Hibernate / JAX-RS (Jersey) / JDBC

This document is a high level recipe for creating a RESTful API using JAX-RS(Jersey), Hibernate, and JDBC.
A little Google'ing, the APIs, and this document should get you started. If you require a more visual approach, then try the videos URL that I provided below. (each is about 10-20 minutes long)

JAX-RS is a specification for building RESTful web services in Java.
Jersey is the implementation of the JAX-RS specification.
Hibernate is an Object Relational Mapper (ORM) to JDBC connected databases and Java Objects.

- **Java SE 1.7+** Java core.
- **Java EE 1.7+** Servlet, persistence, security, validation, json/xml, etc…
- **Jersey 2.22.1** JAX-RS 2.0 implementation.
- **Gson 2.5+** A better Json object support than JavaEE version. (IMHO)
- **Apache Tomcat/7+** Servlet server. Alternatives are Glassfish and JBoss.
- **Apache Maven 3.0.5+** Dependencies and build.
- **Hibernate 4.3.11.Final** Object Relational Mapping. **ORM**
- **JDBC connectors** **sqljdbc4** (Microsoft) **mysql-connector-java** (MySql)
- **Netbeans 8.x+** Some IDE of your choice.

References

Videos

https://www.youtube.com/results?search_query=javabrains+jaxrs

Roadmaps

http://hibernate.org/orm/roadmap/
http://hibernate.org/orm/documentation/getting-started/
http://docs.oracle.com/javaee/7/index.html

APIs

https://jersey.java.net/apidocs/2.22.1/jersey/index.html

https://docs.jboss.org/hibernate/orm/4.3/javadocs/

https://docs.oracle.com/javaee/7/api/toc.htm
- https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html
- https://docs.oracle.com/javaee/7/api/javax/ws/rs/package-summary.html
- https://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html

https://docs.oracle.com/javase/7/docs/api/

http://static.javadoc.io/com.google.code.gson/gson/2.5/com/google/gson/package-summary.html

Developer Guides

https://jersey.java.net/documentation/2.22.1/index.html
http://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html_single/
- http://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html_single/#querycriteria-creating
- http://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html_single/#d5e3699

Initial Setup

Use Maven 3.0+ to create a jersey quickstart app. You can modify the groupId, artifactId, and the package.

```
mvn archetype:generate \
 -DarchetypeArtifactId=jersey-quickstart-webapp \
 -DarchetypeGroupId=org.glassfish.jersey.archetypes \
 -DarchetypeVersion=2.22.1 \
 -DinteractiveMode=false \
 -DgroupId=org.me \
 -DartifactId=MyHibernate \
 -Dpackage=org.me.rest
```

Modify pom.xml

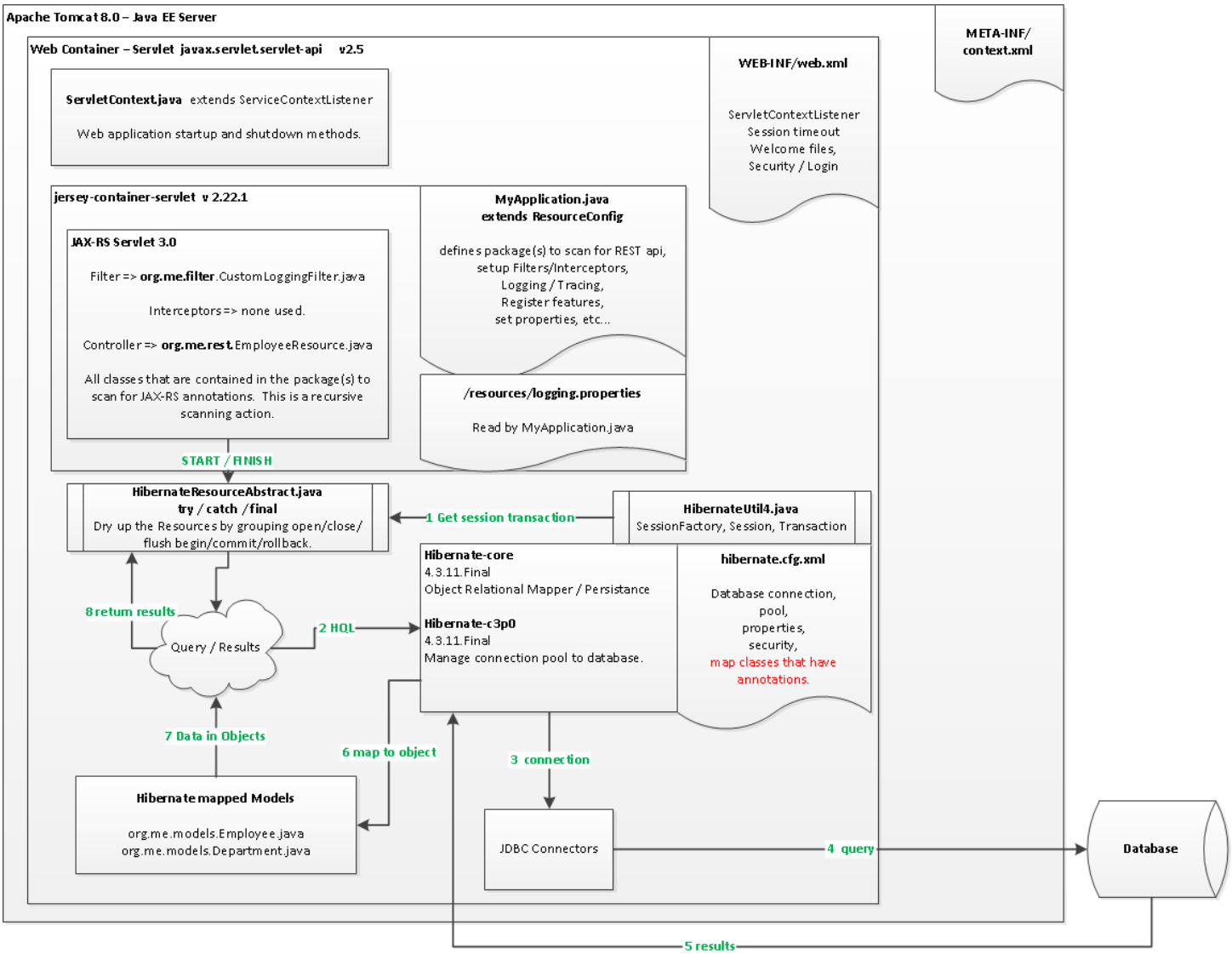
The items in bold have been modified or added from the original pom file.

```
<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/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.fjt</groupId>
    <artifactId>MyHibernate</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>MyHibernate</name>

    <build>
        <finalName>MyHibernate</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                    <compilerArgs>
                        <arg>-Xlint:all</arg>
                        <arg>-Xlint:unchecked</arg>
                    </compilerArgs>
                    <showWarnings>true</showWarnings>
                    <showDeprecation>true</showDeprecation>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.12.4</version>
                <configuration>
                    <skipTests>true</skipTests>
                </configuration>
            </plugin>

        </plugins>
    </build>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.glassfish.jersey</groupId>
                <artifactId>jersey-bom</artifactId>
                <version>${jersey.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
```

```
        </dependencies>
    </dependencyManagement>

    <dependencies>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- JERSEY -->
        <dependency>
            <groupId>org.glassfish.jersey.containers</groupId>
            <artifactId>jersey-container-servlet</artifactId>
        </dependency>

        <dependency>
            <groupId>org.glassfish.jersey.core</groupId>
            <artifactId>jersey-client</artifactId>
        </dependency>

        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-multipart</artifactId>
        </dependency>

        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.5</version>
        </dependency>

        <!-- DATABASE -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>4.3.11.Final</version>
        </dependency>

        <!-- This is for connection pool -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-c3p0</artifactId>
            <version>4.3.11.Final</version>
        </dependency>

        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>sqljdbc4</artifactId>
            <version>4.0</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.38</version>
        </dependency>

    </dependencies>
    <properties>
        <jersey.version>2.22.1</jersey.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

A sample tree of location of files. **Just keep files tidy and in grouped packages.**

```
*** Gson is Google's java class library that deals with JSON.

|-- pom.xml                                                 MAVEN dependencies
`-- src
    |-- main
    |   |-- java
    |   |   `-- nve
    |   |       `-- pesoft
    |   |           |-- HibernateUtil4.java                 SessionFactory, Session, and Transaction.
    |   |           |-- JsonResponse.java                   API response object.
    |   |           |-- MyApplication.java                  Jersey application config
    |   |           |-- ServletContext.java                 Kill SessionFactory on shutdown
    |   |           |-- enums
    |   |           |   |-- PropertyRelation.java           Enums for queryparams
    |   |           |   `-- QuerySource.java                Enums HIBERNATE/CUSTOM_SQL
    |   |           |-- exceptions
    |   |           |   |-- DateUtilsException.java
    |   |           |   |-- JsonUtilsException.java
    |   |           |   `-- SysUtilsException.java
    |   |           |-- interfaces
    |   |           |   `-- ExcludeGson.java                @ExcludeGson
    |   |           |-- models
    |   |           |   |-- AbstractModelHelpers.java       Each model extends this class.
    |   |           |   |-- Recipe.java                     Annotated models to tables mapping in database.
    |   |           |   |-- RecipeBucket.java
    |   |           |   |-- RecipeBucketCompositePK.java    Composite Primary key
    |   |           |   |-- RecipeRule.java
    |   |           |   |-- RecipeRuleSet.java
    |   |           |   |-- RuleSet.java
    |   |           |   `-- gson
    |   |           |       |-- AnnotationExclusionStrategy.java    @ExcludeGson glue
    |   |           |       |-- CollectionAdapter.java              Deserialize List,Map,etc.
    |   |           |       |-- GsonByteArrayAdapter.java           Not working yet.
    |   |           |       |-- GsonUTCDateAdapter.java             Tell Gson how to handle Date
    |   |           |       |-- HibernateProxyTypeAdapter.java      Allow Hibernate objects to be deserialized.
    |   |           |       |-- RecipeBucketDeserializer.java       All standard deserializers.
    |   |           |       |-- RecipeDeserializer.java
    |   |           |       |-- RecipeRuleDeserializer.java
    |   |           |       |-- RecipeRuleSetDeserializer.java
    |   |           |       `-- RuleSetDeserializer.java
    |   |           |-- restapi
    |   |           |   |-- HibernateResourceAbstract.java      Each Resource extends this class.
    |   |           |   |-- RecipeBucketResource.java           Resources are controllers/routing.
    |   |           |   |-- RecipeResource.java
    |   |           |   |-- RecipeRuleResource.java
    |   |           |   |-- RecipeRuleSetResource.java
    |   |           |   |-- RuleSetResource.java
    |   |           |   |-- SpecialResource.java
    |   |           |   `-- providers
    |   |           |       |-- BodyReaderString.java           String deserialization.
    |   |           |       |-- CustomLoggingFilter.java        Logging plus CORS.
    |   |           |       |-- CustomMessageBodyReader.java    Json to Object Deserializer.
    |   |           |       `-- GenericExceptionMapping.java    Return Json response on any Exception.
    |   |           |
    |   |           |-- utils                                   Basic helper utilities.
    |   |           |   |-- DateUtils.java
    |   |           |   |-- JsonUtils.java
    |   |           |   |-- RegExp.java
    |   |           |   `-- SysUtils.java
    |   |-- resources
    |   |   |-- hibernate.cfg.xml                           Hibernate config
    |   |   `-- logging.properties                          Logging config
    |   `-- webapp
    |       |-- META-INF
    |       |   `-- context.xml                             Tomcat/Glassfish deploy
    |       |-- WEB-INF
    |       |   `-- web.xml                                 Register Servlet listener.
```



src/main/webapp/WEB-INF/web.xml

Setup a listener for startup/shutdown, configure the session timeout, and set a welcome file.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <listener>
        <listener-class>org.me.ServletContext</listener-class>
    </listener>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <welcome-file-list>
        <welcome-file>/index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

ServletContext.java

I am placing a HibernateUtils4.shutdown() here so that SessionFactory is closed when we reset or stop Tomcat server.

```
package org.me;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;


public class ServletContext implements ServletContextListener {

    public ServletContext() {
    }

    @Override
    public void contextInitialized(ServletContextEvent contextEvent) {
        System.out.println("\n*** Context Created\n");
        TimeZone.setDefault(TimeZone.getTimeZone("Etc/UTC"));
    }

    @Override
    public void contextDestroyed(ServletContextEvent contextEvent) {
        System.out.println("\n*** Context Destroyed\n");
        HibernateUtil4.shutdown();  // this closes SessionFactory
    }
}
```

src/main/webapp/META-INF/context.xml

This is where we setup a web context on deploy. http://somehost:8085/MyHibernate/....

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/MyHibernate"/>
```

src/main/resources/logging.properties

Basic Java logging.

```
# http://tutorials.jenkov.com/java-logging/configuration.html
handlers=java.util.logging.ConsoleHandler

# Default global logging level.
# ALL SEVERE WARNING INFO CONFIG FINE FINER FINEST
.level=INFO
java.util.logging.ConsoleHandler.level=FINEST
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
java.util.logging.SimpleFormatter.format=%4$-7s [%3$s] %5$s%6$s
#java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneLineFormatter
org.glassfish.jersey.level=INFO
org.glassfish.jersey.tracing.level=INFO
```

MyApplication.java

Read in logging.properties, configure API, set up CustomFilter.

```
package nve.pesoft;


import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.LogManager;
import java.util.logging.Logger;
import javax.ws.rs.ApplicationPath;
import nve.pesoft.restapi.providers.CustomLoggingFilter;
import org.glassfish.jersey.media.multipart.MultiPartFeature;
```

```
import org.glassfish.jersey.server.ResourceConfig;
import org.glassfish.jersey.server.ServerProperties;
import org.glassfish.jersey.server.TracingConfig;
import org.glassfish.jersey.server.filter.HttpMethodOverrideFilter;
import org.glassfish.jersey.server.wadl.WadlFeature;

@ApplicationPath("/api")
public class MyApplication extends ResourceConfig {

    private static final String LOGGING_PROPERTIES_FILE = "logging.properties";
    private static final Logger LOGGER = Logger.getLogger(MyApplication.class.getName());

    public MyApplication() throws IOException {

        this.initLoggingProperties();

        // This is the package where our REST api resources (controllers) reside.
        // All classes in this package will be scanned for annotations.
        packages("nve.pesoft.restapi");

        // WADL output via GET    http://localhost:8085/MyHibernate/api/application.wadl
        register(WadlFeature.class);
        property(ServerProperties.WADL_FEATURE_DISABLE, false);  // set to TRUE to disable

        register(MultiPartFeature.class);

        // MVC                    https://jersey.java.net/documentation/2.22.1/mvc.html
        //register(MvcFeature.class);

        // TEMPLATE
        //register(JspMvcFeature.class);
        //property(JspMvcFeature.TEMPLATE_BASE_PATH, "/templates");
        // https://dzone.com/articles/mustaches-world-java
        // http://mustache.github.io/mustache.5.html
        //register(MustacheMvcFeature.class);
        //property(MustacheMvcFeature.TEMPLATE_BASE_PATH, "/templates");

        //register(FreemarkerMvcFeature.class);
        //property(FreemarkerMvcFeature.TEMPLATE_BASE_PATH, "/templates");

        //register(CustomLoggingFilter.class);

        //
https://jersey.java.net/apidocs/2.22.1/jersey/org/glassfish/jersey/server/filter/HttpMethodOverrideFilter.html
        // if POST and X-HTTP-Method-Override is PUT,DELETE,or GET, then call appropriate method.
        HttpMethodOverrideFilter overrideFilter = new HttpMethodOverrideFilter();
        register(overrideFilter);

        // Tracing support.  https://blogs.oracle.com/sandoz/entry/tracing_in_jersey      X-Jersey-Tracing-
Accept   true
        property(ServerProperties.TRACING, TracingConfig.ON_DEMAND.name());
    }

    private void initLoggingProperties() throws IOException {
        ClassLoader classLoader = getClass().getClassLoader();
        File file = new File(classLoader.getResource(LOGGING_PROPERTIES_FILE).getFile());

        if (file.exists()) {
            try {
                System.out.println("\n\nSTARTING\n\n");
                System.setProperty("java.util.logging.config.file", file.getAbsolutePath());

                String prop1 = System.getProperty("java.util.logging.config.file");
                if (prop1 != null) {
                    LOGGER.log(Level.INFO, prop1);
                }

                LogManager logManager = LogManager.getLogManager();
                logManager.readConfiguration();
            } catch (IOException ex) {
                LOGGER.log(Level.SEVERE, ex.getMessage());
                throw ex;
            }
        }
    }

}
```

GeneralExceptionMapper.java

You can trap for specific Exceptions. I decided to write one to cover them all.

```
package nve.pesoft.restapi.providers;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.ext.ExceptionMapper;
import javax.ws.rs.ext.Provider;
import nve.pesoft.JsonResponse;
import nve.pesoft.utils.RegExp;

@Provider
public class GenericExceptionMapping implements ExceptionMapper<Throwable> {

    public GenericExceptionMapping() {
    }

    @Override
    public Response toResponse(Throwable ex) {
        JsonResponse jsonResponse = new JsonResponse();
        jsonResponse.setStatus(Response.Status.BAD_GATEWAY);
        jsonResponse.setStatusCode(Response.Status.BAD_GATEWAY.getStatusCode());
        jsonResponse.setMessage(ex.getMessage());
        jsonResponse.setCallerClass(ex.getClass().getName());
        if (ex.getCause() != null) {
            jsonResponse.setCause(ex.getCause().getMessage());
        }

        Set<String> noStackTrace = new TreeSet<>();
        noStackTrace.add("com.google.gson.JsonSyntaxException");
        noStackTrace.add("com.google.gson.JsonParseException");

        if (noStackTrace.contains(jsonResponse.getCallerClass()) == false) {
            List<String> stackTraceElementsList = new ArrayList<>();

            StackTraceElement elements[] = ex.getStackTrace();
            for (int ii = 0, num_elements = elements.length; ii < num_elements; ii++) {
                String stackTraceElement = String.format("%s.%s(%s:%d)",
                        elements[ii].getClassName(),
                        elements[ii].getMethodName(),
                        elements[ii].getFileName(),
                        elements[ii].getLineNumber());
                stackTraceElementsList.add(stackTraceElement);
            }
            jsonResponse.setStackTraceElements(stackTraceElementsList);
        } else if (jsonResponse.getCallerClass().equals("com.google.gson.JsonParseException")) {

            if(RegExp.isMatch(".*Unparseable date.*", jsonResponse.getMessage())) {
                jsonResponse.setMessage(jsonResponse.getMessage() + " EXPECTED format => yyyy-MM-dd HH:mm:ss.SSS
zzz");
            }
        } else if (jsonResponse.getCallerClass().equals("com.google.gson.JsonSyntaxException")) {

            if(RegExp.isMatch(".*MalformedJsonException.*", jsonResponse.getMessage())) {
                jsonResponse.setMessage(jsonResponse.getMessage() + " CHECK YOUR BRACES AND COMMAS.");
            }
        }

        Response response = Response.status(jsonResponse.getStatus())
                .type(MediaType.APPLICATION_JSON)
                .entity(jsonResponse.toString())
                .build();
        return response;
    }

}
```

** **Note:** ContainerRequestFilter fires before ReaderInterceptor and MessageBodyReader.
Then, on response
ContainerResponseFilter fires before WriterInterceptor and MessageBodyWriter.

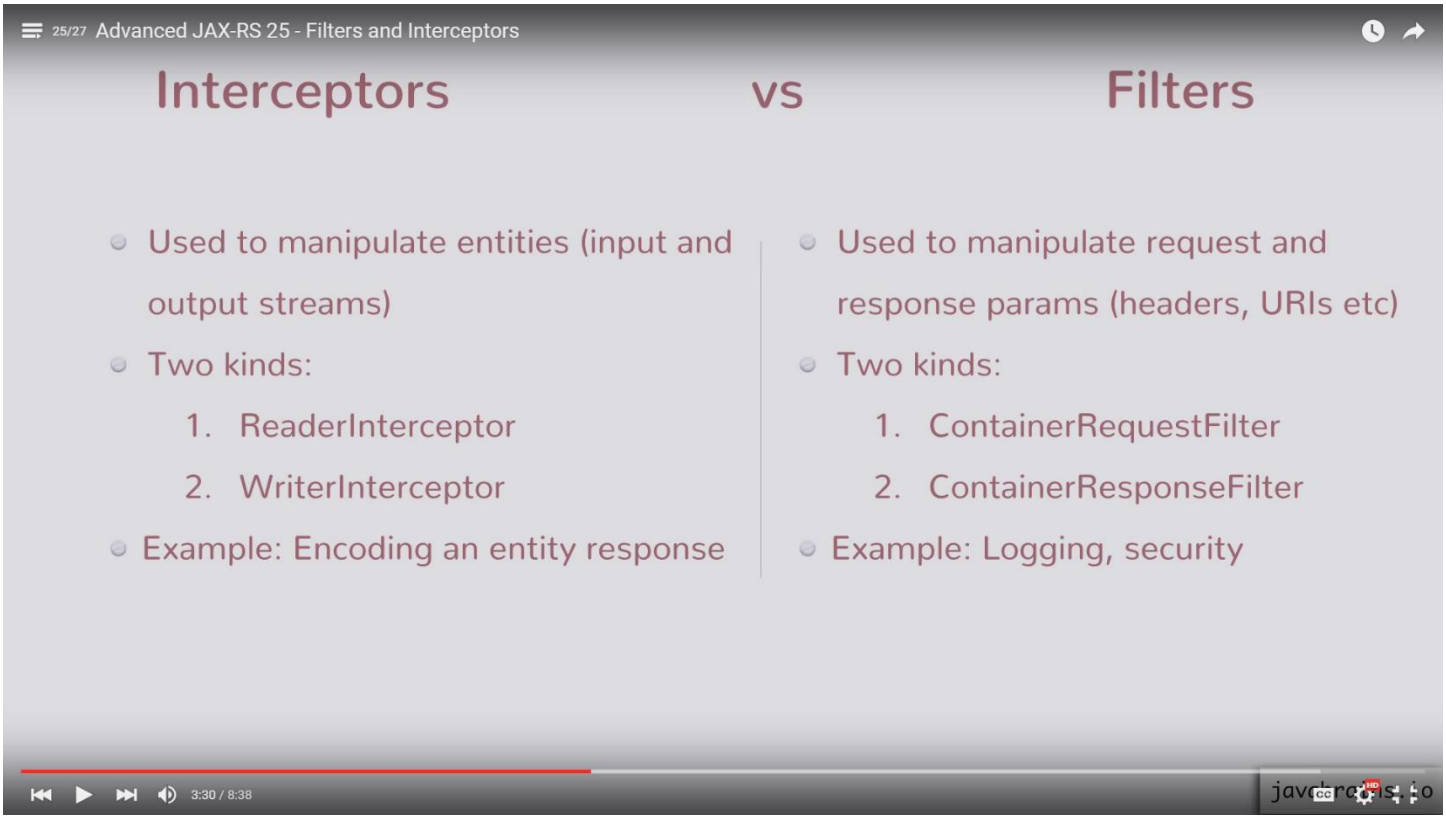
I did not use any Interceptors or MessageBodyWriter.

CustomLoggingFilter.java

```
package nve.pesoft.restapi.providers;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.security.Principal;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.ext.Provider;
import org.glassfish.jersey.message.internal.ReaderWriter;

// http://howtodoinjava.com/jersey/jersey-custom-logging-request-and-response-entities-using-filter/
@Provider
public class CustomLoggingFilter implements ContainerRequestFilter, ContainerResponseFilter {

    private static final Logger LOGGER = Logger.getLogger(CustomLoggingFilter.class.getName());

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
        //LOGGER.log(Level.INFO, "**REQUEST**");
        StringBuilder sb = new StringBuilder();
        Principal principal = requestContext.getSecurityContext().getUserPrincipal();
        String path = requestContext.getUriInfo().getPath();
        String method = requestContext.getMethod().toUpperCase();
        MultivaluedMap<String, String> headers = requestContext.getHeaders();

        sb.append("HTTP REQUEST: ");
```

```
        sb.append(" User: ").append(principal == null ? "unknown" : principal);
        sb.append(" | Path: ").append(path);
        sb.append(" | Method: ").append(method);
        sb.append(" | Header: ").append(headers);
        //sb.append(" | Entity: ").append(getEntityBody(requestContext));
        LOGGER.log(Level.INFO, sb.toString());
    }

    @Override
    public void filter(ContainerRequestContext requestContext, ContainerResponseContext responseContext) throws
IOException {

        // http://stackoverflow.com/questions/21820741/jersey-rest-response-in-angular-js

        // http://www.codingpedia.org/ama/how-to-add-cors-support-on-the-server-side-in-java-with-jersey/
        //      Access-Control-Allow-Origin: specifies the authorized domains to make cross-domain request (you
should include the domains of your REST clients or “*” if you want the resource public and available to everyone
– the latter is not an option if credentials are allowed during CORS requests)
        //    Access-Control-Expose-Headers: lets a server white list headers that browsers are allowed to
access
        //          Access-Control-Max-Age: indicates how long the results of a preflight request can be
cached.
        // Access-Control-Allow-Credentials: indicates if the server allows credentials during CORS requests
        //     Access-Control-Allow-Methods: indicates the methods allowed when accessing the resource
        //     Access-Control-Allow-Headers: used in response to a preflight request to indicate which HTTP
headers can be used when making the actual request.

        // https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
        // GET     The GET method requests a representation of the specified resource. Requests using GET should
only retrieve data and should have no other effect.
        // HEAD    The HEAD method asks for a response identical to that of a GET request, but without the
response body.
        // POST    The POST method requests that the server accept the entity enclosed in the request as a new
web resource identified by the URI.
        // PUT     The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI
refers to an already existing resource, it is modified.
        // DELETE  The DELETE method deletes the specified resource.
        // OPTIONS The OPTIONS method returns the HTTP methods that the server supports for the specified URL.

        // http://www.bennadel.com/blog/2568-preventing-cross-site-request-forgery-csrf-xsrf-with-angularjs-and-
coldfusion.htm

        // https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
        // Origin                - Initiates a request for cross-origin resource sharing (asks server for an
'Access-Control-Allow-Origin' response field).
        // X-Requested-With      - mainly used to identify Ajax requests. Most JavaScript frameworks send this
field with value of XMLHttpRequest.
        // X-HTTP-Method-Override - Requests a web application override the method specified in the request
(typically POST) with the method given in the header field (typically PUT or DELETE).

        // https://blogs.oracle.com/sandoz/entry/tracing_in_jersey (see MyApplication.java)
        // X-Jersey-Tracing-Accept - set to true if this is enabled => property(ServerProperties.TRACING,
TracingConfig.ON_DEMAND.name());

        responseContext.getHeaders().add("Access-Control-Allow-Origin", "*");

        responseContext.getHeaders().add("Access-Control-Allow-Methods",
                "GET,POST,DELETE,PUT,OPTIONS,HEAD");

        responseContext.getHeaders().add("Access-Control-Allow-Headers",
                "Content-Type,Access-Control-Allow-Headers,Authorization,X-Requested-With,Cache-
Control,Origin,Accept,X-Jersey-Tracing-Accept,X-HTTP-Method-Override");

        responseContext.getHeaders().add("X-Developers", "ftrujllo");

        // Build the response for the logger.
        StringBuilder sb = new StringBuilder();

        MultivaluedMap<String, String> headers = responseContext.getStringHeaders();
        Integer status = responseContext.getStatus();

        sb.append("HTTP RESPONSE: STATUS ").append(status).append(" ");
        sb.append(" | Header: ").append(headers);

        LOGGER.log(Level.INFO, sb.toString());
    }

    private String getEntityBody(ContainerRequestContext requestContext) {
```

```
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        InputStream in = requestContext.getEntityStream();

        final StringBuilder b = new StringBuilder();
        try {
            ReaderWriter.writeTo(in, out);

            byte[] requestEntity = out.toByteArray();
            if (requestEntity.length == 0) {
                b.append("");
            } else {
                b.append(new String(requestEntity)).append("\n");
            }
            requestContext.setEntityStream(new ByteArrayInputStream(requestEntity));

        } catch (IOException ex) {
            //Handle logging error
        }
        return b.toString();
    }

}
```



src/main/resources/hibernate.cfg.xml

All of these properties can be programmatically set in HibernateUtil4 class.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

    <session-factory>

        <property name ="hibernate.mapping.precedence property">class</property>

        <property name ="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>
        <property name
="hibernate.connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>
```

```
        <property name
="hibernate.connection.url">jdbc:sqlserver://SOMEHOST.my.org\DATABASE_INSTANCE_NAME:12345;databaseName=database_
name</property>
        <property name ="hibernate.connection.username">SOME_USERNAME</property>
        <property name ="hibernate.connection.password">*******</property>

<!--        <property name ="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name ="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name ="hibernate.connection.url">jdbc:mysql://somehost.my.org:3306/database_name</property>
        <property name ="hibernate.connection.username">someuser_user</property>
        <property name ="hibernate.connection.password">*****</property>-->

        <property name ="hibernate.connection.autocommit" >false</property>

        <!--
                The Hibernate session is bound to the current "thread".
                Therefore, when the transaction is committed, the session is
                closed also.
        -->
        <property name ="hibernate.current_session_context_class">thread</property>

        <property name ="hibernate.show_sql">true</property >
        <property name ="hibernate.format_sql">true</property>
        <property name ="hibernate.jdbc.batch_size">50</property>
        <property name ="hibernate.cache.use_second_level_cache">false</property>

        <!--
                A real connection pool. This is a MUST for production.

                https://developer.jboss.org/wiki/HowToConfigureTheC3P0ConnectionPool

                https://docs.jboss.org/hibernate/orm/4.3/devguide/en-US/html_single/#d5e154
        -->
        <property name="hibernate.c3p0.min_size">1</property>
        <property name="hibernate.c3p0.max_size">10</property>
        <property name="hibernate.c3p0.acquire_increment">1</property>
        <property name="hibernate.c3p0.idle_test_period">10</property>
        <property name="hibernate.c3p0.max_statements">10</property>
        <property name="hibernate.c3p0.timeout">20</property>


        <!--
                Now, map the Hibernate Annotated Objects
        -->
        <mapping class="nve.pesoft.models.Recipe"/>
        <mapping class="nve.pesoft.models.RecipeBucket"/>
        <mapping class="nve.pesoft.models.RecipeRuleSet"/>
        <mapping class="nve.pesoft.models.RuleSet"/>
        <mapping class="nve.pesoft.models.RecipeRule"/>
    </session-factory>

</hibernate-configuration>
```

HibernateUtil4.java

This utility class will initialize and return a SessionFactory.
A SessionFactory spawns Session(s) which turn spawn Transaction(s).

```
package nve.pesoft;

import javax.ws.rs.WebApplicationException;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

/**
 * This bootstrap works for Hibernate 4.3.11.Final
 *
 * http://www.codejava.net/frameworks/hibernate/building-hibernate-sessionfactory-from-service-registry
 *
 */
public class HibernateUtil4 {
```

```
    private static SessionFactory sessionFactory;
    private static boolean debug = false;

    public HibernateUtil4() {
    }

    public static void setDebug(boolean debug) {
        HibernateUtil4.debug = debug;
    }

    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                // loads configuration and mappings
                Configuration configuration = new Configuration().configure("hibernate.cfg.xml");

                //configuration.setProperty("hibernate.connection.password", "@ChangeMe123");

                if (debug) {
                    System.out.println("*** Hibernate Configuration loaded");
                }

                ServiceRegistry serviceRegistry
                        = new StandardServiceRegistryBuilder()
                        .applySettings(configuration.getProperties())
                        .build();

                if (debug) {
                    System.out.println("*** Hibernate serviceRegistry created");
                }

                sessionFactory = configuration.buildSessionFactory(serviceRegistry);

                if (debug) {
                    System.out.println("*** Hibernate sessionFactory created");
                }

            } catch (Exception ex) {
                StringBuilder sb = new StringBuilder();
                sb.append(ex.getMessage());
                System.out.println(sb.toString());
                throw new WebApplicationException(sb.toString());
            }
        }

        if (sessionFactory != null && debug) {
            System.out.println("\n*** Hibernate: sessionFactory returned.\n");
        }
        return sessionFactory;
    }

    public static void shutdown() {
        if (sessionFactory != null) {
            getSessionFactory().close();
            if (debug) {
                System.out.println("*** Hibernate shutdown!");
            }
            sessionFactory = null;
        }
    }
}
```

HibernateResourceAbstract.java

This class allows the grouping of common Hibernate objects that MUST be called the same way for each query.
Each Jersey Resource will extend this class to DRY up the same old repeated stuff.
I could have put the Response code in a MessageBodyWriter, but I decided against that. My reason was I did not know about MessageBodyWriter(s) enough at the time to implement one. 6, 1, ½ dozen, the other….

```
package nve.pesoft.restapi;

import java.text.ParseException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;
import nve.pesoft.HibernateUtil4;
import nve.pesoft.JsonResponse;
import nve.pesoft.ServletContext;
import nve.pesoft.enums.PropertyRelation;
import static nve.pesoft.enums.PropertyRelation.EQUAL;
import static nve.pesoft.enums.PropertyRelation.GREATER_THAN;
import static nve.pesoft.enums.PropertyRelation.GREATER_THAN_OR_EQUAL;
import static nve.pesoft.enums.PropertyRelation.LESS_THAN;
import static nve.pesoft.enums.PropertyRelation.LESS_THAN_OR_EQUAL;
import static nve.pesoft.enums.PropertyRelation.NOT_EQUAL;
import nve.pesoft.exceptions.DateUtilsException;
import nve.pesoft.utils.DateUtils;
import nve.pesoft.utils.JsonUtils;
import nve.pesoft.utils.RegExp;
import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.criterion.Disjunction;
import org.hibernate.criterion.Restrictions;

public abstract class HibernateResourceAbstract {

    private boolean debug;
    private Transaction tx = null;

    protected Session session = null;

    @Context
    protected ServletContext servletContext;
    @Context
    protected HttpServletRequest httpServletRequest;
    @Context
    protected UriInfo uriInfo;
    @Context
    protected HttpHeaders httpHeaders;

    public HibernateResourceAbstract() {
        this.debug = true; // set this to true to DEBUG ALL resources in application.
    }

    protected final void setDebug(boolean debug) {
        this.debug = debug;
    }

    public boolean isDebug() {
        return debug;
    }

    protected void beginTransaction(int timeout) {
        if (debug) {
            HibernateUtil4.setDebug(true);
        } else {
```

```
            HibernateUtil4.setDebug(false);
        }

        SessionFactory sessionFactory = HibernateUtil4.getSessionFactory();
        session = sessionFactory.getCurrentSession();

        if (session != null && session.isOpen()) {
            tx = session.getTransaction();
            tx.setTimeout(timeout);
            tx.begin();
        }
    }

    protected void commitTransaction() {
        System.out.println("\n SESSION " + session.toString() + " \n");
        if (session != null && session.isOpen()) {
            session.flush();
            session.clear();
        }
        if (tx != null && tx.isActive()) {
            tx.commit();
            if (debug) {
                System.out.println("\n*** COMMIT !!!\n");
            }
        }
    }

    protected String getCurrentMethod() {
        String method = httpServletRequest.getMethod().toUpperCase().trim();
        List<String> methodOverrideList = httpHeaders.getRequestHeader("X-HTTP-Method-Override");
        if (methodOverrideList != null && methodOverrideList.size() > 0) {
            method = methodOverrideList.get(0);
        }
        return method;
    }

    // http://racksburg.com/choosing-an-http-status-code/
    protected JsonResponse hibernateObjectToJsonResponse(
            Object hibernateObject) {
        String method = this.getCurrentMethod();
        String uri = httpServletRequest.getRequestURI();

        JsonResponse jsonResponse = new JsonResponse();

        // If Hibernate returns null, then there is no results for query.
        if (hibernateObject == null) {
            jsonResponse.setStatus(Response.Status.NO_CONTENT); // 204
            jsonResponse.setMessage("");
        } else // If object is a List then set appropriate Status and message.
         if (hibernateObject instanceof List<?>) {
                List myList = (List) hibernateObject;

                if (myList.size() < 1) {
                    jsonResponse.setStatus(Response.Status.NO_CONTENT); // 204
                    jsonResponse.setMessage("");
                } else {

                    jsonResponse.setStatus(Response.Status.OK); // 200
                    jsonResponse.setMessage(
                            JsonUtils.hibernateObjectToJsonPrettyNoNulls(
                                    this.session,
                                    myList));
                }
            } else {
                jsonResponse.setStatus(Response.Status.OK); // 200
                jsonResponse.setMessage(
                        JsonUtils.hibernateObjectToJsonPrettyNoNulls(
                                this.session,
                                hibernateObject));
            }

        if (method.equals("POST")) {
            if (jsonResponse.getStatus() == Response.Status.OK) {  // 200
                jsonResponse.setStatus(Response.Status.CREATED);   // 201
                // Jacob requested to send BACK the object after creation for all POSTS.
                jsonResponse.setMessage(JsonUtils.hibernateObjectToJsonPrettyNoNulls(this.session,
hibernateObject));
```

```
            }
        } else if (method.equals("PUT") || method.equals("DELETE")) {
            if (jsonResponse.getStatus() == Response.Status.NO_CONTENT) { // 204

                jsonResponse = this.getErrorJsonResponse(
                        "Unable to update/delete.",
                        Response.Status.BAD_REQUEST); // 400

            } else if (jsonResponse.getStatus() == Response.Status.OK) { // 200
                jsonResponse.setMessage("");
            }
        }

        return jsonResponse;
    }

    protected JsonResponse rollbackTransaction(Exception ex) {
        String method = httpServletRequest.getMethod().toUpperCase().trim();
        String uri = httpServletRequest.getRequestURI();

        JsonResponse jsonResponse = new JsonResponse();
        jsonResponse.setUri(uri);
        jsonResponse.setMethod(method);
        jsonResponse.setStatus(Response.Status.BAD_REQUEST); // 400
        jsonResponse.setStatusCode(Response.Status.BAD_REQUEST.getStatusCode());
        jsonResponse.setMessage(ex.getMessage());
        jsonResponse.setCallerClass(ex.getClass().getName());
        if (ex.getCause() != null) {
            jsonResponse.setCause(ex.getCause().getMessage());
        }

        if (jsonResponse.getCallerClass().equals("java.lang.NullPointerException")) {
            List<String> stackTraceElementsList = new ArrayList<>();

            StackTraceElement elements[] = ex.getStackTrace();
            for (int ii = 0, num_elements = elements.length; ii < num_elements; ii++) {
                String stackTraceElement = String.format("%s.%s(%s:%d)",
                        elements[ii].getClassName(),
                        elements[ii].getMethodName(),
                        elements[ii].getFileName(),
                        elements[ii].getLineNumber());
                stackTraceElementsList.add(stackTraceElement);
            }
            jsonResponse.setStackTraceElements(stackTraceElementsList);
        }

        String tmpJsonString = jsonResponse.toString();
        jsonResponse.setMessage(tmpJsonString);

        // If the Session throws an exception, the transaction must be
        // rolled back and the session discarded. The internal state of the
        // Session might not be consistent with the database after the
        // exception occurs.
        if (debug) {
            System.out.println("\n*** ROLLBACK ***");
        }
        if (tx != null && tx.isActive()) {
            tx.rollback();
        }
        return (jsonResponse);
    }

    protected void closeSession() {
        if (debug) {
            System.out.println("\n*** Calling closeSession()\n");
            if (session != null) {
                System.out.println("\n SESSION " + session.toString() + " \n");
            }
        }

//        The Hibernate session is bound to the current "thread".
//        Therefore, when the transaction is committed, the session is closed also.
//
//        The next block will not execute if you used sessionFactory.getCurrentSession();
        if (session != null && session.isOpen()) {
            session.close();
            if (debug) {
```

```
            System.out.println("\n*** CLOSE session\n");
        }
    }
}

protected JsonResponse getErrorJsonResponse(String message, Response.Status responseStatus) {
    String method = httpServletRequest.getMethod().toUpperCase().trim();
    String uri = httpServletRequest.getRequestURI();

    JsonResponse jsonResponse = new JsonResponse();
    jsonResponse.setMessage(message);
    jsonResponse.setStatus(responseStatus);
    jsonResponse.setMethod(method);
    jsonResponse.setUri(uri);
    jsonResponse.setStatusCode(responseStatus.getStatusCode());
    String str = JsonUtils.objectToJsonPrettyNoNulls(jsonResponse);
    jsonResponse.setMessage(str);

    return jsonResponse;
}

protected JsonResponse getStoredProcedureJsonResponse(int numRowsAffected, String storedProcName) {
    JsonResponse jsonResponse;
    String method = httpServletRequest.getMethod().toUpperCase().trim();

    if (method.equals("GET") && numRowsAffected == 0) { // GET index() has no rows
        jsonResponse = new JsonResponse();
        jsonResponse.setStatus(Response.Status.NO_CONTENT); // 204
        jsonResponse.setStatusCode(Response.Status.NO_CONTENT.getStatusCode());
        jsonResponse.setMessage("");
    } else if (method.equals("GET") && numRowsAffected > 0) { // GET index() is ok
        jsonResponse = new JsonResponse();
        jsonResponse.setStatus(Response.Status.OK); // 200
        jsonResponse.setStatusCode(Response.Status.OK.getStatusCode());
        jsonResponse.setMessage("");
    } else if (method.equals("POST") && numRowsAffected >= 1) { // POST create() is ok
        jsonResponse = new JsonResponse();
        jsonResponse.setStatus(Response.Status.CREATED); // 201
        jsonResponse.setStatusCode(Response.Status.CREATED.getStatusCode());
        jsonResponse.setMessage("");
    } else if (method.equals("PUT") && numRowsAffected >= 1) { // PUT update() is ok
        jsonResponse = new JsonResponse();
        jsonResponse.setStatus(Response.Status.OK); // 200
        jsonResponse.setStatusCode(Response.Status.OK.getStatusCode());
        jsonResponse.setMessage("");
    } else if (method.equals("DELETE") && numRowsAffected >= 1) { // DELETE delete() is ok
        jsonResponse = new JsonResponse();
        jsonResponse.setStatus(Response.Status.OK); // 200
        jsonResponse.setStatusCode(Response.Status.OK.getStatusCode());
        jsonResponse.setMessage("");
    } else {
        String msg = "Stored Procedure FAILED: " + storedProcName;
        jsonResponse = this.getErrorJsonResponse(msg, Response.Status.BAD_REQUEST); // 400
    }

    return jsonResponse;
}

@SuppressWarnings("UnusedAssignment")
protected void filterQuery(Criteria cr, String paramName) throws ParseException, DateUtilsException {
    List<String> paramList = this.uriInfo.getQueryParameters().get(paramName);

    if (paramList != null && paramList.size() > 0) {
        Disjunction or = Restrictions.disjunction();
        for (String tmp : paramList) {
            String[] items = tmp.split(",");

            for (String item : items) {
                PropertyRelation propertyRelation = EQUAL;

                if (RegExp.isMatch("^([engl][qett])\\((.+)\\)$", item.trim())) {
                    String op = RegExp.getSubExps().get(1);
                    System.out.println("FOUND op " + op);
                    switch (op) {
                        case "eq":
                            propertyRelation = EQUAL;
                            item = RegExp.getSubExps().get(2).trim();
```

```
            break;
        case "ne":
            propertyRelation = NOT_EQUAL;
            item = RegExp.getSubExps().get(2).trim();
            break;
        case "lt":
            propertyRelation = LESS_THAN;
            item = RegExp.getSubExps().get(2).trim();
            break;
        case "gt":
            propertyRelation = GREATER_THAN;
            item = RegExp.getSubExps().get(2).trim();
            break;
        case "le":
            propertyRelation = LESS_THAN_OR_EQUAL;
            item = RegExp.getSubExps().get(2).trim();
            break;
        case "ge":
            propertyRelation = GREATER_THAN_OR_EQUAL;
            item = RegExp.getSubExps().get(2).trim();
            break;
        default:
            propertyRelation = EQUAL;
            item = RegExp.getSubExps().get(2).trim();
    }
}

// ========================================================

if (RegExp.isMatch("^[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}.*", item)) {
    Date dateObj = DateUtils.parse(item);
    switch (propertyRelation) {
        case EQUAL:
            or.add(Restrictions.eq(paramName, dateObj));
            break;
        case NOT_EQUAL:
            or.add(Restrictions.ne(paramName, dateObj));
            break;
        case LESS_THAN:
            or.add(Restrictions.lt(paramName, dateObj));
            break;
        case GREATER_THAN:
            or.add(Restrictions.gt(paramName, dateObj));
            break;
        case LESS_THAN_OR_EQUAL:
            or.add(Restrictions.le(paramName, dateObj));
            break;
        case GREATER_THAN_OR_EQUAL:
            or.add(Restrictions.ge(paramName, dateObj));
            break;
        default:
            break;
    }
} else if (RegExp.isMatch("^[0-9]+$", item)) {
    Integer integerObj = Integer.parseInt(item);
    switch (propertyRelation) {
        case EQUAL:
            or.add(Restrictions.eq(paramName, integerObj));
            break;
        case NOT_EQUAL:
            or.add(Restrictions.ne(paramName, integerObj));
            break;
        case LESS_THAN:
            or.add(Restrictions.lt(paramName, integerObj));
            break;
        case GREATER_THAN:
            or.add(Restrictions.gt(paramName, integerObj));
            break;
        case LESS_THAN_OR_EQUAL:
            or.add(Restrictions.le(paramName, integerObj));
            break;
        case GREATER_THAN_OR_EQUAL:
            or.add(Restrictions.ge(paramName, integerObj));
            break;
        default:
            break;
    }
} else {
```

```
                        switch (propertyRelation) {
                            case EQUAL:
                                or.add(Restrictions.eq(paramName, item));
                                break;
                            case NOT_EQUAL:
                                or.add(Restrictions.ne(paramName, item));
                                break;
                            case LESS_THAN:
                                or.add(Restrictions.lt(paramName, item));
                                break;
                            case GREATER_THAN:
                                or.add(Restrictions.gt(paramName, item));
                                break;
                            case LESS_THAN_OR_EQUAL:
                                or.add(Restrictions.le(paramName, item));
                                break;
                            case GREATER_THAN_OR_EQUAL:
                                or.add(Restrictions.ge(paramName, item));
                                break;
                            default:
                                break;
                        }
                    }
                }
            }
            cr.add(or);
        }
    }

}
```

Hibernate JPA Annotations - Contents:

Annotation	Package Detail/Import statement
@Entity	import javax.persistence.Entity;
@Table	import javax.persistence.Table;
@Column	import javax.persistence.Column;
@Id	import javax.persistence.Id;
@GeneratedValue	import javax.persistence.GeneratedValue;
@Version	import javax.persistence.Version;
@OrderBy	import javax.persistence.OrderBy;
@Transient	import javax.persistence.Transient;
@Lob	import javax.persistence.Lob;
Hibernate Association Mapping Annotations	
@OneToOne	import javax.persistence.OneToOne;
@ManyToOne	import javax.persistence.ManyToOne;
@OneToMany	import javax.persistence.OneToMany;
@ManyToMany	import javax.persistence.ManyToMany;
@PrimaryKeyJoinColumn	import javax.persistence.PrimaryKeyJoinColumn;
@JoinColumn	import javax.persistence.JoinColumn;
@JoinTable	import javax.persistence.JoinTable;
@MapsId	import javax.persistence.MapsId;
Hibernate Inheritance Mapping Annotations	
@Inheritance	import javax.persistence.Inheritance;
@DiscriminatorColumn	import javax.persistence.DiscriminatorColumn;
@DiscriminatorValue	import javax.persistence.DiscriminatorValue;

Example of a JPA annotated model

https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html

```
package nve.pesoft.models;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.xml.bind.annotation.XmlRootElement;
import nve.pesoft.utils.DateUtils;

/**
 * A JPA annotated Entity Object to Table mapping.
 *
 * https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html
 *
 * http://chstath.blogspot.com/2007/05/field-access-vs-property-access-in-jpa.html
 *
 * http://www.vogella.com/tutorials/JAXB/article.html
 *
 * See src/main/resources/hibernate.cfg.xml ==>
 * <mapping class="nve.pesoft.models.Recipe"/>
 */
@Entity
@XmlRootElement(namespace = "Recipe")
@Table(name = "Recipe")
public class Recipe extends AbstractModelHelpers implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "RecipeID", nullable = false)
    private Integer RecipeID; // PK

    @Column(name = "DesignID", nullable = false, length = 20)
    private String DesignID;

    @Column(name = "RecipeRevisionNo", nullable = false)
    private Integer RecipeRevisionNo;

    @Column(name = "RecipeName", nullable = true, length = 45)
    private String RecipeName;

    @Column(name = "RecipeDescription", nullable = true, length = 50)
    private String RecipeDescription;

    @Column(name = "ReleaseMode", nullable = true, length = 45)
    private String ReleaseMode;

    @Column(name = "GrammarName", nullable = true, length = 50)
    private String GrammarName;

    @Column(name = "CreateDatetime", nullable = true)
    @Temporal(TemporalType.TIMESTAMP)
    private Date CreateDatetime;

    @Column(name = "UpdateDatetime", nullable = true)
    @Temporal(TemporalType.TIMESTAMP)
    private Date UpdateDatetime;

    @Column(name = "OwnerUserName", nullable = true, length = 45)
    private String OwnerUserName;

//    @OneToMany(targetEntity = RecipeBucket.class, mappedBy = "recipe", fetch = FetchType.LAZY)
//    @ExcludeGson
//    private List<RecipeBucket> recipeBucketList = new ArrayList<>();
//
//    @OneToMany(targetEntity = RecipeRuleSet.class, mappedBy = "recipe", fetch = FetchType.LAZY)
//    @ExcludeGson
//    private List<RecipeRuleSet> recipeRuleSetList = new ArrayList<>();
```

```
    // No arg constructor needed for serialization.  (Gson)
    public Recipe() {
    }

    // Copy constructor - UPPER / LOWER case done here.  One spot.
    public Recipe(Recipe rhsRecipe) {
        this.RecipeID = rhsRecipe.getRecipeID();
        this.DesignID = this.toUpperIfNotNull(rhsRecipe.getDesignID());
        this.RecipeRevisionNo = rhsRecipe.getRecipeRevisionNo();
        this.RecipeName = rhsRecipe.getRecipeName();
        this.RecipeDescription = rhsRecipe.getRecipeDescription();
        this.ReleaseMode = this.toUpperIfNotNull(rhsRecipe.getReleaseMode());
        this.GrammarName = rhsRecipe.getGrammarName();
        this.CreateDatetime = rhsRecipe.getCreateDatetime();
        this.UpdateDatetime = rhsRecipe.getUpdateDatetime();
        this.OwnerUserName = this.toLowerIfNotNull(rhsRecipe.getOwnerUserName());
    }

    public String toHeaderString() {
        StringBuilder sb = new StringBuilder();
        sb.append("RecipeID").append(",");
        sb.append("DesignID").append(",");
        sb.append("RecipeRevisionNo").append(",");
        sb.append("RecipeName").append(",");
        sb.append("RecipeDescription").append(",");
        sb.append("GrammarName").append(",");
        sb.append("ReleaseMode").append(",");
        sb.append("CreateDatetime").append(",");
        sb.append("UpdateDatetime").append(",");
        sb.append("OwnerUserName").append("\n");

        return sb.toString();
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(RecipeID).append(",");
        sb.append(DesignID).append(",");
        sb.append(RecipeRevisionNo).append(",");
        sb.append(RecipeName).append(",");
        sb.append(RecipeDescription).append(",");
        sb.append(GrammarName).append(",");
        sb.append(ReleaseMode).append(",");
        sb.append(DateUtils.getProjectDateString(CreateDatetime)).append(",");
        sb.append(DateUtils.getProjectDateString(UpdateDatetime)).append(",");
        sb.append(OwnerUserName).append("\n");

        return sb.toString();
    }

    // Setters/Getters removed for documentation.
}
```

AbstractModelHelpers

```
package nve.pesoft.models;

public abstract class AbstractModelHelpers {

    public AbstractModelHelpers() {

    }

    protected final String toUpperIfNotNull(String mystring) {
        String result = null;
        if (mystring != null) {
            result = mystring.trim().toUpperCase();
        }
        return result;
    }

    protected final String toLowerIfNotNull(String mystring) {
        String result = null;
        if (mystring != null) {
            result = mystring.trim().toLowerCase();
        }
        return result;
    }

    protected final byte[] copyByteArray(byte[] oldByteArray) {
        byte[] newByteArray = null;

        if(oldByteArray != null) {
            newByteArray = new byte[oldByteArray.length];
            System.arraycopy(oldByteArray, 0, newByteArray, 0, oldByteArray.length);
        }

        return newByteArray;
    }

}
```

Example Resource

```
package nve.pesoft.restapi;

import java.text.ParseException;
import java.util.List;
import javax.persistence.ParameterMode;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import nve.pesoft.HibernateResourceAbstract;
import nve.pesoft.JsonResponse;
import nve.pesoft.enums.QuerySource;
import nve.pesoft.exceptions.DateUtilsException;
import nve.pesoft.models.Recipe;
import nve.pesoft.utils.DateUtils;
import org.hibernate.Criteria;
import org.hibernate.LockOptions;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;
import org.hibernate.procedure.ProcedureCall;
import org.hibernate.procedure.ProcedureOutputs;

@Path("/recipe")
public class RecipeResource extends HibernateResourceAbstract {

    private final static int TIMEOUT_SECONDS = 60;

//    private final static QuerySource querySource = QuerySource.STORED_PROCEDURE;
    private final static QuerySource querySource = QuerySource.HIBERNATE_GENERATED;

    public RecipeResource() {
        this.setDebug(false);
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response index() {
        JsonResponse jsonResponse;
        try {
            this.beginTransaction(TIMEOUT_SECONDS);

              List<Recipe> recipeList = this.session.createCriteria(Recipe.class).list();
              jsonResponse = this.hibernateObjectToJsonResponse(recipeList);

              this.commitTransaction();
        } catch (Exception ex) {
            jsonResponse = this.rollbackTransaction(ex);
        } finally {
            this.closeSession();
        }
        return Response.status(jsonResponse.getStatus()).entity(jsonResponse.getMessage()).build();
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON) // GsonMessageBodyReader.java Marshalls JSON into the Object in the
method params.
    @Produces(MediaType.APPLICATION_JSON)
    public Response create(Recipe recipeFromJson) { // Recipe is the Object

        JsonResponse jsonResponse = null;

        try {
            this.beginTransaction(TIMEOUT_SECONDS);
```

```
            switch (RecipeResource.querySource) {
                case HIBERNATE_GENERATED:
                    Recipe recipe = new Recipe();
                    int recipeRevisionNo = 1;
                    Criteria cr = this.session.createCriteria(Recipe.class);
                    cr.setProjection(Projections.max("RecipeRevisionNo"));
                    cr.add(Restrictions.eq("DesignID", recipeFromJson.getDesignID().toUpperCase()));
                    Object result = cr.uniqueResult(); // This can return NULL if no row.
//Hibernate:
//    select
//        max(this_.RecipeRevisionNo) as y0_
//    from
//        Recipe this_
//    where
//        this_.DesignID=?
                    if (result != null) {
                        int maxRecipeRevisionNo = (int) result;
                        recipeRevisionNo = maxRecipeRevisionNo + 1; // This is where we bump recipe rev no.
                    }   // Notice we ignore  RecipeID, CreatedDatetime, and UpdatedDatetime.
                    // They are all managed by the database insertion and the .onCreate() call below.
                    recipe.setDesignID(recipeFromJson.getDesignID().toUpperCase());
                    recipe.setRecipeRevisionNo(recipeRevisionNo);
                    recipe.setRecipeName(recipeFromJson.getRecipeName());
                    recipe.setOwnerUserName(recipeFromJson.getOwnerUserName());
                    recipe.onCreate(this.getCurrentMethod());
                    this.session.save(recipe);
                    jsonResponse = this.hibernateObjectToJsonResponse(recipe);
                    break;
                case STORED_PROCEDURE:

                    // GOTCHA!!  Every attribute MUST be non-null or stored procedure call will blow up.
                    // dbo.ins_recipe(?, ?, ?, ?)  expects 4 parameters.
                    // If any are null, then it can not bind input variables.
                    // I could've added initialization in Recipe() constructor, but I did not because that would
break json marshalling.

                    String storedProcName = "dbo.ins_recipe";
                    ProcedureCall call = this.session.createStoredProcedureCall(storedProcName)
                            .registerParameter0("iDesignID", String.class, ParameterMode.IN)
                            .registerParameter0("iRecipeName", String.class, ParameterMode.IN)
                            .registerParameter0("iOwnerUserName", String.class, ParameterMode.IN)
                            .registerParameter0("oNumRowsAffected", Integer.class, ParameterMode.OUT);
                    // Bind variables
                    call.getParameterRegistration("iDesignID").bindValue(recipeFromJson.getDesignID());
                    call.getParameterRegistration("iRecipeName").bindValue(recipeFromJson.getRecipeName());

call.getParameterRegistration("iOwnerUserName").bindValue(recipeFromJson.getOwnerUserName());

                    ProcedureOutputs outputs = call.getOutputs(); // This will EXECUTE stored procedure.

                    int numRowsAffected = (int) outputs.getOutputParameterValue("oNumRowsAffected");

                    jsonResponse = this.getStoredProcedureJsonResponse(numRowsAffected, storedProcName);
                    break;
                default:
                    jsonResponse = new JsonResponse();
                    break;
            }
            this.commitTransaction();
        } catch (Exception ex) {
            jsonResponse = this.rollbackTransaction(ex);
        } finally {
            this.closeSession();
        }

        return Response.status(jsonResponse.getStatus()).entity(jsonResponse.getMessage()).build();
    }

    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.APPLICATION_JSON)
    public Response create_form_urlencoded(
            @FormParam("DesignID") String designId,
            @FormParam("RecipeName") String recipeName,
            @FormParam("OwnerUserName") String ownerUserName
    ) {
        Recipe recipeFromForm = new Recipe();
        recipeFromForm.setDesignID(designId);
```

```
        recipeFromForm.setRecipeName(recipeName);
        recipeFromForm.setOwnerUserName(ownerUserName);

        return this.create(recipeFromForm);
    }

    @GET
    @Path("/{recipeId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response show(@PathParam("recipeId") int recipeId) {
        JsonResponse jsonResponse;

        try {
            this.beginTransaction(TIMEOUT_SECONDS);
            Recipe recipe = (Recipe) session.get(Recipe.class, recipeId);
            jsonResponse = this.hibernateObjectToJsonResponse(recipe);

            this.commitTransaction();
        } catch (Exception ex) {
            jsonResponse = this.rollbackTransaction(ex);
        } finally {
            this.closeSession();
        }

        return Response.status(jsonResponse.getStatus()).entity(jsonResponse.getMessage()).build();
    }

    @GET
    @Path("/{recipeId}/edit")
    @Produces(MediaType.APPLICATION_JSON)
    public Response edit(@PathParam("recipeId") int recipeId) {
        return this.show(recipeId);
    }

    @PUT
    @Path("/{recipeId}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response update(
            Recipe recipeFromJson,
            @PathParam("recipeId") int recipeId
    ) {

        JsonResponse jsonResponse;
        try {
            this.beginTransaction(TIMEOUT_SECONDS);

// Setting a pessimistic LOCK with LockOptions.UPGRADE.
// commit or rollback will clear the lock.
//
// http://dev.mysql.com/doc/refman/5.7/en/innodb-locking-reads.html
//
// Locking of rows for update using SELECT FOR UPDATE only applies when autocommit
// is disabled (either by beginning transaction with START TRANSACTION or by
// setting autocommit to 0. (see hibernate.cfg.xml)
//
// If autocommit is enabled, the rows matching the specification are not locked.
//
//Hibernate:                                                        LOCKING READ
//    select
//        recipe0_.RecipeID as RecipeID1_0_0_,
//        recipe0_.CreateDatetime as CreateDa2_0_0_,
//        recipe0_.DesignID as DesignID3_0_0_,
//        recipe0_.OwnerUserName as OwnerUse4_0_0_,
//        recipe0_.RecipeName as RecipeNa5_0_0_,
//        recipe0_.RecipeRevisionNo as RecipeRe6_0_0_,
//        recipe0_.UpdateDatetime as UpdateDa7_0_0_
//    from
//        Recipe recipe0_
//    where
//        recipe0_.RecipeID=? for update
//
//Hibernate:
//    update
//        Recipe
//    set
//        CreateDatetime=?,
//        DesignID=?,
```

```
//        OwnerUserName=?,
//        RecipeName=?,
//        RecipeRevisionNo=?,
//        UpdateDatetime=?
//    where
//        RecipeID=?
            if (recipeFromJson.getUpdateDatetime() == null) {
                jsonResponse = this.getErrorJsonResponse(
                        "Require => UpdateDatetime to be passed in so that UPDATE record dirty read/write can
take place.",
                        Response.Status.BAD_REQUEST,
                        Response.Status.BAD_REQUEST.getStatusCode());
            } else {

                Recipe recipe = (Recipe) session.get(Recipe.class, recipeId, LockOptions.UPGRADE);

                if (recipe == null) { // No row to update.
                    jsonResponse = this.hibernateObjectToJsonResponse(recipe);
                } else if (DateUtils.areDatesEqualToMillisecond(
                        recipeFromJson.getUpdateDatetime(),
                        recipe.getUpdateDatetime()) == false) {

                    jsonResponse = this.getErrorJsonResponse(
                            "Failed dirty read. The current row is NOT the most up to date row.",
                            Response.Status.CONFLICT,
                            Response.Status.CONFLICT.getStatusCode());
                } else {
                    // DO NOT SET RecipeID

                    // allow setting of individual fields.
                    if (recipeFromJson.getDesignID() != null) {
                        recipe.setDesignID(recipeFromJson.getDesignID());
                    }

                    // We can NOT set recipe revision no.  Only create() can bump it.
                    if (recipeFromJson.getRecipeName() != null) {
                        recipe.setRecipeName(recipeFromJson.getRecipeName());
                    }
                    if (recipeFromJson.getOwnerUserName() != null) {
                        recipe.setOwnerUserName(recipeFromJson.getOwnerUserName());
                    }

                    recipe.onUpdate(this.getCurrentMethod());
                    session.update(recipe);
                    jsonResponse = this.hibernateObjectToJsonResponse(recipe);
                }
            }

            this.commitTransaction();
        } catch (Exception ex) {
            jsonResponse = this.rollbackTransaction(ex);
        } finally {
            this.closeSession();
        }

        return Response.status(jsonResponse.getStatus()).entity(jsonResponse.getMessage()).build();
    }

    @PUT
    @Path("/{recipeId}")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.APPLICATION_JSON)
    public Response update_form_urlencoded(
            @PathParam("recipeId") int recipeId,
            @FormParam("DesignID") String designId,
            @FormParam("RecipeName") String recipeName,
            @FormParam("OwnerUserName") String ownerUserName,
            @FormParam("UpdateDatetime") String updateDatetime
    ) {
        Recipe recipeFromForm = new Recipe();
        try {
            recipeFromForm.setDesignID(designId);
            recipeFromForm.setRecipeName(recipeName);
            recipeFromForm.setOwnerUserName(ownerUserName);
            recipeFromForm.setUpdateDatetime(DateUtils.parse(updateDatetime));
        } catch (ParseException | DateUtilsException ex) {
            throw new WebApplicationException(ex.getMessage());
        }
```

```
        return this.update(recipeFromForm, recipeId);
    }

    @DELETE
    @Path("/{recipeId}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete(@PathParam("recipeId") int recipeId
    ) {
        JsonResponse jsonResponse;

        try {
            this.beginTransaction(TIMEOUT_SECONDS);

            Recipe recipe = (Recipe) session.get(Recipe.class, recipeId);

            if (recipe != null) {
                session.delete(recipe);
            }

            jsonResponse = this.hibernateObjectToJsonResponse(recipe);

            this.commitTransaction();
        } catch (Exception ex) {
            jsonResponse = this.rollbackTransaction(ex);
        } finally {
            this.closeSession();
        }

        return Response.status(jsonResponse.getStatus()).entity(jsonResponse.getMessage()).build();
    }
}
```

GsonMessageBodyReader

```
package nve.pesoft.restapi.providers;

import com.google.gson.FieldNamingPolicy;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.lang.annotation.Annotation;
import java.lang.reflect.Type;
import java.util.Date;
import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.MultivaluedMap;
import javax.ws.rs.ext.MessageBodyReader;
import javax.ws.rs.ext.Provider;
import nve.pesoft.models.gson.GsonUTCDateAdapter;
import nve.pesoft.models.Recipe;
import nve.pesoft.models.RecipeBucket;
import nve.pesoft.models.RecipeRule;
import nve.pesoft.models.RecipeRuleSet;
import nve.pesoft.models.RuleSet;
import nve.pesoft.models.gson.GsonByteArrayAdapter;
import nve.pesoft.models.gson.RecipeBucketDeserializer;
import nve.pesoft.models.gson.RecipeDeserializer;
import nve.pesoft.models.gson.RecipeRuleDeserializer;
import nve.pesoft.models.gson.RecipeRuleSetDeserializer;
import nve.pesoft.models.gson.RuleSetDeserializer;

/**
 * The MessageBodyReader is used to deserialize the JSON string into a Java
 * object.
 *
 * MessageBodyReader<Object>, as the name suggests, is an extension that
 * supports reading the message body representation from an input stream and
 * converting the data into an instance of a specific Java type.
 *
 */
@Provider
@Consumes(MediaType.APPLICATION_JSON)
public class CustomMessageBodyReader implements MessageBodyReader<Object> {

    private static final String UTF_8 = "UTF-8";
    private Gson gson;

    public CustomMessageBodyReader() {
    }

    /**
     * Don't forget to add new registerTypeAdapters for new deserializers.
     *
     * @return
     */
    private Gson getGson() {
        if (gson == null) {
            final GsonBuilder gsonBuilder = new GsonBuilder();
            gson = gsonBuilder.disableHtmlEscaping()
                    .registerTypeAdapter(Date.class, new GsonUTCDateAdapter())
                    .registerTypeAdapter(byte[].class, new GsonByteArrayAdapter())
                    .registerTypeAdapter(Recipe.class, new RecipeDeserializer())
                    .registerTypeAdapter(RecipeBucket.class, new RecipeBucketDeserializer())
                    .registerTypeAdapter(RecipeRuleSet.class, new RecipeRuleSetDeserializer())
                    .registerTypeAdapter(RuleSet.class, new RuleSetDeserializer())
                    .registerTypeAdapter(RecipeRule.class, new RecipeRuleDeserializer())
                    .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
                    .setPrettyPrinting()
                    .serializeNulls()
                    .create();
        }
```

```
        return gson;
    }

    /**
     * If type is in our Set, then pass back true.
     *
     * @param type
     * @param type1
     * @param antns
     * @param mt
     * @return
     */
    @Override
    public boolean isReadable(Class<?> type, Type type1, Annotation[] antns, MediaType mt) {
        boolean doAction = false;

        if (type == Recipe.class
                || type == RecipeBucket.class
                || type == RecipeRuleSet.class
                || type == RuleSet.class
                || type == RecipeRule.class
                || type == List.class
                ) {
            doAction = true;
        }

        return doAction;
    }

    /**
     * This method should not be changed.
     *
     * @param type
     * @param genericType
     * @param annotations
     * @param mediaType
     * @param httpHeaders
     * @param entityStream
     * @return
     * @throws IOException
     * @throws WebApplicationException
     */
    @Override
    public Object readFrom(
            Class<Object> type,
            Type genericType,
            Annotation[] annotations,
            MediaType mediaType,
            MultivaluedMap<String, String> httpHeaders,
            InputStream entityStream
    ) throws IOException, WebApplicationException {

        InputStreamReader streamReader = null;
        try {
            streamReader = new InputStreamReader(entityStream, UTF_8);
        } catch (UnsupportedEncodingException ex) {
            throw ex;
        }
        try {
            Type jsonType;
            if (type.equals(genericType)) {
                jsonType = type;
            } else {
                jsonType = genericType;
            }

            return getGson().fromJson(streamReader, jsonType);
        } finally {
            try {
                streamReader.close();
            } catch (IOException ex) {
                throw ex;
            }
        }
    }
}
```

GsonUTCDateAdapter

```
package nve.pesoft.models.gson;

import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonParseException;
import com.google.gson.JsonPrimitive;
import com.google.gson.JsonSerializationContext;
import com.google.gson.JsonSerializer;
import java.lang.reflect.Type;
import java.text.ParseException;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import nve.pesoft.exceptions.DateUtilsException;
import nve.pesoft.utils.DateUtils;

/**
 *
 */
public class GsonUTCDateAdapter implements JsonSerializer<Date>,JsonDeserializer<Date>{

    public GsonUTCDateAdapter() {
    }

    @Override
    public synchronized JsonElement serialize(Date date, Type type, JsonSerializationContext
jsonSerializationContext) {
        return new JsonPrimitive(DateUtils.getDateStringIso8601(date));
    }

    @Override
    public synchronized Date deserialize(JsonElement jsonElement, Type type, JsonDeserializationContext
jsonDeserializationContext) {
        Date date = null;
        try {
            date = DateUtils.parse(jsonElement.getAsString());
        } catch (ParseException e) {
            throw new JsonParseException(e);
        } catch (DateUtilsException ex) {
            Logger.getLogger(GsonUTCDateAdapter.class.getName()).log(Level.SEVERE, null, ex);
        }
        return date;
    }

}
```

JsonResponse

```
package nve.pesoft;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.util.List;
import javax.ws.rs.core.Response;

public class JsonResponse {
    private Response.Status status; //all
    private int statusCode;      // ??
    private String callerClass;  // exception
    private String message;      // all
    private String cause;        // exception
    private String uri;
    private String method;
    private List<String> stackTraceElements; // exception

    public JsonResponse() {

    }

    public JsonResponse(Response.Status status, String message) {
        this.status = status;
        this.message = message;
    }

    @Override
    public String toString() {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setPrettyPrinting();
        Gson gson = gsonBuilder.create();
        String json = gson.toJson(this);
        return json;
    }

    public List<String> getStackTraceElements() {
        return stackTraceElements;
    }

    public void setStackTraceElements(List<String> stackTraceElements) {
        this.stackTraceElements = stackTraceElements;
    }

// Setters/Getters removed for documentation.
}
```

Sample Deserializer

```
package nve.pesoft.models.gson;

import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParseException;
import java.lang.reflect.Type;
import nve.pesoft.models.Recipe;
import nve.pesoft.utils.JsonUtils;

public class RecipeDeserializer implements JsonDeserializer<Recipe> {

    @Override
    public Recipe deserialize(JsonElement je, Type type, JsonDeserializationContext context) throws
JsonParseException {

        JsonObject jsonObject = je.getAsJsonObject();

        // allocate new object.
        Recipe recipe = new Recipe();

        // set attributes.  Notice the getString(), getInteger(), and getDate() handle the NULLs.
        recipe.setRecipeID(JsonUtils.getInteger(jsonObject.get("RecipeID")));
        recipe.setDesignID(JsonUtils.getStringUpper(jsonObject.get("DesignID")));
        recipe.setRecipeRevisionNo(JsonUtils.getInteger(jsonObject.get("RecipeRevisionNo")));
        recipe.setRecipeName(JsonUtils.getString(jsonObject.get("RecipeName")));
        recipe.setRecipeDescription(JsonUtils.getString(jsonObject.get("RecipeDescription")));
        recipe.setGrammarName(JsonUtils.getStringUpper(jsonObject.get("GammarName")));
        recipe.setReleaseMode(JsonUtils.getStringUpper(jsonObject.get("ReleaseMode")));
        recipe.setCreateDatetime(JsonUtils.getDate(jsonObject.get("CreateDatetime")));
        recipe.setUpdateDatetime(JsonUtils.getDate(jsonObject.get("UpdateDatetime")));
        recipe.setOwnerUserName(JsonUtils.getStringUpper(jsonObject.get("OwnerUserName")));

        return recipe;
    }

}
```

DateUtils

```
package nve.pesoft.utils;

import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import nve.pesoft.exceptions.DateUtilsException;

public final class DateUtils {

    private static final String DATE_FORMAT = "yyyy-MM-dd HH:mm:ss.SSS zzz";
    private static final String DATE_FORMAT2 = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'";
    private static final SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT);
    private static final SimpleDateFormat sdf2 = new SimpleDateFormat(DATE_FORMAT2);

    // This is SET in nve.pesoft.ServletContext.java for the entire application.
    // TimeZone.setDefault(TimeZone.getTimeZone("Etc/UTC"));
    // This is for Gson serialization/deserialization and adapters.
    // Change this in one place
    public static String getProjectDateFormatString() {
        return DateUtils.DATE_FORMAT2;
    }

    public static String getProjectDateString(Date date) {
        if (date == null) {
            return null;
        }
        return DateUtils.getDateStringIso8601(date);
    }

    public static boolean areDatesEqualToMillisecond(Date date1, Date date2) {
```

```
        boolean result = false;

        if (date1 != null && date2 != null) {
            String str1 = sdf2.format(date1);
            String str2 = sdf2.format(date2);
            result = str1.equals(str2);
        }

        return result;
    }

    public static Timestamp getTimestamp(Date date) {
        Timestamp resultTimestamp = new Timestamp(date.getTime());
        return resultTimestamp;
    }

    public static String getDateString(Date date) {
        return sdf.format(date);
    }

    public static String getDateStringIso8601(Date date) {
        return sdf2.format(date);
    }

    public static Date parse(String calString) throws ParseException, DateUtilsException {
        Date date;

        String tmpCalString = calString.trim();
        //System.out.println("CALSTRING " + calString);

        if (tmpCalString.matches("^[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}\\s+[0-9]{2}:[0-9]{2}:[0-9]{2}\\.[0-
9]{3}\\s+[A-Z]+$")) {
            date = sdf.parse(tmpCalString);
        } else if (tmpCalString.matches("^[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\\.[0-
9]{3}Z$")) {
            date = sdf2.parse(tmpCalString);
        } else {
            String msg = "ERROR: Unable to parse Date input. => " + calString + "\n=>" + tmpCalString + "<=";
            throw new DateUtilsException(msg);
        }

        return (date);
    }
}
```

JsonUtils

```
package nve.pesoft.utils;

import nve.pesoft.models.gson.HibernateProxyTypeAdapter;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonNull;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.google.gson.JsonSyntaxException;
import com.google.gson.reflect.TypeToken;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.sql.Timestamp;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ws.rs.core.Response;
import nve.pesoft.JsonResponse;
import nve.pesoft.exceptions.DateUtilsException;
```

```
import nve.pesoft.exceptions.JsonUtilsException;
import nve.pesoft.models.gson.AnnotationExclusionStrategy;
import nve.pesoft.models.gson.CollectionAdapter;
import org.hibernate.Session;

public class JsonUtils {

    private static boolean verbose = false;

    public JsonUtils() {
    }

    /**
     * Set this if you want to see more info.
     *
     * @param verbose boolean
     */
    public static void setVerbose(boolean verbose) {
        JsonUtils.verbose = verbose;
    }

    /**
     * Convert a simple Object to JSON in human readable form
     *
     * @param <T> Object type
     * @param obj object instance
     * @return String
     */
    public static <T> String objectToJsonPretty(T obj) {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
        gsonBuilder.setPrettyPrinting();
        gsonBuilder.serializeNulls();
        gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());

        Gson gson = gsonBuilder.create();
        StringBuilder sb = new StringBuilder();
        sb.append(gson.toJson(obj));
        return (sb.toString());
    }

    /**
     * Do not serialize NULL fields.
     *
     * @param <T> Object type
     * @param obj object instance
     * @return JSON String
     */
    public static <T> String objectToJsonPrettyNoNulls(T obj) {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
        gsonBuilder.setPrettyPrinting();
        gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());

        Gson gson = gsonBuilder.create();
        StringBuilder sb = new StringBuilder();
        sb.append(gson.toJson(obj));
        return (sb.toString());
    }

    /**
     * Convert a simple Object to JSON in compact form.
     *
     * @param <T> Generic type
     * @param obj object instance
     * @return String
     */
    public static <T> String objectToJsonCompact(T obj) {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
        gsonBuilder.serializeNulls();
        gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());

        Gson gson = gsonBuilder.create();
        StringBuilder sb = new StringBuilder();
        sb.append(gson.toJson(obj));
        return (sb.toString());
    }
```

```
/**
 * Do not serialize NULL fields.
 *
 * @param <T> Object type
 * @param obj object instance
 * @return JSON String
 */
public static <T> String objectToJsonCompactNoNulls(T obj) {
    GsonBuilder gsonBuilder = new GsonBuilder();
    gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
    gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());

    Gson gson = gsonBuilder.create();
    StringBuilder sb = new StringBuilder();
    sb.append(gson.toJson(obj));
    return (sb.toString());
}

/**
 * Convert a Hibernate Object to JSON in human readable form You MUST have
 * an open Hibernate Session to unproxy object for json serialization.
 *
 * @param <T> Object type
 * @param session
 * @param obj object instance
 * @return String
 */
public static <T> String hibernateObjectToJsonPretty(Session session, T obj) {
    JsonResponse errResponse = new JsonResponse(
            Response.Status.BAD_REQUEST,
            "You MUST have an open Hibernate Session to unproxy object for json serialization.");

    if (session == null || (session.isOpen() == false)) {
        return (errResponse.toString());
    }

    GsonBuilder gsonBuilder = new GsonBuilder();
    gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
    gsonBuilder.setPrettyPrinting();
    gsonBuilder.serializeNulls();

    /**
     * This TypeAdapter unproxies Hibernate proxied objects, and serializes
     * them through the registered (or default) TypeAdapter of the base
     * class.
     */
    gsonBuilder.registerTypeAdapterFactory(HibernateProxyTypeAdapter.FACTORY);
    gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());
    //gsonBuilder.registerTypeAdapter(byte[].class, new GsonByteArrayAdapter());
    gsonBuilder.addSerializationExclusionStrategy(new AnnotationExclusionStrategy());

    Gson gson = gsonBuilder.create();
    StringBuilder sb = new StringBuilder();
    sb.append(gson.toJson(obj));
    return (sb.toString());
}

/**
 * Convert a Hibernate Object to JSON in human readable form You MUST have
 * an open Hibernate Session to unproxy object for json serialization. Do
 * not serialize NULL fields.
 *
 * @param <T> Object type
 * @param session
 * @param obj object instance
 * @return JSON String
 */
public static <T> String hibernateObjectToJsonPrettyNoNulls(Session session, T obj) {
    JsonResponse errResponse = new JsonResponse(
            Response.Status.BAD_REQUEST,
            "You MUST have an open Hibernate Session to unproxy object for json serialization.");

    if (session == null || (session.isOpen() == false)) {
        return (errResponse.toString());
    }
```

```
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
        gsonBuilder.setPrettyPrinting();

        /**
         * This TypeAdapter unproxies Hibernate proxied objects, and serializes
         * them through the registered (or default) TypeAdapter of the base
         * class.
         */
        gsonBuilder.registerTypeAdapterFactory(HibernateProxyTypeAdapter.FACTORY);
        gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());
        //gsonBuilder.registerTypeAdapter(byte[].class, new GsonByteArrayAdapter());
        gsonBuilder.addSerializationExclusionStrategy(new AnnotationExclusionStrategy());

        Gson gson = gsonBuilder.create();
        StringBuilder sb = new StringBuilder();
        sb.append(gson.toJson(obj));
        return (sb.toString());
    }

    /**
     * Convert a Hibernate Object to JSON in human readable form You MUST have
     * an open Hibernate Session to unproxy object for json serialization.
     *
     * @param <T> Generic type
     * @param session
     * @param obj object instance
     * @return String
     */
    public static <T> String hibernateObjectToJsonCompact(Session session, T obj) {
        JsonResponse errResponse = new JsonResponse(
                Response.Status.BAD_REQUEST,
                "You MUST have an open Hibernate Session to unproxy object for json serialization.");

        if (session == null || (session.isOpen() == false)) {
            return (errResponse.toString());
        }

        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
        gsonBuilder.serializeNulls();

        /**
         * This TypeAdapter unproxies Hibernate proxied objects, and serializes
         * them through the registered (or default) TypeAdapter of the base
         * class.
         */
        gsonBuilder.registerTypeAdapterFactory(HibernateProxyTypeAdapter.FACTORY);
        gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());
        gsonBuilder.addSerializationExclusionStrategy(new AnnotationExclusionStrategy());

        Gson gson = gsonBuilder.create();
        StringBuilder sb = new StringBuilder();
        sb.append(gson.toJson(obj));
        return (sb.toString());
    }

    /**
     * Convert a Hibernate Object to JSON in human readable form You MUST have
     * an open Hibernate Session to unproxy object for json serialization. Do
     * not serialize NULL fields.
     *
     * @param <T> Object type
     * @param session
     * @param obj object instance
     * @return JSON String
     */
    public static <T> String hibernateObjectToJsonCompactNoNulls(Session session, T obj) {
        JsonResponse errResponse = new JsonResponse(
                Response.Status.BAD_REQUEST,
                "You MUST have an open Hibernate Session to unproxy object for json serialization.");

        if (session == null || (session.isOpen() == false)) {
            return (errResponse.toString());
        }

        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setDateFormat(DateUtils.getProjectDateFormatString());
```

```
        /**
         * This TypeAdapter unproxies Hibernate proxied objects, and serializes
         * them through the registered (or default) TypeAdapter of the base
         * class.
         */
        gsonBuilder.registerTypeAdapterFactory(HibernateProxyTypeAdapter.FACTORY);
        gsonBuilder.registerTypeHierarchyAdapter(Collection.class, new CollectionAdapter());
        gsonBuilder.addSerializationExclusionStrategy(new AnnotationExclusionStrategy());

        Gson gson = gsonBuilder.create();
        StringBuilder sb = new StringBuilder();
        sb.append(gson.toJson(obj));
        return (sb.toString());
    }

    /**
     * Quick check if JSON string is valid.
     *
     * @param jsonString A valid JSON string
     * @return boolean
     */
    public static boolean isValidJson(String jsonString) {
        try {
            new JsonParser().parse(jsonString);
            return true;
        } catch (JsonSyntaxException jse) {
            return false;
        }
    }

    /**
     * This method will read JSON from a file, removing \r\n if on Windows and
     * replacing with \n only.
     *
     * @param fileName The file should only contain ONE json object or array of
     * objects.
     * @return String
     * @throws JsonUtilsException Captures IO Exception or
     * FileNotExistException.
     */
    public static String readJsonFromFile(String fileName) throws JsonUtilsException {
        String jsonStr = null;
        try {
            File file = new File(fileName);

            if (file.exists() == false) {
                String msg = "File does NOT exist! => " + fileName;
                throw new JsonUtilsException(msg);
            }

            if (verbose) {
                System.out.println("Reading JSON file => " + file.getCanonicalPath() + "\n");
            }
            String line;
            StringBuilder sb = new StringBuilder();
            // BufferedReader removes \r\n and \n with readLine().
            try (BufferedReader br = new BufferedReader(new FileReader(file.getAbsoluteFile()))) {
                while ((line = br.readLine()) != null) {
                    sb.append(line).append("\n"); // here I replace the newline with Unix style.
                }
                jsonStr = sb.toString();  // set the return string
                jsonStr = jsonStr.replaceAll("[ \t\n]+$", "");  // remove whitespace at end always.
            }
        } catch (IOException ex) {
            String msg = "IOExeption from fileName => " + fileName + " for class => " +
JsonUtils.class.getName();
            msg += ex.getMessage();
            throw new JsonUtilsException(msg);
        }
        return (jsonStr);
    }

    /**
     * Write JSON string verbatim to a file.
     *
     * @param jsonStr A valid JSON string
```

```
     * @param fileName Filename to write to
     * @throws JsonUtilsException Captures IO Exception or
     * FileNotExistException.
     */
    public static void writeJsonToFile(String jsonStr, String fileName) throws JsonUtilsException {
        // Ensure the directory is created for fileName.
        String dirname = SysUtils.getDirName(fileName);
        SysUtils.mkdir_p(dirname);

        // write JSON String to file
        File file = new File(fileName);
        try {
            if (!file.exists()) {
                file.createNewFile();
            } else {
                file.delete();
                file.createNewFile();
            }

            try (BufferedWriter bw = new BufferedWriter(new FileWriter(file.getAbsoluteFile()))) {
                bw.write(jsonStr);
            }

            if (verbose) {
                System.out.println("Wrote JSON to file => " + file.getCanonicalPath() + "\n");
            }
        } catch (IOException ex) {
            String msg = "IOExeption from fileName => " + fileName + " for class => " +
JsonUtils.class.getName();
            msg += ex.getMessage();
            throw new JsonUtilsException(msg);
        }
    }

    /**
     * Convert a JSON string to human readable format.
     *
     * @param jsonString A valid JSON string
     * @return String
     * @throws JsonUtilsException Captures JsonSyntaxException
     */
    public static String toPrettyFormat(String jsonString) throws JsonUtilsException {
        String prettyJson = "";

        try {
            GsonBuilder gsonBuilder = new GsonBuilder();
            gsonBuilder.setPrettyPrinting();
            gsonBuilder.serializeNulls();
            Gson gson = gsonBuilder.create();

            JsonParser parser = new JsonParser();
            JsonElement rootJsonElement = parser.parse(jsonString);

            if (rootJsonElement.isJsonObject()) {
                // java.lang.reflect.Type
                Type type = new TypeToken<JsonObject>() {
                }.getType();
                JsonObject jsonObject = gson.fromJson(jsonString, type);
                prettyJson = gson.toJson(jsonObject);
            } else {
                // java.lang.reflect.Type
                Type type = new TypeToken<ArrayList<JsonObject>>() {
                }.getType();
                ArrayList<JsonObject> jsonListObjects = gson.fromJson(jsonString, type);
                prettyJson = gson.toJson(jsonListObjects);
            }
        } catch (JsonSyntaxException ex) {
            String msg = "ERROR: Invalid JSON.\n\n";
            msg += jsonString;
            throw new JsonUtilsException(msg);
        }

        return prettyJson;
    }

    /**
     * Convert JSON string by removing all newlines and white space not quoted.
```

```
     *
     * @param jsonString A valid JSON string
     * @return String
     * @throws JsonUtilsException Captures JsonSyntaxException
     */
    public static String toCompactFormat(String jsonString) throws JsonUtilsException {
        String compactJson;

        try {
            GsonBuilder gsonBuilder = new GsonBuilder();
            gsonBuilder.serializeNulls();
            Gson gson = gsonBuilder.create();

            JsonParser parser = new JsonParser();
            JsonElement rootJsonElement = parser.parse(jsonString);

            if (rootJsonElement.isJsonObject()) {
                // java.lang.reflect.Type
                Type type = new TypeToken<JsonObject>() {
                }.getType();
                JsonObject jsonObject = gson.fromJson(jsonString, type);
                compactJson = gson.toJson(jsonObject);
            } else {
                // java.lang.reflect.Type
                Type type = new TypeToken<ArrayList<JsonObject>>() {
                }.getType();
                ArrayList<JsonObject> jsonListObjects = gson.fromJson(jsonString, type);
                compactJson = gson.toJson(jsonListObjects);
            }
        } catch (JsonSyntaxException ex) {
            String msg = "ERROR: Invalid JSON.\n\n";
            msg += jsonString;
            throw new JsonUtilsException(msg);
        }
        return compactJson;
    }

    public static String getString(JsonElement je) {
        String resultStr = null;

        if (je != null && (je instanceof JsonNull) == false) {
            resultStr = je.getAsString().trim();
        }
        return resultStr;
    }

    public static String getStringUpper(JsonElement je) {
        String resultStr = null;

        if (je != null && (je instanceof JsonNull) == false) {
            resultStr = je.getAsString().trim().toUpperCase();
        }
        return resultStr;
    }

    public static String getStringLower(JsonElement je) {
        String resultStr = null;

        if (je != null && (je instanceof JsonNull) == false) {
            resultStr = je.getAsString().trim().toLowerCase();
        }
        return resultStr;
    }

    public static Integer getInteger(JsonElement je) {
        Integer resultInt = null;

        if (je != null && (je instanceof JsonNull) == false) {
            resultInt = je.getAsInt();
        }
        return resultInt;
    }

    public static Long getLong(JsonElement je) {
        Long resultLong = null;

        if (je != null && (je instanceof JsonNull) == false) {
            resultLong = je.getAsLong();
```

```
        }
        return resultLong;
    }

    public static Float getFloat(JsonElement je) {
        Float resultFloat = null;
        if (je != null && (je instanceof JsonNull) == false) {
            resultFloat = je.getAsFloat();
        }
        return resultFloat;
    }

    public static Double getDouble(JsonElement je) {
        Double resultDouble = null;
        if (je != null && (je instanceof JsonNull) == false) {
            resultDouble = je.getAsDouble();
        }
        return resultDouble;
    }

    public static Boolean getBoolean(JsonElement je) {
        Boolean resultBoolean = null;
        if (je != null && (je instanceof JsonNull) == false) {
            resultBoolean = je.getAsBoolean();
        }
        return resultBoolean;
    }

    public static Date getDate(JsonElement je) {
        Date resultDate = null;

        if (je != null && (je instanceof JsonNull) == false) {
            try {
                resultDate = DateUtils.parse(je.getAsString());
            } catch (ParseException ex) {
                Logger.getLogger(JsonUtils.class.getName()).log(Level.SEVERE, null, ex);
            } catch (DateUtilsException ex) {
                Logger.getLogger(JsonUtils.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        return resultDate;
    }

    public static Timestamp getTimestamp(JsonElement je) {
        Timestamp resultTimestamp = null;

        if (je != null) {
            try {
                Date someDate = DateUtils.parse(je.getAsString());
                resultTimestamp = new Timestamp(someDate.getTime());
            } catch (ParseException ex) {
                Logger.getLogger(JsonUtils.class.getName()).log(Level.SEVERE, null, ex);
            } catch (DateUtilsException ex) {
                Logger.getLogger(JsonUtils.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        return resultTimestamp;
    }

    public static byte[] getTimestampMicrosoft(JsonElement je) {
        byte[] data = null;

        if (je != null) {
            JsonArray ja = je.getAsJsonArray();
            data = new byte[ja.size()];
            for (int ii = 0; ii < ja.size(); ii++) {
                data[ii] = ja.get(ii).getAsByte();
            }
        }
        return data;
    }
}
```

Chrome browser with Postman plugin.

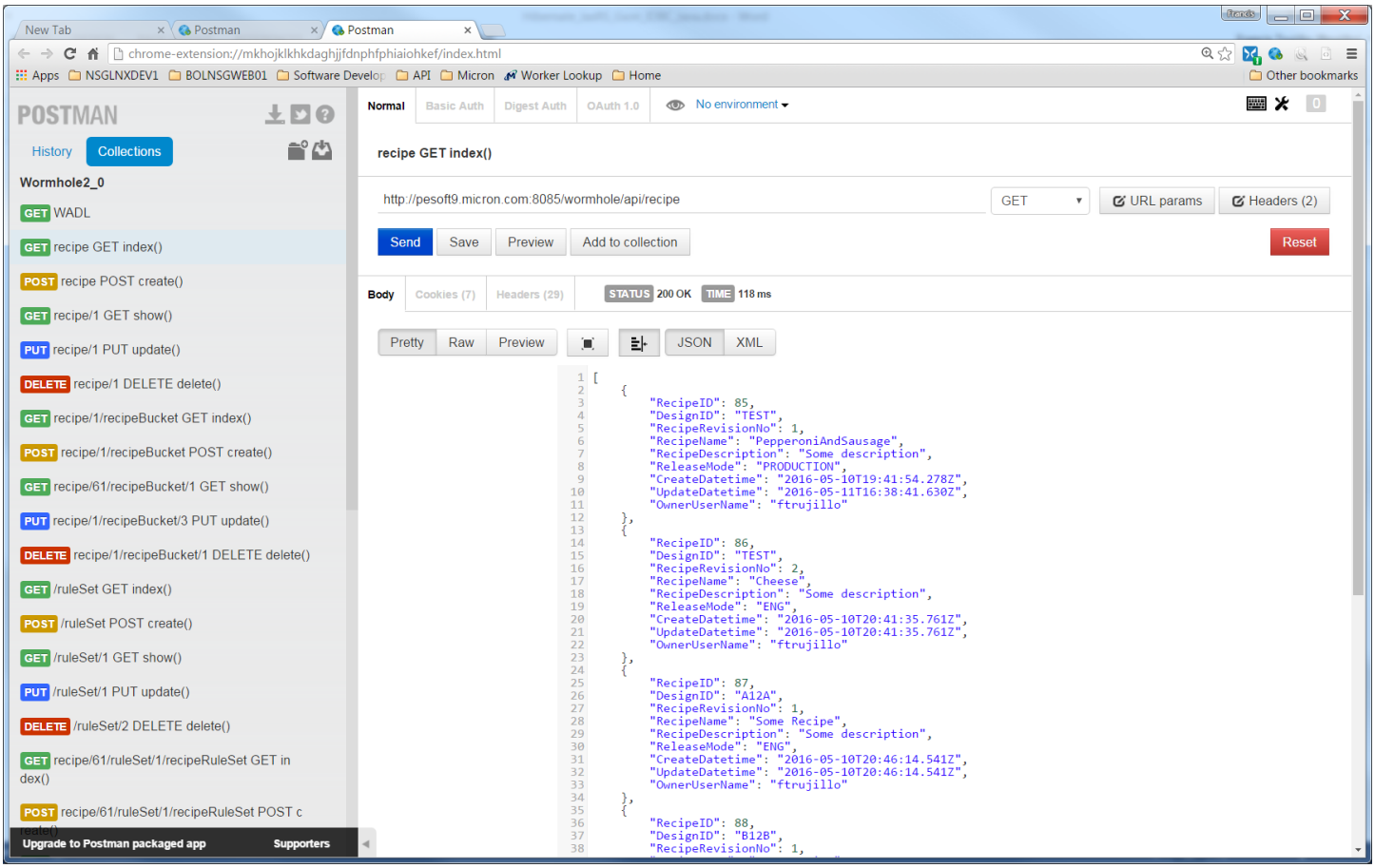
You will need a way to manually test your routes.

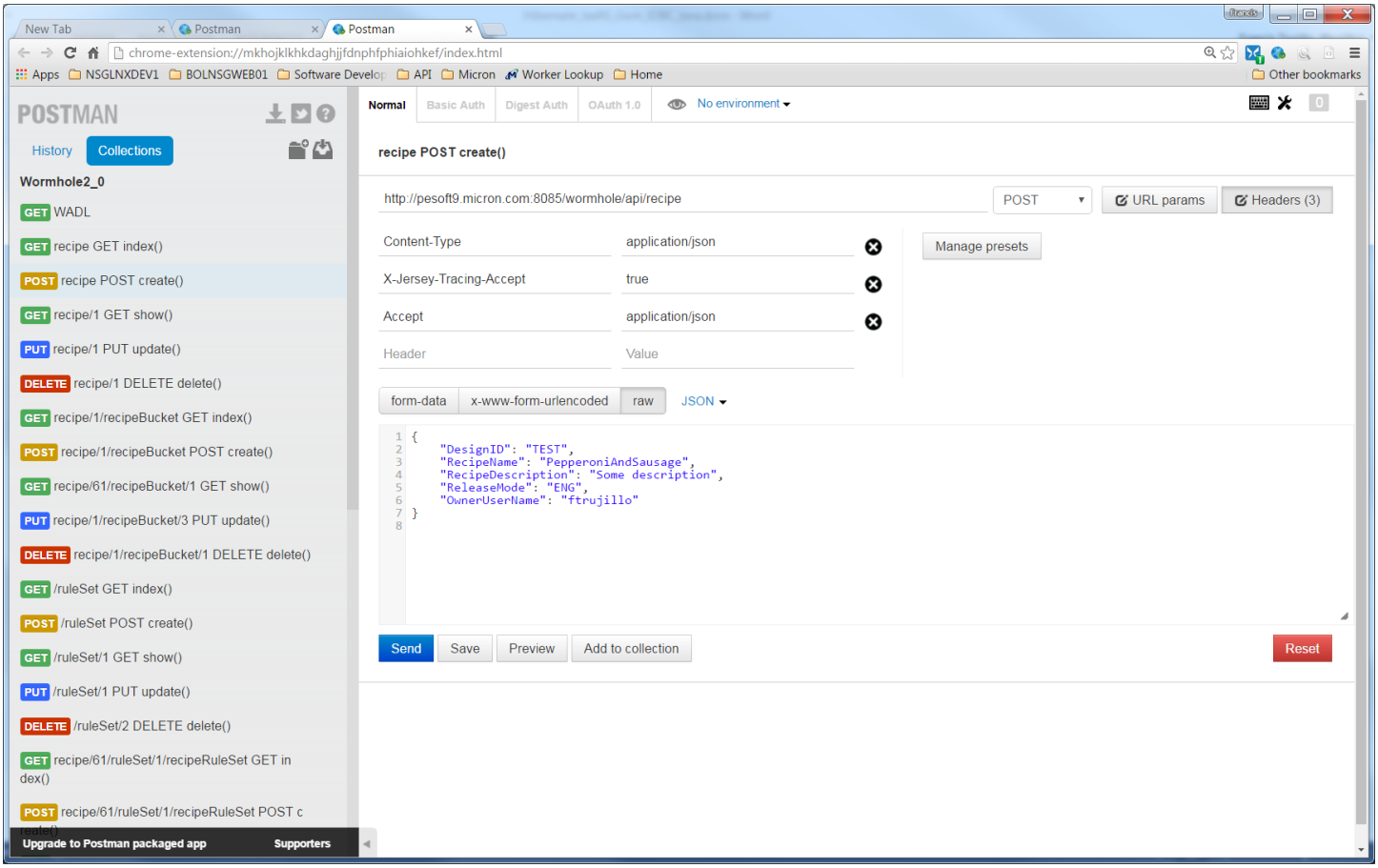
I like this one.

https://chrome.google.com/webstore/detail/postman-rest-client-short/mkhojklkhkdaghjjfdnphfphiaiohkef?hl=en

this one will work as well.

https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=en




New Tab
Postman
Postman
chrome-extension://mkhojklkhkdaghjjfdnphfphiaiohkef/index.html
Apps
NSGLNXDEV1
BOLNSGWEB01
Software Develop
API
Micron
Worker Lookup
Home
Other bookmarks
POSTMAN
History
Collections
Wormhole2_0
GET WADL
GET recipe GET index()
POST recipe POST create()
GET recipe/1 GET show()
PUT recipe/1 PUT update()
DELETE recipe/1 DELETE delete()
GET recipe/1/recipeBucket GET index()
POST recipe/1/recipeBucket POST create()
GET recipe/61/recipeBucket/1 GET show()
PUT recipe/1/recipeBucket/3 PUT update()
DELETE recipe/1/recipeBucket/1 DELETE delete()
GET /ruleSet GET index()
POST /ruleSet POST create()
GET /ruleSet/1 GET show()
PUT /ruleSet/1 PUT update()
DELETE /ruleSet/2 DELETE delete()
GET recipe/61/ruleSet/1/recipeRuleSet GET index()
POST recipe/61/ruleSet/1/recipeRuleSet POST c
Upgrade to Postman packaged app
Supporters
Normal
Basic Auth
Digest Auth
OAuth 1.0
No environment
recipe POST create()
http://pesoft9.micron.com:8085/wormhole/api/recipe
POST
URL params
Headers (3)
Content-Type
application/json
X-Jersey-Tracing-Accept
true
Accept
application/json
Header
Value
Manage presets
form-data
x-www-form-urlencoded
raw
JSON
{
"DesignID": "TEST",
"RecipeName": "PepperoniAndSausage",
"RecipeDescription": "Some description",
"ReleaseMode": "ENG",
"OwnerUserName": "ftrujillo"
}
Send
Save
Preview
Add to collection
Reset