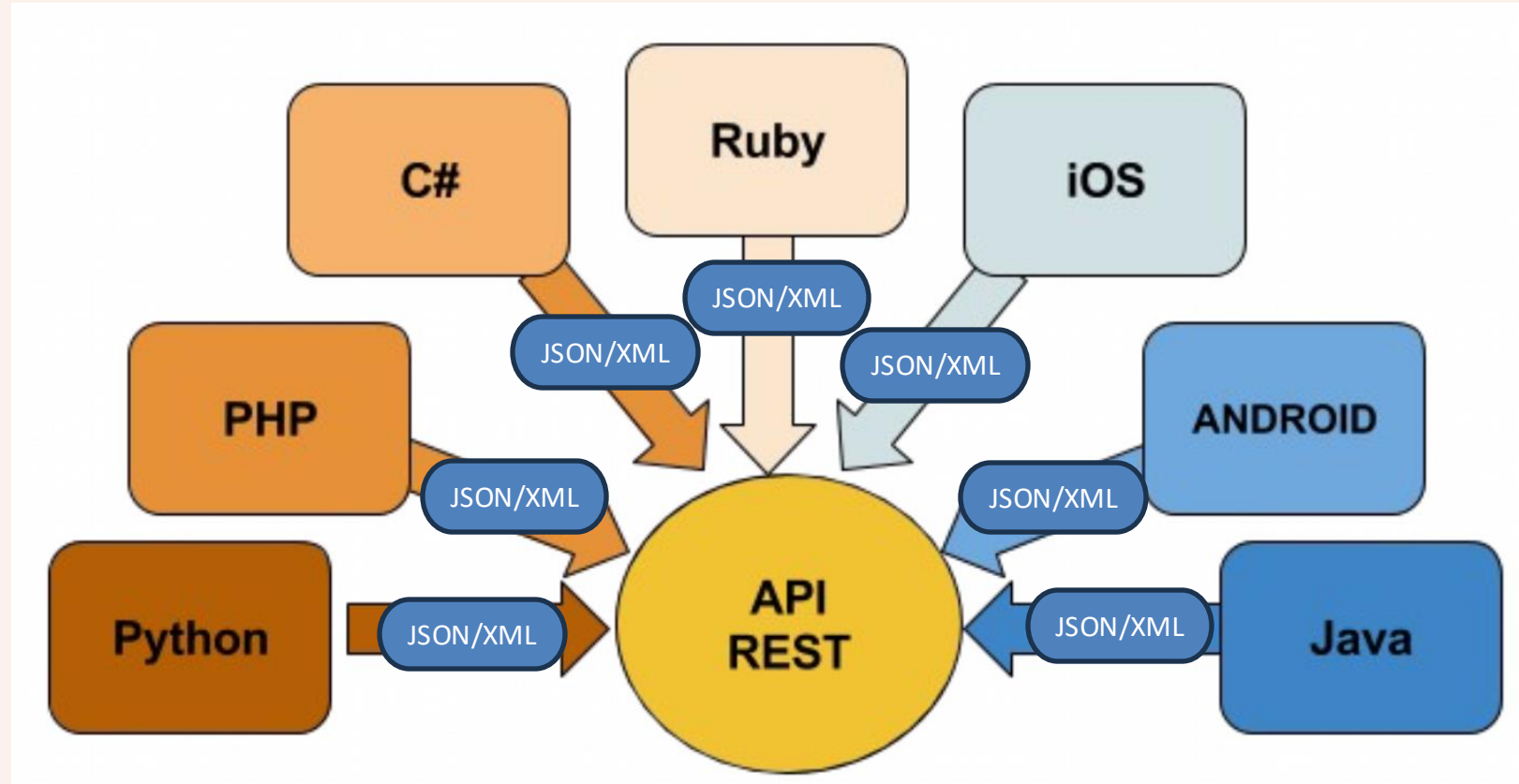
Basado en
estándares.

Puede ser usado
fácilmente en
presencia de
firewall.

Protocolo
cliente/servidor
sin estado

Sólo utiliza HTTP

REST



REST

Para manipular los recursos, REST hace uso de los siguientes verbos(métodos) del protocolo HTTP.

- GET: Para consultar y leer recursos
- POST: Para crear recursos
- PUT: Para editar recursos
- DELETE: Para eliminar recursos.
- PATCH: Para editar partes concretas de un recurso.

REST vs SOAP

Característica	SOAP	REST
Formatos de intercambio	Solo XML	XML y JSON
Uso transaccional	Muy bueno	No recomendable.
Filosofía	Open Web	Empresarial y formal
Complejidad de implementación	difícil, requiere de una curva de aprendizaje.	Relativamente fácil de implementar.
Mantenimiento	Requiere de esfuerzo.	Fácil de mantener.
Seguridad	Es parte del protocolo. Es buena.	No forma parte del protocolo, se implementa empleando tecnologías adicionales como por ejemplo los Web Tokens.
Eficiencia	Sobrecargado	Eficiente
Estandarizado	Si <u>WS-Standards</u>	No

RESTful

En el estilo
arquitectónico REST
todo es visto como un
recurso.

Un servicio web
basado en el estilo
REST es conocido como
RESTful.

Un ***recurso*** en REST es
una entidad que puede
ser identificada de
forma única mediante
una URI

RESTful

- Uso correcto de URI
- Cada página, información en una sección, archivo, se nombra como **recursos**.
- El recurso por lo tanto es la información a la que se quiere acceder, modificar o borrar, **independientemente de su formato**.

RESTful

- Las URL, Uniform Resource Locator , son un tipo de URI, Uniform Resource Identifier, que además de permitir **identificar de forma única el recurso**, permite localizarlo para poder acceder a él o compartir su ubicación.
- Una URL se estructura de la siguiente forma:
 - {protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

RESTful

- Reglas básicas para poner nombre a la URI de un recurso:
 - Los nombres de URI no deben implicar una acción, por lo tanto, debe evitarse usar verbos en ellos.
 - Deben ser únicas.
 - Deben ser independiente de formato.
 - Deben mantener una jerarquía lógica.
 - Los filtrados de información de un recurso no se hacen en la URI.

RESTful Recurso

- Cualquier cosa que puede ser identificado por una URI: documento, imagen, servicio.
- Se analizan los casos de uso para encontrar nombres de dominio que puedan realizar las operaciones de “crear”, “leer”, “actualizar” o “borrar”
 - /clientes
 - /clientes/Id
 - /archivos/pdfs

API's para practicar

- <https://ed.team/blog/las-mejores-apis-publicas-para-practicar>

REST y Verbos HTTP

Método HTTP	Operación CRUD	Respuestas HTTP <u>Status</u>, Elemento específico: (ejemplo: /users/{id})	Respuestas HTTP <u>Status</u> a la colección entera. (ejemplo: /users/)
POST	Create / insert	404	201 Created Res.: el documento con nuevo _id
GET	Read /Select	200 OK Resp.: Un sólo documento	/users/: 200 OK, Resp.: Todos los documentos
PUT	Replace / Update	200 OK ó 204 (Body sin contenido). Ó 404 (id no encontrado)	405 acción no permitida.
PATCH	Update Only	200 (OK) or 204 (Body sin contenido). 404 (No encontrado)	405 acción no permitida.
DELETE	Delete	200 (OK) ó 404 (No encontrado)	405 acción no permitida.

RESTful



GET /users acceder al listado de usuarios



POST /users crear un usuarios nuevo.



GET /users/123 acceder al detalle de un usuario



PUT /users/123 editar el usuario cuyo id es 123, sustituyendo la totalidad de la información anterior por la nueva.

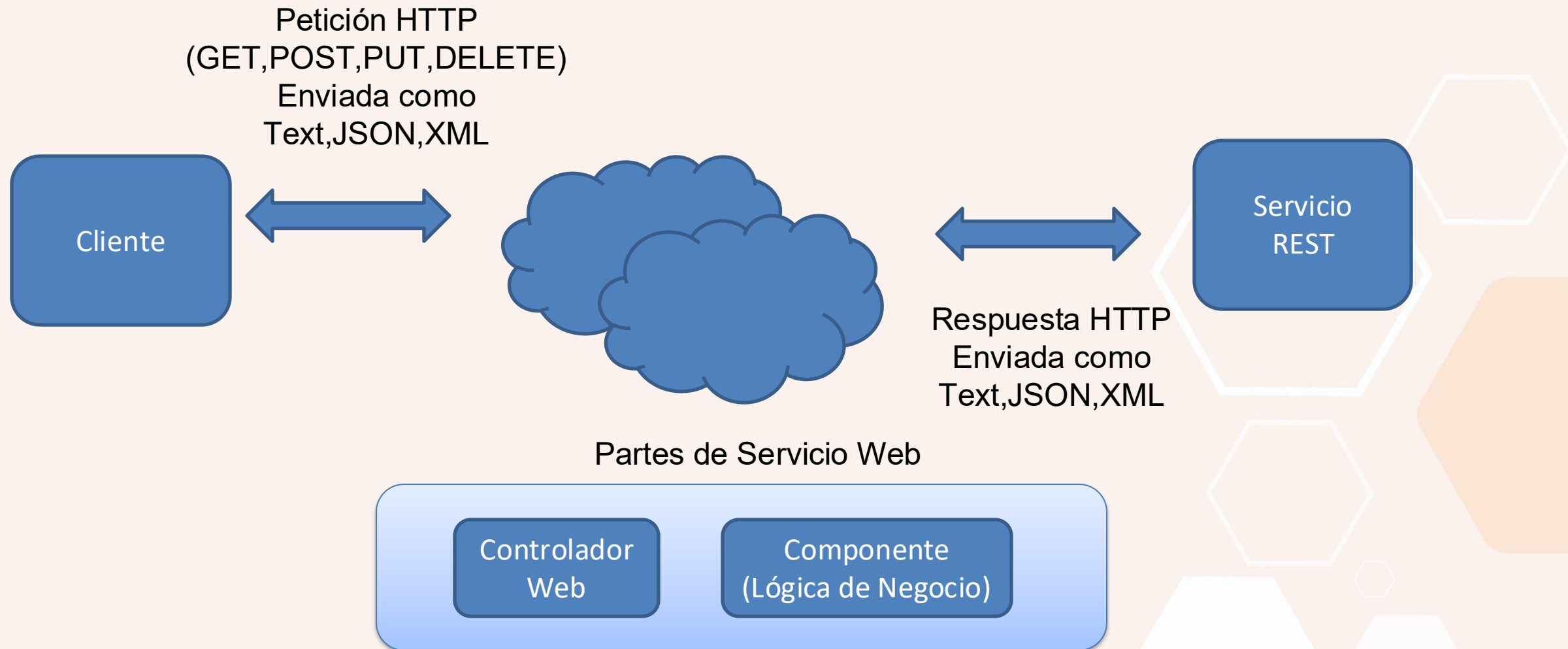


DELETE /users/123 eliminar el usuario cuyo id es 123.

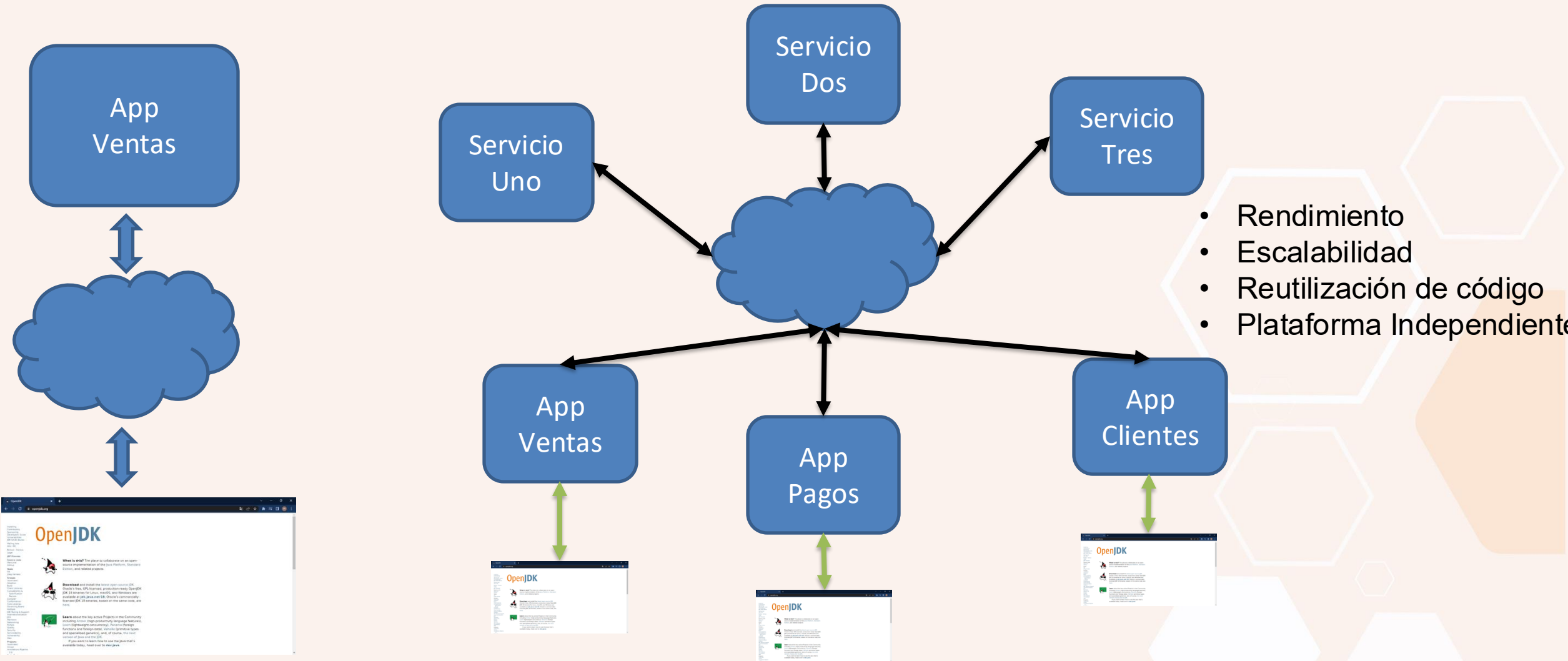


PATCH /users/123 modificar parcial de información de un usuario, como por ejemplo solo modificar el nombre.

Esquema de Intercambio de Datos

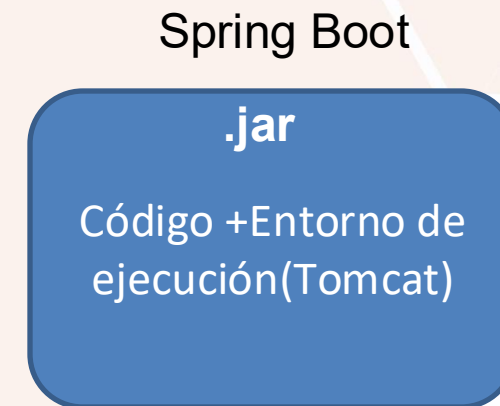


Monolíticas y Servicios Web



Arquitectura Microservicios

- Será un enfoque para desarrollar una aplicación en pequeños servicios fragmentados, en donde cada uno se ejecutará en su propio proceso, es decir incluirá todo lo necesario para su ejecución, sin dependencia de ningún software adicional.
- Contando un despliegue automático e independiente.



**Project**☐ Gradle - Groovy ☐ Gradle - Kotlin☒ Maven**Language**☒ Java ☐ Kotlin ☐ Groovy**Spring Boot**☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M3) ☐ 3.3.5 (SNAPSHOT) ☒ 3.3.4☐ 3.2.11 (SNAPSHOT) ☐ 3.2.10**Project Metadata**

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 23 ☐ 21 ☒ 17

Dependencies[ADD DEPENDENCIES...](#) ⌘ + B**Spring Web** WEB

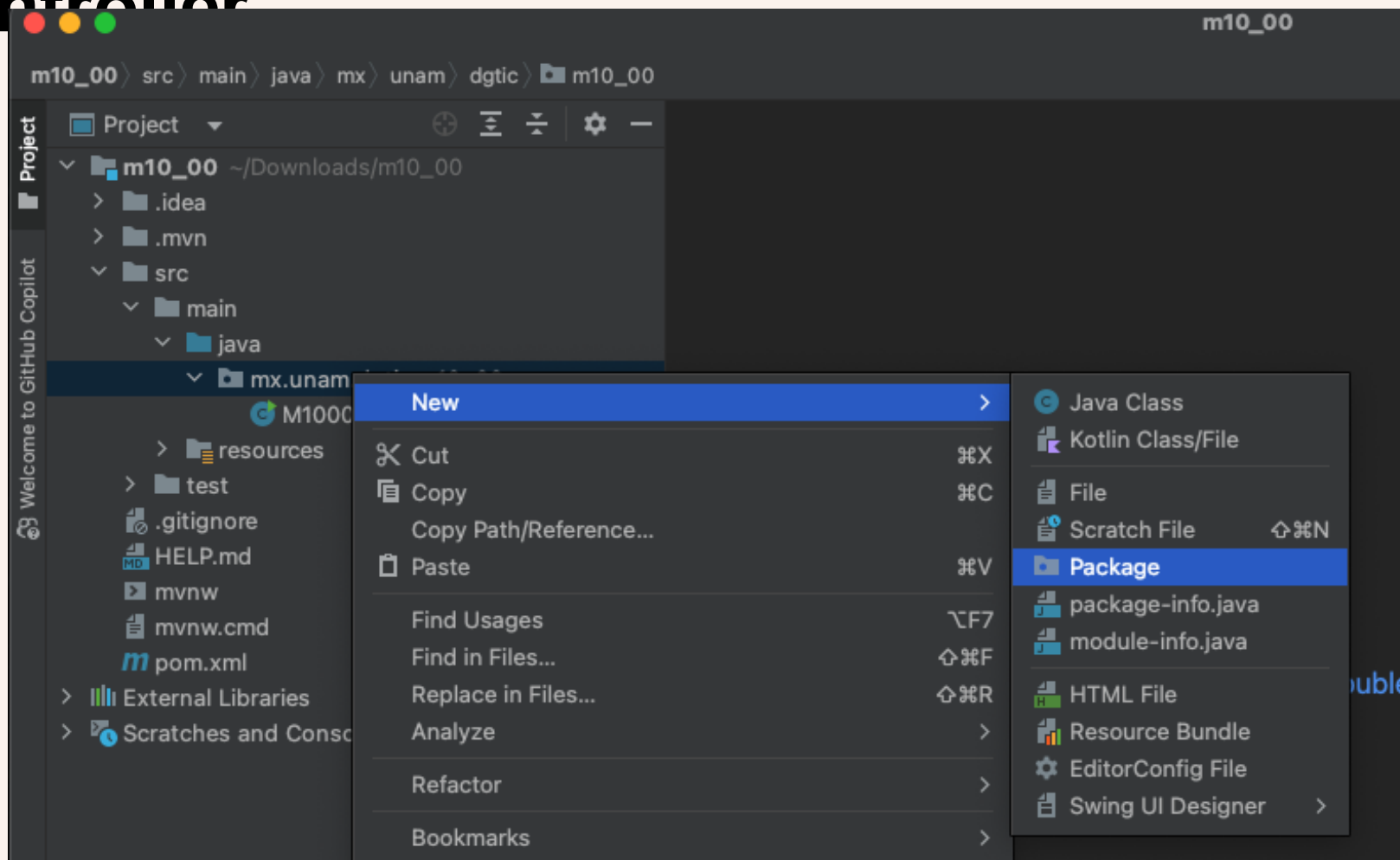
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

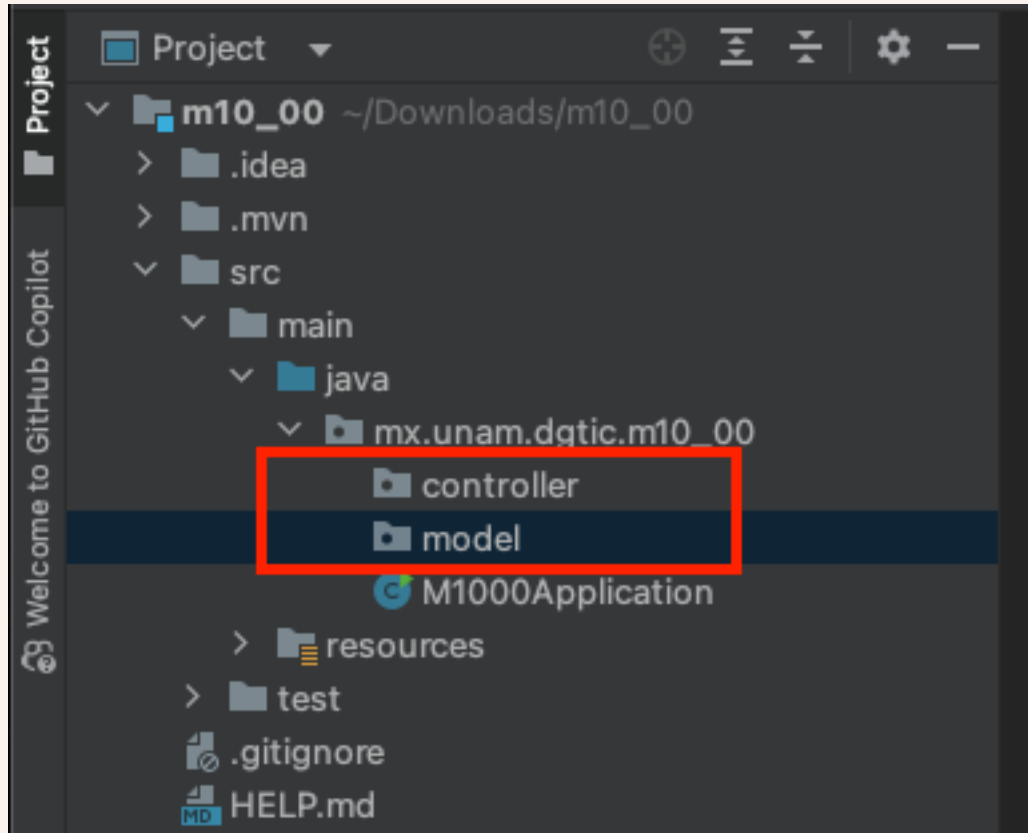
Spring Boot DevTools DEVELOPER TOOLS

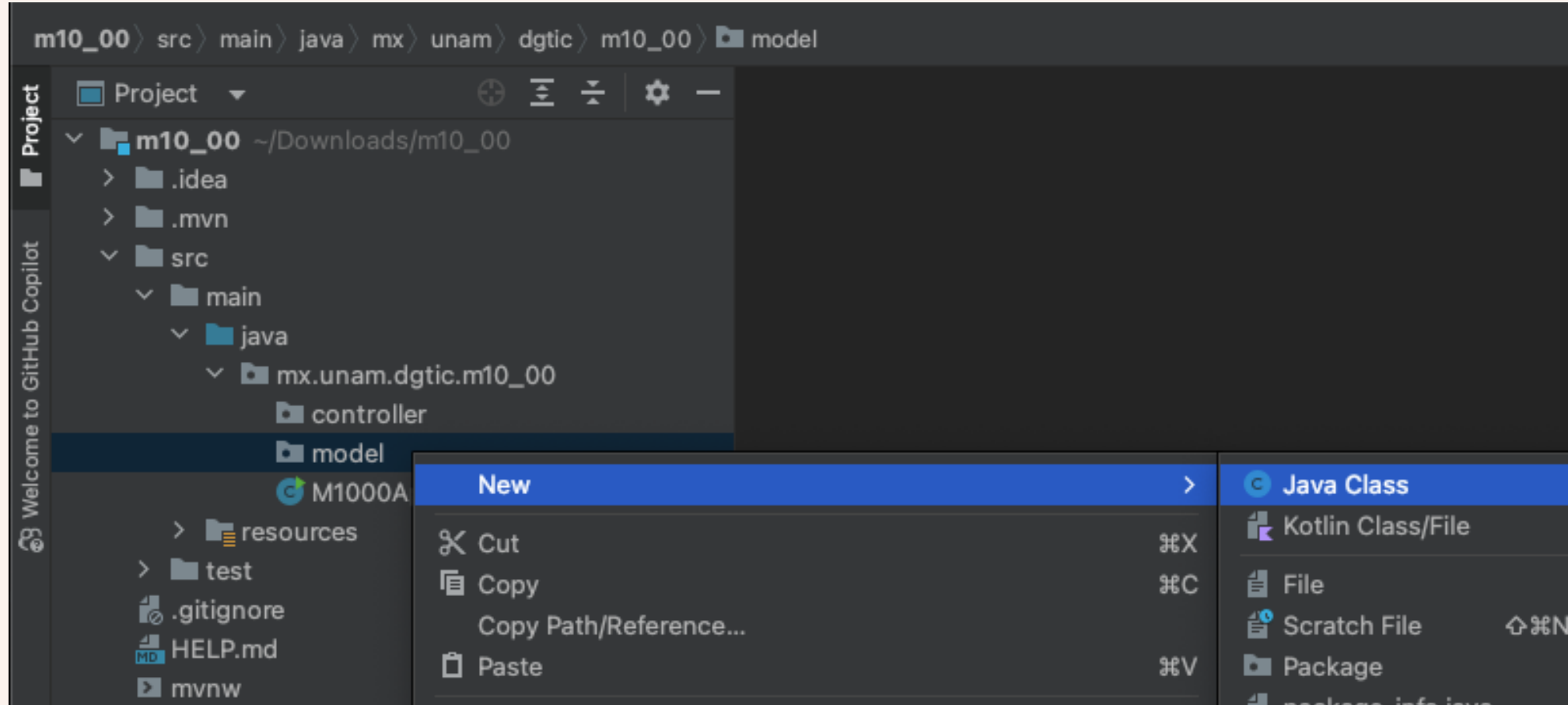
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

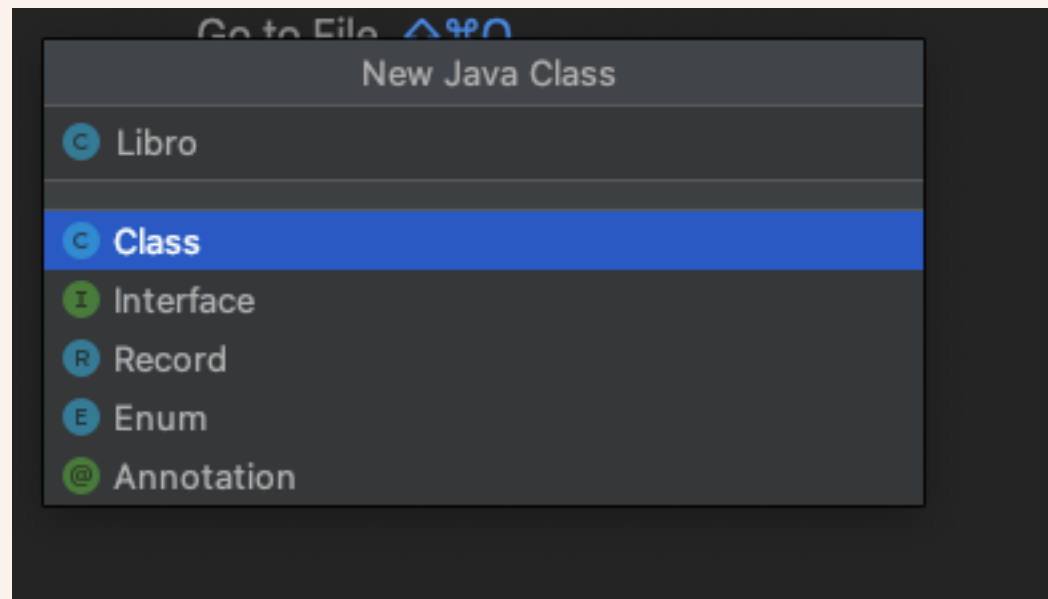
[GENERATE](#) ⌘ + ↵[EXPLORE](#) CTRL + SPACE[SHARE...](#)

RestController



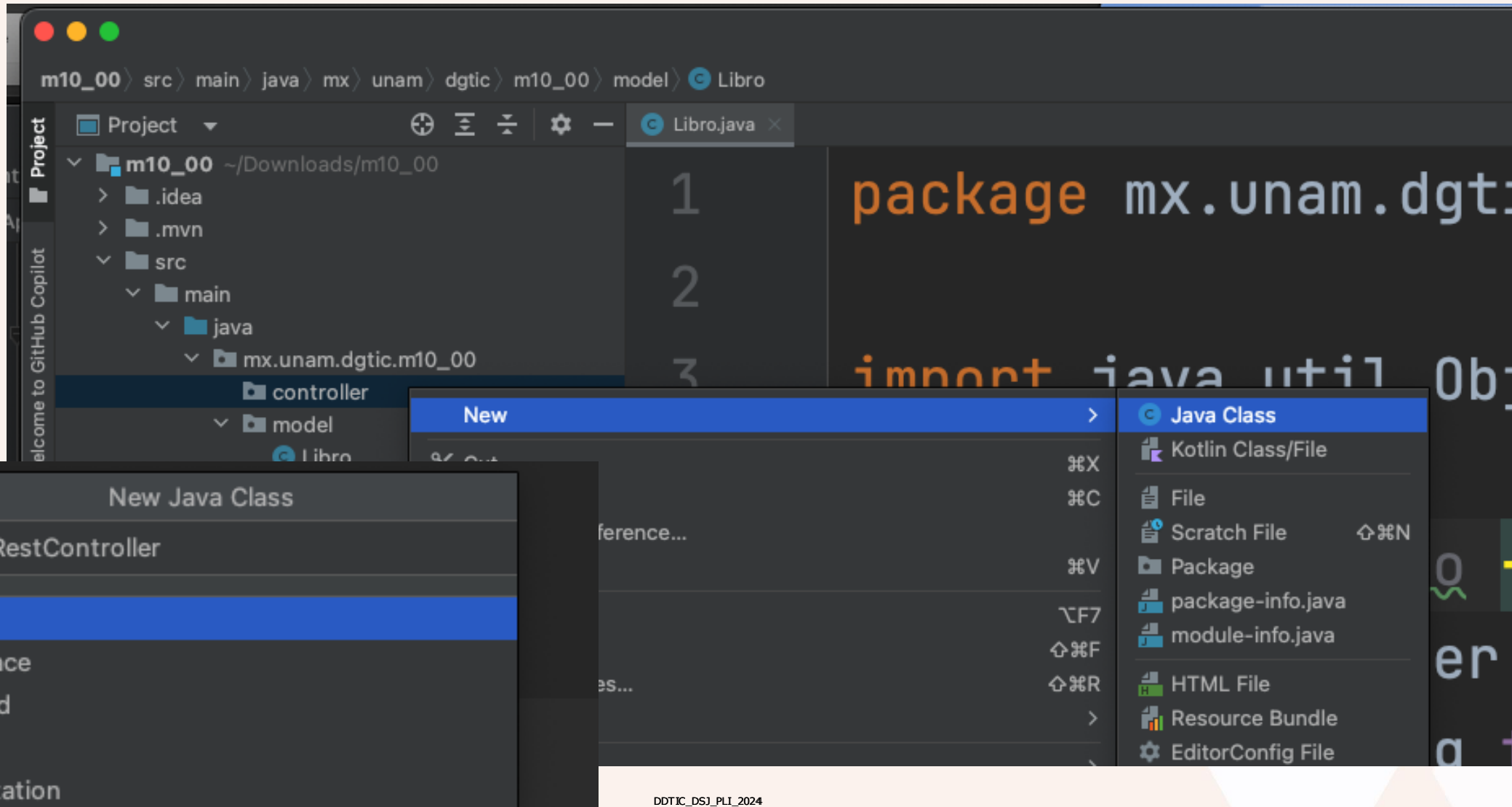


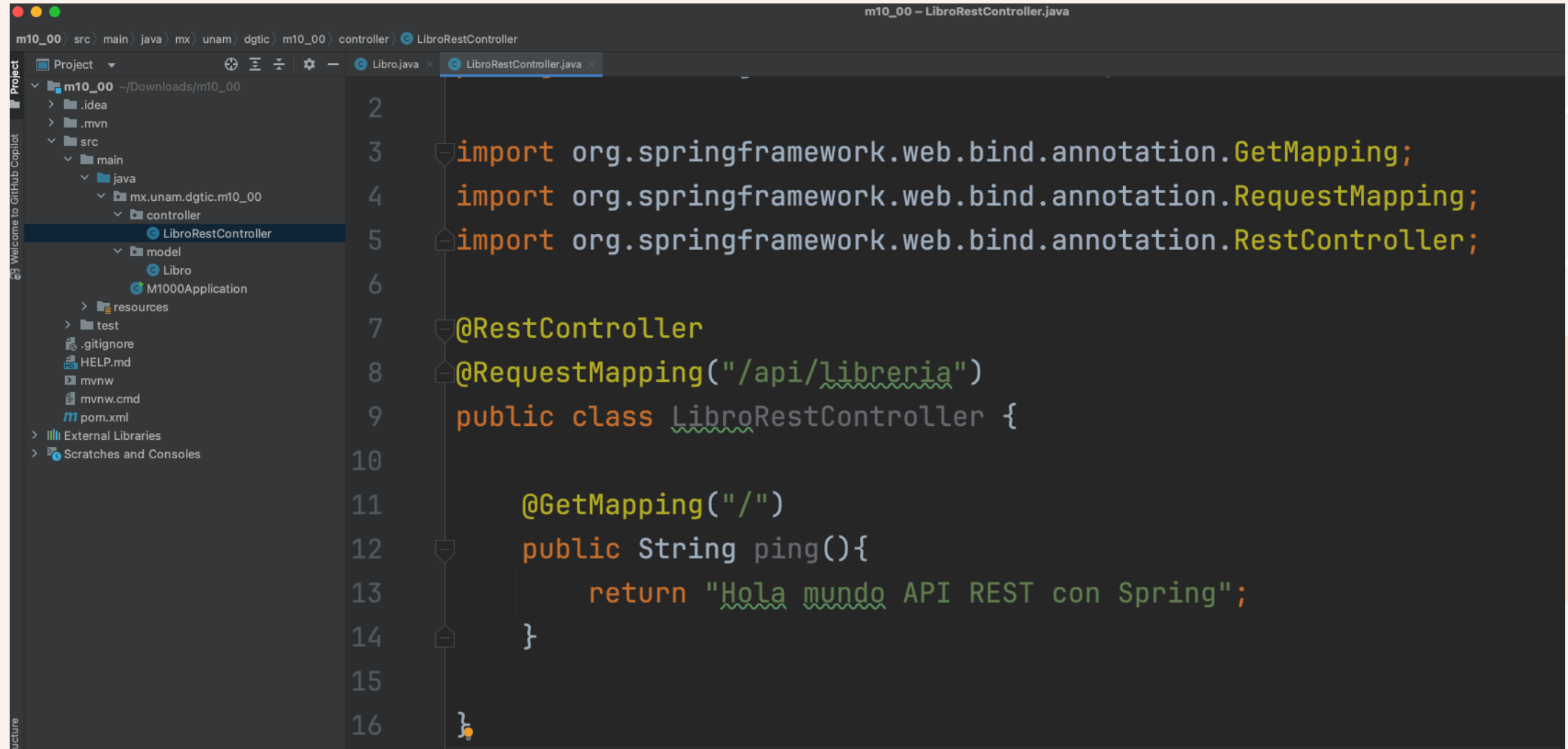




```
Libro.java x
1  package mx.unam.dgtic.m10_00.model;
2
3  import java.util.Objects;
4
5  public class Libro {
6      private Integer id;
7      private String titulo;
8      private String autor;
9
10     public Libro() {}
11
12
13     public Libro(Integer id, String titulo, String autor) {
14         this.id = id;
15         this.titulo = titulo;
16     }
17 }
```

Crear el controlador REST





```
m10_00 - LibroRestController.java
m10_00 src main java mx unam dgtic m10_00 controller LibroRestController
Project
  m10_00 ~/Downloads/m10_00
    .idea
    .mvn
    src
      main
        java
          mx.unam.dgtic.m10_00
            controller
              LibroRestController
            model
              Libro
            M1000Application
          resources
        test
      .gitignore
      HELP.md
      mvnw
      mvnw.cmd
      pom.xml
    External Libraries
    Scratches and Consoles
  Welcome to GitHub Copilot
  Structure

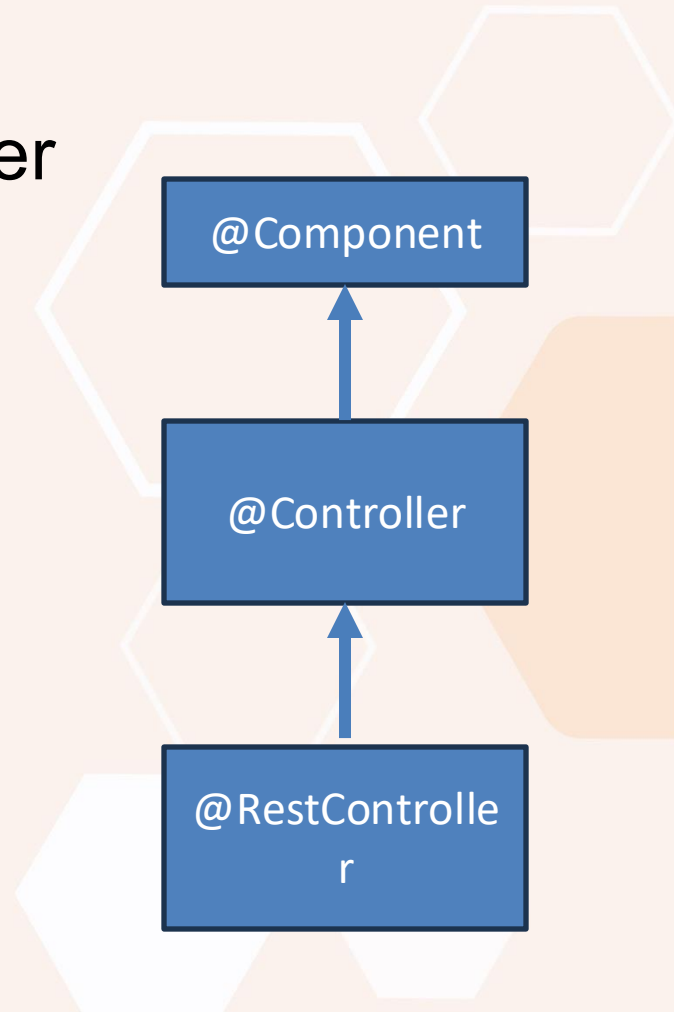
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 @RequestMapping("/api/libreria")
9 public class LibroRestController {
10
11     @GetMapping("/")
12     public String ping(){
13         return "Hola mundo API REST con Spring";
14     }
15
16 }
```

RestController

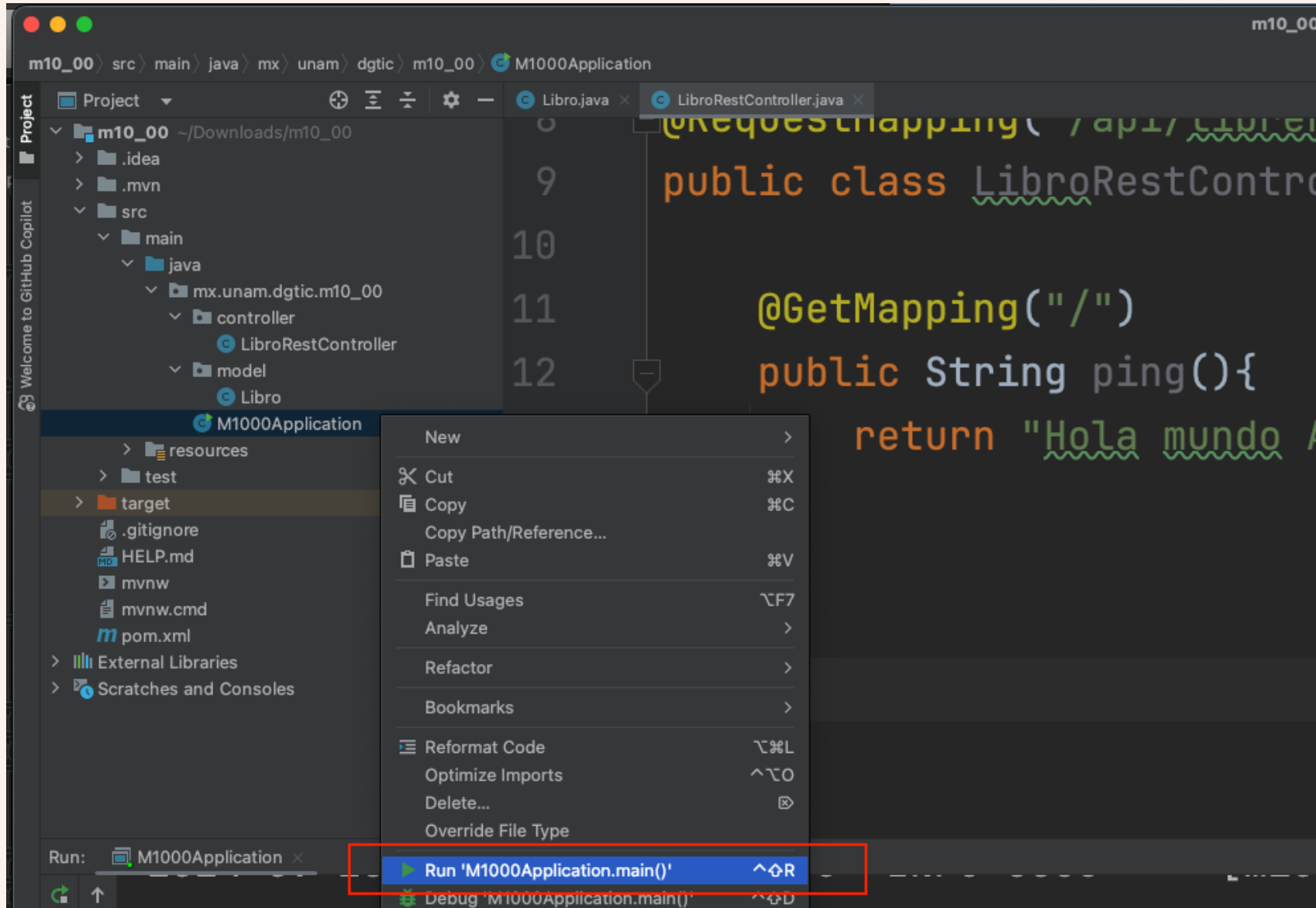
- En el código, se ha definido una clase de controlador REST LibroRestController anotado con **@RestController**, lo que indica que manejará las solicitudes HTTP.
- La anotación **@RequestMapping** especifica la ruta base para todos los puntos finales de API definidos en este controlador.
 - En este caso, la ruta base es /api/libreria.

@RestController

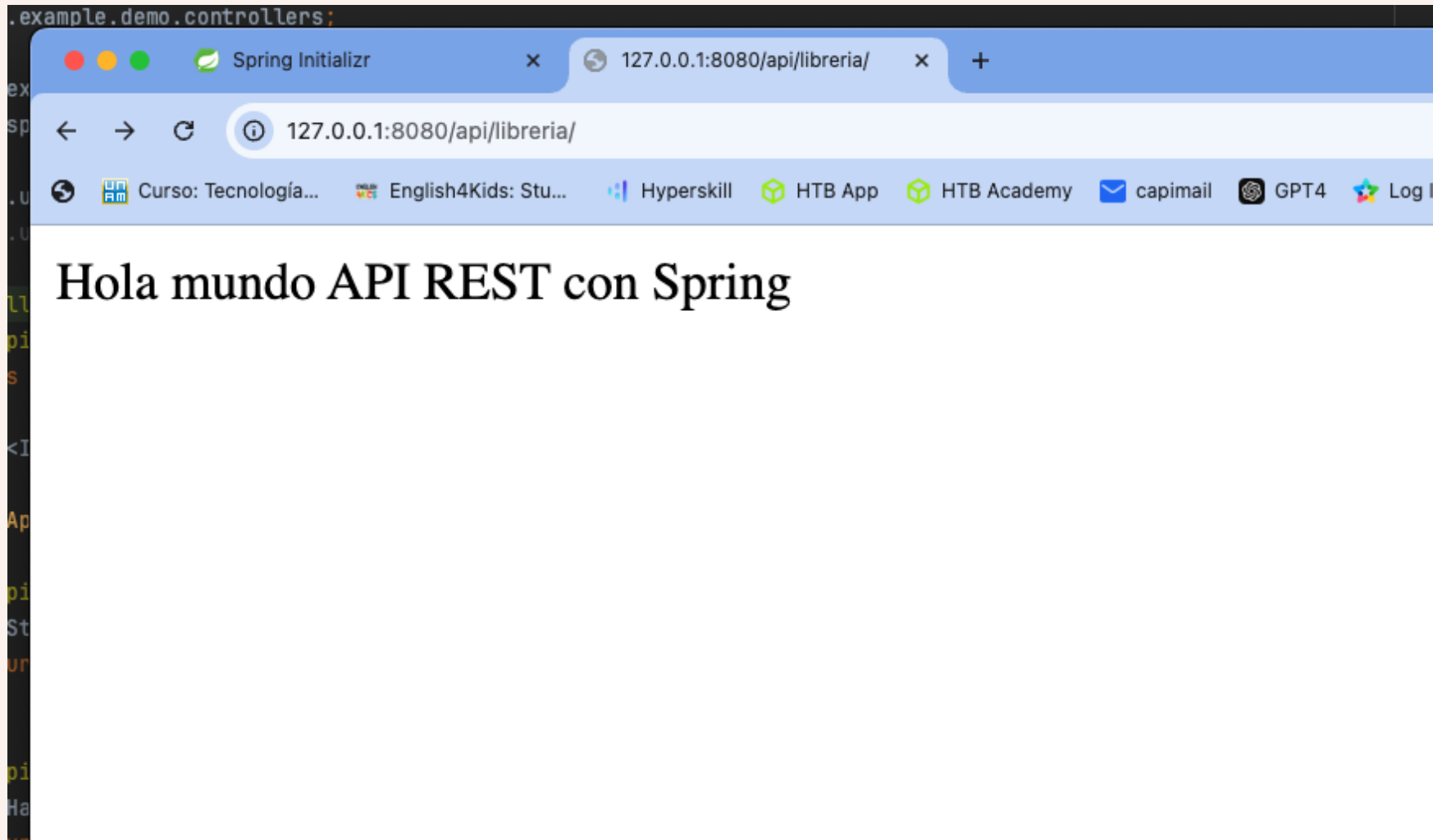
- Es una especialización de la jerarquía de @Component.
- @RestController es equivalente a usar @Controller en combinación con @ResponseBody.
- Convierte automáticamente los objetos Java devueltos en la respuesta (como POJOs o colecciones) a JSON o XML(serialización) y viceversa.



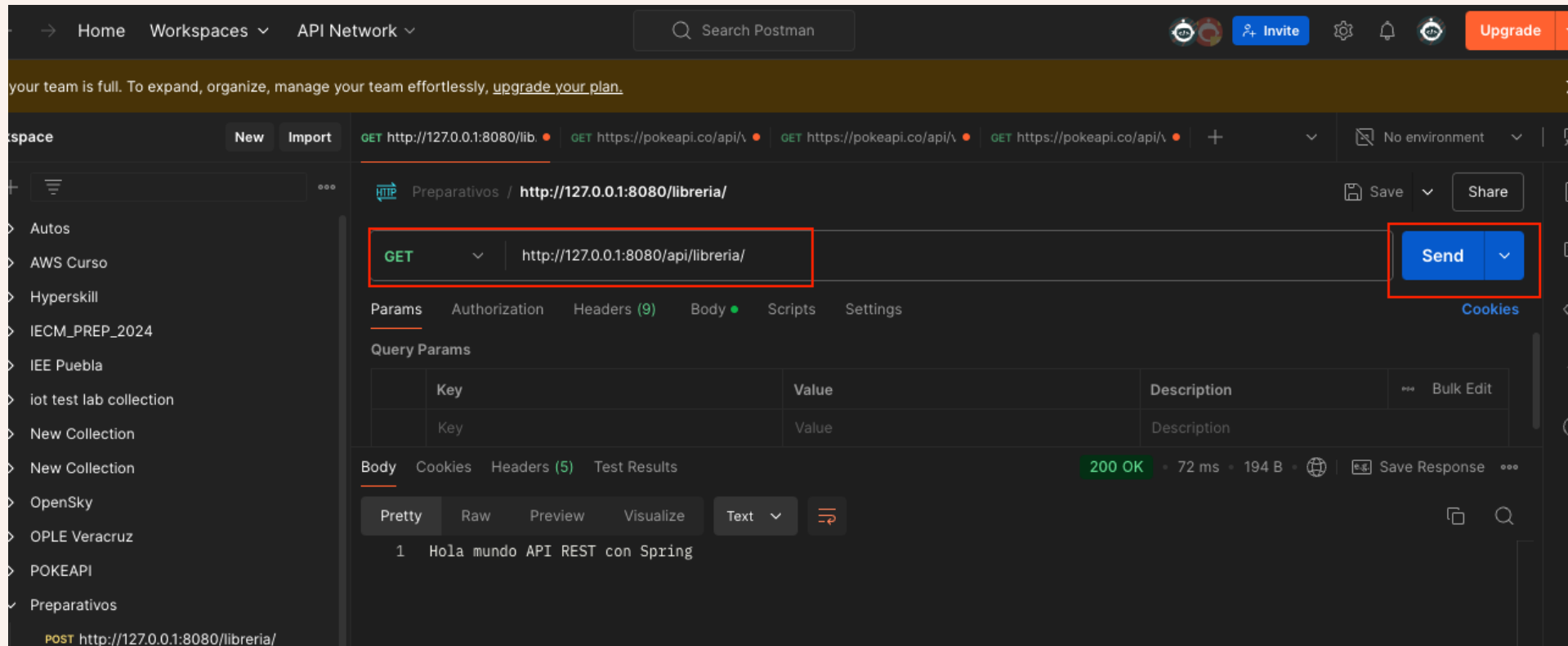
Ejecutar la aplicación



Probarla en ...:8080/api/libreria/



Probar la aplicación con PostMan



Contacto

M. en C. Jesús Hernández Cabrera
Profesor de carrera

jesushc@unam.mx

Redes sociales:



www.linkedin.com/in/hcjesus