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, followed by generating code for a Spring Boot 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.

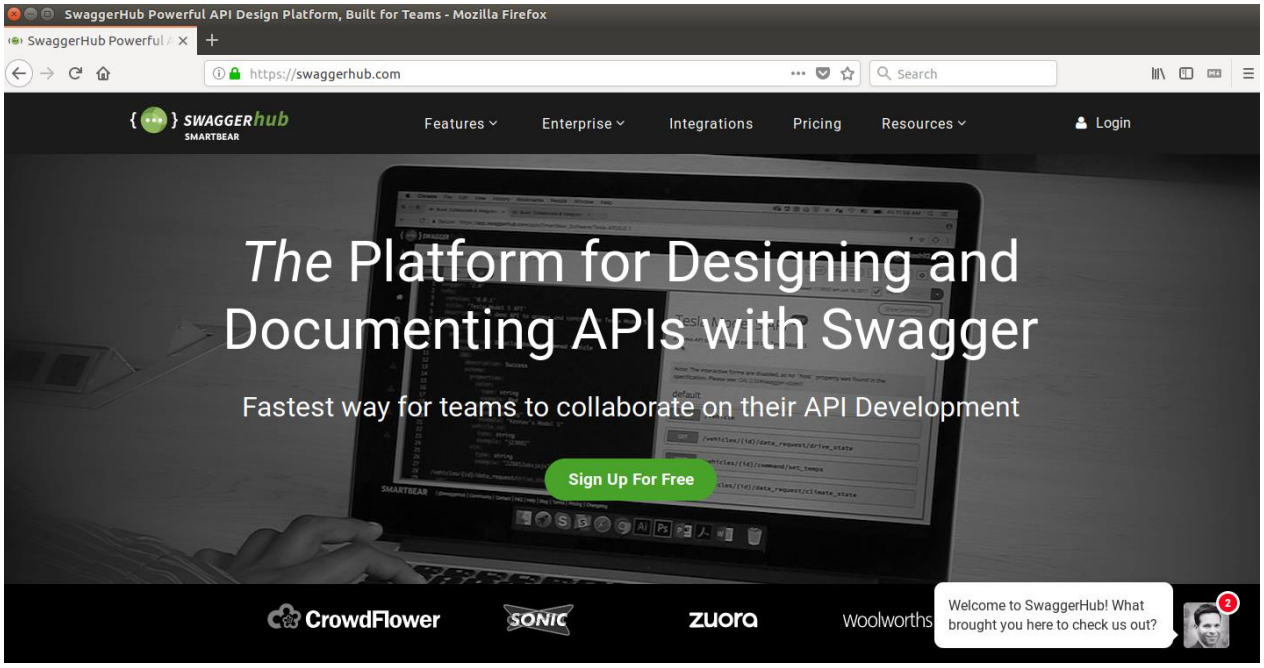
The lab uses the Swagger/OpenAPI 2.0 specification, and not (yet) the OpenAPI 3.0 specification: the tooling seems a bit more stable…

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

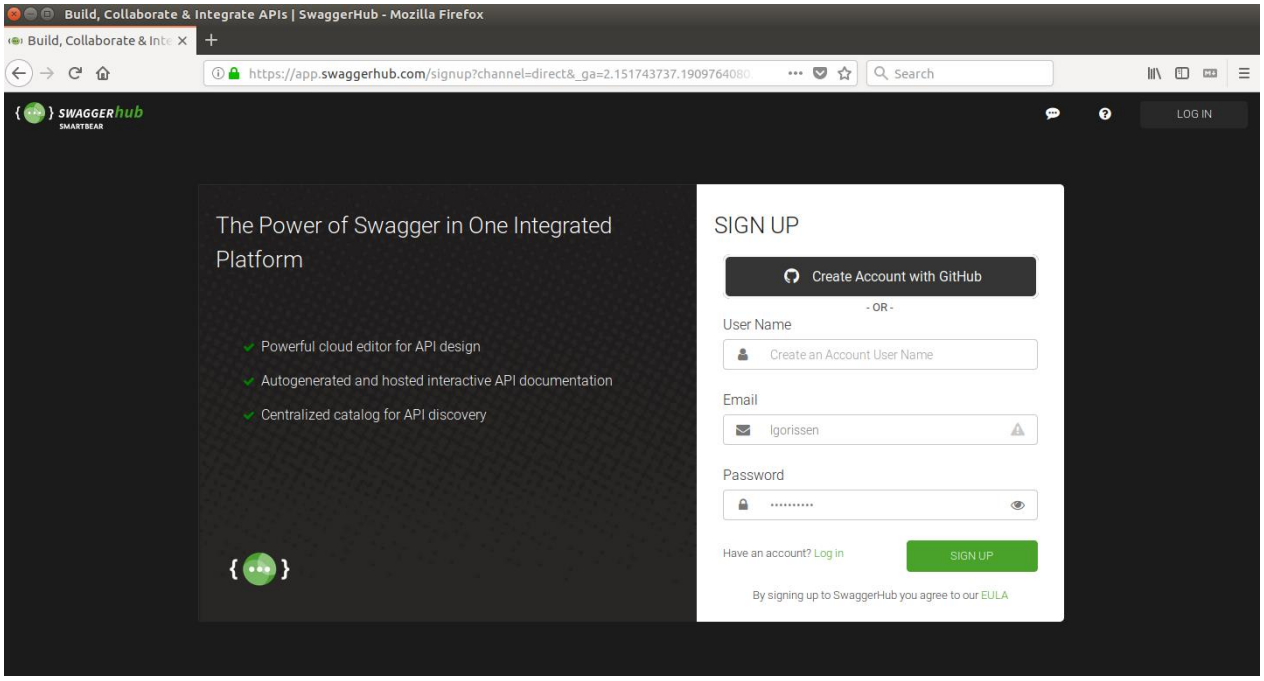
- You are logged in under user/password: developer/welcome01
- You have updated the labs running the `git pull` command in the lab workspace directory `/home/developer/projects/SIGSpringBoot101`

1. Registration with SwaggerHub

Goto swaggerhub.com in your VirtualBox machine:

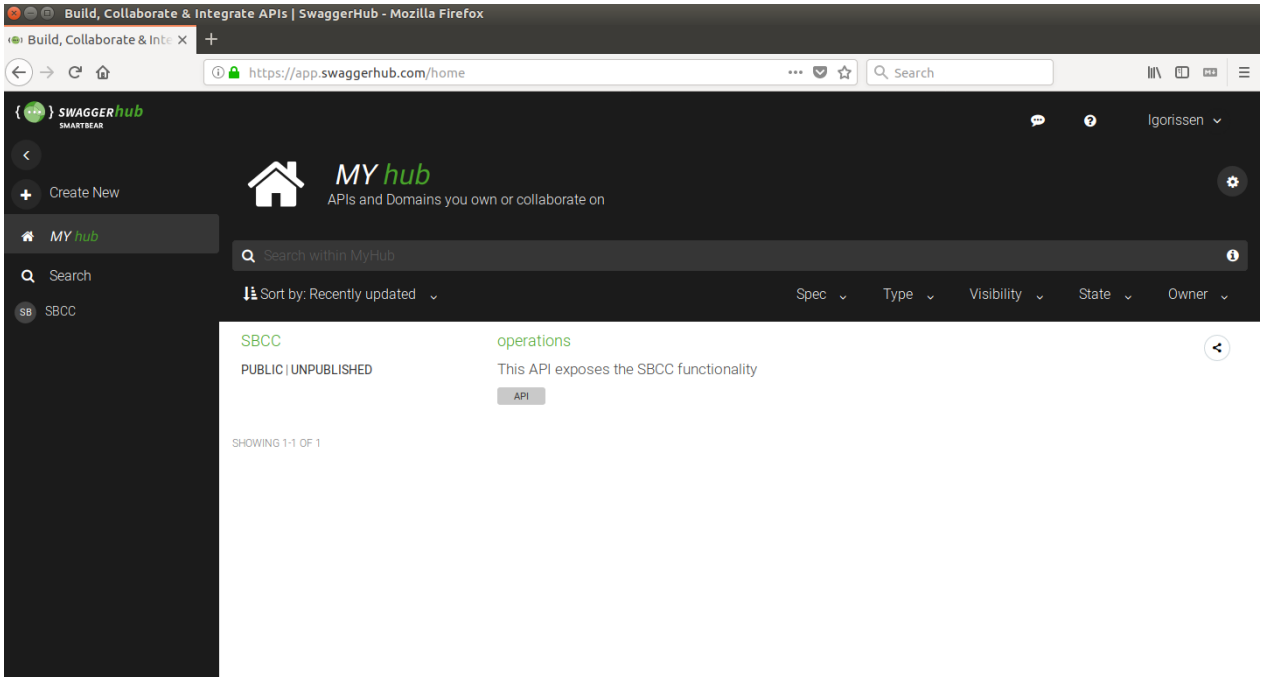


Click the 'Sign Up For Free' button:



Here, you 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.

2. 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. We will first create a simple API in SwaggerHub.

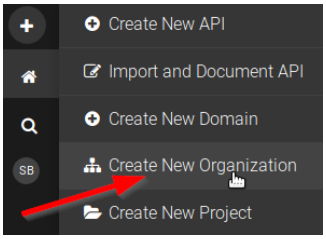
The following steps will be done:

- 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

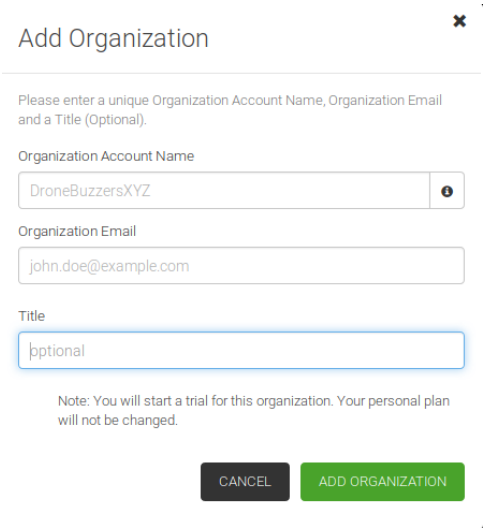
Log in into SwaggerHub, and click the + icon to start adding a new interface:

Start by creating a new organization: Create New Organization



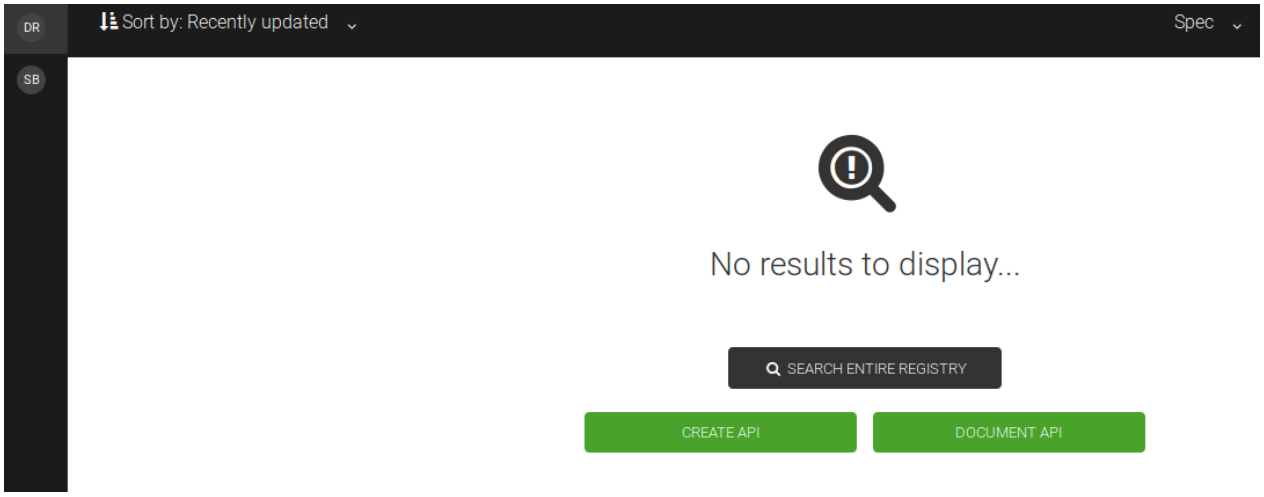
Complete the form as shown below, taking two things into account:

- Organization Account Name: use your own name, e.g. DroneBuzzersXYZ where XYZ are your initials
- Organization e-mail: use your own e-email addres

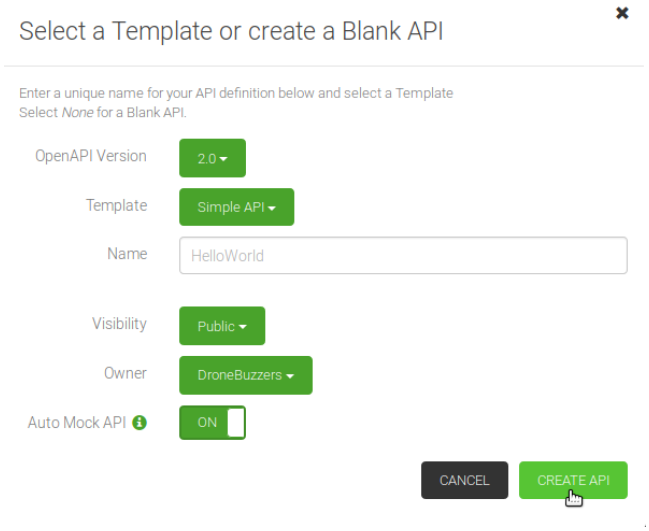


Important: the Organization Account Name that you entered in the above form will appear as part of the URL of the service endpoint. Please note that the screenshots may refer to DroneBuzzers, where you will have the organization name that you entered yourself. Where necessary, we will address the difference…

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



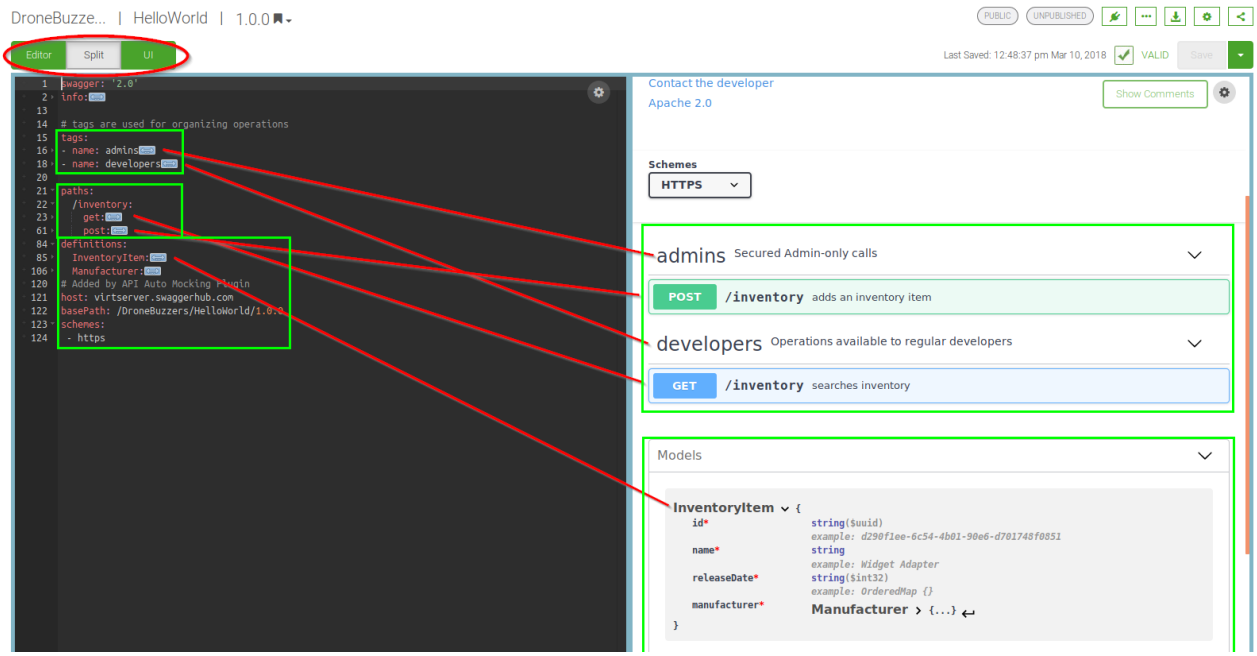
Click on the 'CREATE API' button to start adding the API. Complete the resulting form as shown below:



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:



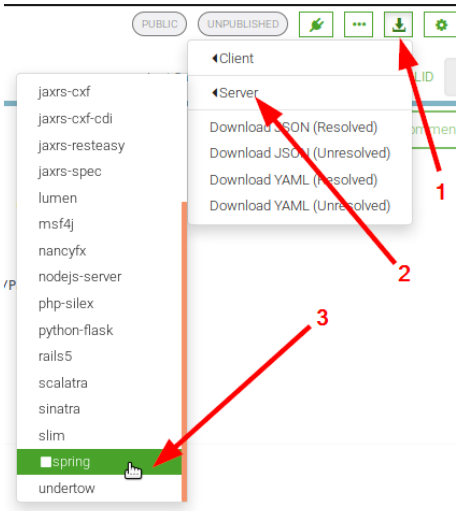
Now is a good moment to spend some time to understand what we're looking at:

- 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 description 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 into the API definition.

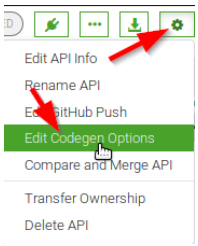
Step 2: generating code in SwaggerHub

SwaggerHub can also generate code for an API definition: for both client and server side. And for many languages. These code generation options are a bit difficult to find: the 3 magic clicks are shown below:



Not how many language options there are: 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:

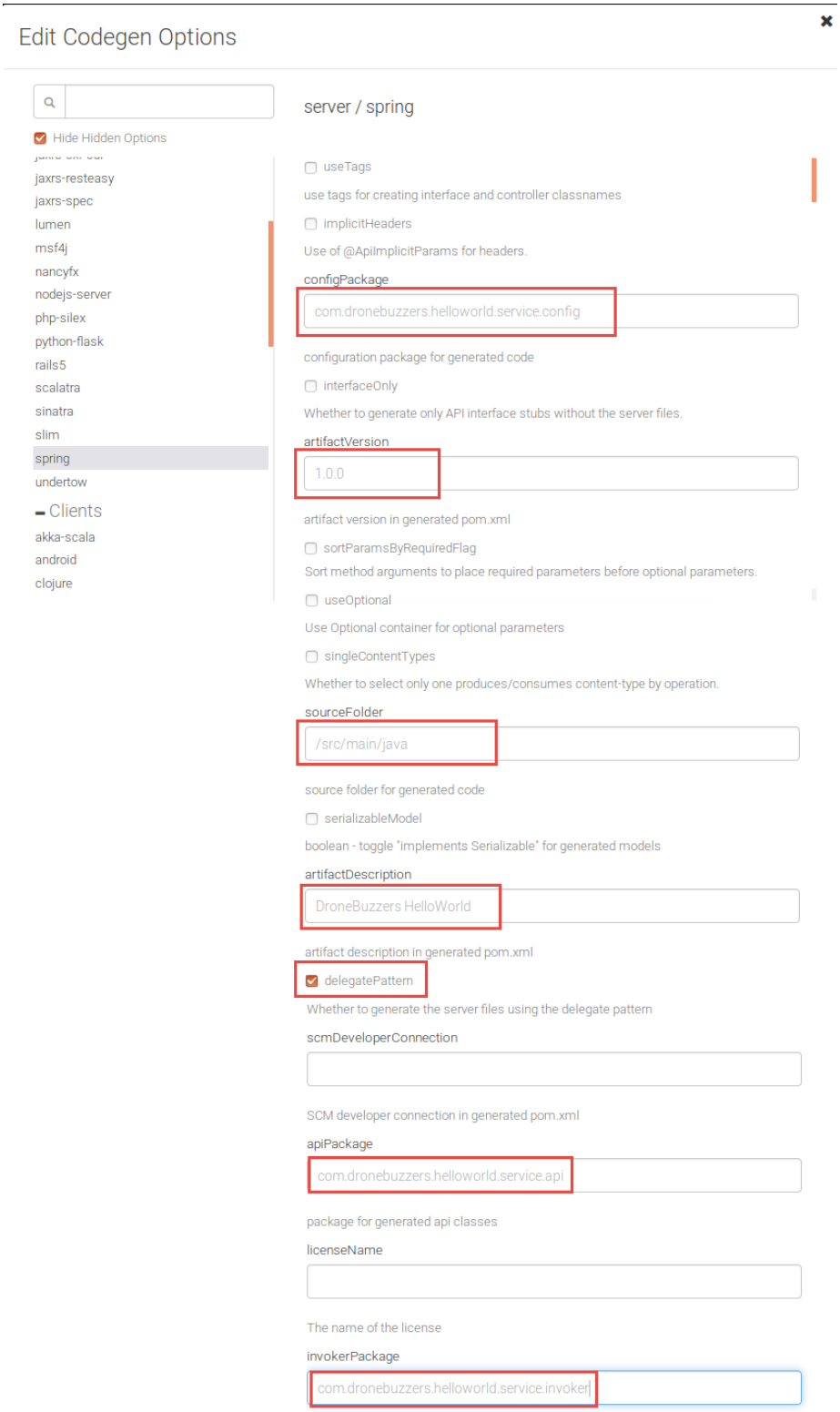


In the pop-up window, select spring in the Servers section and then complete the settings as shown in the table below – and in the screenshot shown underneath:

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	*insert your own email>*
developerOrganizationUrl	*https://www.amis.nl*
fullJavaUtil	*not checked*
bigDecimalAsString	*not checked*
ensureUniquerParams	*not checked*
basePackage	com.dronebuzzers.helloworld.service
developerName	*<insert your own name>*
allowUnicodeIdentifiers	*not checked*
java8	*checked*
Title	DroneBuzzers HelloWorld
localVariablePrefix	*not checked*
groupId	com.dronebuzzers.helloworld
Library	Spring-boot Server application using the SpringFox integration
scmConnection	
hideGenerationTimestamp	
Async	*not checked*
modelPackage	com.dronebuzzers.helloworld.service.model
developerOrganization	*AMIS*
artifactUrl	

And in screenshots:


Edit Codegen Options
server / spring
Hide Hidden Options
jaxrs-resteasy
jaxrs-spec
lumen
msf4j
nancyfx
nodejs-server
php-silex
python-flask
rails5
scalatra
sinatra
slim
spring
undertow
Clients
akka-scala
android
clojure
useTags
use tags for creating interface and controller classnames
implicitHeaders
Use of @ApiImplicitParams for headers.
configPackage
com.dronebuzzers.helloworld.service.config
configuration package for generated code
interfaceOnly
Whether to generate only API interface stubs without the server files.
artifactVersion
1.0.0
artifact version in generated pom.xml
sortParamsByRequiredFlag
Sort method arguments to place required parameters before optional parameters.
useOptional
Use Optional container for optional parameters
singleContentTypes
Whether to select only one produces/consumes content-type by operation.
sourceFolder
/src/main/java
source folder for generated code
serializableModel
boolean - toggle "implements Serializable" for generated models
artifactDescription
DroneBuzzers HelloWorld
artifact description in generated pom.xml
delegatePattern
Whether to generate the server files using the delegate pattern
scmDeveloperConnection
SCM developer connection in generated pom.xml
apiPackage
com.dronebuzzers.helloworld.service.api
package for generated api classes
licenseName
The name of the license
invokerPackage
com.dronebuzzers.helloworld.service.invoker

And:

root package for generated code

dateLibrary

Option. Date library to use

artifactId

helloworld-rest-service

artifactId in generated pom.xml

licenseUrl

The URL of the license

swaggerDocketConfig

Generate Spring Swagger Docket configuration class.

useBeanValidation

Use BeanValidation API annotations

withXml

whether to include support for application/xml content type and include XML annotations in the model (works with libraries that provide support for JSON and XML)

scmUrl

SCM URL in generated pom.xml

responseWrapper

wrap the responses in given type (Future,Callable,CompletableFuture,ListenableFuture,DeferredResult,HystrixCommand,RxOb or fully qualified type)

developerEmail

developer email in generated pom.xml

developerOrganizationUrl

https://www.amis.nl

developer organization URL in generated pom.xml

fullJavaUtil

whether to use fully qualified name for classes under java.util. This option only works for Java API client

bigDecimalAsString

Treat BigDecimal values as Strings to avoid precision loss.

ensureUniqueParams

Whether to ensure parameter names are unique in an operation (rename parameters that are not).

basePackage

com.dronebuzzers.helloworld.service

base package (invokerPackage) for generated code

developerName

Luc Gorissen

developer name in generated pom.xml

allowUnicodeIdentifiers

boolean, toggles whether unicode identifiers are allowed in names or not, default is false

java8

use java8 default interface

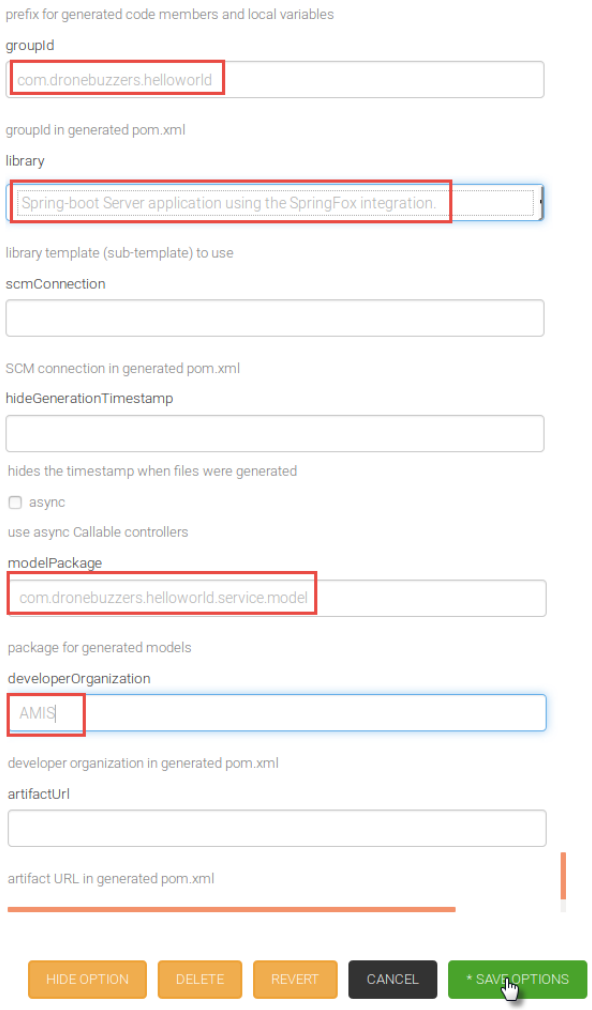
title

DroneBuzzers HelloWorld

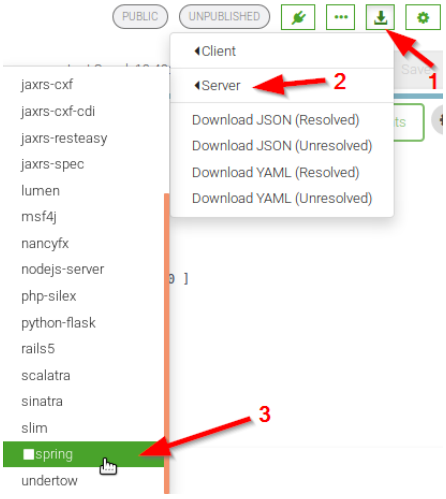
server title name or client service name

localVariablePrefix

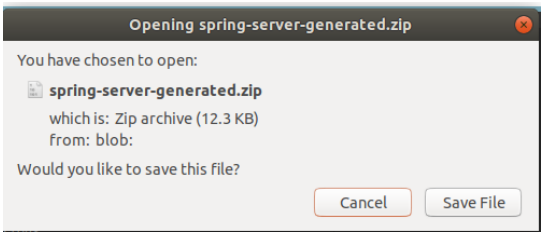
And:



After completing the list, click 'SAVE OPTIONS' and close this screen. Now, the server side code can be generated. Follow these steps:



This will result in a zip file with the generated server side implementation of the API:



Save the file to the local file system in the VM.

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

First, unpack the zip file in the right location: lab 3/helloworld. 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`

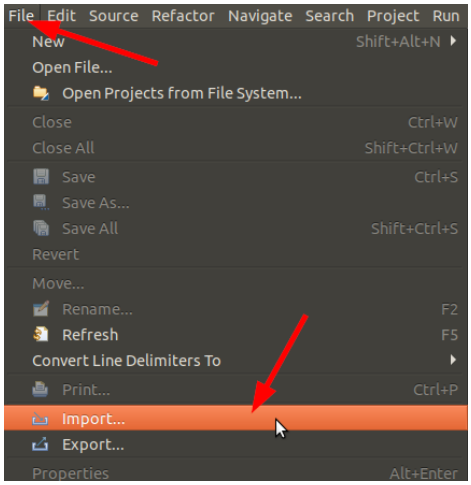
```
developer@developer-VirtualBox: ~/projects/SIGSpringBoot101/lab 3/helloworld

developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$ pwd
/home/developer/projects/SIGSpringBoot101/lab 3/helloworld
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$ cp ../input/helloworld-server-generated.zip .
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$ unzip helloworld-server-generated.zip
Archive:  helloworld-server-generated.zip
  inflating: src/main/java/com/dronebuzzers/helloworld/service/model/Manufacturer.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/model/InventoryItem.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/invoker/RFC3339DateFormat.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/invoker/Swagger2SpringBoot.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/HomeController.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/SwaggerDocumentationConfig.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/CustomInstantDeserializer.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/config/JacksonConfiguration.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/ApiException.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/InventoryApiController.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/NotFoundException.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/InventoryApi.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/ApiOriginFilter.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/ApiResponseMessage.java
  inflating: src/main/java/com/dronebuzzers/helloworld/service/api/InventoryApiDelegate.java
  inflating: src/main/resources/application.properties
  inflating: README.md
  inflating: pom.xml
  inflating: .swagger-codegen/VERSION
developer@developer-VirtualBox:~/projects/SIGSpringBoot101/lab 3/helloworld$
```

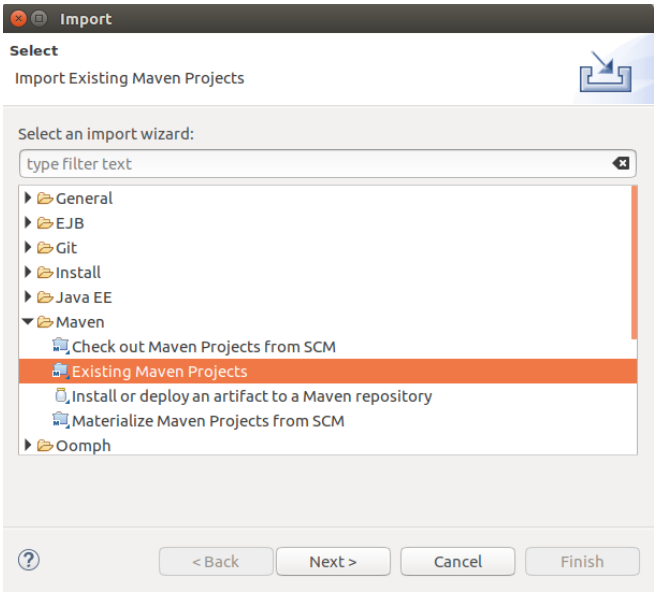
Remove the `helloworld-server-generated.zip` file from the `~/projects/SIGSpringBoot101/lab 3/helloworld` directory.

Next, open Eclipse STS

In the Eclipse STS menu bar, click on File and then Import:

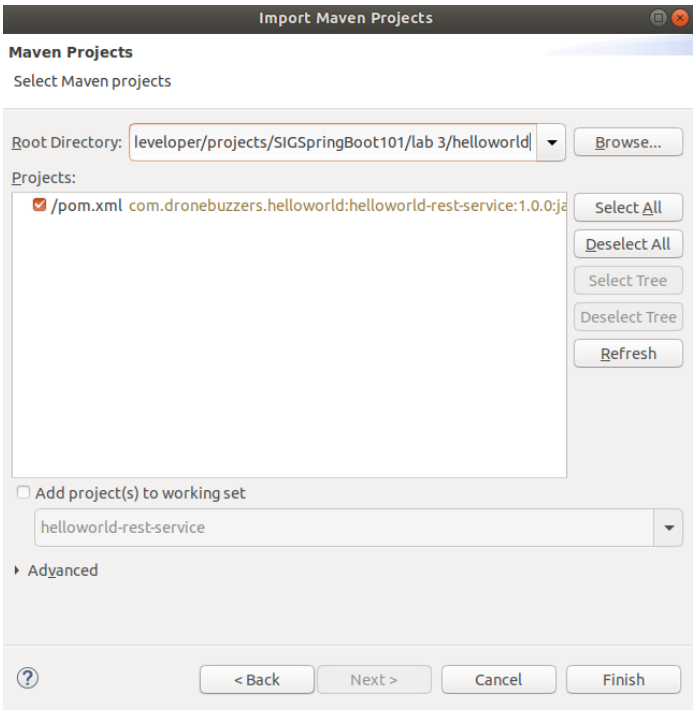


In the resulting Import pop-up, select ‘Existing Maven Projects’:

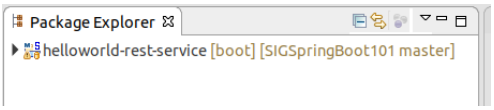


Click Next and then:

- set the Root Directory: `/home/developer/projects/SIGSpringBoot101/lab 3/helloworld`
- select the pom file



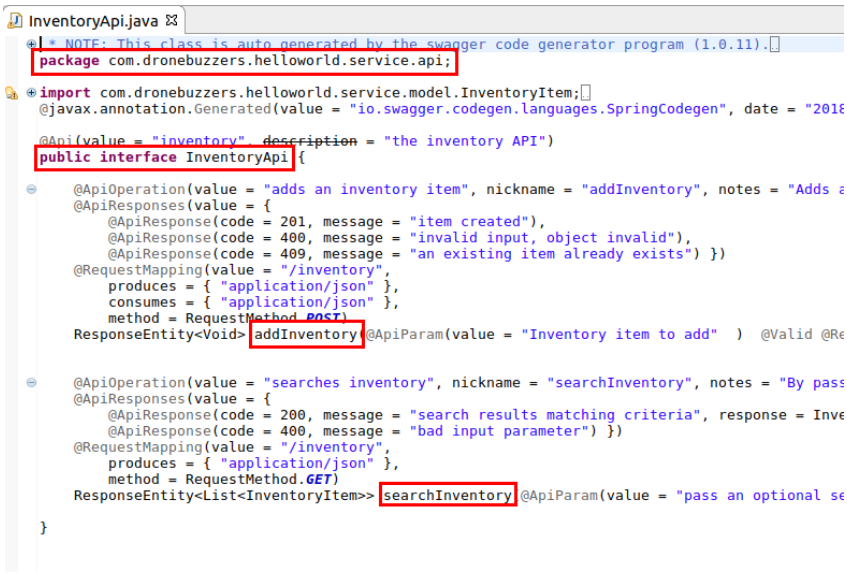
Click Finish and the helloworld-rest-service project should become visible in the Package Explorer:



This is a good moment to take some time to examine all the server side code that SwaggerHub has generated.

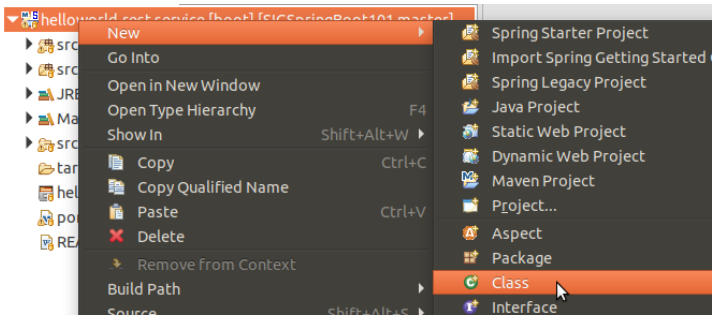
It is important to understand that the project that we have generated only has the API implementation: all business logic is missing. Now, we will continue to add the business logic.

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



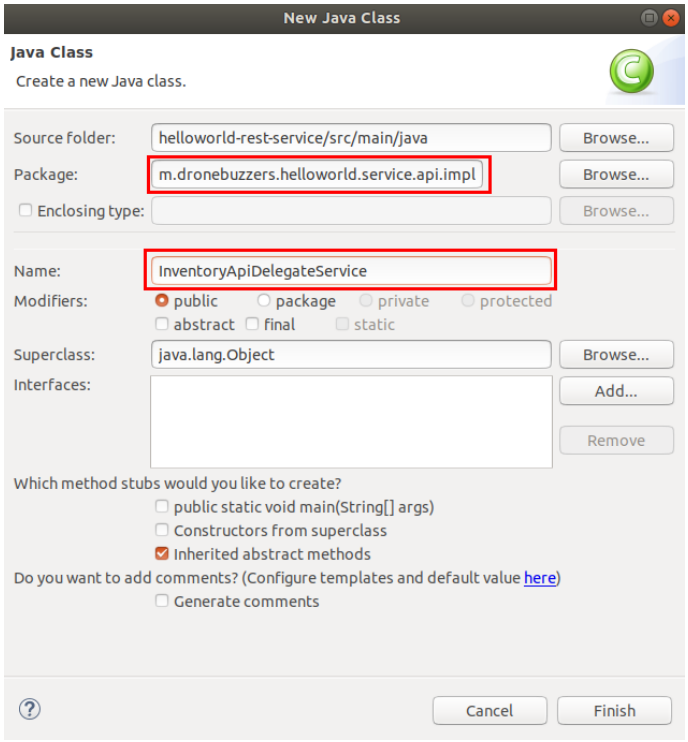
So, we will add a new Java class named InventoryApiDelegateService in the package com.dronebuzzers.helloworld.service.api.impl.

Right-click the project, select New and select Class:



Complete like shown below:

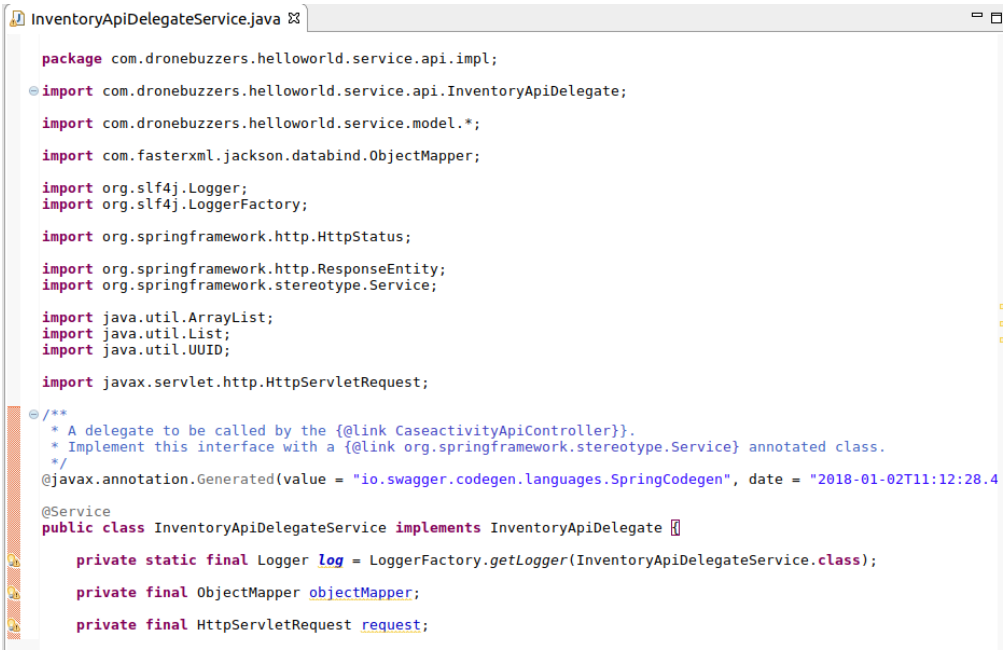
- Package: com.dronebuzzers.helloworld.service.api.impl
- Name: InventoryApiDelegateService



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:

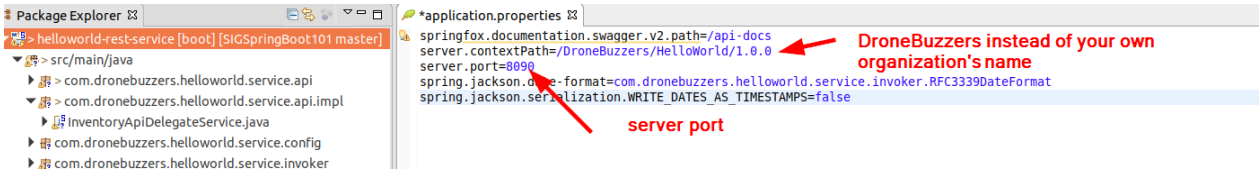


The code is now completed. It defines a single manufacturer (AMIS) with a single inventory item (SIG Spring Boot 101)

Step 4: run and test the API

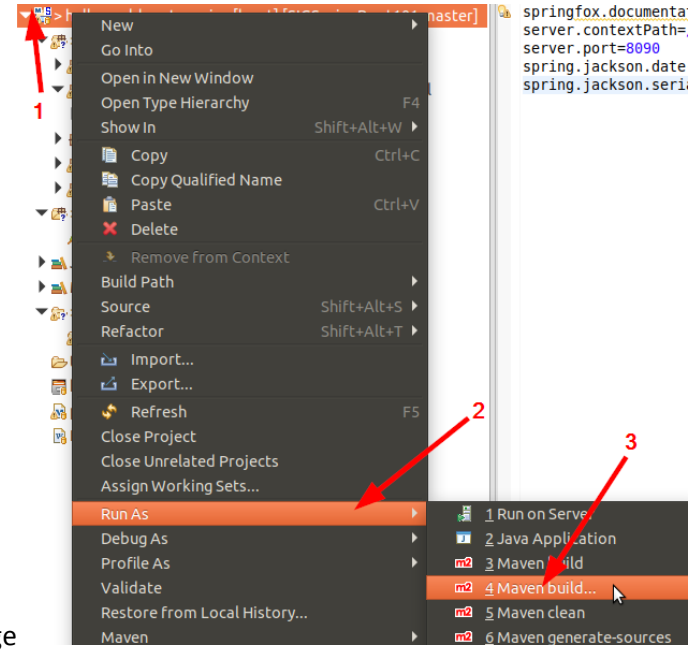
Before actually running the code, have a look at the application.properties file. There are 2 changes that you need to make:

1. Change the server port:
   server.port=8090
2. Change the server contextPath as this will refer to your organization's name:
   server.contextPath=/DroneBuzzers/HelloWorld/1.0.0



We change the above settings so the provided Postman requests will work without changes.

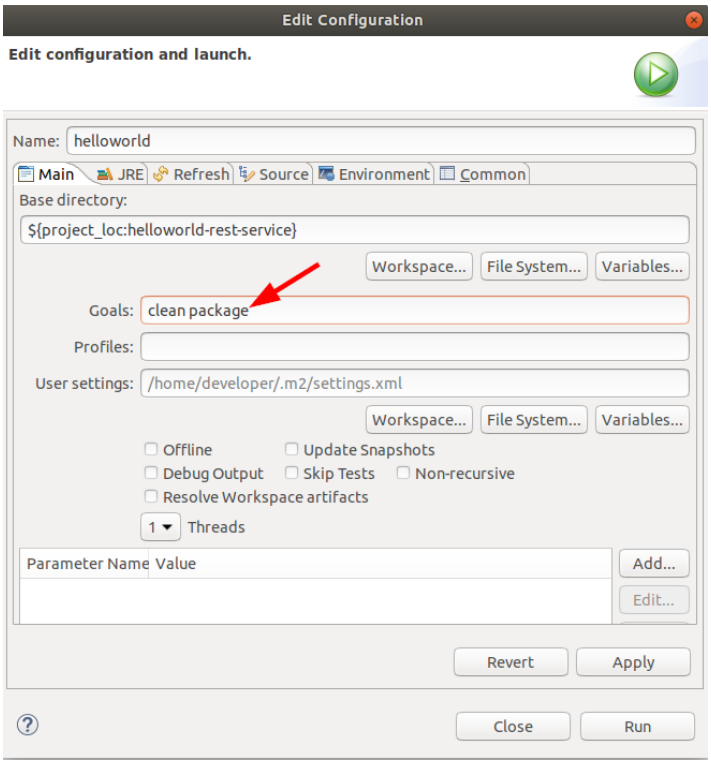
First step is to build the code: right-click the project, click 'Run As' and select the option 'Maven build…':



Clean package

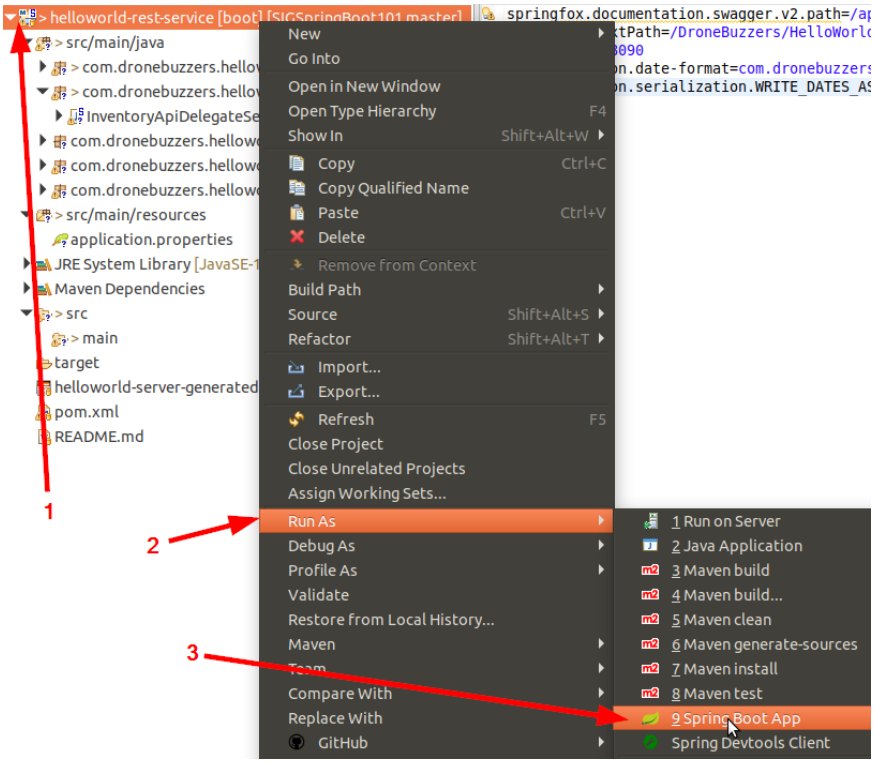
The pop-up window appears as shown below. Complete by setting the Goals:

- Goals: clean package



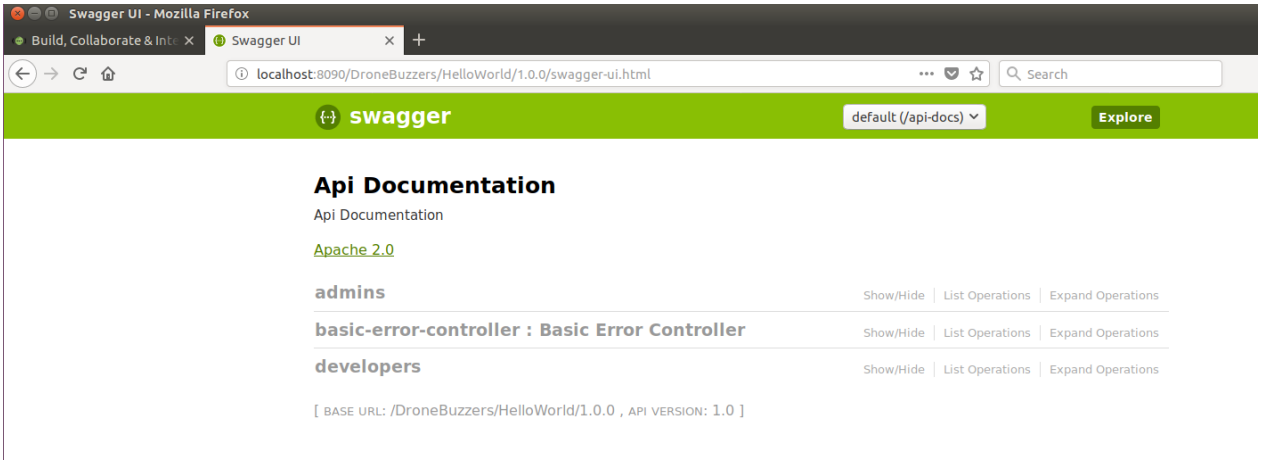
Complete like shown above and click Run. Check in the console that the code is built successfully:

Now that the code is built, it is time to run it. Right-click the project, click 'Run As' and then 'Spring Boot App':



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

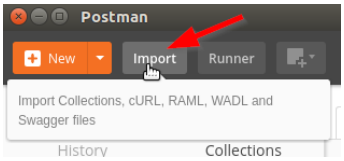
The result should look like:



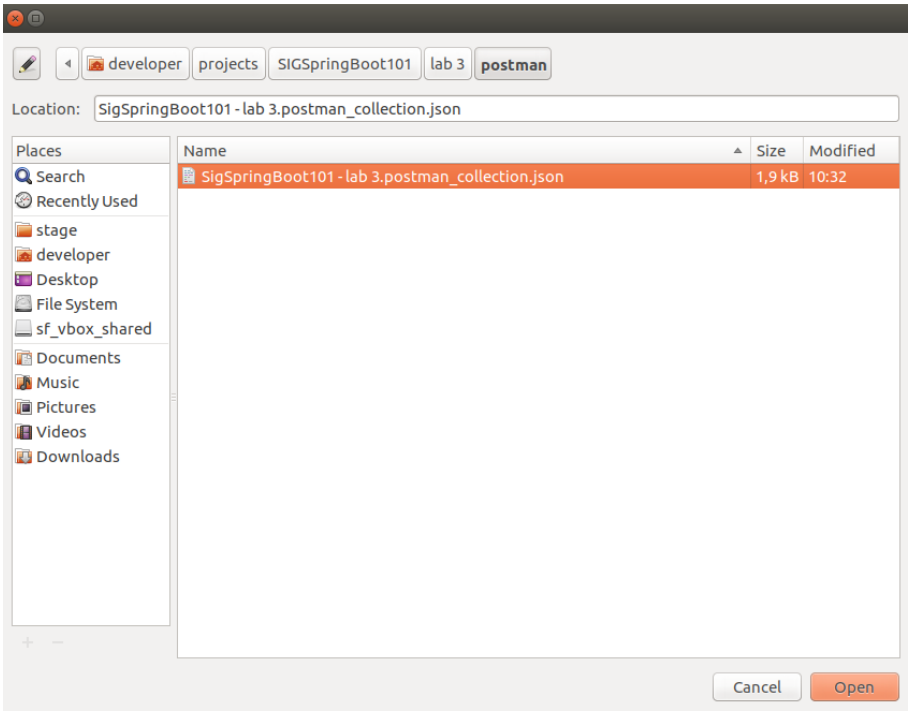
You can check url http://localhost:8090/DroneBuzzers/HelloWorld/1.0.0/api-docs for the raw output...

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

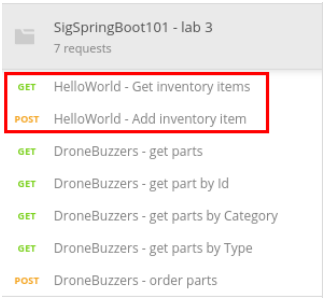
Start Postman and import the Collection of Postman tests for lab 3:



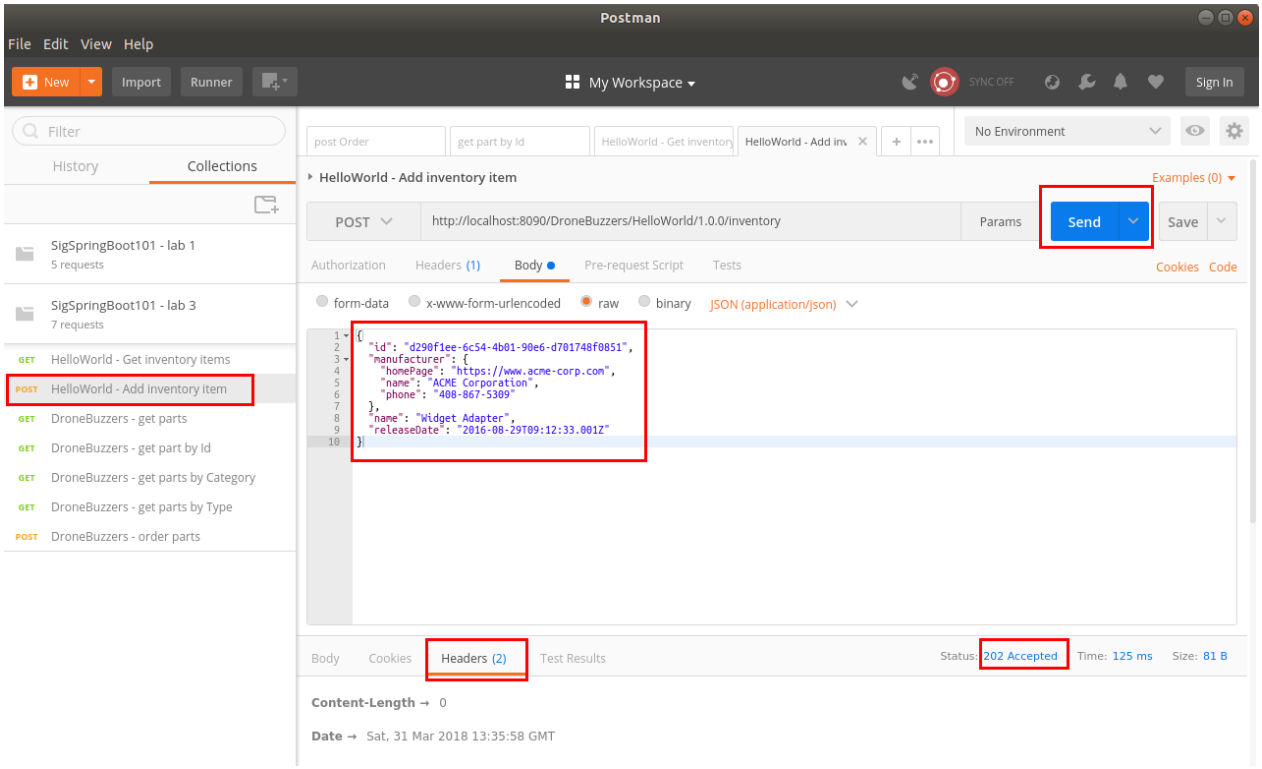
The collection is in the `/home/developer/projects/SIGSpringBoot101/lab 3/postman` directory:



NOTE: the imported collection has 7 requests: in this stage, you should use the first two HelloWorld requests. The other requests will be used later on in this lab.



Next, test both operations:



3. 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:

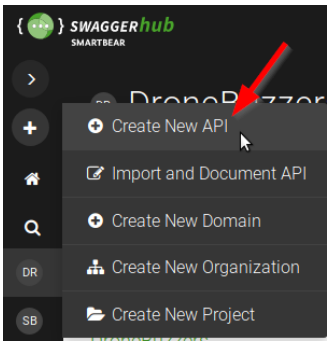
- Step 1: create the API specification in SwaggerHub
- Step 2: generating code in SwaggerHub
- Step 3: import into STS Eclipse and add business logic in the code
- Step 4: run and test the API

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

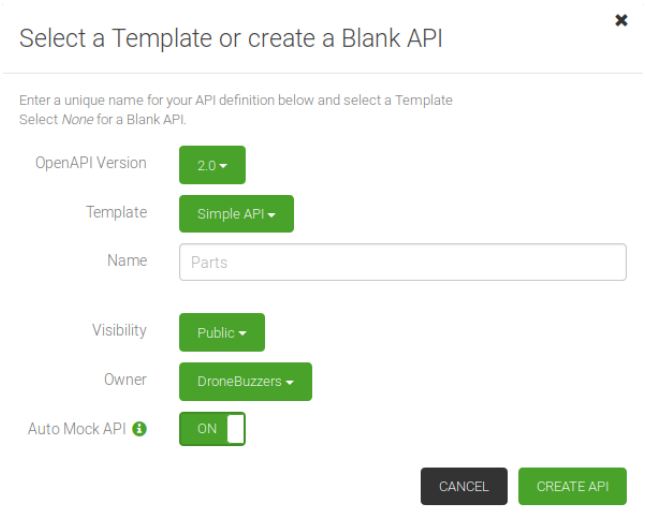
Should you not want to do the complete exercise, some intermediate results are made available in the `/home/developer/projects/SIGSpringBoot101/lab 3/input` directory. Like this text, intermediate results are marked in a box.

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:

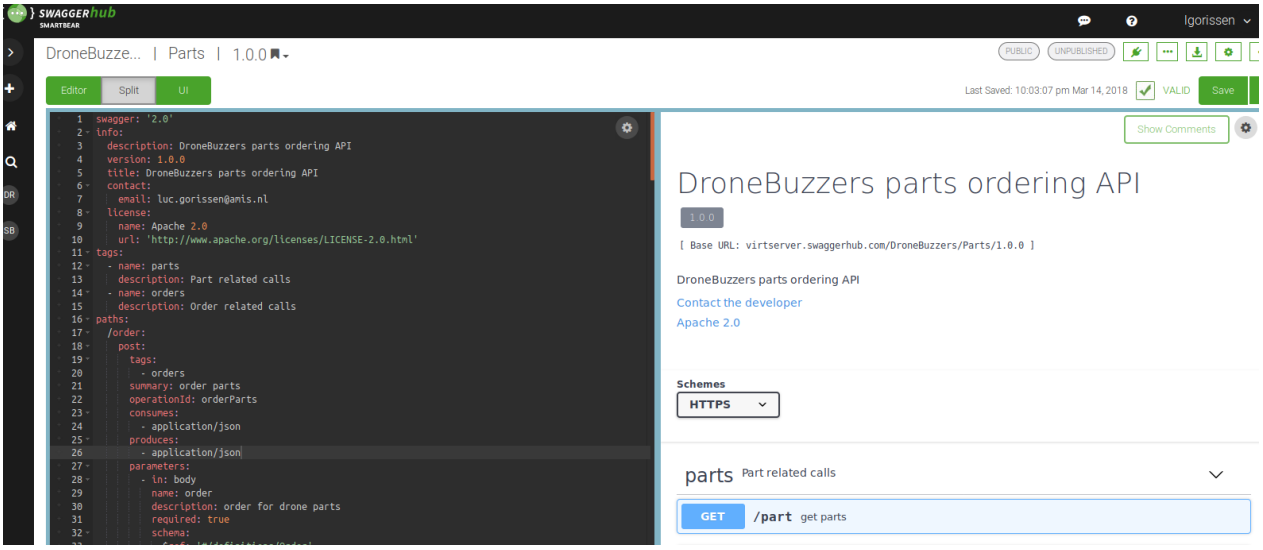


In the editor, replace the contents with the contents of file

`/home/developer/projects/SIGSpringBoot101/lab 3/input/DroneBuzzers_Parts_1.0.0_swagger.json`

The result in SwaggerHub should look somewhat like below:

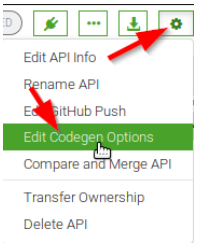
Click OK to let Swagger do the JSON to YAML conversion for our interface definition.



NOTE: also in this part of the exercise, the same considerations apply for the organization's name: DroneBuzzers is assumed throughout the screenshots, whereas you will have your own organization's name

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:

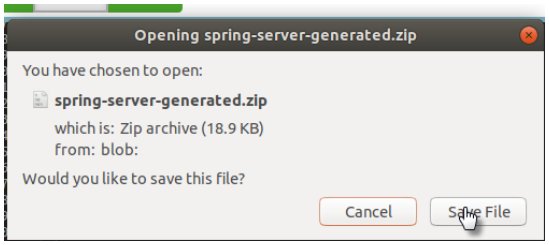


In the pop-up window, select spring in the Servers section and then complete the settings as shown in the table below. This is again quite some work, but can't be avoided: Swagger has separate code generation settings for each API. That does make sense as things like the package names are different for each API.

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	*<insert your own e-mail address>*
developerOrganizationUrl	*https://www.amis.nl*
fullJavaUtil	*not checked*
bigDecimalAsString	*not checked*
ensureUniquerParams	*not checked*
basePackage	com.dronebuzzers.parts.service
developerName	*<insert your own name>*
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. Download the zip file with the generated code for the server side:

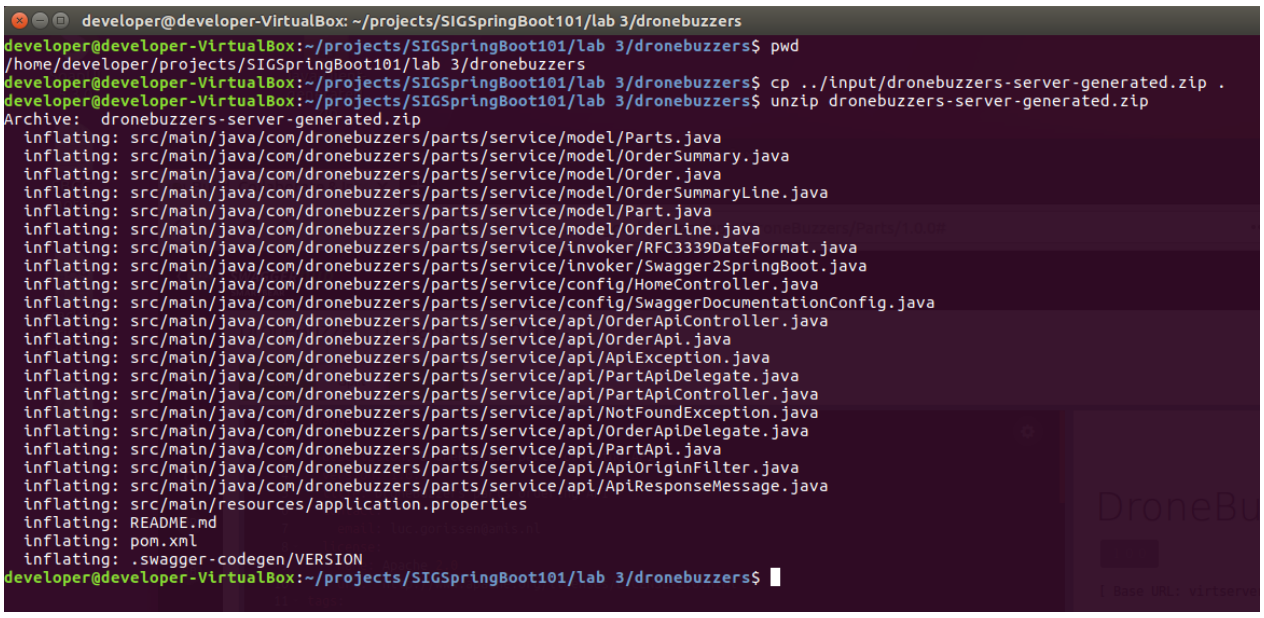


Should you want to skip this step: the generated code is also present in:

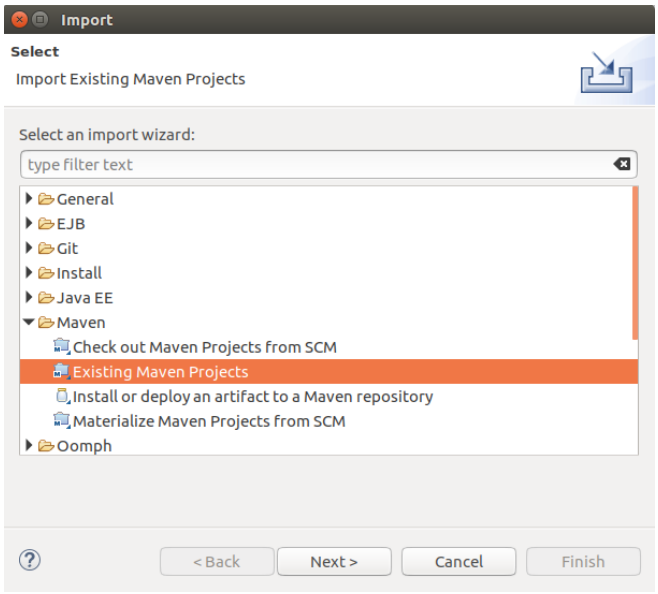
`/home/developer/projects/SIGSpringBoot101/lab 3/input/dronebuzzers-server-generated.zip`

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

The generated code can now be unzipped to directory ..lab 3/dronebuzzers:

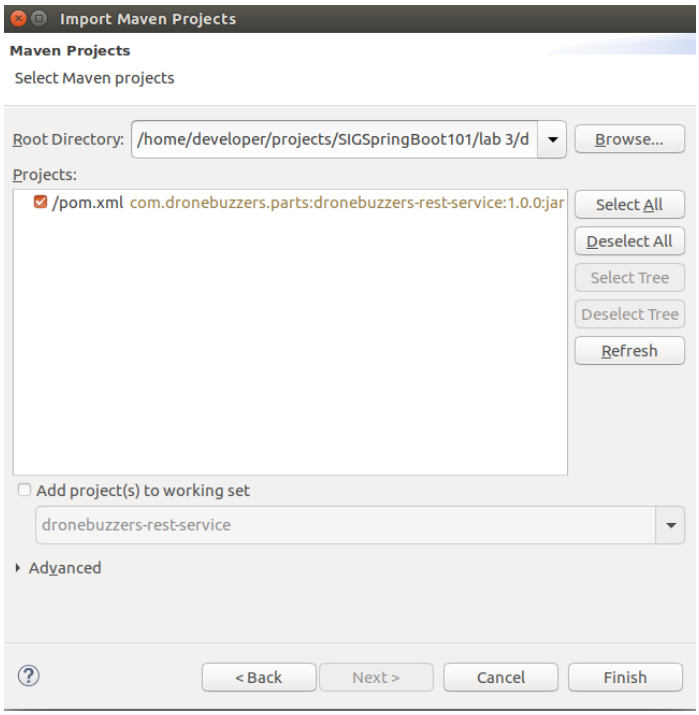


Now, start Eclipse and then import the maven project:



Select the pom file from your project directory:

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



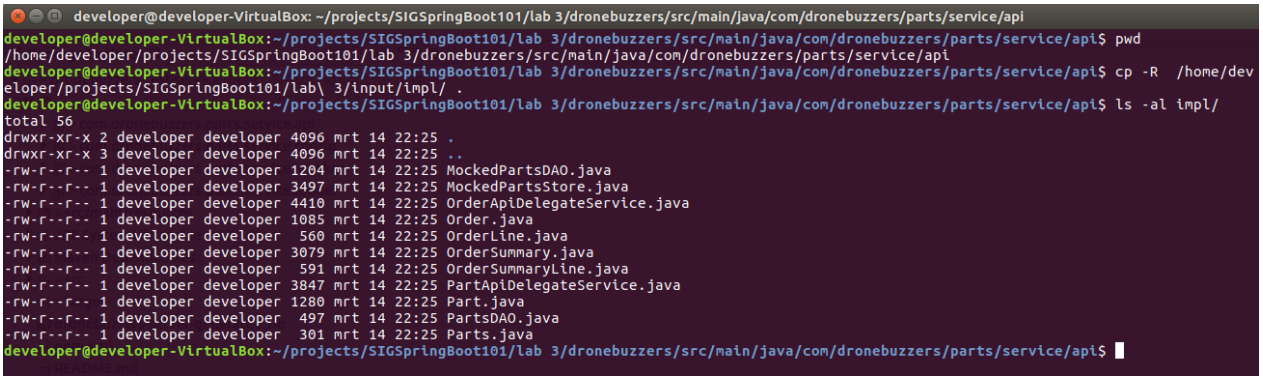
Now, we need to copy the business logic into the code: the impl directory in the input directory

```
lab 3/dronebuzzers/impl
```

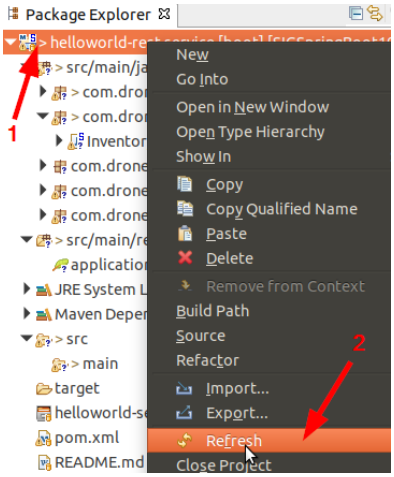
must now be copied to the right directory in the project:

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

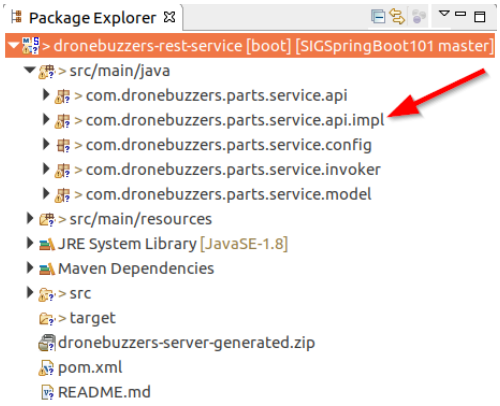
Illustrated in the figure below:



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



Look at the Package Explorer



The completed code is also available in:

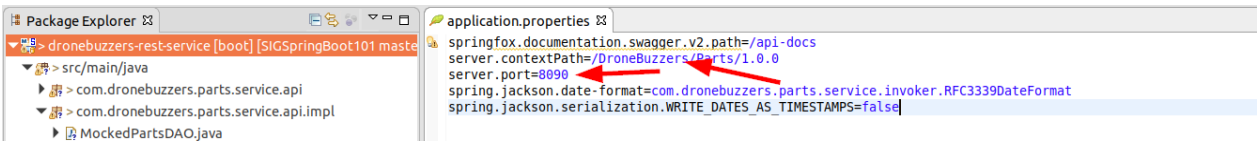
`/home/developer/projects/SIGSpringBoot101/lab 3/dronebuzzers-completed`

Step 4: run and test the API

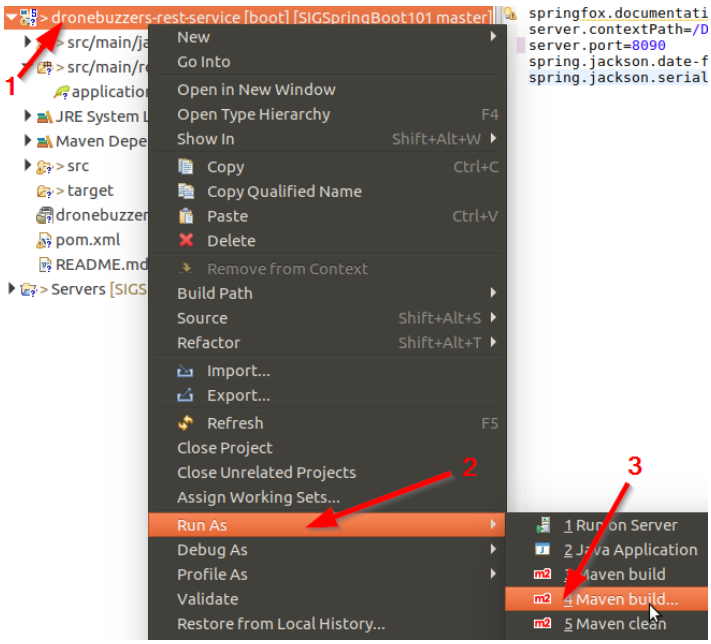
Note: if you have the HelloWorld project still running from earlier in this lab, then now is it a good time to stop it.

Before actually running the code, have a look at the application.properties file. There are 2 changes that you need to make:

1. Change the server port:
   server.port=8090
2. Change the server contextPath as this will refer to your organization's name:
   server.contextPath=/DroneBuzzers/HelloWorld/1.0.0

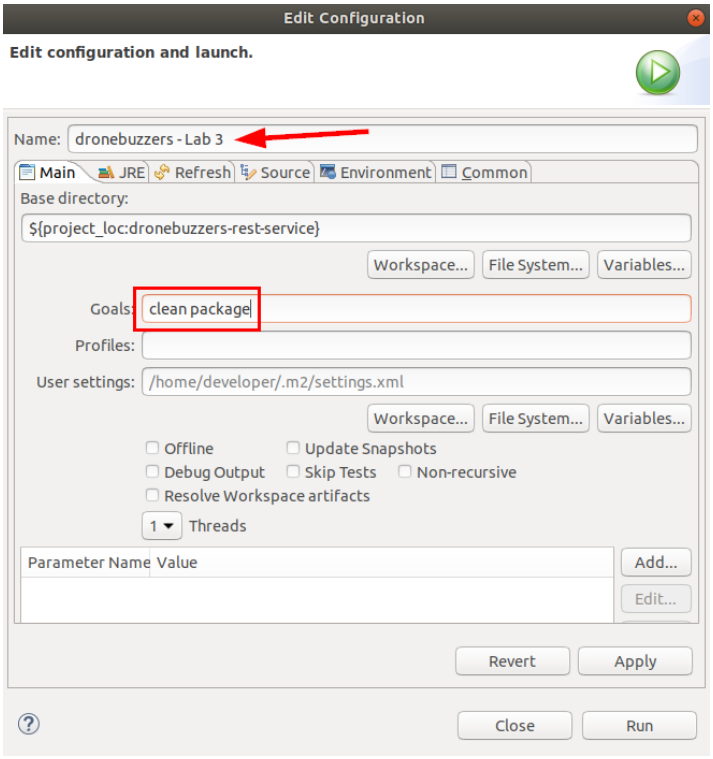


To build the code: right-click the project, click 'Run As' and select the option 'Maven build…':



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

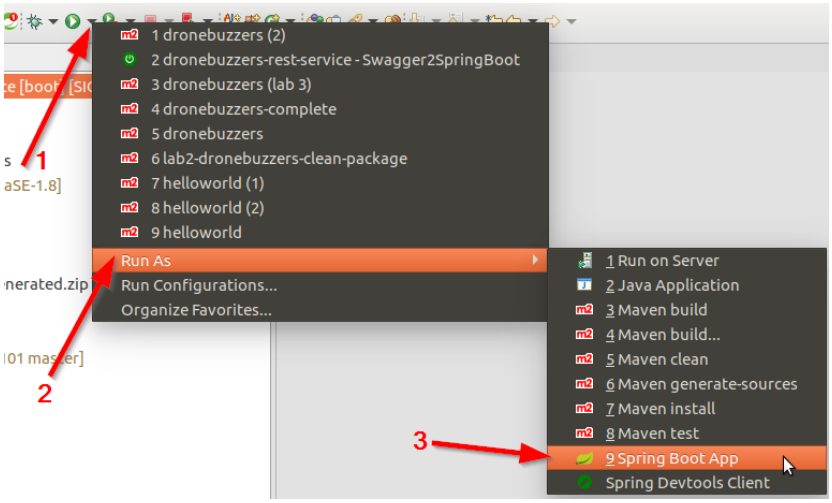
- Name: dronebuzzers – Lab 3
- Goals: clean package



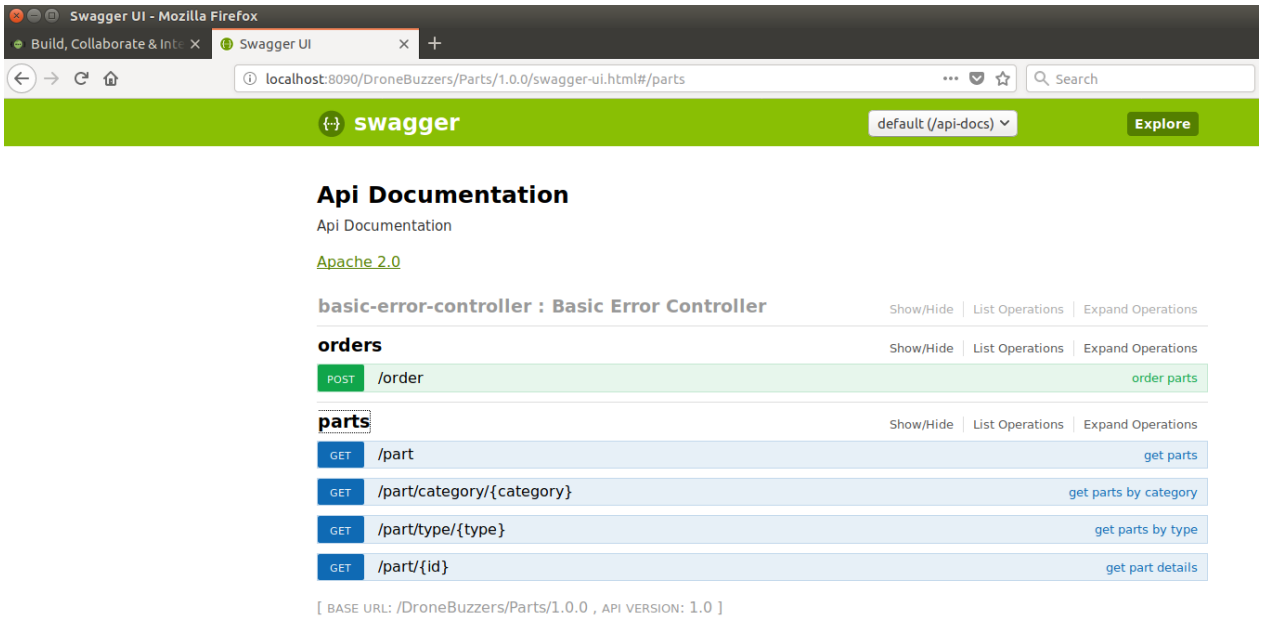
Complete like shown above and click Run.

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  .

If you have not done this during the HelloWorld example: 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: