**Usage Document & Assumptions**

1. Prerequisites

- Eclipse EE

- Apache Maven

- M2E - Maven plugin for Eclipse

- Tomcat

- MySQL server

- JDK

- JRE

2. Installation

- Assume Java is already installed on the computer correctly.

We can proceed to install Eclipse - IDE, Tomcat - web server, and MySQL server - database (xampp in my case).

3. Create a Maven project

- Assume the archetype org.glassfish.jersey.archetype, jersey.quickstart.webapp is available.

We select such archetype as our maven template, and then give it a groupid, artifactid of our own (in my case, org.endpoint.theone as groupid, and myendpoint as artifactid). Version we can leave it as default.

- Assume there is the javax.servlet.http.HttpServlet” was not found error, we can fix this by right clicking on the project > Properties > Project Facets > Runtimes > New, then select the Apache Tomcat version, then finish by applying the server.

Note that JDK 1.7 or above is required for Tomcat 8.

4. pom.xml Setup

- The file pom.xml should already be located at the root folder of the project (myendpoint) after you created the maven project.

- Uncomment the jersey-media-moxy dependency to enable JSON handling.

- Then add the mysql-connector-java dependency so we can use the JDBC driver.

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-moxy</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.39</version>

</dependency>

- Leave the rest in the pom.xml as default.

5. web.xml Setup

- The file web.xml can be found at the src > main > webapp > WEB-INF folder.

- Content in the web.xml should be autogenerated upon project creation. The following is the servlet information for the myendpoint webapp

<servlet>

<servlet-name>Jersey Web Application</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>org.endpoint.theone.myendpoint</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey Web Application</servlet-name>

<url-pattern>/myservice/\*</url-pattern>

</servlet-mapping>

- Note that param-value and url-pattern need to be mapped correctly

6. Class Creation

1. RestService.java (/src/main/java/org/endpoint/theone/myendpoint)

This class provides web service that handles POST requests that take and return JSON.

@Path("requests") specifies URI the requests are coming from

@POST specifies that it is a post request

@Consumes(MediaType.APPLICATION\_JSON) specifies that the request is taking JSON

@Produces(MediaType.APPLICATION\_JSON) specifies that the response will be JSON

1. ServiceModule.java (/src/main/java/org/endpoint/theone/myendpoint/module)

This class consists of the operations that process the request and generate the response. Operations include retrieving the number from database, and multiplying it with the number from the request.

1. Response.java (/src/main/java/org/endpoint/theone/myendpoint/model)

This class is the model of the response object as well as the data definition for JSON. It consists of constructors, setters and getters that can be use in other classes.

1. DBConnection.java (/src/main/java/org/endpoint/theone/myendpoint/dao)

This class manages the connection to the MySQL database with the com.mysql.jdbc.Driver. Our connection url: jdbc:mysql://localhost:3306/

1. DAO.java (/src/main/java/org/endpoint/theone/myendpoint/dao)

This class has a method that accesses the database via the DBConnection class, and returns the assigned number retrieved from the database

1. ErrorMsg.java (/src/main/java/org/endpoint/theone/myendpoint/model)

This class is the model of the errormsg object as well as the data definition for JSON returned when handling exceptions. It consists of constructors, setters and getters that can be use in other classes.

1. GenericExceptionMapper.java (/src/main/java/org/endpoint/theone/myendpoint/exception)

Assume there is invalid request such as server unavailable, and incorrect url. This class handles exceptions in general as throwable. A standard JSON response will be generated.

1. DataFormatErrorException.java (/src/main/java/org/endpoint/theone/myendpoint/exception)

Assume that the JSON in the request is empty, or not an integer, or the description is incorrect. The exception in this class will be thrown to handle the above.

1. DataFormatErrorExceptionMapper.java (/src/main/java/org/endpoint/theone/myendpoint/exception)

This class handles the dataformaterrorexceptions. JSON response specifying the error, as well as the expected input format will be generated.

1. DatabaseErrorException.java (/src/main/java/org/endpoint/theone/myendpoint/exception)

Assume that the MySQL server could be down or unavailable, or there is sql error. The exception in this class will be thrown to handle the database exceptions.

1. DatabaseErrorExceptionMapper.java (/src/main/java/org/endpoint/theone/myendpoint/exception)

This class handles the databaseerrorexceptions. JSON response specifying database error will be generated.

7. Test with Postman – a Google Chrome extension

- Go to the Google web store

- Search for Postman

- Install and Launch

- Type <http://localhost:8080/myendpoint/myservice/requests/>

- Select POST from the request type dropdown

- Click Body > raw

- Select JSON(application/json) from the raw type dropdown

- Type in the request in JSON format

- Click Send

- Scroll down and see the JSON response

Case 1: Invalid URL

Request: http://localhost:8080/myendpoint/myservice/reqqqqq

Response: {"errorMessage":"Invalid Request"}

Case 2: Database problem

Response:

{"errorMessage": "Unable to connect to the database."}

Case 3: JSON in POST Request is not an integer

Request: {“reqValue”: 1.5}

Response: {

"errorMessage": "Incorrect JSON format. reqValue must be an integer.",

"jsonInputExample": "{reqValue: 4}",

"jsonInputFormat": "{reqValue: <integer>}"

}