**Exercise 3**

*Create an embedded Java REST Service using JAX-RS and Spring Boot*

**Prior Knowledge**

Basic understanding HTTP verbs, REST architecture

Some Java coding skill

**Objectives**

Understand what it takes to create REST services. Interact with a REST service using simple web clients in Chrome, on the command line.

See how Gradle can be used.

**Software Requirements**

(see separate document for installation of these)

* Java Development Kit 8
* Gradle build system
* Spring Boot and Jersey
* Visual Studio Code, including extensions:
  + Java
  + Spring Boot
  + SpringInitializr
  + Gradle
* curl
* Google Chrome/Chromium plus Chrome Advanced REST extension

**Overview**

There are many technologies for creating RESTful Web Services in Java. In order to create a simple approach, we are going to use the Java standard for creating REST services, which is called JAX-RS. The “official” Oracle implementation of JAX-RS is Jersey, although there are other implementations such as CXF and RESTEasy.

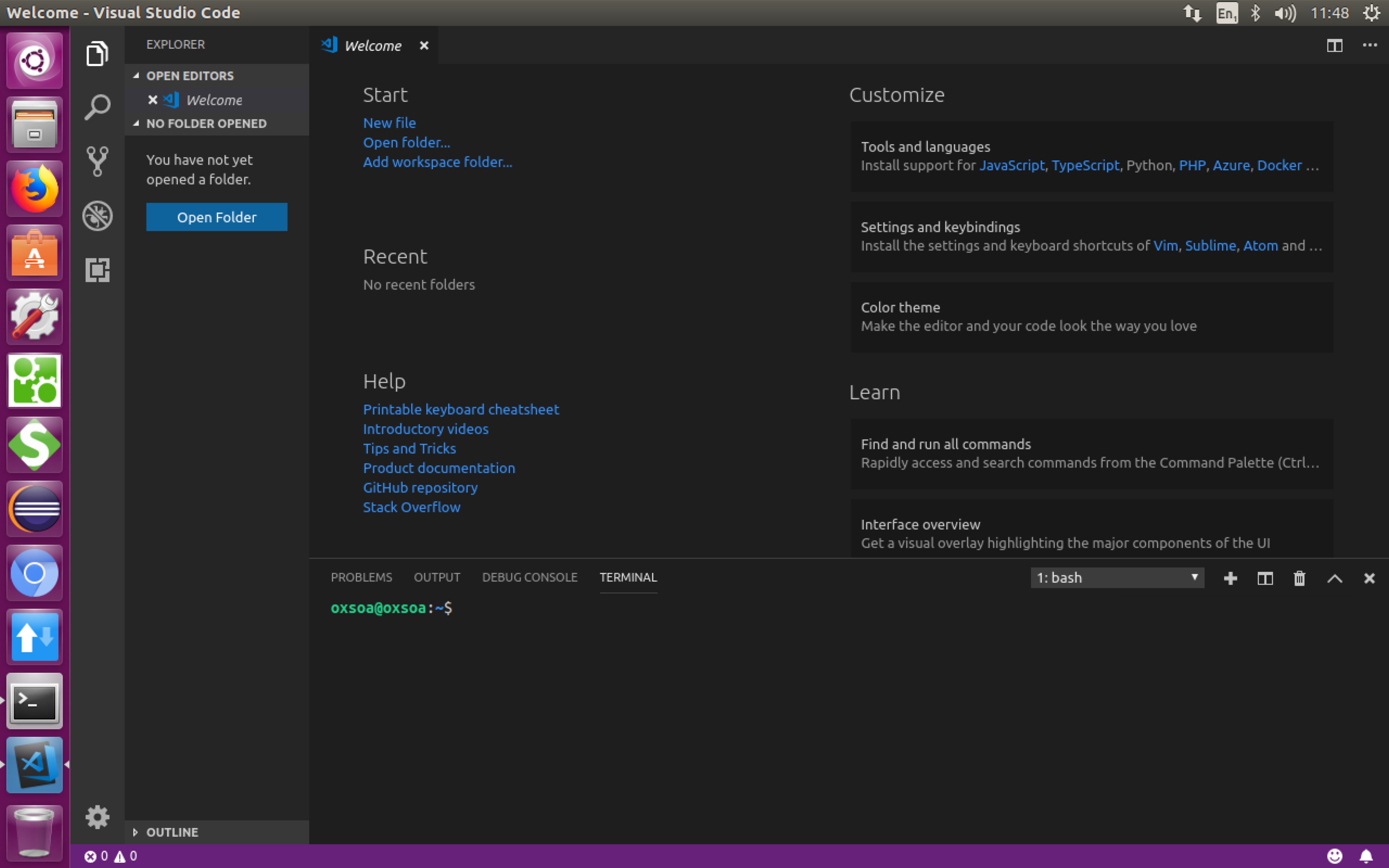
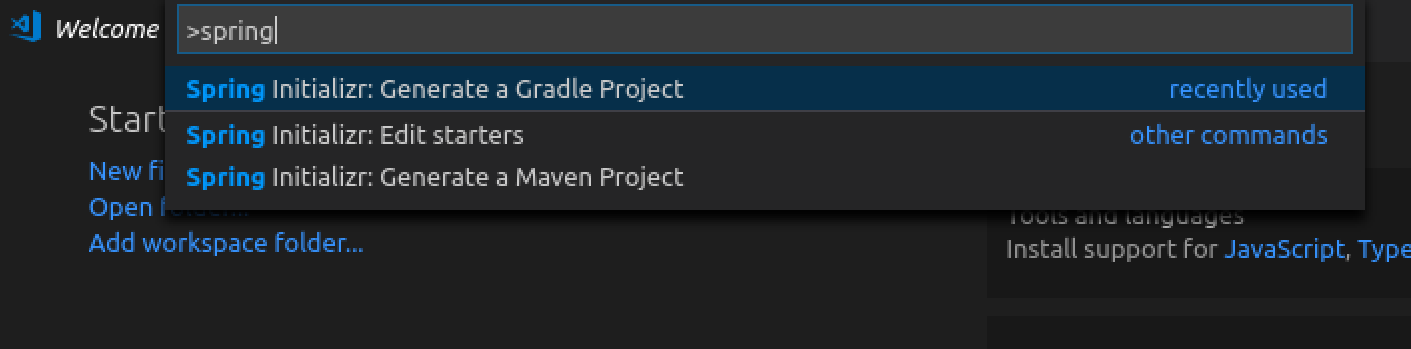
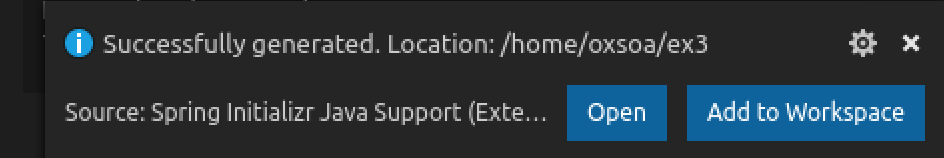
Spring Boot is a framework for running microservices.

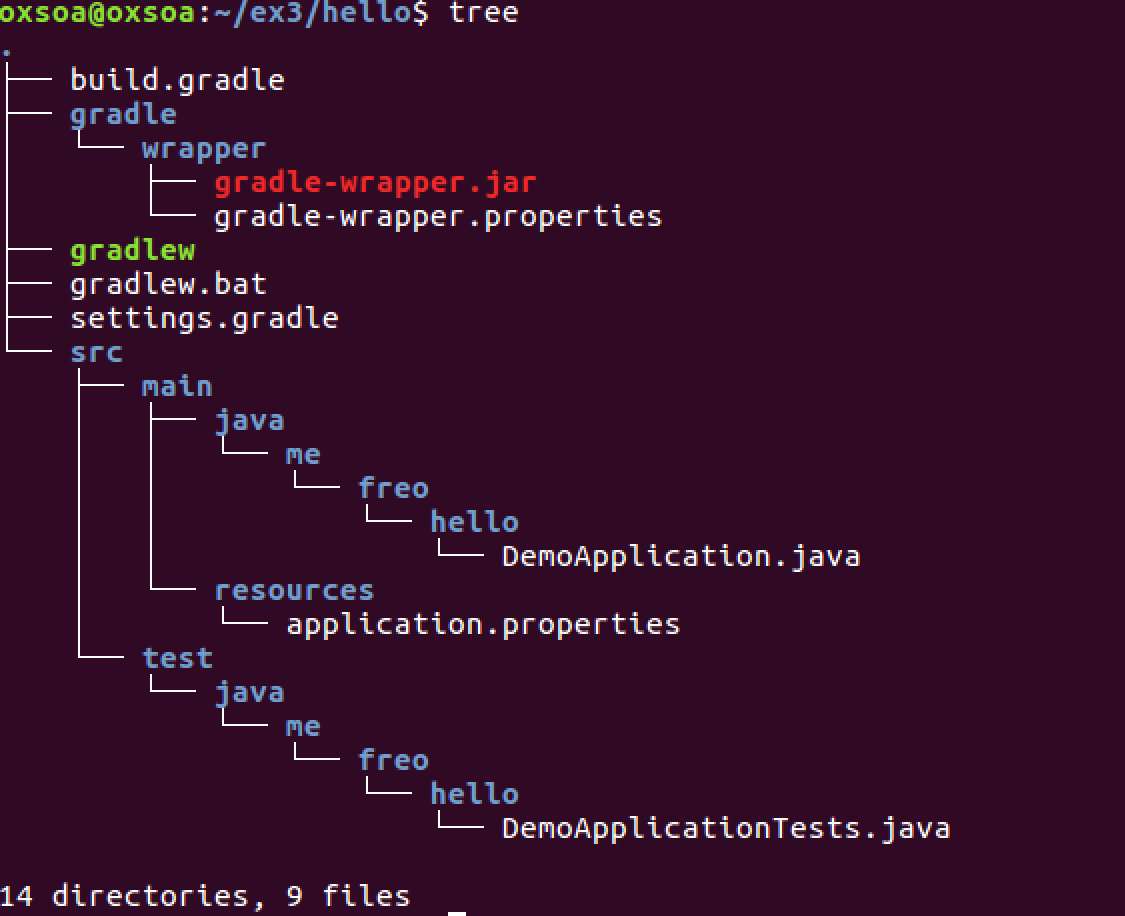
**Steps**

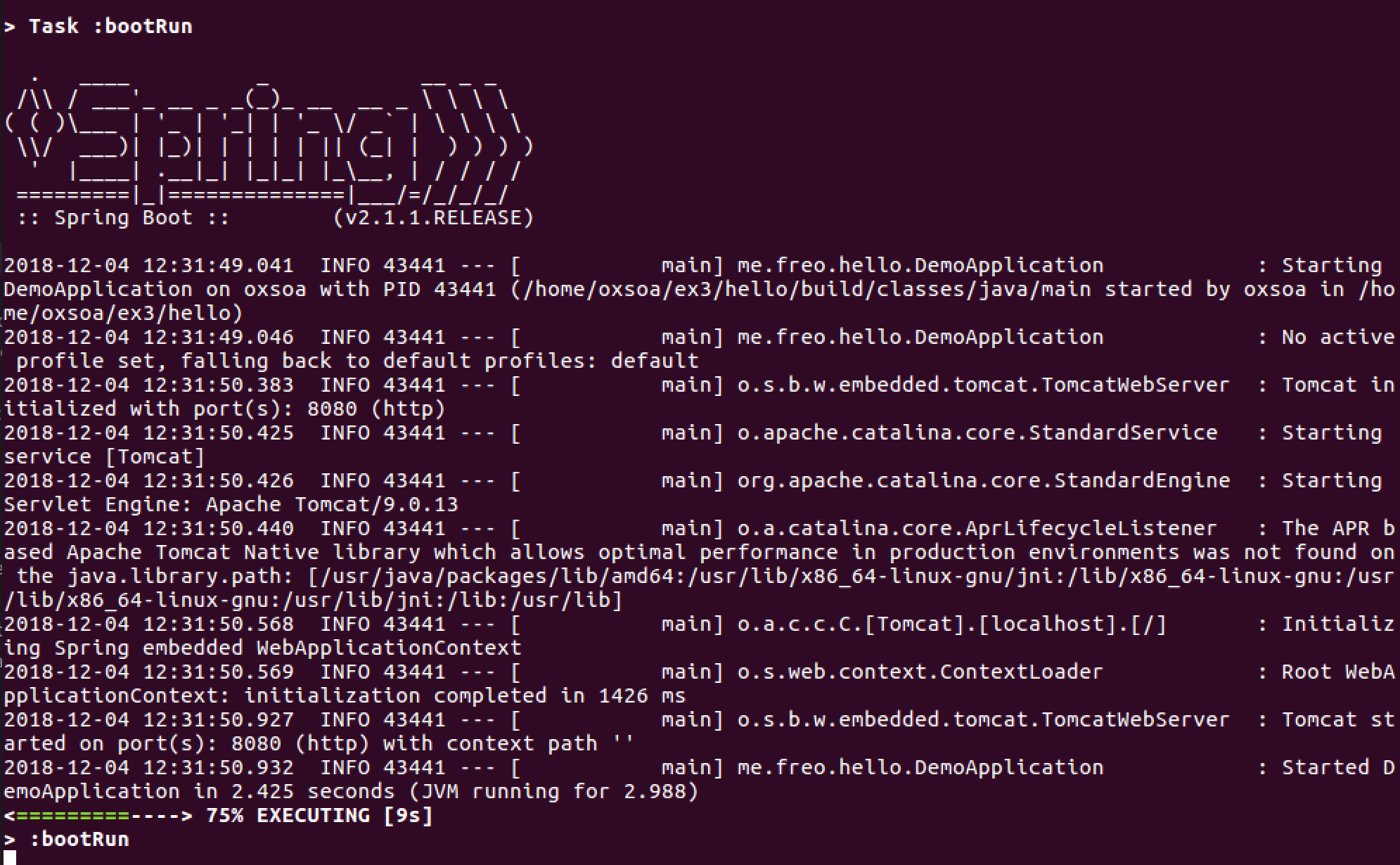
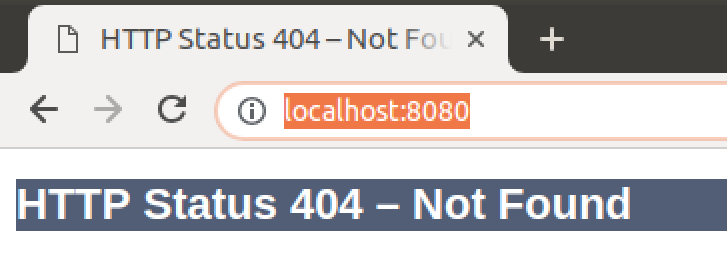
1. Create a new directory:

mkdir ~/ex3

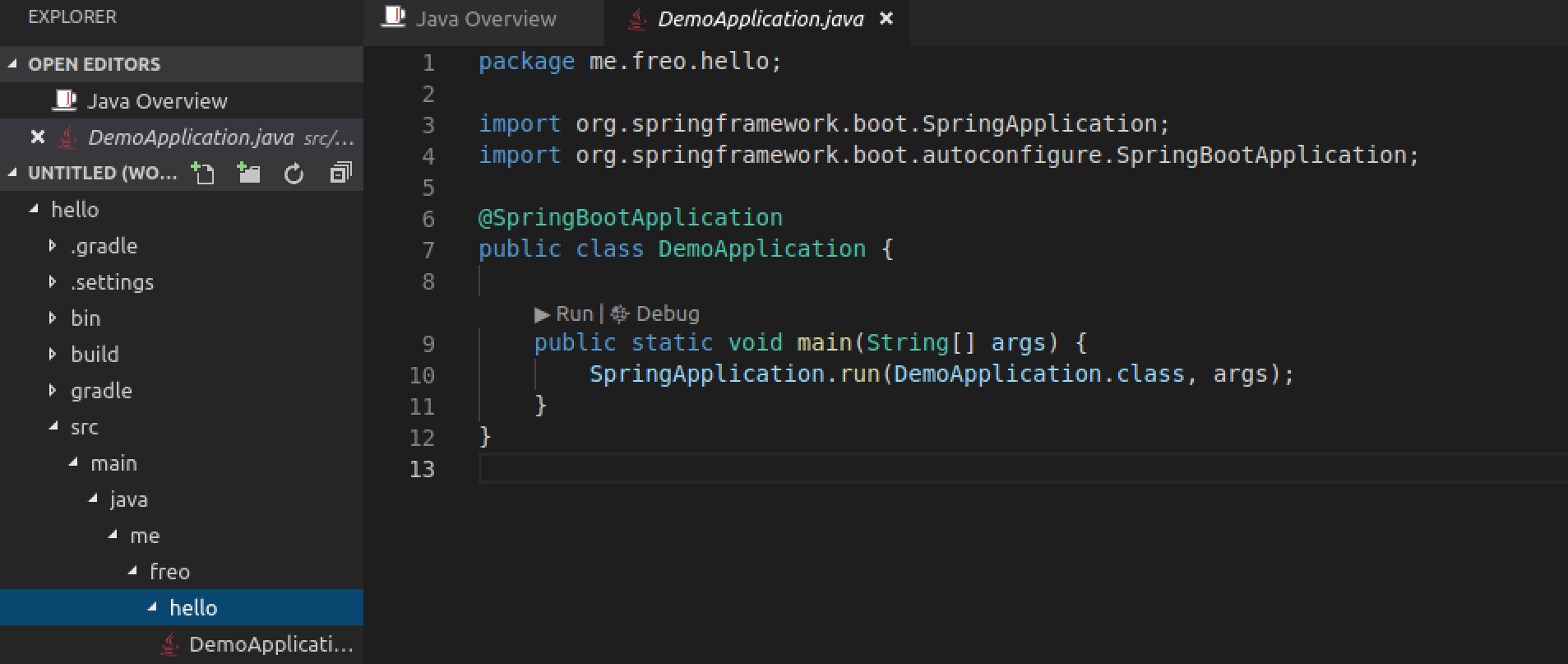
cd ~/ex3

1. Start Visual Studio Code:  
   code  
     
   You should see something like:  
   
2. Show the vscode “Command Palette” (Ctrl-Shift-P)
3. Type “spring” and you should some options appear:  
   
4. Select **Spring Initalizr: Generate a Gradle Project**
5. Select **Java**
6. Type **me.freo** for the Group
7. Type **hello** for the artifact id
8. Choose **2.1.1** for the Spring Boot version
9. Type **Jersey** and then select Jersey (JAX-RS) for the dependency.
10. Then hit **Enter** to continue.
11. The window should be pointing to your ex3 directory and then click **Generate into this Folder.** If clicking doesn’t work, try hitting **Enter**
12. You should see a notification that the code has been generated. Click **Add to Workspace:** ****
13. On a command line:  
    cd ~/ex3/hello  
    tree

You should see:  


1. In the same place, type:  
   gradle build
2. This will build your Spring Boot application. However, so far there is no logic, so it will just start a web server that doesn’t do anything useful. Try it:  
   gradle bootRun  
   You should see something like:  
   
3. You can see a server is running on port 8080, but there are no resources deployed:  
     
   
4. Stop the server / gradle run by hitting Ctrl-C
5. In vscode, Open the Folder (Ctrl-K Ctrl-O, or dropdown menu) ~/ex3/hello
6. You can open the src/main/java/me/freo/hello/DemoApplication.java file.

You should see:



1. In the same Package (me.freo.hello), create a new Java file called **Resource.java**
2. Type the following Java. Hint if you leave out the imports, vscode will add them automatically as you do the rest.  
   This file is our actual “Restful Service”. It contains the logic to respond to HTTP requests based on the verb/path/content-type, etc

package me.freo.hello;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import org.springframework.stereotype.Component;

@Component

@Path("/hello")

public class Resource {

@GET

@Path("/")

public String sayHello() {

return "hello world";

}

}

1. We need one more Java file, called **HelloConfiguration.java** (also in me/freo/hello.
2. The contents are here. Once again, vscode can help you do all this. This is basically telling Spring Boot about your Resource class.

package me.freo.hello;

import javax.annotation.PostConstruct;

import javax.ws.rs.ApplicationPath;

import org.glassfish.jersey.server.ResourceConfig;

import org.springframework.context.annotation.Configuration;

@Configuration

@ApplicationPath("/")

public class HelloConfiguration extends ResourceConfig {

public HelloConfiguration() {

}

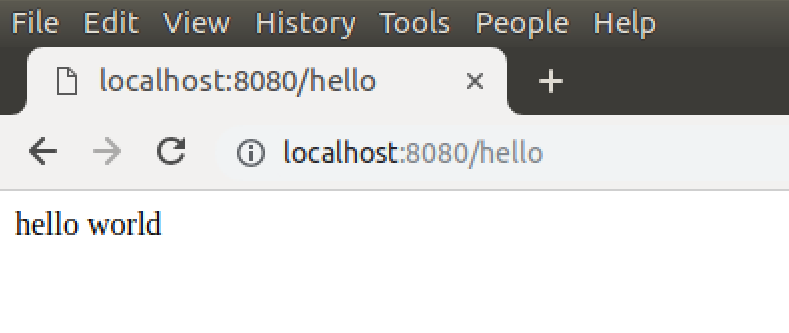
@PostConstruct

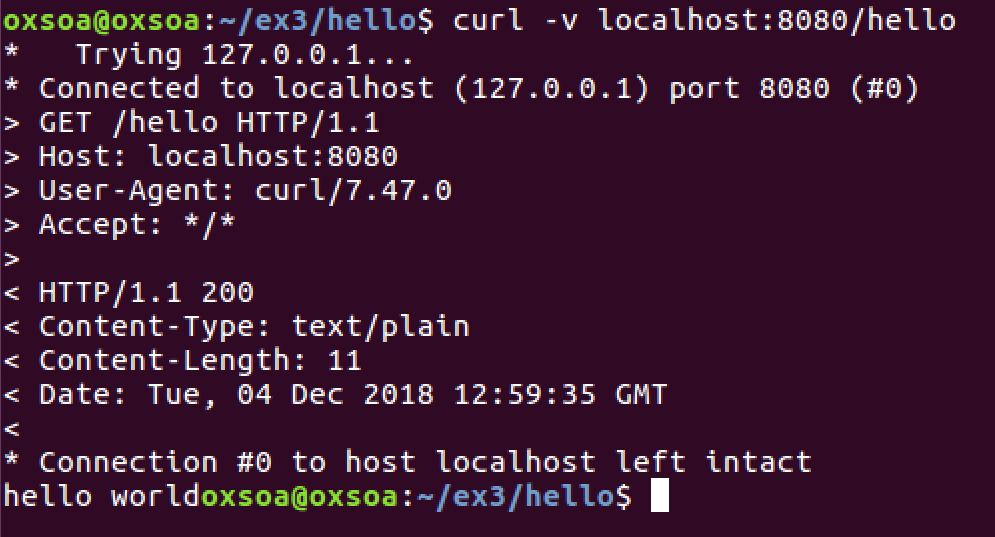
public void setUp() {

register(Resource.class);

}

}

1. Rebuild your gradle project and re-run:  
   gradle build  
   gradle bootRun
2. Now we can see that this has created a better server. Browse to <http://localhost:8080/hello>
3. Try “curl -v”:



1. We are nearly done, but let’s do one final improvement to the Resource. Change Resource.java so that it looks like this:  
   What we have done is to take full control of the HTTP response. Instead of just returning a string, we are ‘hand-crafting’ the Response object. We have also explicitly told JAX-RS to return an application/json media type.

package me.freo.hello;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import org.springframework.stereotype.Component;

@Component

@Path("/hello")

public class Resource {

@GET

@Produces(MediaType.APPLICATION\_JSON)

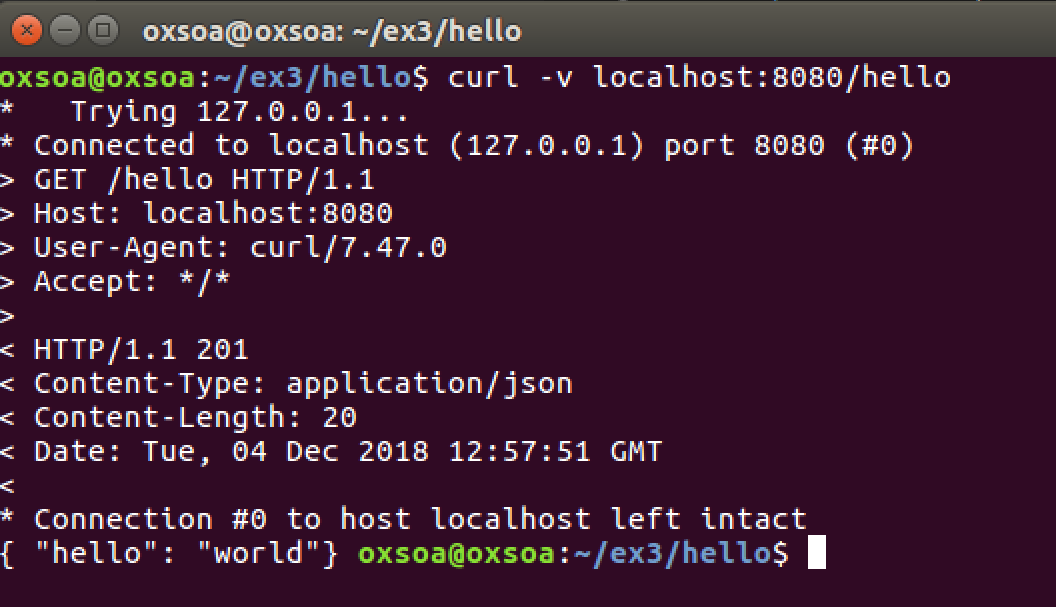
@Path("/")

public Response sayHello() {

return Response.status(201).entity("{ \"hello\": \"world\"} ").build();

}

}

1. Rebuild and rerun.
2. Now “curl -v” this resource. You should see:  
   
3. Notice the different return code, and the Content-Type. And of course the fact it is now JSON.
4. One last thing. So far we have only run this using gradle. That is really a “dev” time thing. What Spring Boot has done is to package this all up into a single executable JAR file. From the ~/ex3/hello directory type:

java -jar build/libs/hello-0.0.1-SNAPSHOT.jar  
  
Once again you should see the Spring logo etc and your service has started.

1. That’s all for now. We have created a simple Spring Boot project, including JAX-RS, and used gradle to build into a standalone JAR file.