Using Reactive types with Thymeleaf

Zoltan Altfatter

What is Thymeleaf

- Server side template engine for Java
  - HTML, XML, JavaScript, CSS, text
  - The view layer in Spring MVC, WebFlux
- Features
  - Templates, iteration, conditionals, layout, i18n

```
<table>
  <thead>
      <tr>
      <th th:text="#{msgs.headers.name}">Name</th>
      <th th:text="#{msgs.headers.price}">Price</th>
      </tr>
  </thead>
  <tbody>
      <tr th:each="prod : ${allProducts}">
      <td th:text="${prod.name}">Oranges</td>
      <td th:text="${#numbers.formatDecimal(prod.price,1,2)}">0.99</td>
      </tr>
  </tbody>
</table>
```

Reactive Thymeleaf

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

Pulls in the following transitive dependencies:

```xml
<dependency>
      <groupId>org.thymeleaf</groupId>
      <artifactId>thymeleaf-spring5</artifactId>
</dependency>
<dependency>
      <groupId>org.thymeleaf.extras</groupId>
      <artifactId>thymeleaf-extras-java8time</artifactId>
</dependency>
```

Reactive Thymeleaf

```java
@GetMapping("/reactive-template")
public String measurements(final Model model) {
   Flux<Measurement> measurements = webClient
           .get()
           .uri("/measurements")
           .accept(MediaType.APPLICATION_STREAM_JSON)
           .retrieve()
           .bodyToFlux(Measurement.class);

   // This object works as a wrapper that avoids Spring WebFlux to resolve before rendering the HTML.
   // Sets Thymeleaf in data-driven mode in order to produce (render) Server-Sent Events as the Flux
produces values
   // Creates a new lazy context variable, wrapping a reactive asynchronous data stream and
specifying a buffer size.

   IReactiveDataDriverContextVariable dataDriver = new
ReactiveDataDriverContextVariable(measurements,1);

   model.addAttribute("measurements", dataDriver);

   // the name of the view
   return "reactive-template";
}
```

Reactive Thymeleaf

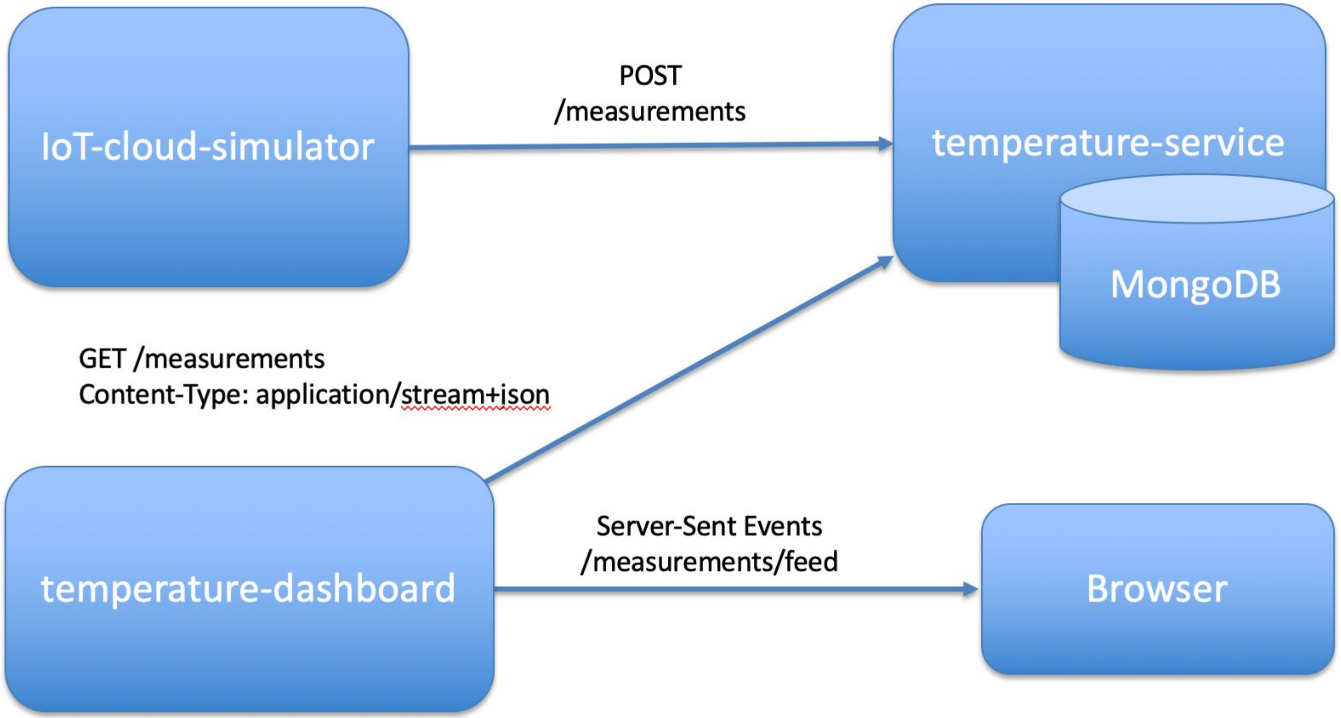
```
<div class="container wrapper">
   <table class="table table-striped">
       <thead>
       <tr>
           <th>Measurements</th>
       </tr>
       </thead>
       <tr th:each="measurement : ${measurements}">
           <td th:text="${measurement}">...</td>
       </tr>
   </table>
</div>
```

SSE (Server-Sent Events)

```
@GetMapping(value = "/measurements/feed", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
@ResponseBody
public Flux<Measurement> measurementsStream() {
   count++;
   return webClient
           .get()
           .uri("/measurements")
           .accept(MediaType.APPLICATION_STREAM_JSON)
           .retrieve()
           .bodyToFlux(Measurement.class)
           .log("feed-"+count);
}


// The browser connects to the server and receives measurements using Server-Sent Events
// The measurements are appended to the chart as they're received
var stockEventSource = new EventSource("/measurements/feed");
stockEventSource.onmessage = function (e) {
   appendMeasurementData(JSON.parse(e.data));
};
```

Current Weather Demo


IoT-cloud-simulator
POST /measurements
temperature-service
MongoDB
GET /measurements
Content-Type: application/stream+json
temperature-dashboard
Server-Sent Events /measurements/feed
Browser