Lab 1 - REST web service in Spring Boot

This lab takes you through the steps needed to create a REST web service in Spring Boot.

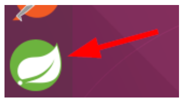
The user story throughout this Meetup is about a company named DroneBuzzers that wants to sell drone parts. In this lab, we will create a REST web service for retrieving info on drone parts and for submitting an order for drone parts.

The starting point for this lab is to have the provided VirtualBox machine up-and-running:

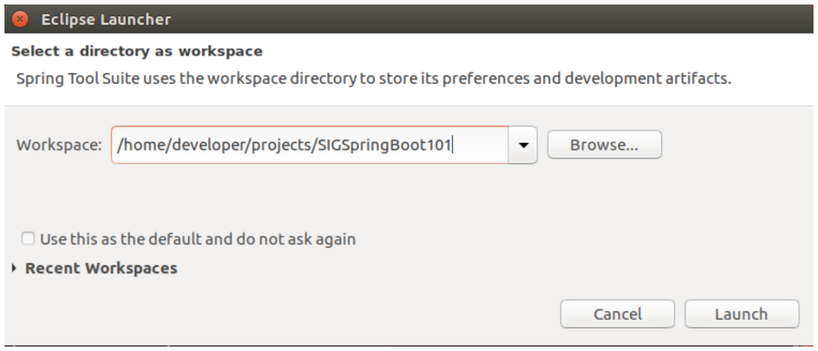
- You are logged in under user/password: developer/welcome01
- You have updated the labs running the `git pull` command in the lab workspace directory `/home/developer/projects/SIGSpringBoot101`

1. Start the development environment

To labs will use Eclipse STS (Spring Tool Suite). Start it by clicking the icon:



The dialogue for the workspace will start:



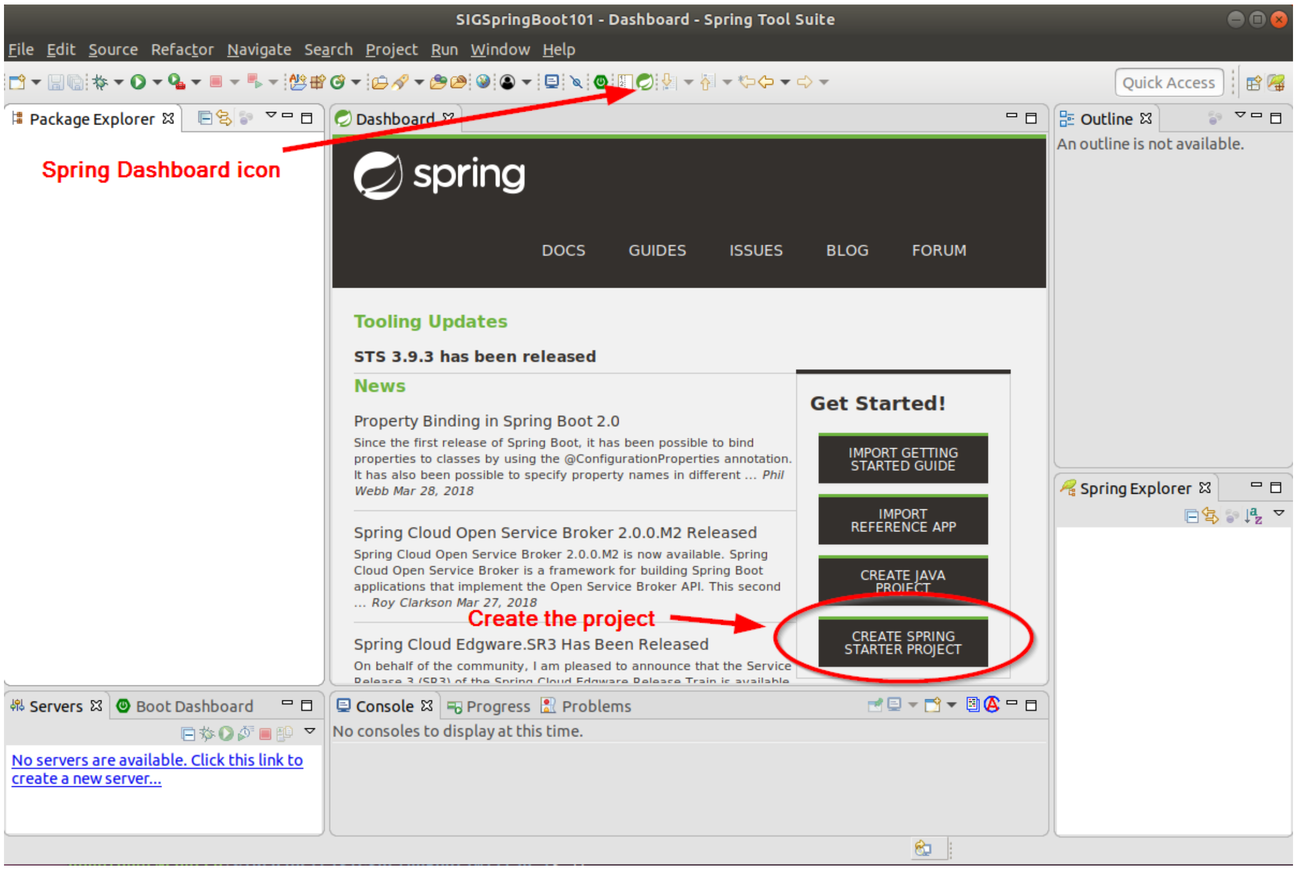
Select a directory as workspace

Spring Tool Suite uses the workspace directory to store its preferences and development artifacts.

Workspace: /home/developer/projects/SIGSpringBoot101

Verify that the workspace is set to `/home/developer/projects/SIGSpringBoot101` and click Launch

The resulting start-up screen should look somewhat like shown below:

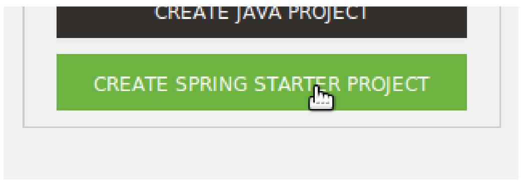

SIGSpringBoot101 - Dashboard - Spring Tool Suite
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Spring Dashboard icon
Dashboard
spring
DOCS GUIDES ISSUES BLOG FORUM
Tooling Updates
STS 3.9.3 has been released
News
Get Started!
IMPORT GETTING STARTED GUIDE
IMPORT REFERENCE APP
CREATE JAVA PROJECT
CREATE SPRING STARTER PROJECT
Create the project
Outline
An outline is not available.
Spring Explorer
Servers
Boot Dashboard
No servers are available. Click this link to create a new server...
Console
Progress
Problems
No consoles to display at this time.

Note the Spring Dashboard has its own icon. If you don't see the Spring Dashboard, click the icon to open it. Ensure that it is opened, as we'll create the project from there.

2. Create the project

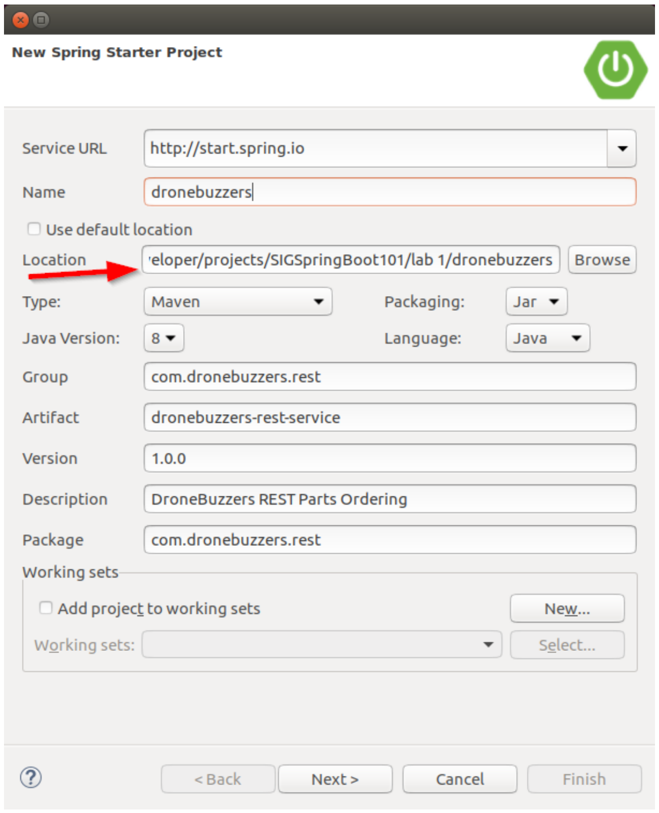
In this section, we'll create the eclipse project for our REST web service.

In the Spring Dashboard, click the 'CREATE SPRING STARTER PROJECT' button:



The wizard for a Spring Starter Project will pop up. Complete it like shown below:

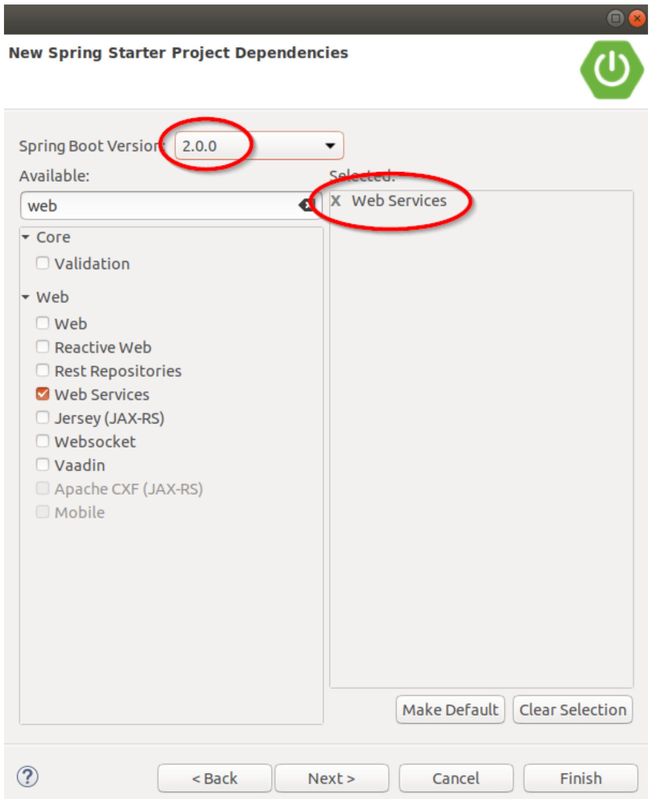
- Name: dronebuzzers:
- Location: /home/developer/projects/SIGSpringBoot101/lab 1/dronebuzzers
- Group: com.dronebuzzers.rest
- Artifact: dronebuzzers-rest-service
- Description: DroneBuzzers REST Parts Ordering
- Package: com.dronebuzzers.rest



Service URL: http://start.spring.io
Name: dronebuzzers
Location: 'eloper/projects/SIGSpringBoot101/lab 1/dronebuzzers
Type: Maven
Packaging: Jar
Java Version: 8
Language: Java
Group: com.dronebuzzers.rest
Artifact: dronebuzzers-rest-service
Version: 1.0.0
Description: DroneBuzzers REST Parts Ordering
Package: com.dronebuzzers.rest

Be careful to enter the appropriate location for your project … to avoid confusion later on …

Click Next and select the ‘Web Services’ dependency:


New Spring Starter Project Dependencies

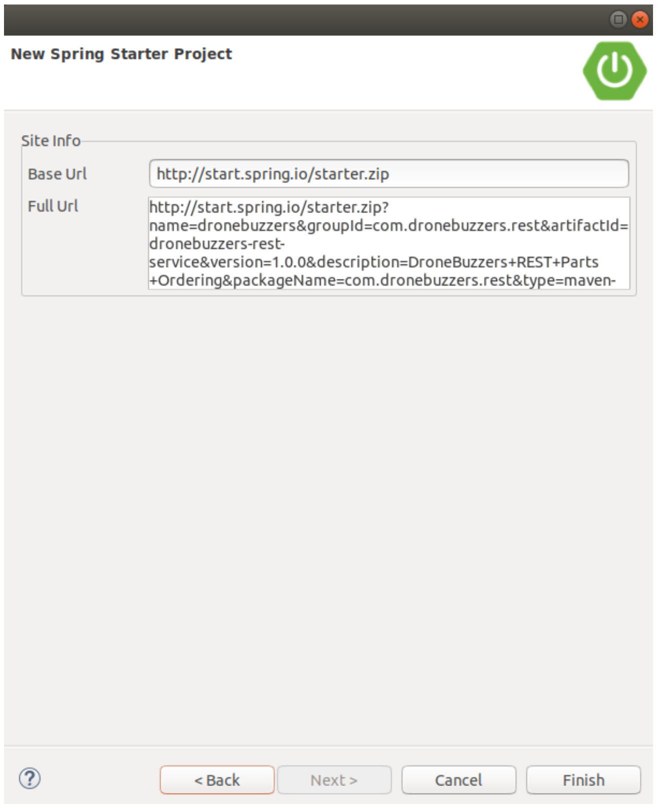
Spring Boot Version: 2.0.0

Available: web

Selected: X Web Services

- Core
  - [ ] Validation
- Web
  - [ ] Web
  - [ ] Reactive Web
  - [ ] Rest Repositories
  - [x] Web Services
  - [ ] Jersey (JAX-RS)
  - [ ] Websocket
  - [ ] Vaadin
  - [ ] Apache CXF (JAX-RS)
  - [ ] Mobile

Click Next and examine the Url that is used for retrieving the starter project:

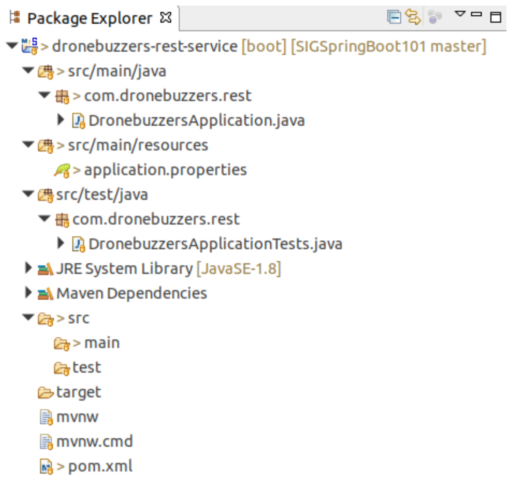


Site Info

Base Url: http://start.spring.io/starter.zip

Full Url:
```
http://start.spring.io/starter.zip?
name=dronebuzzers&groupId=com.dronebuzzers.rest&artifactId=
dronebuzzers-rest-
service&version=1.0.0&description=DroneBuzzers+REST+Parts
+Ordering&packageName=com.dronebuzzers.rest&type=maven-
```

Click Finish and have a look at the directory structure of the project that appeared in the Package Explorer:



```
▼ > dronebuzzers-rest-service [boot] [SIGSpringBoot101 master]
  ▼ > src/main/java
    ▼ > com.dronebuzzers.rest
      ▶ DronebuzzersApplication.java
  ▼ > src/main/resources
      > application.properties
  ▼ src/test/java
    ▼ com.dronebuzzers.rest
      ▶ DronebuzzersApplicationTests.java
  ▶ JRE System Library [JavaSE-1.8]
  ▶ Maven Dependencies
  ▼ > src
      > main
      test
    target
    mvnw
    mvnw.cmd
    > pom.xml
```

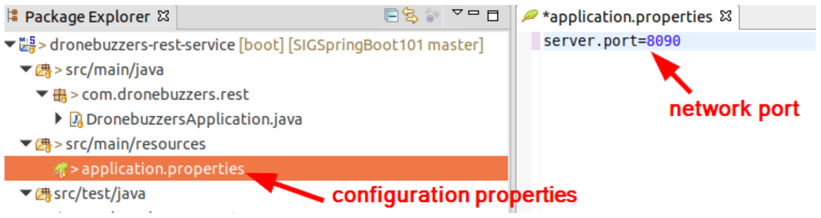
Your project has now been created.

3. Set the port for the service

This starter application project is the basis for a REST web service. As Spring Boot adheres to the Convention over Configuration principle, a lot of settings are not explicitly set: they have a sensible default value. For example, the network port at which our web service runs is by default 8080.

Now, if you would have an Oracle DB running on your development machine, port 8080 may be already occupied. Therefore, we will change the network port to 8090. This setting is made in theapplication.properties file like shown below:

- server.port=8090



```
server.port=8090
```

Save and close the application.properties file. Done.

4. First operation for the web service

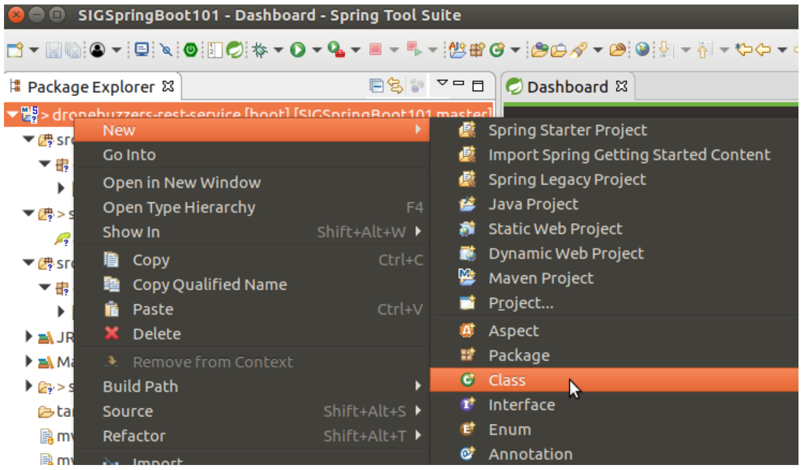
First, we will add an operation for retrieval of information of drone parts. In REST terminology, the resource will be ‘part’, and the http method will be GET.

The following steps will be done:

- Step 1: add a Java class for drone parts: Part
- Step 2: add a RestController Class for the ‘part’ resource
- Step 3: run and test with a browser

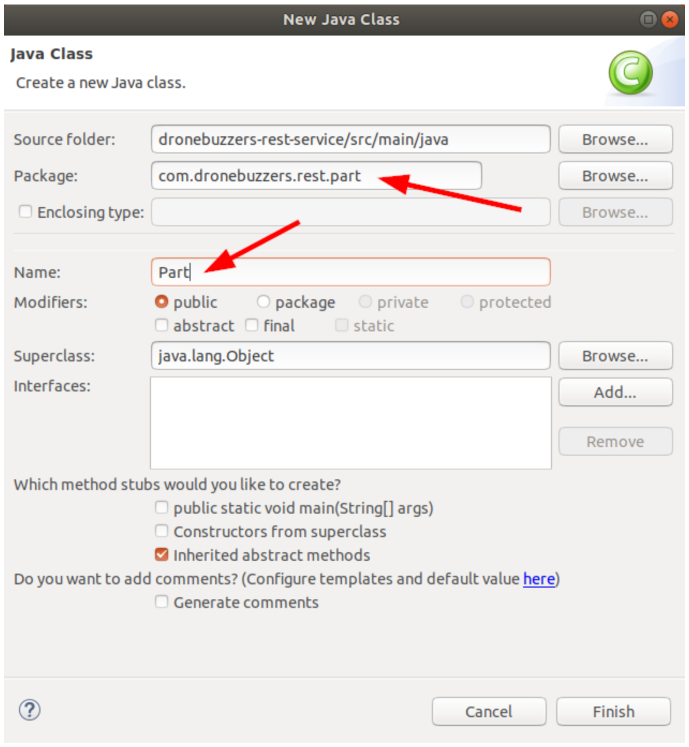
Step 1: add a Java class for drone parts: Part

Right-click on the project and create a new Class:



Set the package and name for the class as shown below:

- Package: com.dronebuzzers.rest.part
- Name: Part


New Java Class

Java Class

Create a new Java class.

Source folder: dronebuzzers-rest-service/src/main/java

Package: com.dronebuzzers.rest.part

☐ Enclosing type:

Name: Part

Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

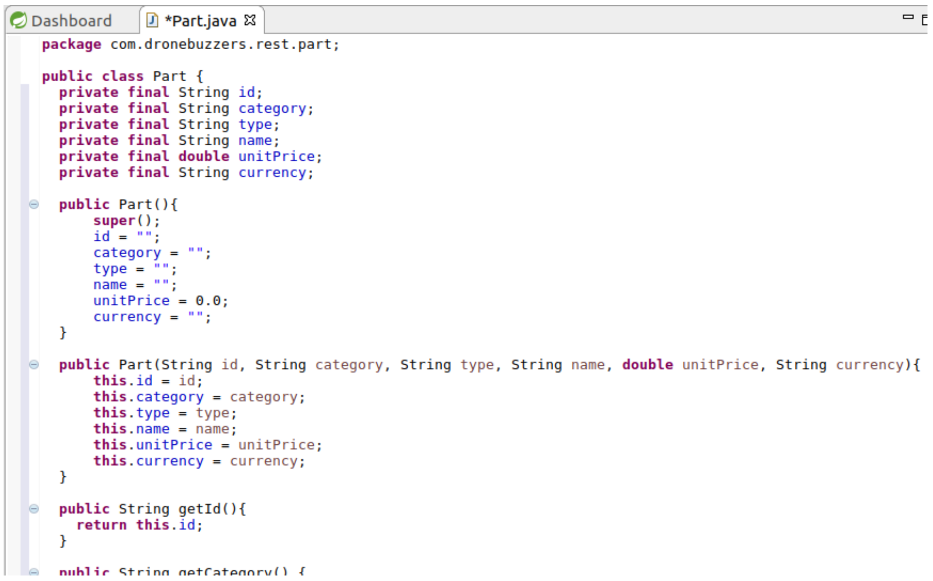
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

Click Finish

Replace the contents of the Part.java file with the contents that is in file:

```
/home/developer/projects/SIGSpringBoot101/lab 1/input/Part.java
```

The result should look like:



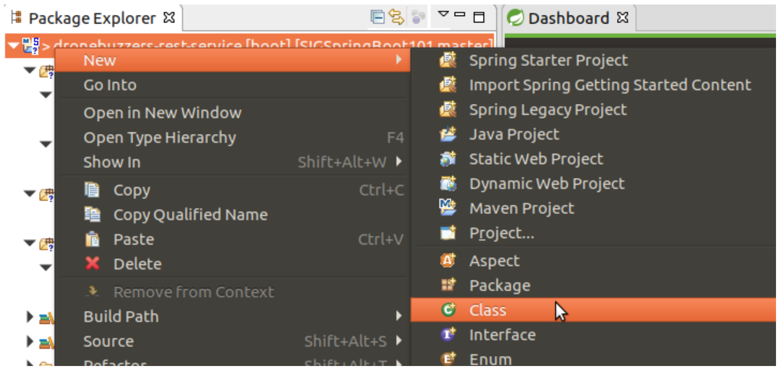
```
package com.dronebuzzers.rest.part;

public class Part {
  private final String id;
  private final String category;
  private final String type;
  private final String name;
  private final double unitPrice;
  private final String currency;

  public Part(){
      super();
      id = "";
      category = "";
      type = "";
      name = "";
      unitPrice = 0.0;
      currency = "";
  }

  public Part(String id, String category, String type, String name, double unitPrice, String currency){
      this.id = id;
      this.category = category;
      this.type = type;
      this.name = name;
      this.unitPrice = unitPrice;
      this.currency = currency;
  }

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

  public String getCategory() {
```

Save the file.

Step 2: add a RestController Class for the ‘part’ resource

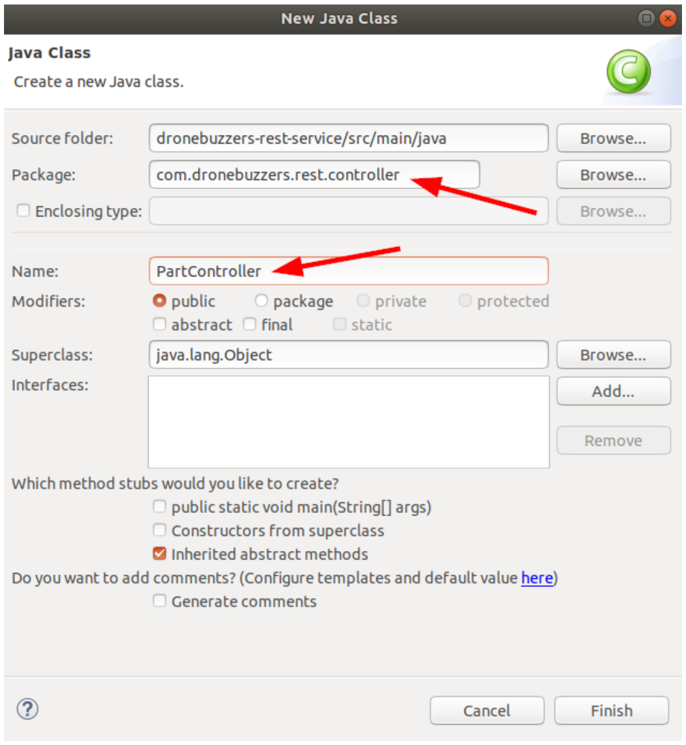
Now, we’ll create a RestController, i.e. a class that defines what http methods and endpoints will be exposed by the service.

Right-click on the project and create a new Class:



Set the package and name for the class as shown below:

- Package: com.dronebuzzers.rest.controller
- Name: PartController



Source folder: dronebuzzers-rest-service/src/main/java
Package: com.dronebuzzers.rest.controller
Enclosing type:
Name: PartController
Modifiers: [✓ public] package private protected
abstract final static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
[✓ Inherited abstract methods]
Do you want to add comments? (Configure templates and default value here)
Generate comments

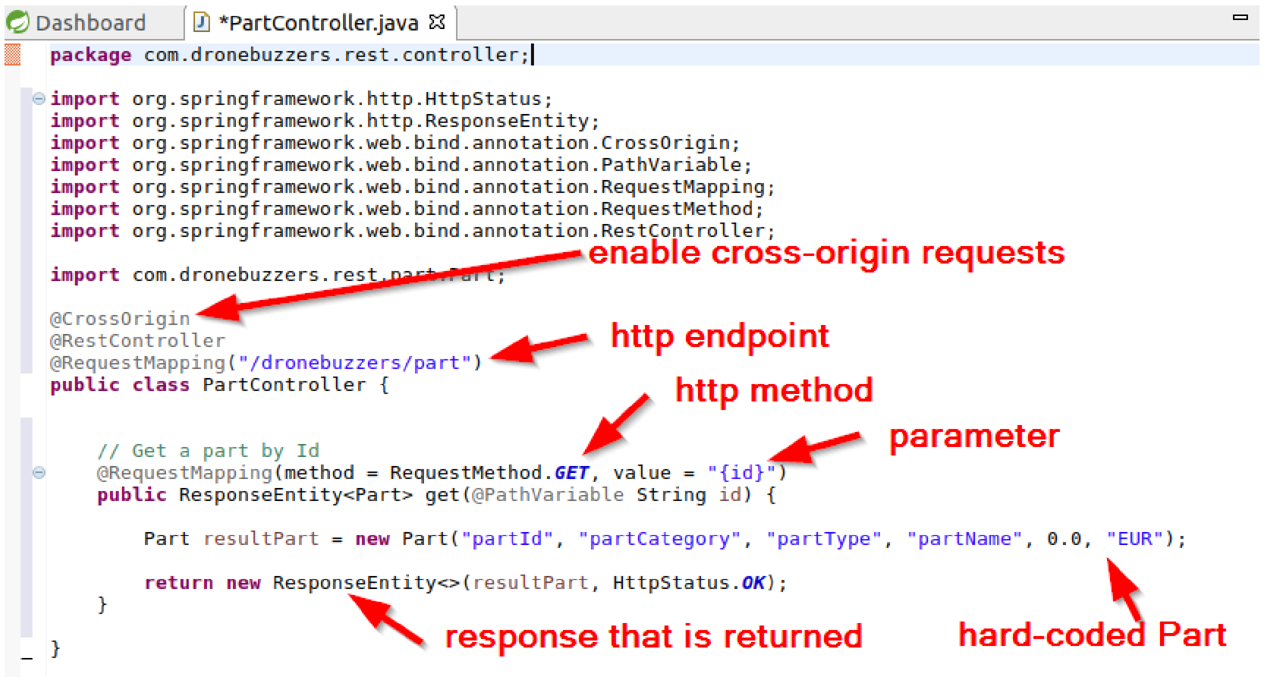
Click Finish

Replace the contents of the Part.java file with the contents that is in file:

```
/home/developer/projects/SIGSpringBoot101/lab 1/input/PartController-hardcoded.java
```

The file has the controller and the first service operation: it returns the details of a specific drone part. An example of a service call is http://localhost:8090/part/12, which returns the details for the drone part with part id 12. We will start with a hard coded result.

The result should look like:



```
package com.dronebuzzers.rest.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.dronebuzzers.rest.part.Part;

@CrossOrigin
@RestController
@RequestMapping("/dronebuzzers/part")
public class PartController {

    // Get a part by Id
    @RequestMapping(method = RequestMethod.GET, value = "{id}")
    public ResponseEntity<Part> get(@PathVariable String id) {

        Part resultPart = new Part("partId", "partCategory", "partType", "partName", 0.0, "EUR");

        return new ResponseEntity<>(resultPart, HttpStatus.OK);
    }

}
```

The above controller defines:

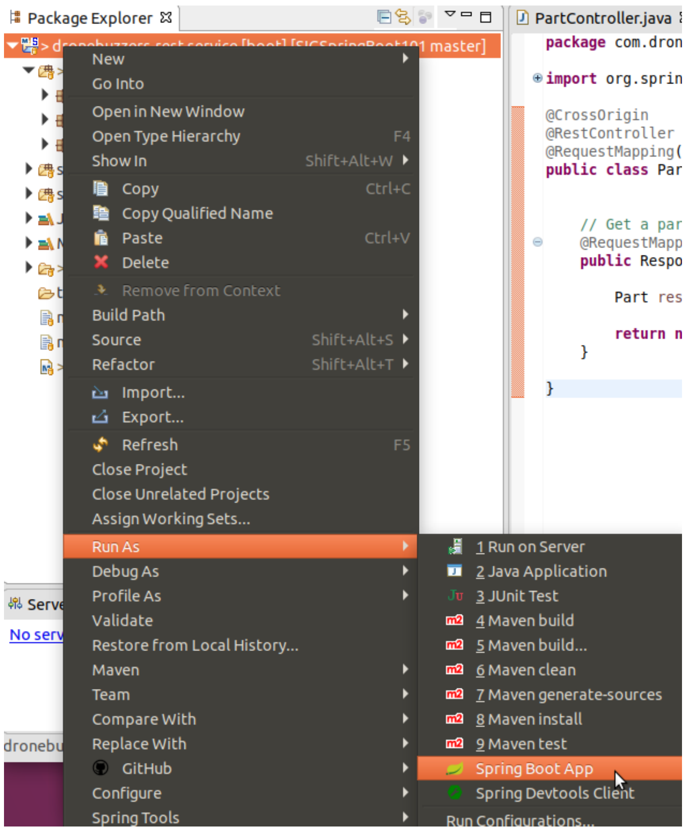
- an http GET operation with an input Path parameter called id
- the endpoint (in our case this adds up to): http://localhost:8090/dronebuzzers/part/{id}
- that returns a hard-coded drone part Part

Save the PartController.java file.

Step 3: run and test with a browser

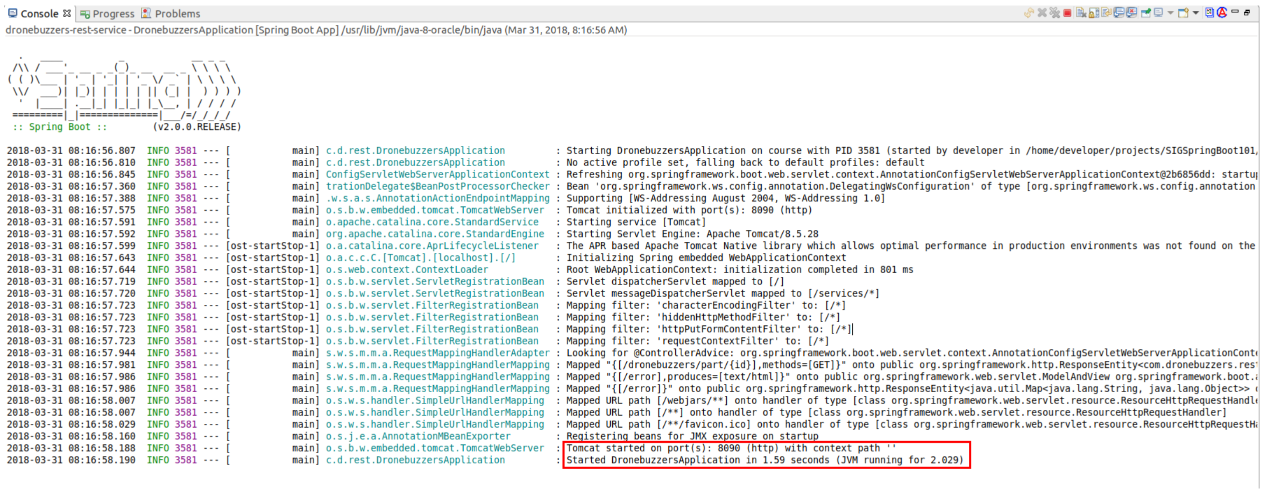
Now, it’s time to run the web service and test if it’s working.

To run the web service, right-click the project, goto ‘Run As’ and then select ‘Spring Boot App’.



```
package com.dron

import org.sprin

@CrossOrigin
@RestController
@RequestMapping(
public class Par

    // Get a par
    @RequestMapp
    public Respo

        Part res

        return n
    }

}
```

The resulting Console window will look like:

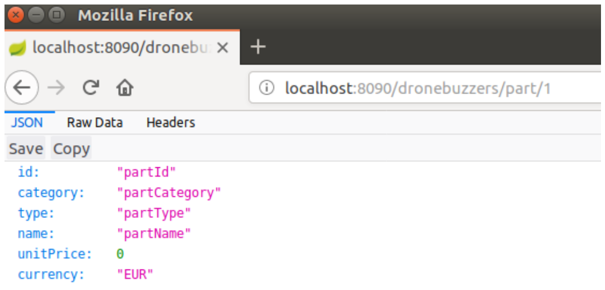


```
dronebuzzers-rest-service - DronebuzzersApplication [Spring Boot App] /usr/lib/jvm/java-8-oracle/bin/java (Mar 31, 2018, 8:16:56 AM)

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

2018-03-31 08:16:56.807  INFO 3581 --- [           main] c.d.rest.DronebuzzersApplication         : Starting DronebuzzersApplication on course with PID 3581 (started by developer in /home/developer/projects/SIGSpringBoot101,
2018-03-31 08:16:56.810  INFO 3581 --- [           main] c.d.rest.DronebuzzersApplication         : No active profile set, falling back to default profiles: default
2018-03-31 08:16:56.845  INFO 3581 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@2b6856dd: startu
2018-03-31 08:16:57.360  INFO 3581 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.ws.config.annotation.DelegatingWsConfiguration' of type [org.springframework.ws.config.annotation
2018-03-31 08:16:57.388  INFO 3581 --- [           main] .w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
2018-03-31 08:16:57.575  INFO 3581 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8090 (http)
2018-03-31 08:16:57.591  INFO 3581 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-03-31 08:16:57.592  INFO 3581 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.28
2018-03-31 08:16:57.599  INFO 3581 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the
2018-03-31 08:16:57.643  INFO 3581 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-03-31 08:16:57.644  INFO 3581 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 801 ms
2018-03-31 08:16:57.719  INFO 3581 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2018-03-31 08:16:57.720  INFO 3581 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet messageDispatcherServlet mapped to [/services/*]
2018-03-31 08:16:57.723  INFO 3581 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-03-31 08:16:57.723  INFO 3581 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-03-31 08:16:57.723  INFO 3581 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-03-31 08:16:57.723  INFO 3581 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-03-31 08:16:57.944  INFO 3581 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationConte
2018-03-31 08:16:57.981  INFO 3581 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/dronebuzzers/part/{id}],methods=[GET]}" onto public org.springframework.http.ResponseEntity<com.dronebuzzers.res
2018-03-31 08:16:57.986  INFO 3581 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.
2018-03-31 08:16:57.986  INFO 3581 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>>
2018-03-31 08:16:58.007  INFO 3581 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandl
2018-03-31 08:16:58.007  INFO 3581 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-31 08:16:58.029  INFO 3581 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestH
2018-03-31 08:16:58.160  INFO 3581 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2018-03-31 08:16:58.188  INFO 3581 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8090 (http) with context path ''
2018-03-31 08:16:58.190  INFO 3581 --- [           main] c.d.rest.DronebuzzersApplication         : Started DronebuzzersApplication in 1.59 seconds (JVM running for 2.029)
```

Notice at the bottom that the Tomcat server is running on the port we defined: 8090. The last line shows that it took only about 2 seconds to start the web service.

Next step is to open a browser and enter url http://localhost:8090/dronebuzzers/part/1

The result looks like:



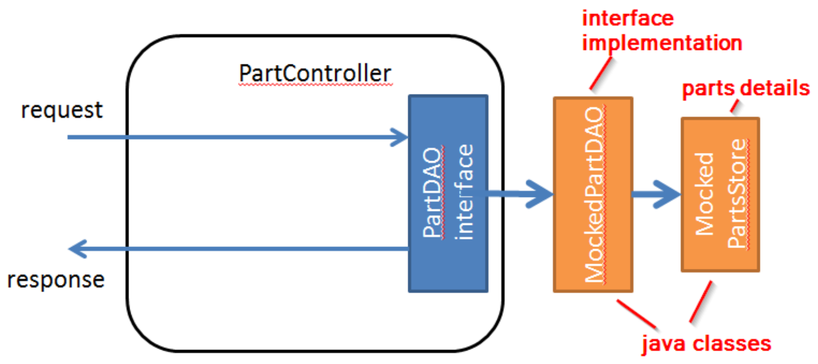
```
id:         "partId"
category:   "partCategory"
type:       "partType"
name:       "partName"
unitPrice:  0
currency:   "EUR"
```

We have now created a REST web service with one single operation. That's a good start. The next sections will add more and more detail.

5. Replacing the hard coded result with …

In the previous section, we added an operation to retrieve drone part details – but with a hard coded result. In this section, we will replace that with a Java class that has a collection of drone parts. However, in real-life situations, the drone parts data are more likely to be stored in a data base. Therefore, we will implement this solution using a Java interface, which makes it easier to switch between solutions later on:

The solution that we will build in this section can be depicted as:



It is easy to replace this solution with a DB back-end…

The following steps will be done:

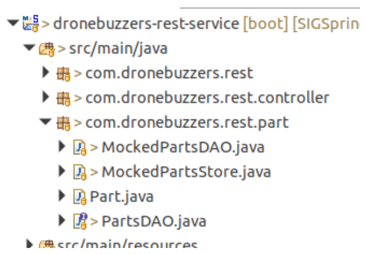
- Step 1: add a Java classes for drone parts: PartDAO, MockedPartDAO and MockedPartsStore
- Step 2: update the PartController
- Step 3: run and test with a browser

Step 1: add Java classes for drone parts: PartDAO, MockedPartDAO and MockedPartsStore

Create the following new Classes and Interface – mind to put them in the right directory:

Type	Class Name	Package
Class	MockedPartsDAO	com.dronebuzzers.rest.part
Class	MockedPartsStore	com.dronebuzzers.rest.part
Interface	PartsDAO	com.dronebuzzers.rest.part

The result looks like:



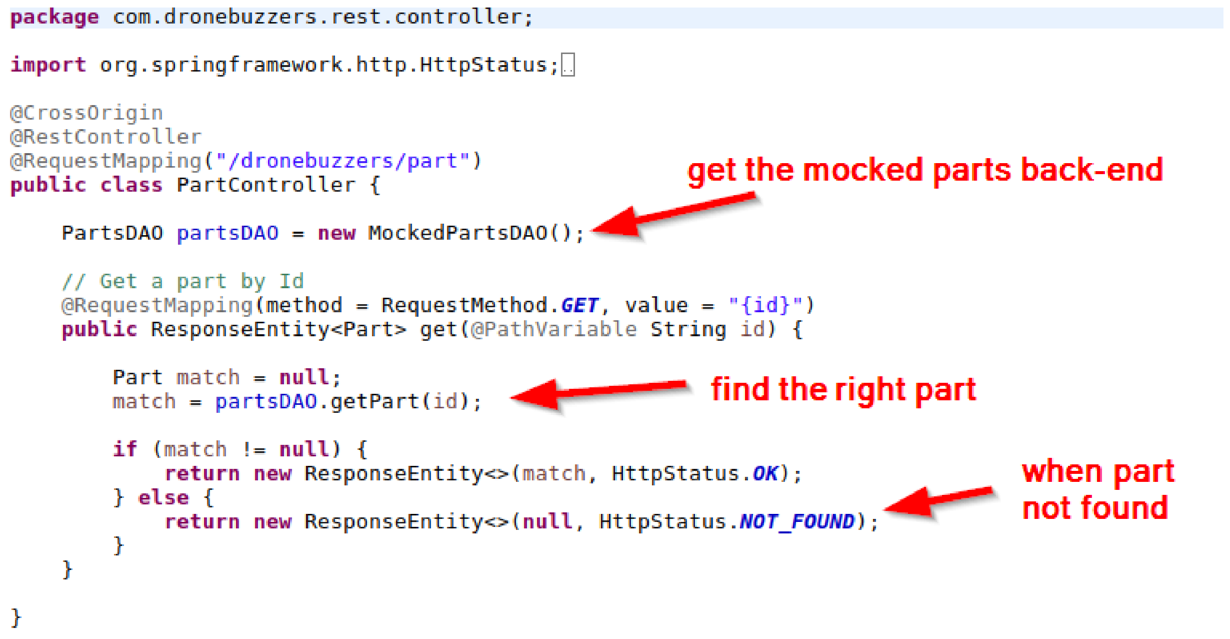
```
▾ > dronebuzzers-rest-service [boot] [SIGSprin
  ▾ > src/main/java
    ▸ > com.dronebuzzers.rest
    ▸ > com.dronebuzzers.rest.controller
    ▾ > com.dronebuzzers.rest.part
      ▸ > MockedPartsDAO.java
      ▸ > MockedPartsStore.java
      ▸ Part.java
      ▸ > PartsDAO.java
  ▸ src/main/resources
```

Now, the files have to get the right content: copy-and-paste the contents of the files in the lab 1/input directory to the files in STS Eclipse:

Type	Class Name	File
Class	MockedPartsDAO	SIGSpringBoot101/lab 1/input/MockPartsDAO.java
Class	MockedPartsStore	SIGSpringBoot101/lab 1/input/MockedPartsStore.java
Interface	PartsDAO	SIGSpringBoot101/lab 1/input/PartsDAO.java
Class	PartController	SIGSpringBoot101/lab 1/input/PartController-1operation.java

Step 2: update the PartController

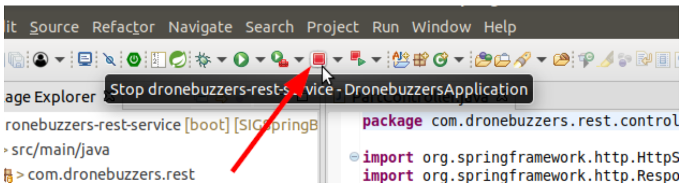
The most interesting change in the previous step is in the PartController class:



```
package com.dronebuzzers.rest.controller;

import org.springframework.http.HttpStatus;

@CrossOrigin
@RestController
@RequestMapping("/dronebuzzers/part")
public class PartController {

    PartsDAO partsDAO = new MockedPartsDAO();

    // Get a part by Id
    @RequestMapping(method = RequestMethod.GET, value = "{id}")
    public ResponseEntity<Part> get(@PathVariable String id) {

        Part match = null;
        match = partsDAO.getPart(id);

        if (match != null) {
            return new ResponseEntity<>(match, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
        }
    }

}
```

Step 3: run and test with a browser

If you still have your application running, first stop it by clicking the Stop button:



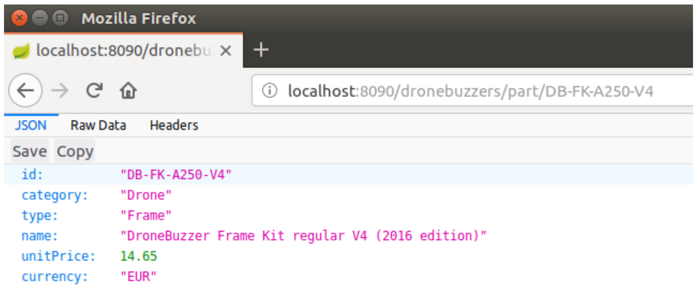
it Source Refactor Navigate Search Project Run Window Help
age Explorer
Stop dronebuzzers-rest-service - DronebuzzersApplication
ronebuzzers-rest-service [boot] [SIGspringB
src/main/java
> com.dronebuzzers.rest

```
package com.dronebuzzers.rest.control
import org.springframework.http.HttpS
import org.springframework.http.Respo
```

Now, you can run the application again (right-click the project – Run As – Spring Boot App), and look in the console log that the application is started without errors. Next, try Url

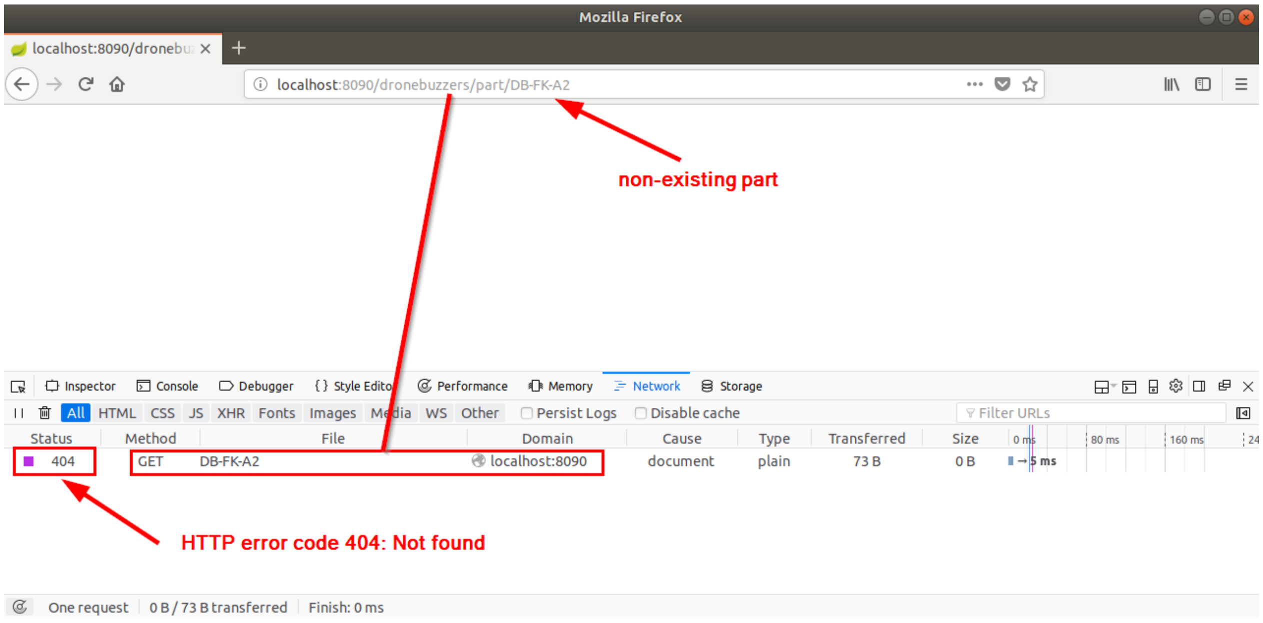
```
http://localhost:8090/dronebuzzers/part/DB-FK-A250-V4
```

Examine the result:



```
id:         "DB-FK-A250-V4"
category:   "Drone"
type:       "Frame"
name:       "DroneBuzzer Frame Kit regular V4 (2016 edition)"
unitPrice:  14.65
currency:   "EUR"
```

Now, click F12 in Firefox to open the Developer tools and then click the Network tab. Then, request the data for a non-existing part, using url http://localhost:8090/dronebuzzers/part/DB-FK-A2



Status	Method	File	Domain	Cause	Type	Transferred	Size
404	GET	DB-FK-A2	localhost:8090	document	plain	73 B	0 B

Click F12 again to hide the Firefox Developer tools.

6. Replacing the hard coded result with …

Now, we have a simple REST web service running with one single eoperation. In this section, we will add multiple operations to the *part* resource:

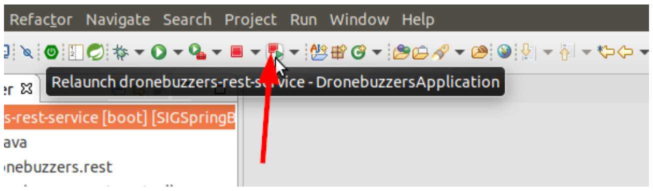
First, add the class Parts – sing the source code from the indicated file:

Class Name	File
com.dronebuzzers.rest.part.Parts	SIGSpringBoot101/lab 1/input/Parts.java

Second, update the PartController with the content from the file below:

Class Name	File
com.dronebuzzers.rest.controller.PartController	SIGSpringBoot101/lab 1/input/PartController-more-operations.java

Now, re-run the service. An easy way to do that from within Eclipse STS is to just click the Relaunch button:


Refactor Navigate Search Project Run Window Help
Relaunch dronebuzzers-rest-service - DronebuzzersApplication
s-rest-service [boot] [SIGSpringB
ava
nebuzzers.rest

Some test queries to try:

Operation	Url
Get all parts	http://localhost:8090/dronebuzzers/part
Get part by Id	http://localhost:8090/dronebuzzers/part/DB-ESC-629-40A
Get parts by category	http://localhost:8090/dronebuzzers/part/category/Drone
Get parts by type	http://localhost:8090/dronebuzzers/part/type/Motor

7. An operation with a Request body …

So far, all operations were completely defined by the Url, i.e. the service calls did not have a Request body.

In this section, we will add an operation to the web service for submitting an order. The order itself will be defined in a Request body.

The following steps will be done:

- Step 1: add Java classes for order handling
- Step 2: add the Class OrderController
- Step 3: run and test with curl

- Step 4: run and test with Postman

Step 1: add Java classes for order handling

Add the following classes – mind to use the right package - package – and copy the Java source code from the indicated files:

Class Name	File
com.dronebuzzers.rest.order.Order	SIGSpringBoot101/lab 1/input/Order.java
com.dronebuzzers.rest.order.OrderLine	SIGSpringBoot101/lab 1/input/OrderLine.java
com.dronebuzzers.rest.order.OrderSummary	SIGSpringBoot101/lab 1/input/OrderSummary.java
com.dronebuzzers.rest.order.OrderSummaryLine	SIGSpringBoot101/lab 1/input/OrderSummaryLine.java

The classes Order and OrderLine are used in the Request body: they represent the order for drone parts.

The classes OrderSummary and OrderSummaryLines are used in the Response body: they represent a summary of the submitted order.

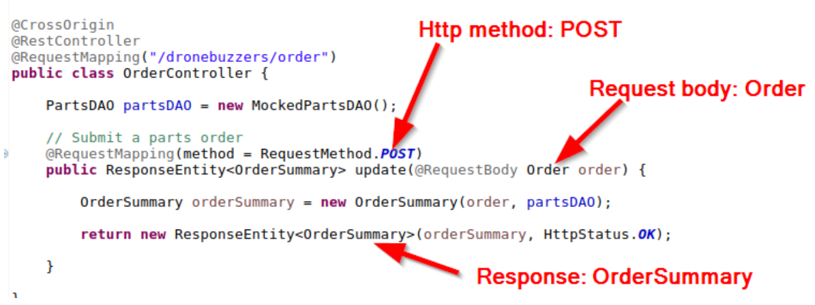
Step 2: add the OrderController Class

So far, we have implemented several operations around drone part handling. All these operations work on the resource *part*. Now that we are implementing functionality around order handling, from REST service point of view, this is an operation on a new resource: *order*. Therefore, we also introduce a new RestController – which will help to keep the code nicely structured.

Add the OrderController Class – and copy the Java source code from the indicated file:

Class Name	File
com.dronebuzzers.rest.controller.OrderController	SIGSpringBoot101/lab 1/input/Order.java

The OrderController class is somewhat different from the PartController:



```
@CrossOrigin
@RestController
@RequestMapping("/dronebuzzers/order")
public class OrderController {

    PartsDAO partsDAO = new MockedPartsDAO();

    // Submit a parts order
    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<OrderSummary> update(@RequestBody Order order) {

        OrderSummary orderSummary = new OrderSummary(order, partsDAO);

        return new ResponseEntity<OrderSummary>(orderSummary, HttpStatus.OK);

    }
}
```

Step 3: run and test with curl

Now, re-run the service. In Eclipse STS, click the Relaunch button.

The directory /home/developer/projects/SIGSpringBoot101/lab 1/input contains the file order.json, which holds the data for a parts order request:

```
{
   "clientId":"TDF",
   "clientReference":"TDF-0067",
   "orderLines":[
      {
         "id":"DB-FK-A250-V4",
         "count":22
      },
      {
         "id":"DB-38406-2350KV",
         "count":4
      }
   ]
}
```

In a Linux terminal, this file can be used to test the submit order operation using the curl command:

```
curl -X POST http://localhost:8090/dronebuzzers/order --header
"Content-Type:application/json" --data "@order.json" --silent |
json_pp
```

Mind to run the command from the directory `/home/developer/projects/SIGSpringBoot101/lab 1/input`

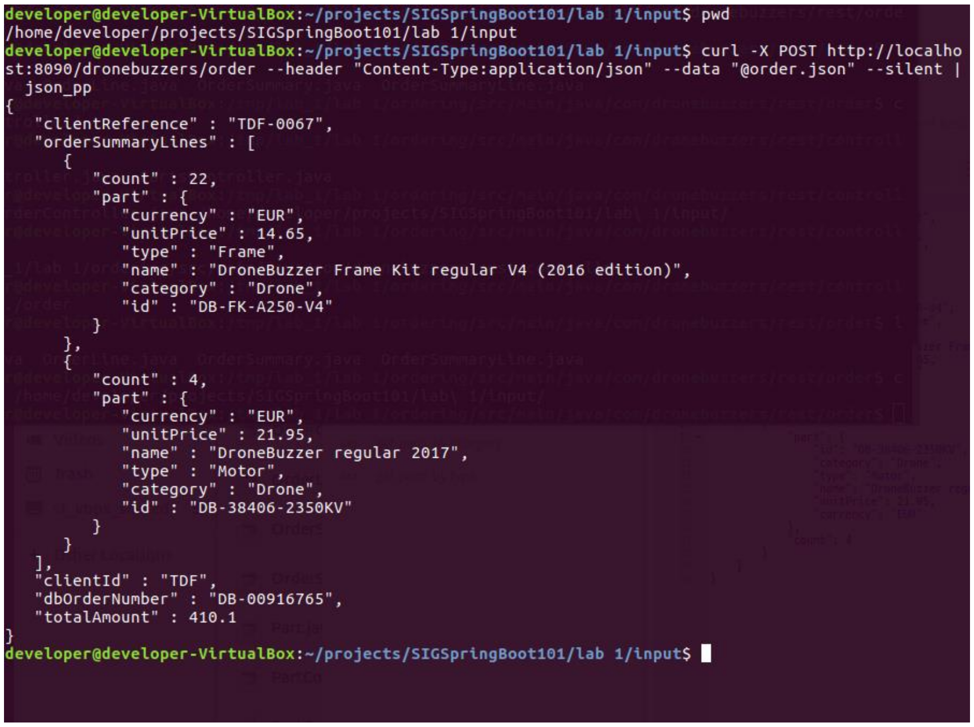
The curl command can be used to transfer a Url. We used the following options:

Curl Options	Meaning
-X POST	Transfer Url with http POST method
http://localhost:8090/dronebuzzers/order	Service endpoint
--header "Content-Type:application/json"	http header named Content-Type and with value

	application/json is added
--data "@order.json"	Sends file order.json as data along with POST request
--silent	Suppress progress meter and error messages

The json_pp command is used for pretty-printing the json service output.

For example



```
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 1/input$ pwd
/home/developer/projects/SIGSpringBoot101/lab 1/input
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 1/input$ curl -X POST http://localho
st:8090/dronebuzzers/order --header "Content-Type:application/json" --data "@order.json" --silent |
 json_pp
{
   "clientReference" : "TDF-0067",
   "orderSummaryLines" : [
      {
         "count" : 22,
         "part" : {
            "currency" : "EUR",
            "unitPrice" : 14.65,
            "type" : "Frame",
            "name" : "DroneBuzzer Frame Kit regular V4 (2016 edition)",
            "category" : "Drone",
            "id" : "DB-FK-A250-V4"
         }
      },
      {
         "count" : 4,
         "part" : {
            "currency" : "EUR",
            "unitPrice" : 21.95,
            "name" : "DroneBuzzer regular 2017",
            "type" : "Motor",
            "category" : "Drone",
            "id" : "DB-38406-2350KV"
         }
      }
   ],
   "clientId" : "TDF",
   "dbOrderNumber" : "DB-00916765",
   "totalAmount" : 410.1
}
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 1/input$
```

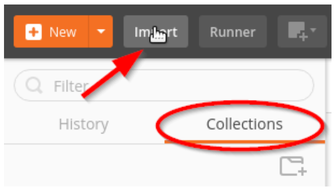
Take some time to examine how request and response objects map to json messages.

Step 4: run and test with Postman

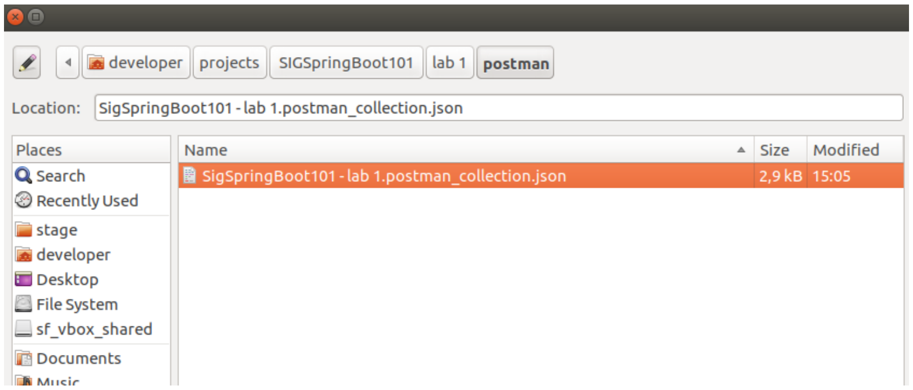
An easier way to run these tests is to use the Postman app. All is needed is a google account.

Start the app by clicking the icon:

The main page will open. Go to the collections tab and click the import button



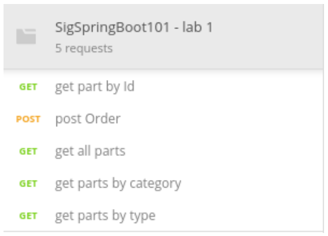
A postman collection is stored in the directory `SIGSpringBoot101/lab 1/postman`, named `SigSpringBoot101 - lab 1.postman_collection.json`:



Location: SigSpringBoot101 - lab 1.postman_collection.json

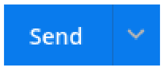
Name	Size	Modified
SigSpringBoot101 - lab 1.postman_collection.json	2,9 kB	15:05

Open that file. In your collections, the SigSpringBoot101 – lab 1 collection is now added:

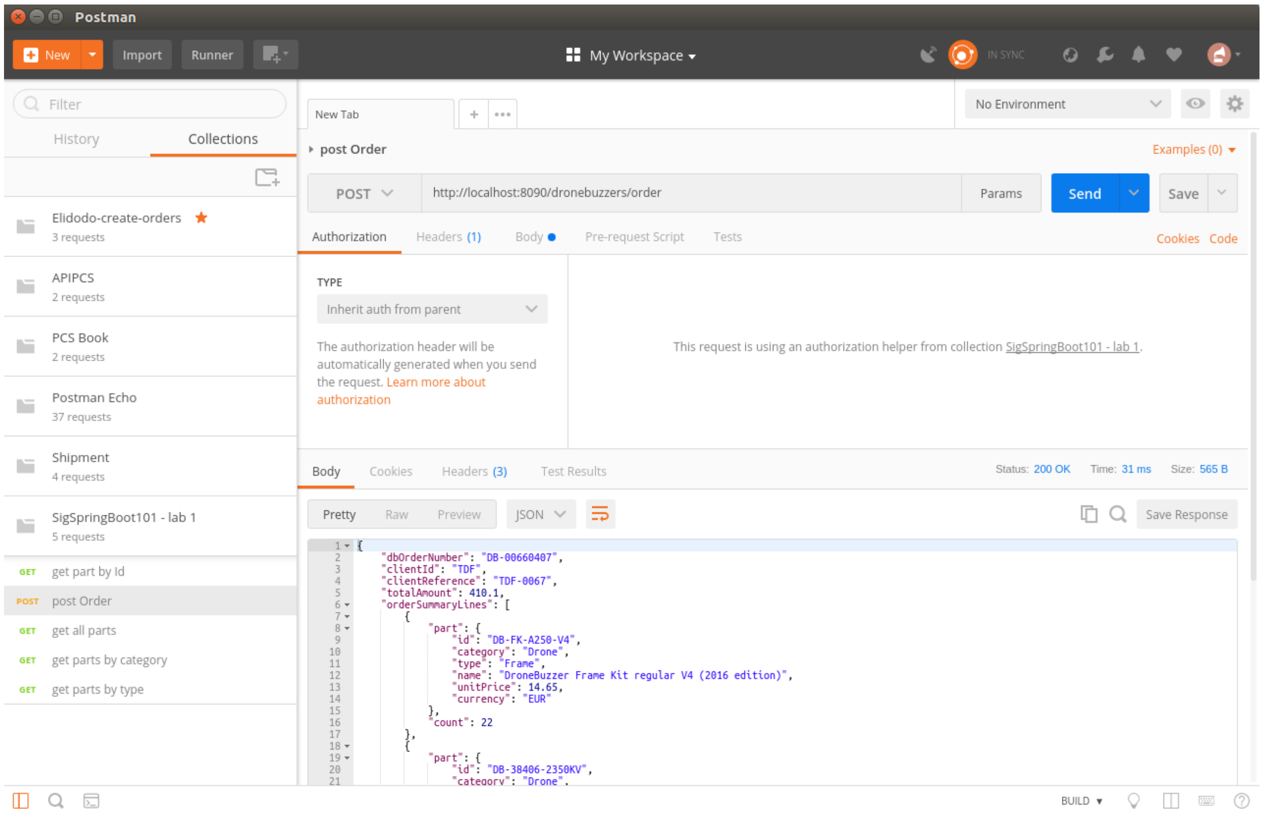


SigSpringBoot101 - lab 1
5 requests

GET	get part by Id
POST	post Order
GET	get all parts
GET	get parts by category
GET	get parts by type

Click the post Order request and click the blue send button 

That should result in a screen like below:



post Order

POST http://localhost:8090/dronebuzzers/order

TYPE

Inherit auth from parent

The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is using an authorization helper from collection SigSpringBoot101 - lab 1.

Status: 200 OK   Time: 31 ms   Size: 565 B

```
{
    "dbOrderNumber": "DB-00660407",
    "clientId": "TDF",
    "clientReference": "TDF-0067",
    "totalAmount": 410.1,
    "orderSummaryLines": [
        {
            "part": {
                "id": "DB-FK-A250-V4",
                "category": "Drone",
                "type": "Frame",
                "name": "DroneBuzzer Frame Kit regular V4 (2016 edition)",
                "unitPrice": 14.65,
                "currency": "EUR"
            },
            "count": 22
        },
        {
            "part": {
                "id": "DB-38406-2350KV",
                "category": "Drone",
```

This is a good time to fiddle with the other requests ;-)

Note: it would be nice to have a GET order operation that retrieves the data on a submitted order. That is left out of this Lab, as it would require additional code that would further complicatie the Lab. However, in Lab 6, submitted orders are stored in a DataBase. Here, also an operation for retrieving order data is added.

8. Service summary

After completion of this lab, you have implemented a REST web service in Java using Spring Boot. The complete code for this REST web service can be found under SIGSpringBoot101/lab 1/dronebuzzers-completed.

This service can deployed and run on a simple Tomcat web server – no complex Java EE machinery is required. Later on we will see how we can take our Spring Boot application out of STS/Eclipse and deploy it on a runtime platform – in Docker, on the cloud and beyond.

The service implements the following operations:

Operation	Url
Get all parts	http://localhost:8090/dronebuzzers/part
Get part by Id	http://localhost:8090/dronebuzzers/part/DB-ESC-629-40A
Get parts by category	http://localhost:8090/dronebuzzers/part/category/Drone
Get parts by type	http://localhost:8090/dronebuzzers/part/type/Motor
Submit an order	http://localhost:8090/dronebuzzers/order (POST)

Also, you have seen how Postman can be used for testing REST services.