

# Spring Boot REST Services using JPA Part 1

**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:** Create a simple payroll service that manages the employees of a company. We'll store employee objects in a (H2 in-memory) database, and access them (via something called JPA). Then we'll wrap that with something that will allow access over the internet (called the Spring MVC layer). This project is based on [Building REST services with Spring](#) and just expands on the screenshots and step by step instructions.

**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 .....	3
Generate Spring Boot Download .....	3
Import the Spring Boot Download .....	3
Import the project: "payroll" .....	3
Project Payroll Discussion .....	4
Main Class: PayrollApplication.....	4
Create Domain Object Model Class .....	4
Create Domain Model Class: Employee.....	4
Copy and paste source code for class: Employee .....	5
Create Repository Class .....	6
Create Repository Interface: EmployeeRepository .....	6
Copy and paste source code for class: EmployeeRepository.....	6
Create Helper Class.....	7
Create Helper Class: LoadDatabase .....	7
Copy and paste source code for class: LoadDatabase .....	7
Create Exception Classes .....	8

Create Exception Class: EmployeeNotFoundException .....	8
Copy and paste source code for class: EmployeeNotFoundException .....	8
Create Exception Class: EmployeeNotFoundExceptionAdvice .....	8
Copy and paste source code for class: EmployeeNotFoundExceptionAdvice .....	9
Create Controller Class .....	10
Create Controller Class: EmployeeController .....	10
Copy and paste source code for class: EmployeeController .....	10
Test the Project Using Spring Boot Tomcat Server .....	11
Test the Project Using Postman .....	11
Create Postman HTTP Requests for Project (payroll) .....	11
GET Request: all employee records .....	11
GET Request: find by employee record id .....	12
POST Request: create an employee .....	14
PUT Request: update an existing employee record .....	14
DELETE Request: delete an employee record .....	15
Project Conclusion .....	17

## 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”.

## Generate Spring Boot Download

Follow the instructions in the document title “Appendix 02 Spring Initializr” to generate a spring boot download for this project.

When the document talks about adding dependencies add only these:

**Spring Web**  
**Spring Data JPA**  
**H2 Database**

Dependencies		ADD DEPENDENCIES... CTRL + B
Spring Web	WEB	Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JPA	SQL	Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
H2 Database	SQL	Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

When the document talks about the items in the “**Project Metadata**” use the values shown below:

“**Group**” use “**com.example**”  
“**Artifact**” use “**payroll**”  
“**Name**” use “**payroll**”  
“**Package name**” use “**com.example.payroll**”

For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Sprint Tool Suite 4 workspace.

## Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

Import the project: “payroll”

Follow the instructions in the document title “Appendix 03 Import Spring Tool Suite Project” and import the “**payroll**” project that was created with Spring Initializr.

## 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.

The project will be a simple payroll application using restful APIs, an H2 memory database, and JPA for database access.

## Main Class: PayrollApplication

This project uses the generated application without modification to start the applications.

The PayrollApplication class created when using the Spring Initializr is used without any additional modification from what was generated. This class contains the “**public static void main(String[] args) {}**” method that starts the application listening when the internal web server is started.

```
package com.example.payroll;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PayrollApplication {

    public static void main(String[] args) {
        SpringApplication.run(PayrollApplication.class, args);
    }

}
```

## Create Domain Object Model Class

Create a Domain Object also known as a Model class with database annotation. The database annotation is what will create the Domain Object structure as a database table with columns. Since this project is using a H2 Database, the structure will only exist in memory with no physical representation.

Create a repository class which provide various operations involving the Domain Object.

## Create Domain Model Class: Employee

Create class named “Employee” in package “com.example.payroll”. This class is an entity class mapping to the H2 memory database. It generates unique record id, contains an employee name and their role. The class has getters and setters along with several override methods.

- “@Entity” is a JPA annotation to make this object ready for storage in a JPA-based data store.

- `id`, `name`, and `role` are attributes of our Employee domain object. `id` is marked with more JPA annotations to indicate it's the primary key and automatically populated by the JPA provider.
- a custom constructor is created when we need to create a new instance, but don't yet have an id."

-- [Building REST services with Spring](#).

Copy and paste source code for class: Employee

Copy and replace the code shell in its entirety.

```
package com.example.payroll;

import java.util.Objects;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
class Employee {

    private @Id @GeneratedValue Long id;
    private String name;
    private String role;

    Employee() {}

    Employee(String name, String role) {

        this.name = name;
        this.role = role;
    }

    public Long getId() {
        return this.id;
    }

    public String getName() {
        return this.name;
    }

    public String getRole() {
        return this.role;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setRole(String role) {
        this.role = role;
    }

    @Override
    public boolean equals(Object o) {

        if (this == o)
            return true;
        if (!(o instanceof Employee))
            return false;
        Employee employee = (Employee) o;
        return Objects.equals(this.id, employee.id) && Objects.equals(this.name, employee.name)
            && Objects.equals(this.role, employee.role);
    }

    @Override
    public int hashCode() {
        return Objects.hash(this.id, this.name, this.role);
    }

    @Override
    public String toString() {
        return "Employee{" + "id=" + this.id + ", name='" + this.name + '\'' + ", role='" + this.role + '\'' + '}';
    }
}
```

## Create Repository Class

Create a repository interface that extends `JpaRepository<>`. There are no customer methods to access the memory database. All access is through CRUD methods defined in the extended `JpaRepository<>` class.

### Create Repository Interface: `EmployeeRepository`

Create interface named “`EmployeeRepository`” in package “`com.example.payroll`” that extends `JpaRepository<Employee, Long>`. This interface allows creating, reading, updating, and deleting employee data in the memory database.

“To get all this free functionality, all we had to do was declare an interface which extends Spring Data JPA’s `JpaRepository`, specifying the domain type as `Employee` and the id type as `Long` .

Spring Data’s [repository solution](#) makes it possible to sidestep data store specifics and instead solve a majority of problems using domain-specific terminology.”

-- [Building REST services with Spring](#).

Copy and paste source code for class: `EmployeeRepository`

Copy and replace the code shell in its entirety.

```
package com.example.payroll;

import org.springframework.data.jpa.repository.JpaRepository;

interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```

## Create Helper Class

Create a helper class to initially load the memory database.

### Create Helper Class: LoadDatabase

Create class named “LoadDatabase” in package “com.example.payroll”. This class is a helper class performing an initial load in the H2 memory database. This class has a logger, @Configuration annotation and contains a bean.

@Configuration annotation indicates that the class has @Bean definition methods. The Spring container can process the class and generate Spring Beans to be used in the application.

### Copy and paste source code for class: LoadDatabase

Copy and replace the code shell in its entirety.

```
package com.example.payroll;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
class LoadDatabase {

    private static final Logger log = LoggerFactory.getLogger(LoadDatabase.class);

    @Bean
    CommandLineRunner initDatabase(EmployeeRepository repository) {

        return args -> {
            log.info("Preloading " + repository.save(new Employee("Bilbo Baggins", "burglar")));
            log.info("Preloading " + repository.save(new Employee("Frodo Baggins", "thief")));
        };
    }
}
```

## Create Exception Classes

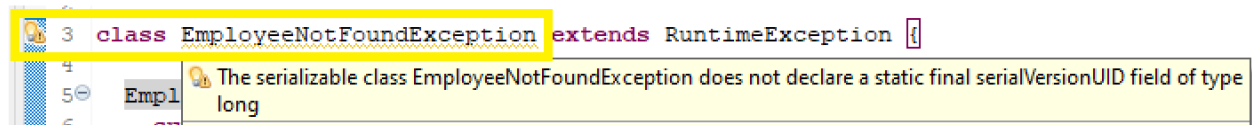
Create customized application exception classes to gracefully handle certain runtime exceptions.

### Create Exception Class: EmployeeNotFoundException

Create class named “EmployeeNotFoundException” in package “com.example.payroll”. This class extends RuntimeException and provides a custom exception for when an employee record is not found.

### Copy and paste source code for class: EmployeeNotFoundException

Copy and replace the code shell in its entirety. The code copied from the website produced a warning. To resolve the warning the serialVersionUID class variable was created.



```
3 class EmployeeNotFoundException extends RuntimeException {  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

```
package com.example.payroll;  
  
class EmployeeNotFoundException extends RuntimeException {  
    private static final long serialVersionUID = 1L;  
  
    EmployeeNotFoundException(Long id) {  
        super("Could not find employee " + id);  
    }  
}
```

### Create Exception Class: EmployeeNotFoundAdvice

Create class named “EmployeeNotFoundAdvice” in package “com.example.payroll”. This class is used to render an HTTP 404 error.

@ControllerAdvice which allows to handle exceptions across the whole application in one global handling component.

- `@ResponseBody` signals that this advice is rendered straight into the response body.
- `@ExceptionHandler` configures the advice to only respond if an `EmployeeNotFoundException` is thrown.
- `@ResponseStatus` says to issue an `HttpStatus.NOT_FOUND`, i.e. an **HTTP 404**.
- The body of the advice generates the content. In this case, it gives the message of the exception.”

-- [Building REST services with Spring](#).



Copy and paste source code for class: EmployeeNotFoundAdvice

Copy and replace the code shell in its entirety.

```
package com.example.payroll;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class EmployeeNotFoundAdvice {

    @ResponseBody
    @ExceptionHandler (EmployeeNotFoundException.class)
    @ResponseStatus (HttpStatus.NOT_FOUND)
    String employeeNotFoundHandler (EmployeeNotFoundException ex) {
        return ex.getMessage();
    }
}
```

## Create Controller Class

Create a controller class as part of the Model View Controller (MVC) architecture. The Model class Employee and associated repository class was created in a previous section. The View would be a web page accessing the application. In this case Postman will be used as the View component. The Controller contains the entry point, the URL mapping into the application.

### Create Controller Class: EmployeeController

Create class named “EmployeeController” in package “com.example.payroll”. This class is a helper class performing an initial load in the H2 memory database. This class has a logger, @Configuration annotation and contains a bean.

@Configuration annotation indicates that the class has @Bean definition methods. The Spring container can process the class and generate Spring Beans to be used in the application.

### Copy and paste source code for class: EmployeeController

Copy and replace the code shell in its entirety.

```
package com.example.payroll;

import java.util.List;

import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
class EmployeeController {

    private final EmployeeRepository repository;

    EmployeeController(EmployeeRepository repository) {
        this.repository = repository;
    }

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

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

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

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

    @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);
        });
    }

    @DeleteMapping("/employees/{id}")
    void deleteEmployee(@PathVariable Long id) {
        repository.deleteById(id);
    }
}

```

## Test the Project Using Spring Boot Tomcat Server

An external build of an executable JAR file is not necessary to test this project. Use the Tomcat server bundled with a Spring Boot project built with the “[Spring Initializr](#)”.

Run the application within the IDE.

Right click inside the main class.

Highlight “Run As” → select “3 Spring Boot App”

## Test the Project Using Postman

This section uses Postman to test this modified project. The testing follow the Postman approach described in other project documents using Postman. Using an existing Postman account or creating a new account by using the document title “Environment Setup 05 Postman Setup” is required for following the test in this section. A basic understanding of using Postman is assumed.

The Postman desktop application is used since we are testing “localhost” URLs. In a version update of Postman, it started restricting using the web-based version for “localhost” testing.

Create a Postman Collection: Spring Boot (payroll)

### Create Postman HTTP Requests for Project (payroll)

This section describes a collection of various HTTP request for testing this project. While creating the requests in this section keep in mind that no custom queries were created. All endpoints will use the default CRUD methods from the extended class JpaRepository.

#### GET Request: all employee records

Request to get the list of all employee records in the database.

URL: “http://localhost:8080/employees”

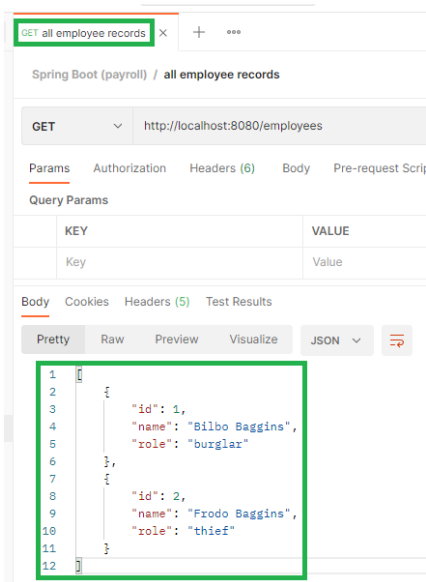
This request maps to the Controller class EmployeeController endpoint:

```

@GetMapping("/employees")
List<Employee> all() {
    return repository.findAll();
}

```

Examine the response: two records were preloaded in the memory database by class LoadDatabase. The response shows the records preloaded.



### GET Request: find by employee record id

Define a query request to find an employee record by the unique record id. In this request there is “/id” notation. This notation is called a path parameter.

“http://localhost:8080/employees/#”

This request maps to the Controller class `EmployeeController` endpoint:

```

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

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

```

### GET Request: find by employee record id success

URL: “http://localhost:8080/employees/1”

Examine the response:

Since the record id is a unique record id identified and generated by the annotation and statement in class `Employee`.

```

9  @Entity
10  class Employee {
11
12      private @Id @GeneratedValue Long id;

```

The query will return one record based on the unique record id.

GET find by employee record | + ...

Spring Boot (payroll) / find by employee record id

GET http://localhost:8080/employees/1

Params Authorization Headers (6) Body Pre-req

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "Bilbo Bagins",
4   "role": "burglar"
5 }
```

*GET Request: find by employee record id not found*

URL: "http://localhost:8080/employees/0"

Examine the response:

The message received in the response was generated by the exception class "EmployeeNotFoundException".

```
EmployeeNotFoundException.java X
1 package com.example.payroll;
2
3 class EmployeeNotFoundException extends RuntimeException {
4     private static final long serialVersionUID = 1L;
5
6     EmployeeNotFoundException(Long id) {
7         super("Could not find employee " + id);
8     }
9 }
```

GET find by employee record | + ...

Spring Boot (payroll) / find by employee record id not found

GET http://localhost:8080/employees/0

Params Authorization Headers (6) Body Pre-request Sc

Query Params

KEY	VALUE
-----	-------

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 Could not find employee 0
```

### POST Request: create an employee

Request to create an employee record in the database.

URL: "http://localhost:8080/employees"

Request body:

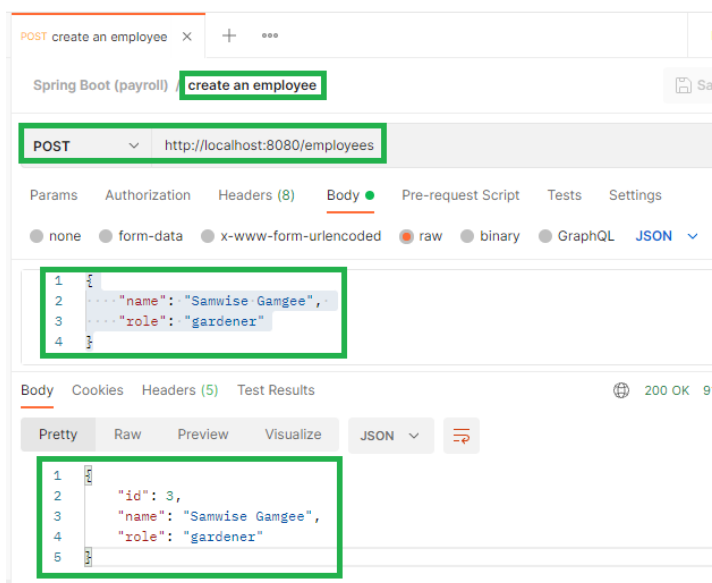
```
{
  "name": "Samwise Gamgee",
  "role": "gardener"
}
```

This request maps to the Controller class `EmployeeController` endpoint:

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

Examine the response:

The response echoes back the record created including its record number, 3 in this case.



### PUT Request: update an existing employee record

Request to update an exist record in the database. A PUT request updates the entire record even if no data is changed. The path parameter in the URL `"/#"` is the unique record id to modify in the database. If the record does not exist, it will be created.

URL: "http://localhost:8080/employees/3"

Request body:

```
{
  "name": "Samwise Gamgee",
  "role": "ring bearer"
}
```

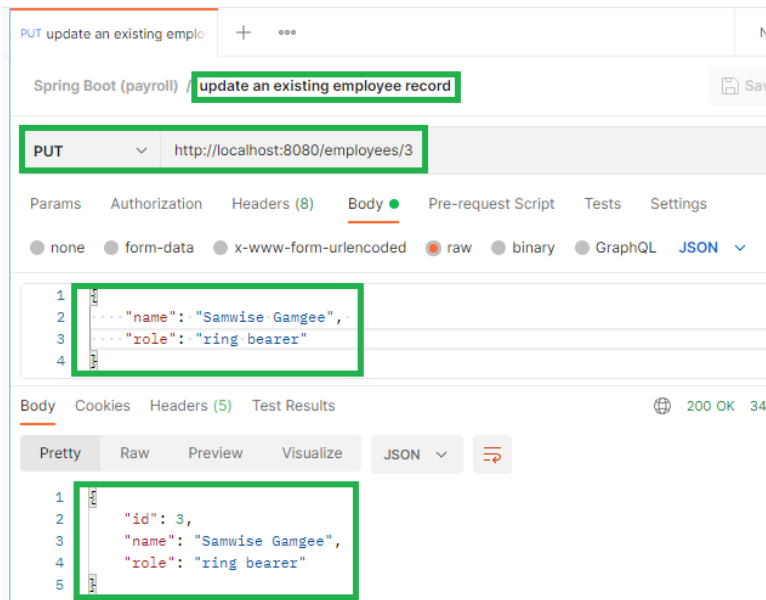
This request maps to the Controller class EmployeeController endpoint:

```
@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);
        });
}
```

Examine the response:

The response echoes back the record created showing a URL to find the record number by its record number 1 in this case.



DELETE Request: delete an employee record

Request to delete an existing record from the database. If the record does not exist, nothing happened, and no error message is seen.

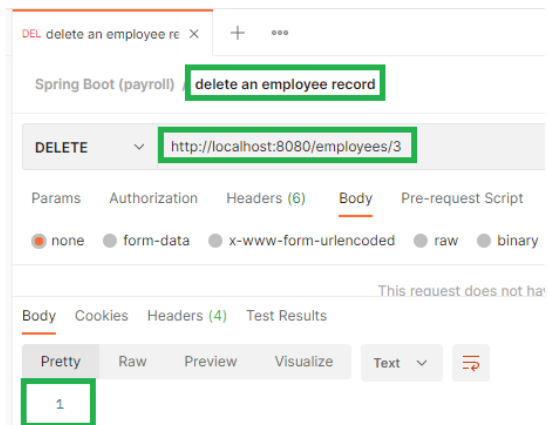
URL: "http://localhost:8080/employees/3"

This request maps to the Controller class EmployeeController endpoint:

```
@DeleteMapping("/employees/{id}")
void deleteEmployee(@PathVariable Long id) {
    repository.deleteById(id);
}
```

Examine the response:

The response does not give a good indication that the record was delete, it just sends a 1.



If you run the "GET all employee records" request, the results show record 3 is not in the list.





## Project Conclusion

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.

### “What makes something RESTful?”

So far, you have a web-based service that handles the core operations involving employee data. But that’s not enough to make things “RESTful”.

- Pretty URLs like `/employees/3` aren’t REST.
- Merely using `GET`, `POST`, etc. isn’t REST.
- Having all the CRUD operations laid out isn’t REST.

In fact, what we have built so far is better described as **RPC (Remote Procedure Call)**. That’s because there is no way to know how to interact with this service. If you published this today, you’d also have to write a document or host a developer’s portal somewhere with all the details.

This statement of Roy Fielding’s may further lend a clue to the difference between **REST** and **RPC**:

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today’s example is the SocialSite REST API. That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating.

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?

— Roy Fielding  
<https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

The side effect of NOT including hypermedia in our representations is that clients MUST hard code URIs to navigate the API.

This leads to the same brittle nature that predated the rise of e-commerce on the web. It’s a signal that our JSON output needs a little help.”

-- [Building REST services with Spring](#)