**Java Reactive Programming**

**Core Features of Reactive Programming**

1. New Programming Paradigm

2. Asynchronous and non-blocking

3. Functional Style code

4. Data Flow as event driven stream

5. Backpressure on data streams

**Synchronous and blocking Asynchronous and non-blocking**

**A menu of a restaurant

AI-generated content may be incorrect.**A menu of a restaurant

AI-generated content may be incorrect.

A diagram of a process

AI-generated content may be incorrect. A diagram of a process

AI-generated content may be incorrect.

**Data Flow as event driven stream**

If any changes happen in the database (Publisher), it will publish an event, and the client (Subscriber) will stream the data; in this, the connection is always in an open state.

Ex :- Cricket live score.

A screen shot of a computer

AI-generated content may be incorrect.

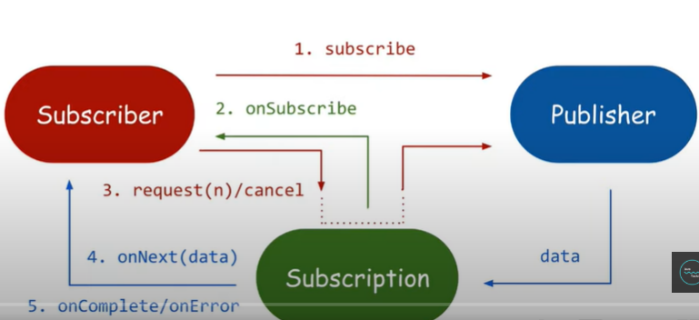
**Backpressure on data streams**

At the time of retrieval of data from the DB, if the response was loaded huge, which may not be handled by the application, then the reactive programming will inform the DB to slow down the response until the processing of the existing response is complete.

A diagram of a data processing process

AI-generated content may be incorrect.

**Specification/Rules**

**** **A diagram of a website

AI-generated content may be incorrect.**

**Publisher or Producer**

Publisher is a Data Source Who will always publish an event

public interface Publisher<T> {

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

}

**Subscriber or Consumer**

Subscriber will subscribe/consume the events from publisher

public interface Subscriber<T> {

public void onSubscribe (Subscription s);

public void onNext TT t);

public void onError (Throwable t);

public void onComplete();

}

**Subscription**

Subscription represents the unique relationship between a Subscriber and a Publisher

public interface Subscription {

public void request(long n);

public void cancel();

}

**Processor**

A Processor represents a processing stage-which is both a subscriber and a publisher and MUST obey the contracts of both.

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

}

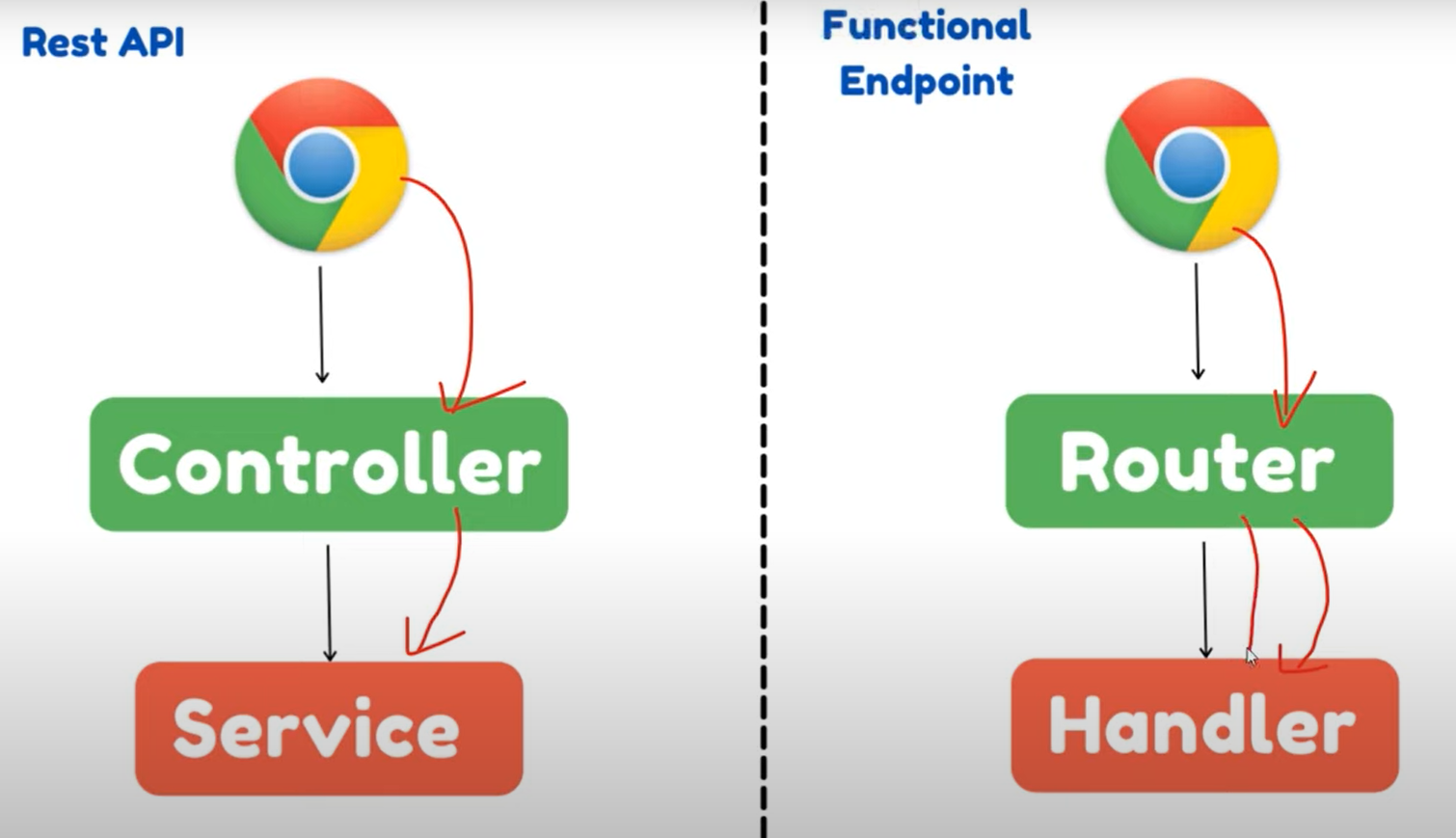
A diagram of a website

AI-generated content may be incorrect.

**Reactive Programming Library**

* Reactor
* RxJava
* Jdk9 Flow Reactive Stream

**REST API vs Functional Endpoints**



| **Feature** | **REST API (@RestController)** | **Functional Endpoints (RouterFunctions)** |
| --- | --- | --- |
| **Programming Style** | Imperative (Annotation-based) | Functional (Declarative) |
| **Threading Model** | Blocking (Servlet-based) | Non-blocking (Reactive Streams) |
| **Best Used For** | Monolithic apps, traditional APIs | Microservices, high-concurrency apps |
| **Performance** | Thread-per-request (Limited scalability) | Event-loop model (Highly scalable) |
| **Dependencies** | Uses spring-boot-starter-web | Uses spring-boot-starter-webflux |
| **Annotations** | @RestController, @RequestMapping | No annotations, uses RouterFunctions |
| **Return Type** | Objects (ResponseEntity<User>) | Reactive (Mono<User>, Flux<User>) |