Spring Boot REST Services using JPA Part 3

Description: Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

Project: This project expands on the program built using documents "Spring Boot REST Services using JPA" Part 1&2 which is based on Building REST services with Spring. The Part 1 program created a simple payroll service that manages the employees of a company. Part 2 updated the controller class adding links to the response to make it more RESTful. This document Part 3 will simplify adding links to the response messages.

Technology: This project uses the following technology:

Integrated Development Environment (IDE):

Spring Tool Suite 4 (Version: 4.15.0.RELEASE)

Java Development Kit (JDK):

Oracle's JDK 8 (1.8)

Other Technology:

Postman – a web and desktop application used for API testing.

Table of Contents

Glossary of Terminology

For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

Copy Existing Project: payroll

This project builds on the project created using “Spring Boot 300 REST Services Part 1” and on the website Building REST services with Spring. The payroll project must exist to following the details in this document. The project built in Part 1 is used as the starting point for this project and not the Part 2 project. The same concepts of adding links to the response is done here but a simplified constructs are used.

Right click on the project “payroll” ➔ select “Copy”
Right click in open area in “Package Explorer” ➔ select “Paste”
Enter “Project name:” **payroll-rest-2** ➔ Click “Copy”

We can skip refactoring any items in this project. The package name and main class can remain unchanged.

Project Payroll Discussion

This project builds on the knowledge from other projects using these documents. A basic understanding of using the Spring Tool Suite 4 for creating program element, resolving import issue, and other items is assumed.

At the end of the first project the following was discussed and the projected ended at the website section titled “What makes something RESTful?”.

The website describing this project “Building REST services with Spring” concludes that what was just built is in fact not a RESTful project but a Remote Procedure Call (RPC) project. Discussion below. In a project document named “Spring Boot 300 REST Services Part 2” this project is updated to become by definition RESTful.

At the end of the second project links were added to two of the application’s endpoints. This modification made those endpoints RESTful.

This project continues modification at the website section titled “Simplifying Link Creation”. It creates a common way to create links for response messages. It also modifies the entity class.

Simplifying Link Creation “In the code earlier, did you notice the repetition in single employee link creation? The code to provide a single link to an employee, as well as to create an "employees" link to the aggregate root, was shown twice. If that raised your concern, good! There’s a solution.” -- Building REST services with Spring

Update POM.xml

Add HATEOAS dependency to the pom.xml file.

> "Spring HATEOAS provides some APIs to ease creating REST representations that follow the HATEOAS principle when working with Spring and especially Spring MVC. The core problem it tries to address is link creation and representation assembly." -- Spring HATEOAS.

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

Update the Maven project

The project should automatically update the Maven library. In some cases it takes a manual update.
Right click project ➔ Highlight "Maven" ➔ Select "Update Project..."
The project should be selected ➔ click "OK"

Create Model Assembler Class

Create a Model Assembler class that uses HATEOAS to simplifies converting an Employee object to an EntityModel object. The EntityModel object will generate the links in response messages.

> "Simply put, you need to define a function that converts `Employee` objects to `EntityModel<Employee>` objects. While you could easily code this method yourself, there are benefits down the road of implementing Spring HATEOAS's `RepresentationModelAssembler` interface—which will do the work for you." -- Building REST services with Spring

Create Model Assembler Class: EmployeeModelAssembler

Create class named "EmployeeModelAssembler" in package "com.example.payroll". This class contains a method to convert an Employee object into an EntityModel object.

Copy and paste source code for class: EmployeeModelAssembler

Copy and replace the code shell in its entirety.

```
package com.example.payroll;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;

import org.springframework.hateoas.EntityModel;
import org.springframework.hateoas.server.RepresentationModelAssembler;
import org.springframework.stereotype.Component;

@Component
class EmployeeModelAssembler implements RepresentationModelAssembler<Employee, EntityModel<Employee>> {

  @Override
  public EntityModel<Employee> toModel(Employee employee) {

    return EntityModel.of(employee, //
        linkTo(methodOn(EmployeeController.class).one(employee.getId())).withSelfRel(),
        linkTo(methodOn(EmployeeController.class).all()).withRel("employees"));
  }
}
```

> "This simple interface has one method: `toModel()`. It is based on converting a non-model object (`Employee`) into a model-based object (`EntityModel<Employee>`)." -- Building REST services with Spring

Update Controller Class

To use the EmployeeModelAssembler class, we use constructor injection like was done for the repository class. The "Building REST services with Spring" website only modified two of the controller endpoints, the first two subsections here. This document will update the two other controller endpoints that returns employee objects.

Update Controller Class: EmployeeController

Add class variable and update the constructor.

```
        private final EmployeeModelAssembler assembler;

        EmployeeController(EmployeeRepository repository, EmployeeModelAssembler assembler) {
                this.repository = repository;
                this.assembler = assembler;
        }
```

```
13 @RestController
14 class EmployeeController {
15
16     private final EmployeeRepository repository;
17     private final EmployeeModelAssembler assembler;
18
19     EmployeeController(EmployeeRepository repository, EmployeeModelAssembler assembler) {
20         this.repository = repository;
21         this.assembler = assembler;
22     }
```

Update Endpoint @GetMapping("/employees/{id}")

Existing code GetMapping by id method:

```
  @GetMapping("/employees/{id}")
  Employee one(@PathVariable Long id) {

    return repository.findById(id)
      .orElseThrow(() -> new EmployeeNotFoundException(id));
  }
```

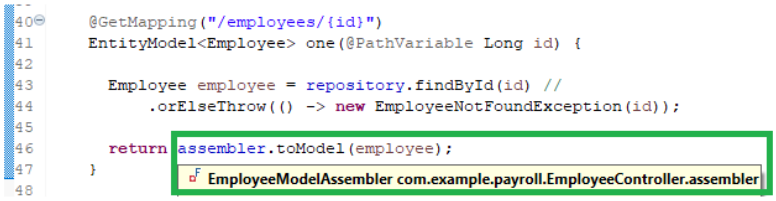
Code update for GetMapping by id method (copy and replace existing code):

```
        @GetMapping("/employees/{id}")
        EntityModel<Employee> one(@PathVariable Long id) {

          Employee employee = repository.findById(id) //
              .orElseThrow(() -> new EmployeeNotFoundException(id));

          return assembler.toModel(employee);
        }
```

Resolve the import errors, add the following import:

```
import org.springframework.hateoas.EntityModel;
```

The new EmployeeModelAssembler class is invoked in the return statement.



Update Endpoint @GetMapping("/employees")

Existing code GetMapping method getting all employees:

```
// Aggregate root
// tag::get-aggregate-root[]
@GetMapping("/employees")
List<Employee> all() {
  return repository.findAll();
}
// end::get-aggregate-root[]
```

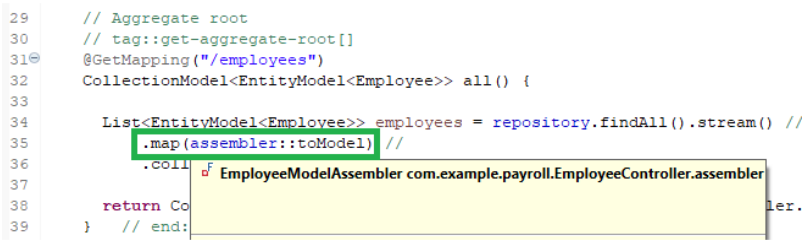
Code update for GetMapping method getting all employees (copy and replace existing code):

```
@GetMapping("/employees")
CollectionModel<EntityModel<Employee>> all() {

  List<EntityModel<Employee>> employees = repository.findAll().stream() //
      .map(assembler::toModel) //
      .collect(Collectors.toList());

  return CollectionModel.of(employees, linkTo(methodOn(EmployeeController.class).all()).withSelfRel());
}         // end::get-aggregate-root[]
```

Resolve the import errors, add the following imports:

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;
import java.util.stream.Collectors;
import org.springframework.hateoas.CollectionModel;
```

Here the new EmployeeModelAssembler class is invoked when creating the list of employees.



Update Endpoint @PostMapping("/employees")

Existing code PostMapping method add an employee:

```
@PostMapping("/employees")
Employee newEmployee(@RequestBody Employee newEmployee) {
        return repository.save(newEmployee);
}
```

Code update for PostMapping method add an employee (copy and replace existing code):

```
@PostMapping("/employees")
EntityModel<Employee> newEmployee(@RequestBody Employee newEmployee) {
	return assembler.toModel(repository.save(newEmployee));
}
```

Update Endpoint @PutMapping("/employees")

Existing code PutMapping method update an employee:

```
@PutMapping("/employees/{id}")
Employee replaceEmployee(@RequestBody Employee newEmployee, @PathVariable Long id) {

	return repository.findById(id).map(employee -> {
		employee.setName(newEmployee.getName());
		employee.setRole(newEmployee.getRole());
		return repository.save(employee);
	}).orElseGet(() -> {
		newEmployee.setId(id);
		return repository.save(newEmployee);
	});
}
```

Code update for PutMapping method update an employee (copy and replace existing code):

```
@PutMapping("/employees/{id}")
EntityModel<Employee> replaceEmployee(@RequestBody Employee newEmployee, @PathVariable Long id) {

	return repository.findById(id).map(employee -> {
		employee.setName(newEmployee.getName());
		employee.setRole(newEmployee.getRole());
		return assembler.toModel(repository.save(employee));
	}).orElseGet(() -> {
		newEmployee.setId(id);
		return assembler.toModel(repository.save(newEmployee));
	});
}
```

Test Changes to Controller Endpoints

Start the application inside the IDE as Spring Boot App.

Test Changes to Endpoint @GetMapping("/employees/{id}")

Send Postman test created in previous document GET “find by employee record id” for record id 1.

Due to the code modifications, the response now shows two links (self, employees). The self link directly access record id of 1. The employees link brings back all employee records in the database.

Test Changes to Endpoint @GetMapping("/employees")

Send Postman test created in previous document GET “all employee records”.

The code modifications for this method also produces links in the response message.

Test Changes to Endpoint @PostMapping("/employees")

Send Postman test created in previous document POST “create an employee”.

The code modifications for this method also produces links in the response message.

Test Changes to Endpoint @PutMapping("/employees")

Send Postman test created in previous document PUT “update an existing employee record”.

The code modifications for this method also produces links in the response message.

Project Conclusion

"At this stage, you've created a Spring MVC REST controller that actually produces hypermedia-powered content! Clients that don't speak HAL can ignore the extra bits while consuming the pure data. Clients that DO speak HAL can navigate your empowered API.
But that is not the only thing needed to build a truly RESTful service with Spring." -- Building REST services with Spring

Evolving REST APIs

Further updates to the project involve updating the employee entity and adding customer order processing.

Evolving REST APIs
"With one additional library and a few lines of extra code, you have added hypermedia to your application. But that is not the only thing needed to make your service RESTful. An important facet of REST is the fact that it's neither a technology stack nor a single standard.

REST is a collection of architectural constraints that when adopted make your application much more resilient. A key factor of resilience is that when you make upgrades to your services, your clients don't suffer from downtime."
-- Building REST services with Spring

- These changes will be discussed in document titled "Spring Boot 300 REST Services Part 4".