**Actividad: Explorando package controller**

Aprender el uso básico de los controladores en Spring, centrándose en cómo crear endpoints GET y recibir parámetros.

* Investigación:
  + Comprender el propósito de los controladores en Spring.
  + Investigar cómo se crean y configuran endpoints GET en un controlador Spring.
* Desarrollo:
  + Crear un controlador básico en un proyecto Spring.
  + Implementar endpoints GET que reciban diferentes tipos de parámetros.
* Compartir Resultados:
  + Presentar y discutir tus hallazgos y código con tus compañeros.

**¿Qué es @RestController?**



La anotación **@RestController** en Spring Framework se utiliza para definir una clase como un controlador que maneja solicitudes **HTTP** y devuelve respuestas directamente en formato JSON o XML, **sin necesidad de usar vistas** (como JSP, Thymeleaf, etc.). Es una combinación de las anotaciones **@Controller** y **@ResponseBody**.

**Detalles de la Anotación @RestController**

* **@Controller:** Indica que la clase es un controlador de Spring MVC, que puede manejar solicitudes HTTP.
* **@ResponseBody:** Indica que el valor devuelto por cada método debe ser escrito directamente en la respuesta HTTP (en lugar de renderizar una vista).

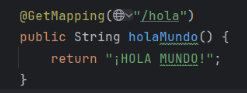
Al usar **@RestController**, no necesitas anotar cada método de la clase con **@ResponseBody** porque esta funcionalidad se aplica automáticamente a todos los métodos.

**¿Qué son las @anotaciones?**

* Investigar y definir
* Desarrolla un ejemplo practico

|  |
| --- |
| En Java, las anotaciones son etiquetas que describe metadata dentro de una función, clase o paquete, el cual es leído durante compilación del programa, un ejemplo de una anotación es @Override, el cual indica que un método esta sobrescribiendo un método de la clase padre, generalmente usado cuando la clase padre es una interfaz o es una clase abstracta.  En Spring Boot se usan para marcar que son y como deberían comportarse ciertas clases métodos o campos dentro del framework de Spring.    En este caso se declara la clase HolaMundoController como RestController dentro de Spring, y su método holaMundoString como un pedido GET mapeado a la solicitud de HTTP “/hola” dentro de la API. |

**¿Qué es @GetMapping?**

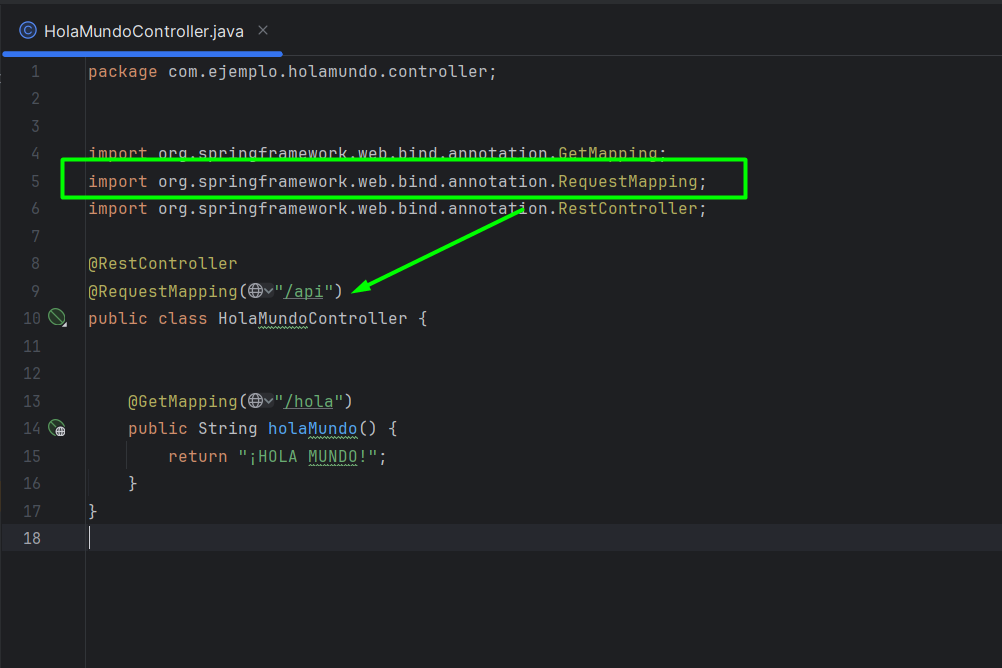


* En Spring Framework, **@GetMapping** es una anotación específica que se utiliza para mapear solicitudes HTTP **GET** a métodos en los controladores. Es una forma especializada y más concisa de usar **@RequestMapping** para manejar solicitudes GET.
* Se usa principalmente para recuperar información del servidor. Las solicitudes GET son **idempotentes**, lo que significa que no deberían tener efectos secundarios en el servidor (por ejemplo, no deberían modificar datos en el servidor).

**Ejercicio práctico**

Ir al proyecto y añadir lo siguiente:

* Importar **RequestMapping**.
* Añadir la anotación de @**RequestMapping**
  + Entre paréntesis agregar “**/api**”



**¿Qué son las @RequestMapping?**

* Investigar y definir
* Desarrolla un ejemplo practico

|  |
| --- |
| Las @RequestMapping y sus variaciones @GetMapping, @PostMapping, @PutMapping y @DeleteMapping se usan para mapear solicitudes de HTTP de una API dentro de métodos y clases.  Por ejemplo una nueva mapeado para una solicitud GET para un adiós seria:  Con el request mapeado a …/api/adios. |

* ¿Qué pasa si cambiamos por **@RequestMapping("/api")** por **@RequestMapping("/app")**?.

|  |
| --- |
| Cambiaria el mapeado dentro de la API de …/api/hola por ejemplo a …/app/hola, la cual no seria una buena practica dentro del uso de una API Rest. |

* Crea una ruta y un método que retorne un mensaje "Hola a todos, mi nombre es **Pedro**”.

|  |
| --- |
| Lo cual nos da |

* Crear una ruta y un método que retorne tu nombre “Hola a todos, mi nombre es \_\_\_\_\_”.
  + Investiga como tomar un campo **nombre** por parámetro en la ruta.
  + Ejemplo:
    - *Ruta: api/hello/****pedro\_hacker***
    - *Retorno: “Hola a todos, mi nombre es pedro\_hacker”*

|  |
| --- |
| Se puede obtener un parámetro dentro de la ruta con la anotación @PathVariable    El cual nos retorna: |

* Crear 4 rutas y métodos que retornen operaciones aritméticas **(suma, resta, multiplicación y división)**
  + Un método por cada operación, de forma interna tienen variables que realizan una operación.
  + Cada método debe retornar un mensaje con el resultado
    - Ejemplo:
      * *api/suma*
      * *api/resta*
      * *api/multiplicacion*
      * *api/division*

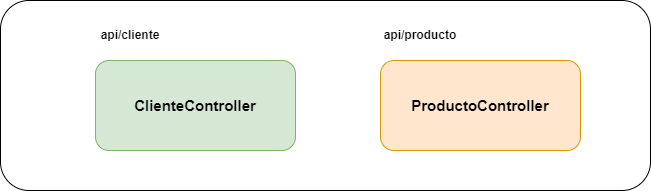
|  |
| --- |
| Metodos:    Los cuales nos dan: |

Crear 4 rutas y métodos que retornen operaciones aritméticas **(suma, resta, multiplicación y división)**

* + Un método por cada operación
  + Debe ingresar dos valores por ruta
    - *api/suma/{****valor\_uno****}/{****valor\_dos****}*
    - **Donde valor\_uno y valor\_dos son un valor ingresado directamente en la ruta.**
  + Cada método debe retornar un mensaje con el resultado
    - Ejemplo:
      * *api/suma/1/2*
      * *api/resta/10/20*
      * *api/multiplicacion/30/10*
      * *Api/division/10/2*

|  |
| --- |
| Los cuales nos dan |

* Crear dos nuevos controladores con sus respectivas rutas.
  + **ClienteController**
  + **ProductoController**



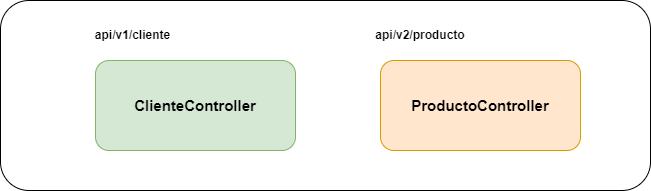
* Añadir una función inicial llamada **index**
  + *Utiliza solo* ***@GetMapping*** *¿Qué pasa si no agregamos paréntesis?*
  + Caso **ClienteController#index** retorna *“bienvenidos al listado de clientes”*
  + Caso **ProductoController#index** retorna *“bienvenidos al listado de productos”*

|  |
| --- |
|  |

* En ProductoController crea una función llamada buscar
  + buscar ***@GetMapping("/{idProducto}")***
  + Crea una lista de productos en Java
  + Recorre los productos esta encontrar el producto indicado por su id 🕵️
    - Si es encontrado ✅: reportan el detalle **producto encontrado**
    - Si no es encontrado ❌ : retorna **error 404**

|  |
| --- |
|  |

* Actualiza los controladores con nuevas rutas de acceso.
  + **ClienteControlle:** abrir en api/v1/cliente
  + **ProductoController:** abrir en api/v2/producto



* Comenta ¿Qué fue lo que pasó?

|  |
| --- |
|  |

💻🦖