Dropwizard Workshop Instructions

This document contains the instructions to develop the workshop exercises.

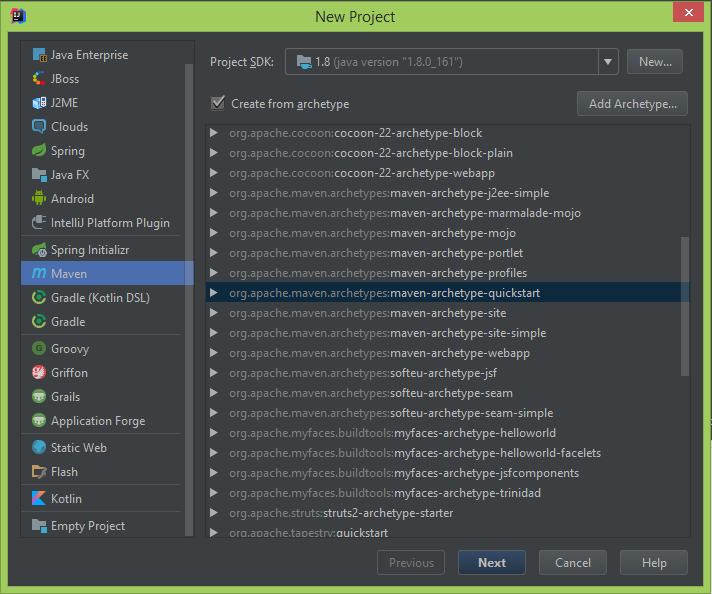
Note: You may copy and paste the codes in this document to your created classes. Alternatively, you can clone the project from [github](https://github.com/wandatinkers/dw-demos) if you want the complete version.

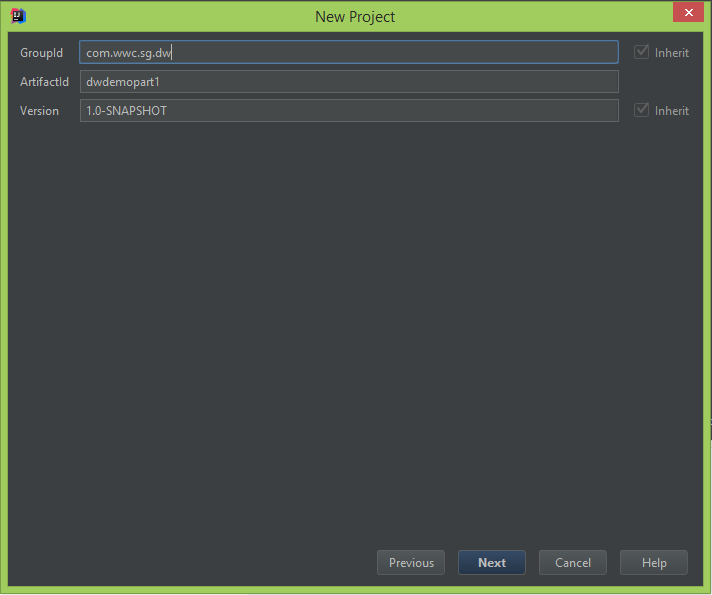
Part 1: Dropwizard Basic Components [dwdemopart1]

This exercise will go through the bare minimum needed to create and run your own dropwizard application.

1.1 Create new Maven project

Open IDE and then click Create New Project. Choose the maven-archetype-quickstart, then enter ***Group Id*** and ***Artifact Id***, and finally press ***Finish***.





## 

## 1.2 Update the pom.xml

Now update the pom.xml file to include the dropwizard-core maven dependency. We will also update the Maven project to use Java version 1.8. After updating the pom.xml file, update the Maven Project(Right Click on pom.xml > Maven > Reimport)

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.wwc.sg.dw</groupId>  
 <artifactId>dw-demo-part1</artifactId>  
 <name>Dropwizard Demo - Basic components</name>  
 <version>1.0-SNAPSHOT</version>

<properties>  
 <dropwizard.version>1.2.2</dropwizard.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>io.dropwizard</groupId>  
 <artifactId>dropwizard-core</artifactId>  
 <version>${dropwizard.version}</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

## 1.3 Create Configuration class

A dropwizard application has one configuration class which specifies the environment specific parameters. This maps to the config yml file.   
This class extends the **io.dropwizard.Configuration** class.

package com.wwc.sg.dw;  
  
import io.dropwizard.Configuration;  
  
public class DWDemoAppConfiguration extends Configuration {  
  
 private String application;  
 private String environment;  
 private String version;  
 private String template;  
  
 public String getApplication() {  
 return application;  
 }  
  
 public void setApplication(String application) {  
 this.application = application;  
 }  
  
 public String getEnvironment() {  
 return environment;  
 }  
  
 public void setEnvironment(String environment) {  
 this.environment = environment;  
 }  
  
 public String getVersion() {  
 return version;  
 }  
  
 public void setVersion(String version) {  
 this.version = version;  
 }  
  
 public String getTemplate() {  
 return template;  
 }  
  
 public void setTemplate(String template) {  
 this.template = template;  
 }  
  
}

## 1.4 Create Resource class

Resources are the meat of a Dropwizard application. Resources exposes our rest endpoints.

package com.wwc.sg.dw.resources;  
  
import javax.ws.rs.GET;  
import javax.ws.rs.Path;  
import javax.ws.rs.PathParam;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.MediaType;  
import javax.ws.rs.core.Response;  
  
@Path("/")  
@Produces(MediaType.APPLICATION\_JSON)  
public class IndexResource {  
  
 private String name;  
 private String environment;  
 private String version;  
 private String template;  
  
 public IndexResource(String name, String environment, String version, String template){  
 this.name = name;  
 this.environment = environment;  
 this.version = version;  
 this.template = template;  
 }  
  
 @GET  
 public Response getVersion() {  
 StringBuffer sb = new StringBuffer();  
 sb.append("Application:" + name + "\n");  
 sb.append("Environment:" + environment + "\n");  
 sb.append("Version:" + version + "\n");  
 return Response.ok(sb.toString()).build();  
 }  
  
 @GET  
 @Path("hello/{name}")  
 public Response printGreeting(@PathParam("name") String name) {  
 final String greeting = String.format(template, name);  
 return Response.ok(greeting).build();  
 }  
}

## 1.5 Create Application class

This is the main entry point for any Dropwizard application. This class extends the ***io.dropwizard.Application*** class and have access to the Configuration and Environment objects. They prepare the runtime environment of the application.

package com.wwc.sg.dw;  
  
import com.wwc.sg.dw.resources.IndexResource;  
import io.dropwizard.Application;  
import io.dropwizard.configuration.ResourceConfigurationSourceProvider;  
import io.dropwizard.setup.Bootstrap;  
import io.dropwizard.setup.Environment;  
  
public class DWDemoApp extends Application<DWDemoAppConfiguration> {  
  
 public static void main(String[] args) throws Exception {  
 new DWDemoApp().run(new String[] { "server", "DWDemoApp.yml"});  
 }  
  
 @Override  
 public void initialize(Bootstrap<DWDemoAppConfiguration> bootstrap) {  
 bootstrap.setConfigurationSourceProvider(  
 new ResourceConfigurationSourceProvider());  
 }  
  
 @Override  
 public void run(DWDemoAppConfiguration configuration, Environment environment) throws Exception {  
 final IndexResource resource = new IndexResource(  
 configuration.getApplication(),  
 configuration.getEnvironment(),  
 configuration.getVersion(),  
 configuration.getTemplate()  
 );  
 environment.jersey().register(resource);  
 }  
}

**1.6** Build your application

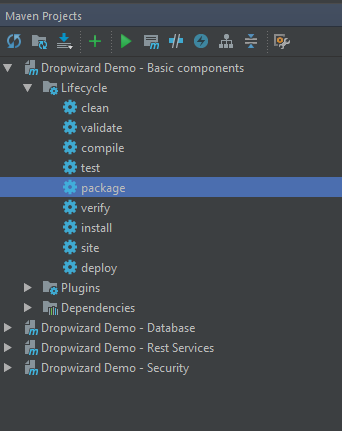
It is recommended to build and package your application as a single deployable artefact.

To do this, we need to use the maven shade plugin.

Add to pom.xml the maven shade plugin.

<build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.7.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 <!-- build fat jar -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-shade-plugin</artifactId>  
 <version>3.1.0</version>  
 <executions>  
 <execution>  
 <phase>package</phase>  
 <goals>  
 <goal>shade</goal>  
 </goals>  
 <configuration>  
 <transformers>  
 <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">  
 <mainClass>com.wwc.sg.dw.DWDemoApp</mainClass>  
 </transformer>  
 </transformers>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
  
</build>

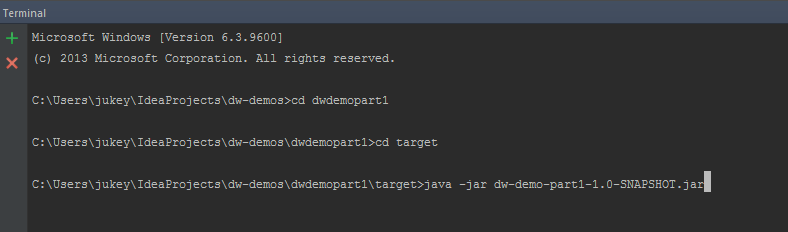
Build and package the maven project. On the Maven Projects window, right-click on ***Package***, then click ***Run Maven Build***. This will create the executable jar file in <project\_dir>/target folder.



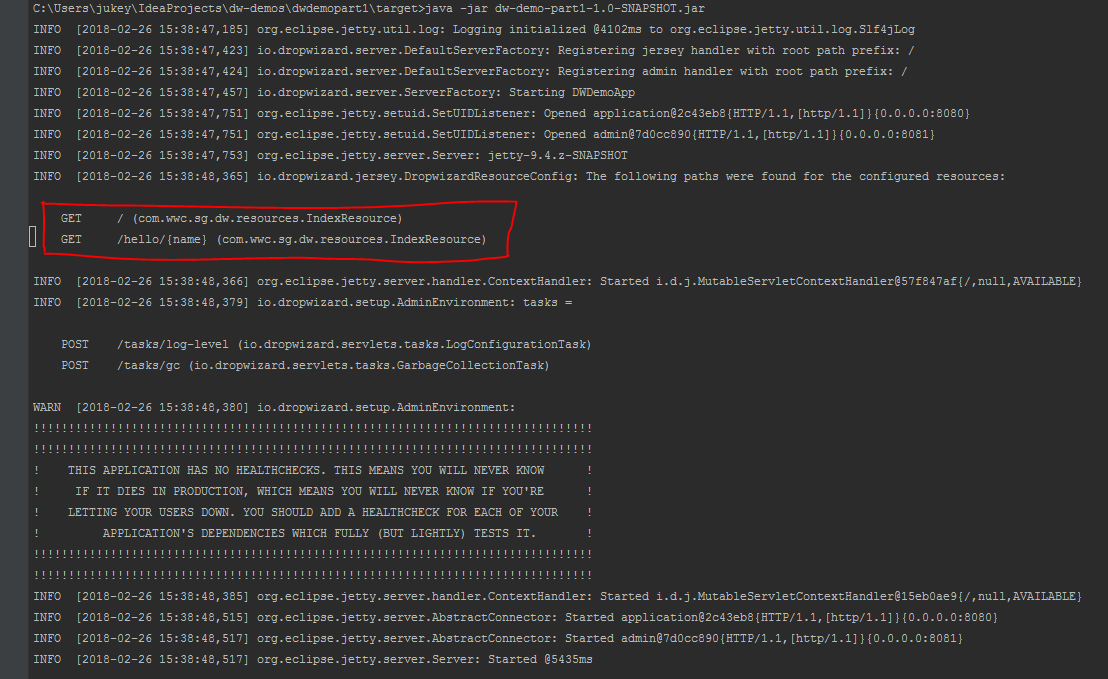
**1.7** Run your application

In Intellij, go to View -> Tool Windows -> Terminal.

Go to dwdemopart1/target, execute: ***java -jar dw-demo-part1-1.0-SNAPSHOT.jar***



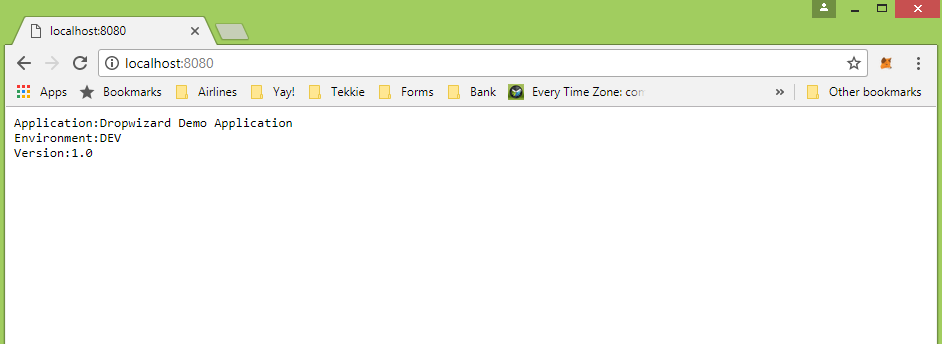
In the application startup log, the exposed endpoints will be listed.



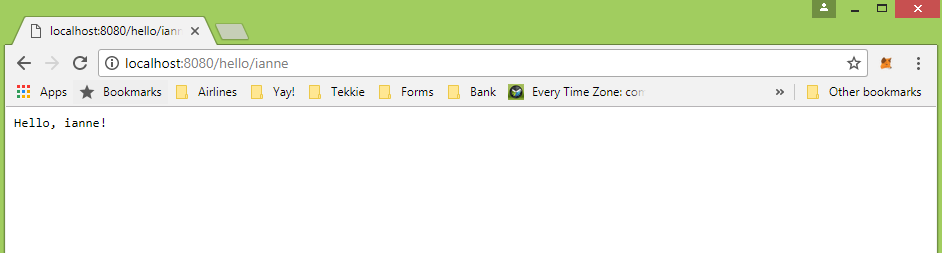
**1.8** Access rest endpoints

In chrome, access the following URI:

1. <http://localhost:8080/>



1. [http://localhost:8080/hello/<your\_name](http://localhost:8080/hello/%3cyour_name)> e.g. <http://localhost:8080/hello/Ianne>



Part 2: Dropwizard Rest Services [dwdemopart2]

In this exercise, we will extend the codes in Part 1 and develop a very simple rest service that will display a list of book titles and its author.

2.1 Create new representation

Representations are real-world objects modelled as POJOs. For this exercise, our model object is that of a Book object.

package com.wwc.sg.dw.representations;  
  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class Book {  
  
 private static AtomicInteger *ctr* = new AtomicInteger(0);  
  
 private int id;  
 private String title;  
 private String author;  
  
 public Book(String title, String author) {  
 this.id = *ctr*.incrementAndGet();  
 this.title = title;  
 this.author = author;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public String getAuthor() {  
 return author;  
 }  
  
 public void setAuthor(String author) {  
 this.author = author;  
 }  
}

2.2 Create a new resource

Resources provide access to manipulate our objects.

package com.wwc.sg.dw.resources;  
  
import com.wwc.sg.dw.representations.Book;  
  
import javax.ws.rs.GET;  
import javax.ws.rs.Path;  
import javax.ws.rs.PathParam;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.MediaType;  
import javax.ws.rs.core.Response;  
import java.util.ArrayList;  
import java.util.List;  
  
@Path("/catalog")  
@Produces(MediaType.*APPLICATION\_JSON*)  
public class BookCatalogResource {  
  
 private static List<Book> *books* = new ArrayList<>();  
 static{  
 *books*.add(new Book("'Autumn'", "Ali Smith"));  
 *books*.add(new Book("Exit West", "Mohsin Hamid"));  
 *books*.add(new Book("Pachinko", "Min Jin Lee"));  
 }  
  
 public BookCatalogResource() {  
 }  
  
 @GET  
 @Path("/book")  
 public Response getBooks() {  
 return Response.*ok*(*books*).build();  
 }  
  
 @GET  
 @Path("/book/{id}")  
 public Response getBooksById(@PathParam("id") int id) {  
 try {  
 Book book = *books*.get(id-1);  
 return Response.*ok*(book).build();  
 } catch (Exception e){  
 return Response.*ok*("No book found.").build();  
 }  
 }  
}

2.3 Register new resource

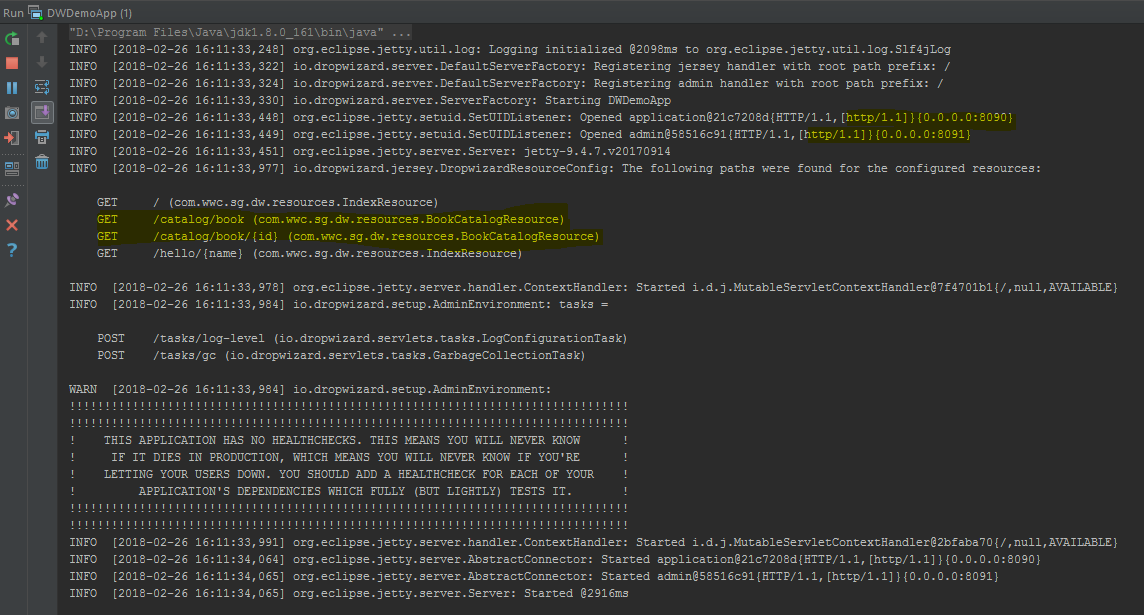
@Override  
public void run(DWDemoAppConfiguration configuration, Environment environment) throws Exception {  
 final IndexResource indexResource = new IndexResource(  
 configuration.getApplication(),  
 configuration.getEnvironment(),  
 configuration.getVersion(),  
 configuration.getTemplate()  
 );  
 environment.jersey().register(indexResource);  
 environment.jersey().register(new BookCatalogResource());  
}

2.4 Build and deploy in a new custom port of the embedded server

1. Add the configuration for the custom part in the config yml file.

**application:** Dropwizard Demo Application  
**environment:** DEV  
**version:** 2.0  
**template:** Hello, %s!  
  
**server:  
 applicationConnectors:** - **type:** http  
 **port:** 8090  
 **adminConnectors:** - **type:** http  
 **port:** 8091

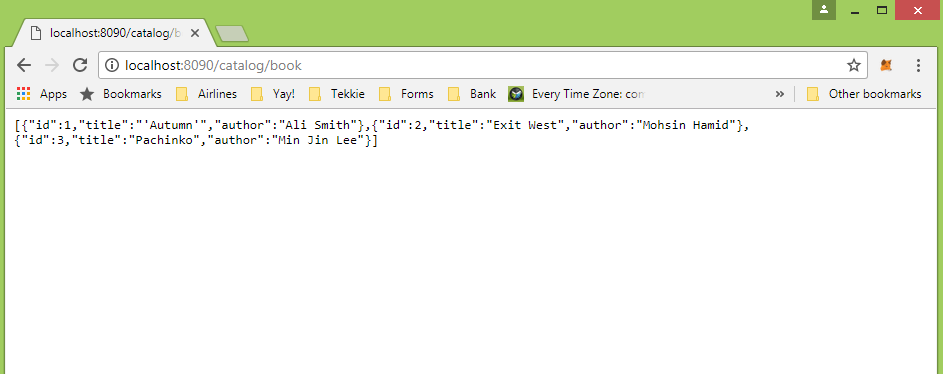
1. Build and run application (Refer to steps 1.6 and 1.7 in Part 1).



2.5 Access new rest endpoints

In chrome, access the following URI:

1. <http://localhost:8090/catalog/book>



1. <http://localhost:8090/catalog/book/1>

