**Integrating Apache CXF, JAX-RS, Spring and Mongo DB (Part II)**

**Author**: Horatiu Stanciu

**Email:** [hstanciu@gmail.com](mailto:hstanciu@gmail.com)

**Date:** 2015-03-25

In this article we will continue the example from [Integrating Apache CXF, JAX-RS, Spring (Part I)](http://horatiustanciu.blogspot.ca/2015/03/integratingapache-cxf-jax-rs-and-spring.html) and we’ll integrate a Mongo DB database. As build tool we will be using Maven.

Similar with before I used for this article the following tools:

**Eclipse Luna 4.4.2**

**Tomcat 7.0.59**

**JDK 1.7**

**Maven 3.0**

Supplementary to Part I you’ll have to install

**MongoDB 3.0.1**

To understand the article, you’ll need Java, JEE, REST web service and Spring knowledge. By following MongoDB installation instruction you can install and load the data from **home-mongodb\scripts\payments.json** (can be found on [Git](https://github.com/hstanciu/CXF_JAXRS_Spring_Maven_MongoDB), where the project is).

Once the tools are properly installed, we’ll do these steps:

1. Create a persistence application
2. Build and configure the persistence application
3. Install the MongoDB locally and load the data
4. Create the web service
5. Deploy and use the service
6. **Create a persistence application**

On hard disk, move to a folder where you will create the project. For me, it was **C:\java\projects\webServiceWorkspace**

Create the project skeleton using the following maven command:

*mvn archetype:generate -DgroupId=biz.korwin -DartifactId=home-mongodb -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false*

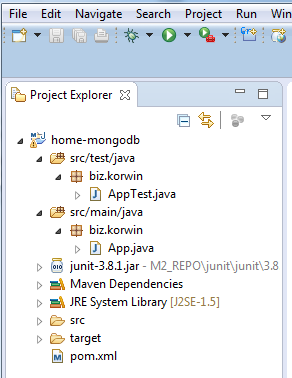
This would create a basic Java application structure. The project will be contained in **home-mongodb** folder. In this project we’ll have the persistence layer. This project will be packaged as a jar file and will be included by the web service project (**home-web-service**) that was developed in [Part I](artifacts).

1. **Build and configure the persistence application**
2. Now in order to quickly import this application in Eclipse run this in **home-mongodb** folder:

*mvn eclipse:eclipse -Dwtpversion=2.0*

This will generate the **.project** and **.classpath** Eclipse files.

1. Import the application in Eclipse. The project structure at this step would look like this:

****

1. Use the following pom.xml 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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>biz.korwin</groupId>

<artifactId>home-mongodb</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<spring.version>4.1.3.RELEASE</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-mongodb</artifactId>

<version>1.6.2.RELEASE</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.10</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.2</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

This would include the needed Spring libraries, logging, Spring-testing, and Spring MongoDB libs.

1. Create the transfer class. This is the model class that will have the Spring MongoDB annotations. It will be used as a value object class to transfer data back and forth between web service and database.

package biz.korwin.web.service.home.model;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

/\*\*

\* Model class. Represents a payment.

\*

\* \*/

@Document //Spring MongoDB annotation. This class will correspond to a collection

public class PaymentTO {

@Id // Spring annotation. This field will correspond to MongoDB’s PK (\_id)

/\* Payment ID\*/

private long id;

/\* Who will be paid; ex: Canada Bank\*/

private String payee;

/\* The payment product; ex: Master Card Petro-Points \*/

private String payeeProduct;

/\* The number of days a new statement will be issued, or -1 if is not cyclical\*/

private int cycle;

/\* Amount to be paid (in cents) \*/

private int amountInCents;

/\* Due date \*/

private String dueDate;

/\* In order for payment to occur at due date, the payment should occur

\* in advance with a couple of days. This date is the limit date when

\* this occurs in order for due date not to be passed.\*/

private String paymentLimitDate;

/\* Date when payment happened. If it didn't happen yet,

\* then this attribute will be null.\*/

private String paidDate;

/\* If true, the payment is not actual.\*/

private boolean deprecated;

public PaymentTO(){}

public PaymentTO(long id,

String payee,

String payeeProduct,

int cycle,

int amountInCents,

String dueDate,

String paymentLimitDate,

String paidDate,

boolean deprecated){

this.id = id;

this.payee = payee;

this.payeeProduct = payeeProduct;

this.cycle = cycle;

this.amountInCents = amountInCents;

this.dueDate = dueDate;

this.paymentLimitDate = paymentLimitDate;

this.paidDate = paidDate;

this.deprecated = deprecated;

}

public PaymentTO(

String payee,

String payeeProduct,

int cycle,

int amountInCents,

String dueDate,

String paymentLimitDate,

String paidDate,

boolean deprecated){

this.id = -1;

this.payee = payee;

this.payeeProduct = payeeProduct;

this.cycle = cycle;

this.amountInCents = amountInCents;

this.dueDate = dueDate;

this.paymentLimitDate = paymentLimitDate;

this.paidDate = paidDate;

this.deprecated = deprecated;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getPayee() {

return payee;

}

public void setPayee(String payee) {

this.payee = payee;

}

public String getPayeeProduct() {

return payeeProduct;

}

public void setPayeeProduct(String payeeProduct) {

this.payeeProduct = payeeProduct;

}

public int getCycle() {

return cycle;

}

public void setCycle(int cycle) {

this.cycle = cycle;

}

public int getAmountInCents() {

return amountInCents;

}

public void setAmountInCents(int amountInCents) {

this.amountInCents = amountInCents;

}

public String getDueDate() {

return dueDate;

}

public void setDueDate(String dueDate) {

this.dueDate = dueDate;

}

public String getPaymentLimitDate() {

return paymentLimitDate;

}

public void setPaymentLimitDate(String paymentLimitDate) {

this.paymentLimitDate = paymentLimitDate;

}

public String getPaidDate() {

return paidDate;

}

public void setPaidDate(String paidDate) {

this.paidDate = paidDate;

}

public boolean isDeprecated() {

return deprecated;

}

public void setDeprecated(boolean deprecated) {

this.deprecated = deprecated;

}

@Override

public String toString() {

return "PaymentTO [id=" + id + ", payee=" + payee + ", payeeProduct="

+ payeeProduct + ", cycle=" + cycle + ", amountInCents="

+ amountInCents + ", dueDate=" + dueDate

+ ", paymentLimitDate=" + paymentLimitDate + ", paidDate="

+ paidDate + ", deprecated=" + deprecated + "]";

}

}

1. Define the DAO layer; we’ll have one interface and one class: **IPaymentDAO.java** and **PaymentDAOImpl.java.** PaymentDAOImpl is using a MongoTemplate to perform the database operations.

**IPaymentDAO.java**

package biz.korwin.web.service.home.dao;

import java.util.List;

import biz.korwin.web.service.home.model.PaymentTO;

/\*\*

\* DAO interface for accessing the Payments data source.

\* \*/

public interface IPaymentDAO {

/\* Returns a payment by payment ID. \*/

public PaymentTO getPaymentById(long id);

/\* Creates a payment. \*/

public PaymentTO create(PaymentTO newPayment);

/\* Deletes a payment. \*/

public boolean delete(long id);

/\* Updates a payment. \*/

public PaymentTO update(PaymentTO updatePayment);

/\* Returns the list of payments that have the same payment limit date. \*/

public List<PaymentTO> getPaymentsByDate(String date);

}

**PaymentDAOImpl.java**

package biz.korwin.web.service.home.dao;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.data.mongodb.core.query.Update;

import org.springframework.stereotype.Repository;

import biz.korwin.web.service.home.model.PaymentTO;

/\*\*

\* DAO implementation for accessing the Payments data source.

\* \*/

@Repository

public class PaymentDAOImpl implements IPaymentDAO{

static final Logger logger = LoggerFactory.getLogger(PaymentDAOImpl.class);

@Autowired

private MongoTemplate mongoTemplate;

@Override

public PaymentTO getPaymentById(long id) {

PaymentTO paymentTO = mongoTemplate.findById(id, PaymentTO.class, "payments");

return paymentTO;

}

@Override

public PaymentTO create(PaymentTO newPayment) {

mongoTemplate.insert(newPayment);

Query query = new Query();

String payee = newPayment.getPayee();

String payeeProduct = newPayment.getPayeeProduct();

int cycle = newPayment.getCycle();

long amountInCents = newPayment.getAmountInCents();

String dueDate = newPayment.getDueDate();

String paymentLimitDate = newPayment.getPaymentLimitDate();

query.addCriteria(Criteria.where("payee").is(payee).and("payeeProduct").is(payeeProduct)

.and("cycle").is(cycle).and("amountInCents").is(amountInCents)

.and("dueDate").is(dueDate).and("paymentLimitDate").is(paymentLimitDate)

.and("paidDate").is(null).and("deprecated").is(false));

PaymentTO paymentTO = mongoTemplate.findOne(query, PaymentTO.class);

return paymentTO;

}

@Override

public boolean delete(long id) {

Query query = new Query();

query.addCriteria(Criteria.where("id").is(id));

PaymentTO paymentTO = mongoTemplate.findAndRemove(query, PaymentTO.class);

if(null == paymentTO){

return false;

}else{

return true;

}

}

@Override

public PaymentTO update(PaymentTO updatePayment) {

Query query = new Query();

query.addCriteria(Criteria.where("id").is(updatePayment.getId()));

String payee = updatePayment.getPayee();

String payeeProduct = updatePayment.getPayeeProduct();

int cycle = updatePayment.getCycle();

long amountInCents = updatePayment.getAmountInCents();

String dueDate = updatePayment.getDueDate();

String paymentLimitDate = updatePayment.getPaymentLimitDate();

String paidDate = updatePayment.getPaidDate();

boolean deprecated = updatePayment.isDeprecated();

Update update = new Update();

update.set("payee", payee);

update.set("payeeProduct", payeeProduct);

update.set("cycle", cycle);

update.set("amountInCents", amountInCents);

update.set("dueDate", dueDate);

update.set("paymentLimitDate", paymentLimitDate);

update.set("paidDate", paidDate);

update.set("deprecated", deprecated);

mongoTemplate.updateFirst(query, update, PaymentTO.class);

PaymentTO paymentTO = mongoTemplate.findOne(query, PaymentTO.class);

return paymentTO;

}

@Override

public List<PaymentTO> getPaymentsByDate(String date) {

// TODO Auto-generated method stub

return null;

}

}

1. The resources are described below, in **src/main/resources**. We have here a Spring application context file (**persistent-applicationContext.xml**) and a log4j file. <mongo:mongo> tag is pointing the host and port and in **mongoTemplate** bean you include the above plus the database name.

**persistent-applicationContext.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:mongo=*"http://www.springframework.org/schema/data/mongo"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd*

*http://www.springframework.org/schema/data/mongo http://www.springframework.org/schema/data/mongo/spring-mongo-1.5.xsd"*>

<context:component-scan base-package=*"biz.korwin"*/>

<mongo:mongo id=*"mongo"* host=*"localhost"* port=*"27017"*/>

<bean id=*"mongoTemplate"* class=*"org.springframework.data.mongodb.core.MongoTemplate"*>

<constructor-arg ref=*"mongo"* />

<constructor-arg value=*"home"* />

</bean>

</beans>

And below is a standard **log4j.xml**:

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug=*"true"*

xmlns:log4j=*'http://jakarta.apache.org/log4j/'*>

<appender name=*"console"* class=*"org.apache.log4j.ConsoleAppender"*>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"*

value=*"%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n"* />

</layout>

</appender>

<root>

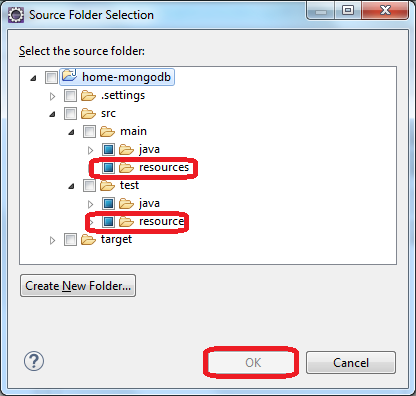
<level value=*"DEBUG"* />

<appender-ref ref=*"console"* />

</root>

</log4j:configuration>

Add the new folders to the project path:

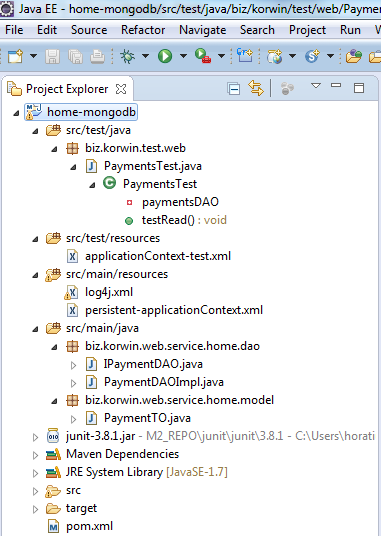
****

1. Do the build

*mvn clean install –DskipTests=true*

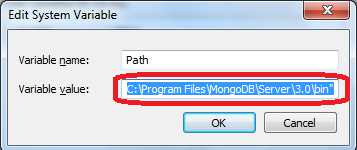
Now the jar file should be ready. After you install and populate some data into MongoDB database, you can run the Junit test from Eclipse and see that you can read some data from database.

At this point the structure of the project should look like this:



1. **Install the MongoDB locally and load the data**
2. Download and install the database. For details, see [**here**](http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/). I used msi file for Windows 7 installation. I needed to create **C:\data\db** folder.

You will need to add your MongoDb **bin** folder to the path. For me it was:



1. To start the server you open a console and type **mongod**.
2. To import our script into MongoDB commands you need to create in **home-mongodb** a folder called script. In this folder save as **payments.json** the following file:

[

{

"\_id":NumberInt(0),

"payee":"CIBC",

"payeeProduct":"PETRO-POINTS™ MasterCard",

"cycle": NumberInt(22),

"amountInCents" : NumberLong(123455),

"dueDate" : "20150330",

"paymentLimitDate" : "20150328",

"paidDate" : "",

"deprecated" : false

},

{

"\_id" : NumberInt(1),

"payee" : "RBC",

"payeeProduct" : "Visa Gold",

"cycle" : NumberInt(25),

"amountInCents" : NumberLong(44322),

"dueDate" : "20150328",

"paymentLimitDate" : "20150326",

"paidDate" : "",

"deprecated" : false

}

]

Open a console and go to **home-mongodb/scripts**. Type the following:

*mongoimport --jsonArray -d home -c payments -h localhost:27017 --type json --file payments.json*

This would import the payments.json file into “**home**” database and “**payments**” collection. You can now type **mongo** (this would open the MongoDB console), and you can type MongoDB JavaScript style commands. You can type:

*use home*

Now you are using the newly created **home** database.

When you are typing

*> db.payments.find().pretty()*

You will have something like this:

{

"\_id" : 1,

"payee" : "RBC",

"payeeProduct" : "Visa Gold",

"cycle" : 25,

"amountInCents" : NumberLong(44322),

"dueDate" : "20150328",

"paymentLimitDate" : "20150326",

"paidDate" : "",

"deprecated" : false

}

{

"\_id" : 0,

"payee" : "CIBC",

"payeeProduct" : "PETRO-POINTS? MasterCard",

"cycle" : 22,

"amountInCents" : NumberLong(123455),

"dueDate" : "20150330",

"paymentLimitDate" : "20150328",

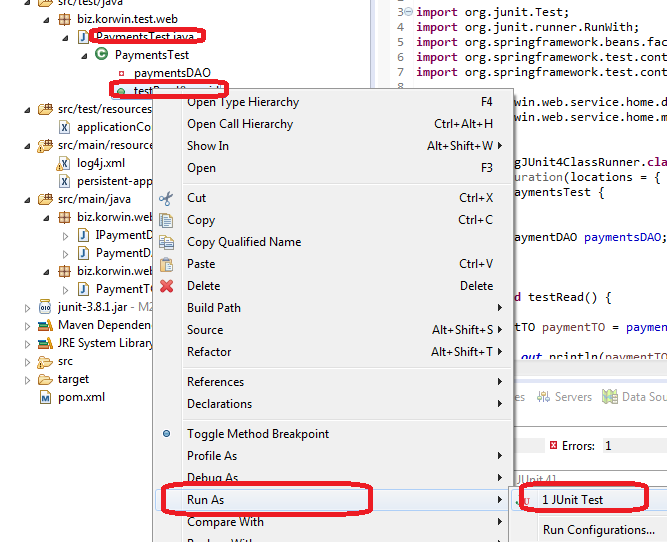
"paidDate" : "",

"deprecated" : false

}

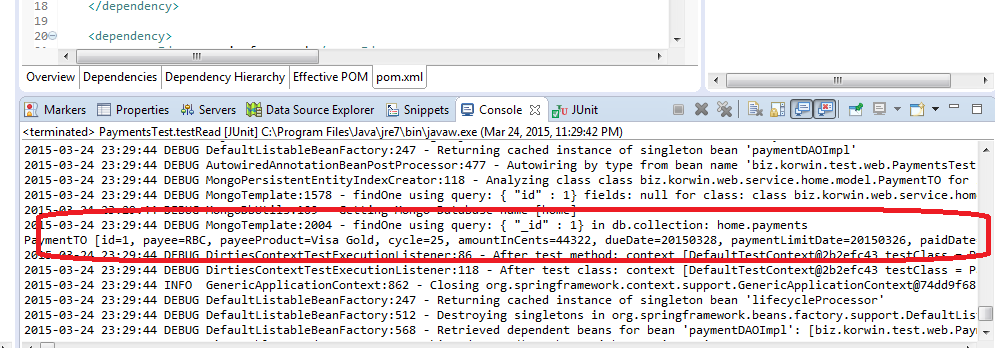
Now you can run the Junit “test” from Eclipse and see if you can retrieve one record (\_id=1).

Keep in mind that the MongoDB server should be started at the moment of importing or testing ☺.



If you will get **java.lang.NoClassDefFoundError: org/junit/runners/model/MultipleFailureException** you should do the following: right click on project in Package Explorer, go to Properties, go to Libraries tab, click on 'Add Library' button, select JUnit, click Next >. You should be able to handle it from there.

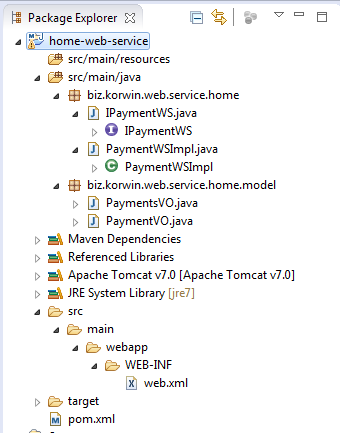
In Eclipse’s console you should see:



1. **Create the web service**

For this, you can follow the steps from [Integrating Apache CXF, JAX-RS, Spring (Part I)](http://horatiustanciu.blogspot.ca/2015/03/integratingapache-cxf-jax-rs-and-spring.html) or you can download the project from <https://github.com/hstanciu/CXF_JAXRS_Spring_Maven>

A couple of changes are needed. The structure of the project from Part I should be like this:



1. Java part

I restructured a little bit the project. I added a service layer that contains an interface and interface implementation (business delegate). This service layer will deal internally with database (DAO manipulation). The service layer is dealing with PaymentVO <->PaymentTO translation and calling the DAO methods:

**IPaymentService.java**

package biz.korwin.web.service.home.service;

import java.util.List;

import biz.korwin.web.service.home.model.PaymentVO;

/\*\*

\* Service interface for implementing the payment web service.

\* \*/

public interface IPaymentService {

/\* Returns a payment by payment ID. \*/

public PaymentVO getPaymentById(long id);

/\* Creates a payment. \*/

public PaymentVO create(PaymentVO newPayment);

/\* Deletes a payment. \*/

public boolean delete(long id);

/\* Updates a payment. \*/

public PaymentVO update(PaymentVO updatePayment);

/\* Returns the list of payments that have the same payment limit date. \*/

public List<PaymentVO> getPaymentsByDate(String date);

}

**PaymentServiceImpl.java**

package biz.korwin.web.service.home.service;

import java.util.ArrayList;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import biz.korwin.web.service.home.dao.IPaymentDAO;

import biz.korwin.web.service.home.model.PaymentTO;

import biz.korwin.web.service.home.model.PaymentVO;

/\*\*

\* Service implementation for the payment web service.

\* \*/

@Service

public class PaymentServiceImpl implements IPaymentService {

@Autowired

private IPaymentDAO paymentDAO;

@Override

public PaymentVO getPaymentById(long id) {

PaymentTO paymentTO = paymentDAO.getPaymentById(id);

PaymentVO paymentVO = new PaymentVO(paymentTO.getId(), paymentTO.getPayee(), paymentTO.getPayeeProduct(),

paymentTO.getCycle(), paymentTO.getAmountInCents(), paymentTO.getDueDate(),

paymentTO.getPaymentLimitDate(), paymentTO.getPaidDate(), paymentTO.isDeprecated());

return paymentVO;

}

@Override

public PaymentVO create(PaymentVO newPayment) {

PaymentTO paymentTO = new PaymentTO(newPayment.getPayee(), newPayment.getPayeeProduct(),

newPayment.getCycle(), newPayment.getAmountInCents(), newPayment.getDueDate(),

newPayment.getPaymentLimitDate(), newPayment.getPaidDate(), newPayment.isDeprecated());

paymentTO = paymentDAO.create(paymentTO);

PaymentVO paymentVO = new PaymentVO(paymentTO.getId(), paymentTO.getPayee(), paymentTO.getPayeeProduct(),

paymentTO.getCycle(), paymentTO.getAmountInCents(), paymentTO.getDueDate(),

paymentTO.getPaymentLimitDate(), paymentTO.getPaidDate(), paymentTO.isDeprecated());

return paymentVO;

}

@Override

public boolean delete(long id) {

return paymentDAO.delete(id);

}

@Override

public PaymentVO update(PaymentVO updatePayment) {

PaymentTO paymentTO = new PaymentTO(updatePayment.getId(), updatePayment.getPayee(), updatePayment.getPayeeProduct(),

updatePayment.getCycle(), updatePayment.getAmountInCents(), updatePayment.getDueDate(),

updatePayment.getPaymentLimitDate(), updatePayment.getPaidDate(), updatePayment.isDeprecated());

paymentTO = paymentDAO.update(paymentTO);

PaymentVO paymentVO = new PaymentVO(paymentTO.getId(), paymentTO.getPayee(), paymentTO.getPayeeProduct(),

paymentTO.getCycle(), paymentTO.getAmountInCents(), paymentTO.getDueDate(),

paymentTO.getPaymentLimitDate(), paymentTO.getPaidDate(), paymentTO.isDeprecated());

return paymentVO;

}

@Override

public List<PaymentVO> getPaymentsByDate(String date) {

// TODO Auto-generated method stub

List<PaymentTO> paymentsTOList = paymentDAO.getPaymentsByDate(date);

List<PaymentVO> listOfPayments = new ArrayList<PaymentVO>();

if(null != paymentsTOList){

for(int i = 0; i < paymentsTOList.size(); i++){

PaymentTO paymentTO = paymentsTOList.get(i);

if(null != paymentTO){

PaymentVO paymentVO = new PaymentVO(paymentTO.getId(), paymentTO.getPayee(), paymentTO.getPayeeProduct(),

paymentTO.getCycle(), paymentTO.getAmountInCents(), paymentTO.getDueDate(),

paymentTO.getPaymentLimitDate(), paymentTO.getPaidDate(), paymentTO.isDeprecated());

listOfPayments.add(paymentVO);

}

}

}

return listOfPayments;

}

}

PaymentsVO.java would stay unchanged and PaymentVO.java will have some constructors added.

**PaymentVO.java**

package biz.korwin.web.service.home.model;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* Model class. Represents a payment.

\*

\* \*/

@XmlRootElement(name="payment")

@XmlAccessorType(XmlAccessType.FIELD)

public class PaymentVO {

/\* Payment ID\*/

private long id;

/\* Who will be paid; ex: Canada Bank\*/

private String payee;

/\* The payment product; ex: Master Card Petro-Points \*/

private String payeeProduct;

/\* The number of days a new statement will be issued, or -1 if is not cyclical\*/

private int cycle;

/\* Amount to be paid (in cents) \*/

private int amountInCents;

/\* Due date \*/

private String dueDate;

/\* In order for payment to occur at due date, the payment should occur

\* in advance with a couple of days. This date is the limit date when

\* this occurs in order for due date not to be passed.\*/

private String paymentLimitDate;

/\* Date when payment happened. If it didn't happen yet,

\* then this attribute will be null.\*/

private String paidDate;

/\* If true, the payment is not actual.\*/

private boolean deprecated;

public PaymentVO(){}

public PaymentVO(String payee,

String payeeProduct,

int cycle,

int amountInCents,

String dueDate,

String paymentLimitDate,

String paidDate,

boolean deprecated){

this.payee = payee;

this.payeeProduct = payeeProduct;

this.cycle = cycle;

this.amountInCents = amountInCents;

this.dueDate = dueDate;

this.paymentLimitDate = paymentLimitDate;

this.paidDate = paidDate;

this.deprecated = deprecated;

}

public PaymentVO(long id,

String payee,

String payeeProduct,

int cycle,

int amountInCents,

String dueDate,

String paymentLimitDate,

String paidDate,

boolean deprecated){

this.id = id;

this.payee = payee;

this.payeeProduct = payeeProduct;

this.cycle = cycle;

this.amountInCents = amountInCents;

this.dueDate = dueDate;

this.paymentLimitDate = paymentLimitDate;

this.paidDate = paidDate;

this.deprecated = deprecated;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getPayee() {

return payee;

}

public void setPayee(String payee) {

this.payee = payee;

}

public String getPayeeProduct() {

return payeeProduct;

}

public void setPayeeProduct(String payeeProduct) {

this.payeeProduct = payeeProduct;

}

public int getCycle() {

return cycle;

}

public void setCycle(int cycle) {

this.cycle = cycle;

}

public int getAmountInCents() {

return amountInCents;

}

public void setAmountInCents(int amountInCents) {

this.amountInCents = amountInCents;

}

public String getDueDate() {

return dueDate;

}

public void setDueDate(String dueDate) {

this.dueDate = dueDate;

}

public String getPaymentLimitDate() {

return paymentLimitDate;

}

public void setPaymentLimitDate(String paymentLimitDate) {

this.paymentLimitDate = paymentLimitDate;

}

public String getPaidDate() {

return paidDate;

}

public void setPaidDate(String paidDate) {

this.paidDate = paidDate;

}

public boolean isDeprecated() {

return deprecated;

}

public void setDeprecated(boolean deprecated) {

this.deprecated = deprecated;

}

}

There is small change from Part I for **IPaymentWS.java**

package biz.korwin.web.service.home;

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.core.MediaType;

import javax.ws.rs.core.Response;

import biz.korwin.web.service.home.model.PaymentVO;

import biz.korwin.web.service.home.model.PaymentsVO;

@Path("/payments")

public interface IPaymentWS {

/\* Returns the list of payments that have the same payment limit date. \*/

@GET

@Consumes({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

@Produces({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

@Path("/date/{date}")

public PaymentsVO getPaymentsByDate(@PathParam("date") String date);

/\* Returns a payment by payment ID. \*/

@GET

@Consumes({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

@Produces({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

@Path("/{id}")

public PaymentVO getPaymentById(@PathParam("id") long id);

/\* Creates a payment. \*/

@POST

@Consumes({MediaType.APPLICATION\_FORM\_URLENCODED})

@Produces({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

public PaymentVO create(@FormParam("payee")String payee,

@FormParam("payeeProduct")String payeeProduct,

@FormParam("cycle")int cycle,

@FormParam("amountInCents")int amountInCents,

@FormParam("dueDate")String dueDate,

@FormParam("paymentLimitDate")String paymentLimitDate,

@FormParam("deprecated")boolean deprecated);

/\* Deletes a payment. \*/

@DELETE

@Path("/{id}")

@Produces({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

public Response delete(@PathParam("id") long id);

/\* Updates a payment. \*/

@PUT

@Path("/{id}")

@Produces({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML})

public PaymentVO update(@PathParam("id") long id,

@FormParam("payee")String payee,

@FormParam("payeeProduct")String payeeProduct,

@FormParam("cycle")int cycle,

@FormParam("amountInCents")int amountInCents,

@FormParam("dueDate")String dueDate,

@FormParam("paymentLimitDate")String paymentLimitDate,

@FormParam("deprecated")boolean deprecated);

}

and the implementing class looks like this; the new thing is the injection of service. Now the web service is not dummy, but it connects through business delegate to DAO.

**PaymentWSImpl.java**

package biz.korwin.web.service.home;

import java.text.SimpleDateFormat;

import java.util.List;

import javax.ws.rs.core.Response;

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

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import biz.korwin.web.service.home.model.PaymentVO;

import biz.korwin.web.service.home.model.PaymentsVO;

import biz.korwin.web.service.home.service.IPaymentService;

/\*\*

\* Web service implementation.

\* \*/

@Component("paymentsService")

public class PaymentWSImpl implements IPaymentWS {

SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMdd");

@Autowired

IPaymentService paymentService;

@Override

public PaymentsVO getPaymentsByDate(String date) {

List<PaymentVO> listOfPayments = paymentService.getPaymentsByDate(date);

PaymentsVO payments = new PaymentsVO();

payments.setPayments(listOfPayments);

return payments;

}

@Override

public Response delete(long id) {

if(paymentService.delete(id)){

return Response.status(Status.OK).build();

}else{

return Response.status(Status.EXPECTATION\_FAILED).build();

}

}

@Override

public PaymentVO update(long id, String payee,

String payeeProduct,

int cycle,

int amountInCents,

String dueDate,

String paymentLimitDate,

boolean deprecated) {

PaymentVO updatePayment = new PaymentVO();

updatePayment.setId(id);

updatePayment.setPayee(payee);

updatePayment.setPayeeProduct(payeeProduct);

updatePayment.setCycle(cycle);

updatePayment.setAmountInCents(amountInCents);

updatePayment.setDueDate(dueDate);

updatePayment.setPaymentLimitDate(paymentLimitDate);

//updatePayment.setPaidDate(paidDate);

updatePayment.setDeprecated(deprecated);

PaymentVO paymentUpdated = paymentService.update(updatePayment);

return paymentUpdated;

}

@Override

public PaymentVO getPaymentById(long id) {

PaymentVO payment = paymentService.getPaymentById(id);

return payment;

}

@Override

public PaymentVO create(String payee, String payeeProduct,

int cycle, int amountInCents, String dueDate,

String paymentLimitDate, boolean deprecated) {

PaymentVO payment = new PaymentVO();

payment.setAmountInCents(amountInCents);

payment.setCycle(cycle);

payment.setDueDate(dueDate);

payment.setPaidDate(null);

payment.setPayee(payee);

payment.setPayeeProduct(payeeProduct);

payment.setPaymentLimitDate(paymentLimitDate);

PaymentVO createdPayment = paymentService.create(payment);

return createdPayment;

}

}

1. Configuration files

For configuration, we have two XML files in WEB-INF folder: **applicationContext.xml** and **web.xml**. Let’s take one by one:

**applicationContext.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:jaxrs=*"http://cxf.apache.org/jaxrs"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd"*>

<import resource=*"classpath:META-INF/cxf/cxf.xml"*/>

<import resource=*"classpath:META-INF/cxf/cxf-servlet.xml"*/>

<context:component-scan base-package=*"biz.korwin"*/>

<jaxrs:server id=*"restfulId"* address=*"/"*>

<jaxrs:serviceBeans>

<ref bean=*"paymentsService"*/>

</jaxrs:serviceBeans>

<jaxrs:providers>

<bean id=*"jacksonProvider"* class=*"org.codehaus.jackson.jaxrs.JacksonJsonProvider"*/>

</jaxrs:providers>

</jaxrs:server>

<import resource=*"classpath\*:\*\*/persistent-applicationContext.xml"*/>

</beans>

The only change from **Part I** is that we are including the **persistence-applicationContext.xml** that is needed by the **home-mongodb** project.

**NOTE:** Make sure you don’t name the Spring application context file the same way. If you have more than one file, you should have appropriate and different names.

**web.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app 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"*

version=*"2.5"*>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>CXF Servlet</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

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

</servlet>

<servlet-mapping>

<servlet-name>CXF Servlet</servlet-name>

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

</servlet-mapping>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/applicationContext.xml</param-value>

</context-param>

</web-app>

Web.xml does not have big change.

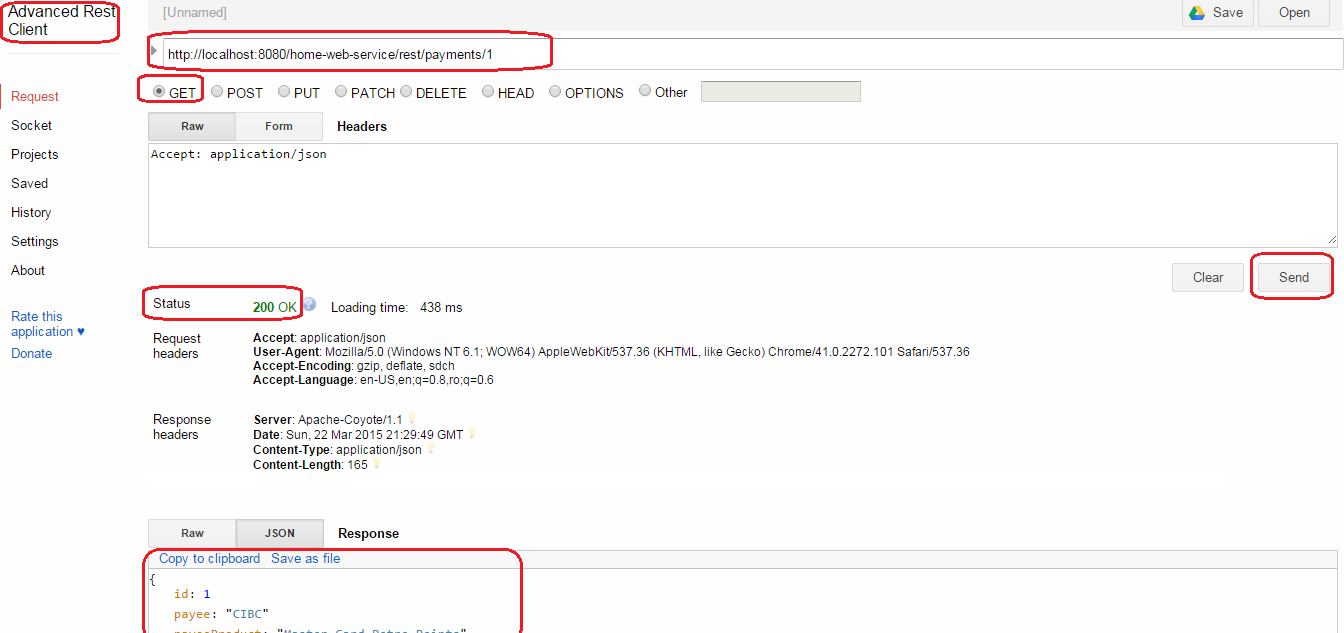
1. **Deploy and use the service**

In order to build and package the web service go to **C:\java\projects\webServiceWorkspace** and type

*mvn clean package*

This will create in **target** folder home-web-service.war file. This file will have the home-mongodb.jar included. Take this file and drop it into Tomcat’s **webapps** folder. Restart Tomcat.

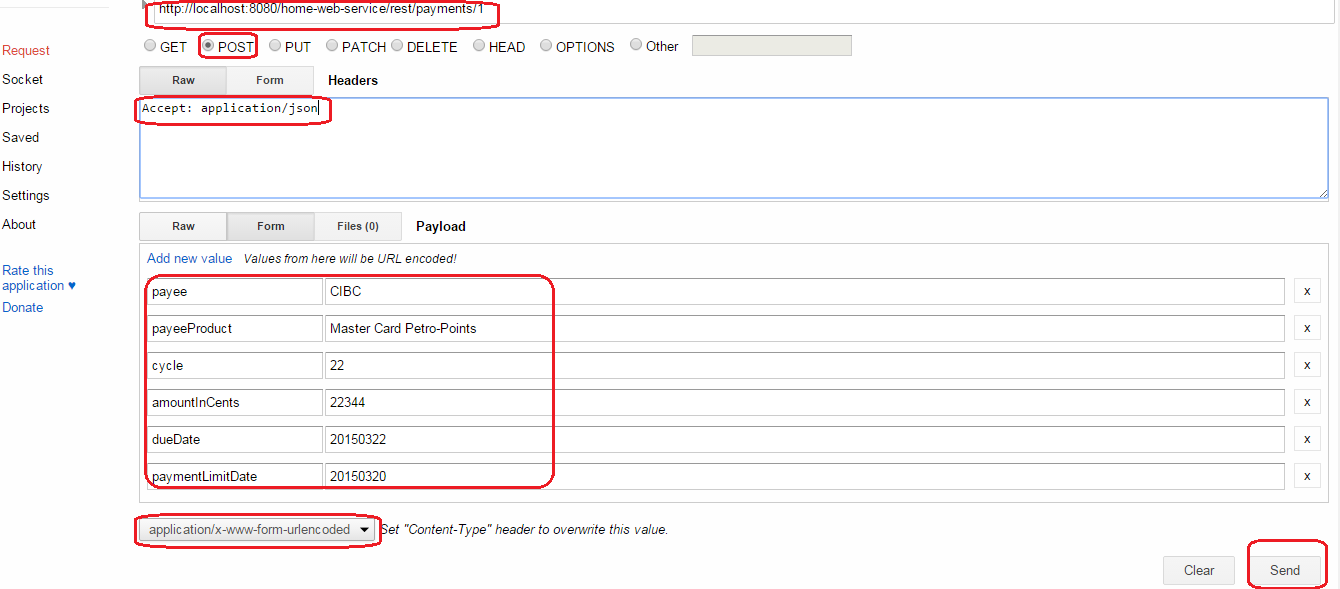
For testing I used Google’s Chrome and I installed a Google application called Advanced Rest Client. Any REST client that allows you to send different types of HTTP requests, would do it.



This is a GET request that will trigger **IPaymentsWS**’s

public PaymentVO getPaymentById(@PathParam("id") long id);

Similar with this you can make PUT and DELETE calls. For POST, you can call the web service this way:



**NOTE:**

The project is on <https://github.com/hstanciu/CXF_JAXRS_Spring_Maven_MongoDB>