Create a ToDo REST API with CRUDS Operations

1. File –> New –> Spring Starter Project

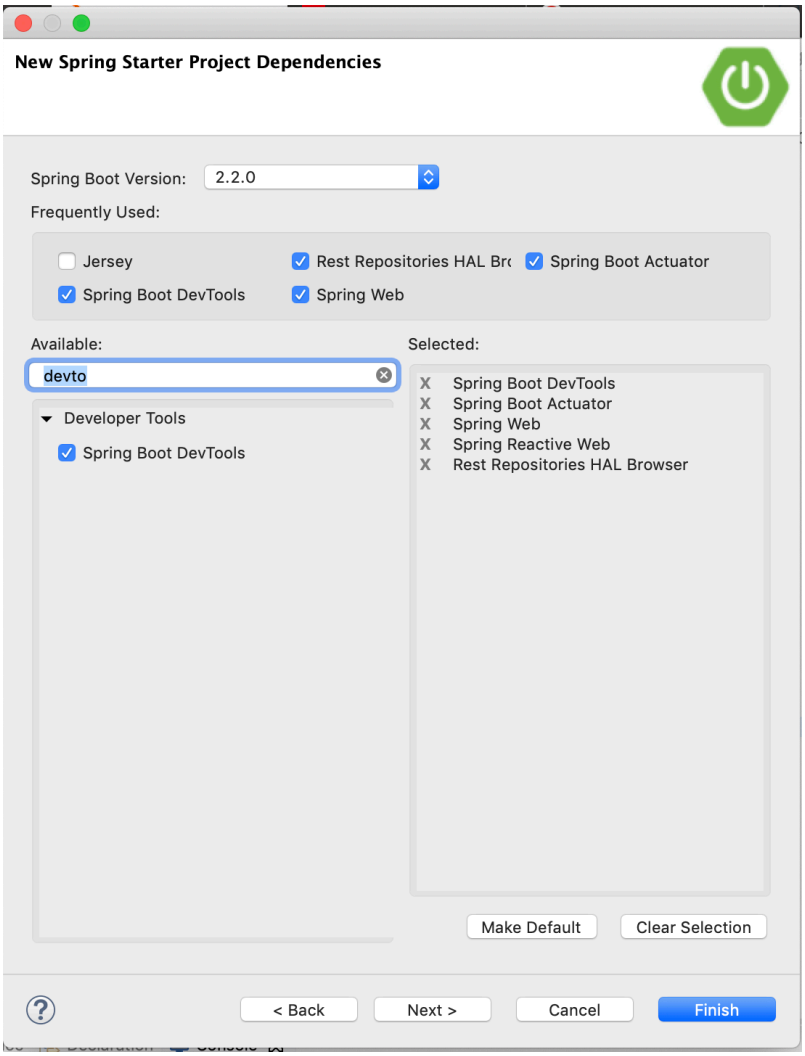
2. Name = com.reactive

3. Group = demo.reactive

4. Package = com.demo

5. Click on Next

6. Select “Spring Web”, “Webflux”, “Actuator”, “DevTools” & “HAL Explorer” modules



7. Click on Finish

8. Using ***Mono*** to emit one element

8.1 Add a Test case for ***Mono*** in

src/test/java/com.demo.DemoReactiveApplicationTests.java

```
@Test
public void monoExample() throws InterruptedException {

     Mono<String> stubMonoWithADelay =
Mono.just("Nanda").delayElement(Duration.ofSeconds(2));


}
```

8.2 We want to listen to the events from **Mono** and log them to the console.

We can do that using the statement specified here:

Add this statement within the above test case:

```
stubMonoWithADelay.subscribe(System.out::println);
```

8.3 RUN the TEST

8.4 Nothing prints!!

Because the Test execution ends before Mono emits the element after 2 seconds.

To prevent this, let's delay the execution of Test using Thread.sleep

```
Thread.sleep(4000);
```

9. When we create a subscriber using
stubMonoWithADelay.subscribe(System.out::println),
we are using the functional programming feature introduced in Java 8;

System.out::println is a method definition.
We are passing the method definition as a parameter to a method.

This is possible because of a specific functional interface called
Consumer.

A functional interface is an interface with only one method.

The ***Consumer functional interface*** is used to define an operation that accepts a single input argument and returns no result.

Instead of using a lambda expression, we can explicitly define Consumer as well.

src/test/java

SystemOutConsumer.java

```
package com.demo;

import java.util.Date;
import java.util.function.Consumer;

class SystemOutConsumer implements Consumer<String> {

     @Override
     public void accept(String t) {
          System.out.println("Received " + t + " at " + new
Date());
     }

}
```

10. Add SystemOutConsumer as a subscriber in the earlier test case.

DemoReactiveApplicationTests.java

```
stubMonoWithADelay.subscribe(new SystemOutConsumer());
```

11. RUN the TEST

```
Problems   @ Javadoc   Declaration   Console
<terminated> DemoReactiveApplicationTests [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_2
2019-10-20 18:47:13.594  INFO 1765 --- [           main] com.demo.DemoR
2019-10-20 18:47:13.595  INFO 1765 --- [           main] com.demo.DemoR
2019-10-20 18:47:14.455  WARN 1765 --- [           main] o.s.d.rest.webr
2019-10-20 18:47:14.455  WARN 1765 --- [           main] o.s.d.rest.webr
2019-10-20 18:47:14.455  WARN 1765 --- [           main] o.s.d.rest.webr
2019-10-20 18:47:15.158  INFO 1765 --- [           main] o.s.s.concurrer
2019-10-20 18:47:15.700  INFO 1765 --- [           main] o.s.b.a.e.web.l
2019-10-20 18:47:15.749  INFO 1765 --- [           main] com.demo.DemoR
Received Nanda at Sun Oct 20 18:47:18 EDT 2019
2019-10-20 18:47:20.026  INFO 1765 --- [extShutdownHook] o.s.s.concurrer
```

12. Create another Subscriber **WelcomeConsumer**

WelcomeConsumer.java

```
package com.demo;

import java.util.Date;
import java.util.function.Consumer;

class WelcomeConsumer implements Consumer<String> {

     @Override
     public void accept(String t) {
          System.out.println("Welcome to " + t + " at " + new
Date());
     }

}
```

13. Add SystemOutConsumer as a subscriber in the earlier test case.

DemoReactiveApplicationTests.java

```
stubMonoWithADelay.subscribe(new WelcomeConsumer());
```

14. RUN the TEST

```
Problems   @ Javadoc   Declaration   Console
<terminated> DemoReactiveApplicationTests [JUnit] /Library/Java/JavaVirtualMachin
2019-10-20 18:50:34.449  INFO 1774 --- [           main] com.c
2019-10-20 18:50:35.307  WARN 1774 --- [           main] o.s.c
2019-10-20 18:50:35.307  WARN 1774 --- [           main] o.s.c
2019-10-20 18:50:35.307  WARN 1774 --- [           main] o.s.c
2019-10-20 18:50:36.007  INFO 1774 --- [           main] o.s.s
2019-10-20 18:50:36.551  INFO 1774 --- [           main] o.s.b
2019-10-20 18:50:36.603  INFO 1774 --- [           main] com.c
Received Nanda at Sun Oct 20 18:50:38 EDT 2019
Welcome to Nanda at Sun Oct 20 18:50:38 EDT 2019
2019-10-20 18:50:40.874  INFO 1774 --- [extShutdownHook] o.s.s
```

15. Using ***Flux*** to emit multiple elements

DemoReactiveApplicationTests.java

```
@Test
public void simpleFluxStream() {
     Flux<String> stubFluxStream = Flux.just("Jane", "Joe");
     stubFluxStream.subscribe(new SystemOutConsumer());
}
```

16. RUN the TEST

```
Problems   @ Javadoc   Declaration   Console
<terminated> DemoReactiveApplicationTests [JUnit] /Library/Java/JavaVirtualMachines/jdk1.
2019-10-20 18:54:40.219  INFO 1778 --- [           main] com.demo.D
2019-10-20 18:54:41.094  WARN 1778 --- [           main] o.s.d.rest
2019-10-20 18:54:41.094  WARN 1778 --- [           main] o.s.d.rest
2019-10-20 18:54:41.094  WARN 1778 --- [           main] o.s.d.rest
2019-10-20 18:54:41.863  INFO 1778 --- [           main] o.s.s.conc
2019-10-20 18:54:42.381  INFO 1778 --- [           main] o.s.b.a.e.
2019-10-20 18:54:42.429  INFO 1778 --- [           main] com.demo.D
Received Jane at Sun Oct 20 18:54:42 EDT 2019
Received Joe at Sun Oct 20 18:54:42 EDT 2019
2019-10-20 18:54:42.682  INFO 1778 --- [extShutdownHook] o.s.s.conc
```

17. Add a following code for Flux with multiple subscribers

DemoReactiveApplicationTests.java

```
private static List<String> streamOfNames = Arrays.asList("John", "Adam",
"Joe", "Doe", "Jane");

@Test
public void fluxStreamWithDelay() throws InterruptedException {
      Flux<String> stubFluxWithNames =
Flux.fromIterable(streamOfNames).delayElements(Duration.ofMillis(1000));
      stubFluxWithNames.subscribe(new SystemOutConsumer());
      stubFluxWithNames.subscribe(new WelcomeConsumer());
      Thread.sleep(10000);
}
```

18. RUN the TEST

```
Problems   @ Javadoc   Declaration   Console
<terminated> DemoReactiveApplicationTests [JUnit] /Library/Java/JavaVirtualMachine
2019-10-20 18:59:49.043  INFO 1783 --- [           main] com.de
2019-10-20 18:59:49.044  INFO 1783 --- [           main] com.de
2019-10-20 18:59:49.940  WARN 1783 --- [           main] o.s.d.
2019-10-20 18:59:49.940  WARN 1783 --- [           main] o.s.d.
2019-10-20 18:59:49.941  WARN 1783 --- [           main] o.s.d.
2019-10-20 18:59:50.699  INFO 1783 --- [           main] o.s.s.
2019-10-20 18:59:51.344  INFO 1783 --- [           main] o.s.b.
2019-10-20 18:59:51.401  INFO 1783 --- [           main] com.de
Received John at Sun Oct 20 18:59:52 EDT 2019
Welcome to John at Sun Oct 20 18:59:52 EDT 2019
Welcome to Adam at Sun Oct 20 18:59:53 EDT 2019
Received Adam at Sun Oct 20 18:59:53 EDT 2019
Received Joe at Sun Oct 20 18:59:54 EDT 2019
Welcome to Joe at Sun Oct 20 18:59:54 EDT 2019
Received Doe at Sun Oct 20 18:59:55 EDT 2019
Welcome to Doe at Sun Oct 20 18:59:55 EDT 2019
Welcome to Jane at Sun Oct 20 18:59:56 EDT 2019
Received Jane at Sun Oct 20 18:59:56 EDT 2019
2019-10-20 19:00:01.677  INFO 1783 --- [extShutdownHook] o.s.s.
```

19. Creating a Spring ***Reactive controller*** is very similar to creating a Spring MVC controller.

The basic constructs are the same: ***@RestController*** and the different ***@RequestMapping*** annotations.

So, create a simple reactive controller

StockPriceEventController.java

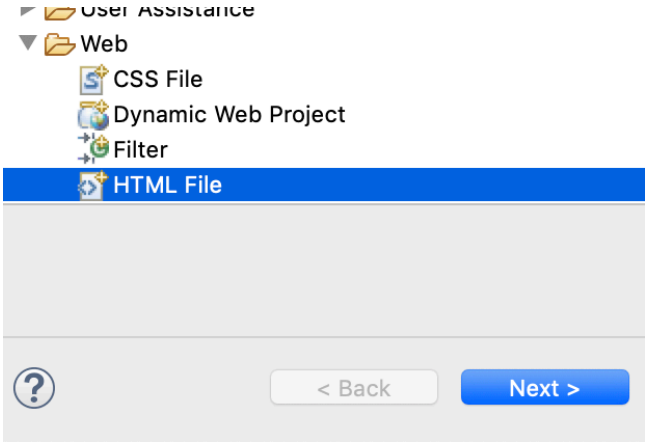
```java
package com.demo.controller;

import java.time.Duration;
import java.util.Date;
import java.util.concurrent.ThreadLocalRandom;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import reactor.core.publisher.Flux;

@RestController
public class StockPriceEventController {

    @GetMapping("/stocks/price/{stockCode}")
    Flux < String > retrieveStockPriceHardcoded
                                (@PathVariable("stockCode")
String stockCode) {
        return Flux.interval(Duration.ofSeconds(5))
                    .map(l -> getCurrentDate() + " : "
                           + getRandomNumber(100, 125))
                    .log();
    }

    private String getCurrentDate() {
        return (new Date()).toString();
    }

    private int getRandomNumber(int min, int max) {
        return ThreadLocalRandom.current().nextInt(min, max +
1);
    }

}
```

20. **Creating HTML View**

Let's create a view to show the current value of the stock on the screen.

We will create a simple static HTML page (resources/static/stock-price.html) with a button to start retrieving from the stream.

src/main/resources/static folder
 -> Right Click
 -> New -> Other



stock-price.html as file name

```
<div>
     <button id="subscribe-button">Get Latest IBM Price</button>
     <ul id="display"></ul>
</div>
```

21. We want to create a ***JavaScript*** method to register with the stream and append new elements to a specific ***div***.

So here is the JavaScript code:

src/main/resources/static folder
 -> Right Click
 -> New -> File

script.js

```
function registerEventSourceAndAddResponseTo(uri, elementId) {
     var stringEvents = document.getElementById(elementId);

     while (stringEvents.hasChildNodes()) {
          stringEvents.removeChild(stringEvents.lastChild);
     }

     var stringEventSource = new EventSource(uri);

     stringEventSource.onmessage = function(e) {
          var newElement = document.createElement("li");
          newElement.innerHTML = e.data;
          stringEvents.appendChild(newElement);
     }
}
```

22. The EventSource interface is used to receive server-sent events.

It connects to a server over HTTP and receives events in a **text/event-stream** format.

When it receives an element, the **onmessage** method is called.
The connection remains open until the **close** method is called.

Here is the code to register the **onclick** event for the **Get latest IBM price** button along with the stock price API:

script.js

```
function registerButton() {
     addEvent("click", document.getElementById('subscribe-
button'), function() {
          registerEventSourceAndAddResponseTo("/stocks/price/
IBM", "display");
     });
}

function addEvent(evnt, elem, func) {
     if (typeof (EventSource) !== "undefined") {
          elem.addEventListener(evnt, func, false);
     } else { // No much to do
          elem[evnt] = func;
     }
}
```

23. Add onclick event for the button by calling a function on load of a web page

static-price.html

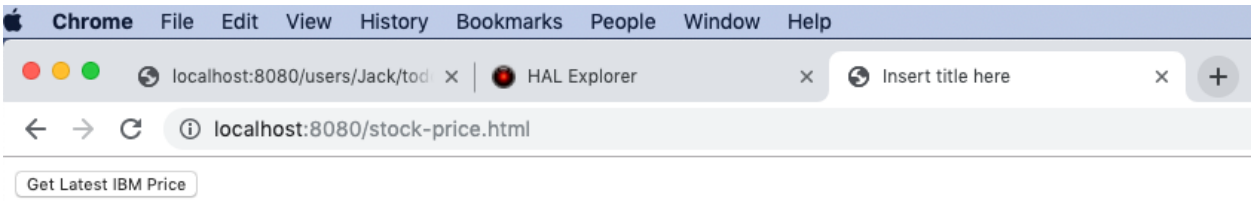
```
<body onload="registerButton();">
```

24. Launch the **SpringReactiveExampleApplication** application class as a Java application.

25. Open the Browser & visit:
http://localhost:8080/stock-price.html

When the **Get Latest IBM Price** button is clicked, EventSource kicks in and registers for events from "***/stocks/price/IBM***".

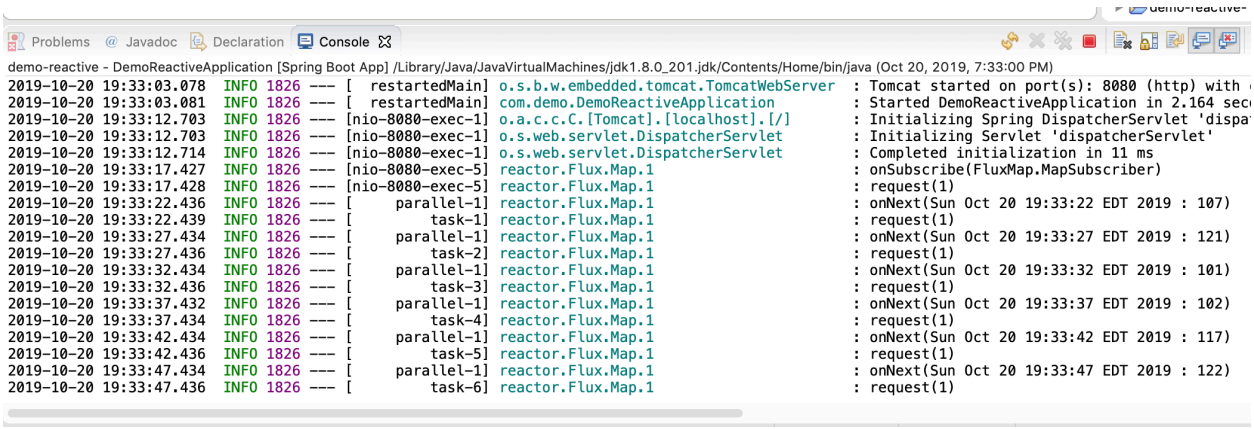
As soon as an element is received, it is shown on the screen.



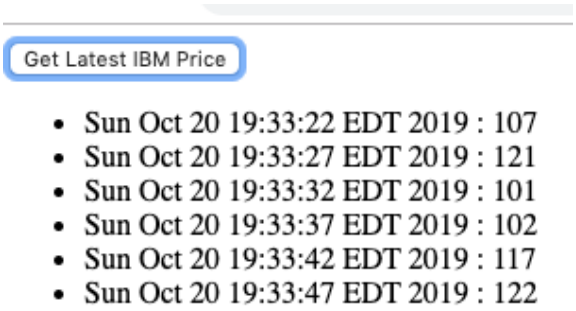
26. The following screenshot shows the screen after a few events are received.

You can observe that an event is received every 5 seconds:

26.1 Server logs updating Stock price every 5 seconds



26.2 Stock prices on HTML Web Page updating every 5 seconds



27. Close the Browser and observe Server logs for Spring Boot App

There must be this Exception:

```
java.lang.IllegalStateException: Calling [asyncError()] is not
valid for a request with Async state [MUST_DISPATCH]
    at
org.apache.coyote.AsyncStateMachine.asyncError(AsyncStateMachine
.java:440) ~[tomcat-embed-core-9.0.13.jar:9.0.13]
    at
org.apache.coyote.AbstractProcessor.action(AbstractProcessor.jav
a:512) [tomcat-embed-core-9.0.13.jar:9.0.13]
    at org.apache.coyote.Request.action(Request.java:430)
~[tomcat-embed-core-9.0.13.jar:9.0.13]
```

This is because Reactive Applications are fully compatible with Jetty/ Netty servers than Tomcat.

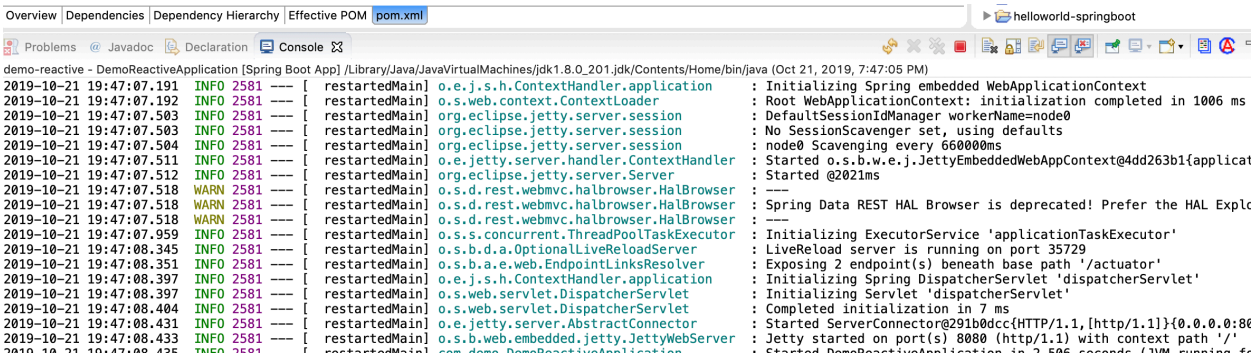
So, change your embedded server to Jetty as follows (#28):

28.

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <exclusions>
            <exclusion>
                  <artifactId>spring-boot-starter-tomcat</artifactId>
                  <groupId>org.springframework.boot</groupId>
            </exclusion>
      </exclusions>
</dependency>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

29. RUN the APP.

29.1 Observe the LAST LINE in the logs here about Jetty!



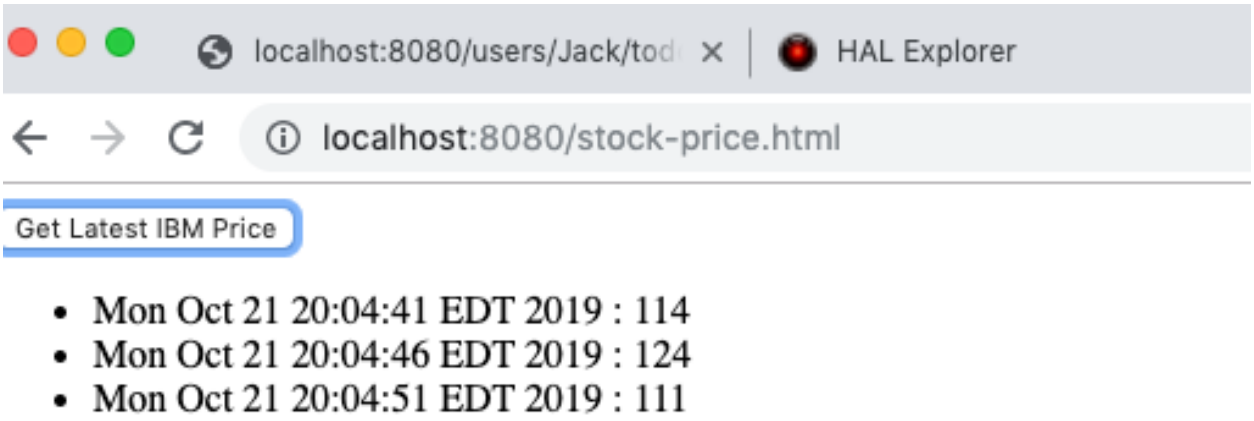
29.2 Open the Browser and visit:
http://localhost:8080/stock-price.html

29.3 Click on the Button

29.4 Server logs

```
2019-10-21 20:04:26.413  INFO 2637 --- [  restartedMain] o.s.b.web.embedded.jetty.JettyWebServer  : Jetty started on port(s) 8080 (http/1.1) with co
2019-10-21 20:04:26.415  INFO 2637 --- [  restartedMain] com.demo.DemoReactiveApplication         : Started DemoReactiveApplication in 2.545 seconds
2019-10-21 20:04:30.754  INFO 2637 --- [tp1095864726-30] org.eclipse.jetty.util.TypeUtil          : JVM Runtime does not support Modules
2019-10-21 20:04:36.760  INFO 2637 --- [tp1095864726-49] reactor.Flux.Map.1                       : onSubscribe(FluxMap.MapSubscriber)
2019-10-21 20:04:36.761  INFO 2637 --- [tp1095864726-49] reactor.Flux.Map.1                       : request(1)
2019-10-21 20:04:41.771  INFO 2637 --- [     parallel-1] reactor.Flux.Map.1                       : onNext(Mon Oct 21 20:04:41 EDT 2019 : 114)
2019-10-21 20:04:41.774  INFO 2637 --- [         task-1] reactor.Flux.Map.1                       : request(1)
2019-10-21 20:04:46.763  INFO 2637 --- [     parallel-1] reactor.Flux.Map.1                       : onNext(Mon Oct 21 20:04:46 EDT 2019 : 124)
2019-10-21 20:04:46.765  INFO 2637 --- [         task-2] reactor.Flux.Map.1                       : request(1)
2019-10-21 20:04:51.764  INFO 2637 --- [     parallel-1] reactor.Flux.Map.1                       : onNext(Mon Oct 21 20:04:51 EDT 2019 : 111)
2019-10-21 20:04:51.766  INFO 2637 --- [         task-3] reactor.Flux.Map.1                       : request(1)
```

29.5 Stock prices on the page



30. Close the Browser & Observe the Server logs

It invokes a cancel() method to remove subscription

As there are no other subscribers, the price change event pauses.

```
2019-10-21 20:04:56.763  INFO 2637 --- [     parallel-1] reactor.Flux.Map.1                    : onNext(Mon Oct 21 20:04:56 EDT 2019 : 104)
2019-10-21 20:04:56.764  INFO 2637 --- [         task-4] reactor.Flux.Map.1                    : request(1)
2019-10-21 20:05:01.765  INFO 2637 --- [     parallel-1] reactor.Flux.Map.1                    : onNext(Mon Oct 21 20:05:01 EDT 2019 : 124)
2019-10-21 20:05:01.771  INFO 2637 --- [         task-5] reactor.Flux.Map.1                    : cancel()
```

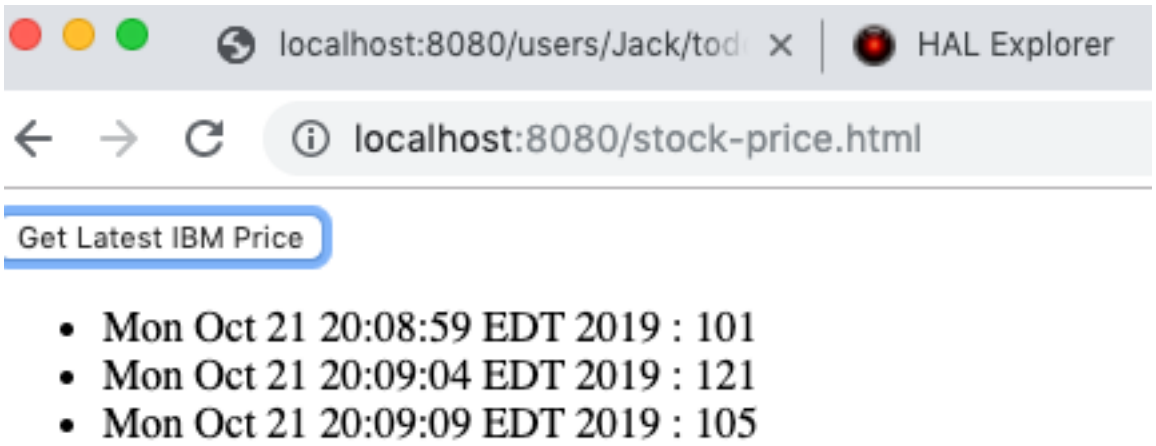
31. Open the browser ***again and visit:***
http://localhost:8080/stock-price.html

Click on the Button

32. The pricing event continues when there is a new Subscriber

```
2019-10-21 20:05:01.765  INFO 2637 --- [     parallel-1] reactor.Flux.Map.1                    : onNext(Mon Oct 21 20:05:01 EDT 2019 : 124)
2019-10-21 20:05:01.771  INFO 2637 --- [         task-5] reactor.Flux.Map.1                    : cancel()
2019-10-21 20:08:54.668  INFO 2637 --- [tp1095864726-49] reactor.Flux.Map.2                    : onSubscribe(FluxMap.MapSubscriber)
2019-10-21 20:08:54.668  INFO 2637 --- [tp1095864726-49] reactor.Flux.Map.2                    : request(1)
2019-10-21 20:08:59.673  INFO 2637 --- [     parallel-2] reactor.Flux.Map.2                    : onNext(Mon Oct 21 20:08:59 EDT 2019 : 101)
2019-10-21 20:08:59.674  INFO 2637 --- [         task-6] reactor.Flux.Map.2                    : request(1)
2019-10-21 20:09:04.673  INFO 2637 --- [     parallel-2] reactor.Flux.Map.2                    : onNext(Mon Oct 21 20:09:04 EDT 2019 : 121)
2019-10-21 20:09:04.675  INFO 2637 --- [         task-7] reactor.Flux.Map.2                    : request(1)
2019-10-21 20:09:09.673  INFO 2637 --- [     parallel-2] reactor.Flux.Map.2                    : onNext(Mon Oct 21 20:09:09 EDT 2019 : 105)
2019-10-21 20:09:09.675  INFO 2637 --- [         task-8] reactor.Flux.Map.2                    : request(1)
```

33.



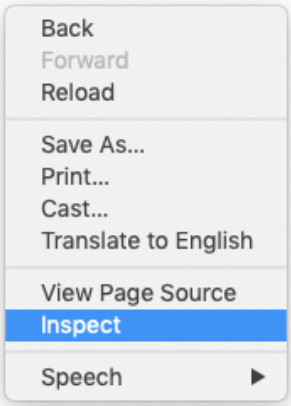
Watch Server logs for MongoDB connection

```
7892 --- [  restartedMain] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
7892 --- [  restartedMain] o.s.web.servlet.DispatcherServlet        : Completed initialization in 13 ms
7892 --- [  restartedMain] o.e.jetty.server.AbstractConnector       : Started ServerConnector@1a814c52{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
7892 --- [  restartedMain] o.s.b.web.embedded.jetty.JettyWebServer  : Jetty started on port(s) 8080 (http/1.1) with context path '/'
7892 --- [  restartedMain] com.demo.DemoReactiveApplication         : Started DemoReactiveApplication in 3.289 seconds (JVM running for 3.76)
7892 --- [ntLoopGroup-2-2] org.mongodb.driver.connection            : Opened connection [connectionId{localValue:3, serverValue:3}] to localhost:27017
```

Click on “Show IBM Details” button &

Watch Error in the browser

You need to open the Debugger tool first:



EventSource's response has a MIME type ("text/plain") that is not "text/event-stream". Aborting the connection.