Tabla de Contenidos





El concepto de Spring 5 Reactive es un concepto relativamente nuevo y para el cual necesitamos tener instalado Spring 5 y WebFlux para poder hacer uso de él. La programación Reactiva se apoya fundamentalmente en el concepto de Reactive Streams flujos de trabajo fuertemente asíncronos que permiten una flexibilidad alta a la hora de trabajar con ellos. Vamos a construir unos ejemplos fundamentales con Spring 5 y ThymeLeaf para entender mejor como funciona este tipo de programación asíncrona. El primer paso es ir a Spring initializer y descargarnos un proyecto que incluya las siguientes dependencias:

```
<dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
thymeleaf</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
webflux</artifactId>
                </dependency>
```

```
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
```

Una vez que tenemos las dependencias instaladas el siguiente paso es crearnos con Spring 5 un Repositorio que nos devuelva información . En este caso voy a apostar por un repositorio sencillo en memoria que contenga una lista de productos.

```
package com.arquitecturajava.spring5.reactiva;

public class Producto {

	private int numero;
	private String concepto;
	private int importe;
	public int getNumero() {
		return numero;
	}
	public void setNumero(int numero) {
		this.numero = numero;
	}
	public String getConcepto() {
		return concepto;
	}
	public void setConcepto(String concepto) {
		this.concepto = concepto;
	}
	public int getImporte() {
		return importe;
```

```
	}
	public void setImporte(int importe) {
		this.importe = importe;
	}
	public Producto(int numero, String concepto, int importe) {
		super();
		this.numero = numero;
		this.concepto = concepto;
		this.importe = importe;
	}
	public Producto() {
		super();
	}
}
```

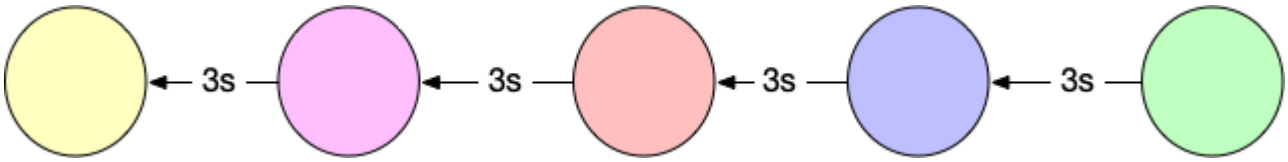
Construida la clase Producto , el siguiente paso es construir la clase Repositorio con una lista de elementos estáticos en memoria.

```
package com.arquitecturajava.spring5.reactiva;

import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import reactor.core.publisher.Flux;

@Repository
public class ProductoRepository {

private static List<Producto> lista= new ArrayList<>();
```

```
        static {
                lista.add(new Producto(1,"ordenador",200));
                lista.add(new Producto(2,"tablet",300));
                lista.add(new Producto(3,"auricular",200));
        }
        public Flux<Producto> buscarTodos() {
                return
Flux.fromIterable(lista).delayElements(Duration.ofSeconds(3));
        }
}
```

# Spring 5 Reactive y Flux

Como se puede observar tenemos un método un poco especial buscarTodos() que devuelve un Flux . Un Flux no es ni mas ni menos que una colección de elementos asíncronos que nos van llegando cada x tiempo. En este caso ese tiempo es 3 segundos. Es decir cada item nos llega después de que han pasado 3 segundos.

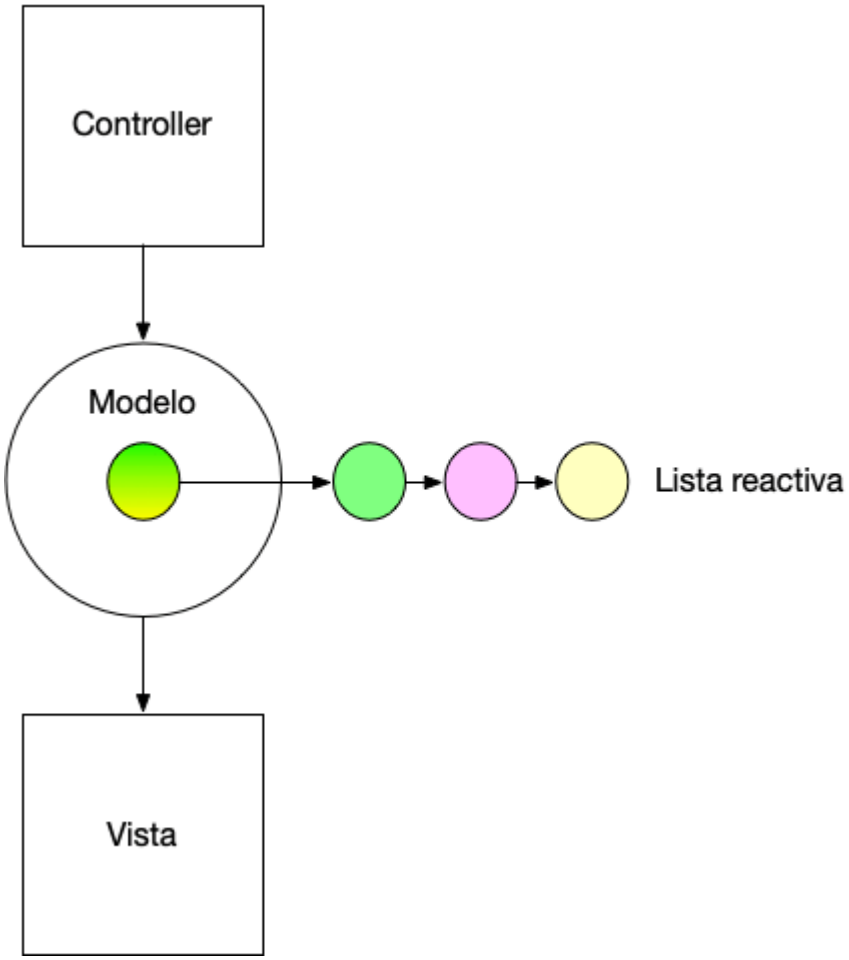



Es momento de definir un Controlador que trabaje con este flujo de elementos asíncronos:

```
package com.arquitecturajava.spring5.reactiva;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import
org.thymeleaf.spring5.context.webflux.IReactiveDataDriverContextVariab
```

```
le;
import
org.thymeleaf.spring5.context.webflux.ReactiveDataDriverContextVariabl
e;

@Controller
public class ControladorReactivo {

        @Autowired
        private ProductoRepository repositorio;
        @RequestMapping("/lista")
        public String lista(Model modelo) {
                        //variable reactiva
                 IReactiveDataDriverContextVariable listaReactiva =
                        new
ReactiveDataDriverContextVariable(repositorio.buscarTodos(), 1);

                modelo.addAttribute("listaProductos", listaReactiva);
                return "lista";
        }
}
```

Acabamos de construir un Controlador con Spring 5 Reactive este controlador solo dispone de una URL de acceso que es /lista . Esta URL es evidentemente una vista de Thymeleaf que recibe como parte de su modelo en vez de un objeto o una lista clasica , recibe un ReactiveDataDriverContextVariable . Es decir una variable Reactiva que tendrá que presentar en el interface de usuario



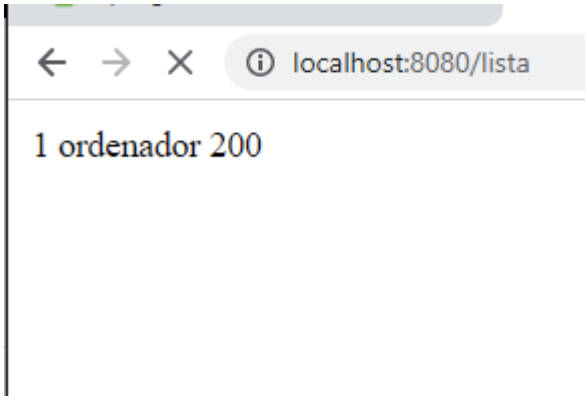

## Thymeleaf y Vistas

Vamos a ver el contenido de la vista que se trata de un html relativamente sencillo:

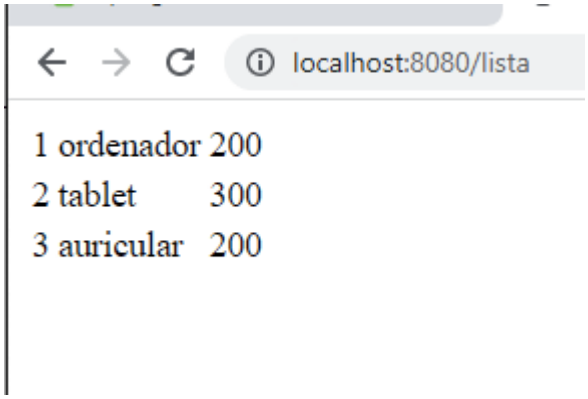
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>


</head>
<body>
```

```
<table>
<tr th:each="producto:${listaProductos}">
<td th:text="${producto.numero}"></td>
<td th:text="${producto.concepto}"></td>
<td th:text="${producto.importe}"></td>

</tr>
</table>

</body>
</html>
```

Si ahora ejecutamos la aplicación nos daremos cuenta que la información nos va llegando poco a poco . Al pasar los primeros 3 segundos nos llegará el primer item:




Pasados 3 segundos nos llegara el segundo elemento y pasados otros 3 el último:




Acabamos de diseñar un ejemplo de Spring 5 Reactive con ThymeLeaf

# Spring 5 Reactive Services (Premium)

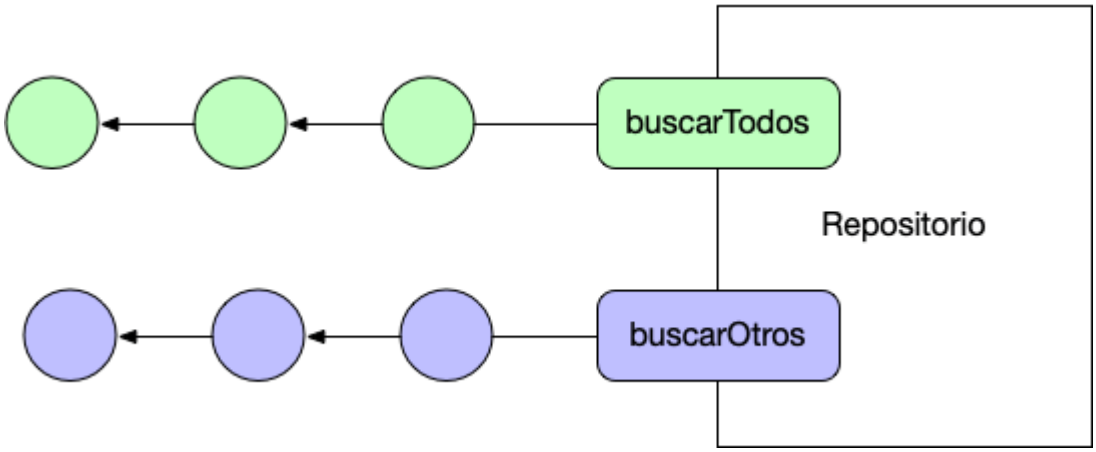
[ihc-hide-content ihc_mb_type="show" ihc_mb_who="4" ihc_mb_template="1" ]

Es momento de avanzar y ver como podemos hacer cosas algo más complejas con Spring 5 y flujos reactiva para entender mejor los conceptos antes expuestos. Para ello vamos a modificar el repositorio para que tenga dos métodos:

```
package com.arquitecturajava.spring5.reactiva;

import java.time.Duration;
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import reactor.core.publisher.Flux;

@Repository
```

```
public class ProductoRepository {

private static List<Producto> lista= new ArrayList<>();
private static List<Producto> lista2= new ArrayList<>();
        static {
                lista.add(new Producto(1,"ordenador",200));
                lista.add(new Producto(2,"tablet",300));
                lista.add(new Producto(3,"auricular",200));
                lista2.add(new Producto(4,"movil",200));
                lista2.add(new Producto(5,"teclado",50));
                lista2.add(new Producto(6,"raton",25));
        }
        public Flux<Producto> buscarTodos() {
                return
Flux.fromIterable(lista).delayElements(Duration.ofSeconds(3));
        }
        public Flux<Producto> buscarOtros() {
                return
Flux.fromIterable(lista2).delayElements(Duration.ofSeconds(3));
        }
}
```

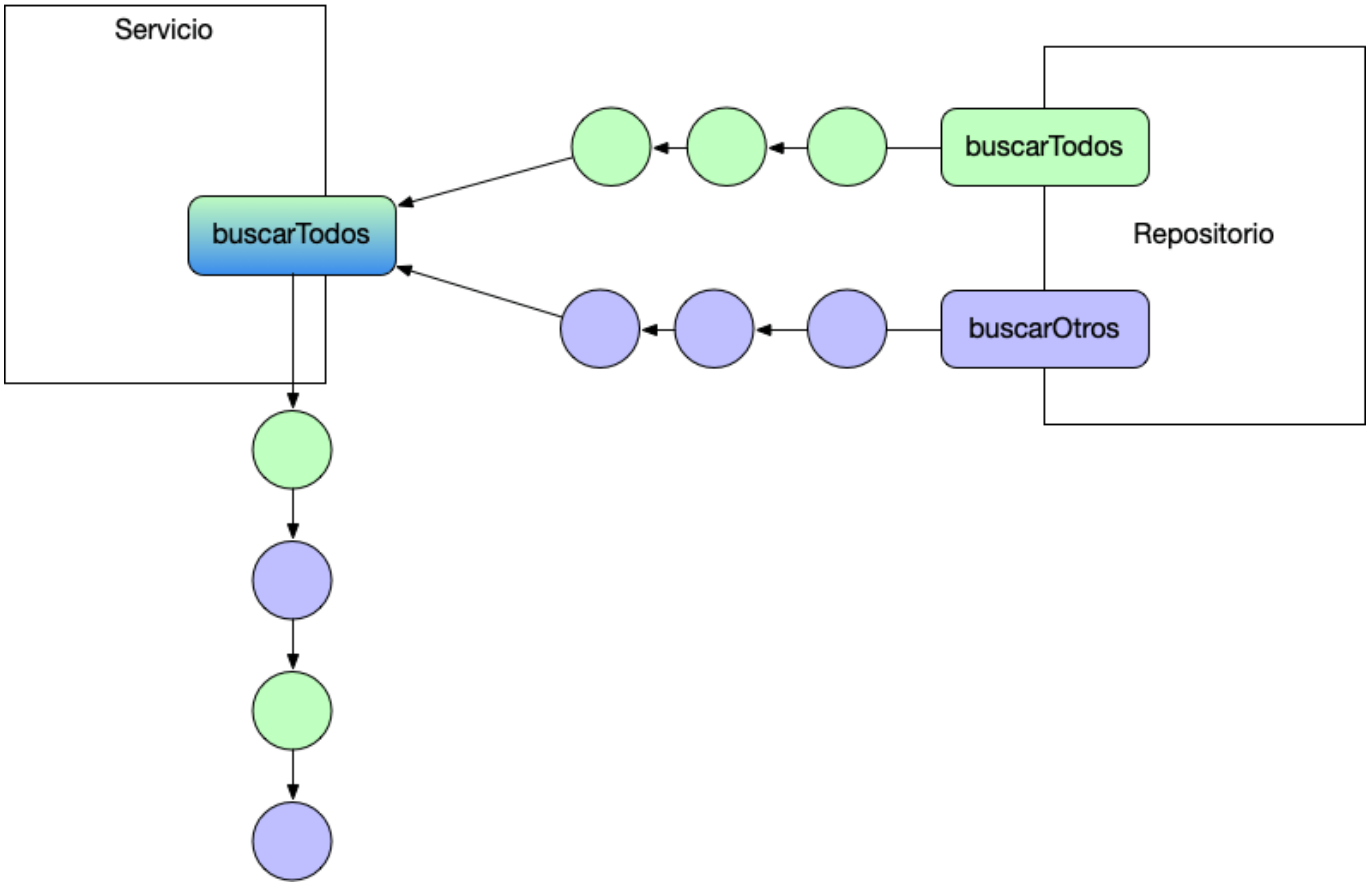
Estos dos métodos cada uno devuelve su propia lista reactiva .




Nosotros queremos obtener una lista reactiva final que podamos manejar en la capa de presentación eso lo haremos a nivel de capa de servicio.

```
package com.arquitecturajava.spring5.reactiva;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import reactor.core.publisher.Flux;

@Service
public class ProductoService {

	@Autowired
	ProductoRepository repositorio;
	public Flux<Producto> buscarTodos() {
		Flux<Producto> flujo1=repositorio.buscarTodos();
		Flux<Producto> flujo2=repositorio.buscarOtros();
```

```
                return Flux.merge(flujo1,flujo2);
        }
}
```



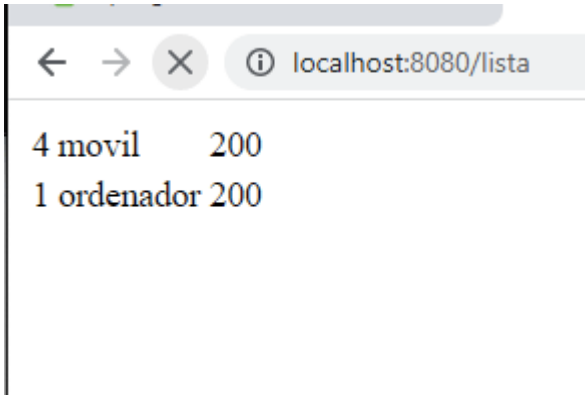

Una vez mezclados ambos flujos con una operación de merge es momento de ver como afecta esto al Controlador

```
package com.arquitecturajava.spring5.reactiva;
```

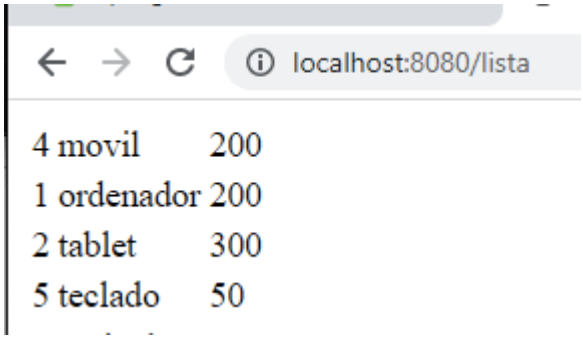
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import
org.thymeleaf.spring5.context.webflux.IReactiveDataDriverContextVariab
le;
import
org.thymeleaf.spring5.context.webflux.ReactiveDataDriverContextVariabl
e;

@Controller
public class ControladorReactivo {

        @Autowired
        private ProductoService servicio;
        @RequestMapping("/lista")
        public String lista(Model modelo) {
                        //variable reactiva
                 IReactiveDataDriverContextVariable listaReactiva =
                        new
ReactiveDataDriverContextVariable(servicio.buscarTodos(), 1);

                modelo.addAttribute("listaProductos", listaReactiva);
                return "lista";
        }
}
```

Como se puede observar prácticamente no hay cambios ya que lo único que cambia el Controlador es el Repositorio por el Servicio. Es momento de ver como en la capa de

presentación estos cambios son reflejados:




Ahora mismo nos podemos dar cuenta como los datos van llegando de los dos Flujos (Flux) de forma simultanea:




Por último los datos totalmente cargados: