**Lab 3  
REST web service - contract first in SwaggerHub**

This lab will show you how you can specify a REST interface on SwaggerHub – a cloud platform that we will use for specifying our API and then generating code for a SpringBoot implementation. SwaggerHub does this using the Swagger/OpenAPI specification.

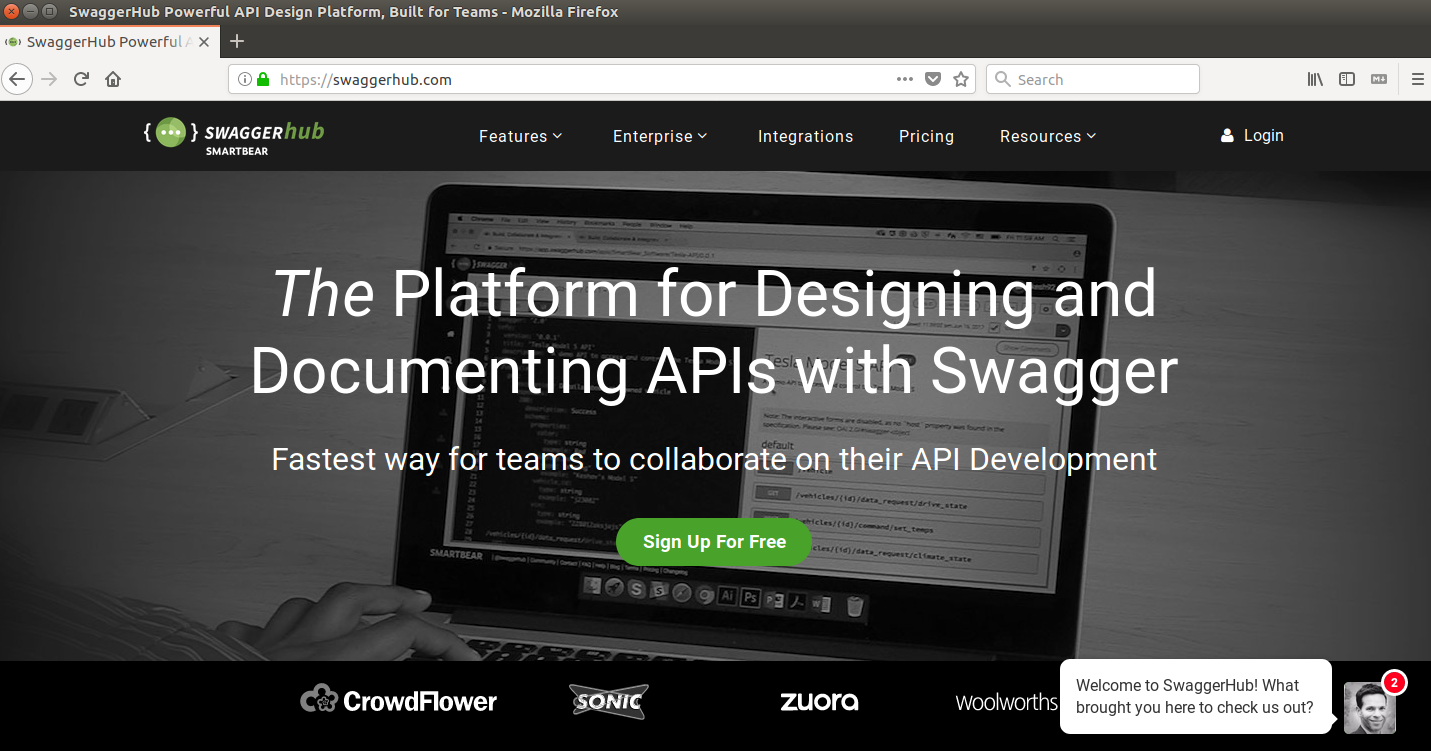
You can consider using SwaggerHub if you want to develop your REST services in a contract first fashion.

The lab uses the Swagger/OpenAPI 2.0 specification, and not (yet) the OpenAPI 3.0 specification: the tooling seems a bit more stable…

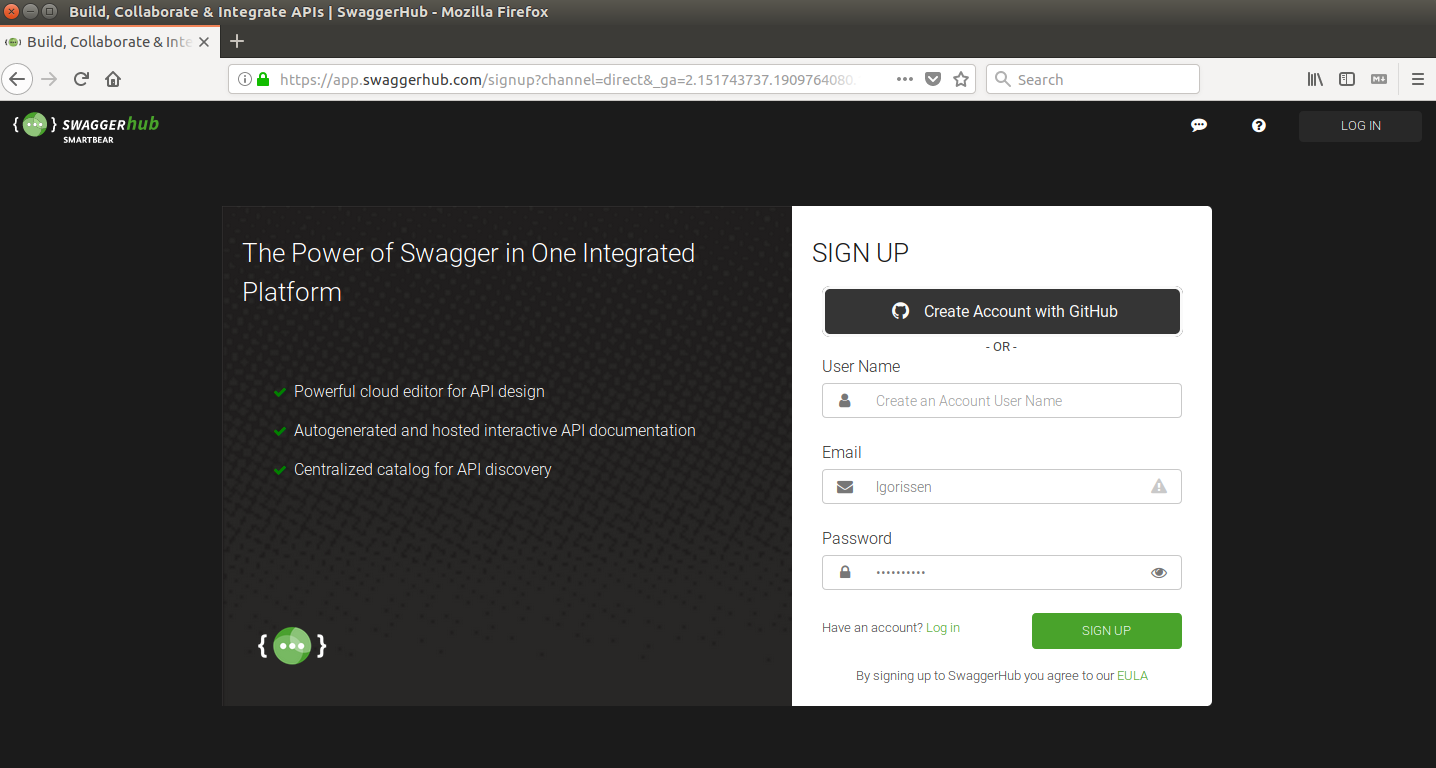
The starting point for this lab is to have the provided VirtualBox machine up-and-running. You are logged in under user/password developer/welcome1. We will then develop

# Registration with SwaggerHub

Goto swaggerhub.com

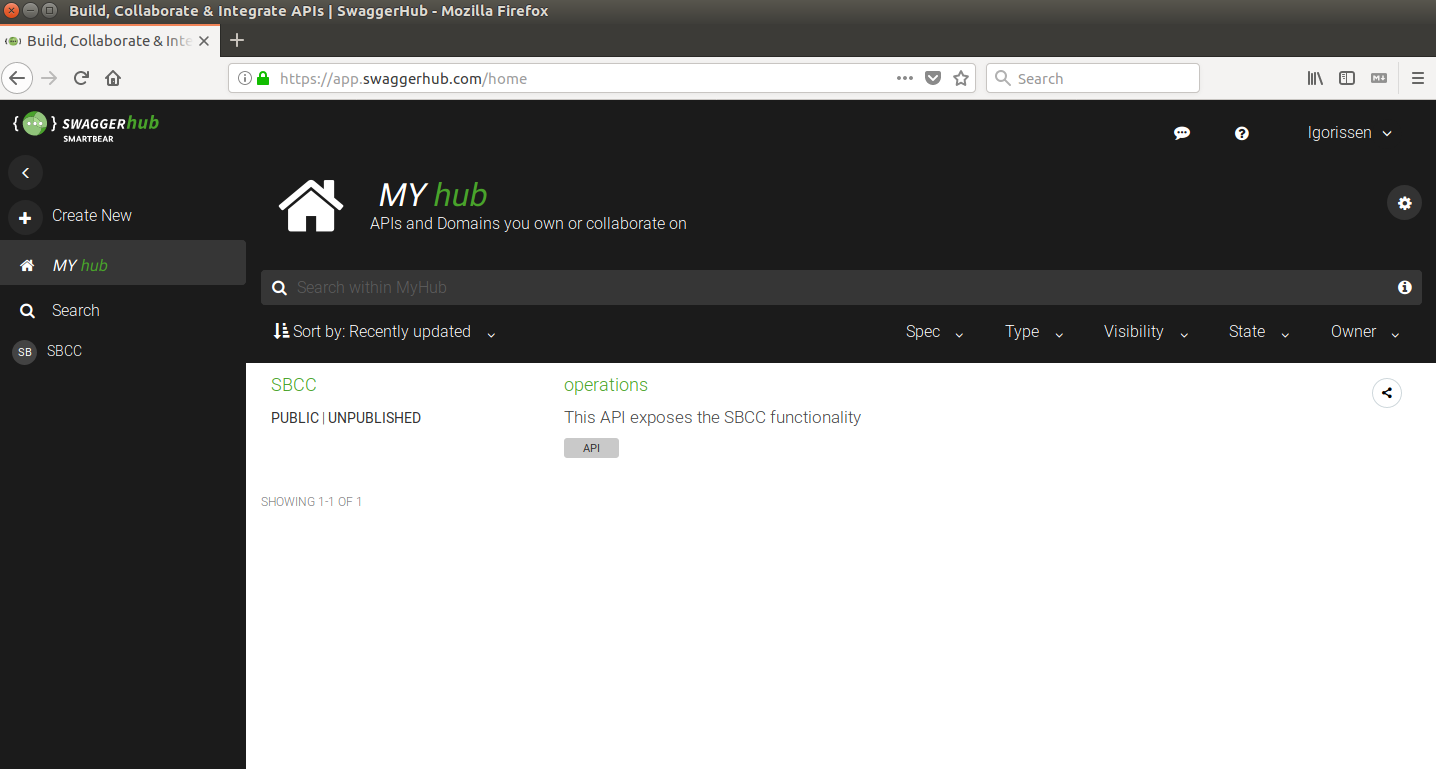


Click the ‘Sign Up For Free’ button:



Here, you can can pick your preferred option for registration.

After completing the registration and logging in, you should have a page pretty similar to the one below:



Note in the above screen, that there is already an interface present, named SBCC – it shouldn’t be there in your screen.

# A HelloWorld API in SwaggerHub

Now that you have your account in SwaggerHub and you are logged in, you are ready to get to work with SwaggerHub. We will first create a simple API in SwaggerHub.

The following steps will be done:

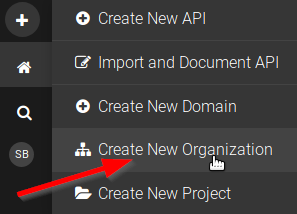
* Step 1: create the API specification in SwaggerHub
* Step 2: generating code in SwaggerHub
* Step 3: import into eclipse and add business logic in the code
* Step 4: run and test the API

**Step 1: create the API specification in SwaggerHub**

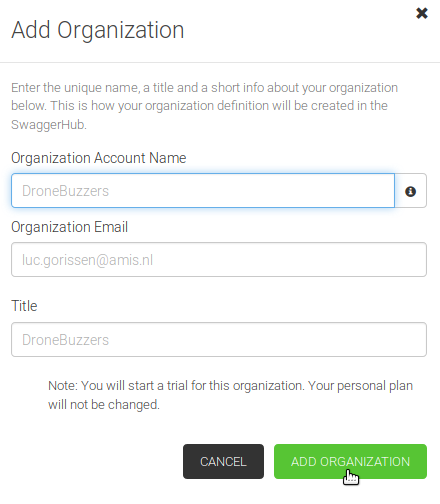
Log in into SwaggerHub, and click the + icon to start adding a new interface:



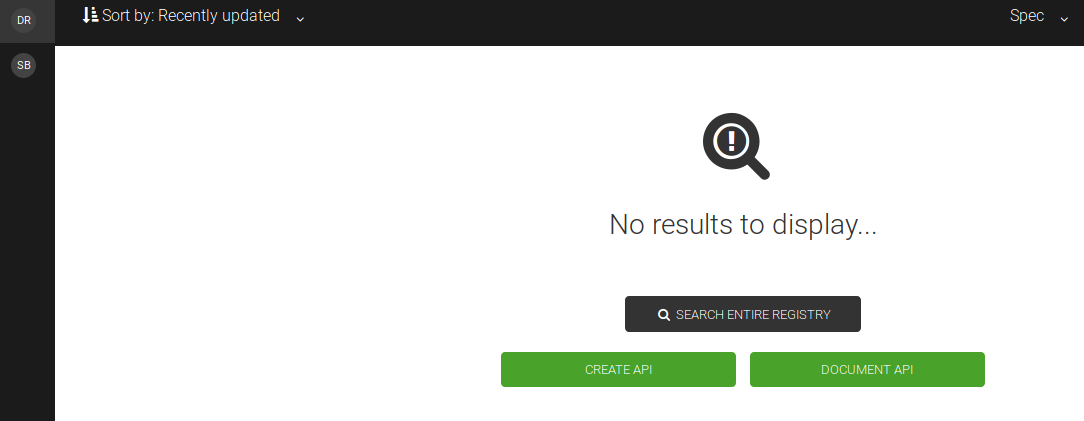
Start by creating a new organization: Create New Organization



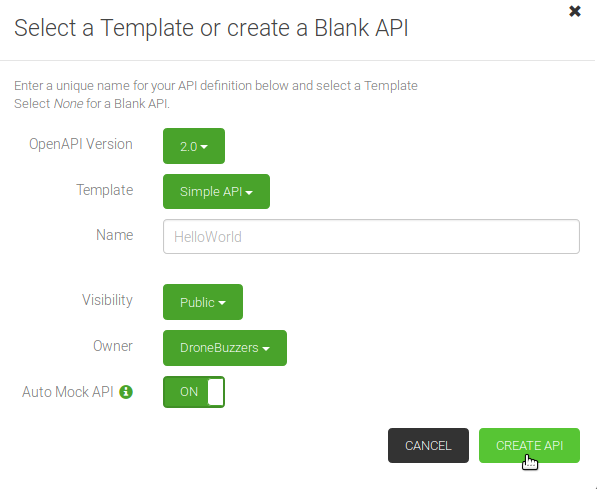
Complete the form as shown below:



After clicking the ‘Add Organization’ button, the screen should look like:



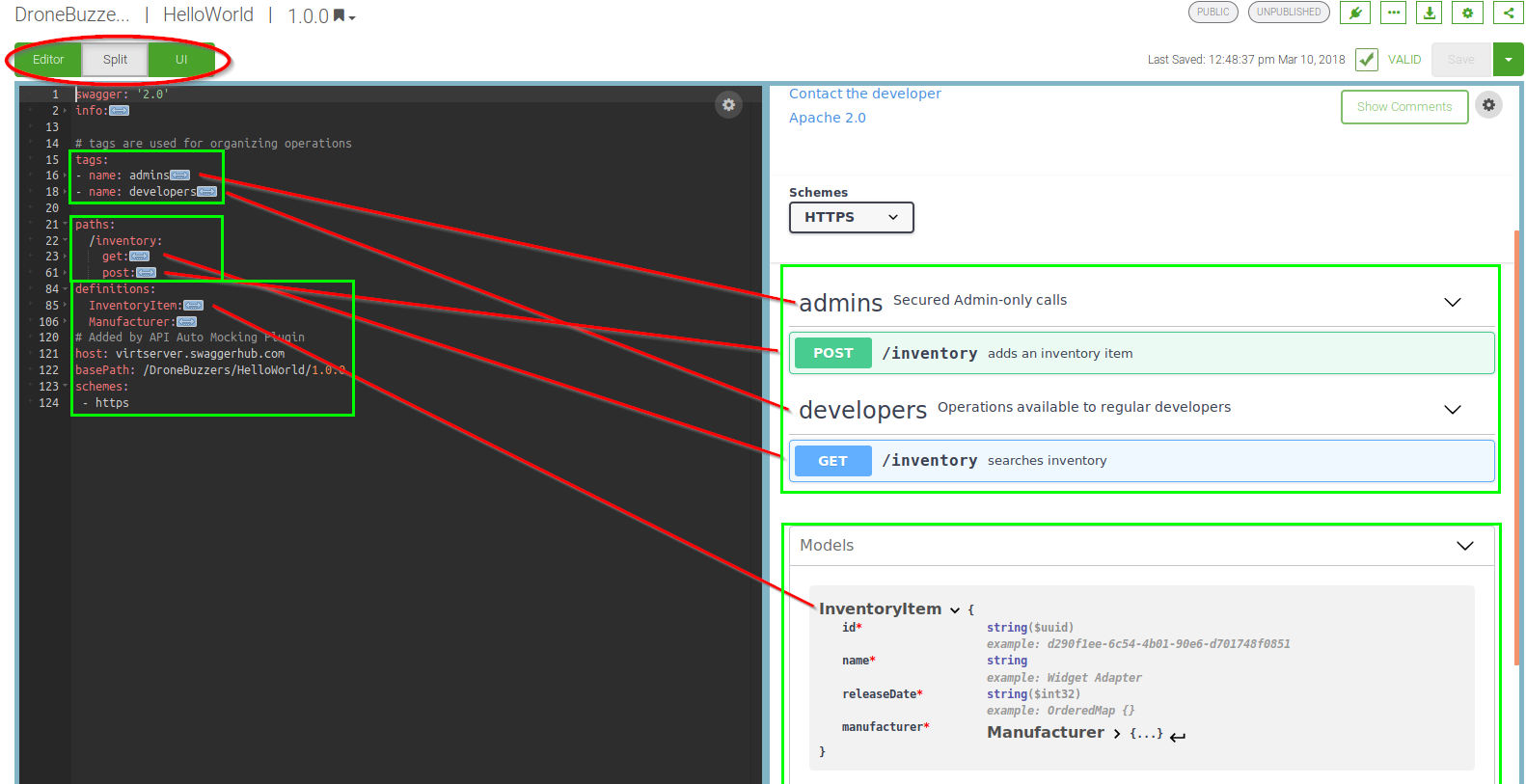
Click on the ‘CREATE API’ button to start adding the API. Complete the resulting form as shown below:



With respect to the above form some remarks:

* The OpenAPI version is set to 2.0. This version is also known as ‘Swagger 2.0’.
* The template is ‘Simple API’. For a good overview of how a more realistic interface specification can look like, it is good to examine the Petstore example that is also available. Just don’t do that now ;-)
* Visibility is set to Public as the number of private APIs that can be created with a free account is fixed to 1

Click CREATE API and wait for the magic to finish:



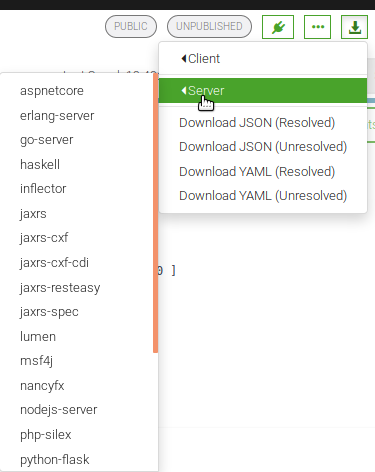
Now is a good moment to spend some time to understand what we’re looking at:

* The screen is now in split view (red oval on top): it shows the OpenAP/Swagger 2.0 interface definition on the left. On the right is the describes of the API in Swagger documentation style.
* The top left green rectangle shows the tags that can be used for logical grouping of operations
* The middle left green rectangle describes the paths to the endpoints and for each endpoint the operations and parameters
* The bottom left green rectangle has the definitions of the data types that can be consumed/produced by the operations
* The right side of the screen shows the Swagger documentation corresponding to the interface definition

Note how the comments and examples for the Swagger documentation are incorporated in the API definition.

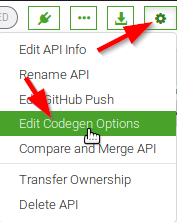
**Step 2: generating code in SwaggerHub**

SwaggerHub can also generate code for an API definition: for both client and server side. And for many languages. A quick check shows the first language options:



Impressive!

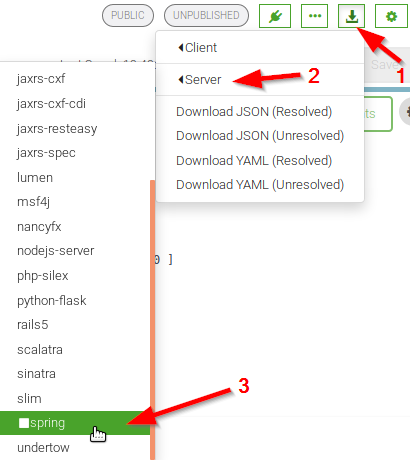
Before we start our ‘spring’ code generation, we will first set the code generation options. Go to the code generation options as shown below:



In the pop-up window, select spring in the Servers section and then complete the settings as shown in the table below:

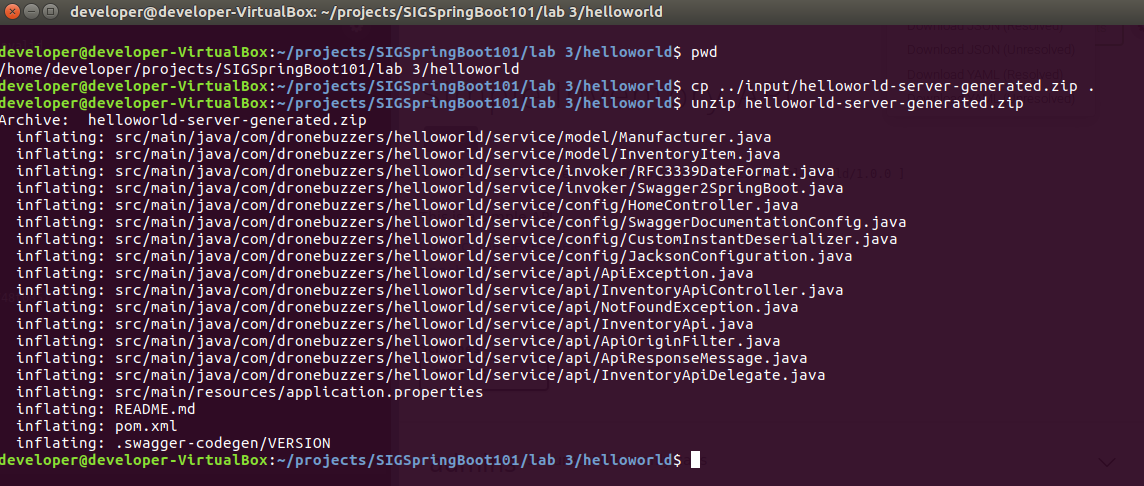
|  |  |
| --- | --- |
| Setting | Value |
| useTags | *not checked* |
| implicitHeaders | *not checked* |
| configPackage | com.dronebuzzers.helloworld.service.config |
| interfaceOnly | *not checked* |
| artifactVersion | 1.0.0 |
| sortParamsByRequiredFlag | *not checked* |
| useOptional | *not checked* |
| singleContentTypes | *not checked* |
| sourceFolder | /src/main/java |
| serializableModel | *not checked* |
| artifactDescription | DroneBuzzers HelloWorld |
| delegatePattern | *checked* |
| scmDeveloperConnection |  |
| apiPackage | com.dronebuzzers.helloworld.service.api |
| licenseName |  |
| invokerPackage | com.dronebuzzers.helloworld.service.invoker |
| dateLibrary |  |
| artifactId | helloworld-rest-service |
| licenseUrl |  |
| swaggerDocketConfig | *checked* |
| useBeanValidation | *not checked* |
| withXml | *not checked* |
| responseWrapper |  |
| developerEmail | [*luc.gorissen@amis.nl*](mailto:luc.gorissen@amis.nl) |
| developerOrganizationUrl | [*https://www.amis.nl*](https://www.amis.nl) |
| fullJavaUtil | *not checked* |
| bigDecimalAsString | *not checked* |
| ensureUniquerParams | *not chekced* |
| basePackage | com.dronebuzzers.helloworld.service |
| developerName | *Luc Gorissen* |
| allowUnicodeIdentifiers | *not checked* |
| java8 | *checked* |
| Title | DroneBuzzers HelloWorld |
| localVariablePrefix | *not checked* |
| groupId | com.dronebuzzers.helloworld |
| Library | Sping-boot Server application using the SpringFox integration |
| scmConnection |  |
| hideGenerationTimestamp |  |
| Async | *not checked* |
| modelPackage | com.dronebuzzers.helloworld.service.model |
| developerOrganization | *AMIS* |
| artifactUrl |  |

After completing the list, click ‘SAVE OPTIONS’. Now, the server side code can be generated:



**Step 3: import into eclipse and add business logic in the code**

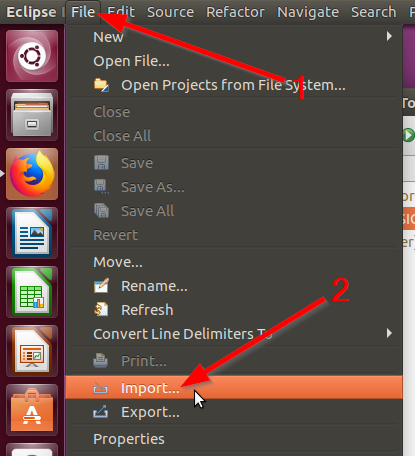
First, unpack the zip file in the right location. You can either use your own downloaded zip file from the previous step, or use the one provided: /home/developer/projects/SIGSpringBoot101/lab 3/input/helloworld-server-generated.zip



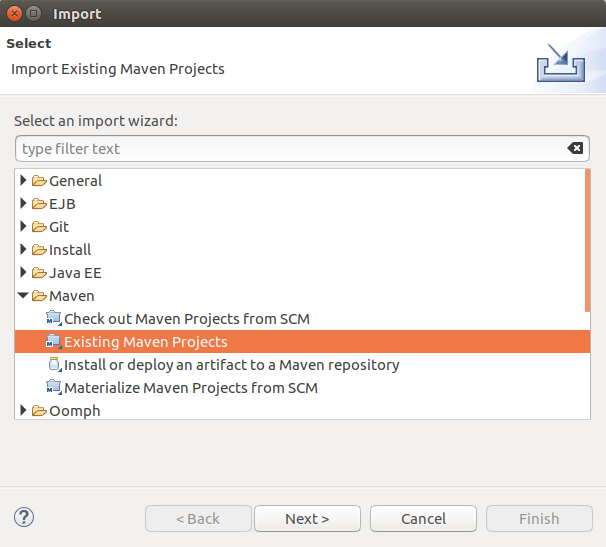
Remove the helloworld-server-generated.zip file from the ~/projects/SIGSpringBoot101/lab 3/helloworld directory.

Next, open Eclipse 

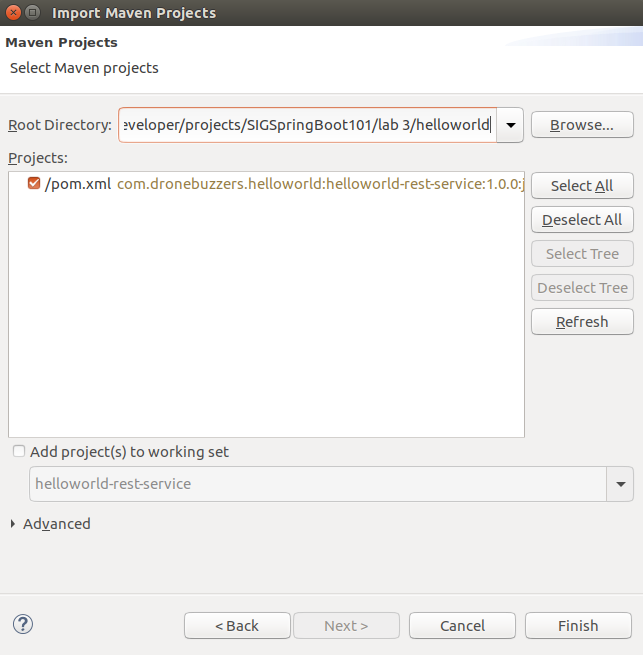
Pay attention: go to the Ubuntu top of the window and from the menu bar click on File and then Import:



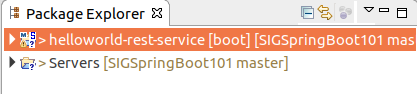
In the resulting Import pop-up, select ‘Existing Maven Projects’:



Click Next and then (1) set the Root Directory and (2) select the pom file:

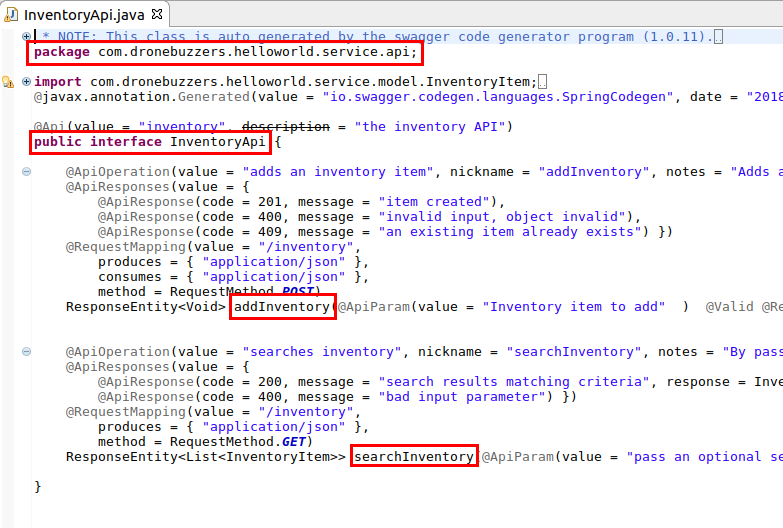


Click Finish and the helloworld-rest-service project should become visible in the Package Explorer:



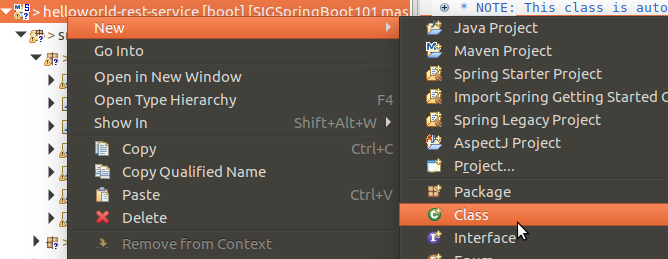
It is important to understand that the project that we have generated only has the API implementation: all business logic is missing. Now, we will continue to add the business logic.

The business logic that we will add will be an implementation of the interface InventoryApi:

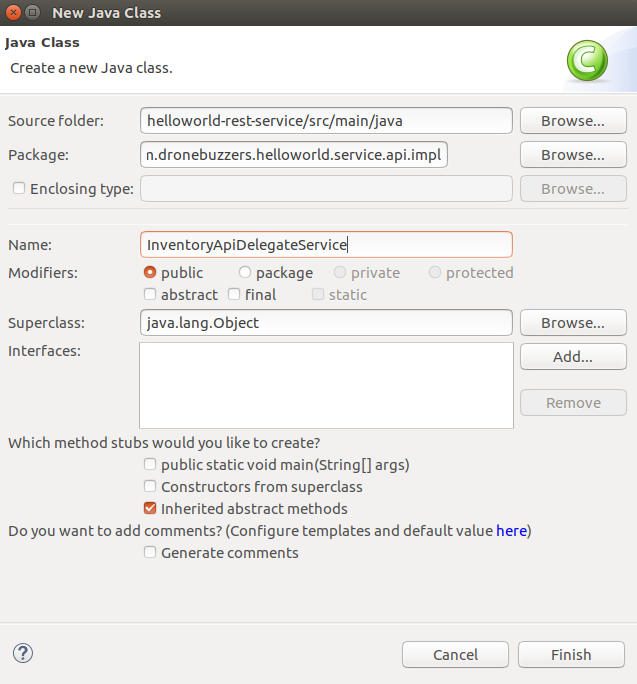


So, we will add a new java class named InventoryApiDelegateService in the package com.dronebuzzers.helloworld.service.api.imp.

Right-click the project and select Class:



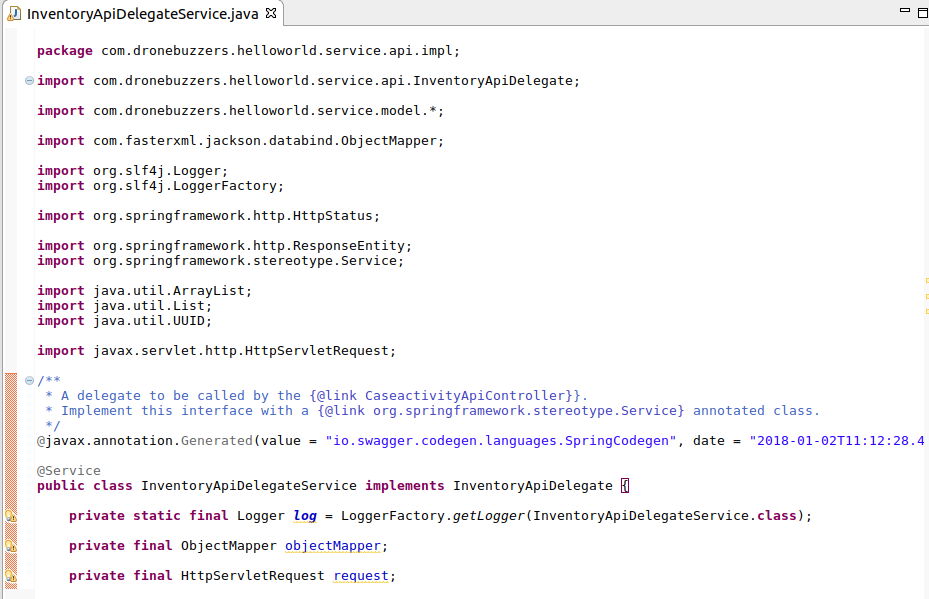
Complete like shown below:



Copy-paste in the sample code that can be found in file:

/home/developer/projects/SIGSpringBoot101/lab 3/input/InventoryApiDelegateService.java

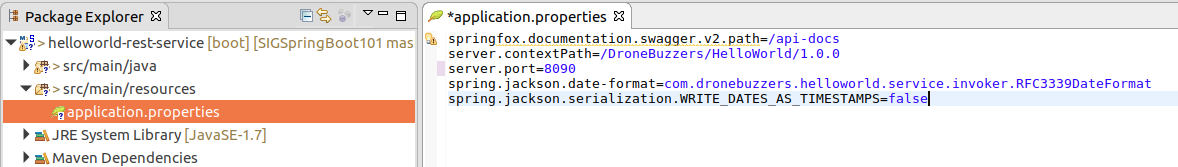
That should make it look like shown below:



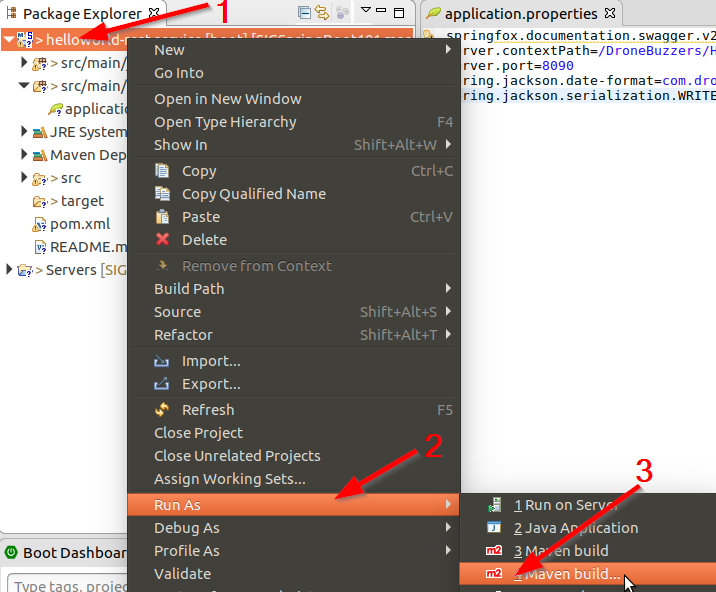
The code is now completed.

**Step 4: run and test the API**

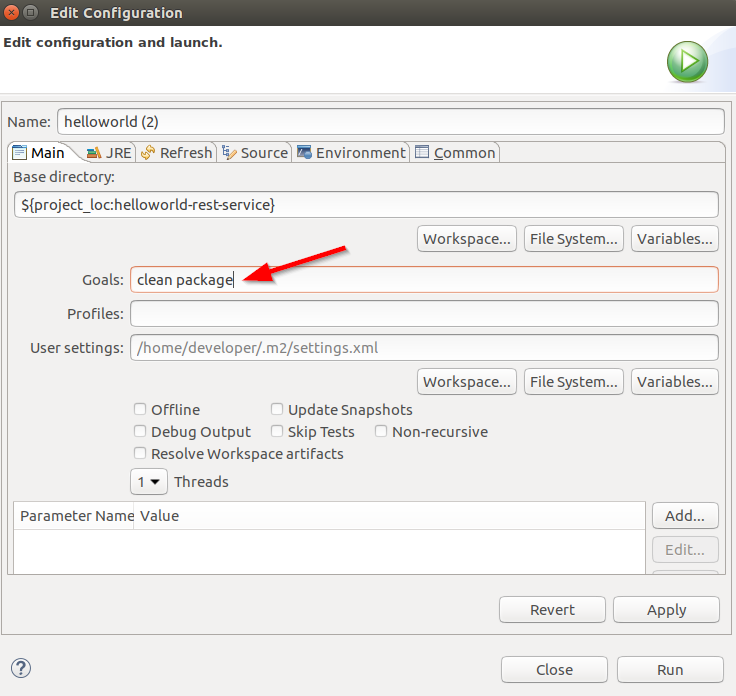
Before actually running the code, have a look at the application.properties file. Change the server.port setting to 8090, as port 8080 may be in use already:



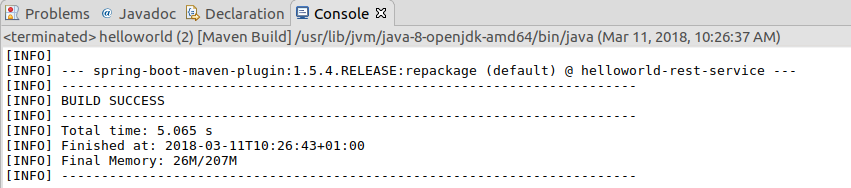
First step is to build the code: right-click the project, click ‘Run As’ and select the option ‘Maven build…’:



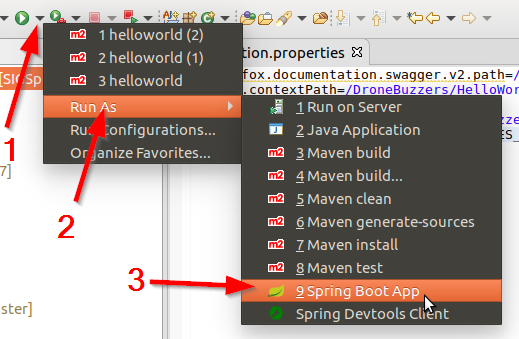
The pop-up window as shown below will be shown:



Complete like shown above and click Run. Check in the console that the code is built successfully:

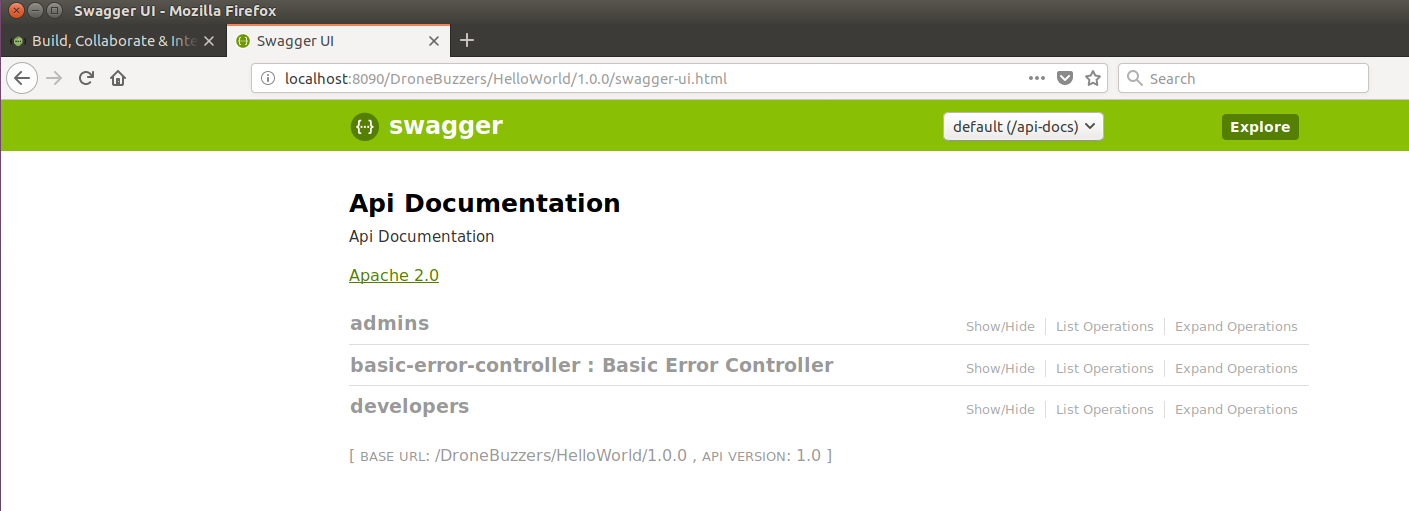


Now that the code is built, it is time to run it:



Verify that the code is running by going to url <http://localhost:8090/DroneBuzzers/HelloWorld/1.0.0/swagger-ui.html> in your browser:

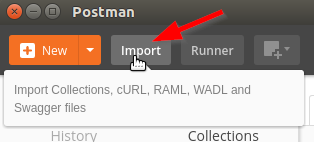
The result should look like:



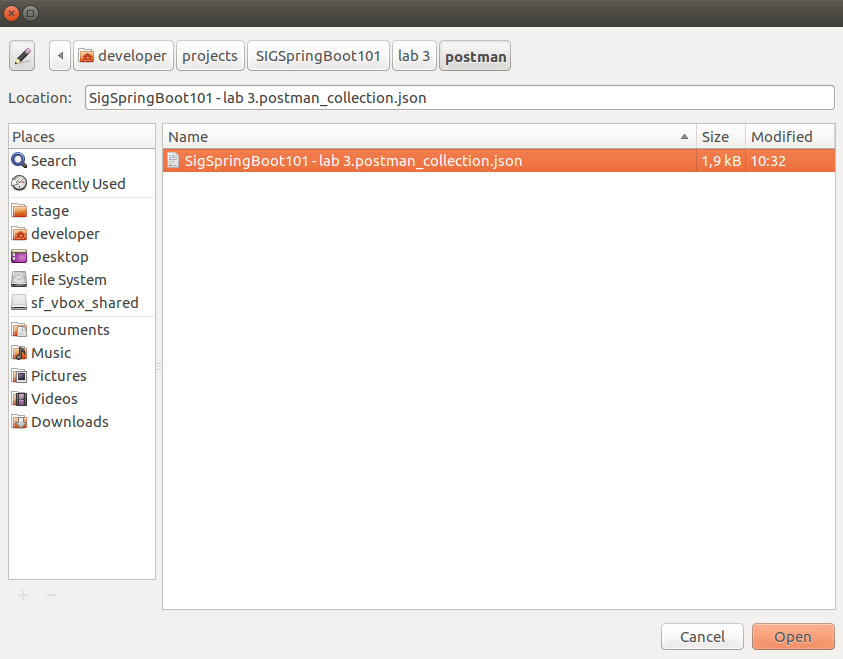
You can check url <http://localhost:8090/DroneBuzzers/HelloWorld/1.0.0/api-docs> for the raw output...

Now that the code is running, we will use Postman again to test it.

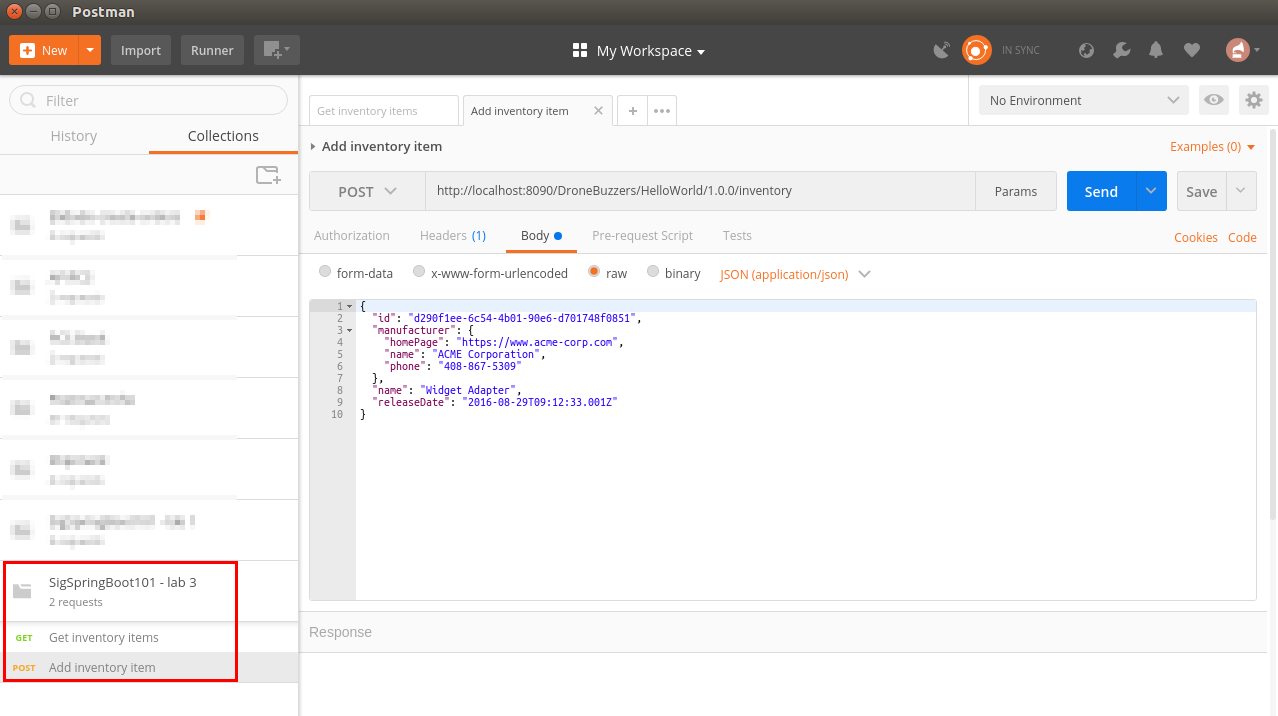
Start Postman  and import the Collection of Postman tests for lab 3:



The collection is in the /home/developer/projects/SIGSpringBoot101/lab 3/postman directory:



Next, test both operations:



# The DroneBuzzers API in SwaggerHub

The HelloWorld API in the previous section was a simple API that illustrated all the steps to develop a REST service in a contract-first style.

The same steps can now be done for the DroneBuzzers API, which is a bit more detailed interface.

Similar to the HelloWorld API, the following steps will be done:

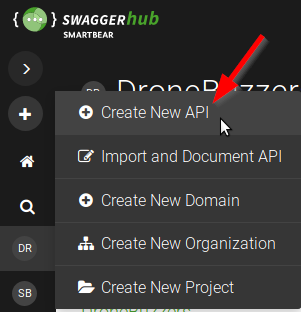
* Step 1: create the API specification in SwaggerHub
* Step 2: generating code in SwaggerHub
* Step 3: import into eclipse and add business logic in the code
* Step 4: run and test the API

The steps should be familiar, so in this section they will be described with less detail..

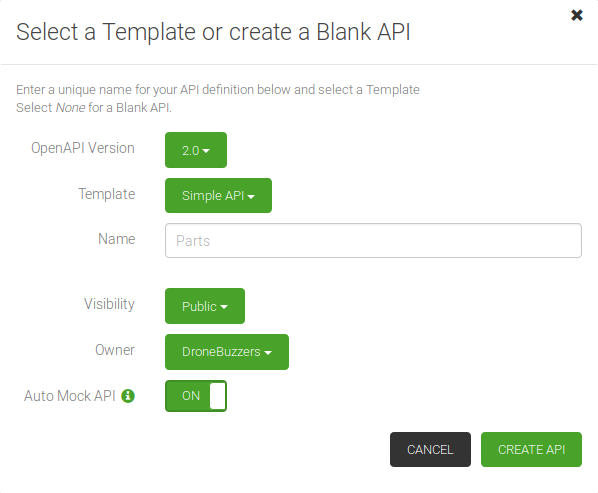
Should you not want to do the complete exercise, some intermediate results are made available in the /home/developer/projects/SIGSpringBoot101/lab 3/input directory. Like this text, intermediate results are marked in a box.

**Step 1: create the API specification in SwaggerHub**

In SwaggerHub, create a new API:

****

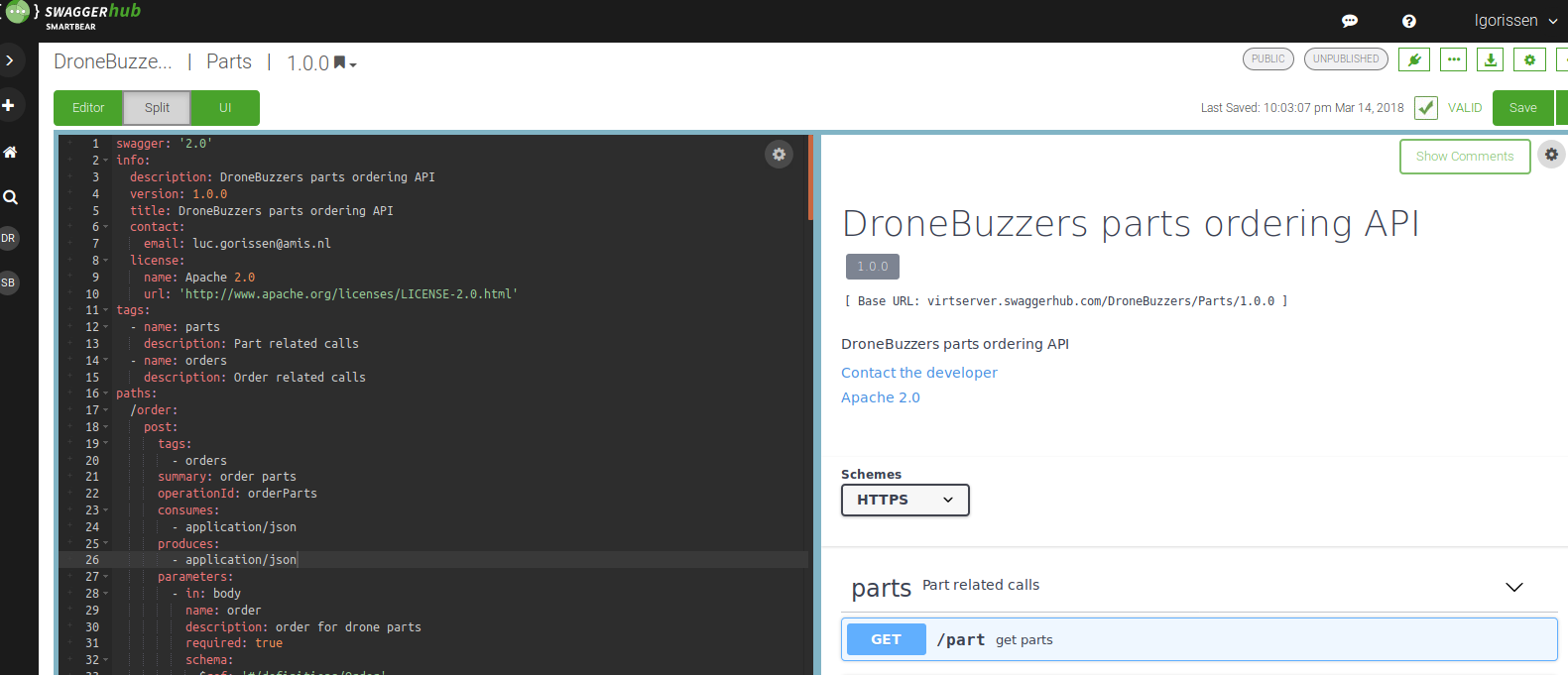
Complete the form for the API named Parts as shown below:



In the editor, replace the contents with the contents of file

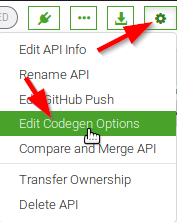
/home/developer/projects/SIGSpringBoot101/lab 3/input/DroneBuzzers\_Parts\_1.0.0\_swagger.json

The result in SwaggerHub should look somewhat like below:



**Step 2: generating code in SwaggerHub**

Before generating the code for the server side of the DroneBuzzers Parts API, go to the code generation options as shown below:



In the pop-up window, select spring in the Servers section and then complete the settings as shown in the table below:

|  |  |
| --- | --- |
| Setting | Value |
| useTags | *not checked* |
| implicitHeaders | *not checked* |
| configPackage | com.dronebuzzers.parts.service.config |
| interfaceOnly | *not checked* |
| artifactVersion | 1.0.0 |
| sortParamsByRequiredFlag | *not checked* |
| useOptional | *not checked* |
| singleContentTypes | *not checked* |
| sourceFolder | /src/main/java |
| serializableModel | *not checked* |
| artifactDescription | DroneBuzzers Parts |
| delegatePattern | *checked* |
| scmDeveloperConnection |  |
| apiPackage | com.dronebuzzers. parts.service.api |
| licenseName |  |
| invokerPackage | com.dronebuzzers. parts.service.invoker |
| dateLibrary |  |
| artifactId | dronebuzzers-rest-service |
| licenseUrl |  |
| swaggerDocketConfig | *checked* |
| useBeanValidation | *not checked* |
| withXml | *not checked* |
| responseWrapper |  |
| developerEmail | [*luc.gorissen@amis.nl*](mailto:luc.gorissen@amis.nl) |
| developerOrganizationUrl | [*https://www.amis.nl*](https://www.amis.nl) |
| fullJavaUtil | *not checked* |
| bigDecimalAsString | *not checked* |
| ensureUniquerParams | *not chekced* |
| basePackage | com.dronebuzzers.parts.service |
| developerName | *Luc Gorissen* |
| allowUnicodeIdentifiers | *not checked* |
| java8 | *checked* |
| title | DroneBuzzers Parts |
| localVariablePrefix | *not checked* |
| groupId | com.dronebuzzers. parts |
| library | Sping-boot Server application using the SpringFox integration |
| scmConnection |  |
| hideGenerationTimestamp |  |
| async | *not checked* |
| modelPackage | com.dronebuzzers.parts.service.model |
| developerOrganization | *AMIS* |
| artifactUrl |  |

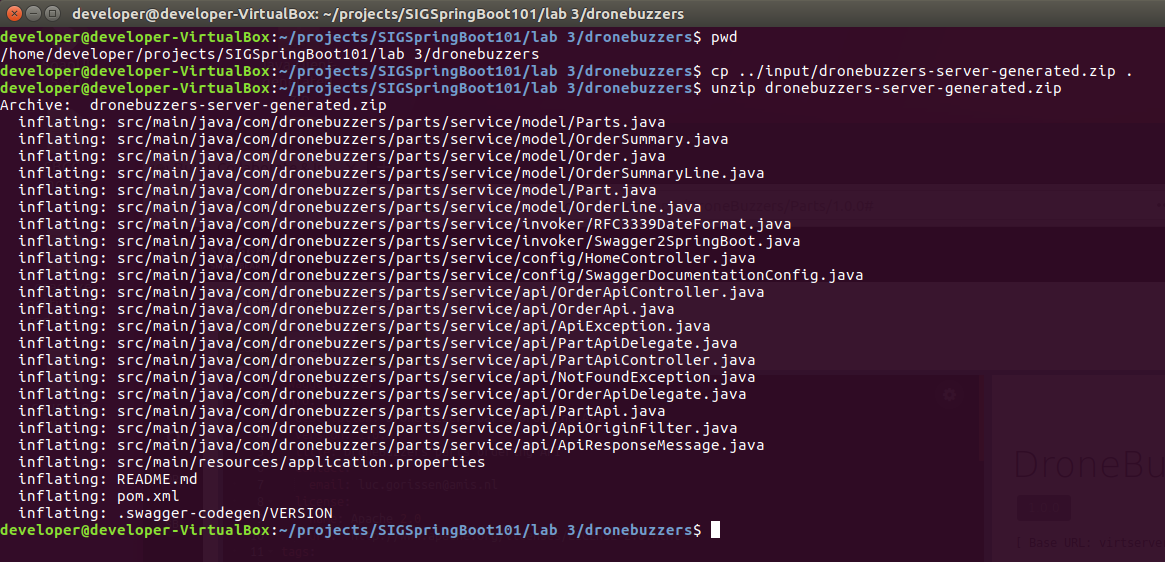
Now, the server side code can be generated.

Should you want to skip this step: the generated code is also present in:

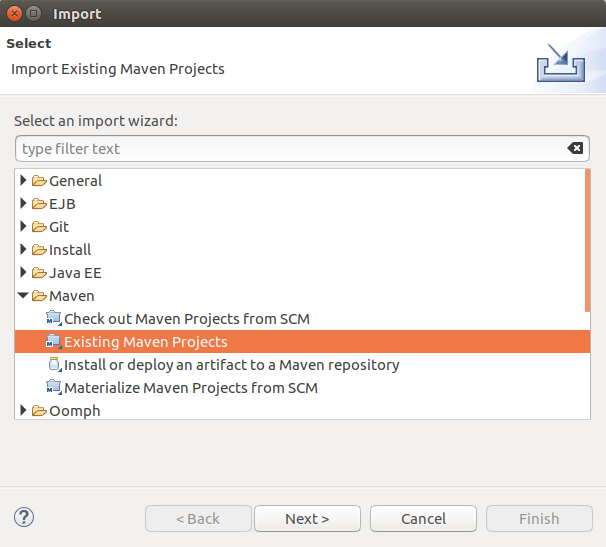
/home/developer/projects/SIGSpringBoot101/lab 3/input/dronebuzzers-server-generated.zip

**Step 3: import into eclipse and add business logic in the code**

The generated code can now be unzipped:

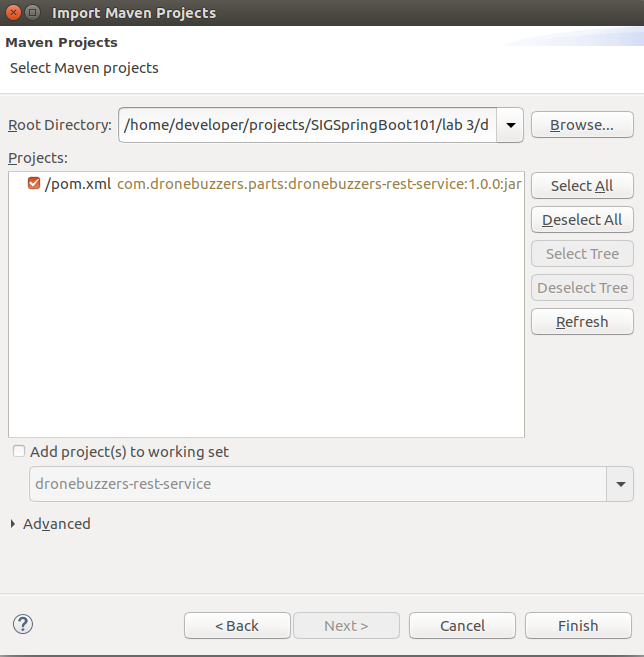


Now, start Eclipse and then import the maven project:



Select the pom file from your project directory:

/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers/pom.xml



Now, we need to copy the business logic into the code: the impl directory in the input directory

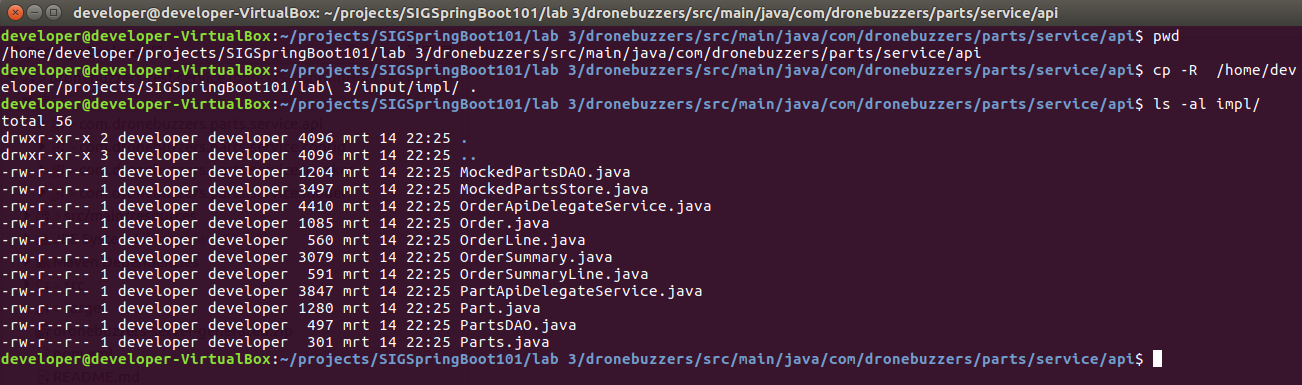
/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers/pom.xml

must now be copied to the right directory in the project:

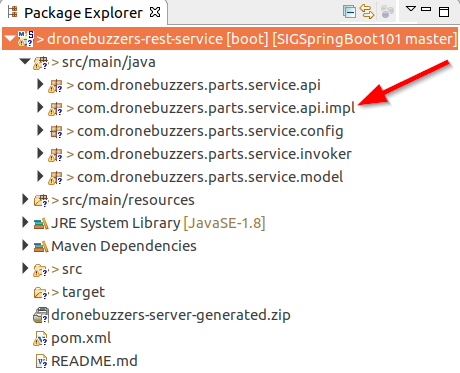
/home/developer/projects/SIGSpringBoot101/

lab 3/dronebuzzers/src/main/java/com/dronebuzzers/parts/service/api

Illustrated in the figure below:



Right-click the project in eclipse and clicking Refresh should make the impl package visible:

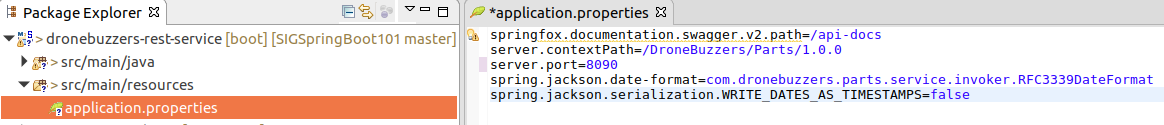


The completed code is also available in:

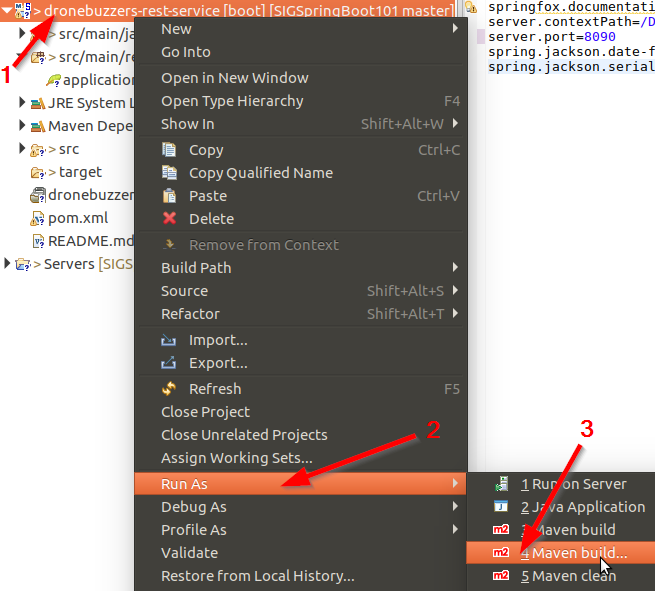
/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers-completed

**Step 4: run and test the API**

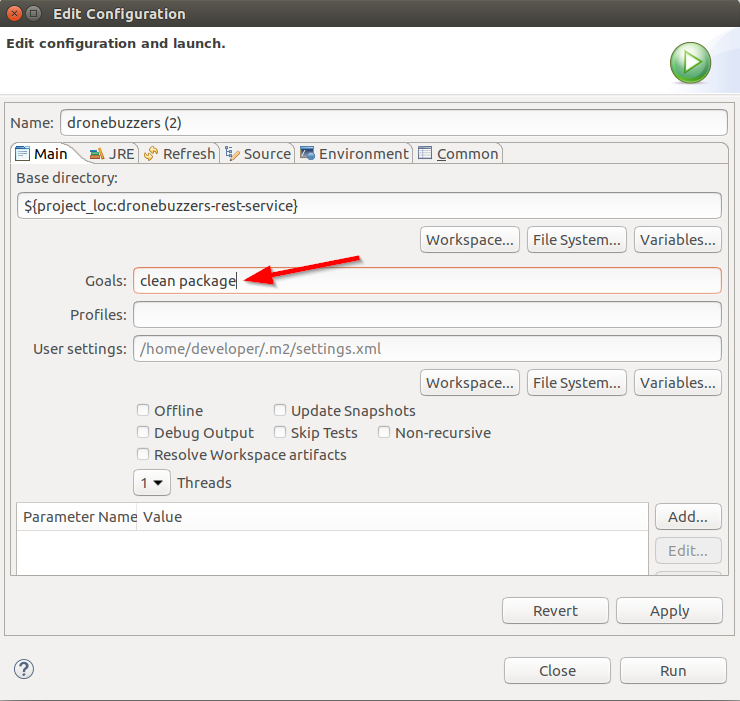
Before actually running the code, have a look at the application.properties file. Change the server.port setting to 8090, as port 8080 may be in use already:



To build the code: right-click the project, click ‘Run As’ and select the option ‘Maven build…’:

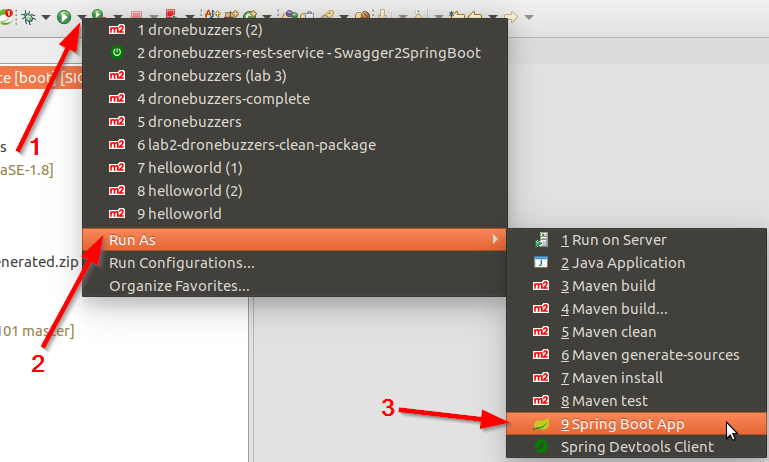


The pop-up window as shown below will be shown:

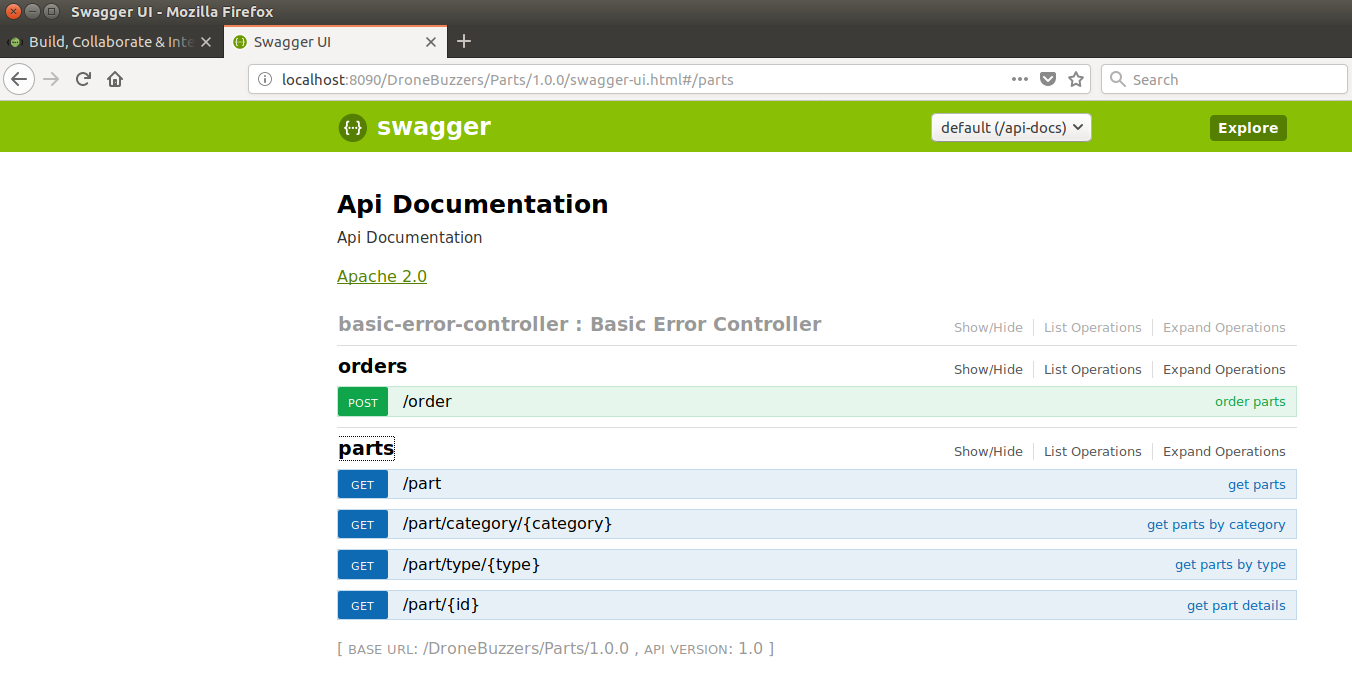


Complete like shown above and click Run. Check in the console that the code is built successfully:

Now that the code is built, it is time to run it:



Verify that the code is running by going to url  
<http://localhost:8090/DroneBuzzers/Parts/1.0.0/swagger-ui.html> in your browser:



For testing, start Postman  and import the Collection of Postman tests for lab 3 from location

/home/developer/projects/SIGSpringBoot101/lab 3/postman

Test the interface with the last 5 operations:

