

Advanced Route mapping and Returning Objects as Response

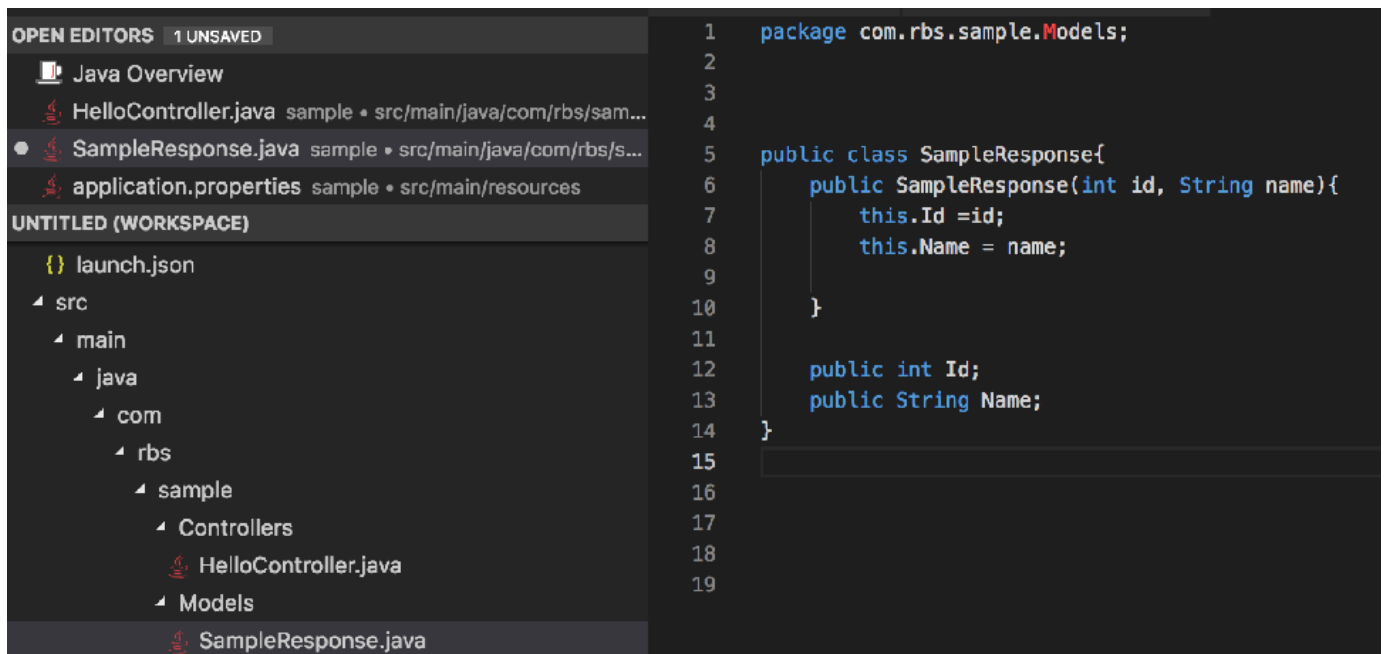
Model

Lets make our example from previous chapter a little bit more dynamic

In MVC framework, models represent the data definition that are part of request and response.

We will create our first sample model in a new folder create 'models' under 'sample'.

We name it as 'SampleResponse' which has two properties Id, Name and a constructor that initialize the properties.



The screenshot shows an IDE with two panels. The left panel displays the 'Project Explorer' with a tree structure: 'src' > 'main' > 'java' > 'com' > 'rbs' > 'sample' > 'Models'. The 'SampleResponse.java' file is selected under the 'Models' folder. The right panel shows the code editor with the following Java code:

```
1 package com.rbs.sample.Models;
2
3
4
5 public class SampleResponse{
6     public SampleResponse(int id, String name){
7         this.Id =id;
8         this.Name = name;
9     }
10
11
12     public int Id;
13     public String Name;
14 }
15
16
17
18
19
```

Parameters

Next, we will make our previous controller Index function a little bit more generic.

Our code should now accept two query params, id and secret, pass them on to build the SampleResponse model defined in previous step dynamically, and return the object model as response.

```

package com.rbs.sample.Controllers;

import com.rbs.sample.Models.SampleResponse;

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

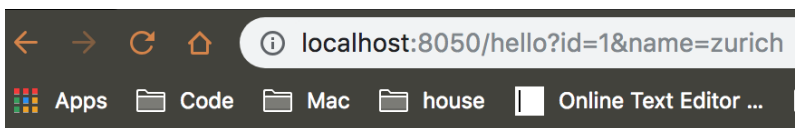
@RestController
public class HelloController {
    @GetMapping("/hello")
    public SampleResponse index(@RequestParam(value="name") String name,
                               @RequestParam(value="id") int id){
        return new SampleResponse(id,name);
    }
}

```

We have two new imports. One is importing the reference of the model and other is a Spring library called RequestParam, that helps you to define your route parameters.

Also, note, how we are defining the query parameters using @RequestParam and using the values to construct SampleResponse Model and return it as response.

So, now our previously static index route will take two new params, and the response will be generated dynamically depending on the parameters supplied.



```

Id: 1,
Name: "zurich"

```

Creating a Record

Now, let's try creating a new record for our model.

Obviously, we haven't got a DB behind, so will not be a real create, but we will demonstrate on how it easy it can be done.

So, the previous example was a GET, and this time we will create a POST to add a record sampleResponse.

```
@PostMapping(path = "/hello", consumes = "application/json", produces = "application/json")

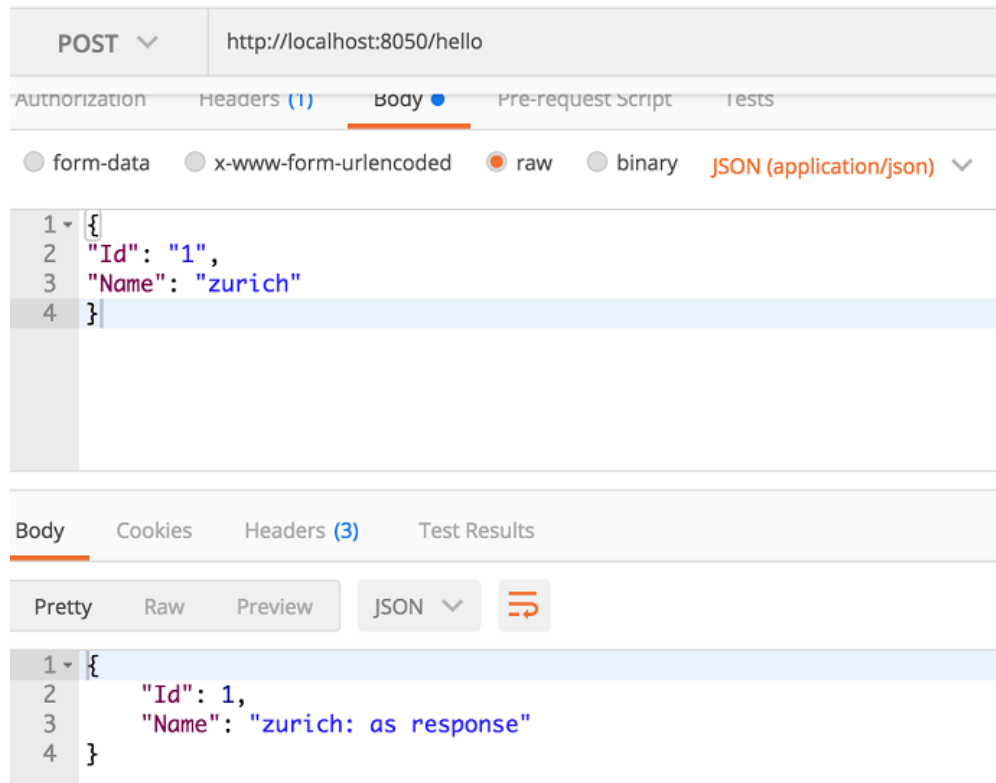
public SampleResponse index(@RequestBody SampleResponse response){
    response.Name = response.Name + ": as response";
    return response;
}
```

This code will allow API consumers to add json object containing a new sample record and have it added into the system. Notice the slightly different way of defining the mapping and declaration of a RequestBody instead of Param.

This code is capturing the request model, slightly altering the value and then return it as response.

That's all there is to build a POST method.

To test, we have to use a tool like Postman, POST requests are hard to test from Browser.



Please go through the url, verb, request body, content/type (the orange bit) and the response body to appreciate what happens here.

As mentioned at the start, framework features like RestController, RequestParam, GetMapping and PostMapping (among others) makes Spring ideal for rapid fire production grade web services and RESTful APIs.

Next Chapter: Call an upstream API to get data