Primer paso: descargar e instalar el Postman [Download Postman | Get Started for Free](https://www.postman.com/downloads/)

Luego abrir el proyecto spring boot-webflux. Hacer clic derecho al archivo SpringBootWebfluxApplication y elegir la opción *Debug ‘SpringBootWebf…main()’*

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

A diferencia de otros Ides como Netbeans donde hay que descargar y configurar un servidor, Intellij viene con su propio servidor interno. Una vez que la aplicación está ejecutándose (ya sea en Debug o no) se nos habilitará la dirección web por defecto localhost:8080

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Para obtener las urls de las funciones que expone el webservice al que corresponde el proyecto spring-boot-webflux, se debe ir al paquete Controllers. Dentro hay dos archivos: ProductoController y ProductoRestCongtroller. En cada uno se debe buscar las líneas al inicio donde se declaran las clases:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En esta clase no está declarado un @RequestMapping, de modo que la url será simplemente el dominio donde está el webservice. En este ejemplo es localhost:8080.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En esta clase sí se declara un @RequestMapping, de modo que la url es dominio/cadena\_dentro\_del\_Requestmapping. Para este ejemplo es localhost:8080/api/productos

Hasta este punto, las urls a consumir no están completas. Para saber exactamente las urls a consumir desde el Postman, se debe ir a los métodos expuestos por casa clase, exactamente aquellos que tienen la anotación @GetMapping:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En estos casos, las urls completas con dominio/cadena\_en\_el\_requestmapping/cadena\_getMapping, para este ejemplo la url del método listar es localhost:8080/listar o también localhost:8080 dado que tiene dos cadenas declaradas dentro de @GetMapping. La url a consumir para la función listarDataDriver es localhost:8080/listar-datadriver.

En Spring-boot las funciones GET se declaran con @GetMapping, las funciones POST, PUT y DELETE se declaran con @POSTMAPPING, @PUTMAPPING y @DELETEMAPPING respectivamente.

En la clase ProductRestController hay algunas diferencias:

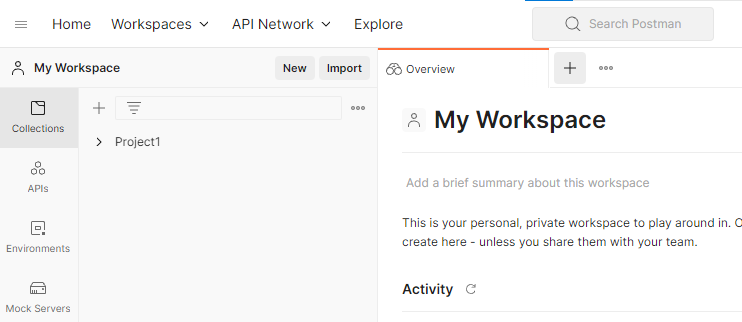
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En el caso de la función index(), la url será localhost:8080/api/productos

En el caso de la función show() la url será localhsot:8080/api/productos/1 donde “1” es el id del producto que deseamos obtener.

A continuación, en el Postman, se crea un WorkSpace y dentro de éste, una Colección:



Para crear un Request, se debe hacer click en el botón con el símbolo +.

Cada request consumirá una función específica en el proyecto spring-boot-webflux. La url correspondiente a cada función se ingresa en la caja de texto de la parte superior derecha:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En donde dice “GET”, se puede elegir el tipo de request (POST, PUT, etc). En este ejemplo, todos los métodos son GET. A continuación se presiona el botón Save y luego Send para enviar la petición.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En la parte inferior debe aparecer la respuesta del servicio web, incluyendo cualquier error o excepción no capturada. Por ejemplo, si se puso mal la url deberá devolver el error 404.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para visualizar mejor los resultados, se recomienda ir a la pestaña “Preview” en lugar de “Pretty”. En otros casos, la visualización es mejor en la pestaña “Pretty”.

En el caso de la función show(), si queremos ver la información del producto con id = 5, la url será: <http://localhost:8080/api/productos/5>

En este caso no devolverá nada porque no hay productos con ids asignados.

Una captura de pantalla de una red social

Descripción generada automáticamente

Para asignar ids a los productos, se deben hacer unos cambios en el proyecto spring-boot-webflux. En la clase Producto se debe añadir una sobrecarga al constructor que reciba un id, esta clase ya tiene una variable id declarada del tipo String:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

A continuación, en la función run() de la clase SpringBootWebfluxApplication se deben asignar algunos ids a los productos declarados:

Texto

Descripción generada automáticamente

Se vuelve a depurar o ejecutar el proyecto spring-boot-webflux y se cambia la url en el request a la función show() a alguno de los ids que hemos asignado:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Con estos cambios, el webservice debe devolver los datos del producto con id = 1:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente