Internet Applications Design and Implementation 2017/2018 (Lab 7: Java Spring)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Leitão (jc.leitao@fct.unl.pt)
João Costa Seco (joao.seco@fct.unl.pt)

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Lab7 Goal:

- Introduction to Java Spring.
- Example and Running the Example.
- Build your REST endpoints for the project.

Java Spring

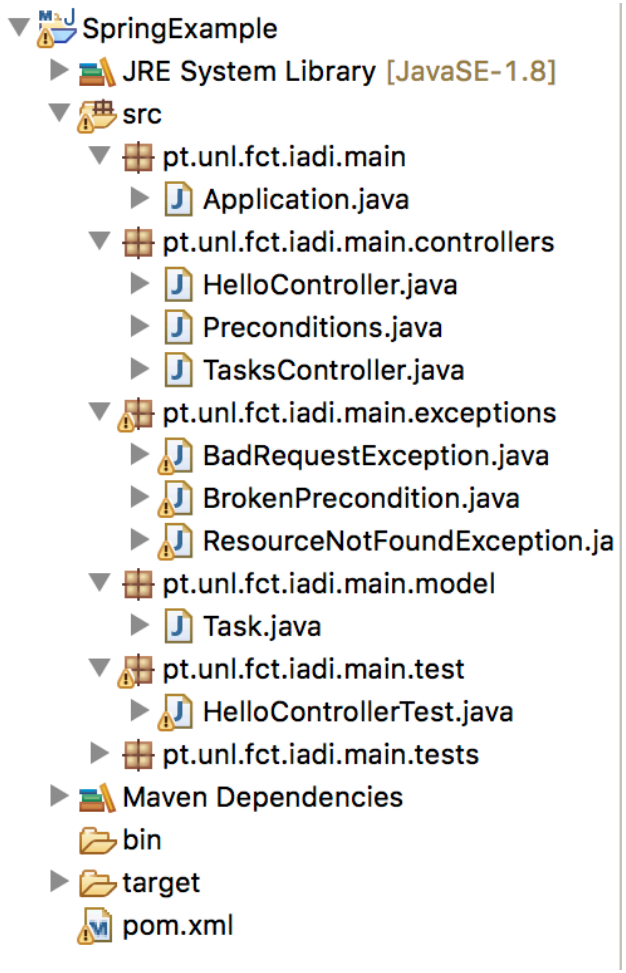
- De facto standard for building (complex) web services at the industry level.
- Based on Components.
- Uses Annotations.
- Wiring of Components is managed by the framework it self.
- Generates and Injects code to achieve your goals with minimal effort.
- Check more here: https://spring.io

Spring by example…

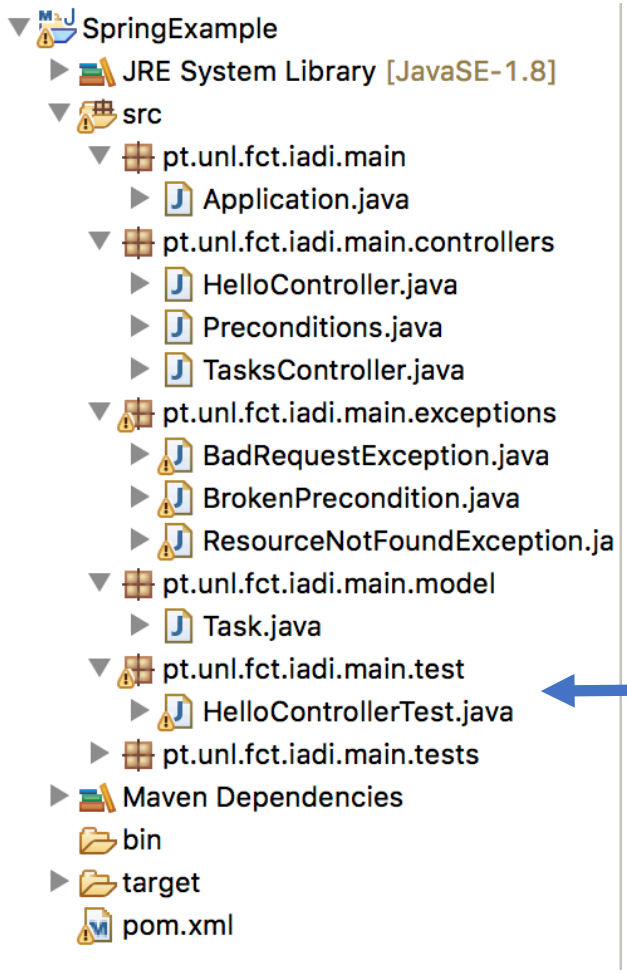
- Go to the public repository of CIAI:
- https://bitbucket.org/costaseco/ciai-1718-public

- Update the repository.
- You should have a “Project” named SpringExample
- Import it into your favorite Java IDE
- Convert it to a Maven project.

Spring by example…

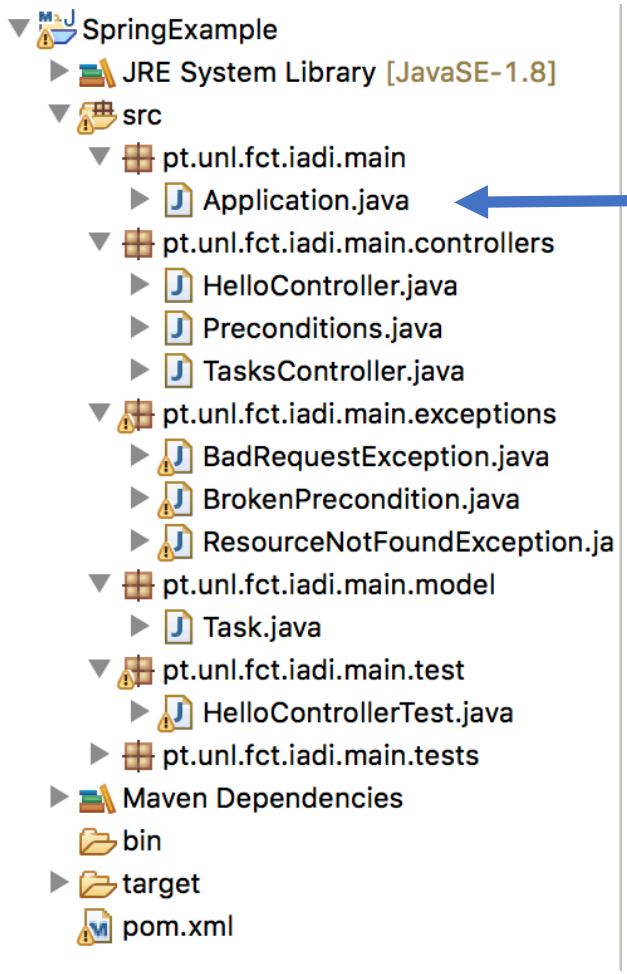


Spring by example…



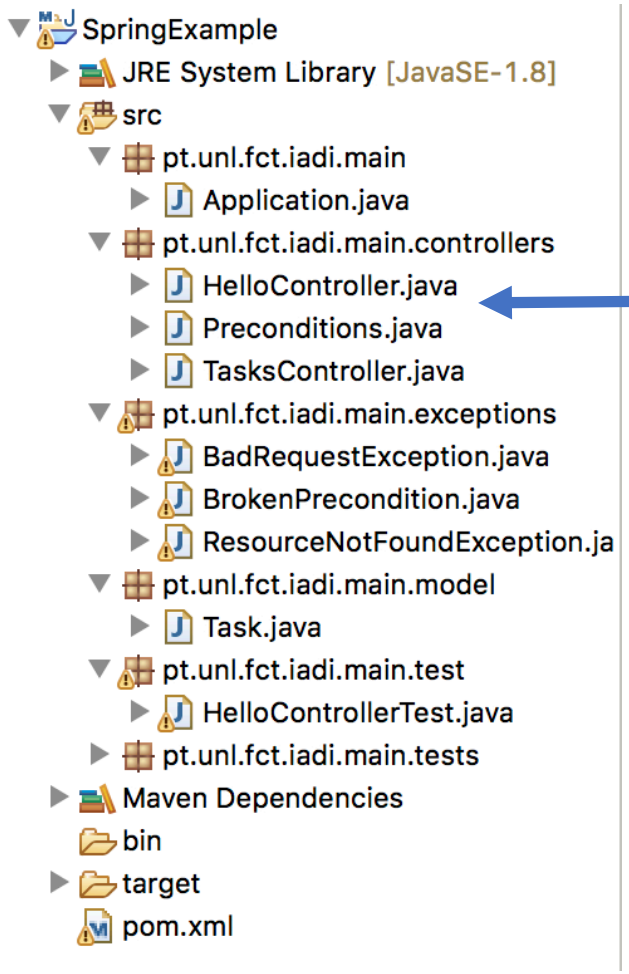
These are test tools… we are not going to discuss them today…

Spring by example…



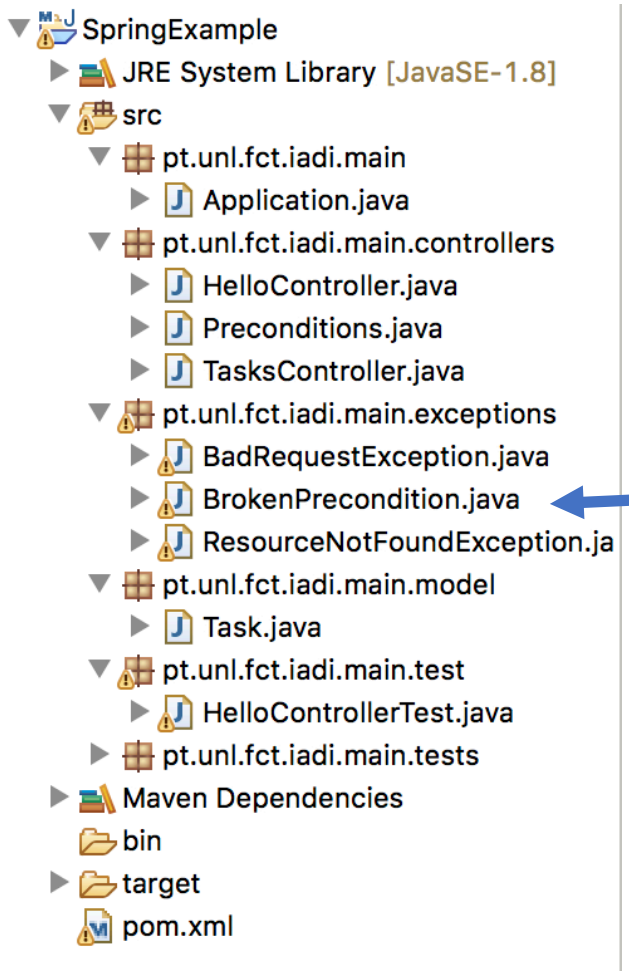
Here is where your application starts...

Spring by example…



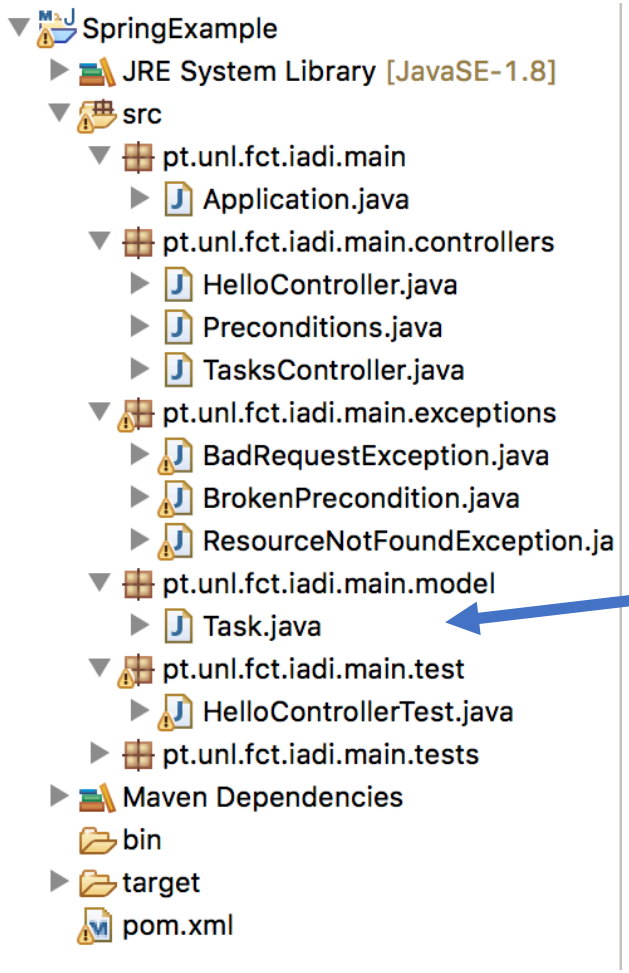
These are Controllers that expose a REST interface, and an additional class that exports static methods to validate inputs of your Controllers.

Spring by example…



These are Exceptions that can be thrown by your Controllers (we will get back to them in a bit)

Spring by example…



Finally this is the class that materializes the data that your application is going to manipulate (in this case we are only manipulating Tasks)

Spring by example…


```java
package pt.unl.fct.iadi.main.model;

import pt.unl.fct.iadi.main.exceptions.BrokenPrecondition;

public class Task {
    int id;
    String description;
    Date creationDate;
    Date dueDate;

    public Task() {}

    public Task(int id, String description, Date creationDate, Date dueDate) {
        this.id = id;
        this.description = description;
        this.creationDate = creationDate;
        this.dueDate = dueDate;
    }

    public int getId() {
        return id;
    }

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

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Date getCreationDate() {
        return creationDate;
    }

    public void setCreationDate(Date creationDate) {
        this.creationDate = creationDate;
    }

    public Date getDueDate() {
        return dueDate;
    }

    public void setDueDate(Date dueDate) {
        this.dueDate = dueDate;
    }

    public static void valid(Task t) {
        if( t.getId() == 0 ||
            t.getDescription() == null ||
            t.getCreationDate() == null ) {
            // can also tests dueDate >= creationDate
            throw new BrokenPrecondition();
        }
    }
}
```

The Task class: This is a very *simple class*, nothing of special here, except that you **must ensure** that there is an **empty constructor**, and that you have **getters and setters** for all class variables.

Spring by example…


```
1  package pt.unl.fct.iadi.main;
2
3  import java.util.Arrays;
10
11 // Inspired in: https://spring.io/guides/gs/spring-boot/
12
13 @SpringBootApplication
14 public class Application {
15
16     public static void main(String[] args) {
17         SpringApplication.run(Application.class, args);
18     }
19
20     @Bean
21     public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
22         return args -> {
23
24             System.out.println("Let's inspect the beans provided by Spring Boot:");
25
26             String[] beanNames = ctx.getBeanDefinitionNames();
27             Arrays.sort(beanNames);
28             for (String beanName : beanNames) {
29                 System.out.println(beanName);
30             }
31         };
32     }
33
34 }
```

The Application class (where everything begins)

Spring by example…


```java
package pt.unl.fct.iadi.main;

import java.util.Arrays;

// Inspired in: https://spring.io/guides/gs/spring-boot/

@SpringBootApplication
public class Application {

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

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {

            System.out.println("Let's inspect the beans provided by Spring Boot:");

            String[] beanNames = ctx.getBeanDefinitionNames();
            Arrays.sort(beanNames);
            for (String beanName : beanNames) {
                System.out.println(beanName);
            }
        };
    }

}
```

@SpringBootApplication:

Actually a meta annotation that includes various anotations.

This actually defines the start of your application. It will trigger the scan of Controllers (through annotations) that will be made available alongside the application.

The Application class (where everything begins)

Spring by example…


```java
package pt.unl.fct.iadi.main;

import java.util.Arrays;

// Inspired in: https://spring.io/guides/gs/spring-boot/

@SpringBootApplication
public class Application {

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

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {

            System.out.println("Let's inspect the beans provided by Spring Boot:");

            String[] beanNames = ctx.getBeanDefinitionNames();
            Arrays.sort(beanNames);
            for (String beanName : beanNames) {
                System.out.println(beanName);
            }
        };
    }

}
```

@SpringBootApplication:

Actually a meta annotation that includes various anotations.

This actually defines the start of your application. It will trigger the scan of Controllers (through annotations) that will be made available alongside the application.

The Application class (where everything begins)

Spring by example…


```
1  package pt.unl.fct.iadi.main;
2
3  import java.util.Arrays;
10
11 // Inspired in: https://spring.io/guides/gs/spring-boot/
12
13 @SpringBootApplication
14 public class Application {
15
16     public static void main(String[] args) {
17         SpringApplication.run(Application.class, args);
18     }
19
20     @Bean
21     public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
22         return args -> {
23
24             System.out.println("Let's inspect the beans provided by Spring Boot:")
25
26             String[] beanNames = ctx.getBeanDefinitionNames();
27             Arrays.sort(beanNames);
28             for (String beanName : beanNames) {
29                 System.out.println(beanName);
30             }
31         };
32     }
33
34 }
```

This function (which is tagged with the @Bean annotation) will, at startup, print to the console/log of your application all the Beans currently being used by your application.

Most of these are generated automatically.

The Application class (where everything begins)

Spring by example…


```
1  package pt.unl.fct.iadi.main;
2
3  import java.util.Arrays;
10
11 // Inspired in: https://spring.io/guides/gs/spring-boot/
12
13 @SpringBootApplication
14 public class Application {
15
16     public static void main(String[] args) {
17         SpringApplication.run(Application.class, args);
18     }
19
20     @Bean
21     public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
22         return args -> {
23
24             System.out.println("Let's inspect the beans provided by Spring Boot:")
25
26             String[] beanNames = ctx.getBeanDefinitionNames();
27             Arrays.sort(beanNames);
28             for (String beanName : beanNames) {
29                 System.out.println(beanName);
30             }
31         };
32     }
33
34 }
```

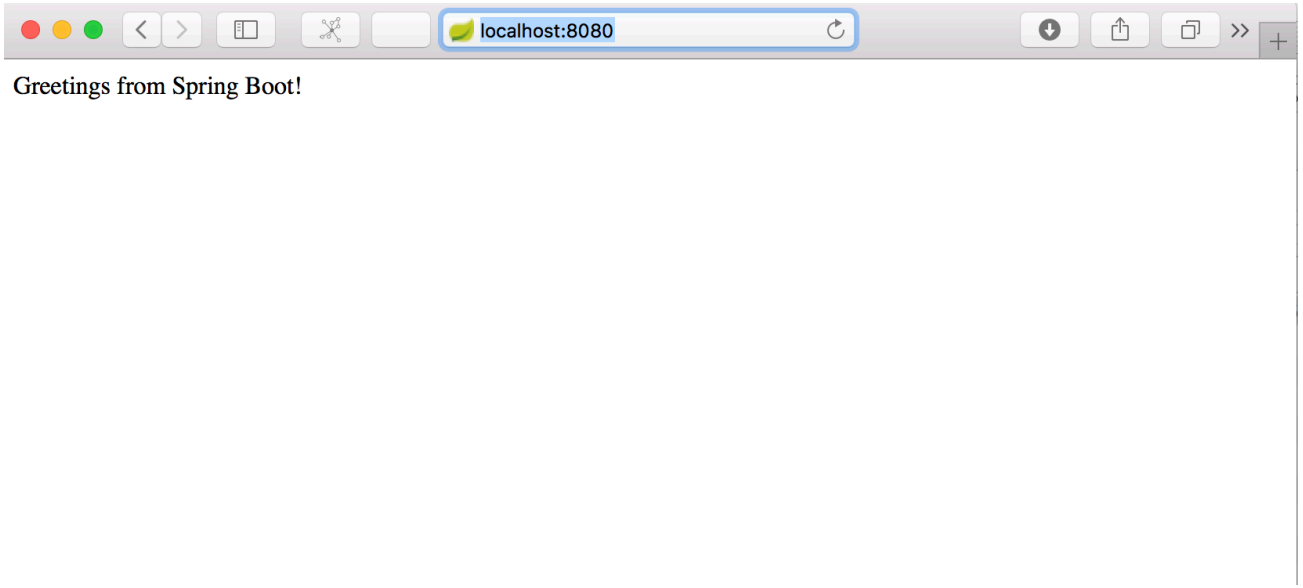
This function (which is tagged with the @Bean annotation) will, at startup, print to the console/log of your application all the Beans currently being used by your application.

Most of these are generated automatically.

The Application class (where everything begins)

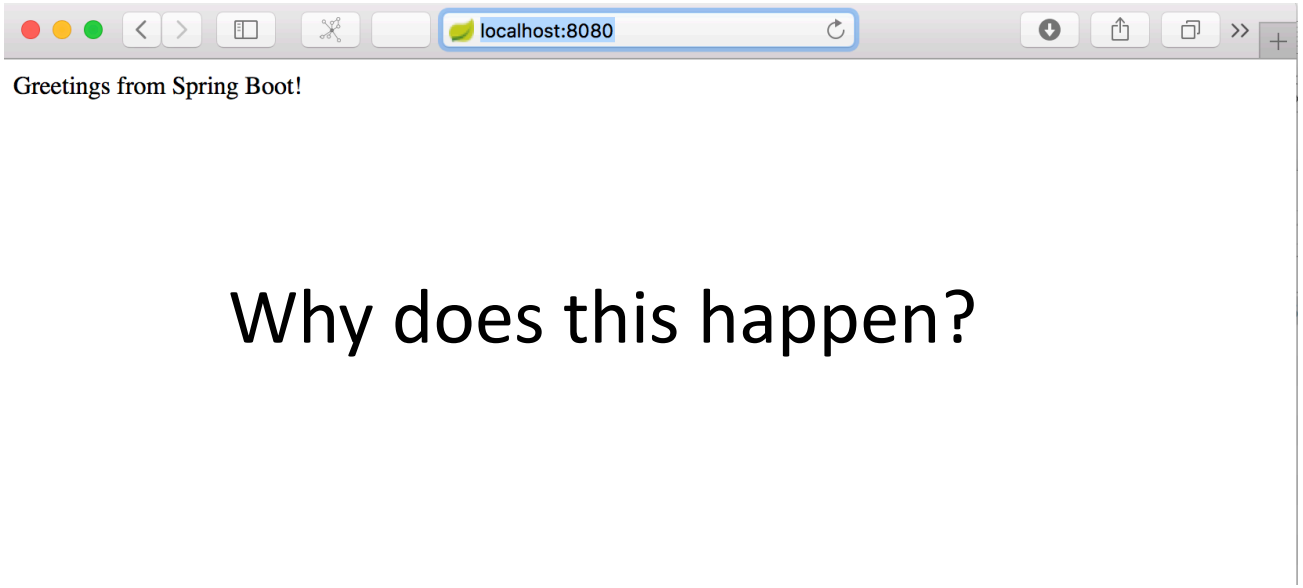
Spring by example…

- Go ahead and Run the application… (even within eclipse or your IDE).
- After that if you point your browser to: http://localhost:8080 you will get:



Spring by example…

- Go ahead and Run the application… (even within eclipse or your IDE).
- After that if you point your browser to: http://localhost:8080 you will get:



Why does this happen?

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

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

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!\n";
    }

}
```

The HelloController class:

- Notice that you have a class tagged with the **RestController** annotation in your project.

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

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

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!\n";
    }

}
```

The HelloController class:

- This annotation provides information to the Spring framework that this class is providing a REST interface.

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

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

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!\n";
    }

}
```

The HelloController class:

- The @RequestMapping(“/”) annotation indicates that this is what is executed by your application when you access the root of it.

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.RestController;
5
6  @RestController
7  public class HelloController {
8
9      @RequestMapping("/")
10     public String index() {
11         return "Greetings from Spring Boot!\n";
12     }
13
14 }
```

This is similar to the @Path annotation in Jersey.

The HelloController class:

- The @RequestMapping(“/”) annotation indicates that this is what is executed by your application when you access the root of it.

Spring by example…

- You have another @RestController class in your project… there is where more interesting things are made...

- This is the TasksController class.

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

import org.springframework.web.bind.annotation.*;

// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    // Using a simple storage for the purpose of illustrating a controller
    // The storage and manipulation of tasks will afterwards be factorized
    // in a Service class
    HashMap<Integer,Task> tasks = new HashMap<>();
    private int tasksSequence = 0;


    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.values().toArray(new Task[]{})
                :
                tasks
                .values()
                .stream()
                .filter(t -> t.getDescription().contains(search))
                .toArray(Task[]::new);
    }

```

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapping(value="", method= RequestMethod.GET)
22     Task[] getAll(@RequestParam(required=false, value="") String search) {
23         return search == null || search.equals("") // just in case
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

@RestController class

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapping(value="", method= RequestMethod.GET)
22     Task[] getAll(@RequestParam(required=false, value="") String search) {
23         return search == null || search.equals("") // just in case
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

@RequestMapping(value=“/tasks”)

This annotation associates all endpoints with /tasks in your application to methods of this Controller…

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

import org.springframework.web.bind.annotation.*;

// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    // Using a simple storage for the purpose of illustrating a controller
    // The storage and manipulation of tasks will afterwards be factorized

    HashMap<Integer,Task> tasks = new HashMap<>();
    private int tasksSequence = 0;

    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.values().toArray(new Task[]{})
                :
                tasks
                .values()
                .stream()
                .filter(t -> t.getDescription().contains(search))
                .toArray(Task[]::new);
    }

}
```

State is being stored in memory

Real application will rely on a Database for this purpose... This is not safe since the framework might execute multiple instances of the Controller (that will not share memory)

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapping(value="", method= RequestMethod.GET)
22                                                          ring search) {
23         return search == null || search.equals("") // just in case
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

RequestMapping(**value=“”,** method=RequestMethod.GET)

This value here is again a bidding between endpoint address and this method.
It is relative to the class binding, so empty means that this is associated to: ”/tasks”

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapping(value="", method=RequestMethod.GET)
22     Task[] getAll(@RequestParam(required=false, value="") String search) {
23         return search == null || search.equals("") // just in case
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

@RequestPara(required=false,
value=“”)

This associates a binding between the na input variable and a parameter of the endpoint.

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapping(value="", method=RequestMethod.GET)
22     Task[] getAll(@RequestParam(required=false, value="") String search) {
23         return search == null || search.equals("") // just in case
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

@RequestPara(**required=false**, value="")

This associates a binding between the na input variable and a parameter of the endpoint.

This option denotes that the parameter is not mandatory.

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapp
22     Task[] getAl(@RequestParam(required=false, value="") String search) {
23         return
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

@RequestPara(required=false, **value=“”**)

This associates a binding between the na input variable and a parameter of the endpoint.

This option denotes the default value if none is specified.

Spring by example…


```
1  package pt.unl.fct.iadi.main.controllers;
2
3  import org.springframework.web.bind.annotation.*;
7
8  // Inspired in: https://spring.io/guides/gs/rest-service/
9
10 @RestController
11 @RequestMapping(value="/tasks")
12 public class TasksController {
13
14     // Using a simple storage for the purpose of illustrating a controller
15     // The storage and manipulation of tasks will afterwards be factorized
16     // in a Service class
17     HashMap<Integer,Task> tasks = new HashMap<>();
18     private int tasksSequence = 0;
19
20
21     @RequestMapping(value="", method= RequestMethod.GET)
22     Task[] getAll(@RequestParam(required=false, value="") String search) {
23         return search == null || search.equals("") // just in case
24                 ?
25                 tasks.values().toArray(new Task[]{})
26                 :
27                 tasks
28                 .values()
29                 .stream()
30                 .filter(t -> t.getDescription().contains(search))
31                 .toArray(Task[]::new);
32     }
33
```

String search

This is the name of the query parameter supported by this endpoint.

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

import org.springframework.web.bind.annotation.*;

// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    // Using a simple storage for the purpose of illustrating a controller
    // The storage and manipulation of tasks will afterwards be factorized
    // in a Service class
    HashMap<Integer,Task> tasks = new HashMap<>();
    private int tasksSequence = 0;


    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.values().toArray(new Task[]{})
                :
                tasks
                .values()
                .stream()
                .filter(t -> t.getDescription().contains(search))
                .toArray(Task[]::new);
    }

```

String search

You can access this enpoint using:

http://localhost:8080/tasks

Or

http://localhost:8080/tasks?search=myQuery

Spring by example…


```java
package pt.unl.fct.iadi.main.controllers;

import org.springframework.web.bind.annotation.*;

// Inspired in: https://spring.io/guides/gs/rest-service/

@RestController
@RequestMapping(value="/tasks")
public class TasksController {

    // Using a simple storage for the purpose of illustrating a controller
    // The storage and manipulation of tasks will afterwards be factorized
    // in a Service class
    HashMap<Integer,Task> tasks = new HashMap<>();
    private int tasksSequence = 0;


    @RequestMapping(value="", method= RequestMethod.GET)
    Task[] getAll(@RequestParam(required=false, value="") String search) {
        return search == null || search.equals("") // just in case
                ?
                tasks.values().toArray(new Task[]{})
                :
                tasks
                .values()
                .stream()
                .filter(t -> t.getDescription().contains(search))
                .toArray(Task[]::new);
    }

```

String search

You can access this enpoint using:

http://localhost:8080/tasks

Or

http://localhost:8080/tasks?search=myQuery

Spring by example…


```
34
35      @RequestMapping(value="", method = RequestMethod.POST)
36      void createTask(@RequestBody Task t) {
37          t.setId(++tasksSequence);
38          Task.valid(t);
39
40          tasks.put(t.getId(), t);
41      }
42
```

This method is similar to the previous one, it is accessible in the same endpoint address (i.e, URL, but it does not support query parameter) but it is for the REST method **POST.**

Spring by example…


```
34
35      @RequestMapping(value="", method = RequestMethod.POST)
36      void createTask(@RequestBody Task t) {
37          t.setId(++tasksSequence);
38          Task.valid(t);
39
40          tasks.put(t.getId(), t);
41      }
42
```

As it is common on POST methods, it does get a input through the body of the HTML request.

Spring by example…


```
34
35      @RequestMapping(value="", method = RequestMethod.POST)
36      void createTask(@RequestBody Task t) {
37          t.setId(++tasksSequence);
38          Task.valid(t);
39
40          tasks.put(t.getId(), t);
41      }
42
```

This is a validation to check if the input is valid…

If the input is invalid it will throw an Exception…

Spring by example…


```
34
35      @RequestMapping(value="", method = RequestMethod.POST)
36
37          public static void valid(Task t) {
38              if( t.getId() == 0 ||
39                  t.getDescription() == null ||
40                  t.getCreationDate() == null ) {
41                  // can also tests dueDate >= creationDate
42                  throw new BrokenPrecondition();
              }
          }
      }
```

This is a validation to check if the input is valid…

If the input is invalid it will throw an Exception…

Spring by example…


```
1  package pt.unl.fct.iadi.main.exceptions;
2
3  import org.springframework.http.HttpStatus;
5
6  @ResponseStatus(HttpStatus.PRECONDITION_FAILED)
7  public class BrokenPrecondition extends RuntimeException {
8      //
9  }
10
```

The interesting aspect of this exception is that it owns an Annotation @ResponseStatus

Spring by example…


```
1  package pt.unl.fct.iadi.main.exceptions;
2
3  import org.springframework.http.HttpStatus;
5
6  @ResponseStatus(HttpStatus.PRECONDITION_FAILED)
7  public class BrokenPrecondition extends RuntimeException {
8      //
9  }
10
```

The interesting aspect of this exception is that it owns an Annotation @ResponseStatus

This annotation informs the runtime of the HTTP response code to send if such an Exception is caught, in this case 412.

Spring by example…


```
43
44      @RequestMapping(value="/{id}", method = RequestMethod.GET)
45      Task showTask(@PathVariable int id) {
46          Task t = tasks.get(id);
47          Preconditions.checkFound(t);
48
49          return t;
50      }
51
```

This method is another instance of a GET method, but this time with a different URL associated.

The value component of the @RequestMapping states /{id}.

Id is a special value that will be associated with the input parameter id (notice the @PathVariable Annotation).

Spring by example…


```
43
44      @RequestMapping(value="/{id}", method = RequestMethod.GET)
45      Task showTask(@PathVariable int id) {
46          Task t = tasks.get(id);
47          Preconditions.checkFound(t);
48
49          return t;
50      }
51
```

This method is another instance of a GET method, but this time with a different URL associated.

The value component of the @RequestMapping states /{id}.

Id is a special value that will be associated with the input parameter id (notice the @PathVariable Annotation).

Spring by example…


```
43
44      @RequestMapping(value="/{id}", method = RequestMethod.GET)
45      Task showTask(@PathVariable int id) {
46          Task t = tasks.get(id);
47          Preconditions.checkFound(t);
48
49          return t;
50      }
51
```

If you want to access the Task with id 100, you would use this endpoint in the following manner:

http://localhost:8080/tasks/100

Spring by example…


```
52
53      @RequestMapping(value="/{id}", method = RequestMethod.POST)
54      void updateTask(@PathVariable int id, @RequestBody Task t) {
55          Preconditions.checkCondition(t.getId()==id);
56          Task t2 = tasks.get(id);
57          Preconditions.checkFound(t2);
58          Task.valid(t);
59
60          tasks.put(id, t); // updates the existing task with the request's info
61      }
62
```

Nothing new here… although we are using static methods of the Preconditions class to check some aspects of the input.

These static methods throw (annotated) Exceptions when some condition fails.

Spring by example…


```
63
64      @RequestMapping(value="/{id}", method = RequestMethod.DELETE)
65      void deleteTask(@PathVariable int id) {
66          Task t = tasks.get(id);
67          Preconditions.checkFound(t);
68
69          tasks.remove(id);
70      }
71  }
72
```

Spring by example…

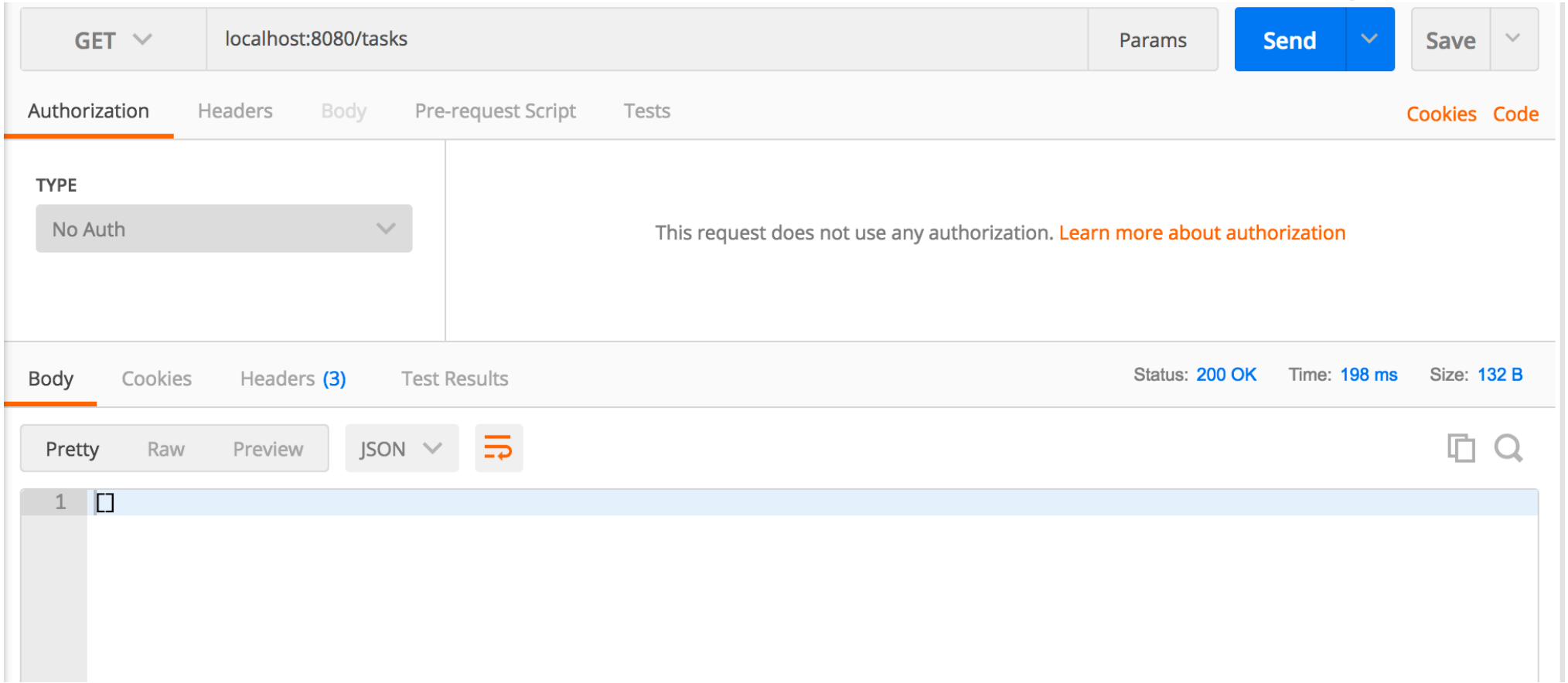

```
63
64      @RequestMapping(value="/{id}", method = RequestMethod.DELETE)
65      void deleteTask(@PathVariable int id) {
66          Task t = tasks.get(id);
67          Preconditions.checkFound(t);
68
69          tasks.remove(id);
70      }
71  }
72
```

Another endpoint, this time for the REST method DELETE.

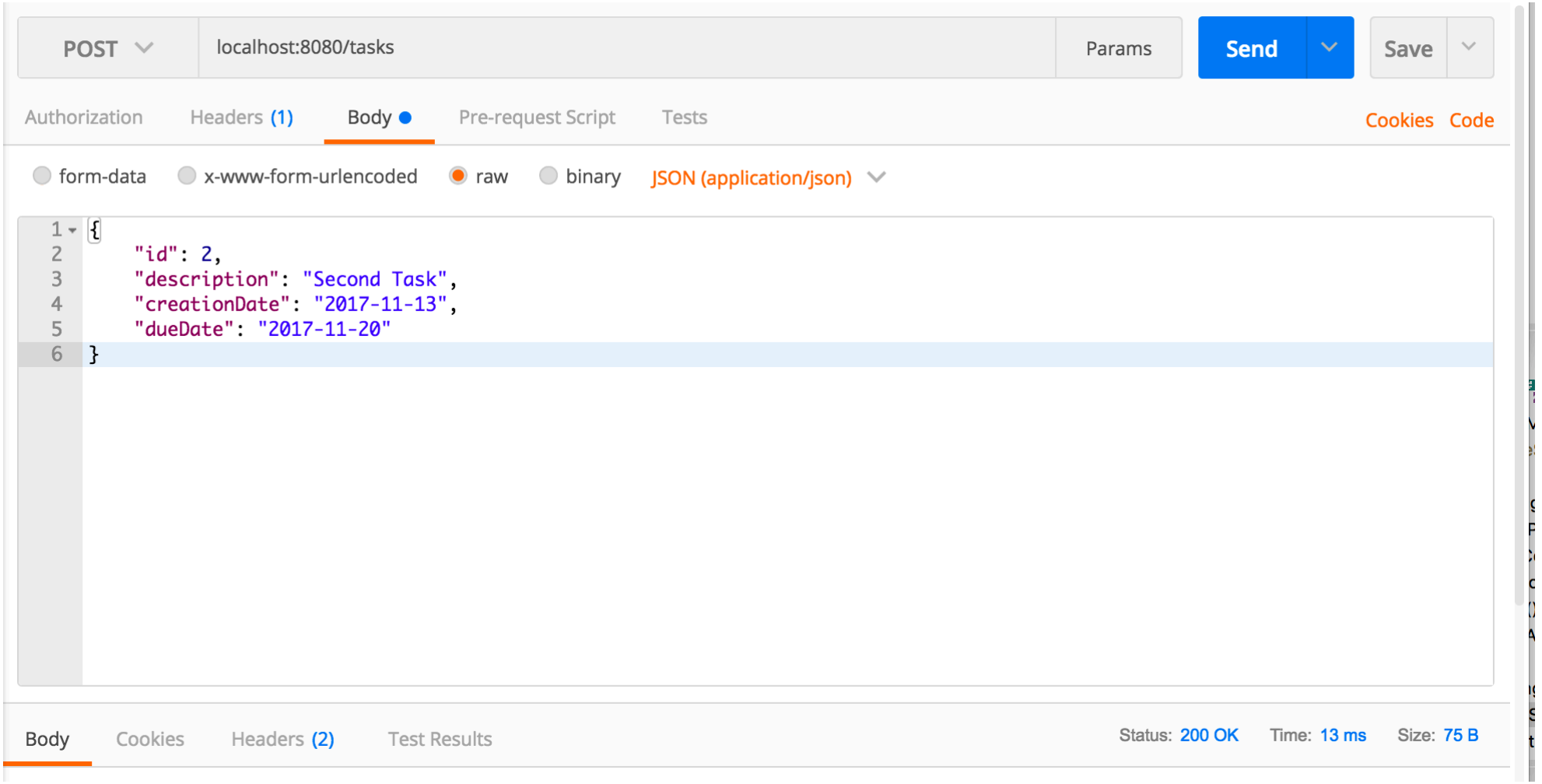
Spring by example…

- Ok… lets Test this...
- Put the Application running (if it not running yet) and lets exercise the endpoints presented before.
- Sugestion: Use Postman to do these tests...
- (Here are a few examples)

Spring by example…



Spring by example…



Spring by example…

- Ok… lets Test this...

- Put the Application running (if it not running yet) and lets exercise the endpoints presented before.

- Sugestion: Use Postman to do these tests...
- Play with it for a while:
  - Violate Preconditions, such as data format in the json of a Post, or ask for some tasks that does not exists...

Reminder of the Class

- Project...

- In your project you will need to create multiple REST endpoints to support the operation of your application.

- Use this example to start doing that.
- Focus for instance in the ArtPieces...