**Challenge 2 & 3:** *Create REST API Using Dropwizard*

**Table of Contents**

[**Libraries included inside dropwizard**](#_bmmk7r4mew31) **2**

[**Setup dropwizard with maven**](#_2qjh87j3lqkp) **3**

[**Create REST Application Class**](#_kawfosy1dzsr) **5**

[**Build Resource Representations**](#_me4ccisjfpyv) **7**

[**Verify REST APIs**](#_shd02t75kvbu) **9**

[Build application uber jar file](#_g9id2ahouj5x) 9

[Start application in jetty server](#_mnro8wb9xg6b) 9

**Programming with Java Dropwizard**

Dropwizard is an open source Java framework for the rapid development of REST APIs. Dropwizard is kind of ecosystem which contains all the dependencies (such as Jersey, jackson or jetty) bundled into single package or can be added as separate module. If you are not using dropwizard, you will end up collecting all dependencies yourself and it often results into class loading issues due to version mismatch between various java libraries. Dropwizard solve this problem for you, and pulls together stable, mature libraries into a simple, light-weight package that lets you focus on getting things done.

## **Libraries included inside dropwizard**

Once you include dropwizard into your project, you get following libraries added to your classpath.

* [Jersey](https://howtodoinjava.com/jersey-jax-rs-tutorials/) – For building RESTful web applications.
* [Jetty](https://www.eclipse.org/jetty/) – Dropwizard uses the Jetty HTTP library to embed an HTTP server directly into your project.
* [Jackson](https://github.com/FasterXML/jackson) – For object to/from JSON conversion. It allows to export your domain models directly with JAXB annotations.
* [Guava](https://github.com/google/guava) – highly optimized immutable data structures to speedup development.
* [Logback](http://logback.qos.ch/) and [SLF4j](http://www.slf4j.org/) – for performant and flexible logging.
* [Hibernate Validator](http://hibernate.org/subprojects/validator.html) – an easy declarative framework for validating user input and generating helpful and i18n-friendly error messages.
* [Apache HTTPClient](https://hc.apache.org/httpcomponents-client-ga/index.html) – For both low- and high-level interaction with other web services.
* [JDBI](http://jdbi.org/) – the most straightforward way to use a relational database with Java.
* [Liquidbase](https://www.liquibase.org/) – to keep your database schema in check throughout your development and release cycle.
* [FreeMarker](https://freemarker.sourceforge.io/) – templating systems.
* [Mustache](https://mustache.github.io/) – simple templating systems for more user-facing applications.
* [Joda Time](https://www.joda.org/joda-time/) – very complete and sane library for handling dates and times.

## 

## **Setup dropwizard with maven**

Our project will be based on the maven-archetype-quickstart archetype. You can create the project either using command prompt or use eclipse to create a simple maven java project.

|  |
| --- |
| mvn archetype:generate -DgroupId=com.howtodoinjava.demo -DartifactId=DropWizardExample  -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false |

Generated project will have pom.xml file as well. Add dropwizard dependency there.

|  |
| --- |
| <properties>  <dropwizard.version>1.0.0</dropwizard.version>  </properties>  <dependencies>  <dependency>  <groupId>io.dropwizard</groupId>  <artifactId>dropwizard-core</artifactId>  <version>${dropwizard.version}</version>  </dependency>  </dependencies> |

This will download all jar files and add them into your classpath. To add build and package support to our project, we will use the maven-shade plugin, which will allow us to package our project completely, along with its dependencies, into a single standalone JAR file (Fat/Uber JAR) that can be distributed and executed as is.

Complete pom.xml file look like this.

<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.java.demo</groupId>

<artifactId>OrclDevopsChallenge</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>OrclDevopsChallenge</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<dropwizard.version>1.0.0</dropwizard.version>

<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.source>1.8</maven.compiler.source>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.dropwizard</groupId>

<artifactId>dropwizard-core</artifactId>

<version>${dropwizard.version}</version>

</dependency>

</dependencies>

<build>

<finalName>OrclDevopsChallenge-${version}</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>2.1</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>com.java.demo.OrclDevopsChallenge.App</mainClass>

</transformer>

<transformer implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer">

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

## **Create REST Application Class**

Application class is entry point for any dropwizard application. It needs to extend the io.dropwizard.Application class and implement the initialize(Bootstrap<Configuration>) and run(Configuration, Environment) methods. They prepare the runtime environment of the application.

To invoke the run method, you will need to have public static void main(String[] args) {} method, which will invoked by java -jar command when you run the application as jar file.

**App.class File:**

package com.java.demo.OrclDevopsChallenge;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.java.demo.OrclDevopsChallenge.controller.RestController;

import io.dropwizard.Application;

import io.dropwizard.Configuration;

import io.dropwizard.setup.Bootstrap;

import io.dropwizard.setup.Environment;

/\*\*

\* Hello world!

\*

\*/

public class App extends Application<Configuration>

{

private static final Logger LOGGER = LoggerFactory.getLogger(App.class);

@Override

public void initialize(Bootstrap<Configuration> b) {

}

@Override

public void run(Configuration c, Environment e) throws Exception {

LOGGER.info("Registering REST resources");

e.jersey().register(new RestController());

}

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

{

new App().run(args);

}

}

For executing the JAR file, we add the server argument to the command which starts the embedded HTTP Server (Jetty) to run our service.

java -jar target\OrclDevopsChallenge-0.0.1-SNAPSHOT.jar server

**Create REST Resource and APIs**

Now when you have added Application bootstrap class, you can now add REST resources containing REST APIs. In this example, I have created and employee management application – so it has APIs for create/update/delete employee records. This class will be responsible for handling HTTP requests and generating JSON responses.

As we have Jersey in our classpath, we will use it to build REST APIs.

**RestController.class File:**

package com.java.demo.OrclDevopsChallenge.controller;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import com.java.demo.OrclDevopsChallenge.service.FibonacciService;

@Path("/api")

public class RestController {

@GET

@Path("/helloWorld")

public Response simpleHelloWorld() {

return Response.ok("Hello World!!!").build();

}

@GET

@Path("/fibonacci/{range}")

public Response getFibonacciSeries(@PathParam("range") Integer range) {

if (range != null && !(range<1 || range>100))

return Response.ok(FibonacciService.getFibonacciSeries(range), MediaType.APPLICATION\_JSON).build();

else

return Response.status(Status.BAD\_REQUEST).entity("Invalid Range for Fibonacci Series").build();

}

}

## **Build Resource Representations**

Representation is what holds the data and serialized into JSON. It’s model for RESTful application. When using Jersey with Jackson, all you need to build a resource representation is – a simple POJO following java bean standards. Jackson constructs the JSON string recursively according to the getter methods of each class and their return type.

**FibonacciDto.class File:**

package com.java.demo.OrclDevopsChallenge.dto;

import java.util.List;

import javax.validation.constraints.NotNull;

import com.fasterxml.jackson.annotation.JsonProperty;

public class FibonacciDto {

@NotNull

@JsonProperty("member-count")

private Integer memberCount;

@JsonProperty("sequence")

private List<Integer> sequence;

@JsonProperty("total")

private Integer total;

public FibonacciDto(Integer memberCount, List<Integer> sequence,

Integer total) {

super();

this.memberCount = memberCount;

this.sequence = sequence;

this.total = total;

}

public Integer getMemberCount() {

return memberCount;

}

public void setMemberCount(Integer memberCount) {

this.memberCount = memberCount;

}

public List<Integer> getSequence() {

return sequence;

}

public void setSequence(List<Integer> sequence) {

this.sequence = sequence;

}

public Integer getTotal() {

return total;

}

public void setTotal(Integer total) {

this.total = total;

}

}

## **Verify REST APIs**

Now when we have created and added validations for REST APIs, let’s test this out.

#### **Build application uber jar file**

|  |
| --- |
| > mvn clean package |

#### **Start application in jetty server**

|  |
| --- |
| > java -jar target\OrclDevopsChallenge-0.0.1-SNAPSHOT.jar server |