**Exercise 7**

*Making our service into a single JAR microservice with a Redis backend*

**Prior Knowledge**

Exercise 6b

**Objectives**

Looking at deployment models for Services

NoSQL backends

**Software Requirements**

(see separate document for installation of these)

* Java Development Kit 8
* Gradle build system
* Jetty and Jersey
* Eclipse Neon
* curl
* Google Chrome/Chromium plus Chrome Advanced REST extension
* Redis

**Overview**

*We have built a reasonable RESTful service, but which can be exported as a WAR and run. It has no real backend as it is based on an in-memory singleton.*

*However, we would like to create a simpler deployment model based on a single JAR, and we would like a reasonable backend database to house the results.*

**Steps**

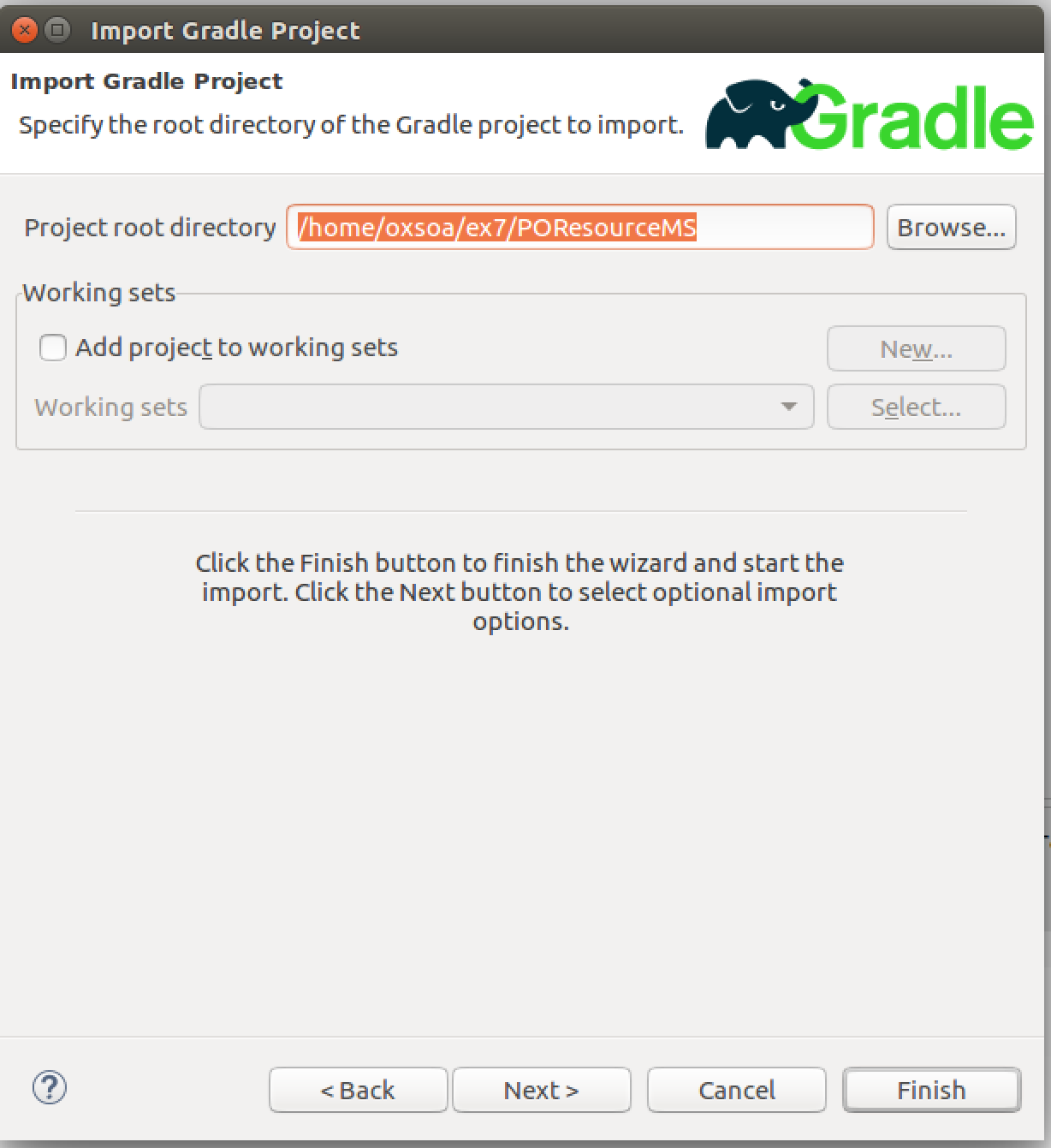
1. You can do this to your existing POResource project. However, because I’d like us to add redis support, I propose that we start from my completed version of Exercise 6.   
   You can checkout this version by doing the following command-line magic.  
     
   mkdir ~/ex7  
   cd ~/ex7  
   git clone <https://github.com/pzfreo/POResourceMS.git>  
   cd POResourceMS
2. We would like to change the build to support creating a single JAR file. We also need to create a new class that supports this.
3. First, lets add some new parts to the build.gradle file.   
   *Hint: You could load the project into Eclipse, but I propose that we get the gradle build improved first, so I would suggest using a Linux editor like Atom, gedit or nano. This means that when we load the project into Eclipse, it will be aware of the new plugins.*  
   In the plugins {} section, add the following additional line:  
   This is a plugin that mirrors the shadow plugin for maven. This packages all the code required including all dependencies into a single JAR file. The result is a JAR file that has no external dependencies.

plugins {

id 'com.github.johnrengelman.shadow' version '1.2.3'

}

1. Having defined the plugin (using the gradle plugin extension mechanism) we now need to use it:  
   Add the line to the main section of the gradle build (under the other similar lines)  
     
   apply plugin: 'com.github.johnrengelman.shadow'
2. We also need the *application* plugin, which works with shadow to build self-contained executable JARs (see <https://en.wikipedia.org/wiki/JAR_(file_format)#Executable_JAR_files> )  
     
   Add the line:  
   apply plugin: 'application'
3. We need to tell the application plugin the name of our “Main” executable class:  
     
   Add the following line underneath the “apply plugin” lines  
   mainClassName = 'freo.me.rest.Main'
4. Save your changes to the file.
5. Now import the project into Eclipse:  
   **File->Import Gradle->Gradle Project  
   Next>**Browse to the Project root directory: /home/oxsoa/ex7/POResourceMS  
   **Finish**



1. Now add a class called **Main** to the package src/main/java/freo.me.rest  
   The code is available here:  
   <https://github.com/pzfreo/ox-soa2/blob/master/code/before/ex7/Main.java>
2. Here is a code listing:

package freo.me.rest;

import java.net.URI;

import javax.ws.rs.core.UriBuilder;

import org.eclipse.jetty.server.Server;

import org.glassfish.jersey.jetty.JettyHttpContainerFactory;

import org.glassfish.jersey.server.ResourceConfig;

public class Main {

public static void main(String[] args) throws Exception {

URI baseUri = UriBuilder.fromUri("http://localhost/")  
 .port(8080).build();

// This is fairly self-explanatory.

// You can define the URL on which the server will listen.

ResourceConfig config =   
 new ResourceConfig(POResource.class);

// This is where we identify that the class   
// POResource is the JAX-RS

// Resource (aka Service) that we want to expose.

Server server = JettyHttpContainerFactory.  
 createServer(baseUri, config);

// Here is where we create the Jetty Server object.

try {

server.start();

// This initiates the startup of the server.

server.join();

// Wait until the server finishes initiation

} finally {

server.destroy();

// Obvious!

}

}

}

1. Now you can build this into a shadowJar. You can either use the gradle plugin, or the command line (from directory ~/ex7/POResourceMS)  
   gradle clean shadowJar
2. This creates a file:  
   build/libs/POResourceMS-all.jar
3. Try it out by executing:  
   java –jar build/libs/POResourceMS-all.jar
4. Test it. The URL is <http://localhost:8080/purchase>
5. Extension: Check out the other build targets:  
   gradle runShadow  
   gradle distShadowZip  
   gradle installShadowApp  
   *Hint: execute these with –info to see more of what is happening*
6. Adding redis