Build Reactive MicroServices

using

Spring WebFlux/SpringBoot

Dilip

About Me

- Dilip
- Building Software’s since 2008
- Teaching in **UDEMY** Since 2016

What’s Covered ?

- Introduction to Reactive Programming
- Advantages of Reactive Programming over traditional programming models
- Writing Reactive Programming code using **Project Reactor**
- Introduction **Spring WebFlux**
- Reactive Services using Spring WebFlux
  - **Three Reactive Rest Services** using Spring WebFlux
- JUnit test cases using **JUnit5**

Development > Programming Languages > Java

Reactive Programming in Modern Java using Project Reactor

Learn to write fast performing Asynchronous and NonBlocking code using the Reactive Programming principles and Reactor.

4.6 ★★★★☆ (86 ratings) 636 students

Created by Pragmatic Code School

Last updated 8/2021 English English [Auto]

Preview this course

$99.99

Add to cart

Buy now

30-Day Money-Back Guarantee

What you'll learn

- What is Reactive Programming?
- Write Reactive Code using Project Reactor
- Reactive Streams Specification
- Unit Test the Reactive Code using JUnit5
- When to use Reactive Programming ?
- Different Operators that are part of Project Reactor
- Build Non Blocking Rest Clients using Spring WebClient
- Reactive Types Flux/Mono

Curated for the Udemy Business collection

This course includes:

- 7 hours on-demand video
- 1 article
- 64 downloadable resources
- 1 practice test
- Full lifetime access
- Access on mobile and TV
- Assignments
- Certificate of completion

Apply Coupon Gift this course

Targeted Audience

- Any Developer who is curious to learn about **Reactive Programming**
- Any Developer who is interested in learning **Spring WebFlux**
- Any Developer who is interested in **Building Reactive(Non Blocking) APIs**

Source Code

Thank You!

Why Reactive Programming ?

Evolution of Programming

Past (10 -15) years ago

- Monolith Applications
- Deployed in Application Server
- Does not embrace distributed systems

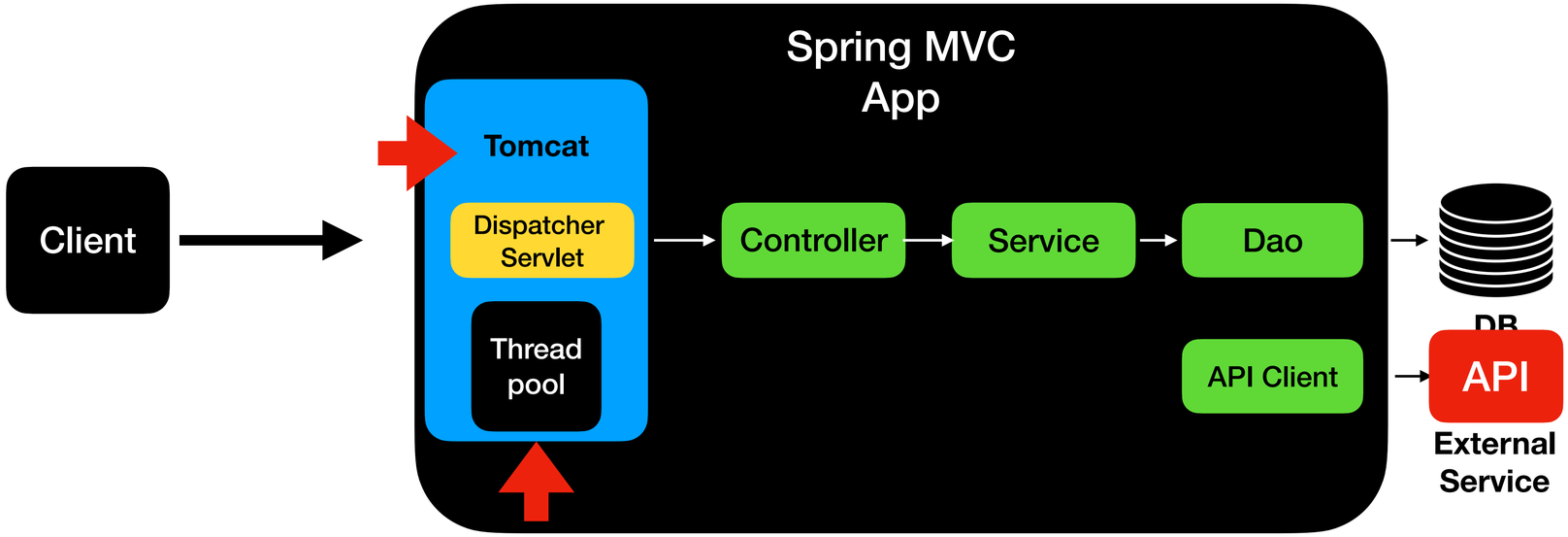
Current

- Microservices Applications
- Deployed in Cloud Environments
- Embrace Distributed Systems

Expectations of the Application

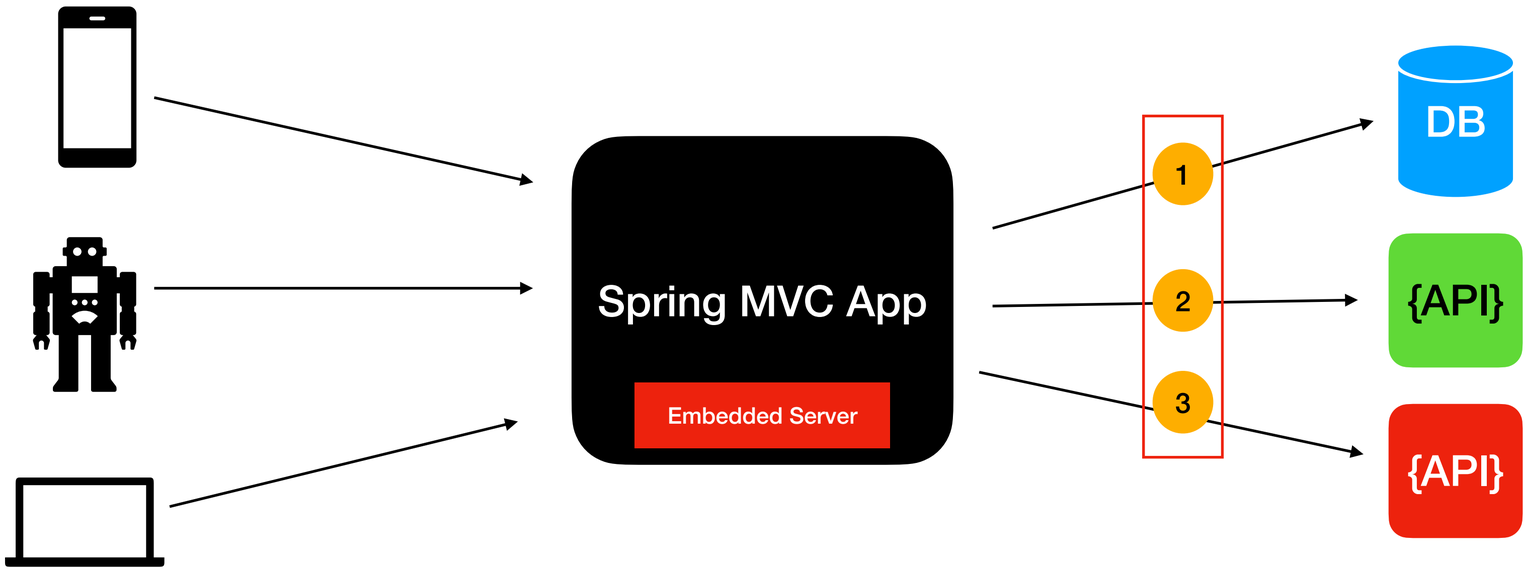
- Response times are expected in milliseconds
- No Downtime is expected
- Scale up automatically based on the load

Restful API using Spring Boot/MVC



- Concurrency is **Thread Per Request** model
- This style of building APIs are called **Blocking APIs**
- Wont scale for today's application needs

Restful API using Spring Boot/MVC



Latency = Summation of (DB + API + API) response times

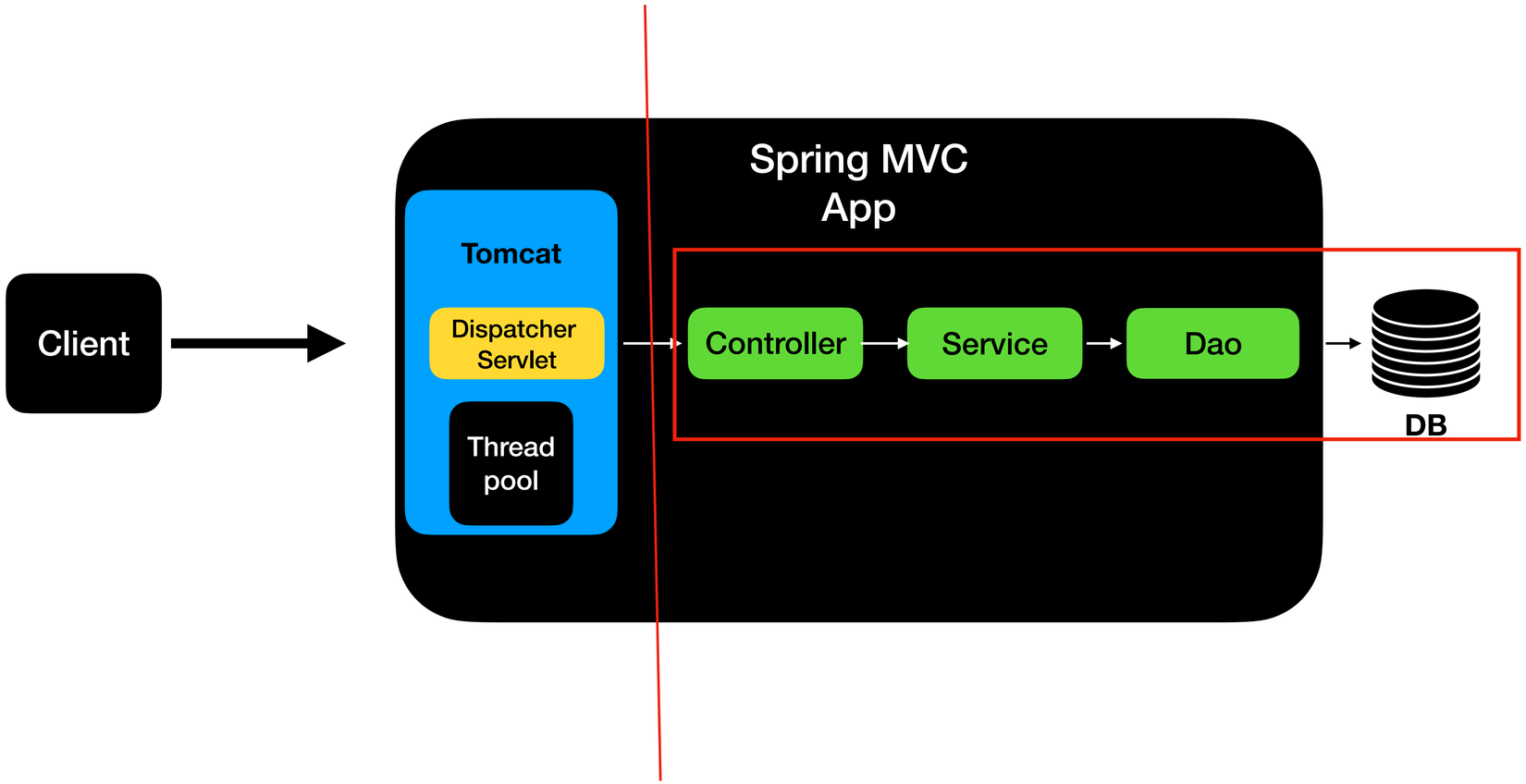
Spring MVC Limitations

- Thread pool size of Embedded tomcat in Spring MVC's is **200**
- Can we increase the thread pool size based on the need ?
  - Yes, only to a certain limit.
- Let's say you have a use case to support **10000** concurrent users.
  - Can we create a thread pool of size **10000** Threads ?
    - No

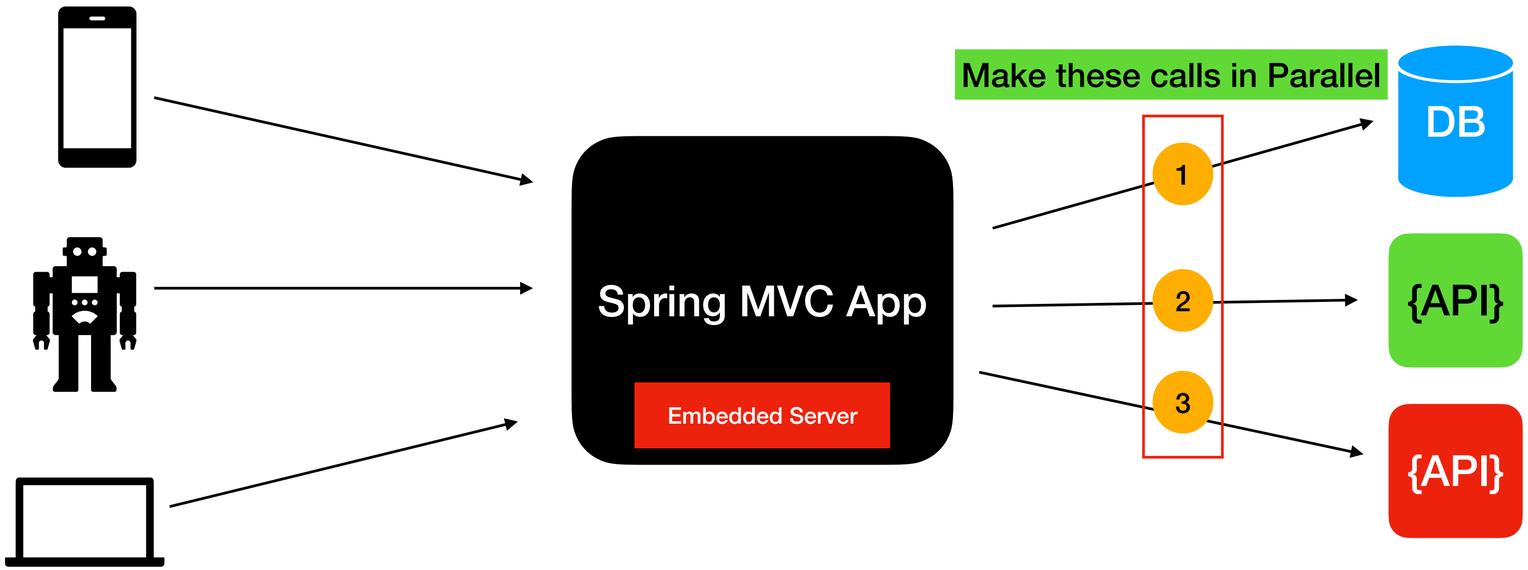
Thread and its Limitations

- Thread is an expensive resource
- It can easily take up to 1MB of heap space
- More threads means more memory consumption by the thread itself
- Less heap space for actually processing the request

Restful API using Spring Boot/MVC



Restful API using Spring Boot/MVC



Latency = Summation of (DB + API + API) response times

Lets explore the asynchrony options in Java

- Callbacks

- Futures

Callbacks

Callbacks

- Asynchronous methods that accept a callback as a parameter and invokes it when the blocking call completes.
- Writing code with Callbacks are hard to compose and difficult to read and maintain
- **Callbackhell**

Future

Concurrency APIs in Java

Future	CompletableFuture

• Released in Java 5
• Write Asynchronous Code
• **Disadvantages:**
• No easy way to combine the result from multiple futures
• Future.get()
  • This is a blocking call
	• Released in Java8
	• Write Asynchronous code in a functional style
	• Easy to compose/combine MultipleFutures
	• **Disadvantages**:
	  • Future that returns many elements
	  • Eg., CompletableFuture<List<Result> will need to wait for the whole collection to built and readily available
	  • CompletableFuture does not have a handle for infinite values

Drawbacks of Spring MVC

- Concurrency is limited in Spring MVC
- Blocking code leads to inefficient usage of threads.
- Servlet API at the server level is a blocking one

Reactive programming to the rescue

Is there a better option available?

Summary

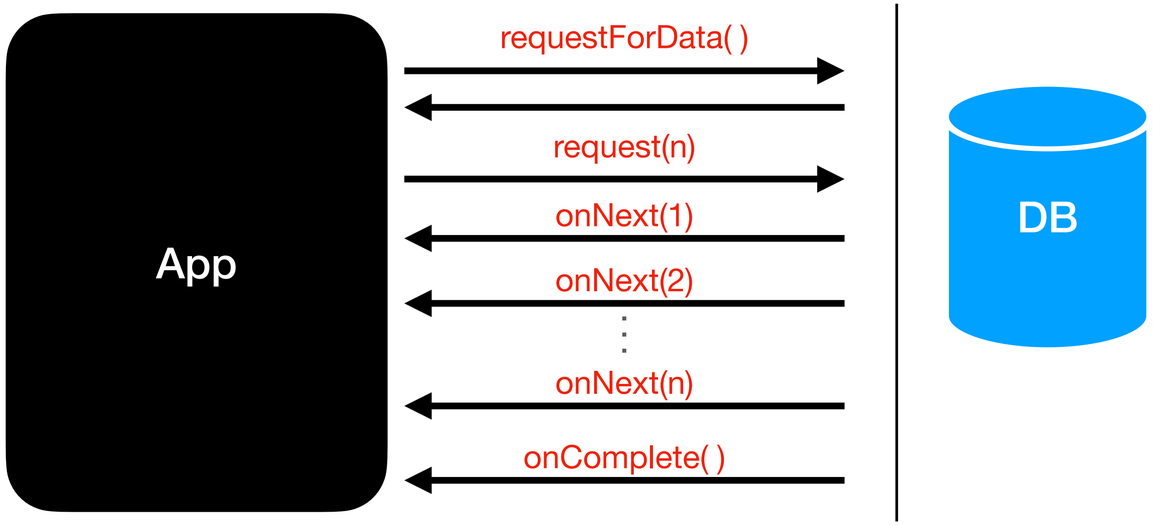
- Does this mean we should stop using Spring MVC?
  - No
  - This still works very well for many use cases

What is Reactive Programming ?

What is Reactive Programming ?

- Reactive Programming is a new programming paradigm
- Asynchronous and non blocking
- Data flows as an **Event/Message** driven stream

Reactive Programming



- This is not a blocking call anymore

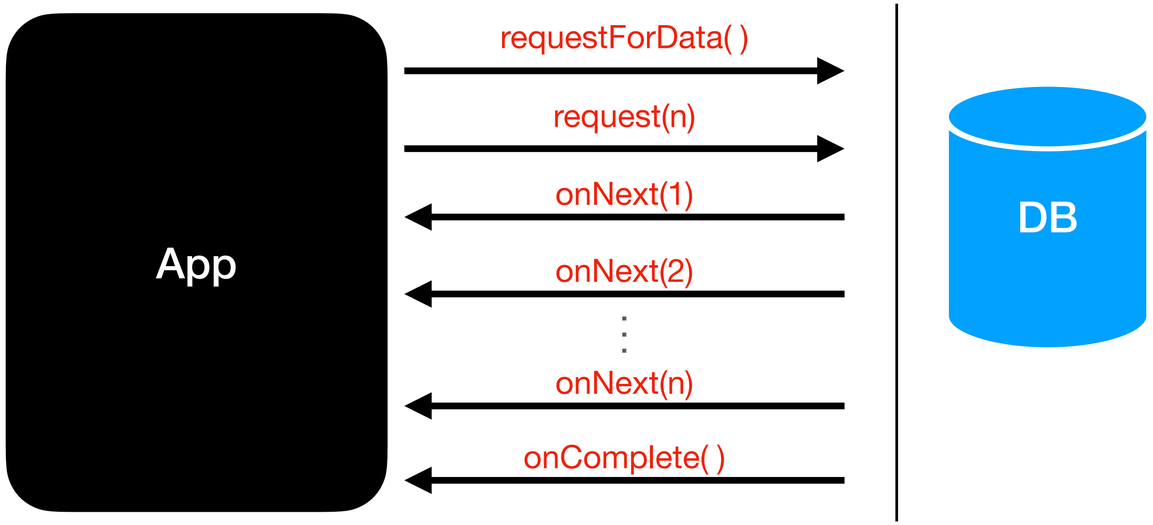
Push Based data streams model

- Calling thread is released to do useful work

What is Reactive Programming ?

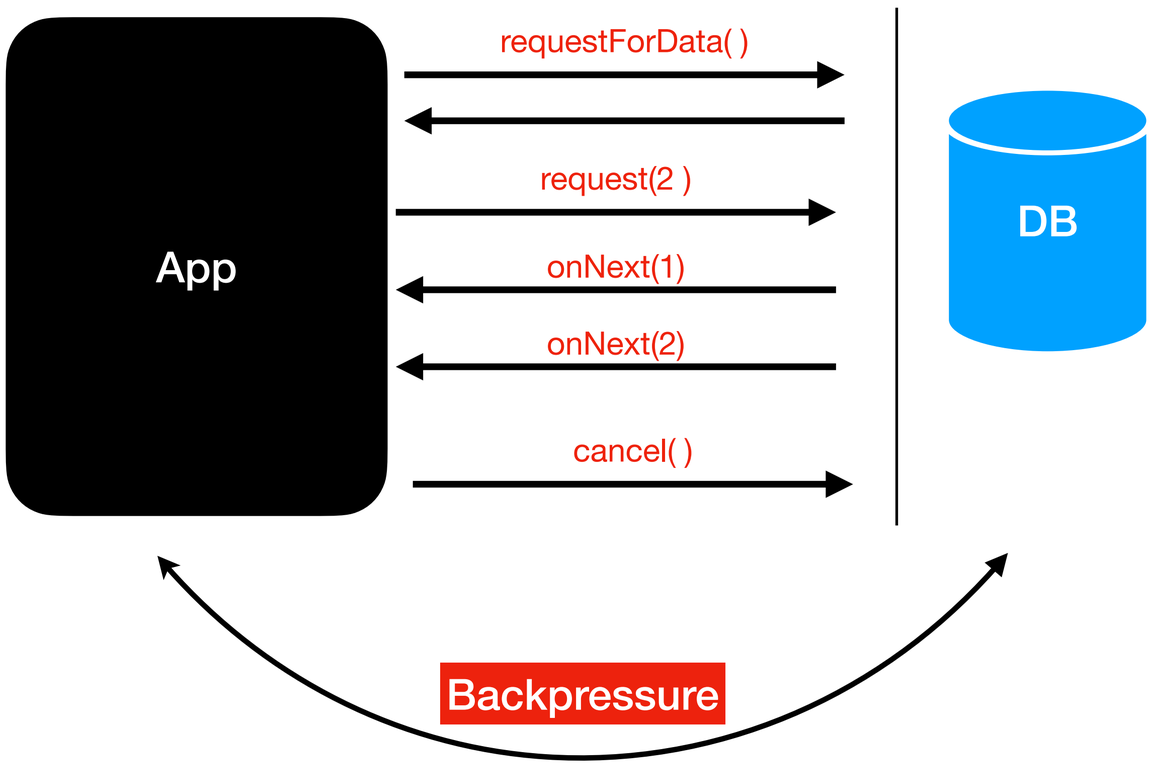
- Reactive Programming is a new programming paradigm
- Asynchronous and non blocking
- Data flows as an **Event/Message** driven stream
- Functional Style Code
- BackPressure on Data Streams

Backpressure



Overwhelm the app with more data

Backpressure



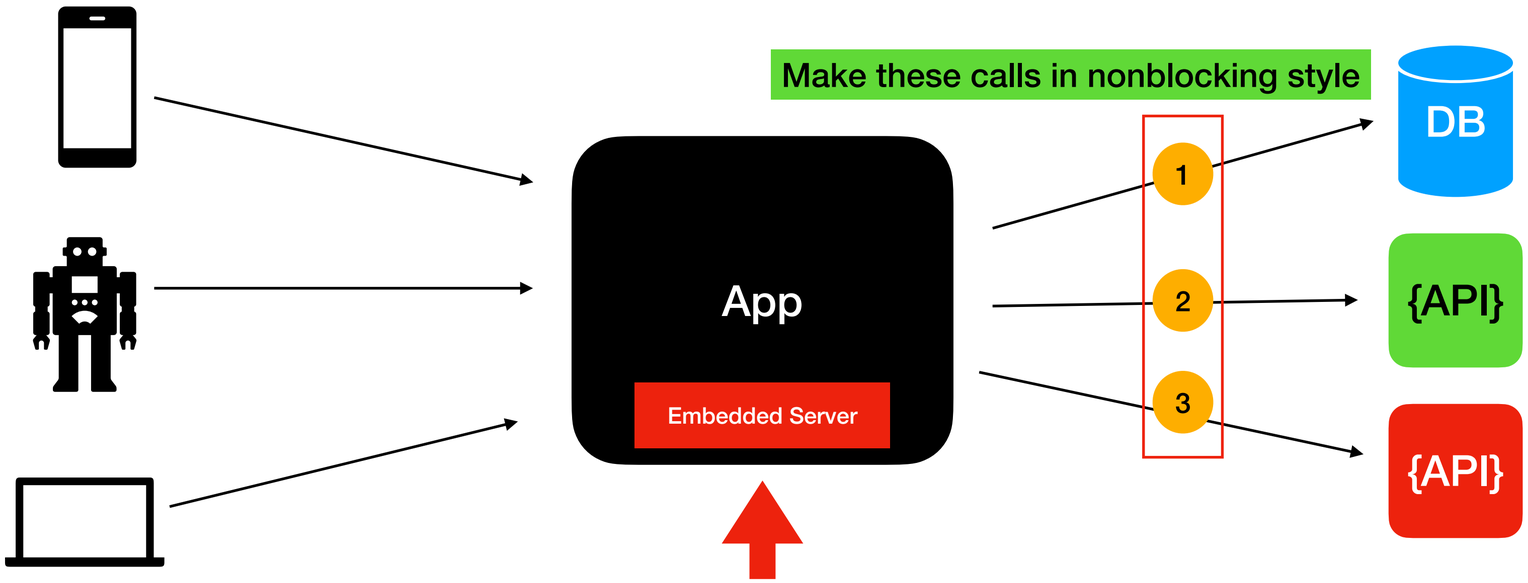
Push-based data flow model

Push-Pull based data flow model

When to use Reactive Programming ?

Use Reactive Programming when there is need to build and support app that can handle high load

Reactive App Architecture



- Handle request using non blocking style
  - Netty is a non blocking Server uses Event Loop Model
- Using Project Reactor for writing non blocking code
- Spring WebFlux uses the Netty and Project Reactor for building non blocking or reactive APIs

Reactive Streams

Reactive Streams are the foundation for Reactive programming.

Reactive Streams

- Reactive Streams Specification is created by engineers from multiple organizations:
  - Lightbend
  - Netflix
  - VmWare (Pivotal)

Reactive Streams Specification

- Reactive Streams Specification:
  - Publisher
  - Subscriber
  - Subscription
  - Processor

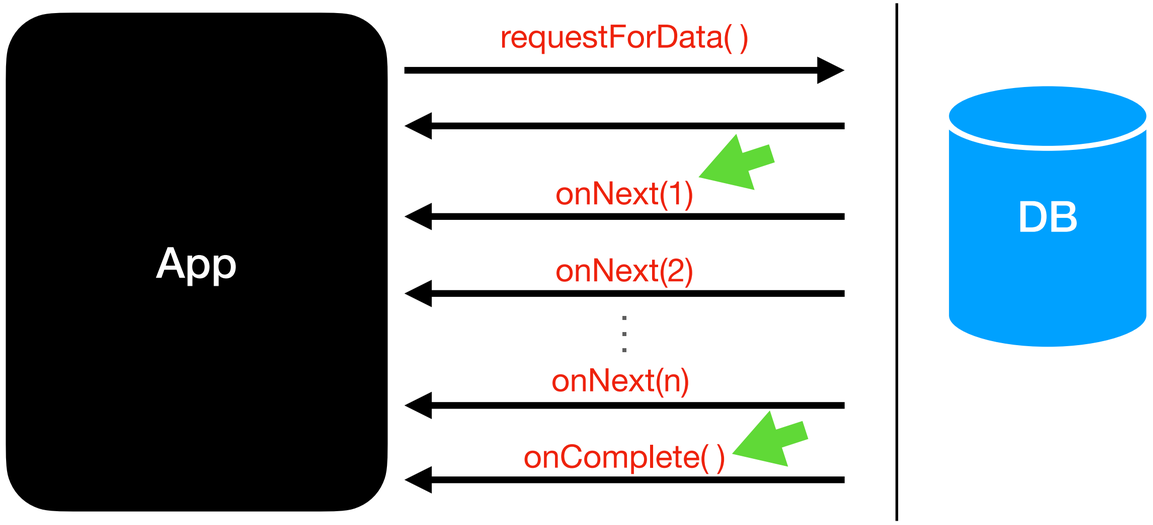
Publisher

```
public interface Publisher<T> {

    public void subscribe(Subscriber<? super T> s);  1
}
```

- Publisher represents the DataSource
  - Database
  - RemoteService etc.,

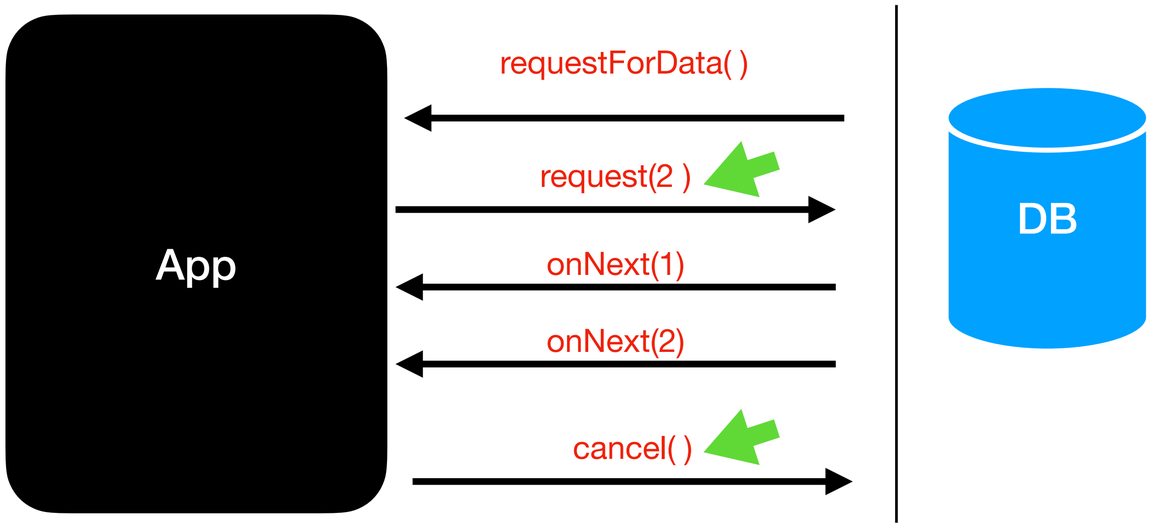
Subscriber

```
public interface Subscriber<T> {

    public void onSubscribe(Subscription s);   1

    public void onNext(T t);   2

    public void onError(Throwable t);   3

    public void onComplete();   4
}
```



Subscription

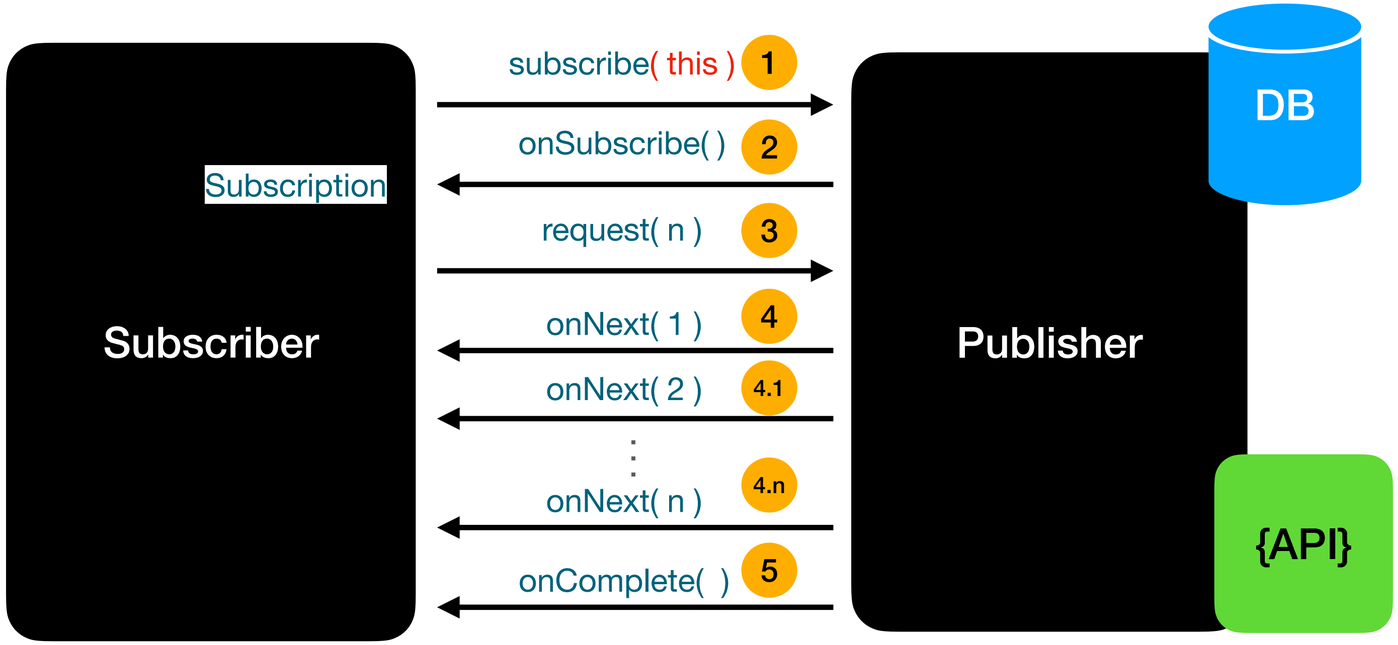
```
public interface Subscription {

    public void request(long n);

    public void cancel();
}
```



Subscription is the one which connects the app and datasource

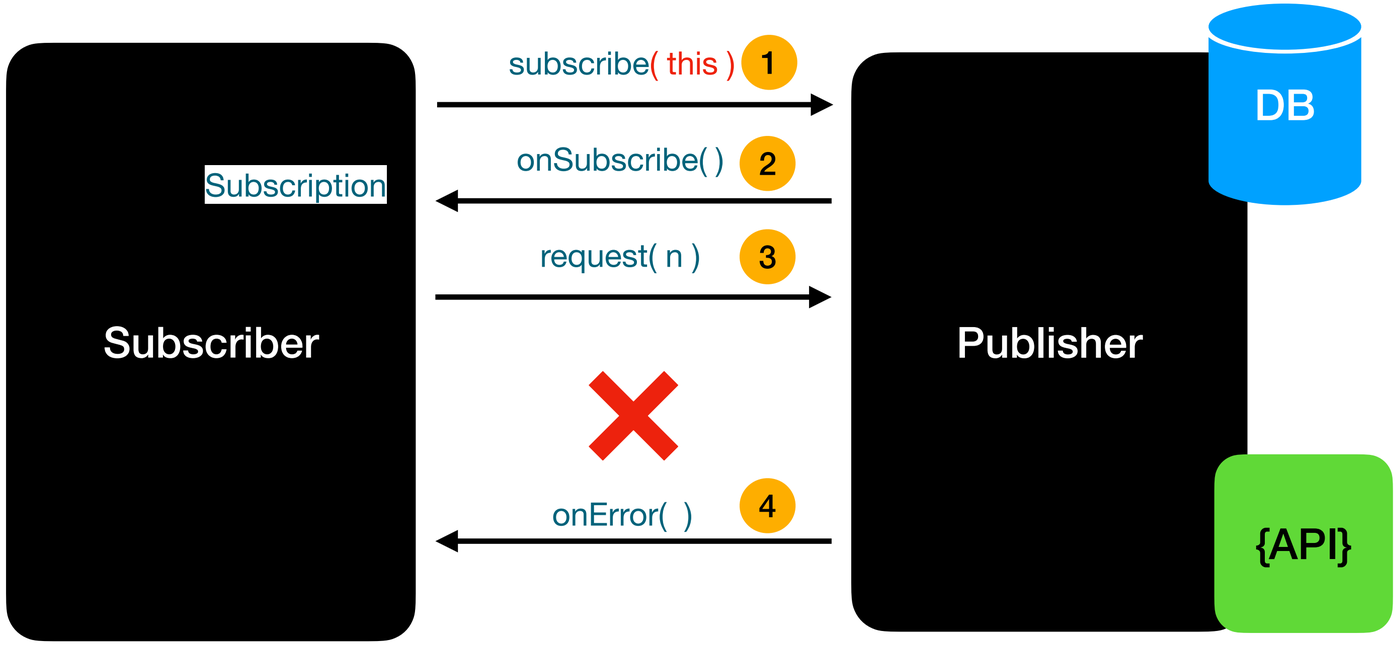
Reactive Streams - How it works together ?

Success Scenario



Reactive Streams - How it works together ?

Error/Exception Scenario



- Exceptions are treated like the data
- The Reactive Stream is dead when an exception is thrown

Processor

```
public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {

}
```

- Processor extends Subscriber and Publisher
  - Processor can behave as a Subscriber and Publisher
  - Not really used this on a day to day basis

Flow API

- Release as part of Java 9
- This holds the contract for reactive streams
- No implementation of the Reactive Streams is available as part of JRE

What is a
Nonblocking or Reactive
RestFul API ?

NonBlocking or Reactive RestFul API

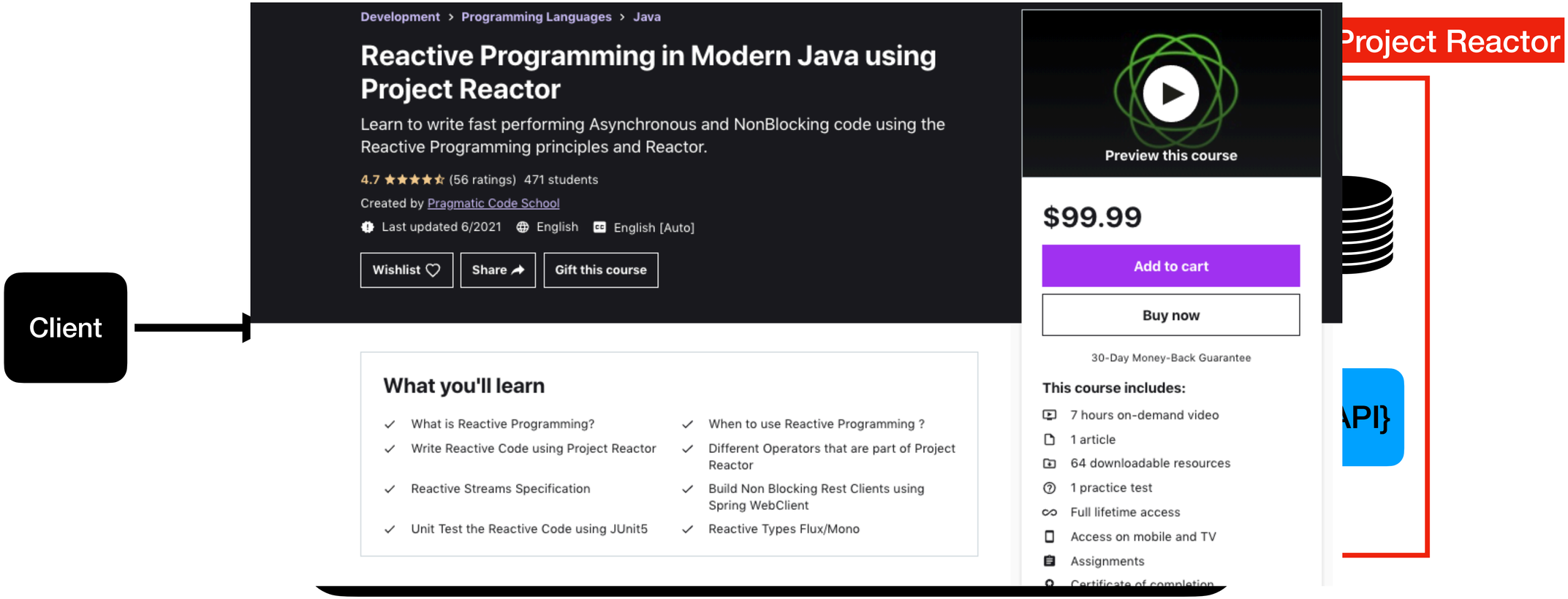
- A Non-Blocking or Reactive RestFul API has the behavior of providing end to end non-blocking communication between the client and service

- Non-Blocking or Reactive == Not Blocking the thread

- Thread involved in handling the **httprequest** and **httpresponse** is not blocked at all

- **Spring WebFlux** is a module that's going to help us in achieving the **Non-Blocking** or **Reactive** behavior

NonBlocking or Reactive API using Spring WebFlux



Options for Reactive RestFul API Using Spring WebFlux

Setting up the Project For this Course

Section Overview

Section Overview

- Explo
-

Development > Programming Languages > Java

Reactive Programming in Modern Java using Project Reactor

Learn to write fast performing Asynchronous and NonBlocking code using the Reactive Programming principles and Reactor.

4.7 ★★★★☆ (56 ratings) 471 students

Created by Pragmatic Code School

Last updated 6/2021 English English [Auto]

Wishlist Share Gift this course

Preview this course

$99.99

Add to cart

Buy now

30-Day Money-Back Guarantee

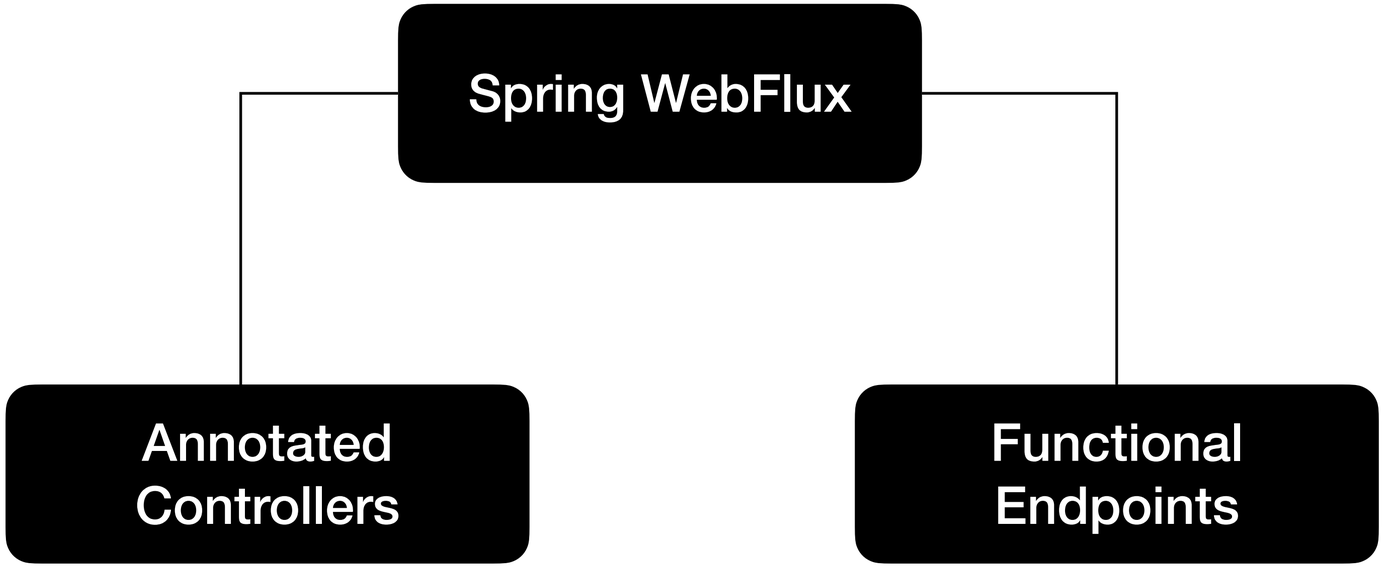
This course includes:

- 7 hours on-demand video
- 1 article
- 64 downloadable resources
- 1 practice test
- Full lifetime access
- Access on mobile and TV
- Assignments
- Certificate of completion

What you'll learn

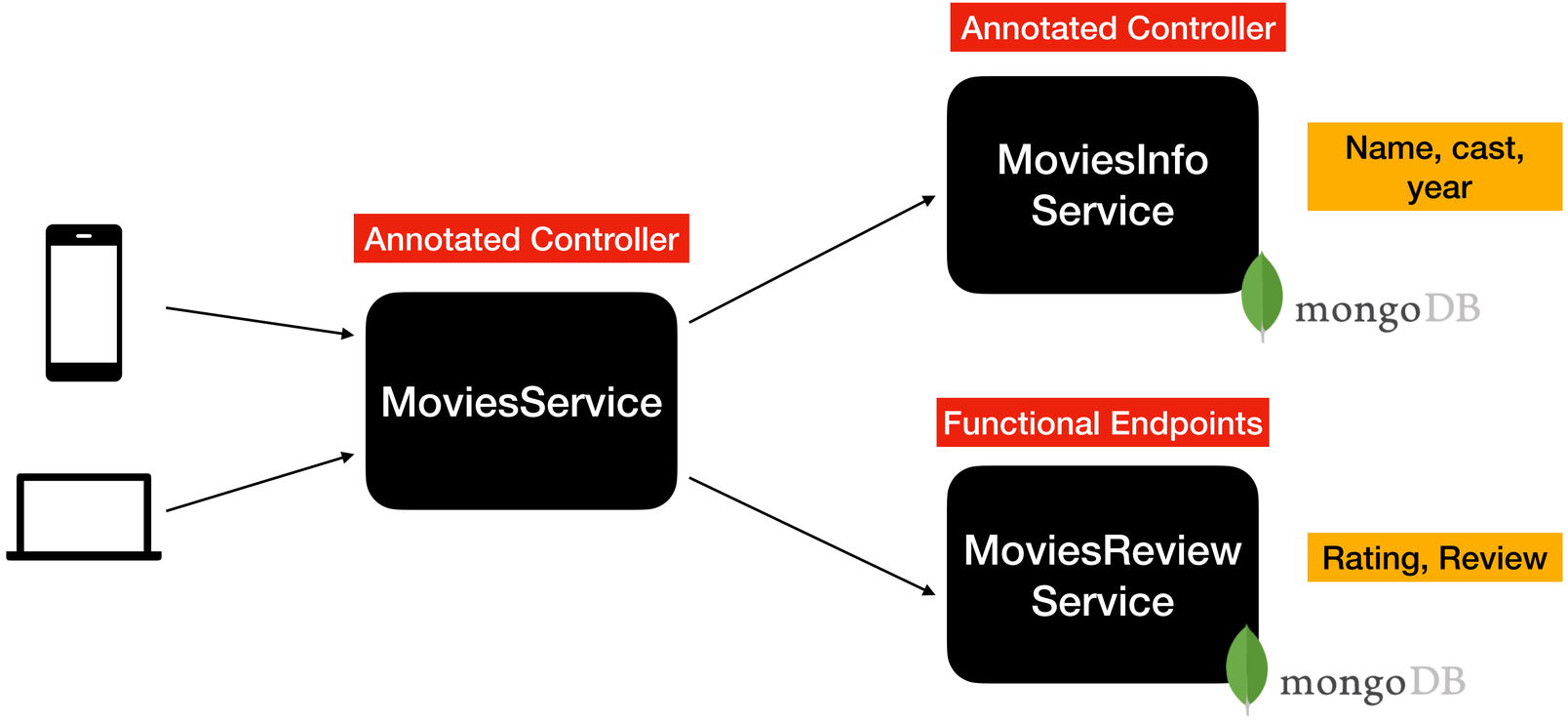
- ✓ What is Reactive Programming?
- ✓ Write Reactive Code using Project Reactor
- ✓ Reactive Streams Specification
- ✓ Unit Test the Reactive Code using JUnit5
- ✓ When to use Reactive Programming ?
- ✓ Different Operators that are part of Project Reactor
- ✓ Build Non Blocking Rest Clients using Spring WebClient
- ✓ Reactive Types Flux/Mono

Spring WebFlux

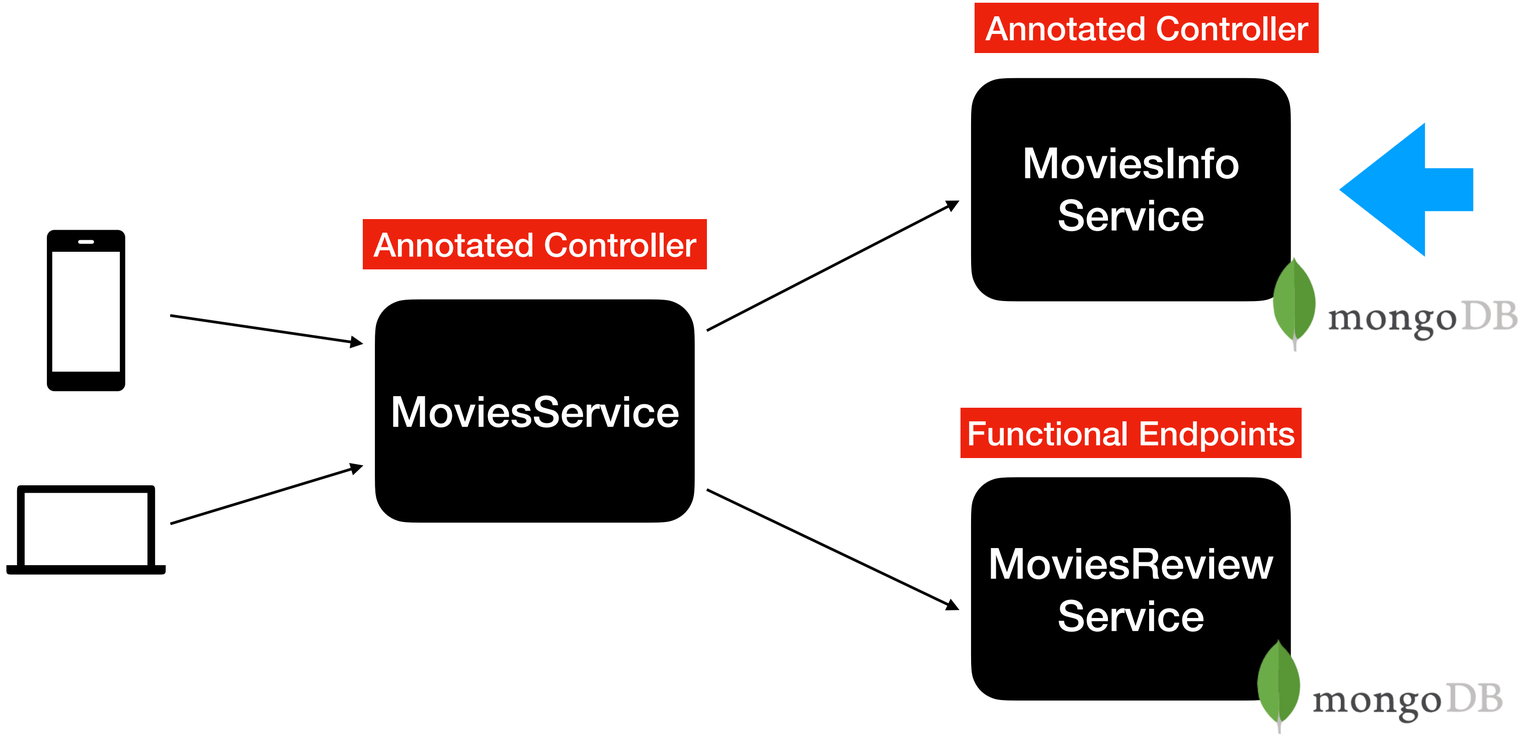


What are we going to build in this course ?

Movies Application using MicroServices Pattern



Movies Application using MicroServices Pattern



Streaming Endpoint Using Spring Webflux

Streaming Endpoint

- Streaming Endpoint is a kind of Endpoint which continuously sends updates to the clients as the new data arrives
- This concept is similar to Server Sent Events(SSE)
- Easy to implement in Spring WebFlux
- **Examples** : Stock Tickers, Realtime updates of Sports Events

Automated Testing Using JUnit5

Automated Tests

- Automated Tests plays a vital role in delivering quality Software

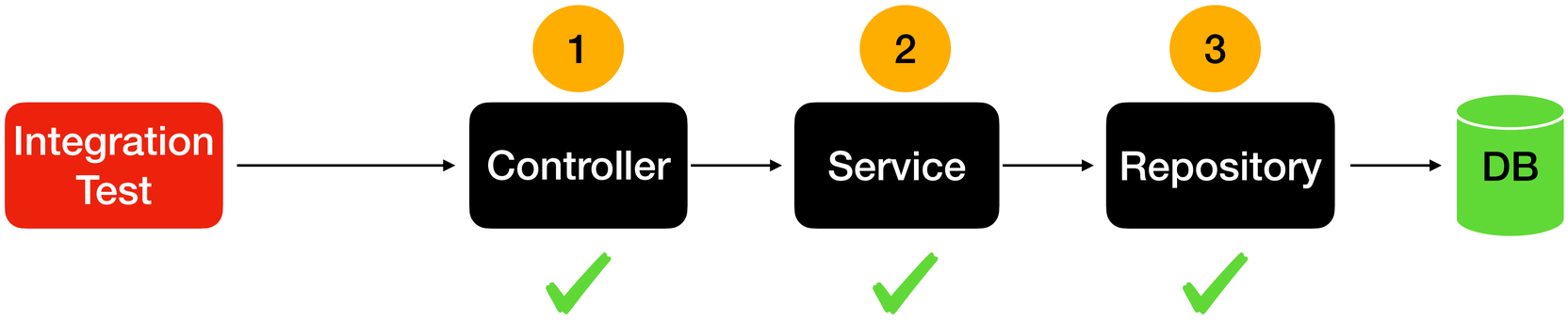
- Two types of Automated Tests:

  - Integration Tests

  - Unit Tests

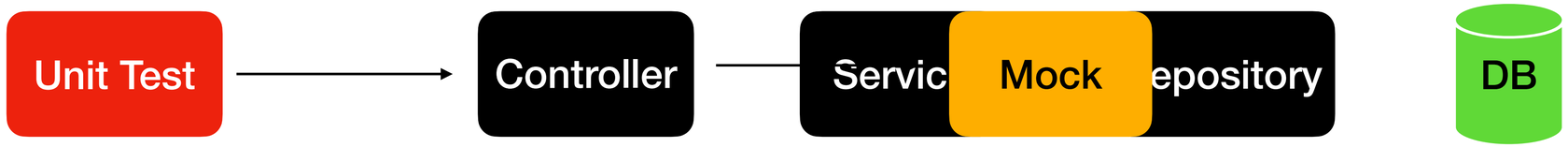
Integration Tests

- Integration test is a kind of test which actually test the application end to end

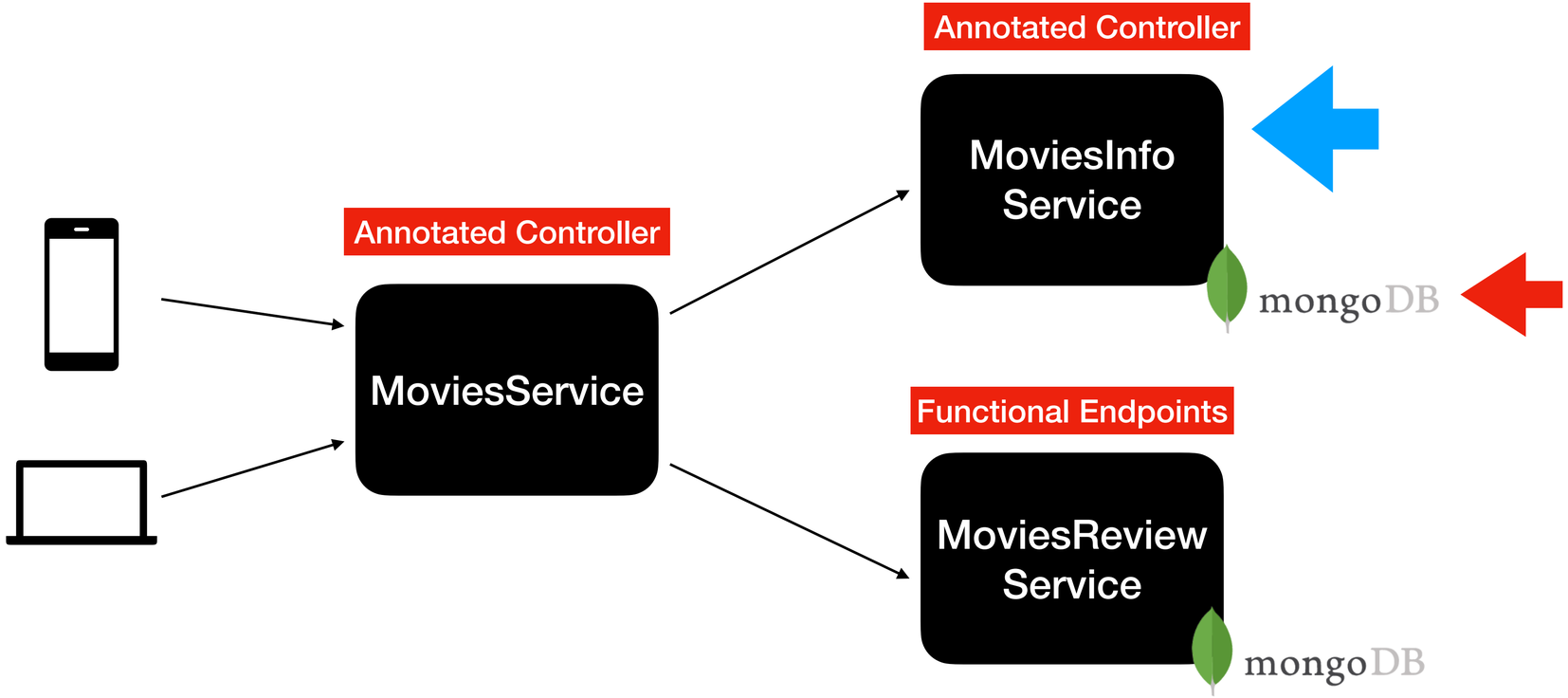


Unit Tests

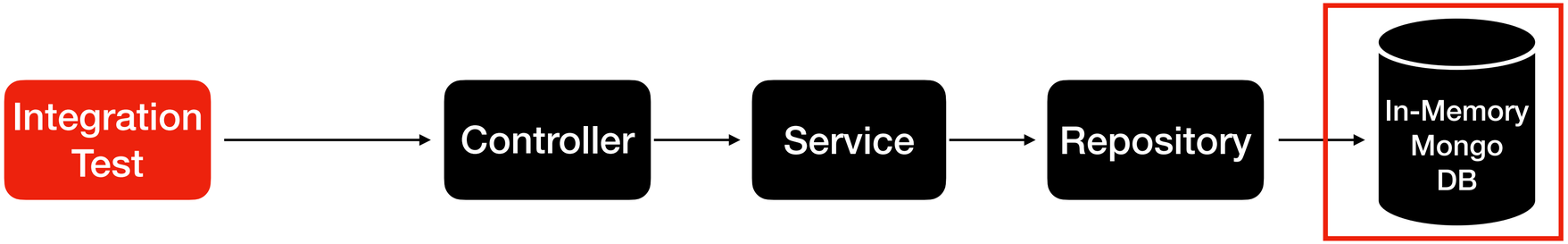
- Unit test is a kind of test which tests only the class and method of interest and mocks the next layer of the code



Movies Application using MicroServices Pattern



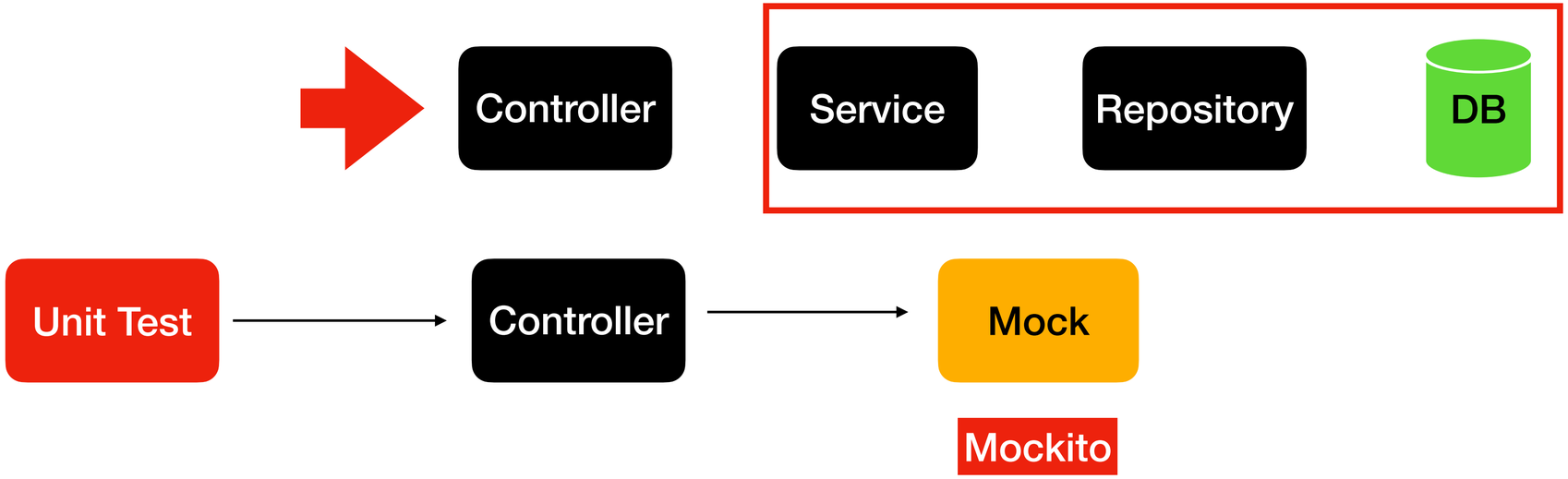
Integration Tests using Embedded MongoDB




```
testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'
```

Unit Tests

- Unit test is a kind of test which tests only the class and method of interest and mocks the next layer of the code



Benefits of Unit Tests

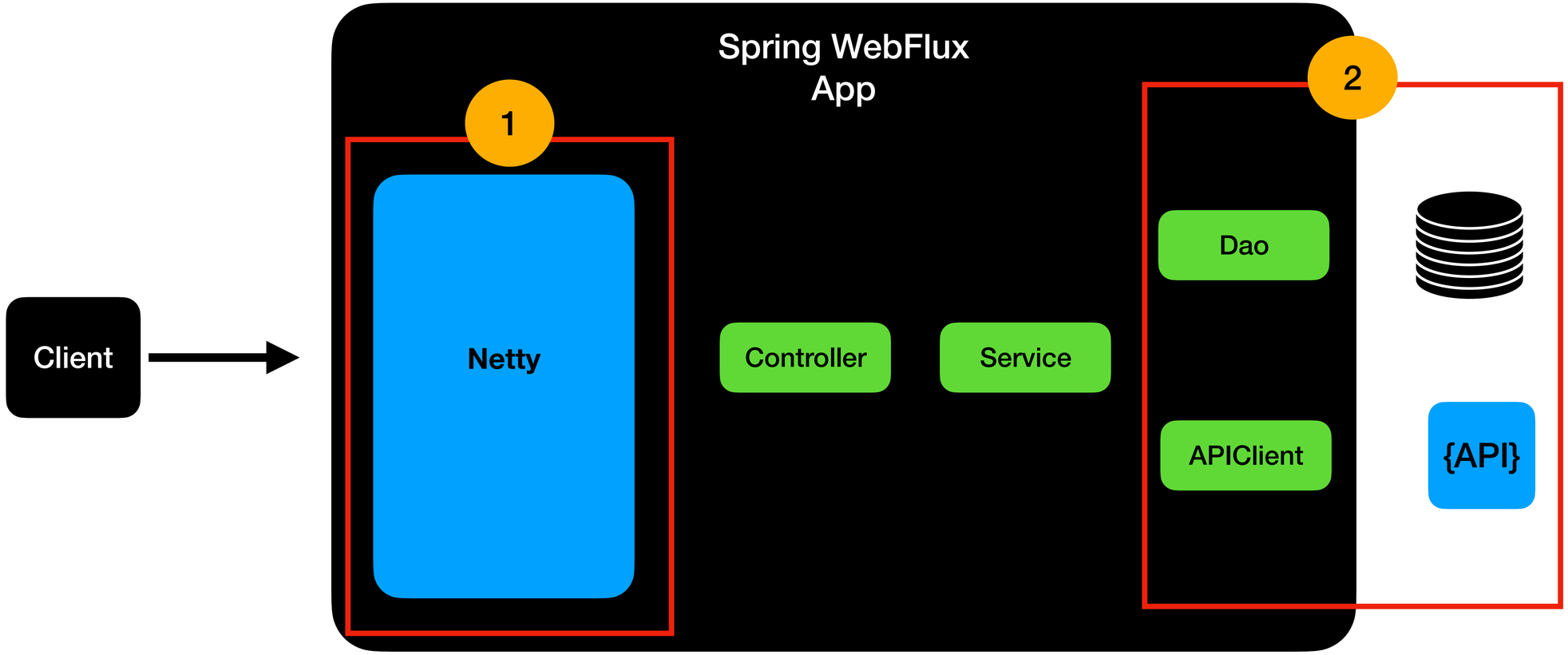
- Unit Tests are faster compared to Integration Tests
- Unit Tests are handy for performing Bean Validations

Spring WebFlux Test has an annotation named “@WebFlux” test

Using ResponseEntity In Spring WebFlux

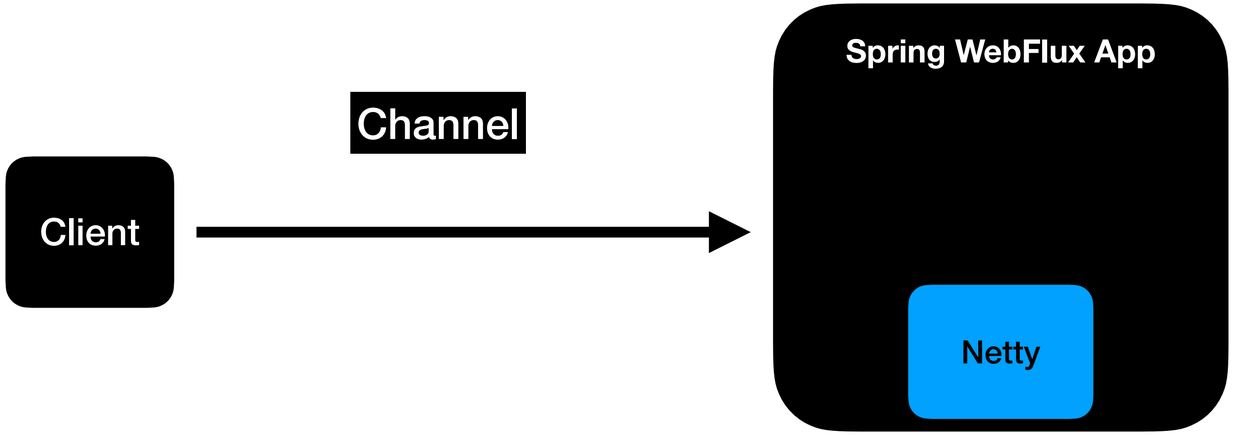
How Netty Works with Spring WebFlux ?

NonBlocking or Reactive API using Spring WebFlux



How does
Netty
handle the request?

Netty (WebFlux's Default Server)



- Channel represents an open connection between the client and server
- Request and Response is sent via the channel

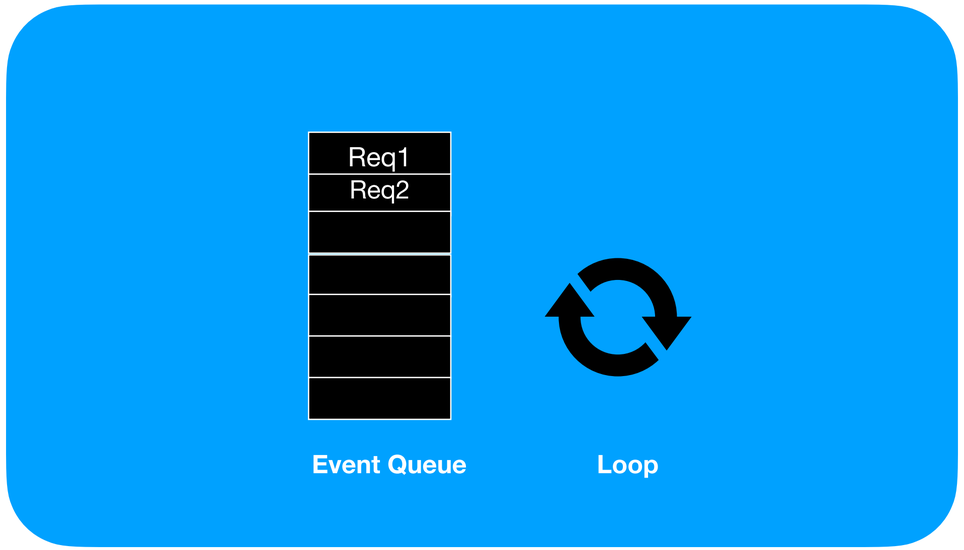
Channel

- Channel has ChannelHandlers
  - Accepting the client connection
  - Reading the data as bytes from the network to a Java Object(Transformation)
  - Writing the data back to the client
- This is all taken care for us by Spring WebFlux
- As a developer, we just focus on writing the application related code

Channel and EventLoop

- Netty , uses **EventLoop model** to handle the connections in a nonblocking fashion
- An EventLoop is powered by one single thread
  - NodeJs uses the same pattern.
  - Node js has just one thread/one eventloop to handle client requests
- Number of Eventloops to handle the request is equal to no of cores in your machine
  - **Eventloops** are part of the EventLoopGroup

EventLoop



How Channel and EventLoop linked ?

- Any time a channel is created it gets assigned to an EventLoop

- This **EventLoop** is responsible for handling the different events that occurs in the lifetime of a channel

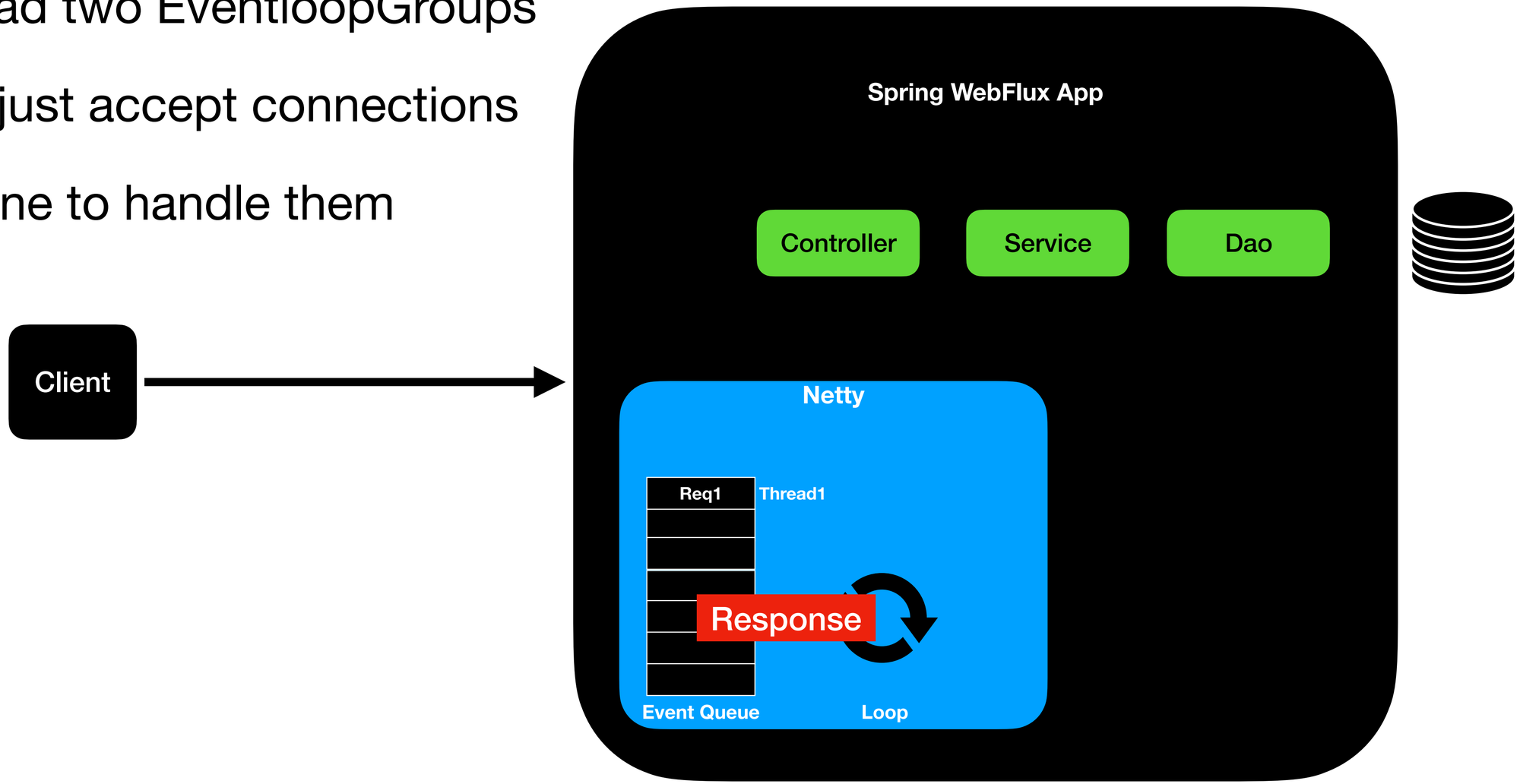
Channel Lifecycle

1. **ChannelUnregistered** — Channel is Created and its not registered with the Eventloop
2. **ChannelRegistered** — Channel is registered with the Eventloop
3. **ChannelActive** — Channel is active and its now possible to send and receive the data
4. **ChannelInActive** — Channel is not connected to the client anymore and ready to be closed

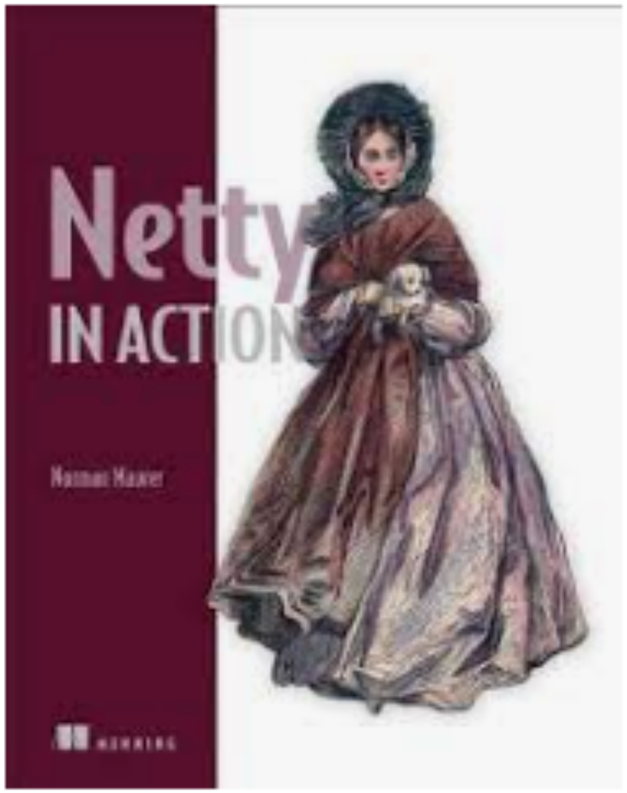
All these Lifecycle changes are treated as events

How Netty handles the request ?

- Netty had two EventloopGroups
- One to just accept connections
- Other one to handle them



Netty in Action



Functional Web
In
Spring WebFlux

Functional Web

- This is an alternative programming model for building RESTFUL APIs in Spring WebFlux
- Functional web module uses the functional programming aspects:
  - Lambdas
  - Method References
  - Functional Interfaces

Functional Web

Rest Endpoints are configured

Code to handle the request is located

Router

Handler

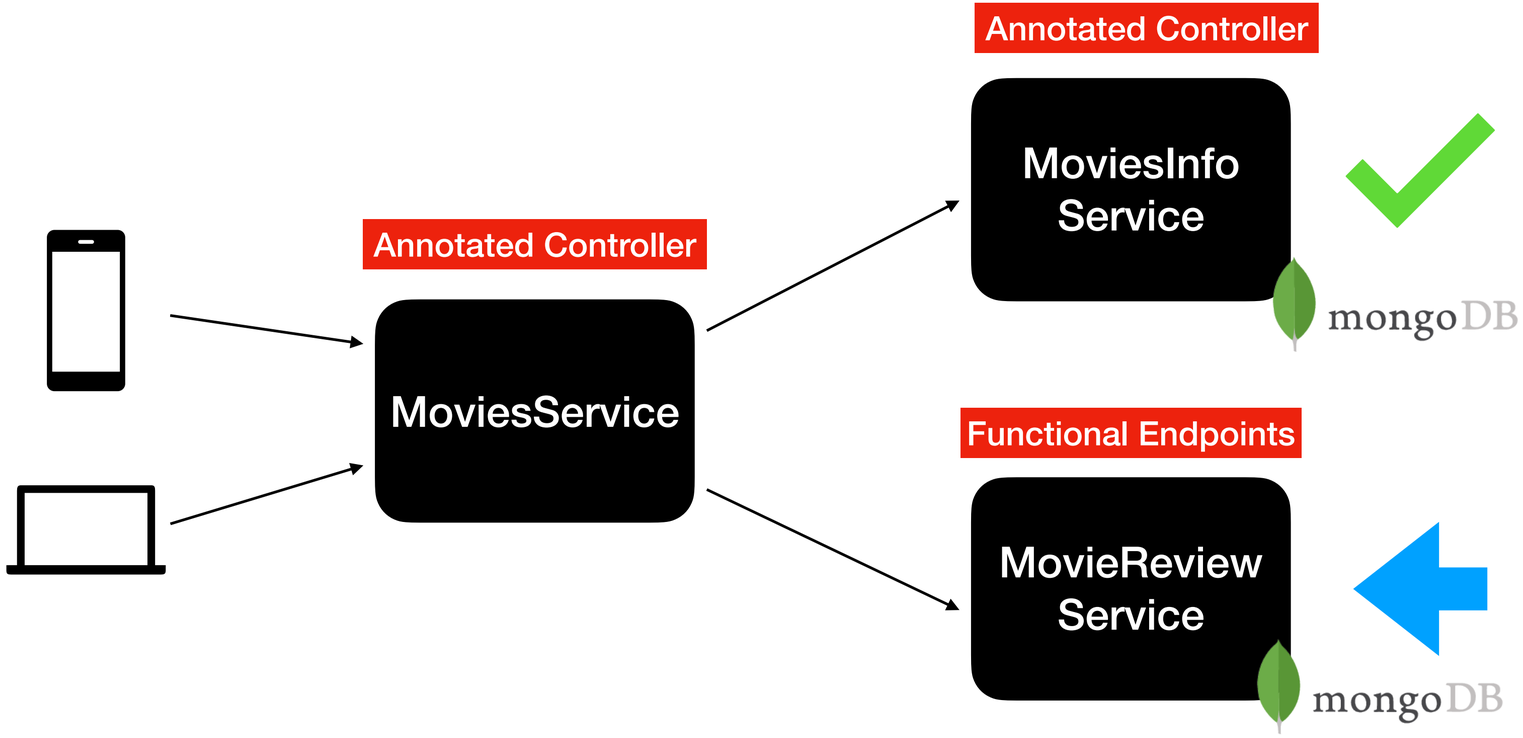
```
@GetMapping( ...value: "/movie_infos/{id}")
fun movieInfoById(@PathVariable( value: "id") movieInfoId: Long): Mono<MovieInfo> {
    return movieInfoService.getMovieById(movieInfoId)
        .doOnNext {  it: MovieInfo!
            logger.info { "Movie is $it" }
        }
}
```

Is there an advantage in building RestFul APIs using Functional Web?

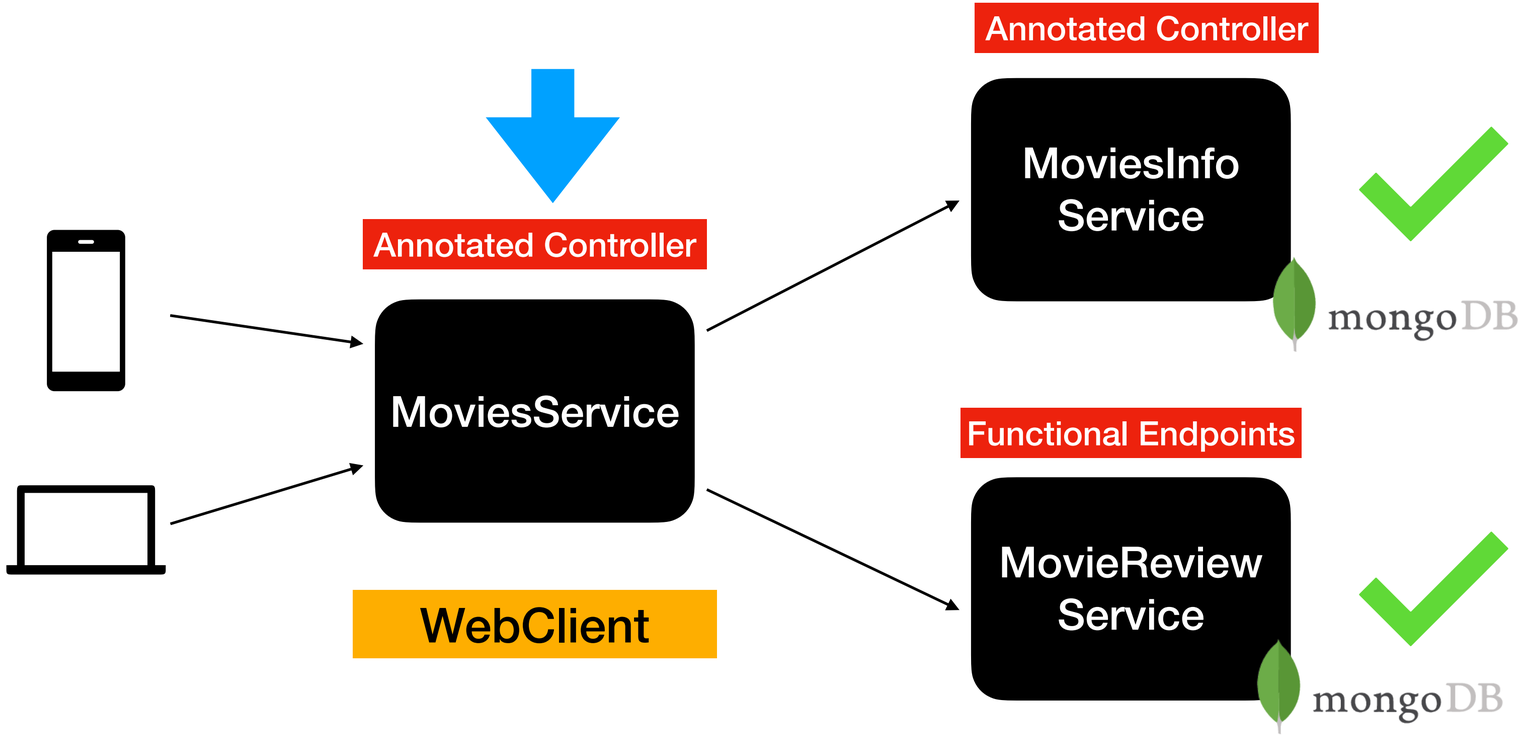
Functional Web

- Benefits:
  - All the RestFul APIs endpoints are configured in one single file
  - Code is lightweight compared to the Controller alternative
- Challenges:
  - Need to have knowledge about functional programming
  - Bean Validation is different in Functional Web
  - Exception handling in Functional Web is different from the Controller approach

Movies Application using MicroServices Pattern



Movies Application using MicroServices Pattern



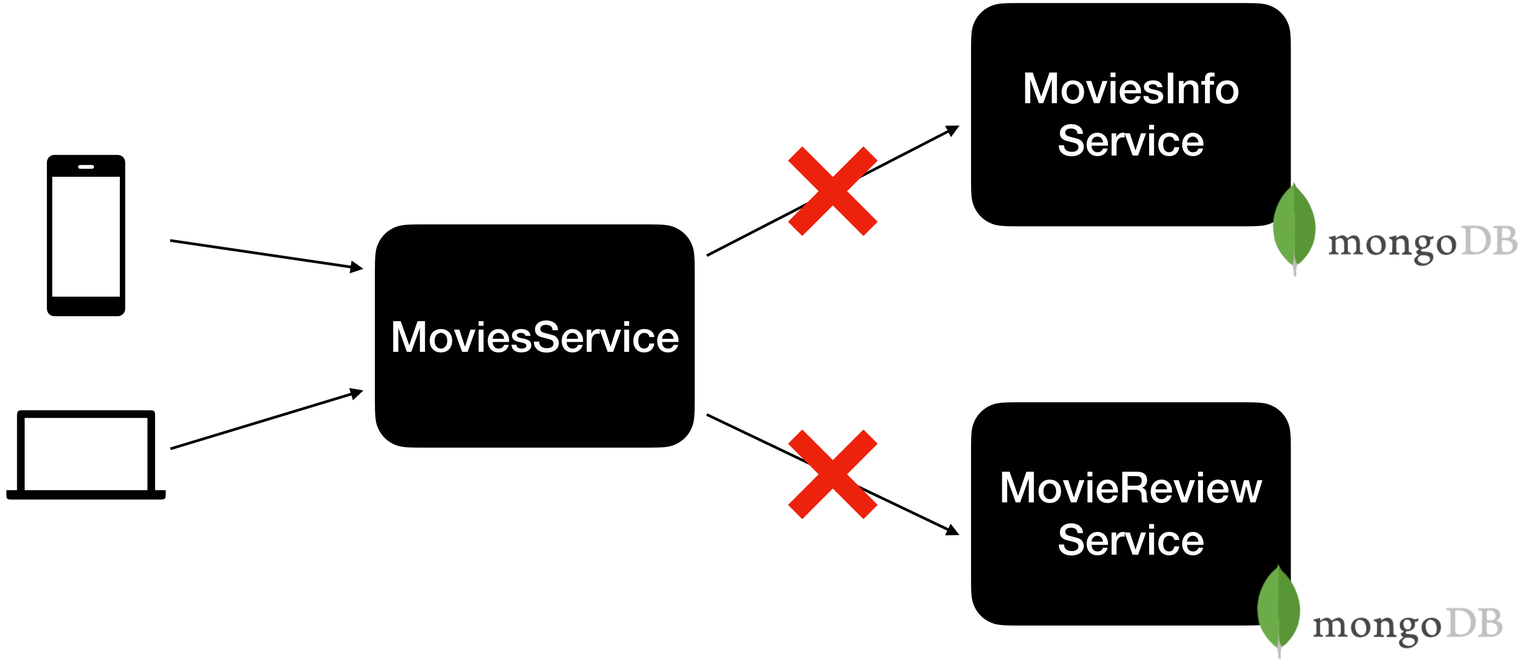
Webclient

WebClient

- It is a reactive non-blocking Rest Client
- It uses a functional style API
- It allows the application to interact with other services in a non-blocking fashion
- Its auto configured in to the application by Spring Boot

Exceptions in Service to Service Communication

Movies Application using MicroServices Pattern



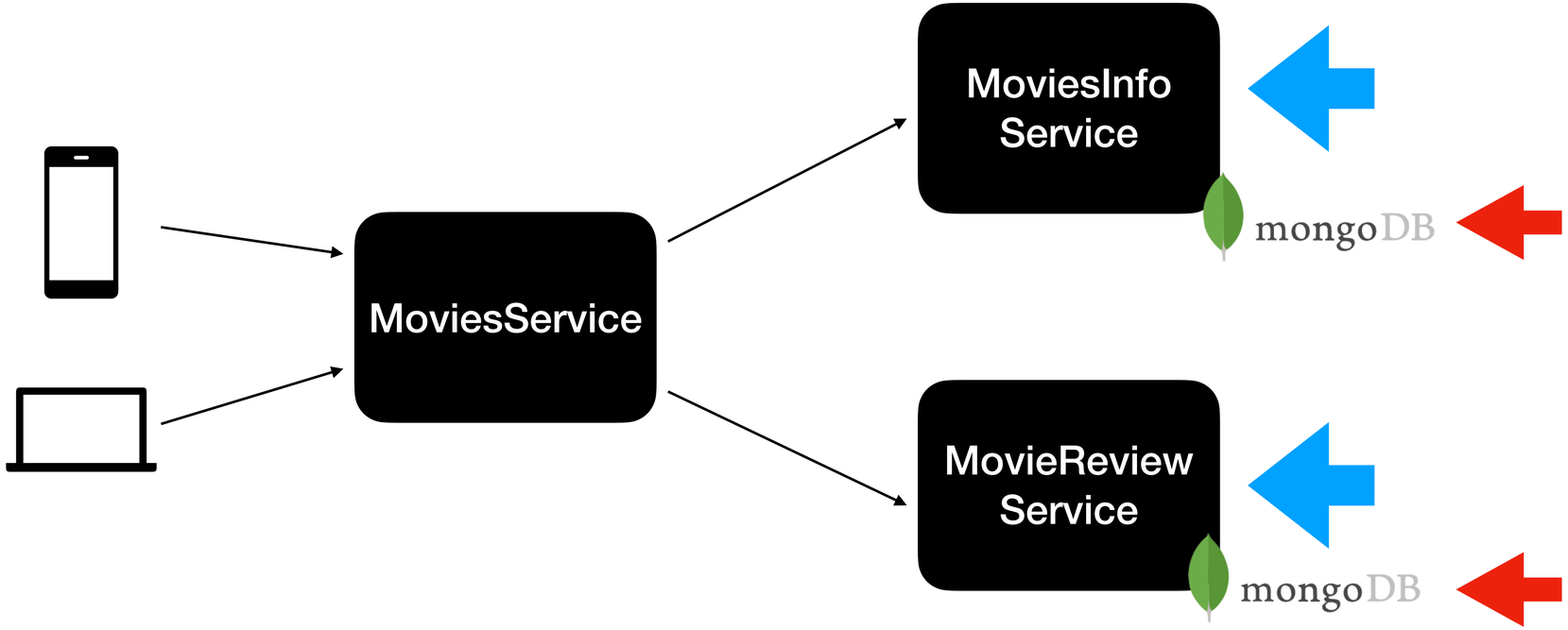
Http Failures

- Http Failure falls in to two categories:

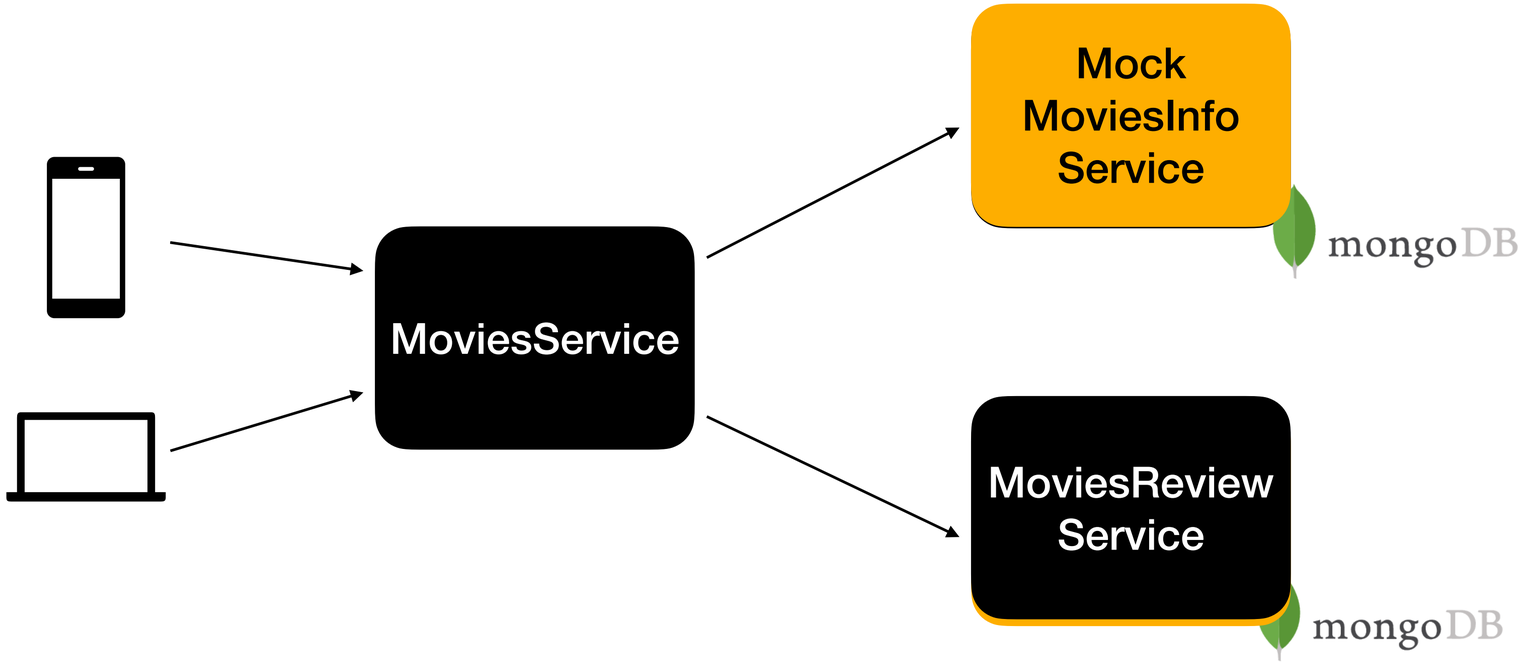
  - 4xx - Client Error
    - 400(Bad Request), 404 (Not Found) and etc.,
  - 5xx - Server Error
    - 500(Internal Server Error), 503 (Service Unavailable) and etc.,

Introduction to Wiremock

Movies Application using MicroServices Pattern



Movies Application using MicroServices Pattern



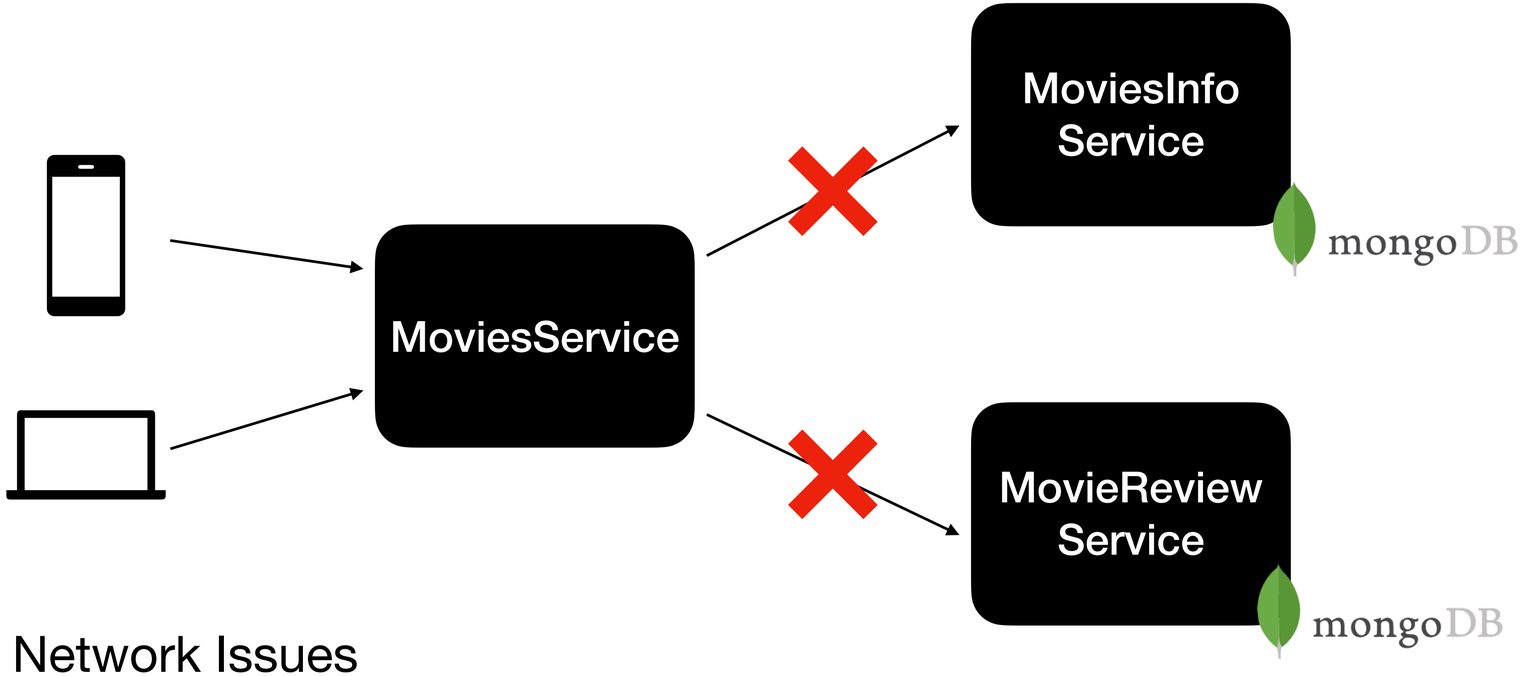
Wiremock to the Rescue

Benefits of WireMock

- Easy to test the success scenarios (2xx)
  - Test the contract
  - Serialization/Deserialization
- Easy to simulate error scenarios
  - 4xx
  - 5xx
  - SocketTimeout Exceptions and more..,

Why Retry failed HTTP calls ?

Movies Application using MicroServices Pattern



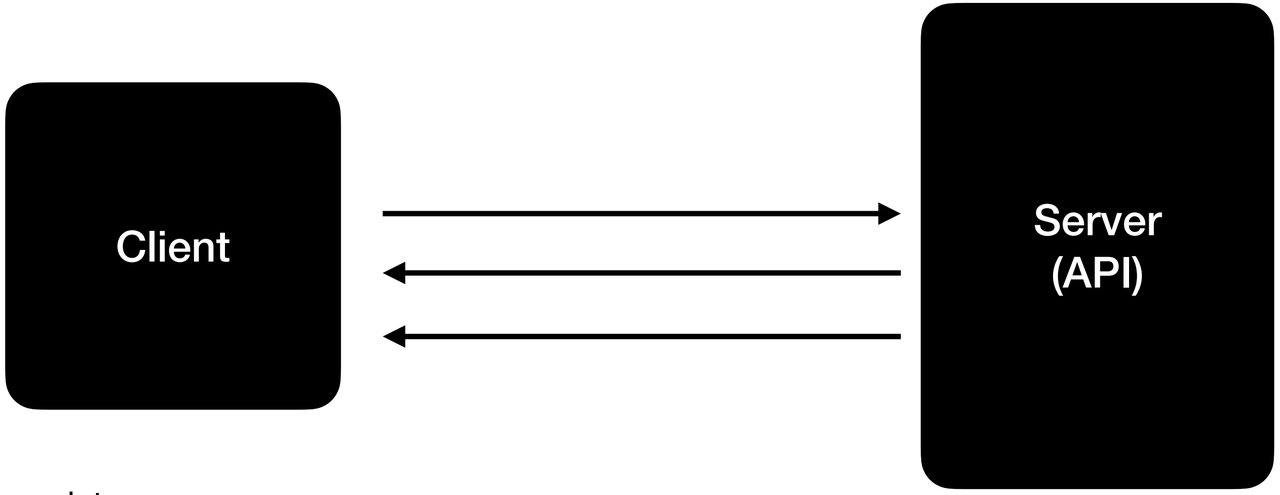
- Intermittent Network Issues
- Slow Network
- External Service is down

Handle Network Errors

- Retrying Failed Calls
  - Retry the failed call N number of times before giving up
  - Retry the failed call with a backoff
  - Retry Specific Exceptions
    - Retry only 5xx not 4xx exceptions

Server Sent Events (SSE)

Sever Sent Events



- Uber App - Realtime updates of the driver location
- Dominos, DoorDash etc.

- Data is sent in the form of events
- Its Unidirectional once the connection is established between the client and server
- Long live client connections

Publish new MovieInfo as a ServerSentEvent(SSE)

Sinks