

## ***“Web Services con REST”***

### **Módulo 7 / 2**

© *Todos los logos y marcas utilizados en este documento, están registrados y pertenecen a sus respectivos dueños.*

## Objetivo

En los últimos años, **REST** se ha convertido en una excelente alternativa frente a las arquitecturas tradicionales como SOAP/WSDL/WS. Spring MVC provee un excelente soporte e integración con la tecnología REST.

**REST** es la abreviación de **R**epresentational **S**tate **T**ransfer, a diferencia de SOAP el cual establece las reglas de comunicación tanto de entrada como de salida de datos, en **REST** las reglas son sólo de entrada. Mientras que SOAP se basa en estándares bien definidos de comunicación, REST depende directamente del protocolo HTTP.

**REST Web Services** utiliza especificación en formato XML, JSON básicamente. Estas normas ad hoc significa que la forma para acceder a un servicio web REST es diferente para cada servicio. Los servicios web REST típicamente utilizan parámetros de URL (GET) o la información de la ruta o path para solicitar los datos y utiliza POST para enviar datos al servicio web.

El objetivo de este laboratorio es comprender e implementar servicios con REST, veremos algunos ejemplos con JSON, Feed RSS y Atom.

*"Quemar etapas"*

*Es importante que saques provecho de cada módulo y consultes todos los temas que se van tratando, sin adelantar etapas.*

## Introducción

Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura — descritos más abajo, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utiliza XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST de Fielding y también es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto (RPC), pero sin usar SOAP. Estos dos usos diferentes del término REST causan cierta confusión en las discusiones técnicas, aunque RPC no es un ejemplo de REST.

Los sistemas que siguen los principios REST se llaman con frecuencia RESTful.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)
- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (ABMC en castellano: Alta, Baja, Modificación y Consulta) que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

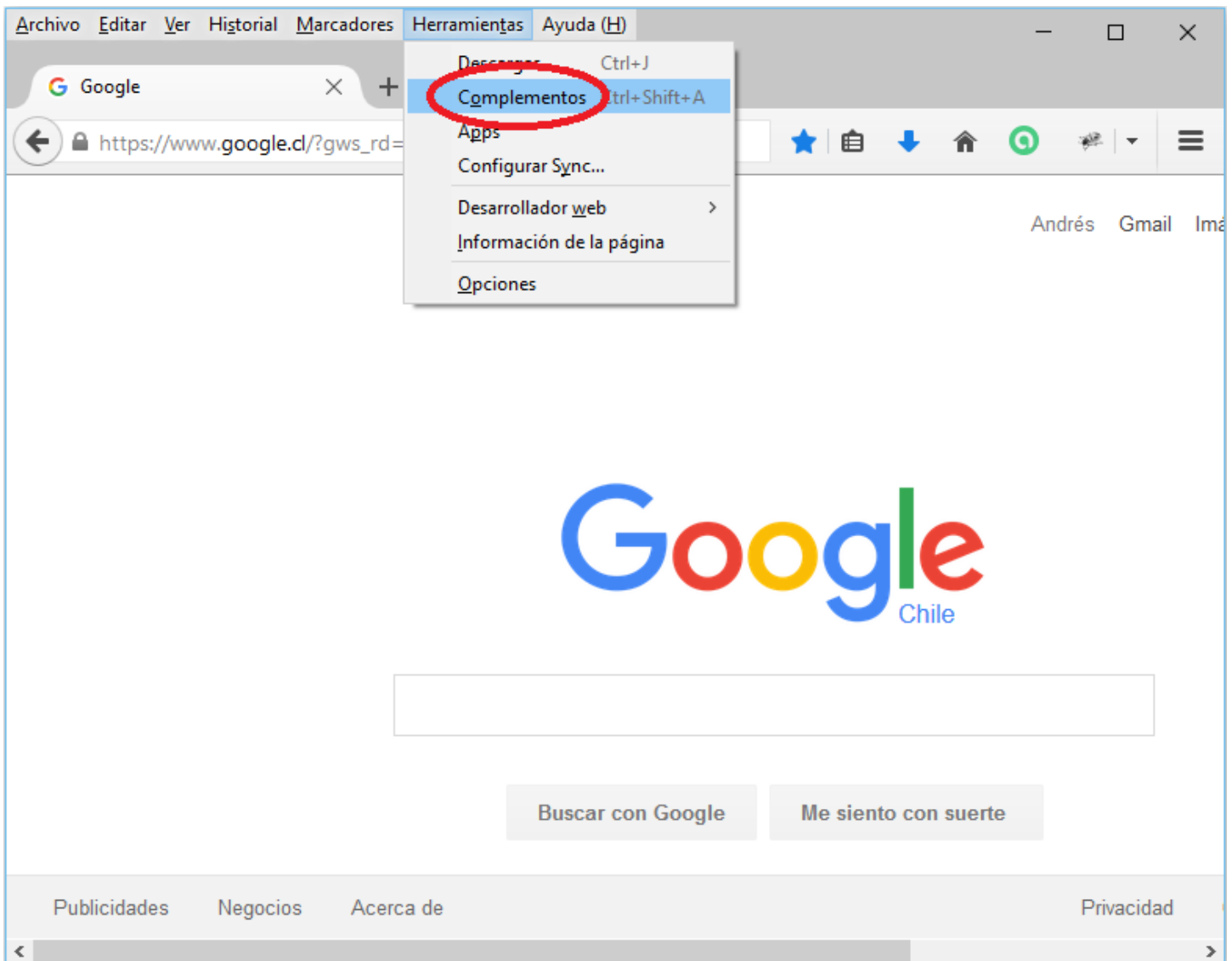
- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

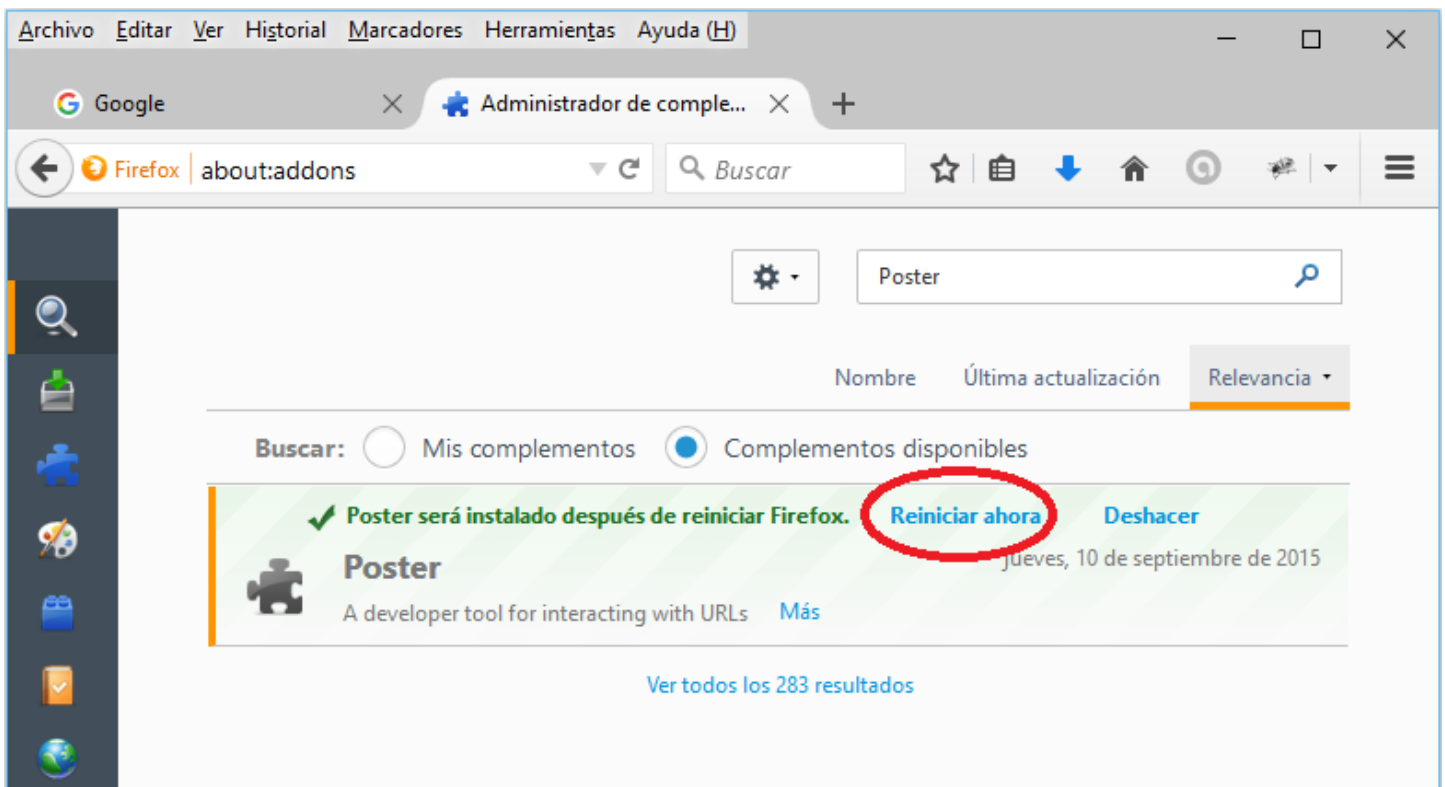
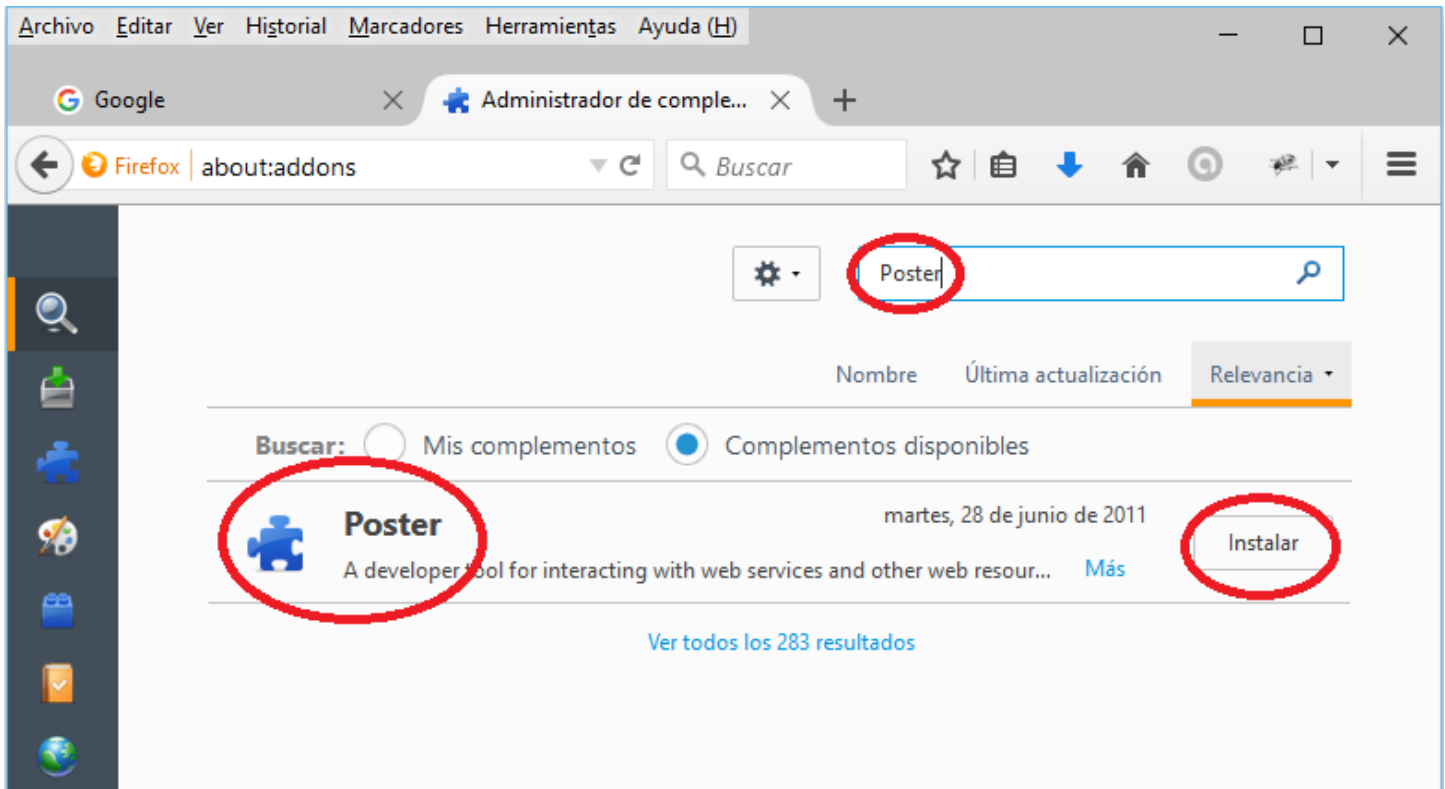
## Ejercicio 1: Generar y ejecutar el ejemplo "Un simple REST con JSON"

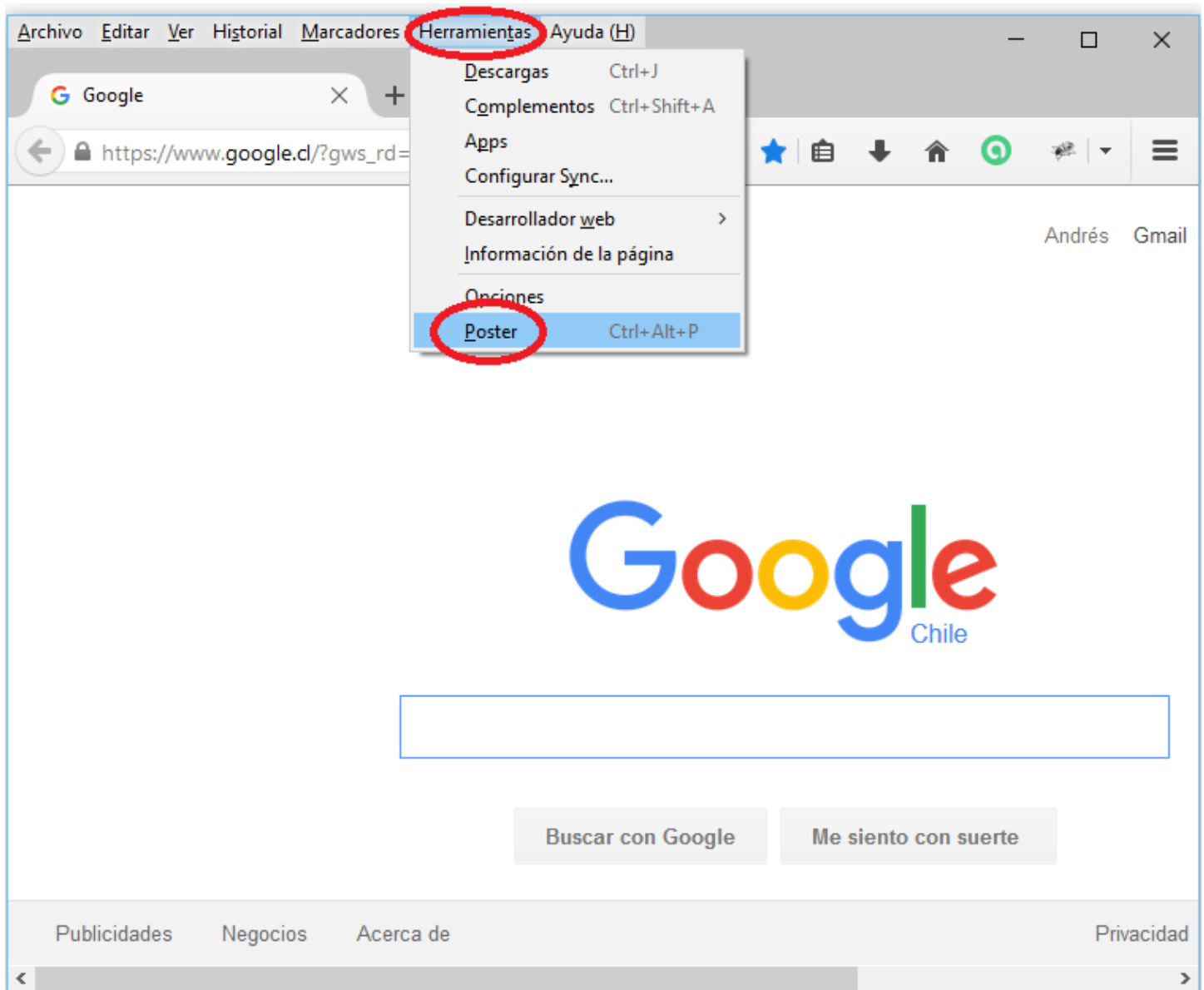
En el ejemplo veremos la configuración del **spring content negotiation** para que soporte el formato JSON en la respuesta, además:

- Aprenderemos a configurar como vista JSON la clase de Spring `org.springframework.web.servlet.view.json.MappingJacksonJsonView`
- Instalaremos el plug-in de Firefox Poster, como cliente REST para las pruebas
- También veremos `RestTemplate` como una clase cliente REST

### 1. Instalar el plug-in Poster en Firefox








## 2. Ejecutamos el proyecto

- Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
- Clic derecho sobre el proyecto y **Maven->Update Project...**
- Ejecutamos la aplicación: clic derecho sobre **mvc\_rest\_json-> Run As on Server**
- Observe que la respuesta JSON se despliega en pantalla



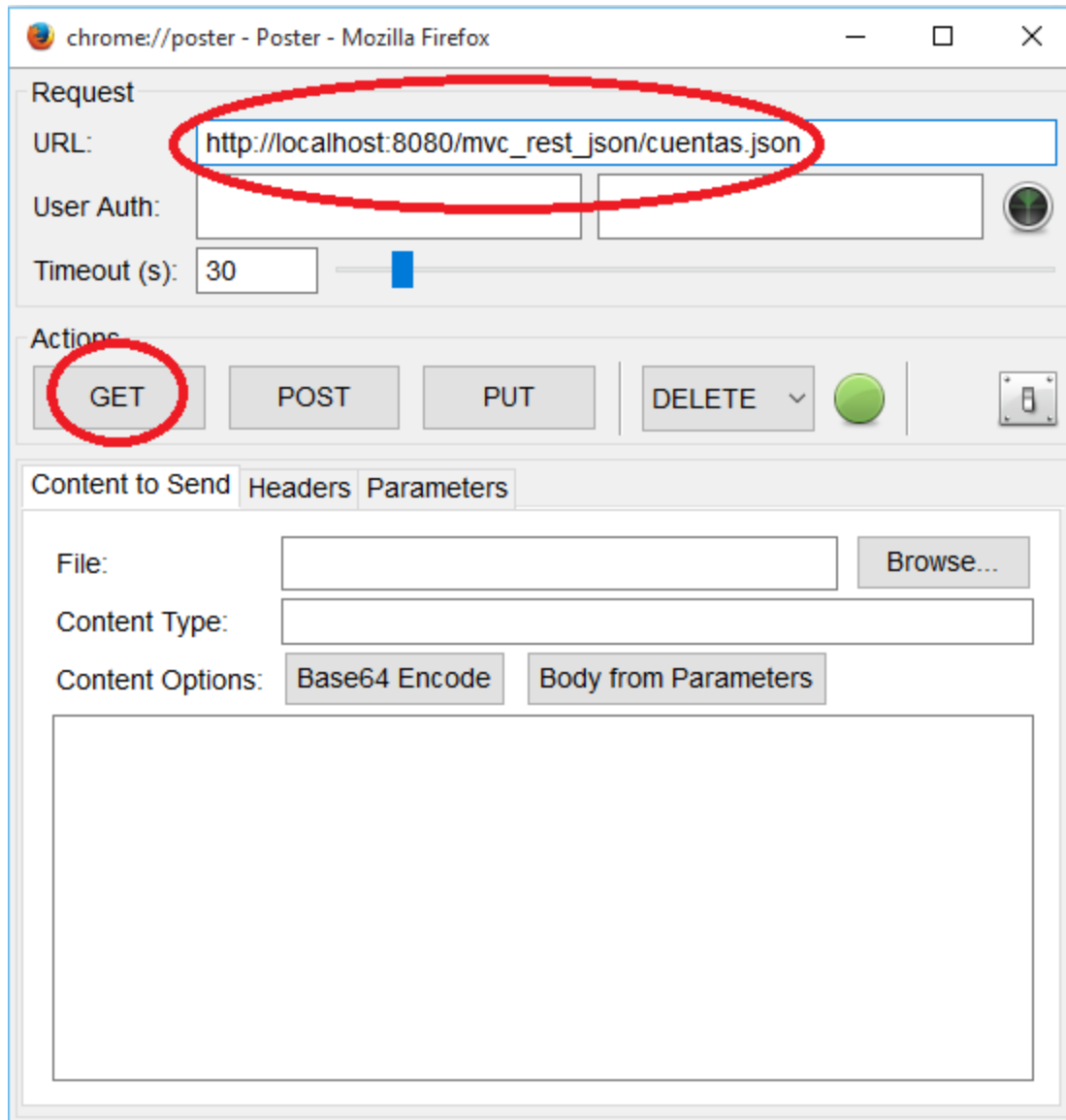
The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/mvc_rest_json/`. The browser's developer tools or response view shows a JSON array of three account objects. The JSON data is as follows:

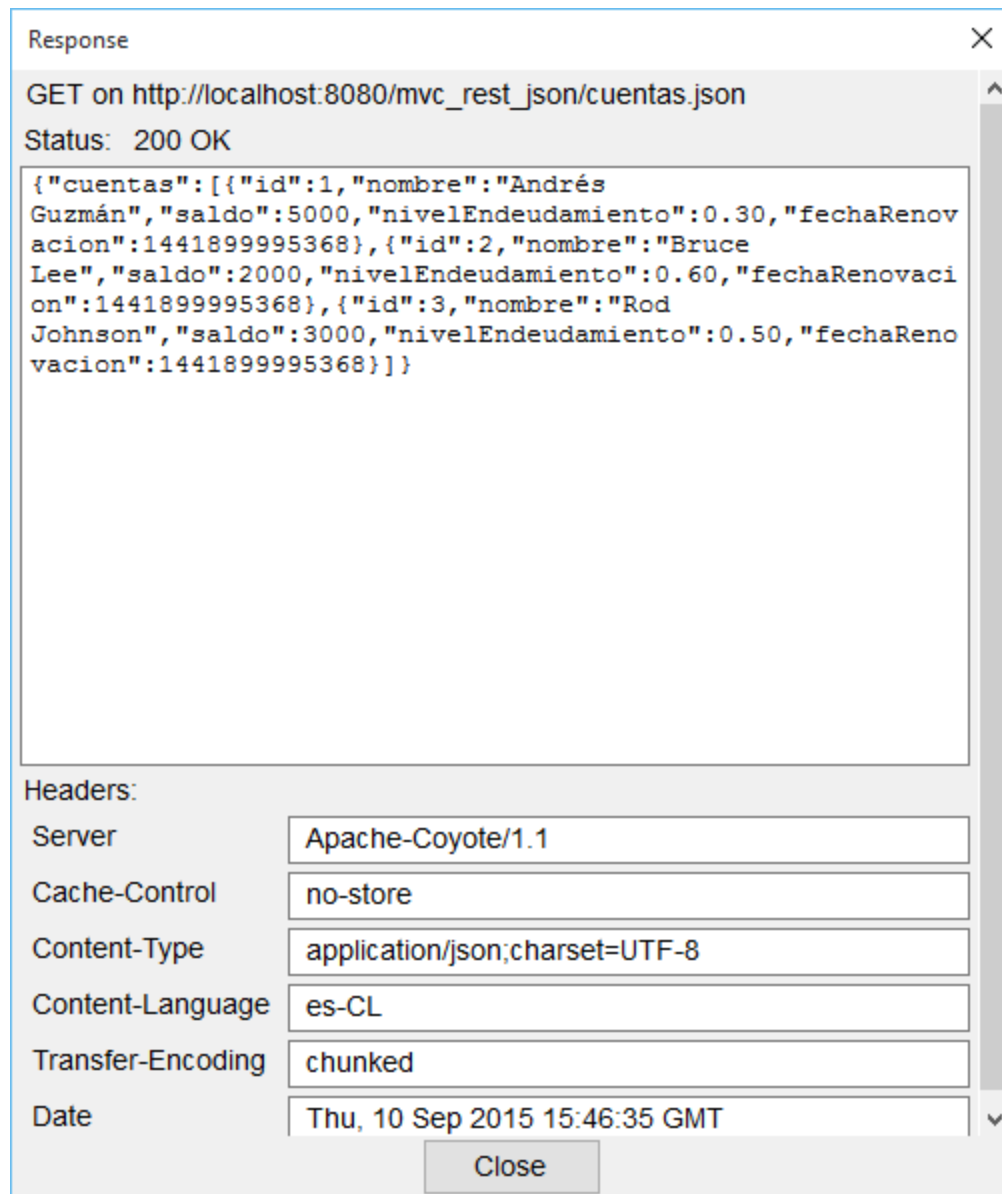
```
{
  "cuentas": [
    {
      "id": 1,
      "nombre": "Andrés Guzmán",
      "saldo": 5000,
      "nivelEndeudamiento": 0.30,
      "fechaRenovacion": 1441899863014
    },
    {
      "id": 2,
      "nombre": "Bruce Lee",
      "saldo": 2000,
      "nivelEndeudamiento": 0.60,
      "fechaRenovacion": 1441899863014
    },
    {
      "id": 3,
      "nombre": "Rod Johnson",
      "saldo": 3000,
      "nivelEndeudamiento": 0.50,
      "fechaRenovacion": 1441899863014
    }
  ]
}
```



## 3. Accedemos al servicio a través de Poster

- Para el campo URL, ingresamos `http://localhost:8080/mvc_rest_json/cuentas.json`
- Clic GET





4. Abrir y estudiar la clase controladora **CuentaController****/src/main/java/com/bolsadeideas/ejemplos/cuenta/controllers/CuentaController.java**

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value="/cuentas")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    @RequestMapping(method=RequestMethod.GET)
    public String getAccounts(Model model) {
        // Agregamos algunos datos de ejemplo
        Cuenta account = new Cuenta( new Long(0001), "Andres Guzmán",
            new BigDecimal("5000"), new BigDecimal("0.30"), new Date());
        cuentas.put(new Long(0001), account);

        account = new Cuenta( new Long(0002), "Bruce Lee",
            new BigDecimal("2000"), new BigDecimal("0.60"), new Date());
        cuentas.put(new Long(0002), account);

        account = new Cuenta( new Long(0003), "Rod Johnson",
            new BigDecimal("3000"), new BigDecimal("0.50"), new Date());
        cuentas.put(new Long(0003), account);

        // Agregamos el map de cuentas al atributo del model "cuentas",
        // para desplegarlos en el reporte de la vista json
        model.addAttribute("cuentas", new ArrayList<Cuenta>(cuentas.values()));
        return "cuentas";
    }

    @RequestMapping(value="{id}", method=RequestMethod.GET)
    public String getView(@PathVariable Long id, Model model) {
        Cuenta cuenta = this.cuentas.get(id);
        model.addAttribute(cuenta);
        return "cuenta";
    }
}
```

## 5. Abrir y estudiar el archivo de configuración del contexto de Spring: servlet-context.xml. /src/main/webapp/WEB-INF/spring/appServlet/servlet-context.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:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd
           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.xsd">

    <!-- Escanea o busca en el package base de la aplicación clases beans anotados
         con @Components, @Controller, @Service -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos.cuenta.controllers" />

    <!-- Habilita la anotacion de Spring MVC @Controller -->
    <mvc:annotation-driven />

    <!-- Views configuradas y mapeadas en un bean, ej: id="cuentas" (clases PDF, XLS, etc) -->
    <bean id="contentNegotiatingResolver"
          class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
        <property name="order"
            value="#{T(org.springframework.core.Ordered).HIGHEST_PRECEDENCE}" />
    </bean>

    <!-- BeanNameViewResolver -->
    <bean id="beanNameViewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver">
        <property name="order" value="#{contentNegotiatingResolver.order+1}" />
    </bean>

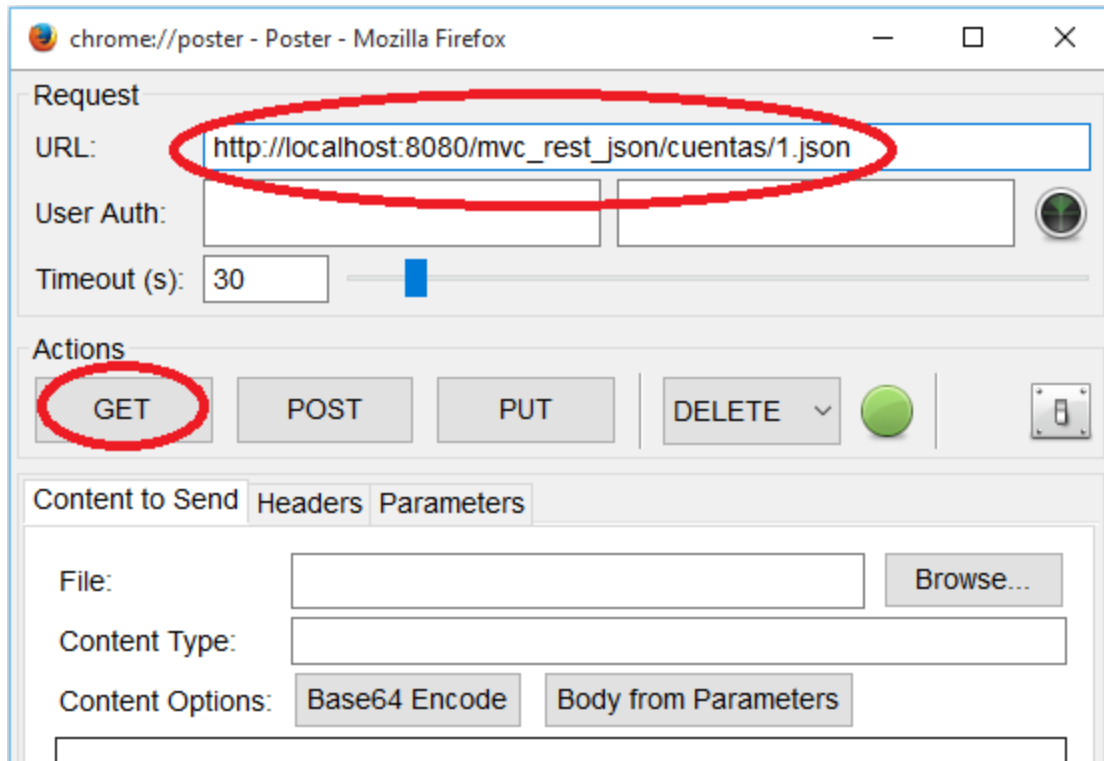
    <!-- La vista "cuentas" manejada por la clase "MappingJackson2JsonView" -->
    <bean id="cuentas"
          class="org.springframework.web.servlet.view.json.MappingJackson2JsonView" />

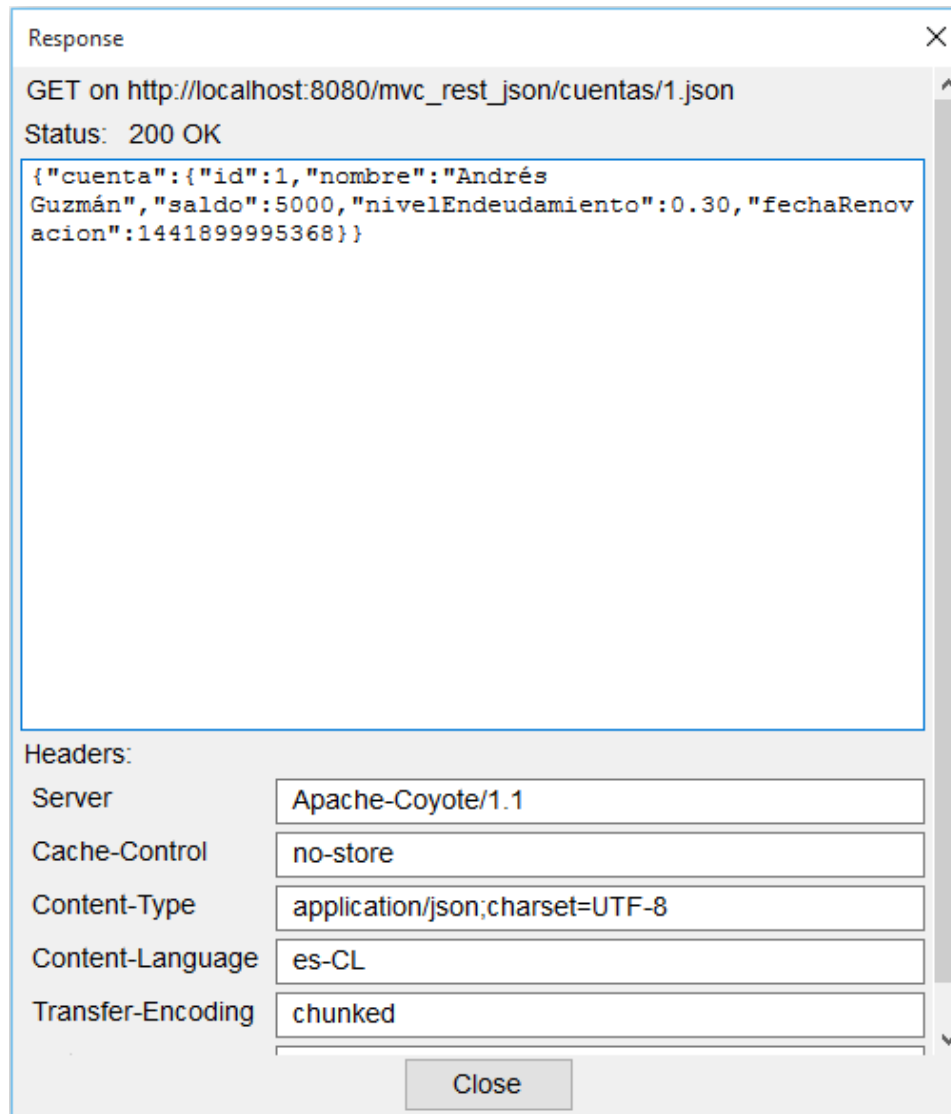
    <!-- La vista "cuenta" manejada por la clase "MappingJackson2JsonView" -->
    <bean id="cuenta"
          class="org.springframework.web.servlet.view.json.MappingJackson2JsonView" />

    <!-- Resuelve la ubicacion de las vistas .jsp de @Controllers en la ruta /WEB-INF/views -->
    <bean id="internalResourceResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
        <property name="order" value="#{beanNameViewResolver.order+1}" />
    </bean>
</beans>
```

6. Accediendo a una cuenta individual, por el id.

- Accedemos a la Uri [http://localhost:8080/mvc\\_rest\\_json/cuentas/1.json](http://localhost:8080/mvc_rest_json/cuentas/1.json)





- El resultado se despliega y muestra en formato JSON. Esto es correcto, justamente lo que se esperaba ya que retornará el nombre de la vista "cuenta", que es resuelto por el View **MappingJackson2JsonView**.

---

```
<!-- La vista "cuenta" manejada por la clase "MappingJackson2JsonView" -->
<bean id="cuenta"
      class="org.springframework.web.servlet.view.json.MappingJackson2JsonView" />
```

---

- Si no estuviera configurada la vista con Json (**MappingJackson2JsonView**), tomaría el formato por defecto JSP configurado con el View Resolver **InternalResourceViewResolver**.

7. Otro punto importante a tener en cuenta, es tener las dependencias **maven** de **MappingJackson2JsonView** en el pom.xml:

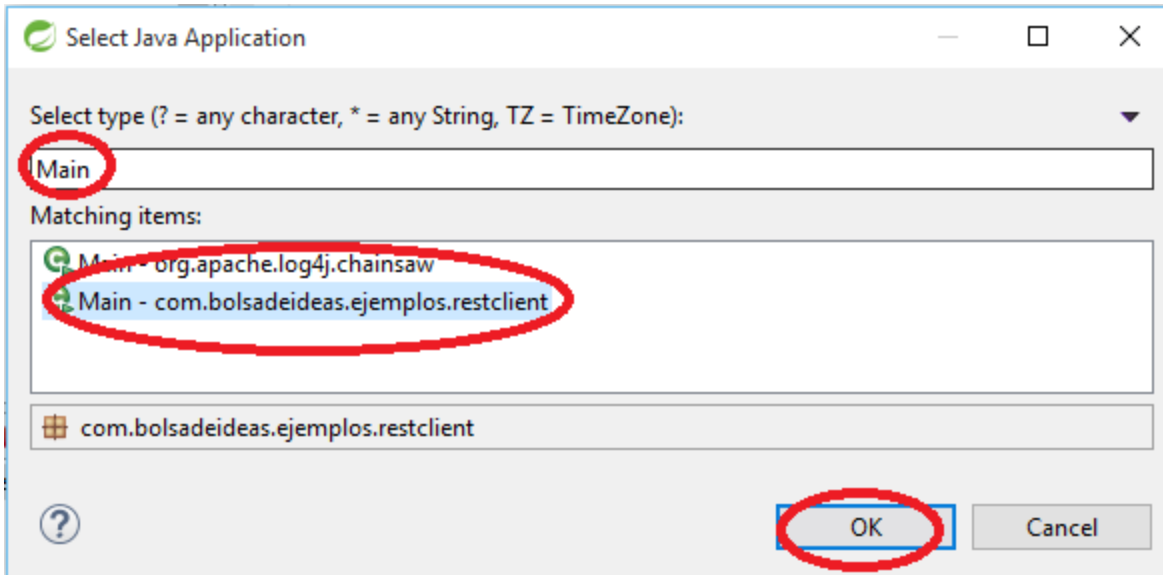
ETC...

```
<!-- Jackson JSON Mapper -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.6.1</version>
</dependency>
```

ETC...

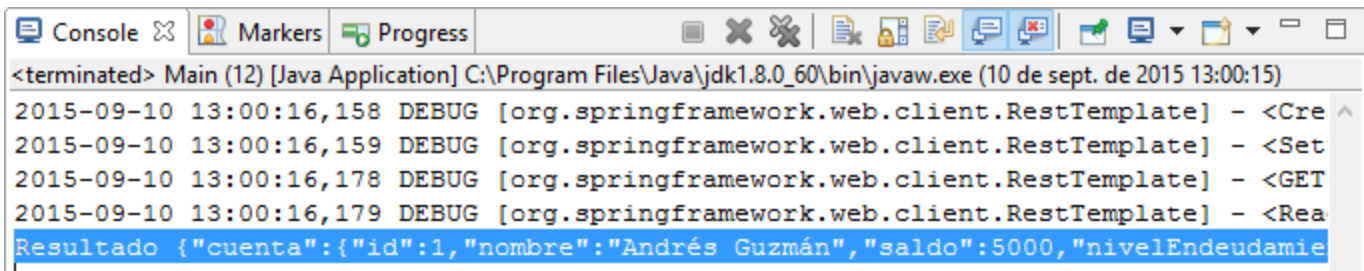
---

## 8. Generar y ejecutar el ejemplo cliente "rest\_resttemplate\_json"



- Observe el resultado en la consola:

Resultado {"cuenta":{"id":1,"nombre":"Andrés Guzmán","saldo":5000,"nivelEndeudamiento":0.30,"fechaRenovacion":1441899995368}}





- Abrir y estudiar clase Main

---

```
package com.bolsadeideas.ejemplos.restclient;

import java.util.HashMap;
import java.util.Map;

import org.springframework.web.client.RestTemplate;

public class Main {

    private static RestTemplate restTemplate = new RestTemplate();

    public static void main(String[] args) {

        String testURL = "http://localhost:8080/mvc_rest_json/cuentas/{id}.json";

        Map<String, String> variables = new HashMap<String, String>(1);
        variables.put("id", "1");
        String jsonResult = restTemplate.getForObject(testURL,
            String.class, // tipo respuesta
            variables);    // variables para el URI template
        System.out.println("Resultado " + jsonResult);

    }

}
```

---

### Ejercicio 3: Generar y ejecutar el ejemplo "REST con Feed RSS"



RSS son las siglas de **Really Simple Syndication**, un formato XML para syndicar o compartir contenido en la web. Se utiliza para difundir información actualizada frecuentemente a usuarios que se han suscrito a la fuente de contenidos. El formato permite distribuir contenidos sin necesidad de un navegador, utilizando un software diseñado para leer estos contenidos RSS. A pesar de eso, es posible utilizar el mismo navegador para ver los contenidos RSS.

Las últimas versiones de los principales navegadores permiten leer los RSS sin necesidad de software adicional. RSS es parte de la familia de los formatos XML desarrollado específicamente para todo tipo de sitios que se actualicen con frecuencia y por medio del cual se puede compartir la información y usarla en otros sitios web o programas.

Por poner un ejemplo sencillo: Igual que HTML sirve para escribir páginas en un formato entendible por los navegadores, RSS sirve para enumerar artículos o páginas dentro de un sitio, en un formato que pueden entender programas denominados lectores RSS.

Tenemos algunos elementos obligatorios en un feed, que es parte de un estándar. Cada canal tiene que tener un título, descripción, enlace y elementos de descripción.

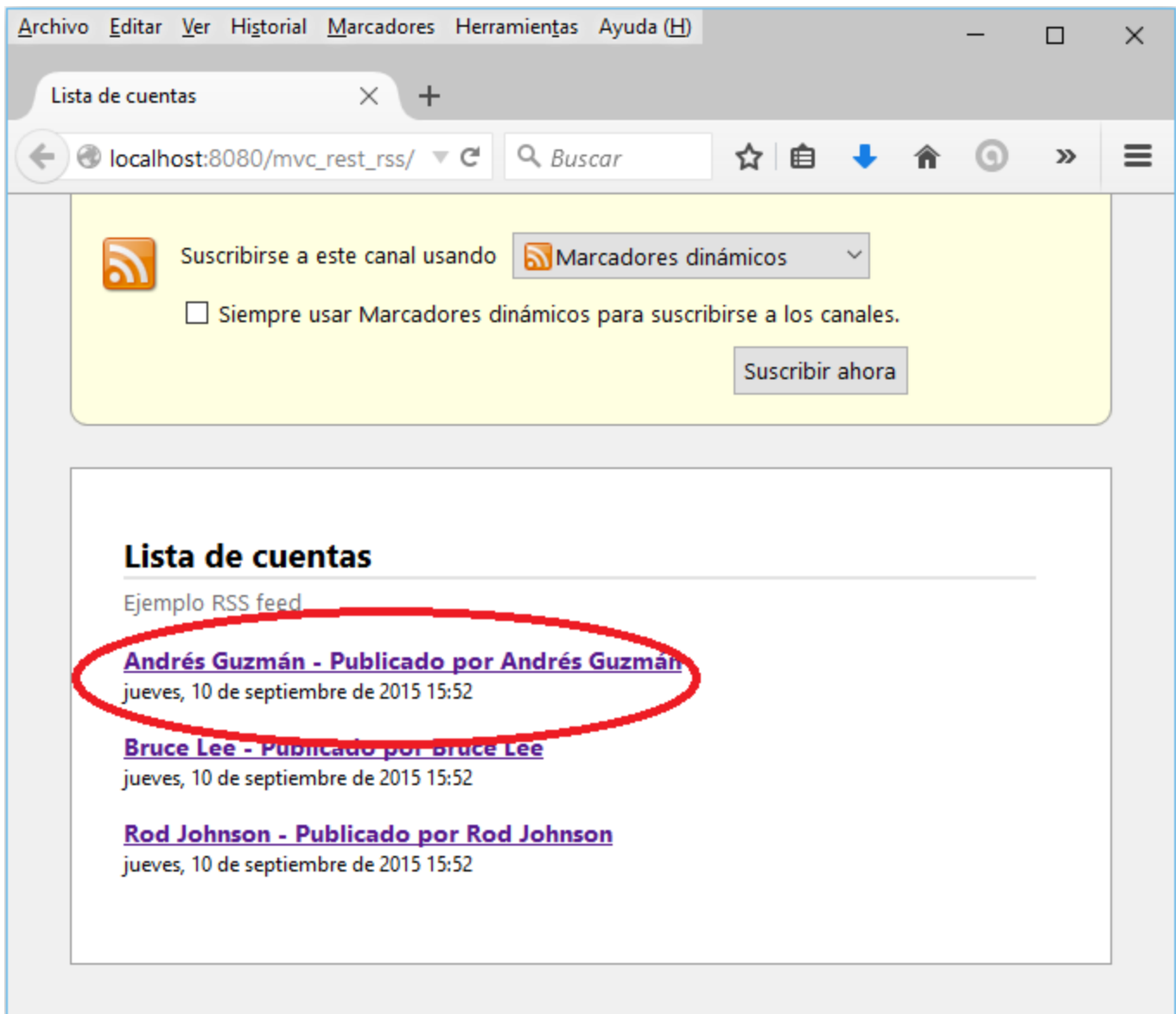
**En el archivo RSS simplemente están los datos de las novedades del sitio**, como el título, fecha de publicación o la descripción. El programa que lea el RSS será encargado de darle estilo o apariencia a los datos que se incluyan en el archivo y presentarlos de una manera atractiva al usuario y de fácil lectura.

Que RSS sea un formato basado en XML significa que el archivo RSS se compone por una serie de etiquetas definidas que tendrán un formato dado, que respetará las reglas generales de XML

En el ejemplo veremos la configuración del **spring content negotiation** para que en la respuesta soporte el formato Feed RSS, además aprenderemos a configurar la vista RSS feed usando la clase `AbstractRssFeedView`.

1. Ejecutamos el proyecto

- Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
- Clic derecho sobre el proyecto y **Maven->Update Project...**
- Ejecutamos la aplicación: clic derecho sobre **mvc\_rest\_rss-> Run As on Server**
- Observe que la respuesta en el browser:



2. Clic en una cuenta para ver el contenido original en XHTML:



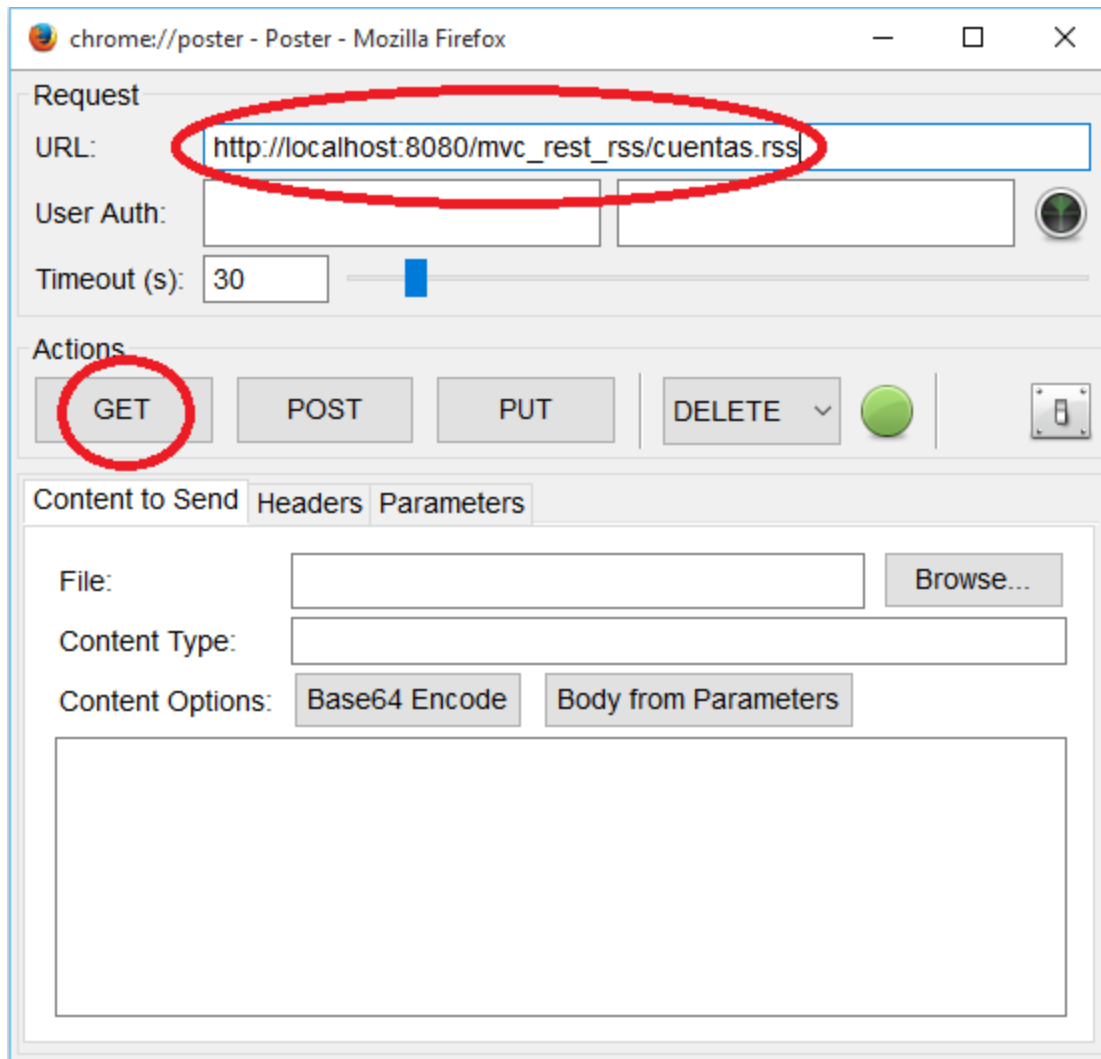
The screenshot shows a web browser window with the address bar displaying 'localhost:8080/mvc\_rest\_rss/'. The page title is 'Mi Cuenta en JSP mvc\_rest\_rss'. Below the title, there is a blue header bar with the text 'Mi Cuenta en JSP'. Underneath the header, there is a table with four columns: 'Id', 'Nombre', 'Saldo', and 'Endeudamiento'. The table contains one row of data: '1', 'Andrés Guzmán', '5000', and '0.30'. Below the table, there is a button labeled 'Volver', which is circled in red.

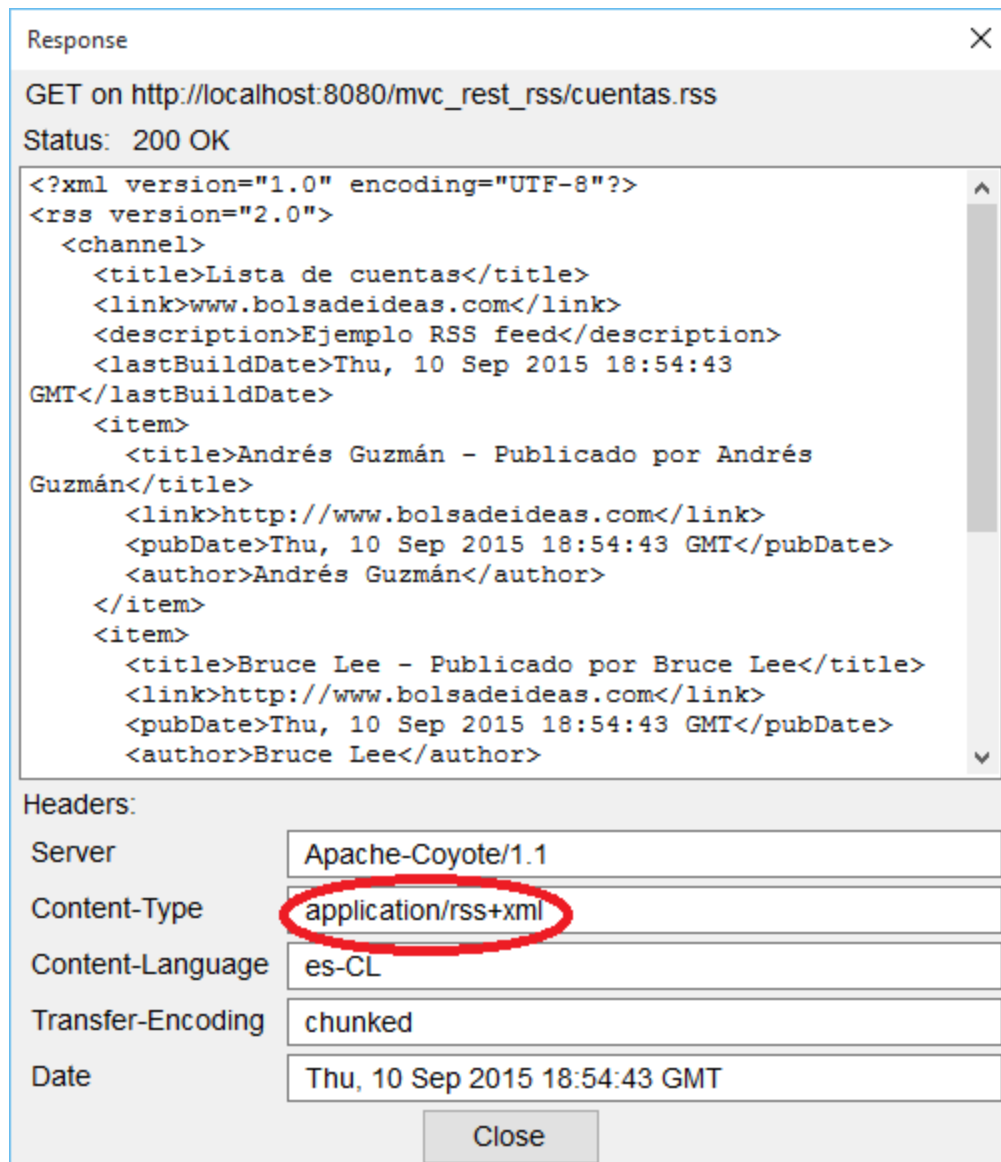
Id	Nombre	Saldo	Endeudamiento
1	Andrés Guzmán	5000	0.30

Volver

## 3. Accedemos al servicio a través de Poster

- Para el campo URL, ingresamos [http://localhost:8080/mvc\\_rest\\_rss/cuentas.rss](http://localhost:8080/mvc_rest_rss/cuentas.rss)
- Clic GET





4. Abrir y estudiar la clase controladora **CuentaController****/src/main/java/com/bolsadeideas/ejemplos/cuenta/controllers/CuentaController.java**

```
package com.bolsadeideas.ejemplos.cuenta.controllers;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value="/cuentas")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    @RequestMapping(method=RequestMethod.GET)
    public String getAccounts(Model model) {
        // Agregamos algunos datos de ejemplo
        Cuenta account = new Cuenta(new Long(0001), "Andres Guzmán",
            new BigDecimal("5000"), new BigDecimal("0.30"), new Date());
        cuentas.put(new Long(0001), account);

        account = new Cuenta(new Long(0002), "Bruce Lee", new BigDecimal("2000"),
            new BigDecimal("0.60"), new Date());
        cuentas.put(new Long(0002), account);

        account = new Cuenta(new Long(0003), "Rod Johnson", new BigDecimal("3000"),
            new BigDecimal("0.50"), new Date());
        cuentas.put(new Long(0003), account);

        // Agregamos el map de cuentas al atributo del model "cuentas",
        // para desplegarlos en el reporte de la vista rss
        model.addAttribute("cuentas", new ArrayList<Cuenta>(cuentas.values()));
        return "cuentas";
    }

    @RequestMapping(value="{id}", method=RequestMethod.GET)
    public String getView(@PathVariable Long id, Model model) {
        Cuenta cuenta = this.cuentas.get(id);
        model.addAttribute(cuenta);
        return "cuenta";
    }
}
```

## 5. Abrir y estudiar la clase de vista RSS.

**/src/main/java/com/bolsadeideas/ejemplos/cuenta/views/CuentaRssView.java.**

```
package com.bolsadeideas.ejemplos.cuenta.views;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.view.feed.AbstractRssFeedView;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;
import com.rometools.rome.feed.rss.Channel;
import com.rometools.rome.feed.rss.Item;

public class CuentaRssView extends AbstractRssFeedView {

    // Poblar los datos para el feed (title, link, description, etc.).
    @Override
    protected void buildFeedMetadata(Map<String, Object> model, Channel feed,
        HttpServletRequest request) {
        feed.setTitle("Lista de cuentas");
        feed.setDescription("Ejemplo RSS feed");
        feed.setLink("http://localhost:8080/mvc_rest_rss/cuentas");

        List<Cuenta> cuentas = (List<Cuenta>) model.get("cuentas");

        for (Cuenta cuenta : cuentas) {
            Date fecha = cuenta.getFechaRenovacion();
            if ((feed.getLastBuildDate() == null)
                || (fecha.compareTo(feed.getLastBuildDate()) > 0)) {
                feed.setLastBuildDate(fecha);
            }
        }
    }

    // Construimos el feed a partir de las cuentas
    @Override
    protected List<Item> buildFeedItems(Map<String, Object> model, HttpServletResponse response) throws Exception {
        List<Cuenta> cuentas = (List<Cuenta>) model.get("cuentas");
        List<Item> items = new ArrayList<Item>(cuentas.size());

        for (Cuenta cuenta : cuentas) {
            Item item = new Item();
            item.setAuthor(cuenta.getNombre());
            item.setTitle(String.format("%s - Publicado por %s",
                cuenta.getNombre(), cuenta.getNombre()));
            item.setPubDate(cuenta.getFechaRenovacion());
            item.setLink("http://localhost:8080/mvc_rest_rss/cuentas/"+cuenta.getId());
            items.add(item);
        }
        return items;
    }
}
```



## 6. Abrir y estudiar el archivo de configuración del contexto de Spring: servlet-context.xml. /src/main/webapp/WEB-INF/spring/appServlet/servlet-context.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:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-
mvc.xsd
           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.xsd">

    <!-- Escanea o busca en el package base de la aplicación clases beans anotados
         con @Components, @Controller, @Service -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos.cuenta.controllers" />

    <!-- Habilita la anotacion de Spring MVC @Controller -->
    <mvc:annotation-driven />

    <!-- Views configuradas y mapeadas en un bean, ej: id="cuentas" (clases RSS, PDF, XLS, etc) -->
    <bean id="contentNegotiatingResolver"
          class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
        <property name="order"
            value="#{T(org.springframework.core.Ordered).HIGHEST_PRECEDENCE}" />
    </bean>

    <!-- beanNameViewResolver Usamo esta config. para que el nombre del bean sea también el nombre
         de la vista(view resolver) -->
    <bean id="beanNameViewResolver"
          class="org.springframework.web.servlet.view.BeanNameViewResolver">
        <property name="order" value="#{contentNegotiatingResolver.order+1}" />
    </bean>

    <!-- La vista "cuentas" usa RSS view resolver -->
    <bean id="cuentas" class="com.bolsadeideas.ejemplos.cuenta.views.CuentaRssView" />

    <!-- Resuelve la ubicacion de las vistas .jsp de @Controllers en la ruta
         /WEB-INF/views -->
    <bean id="internalResourceResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
        <property name="order" value="#{beanNameViewResolver.order+1}" />
    </bean>
</beans>
```

7. Finalmente hay que tener configuradas las dependencias **maven** de **Rome RSS** en el pom.xml:

---

ETC...

```
<!-- Rome RSS -->
<dependency>
  <groupId>com.rometools</groupId>
  <artifactId>rome</artifactId>
  <version>1.5.1</version>
</dependency>
```

ETC...

---

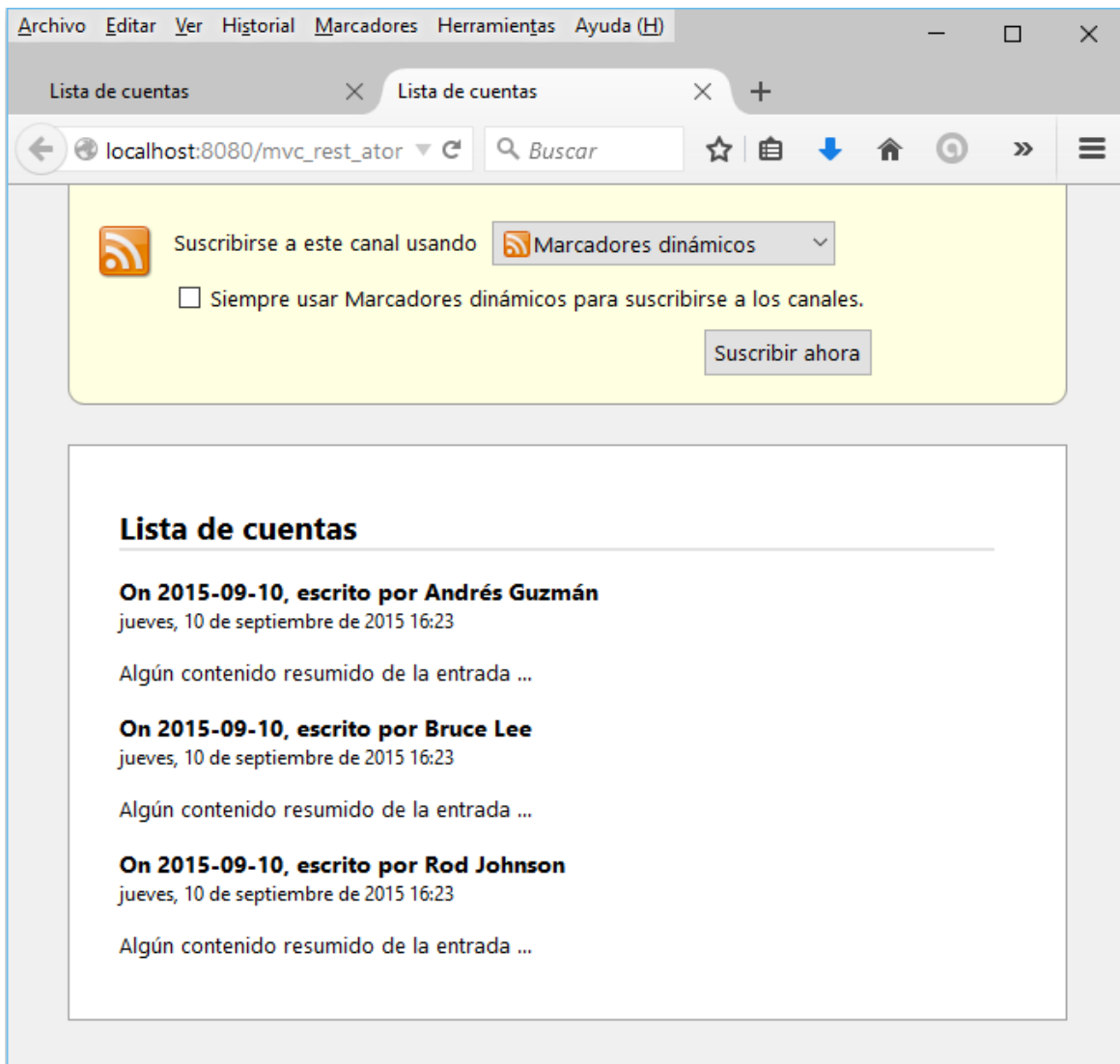
## Ejercicio 4: Generar y ejecutar el ejemplo "REST con Feed Atom"

El formato Atom fue desarrollado como una alternativa a RSS, en un archivo en formato XML usado para [Redifusión web](#).

En el ejemplo aprenderemos a configurar la vista Atom feed usando la clase AbstractAtomFeedView.

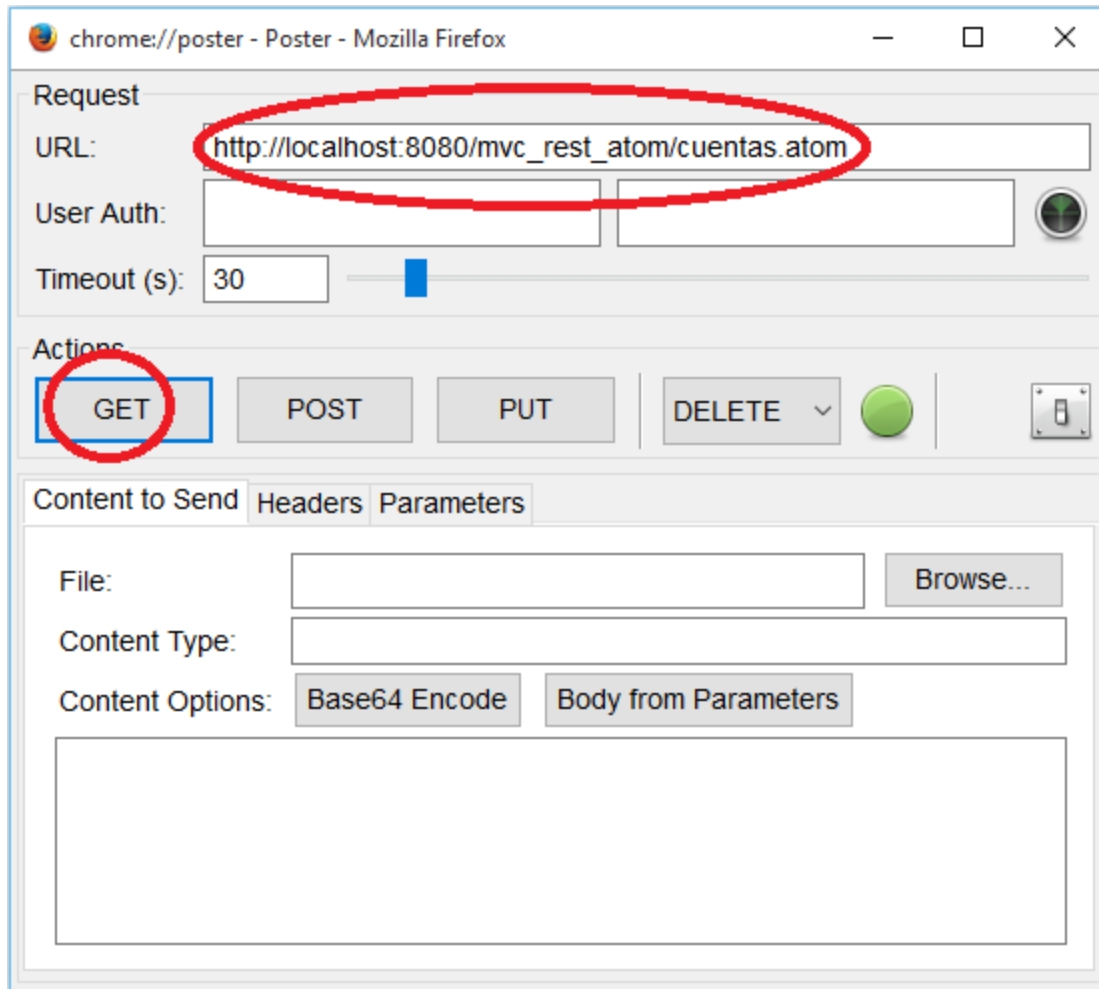
### 1. Ejecutamos el proyecto

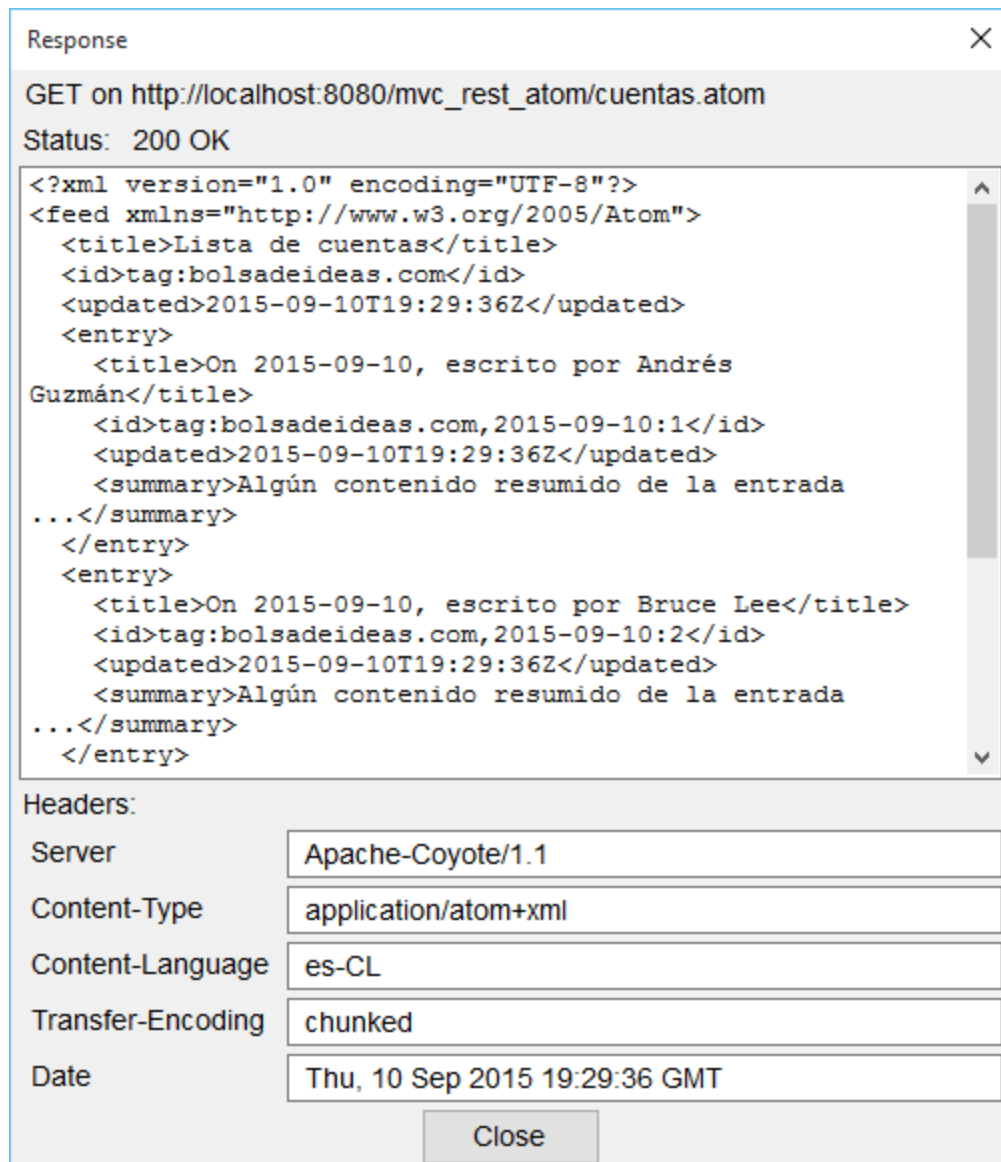
- Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
- Clic derecho sobre el proyecto y **Maven->Update Project...**
- Ejecutamos la aplicación: clic derecho sobre **mvc\_rest\_atom-> Run As on Server**
- Observe que la respuesta en el browser:



## 2. Accedemos al servicio a través de Poster

- Para el campo URL, ingresamos [http://localhost:8080/mvc\\_rest\\_atom/cuentas.atom](http://localhost:8080/mvc_rest_atom/cuentas.atom)
- Clic GET





3. Abrir y estudiar la clase controladora **CuentaController****/src/main/java/com/bolsadeideas/ejemplos/cuenta/controllers/CuentaController.java**

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value="/cuentas")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    @RequestMapping(method=RequestMethod.GET)
    public String getAccounts(Model model) {

        // Agregamos algunos datos de ejemplo
        Cuenta account = new Cuenta( new Long(0001), "Andres Guzmán",
                                     new BigDecimal("5000"), new BigDecimal("0.30"), new Date());
        cuentas.put(new Long(0001), account);

        account = new Cuenta( new Long(0002), "Bruce Lee",
                              new BigDecimal("2000"), new BigDecimal("0.60"), new Date());
        cuentas.put(new Long(0002), account);

        account = new Cuenta( new Long(0003), "Rod Johnson", new BigDecimal("3000"),
                              new BigDecimal("0.50"), new Date());
        cuentas.put(new Long(0003), account);

        // Agregamos el map de cuentas al atributo del model "cuentas",
        // para desplegarlos en el reporte de la vista ATOM
        model.addAttribute("cuentas", new ArrayList<Cuenta>(cuentas.values()));
        return "cuentas";
    }
}
```

## 4. Abrir y estudiar la clase de vista Atom.

**/src/main/java/com/bolsadeideas/ejemplos/cuenta/views/CuentaAtomView.java.**

```

package com.bolsadeideas.ejemplos.cuenta.views;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;
import com.sun.syndication.feed.atom.Content;
import com.sun.syndication.feed.atom.Entry;
import com.sun.syndication.feed.atom.Feed;
import org.springframework.web.servlet.view.feed.AbstractAtomFeedView;

public class CuentaAtomView extends AbstractAtomFeedView {

    @Override
    protected void buildFeedMetadata(Map<String, Object> model, Feed feed, HttpServletRequest request) {
        feed.setId("tag:bolsadeideas.com");
        feed.setTitle("Lista de cuentas");
        @SuppressWarnings("unchecked")
        List<Cuenta> cuentas = (List<Cuenta>)model.get("cuentas");
        for (Cuenta cuenta : cuentas) {
            Date fecha = cuenta.getFechaRenovacion();
            if (feed.getUpdated() == null || fecha.compareTo(feed.getUpdated()) > 0) {
                feed.setUpdated(fecha);
            }
        }
    }

    @Override
    protected List<Entry> buildFeedEntries(Map<String, Object> model,
        HttpServletRequest request, HttpServletResponse response) throws Exception {

        @SuppressWarnings("unchecked")
        List<Cuenta> cuentas = (List<Cuenta>)model.get("cuentas");
        List<Entry> entradas = new ArrayList<Entry>(cuentas.size());

        for (Cuenta cuenta : cuentas) {
            Entry entrada = new Entry();
            String fecha = String.format("%1$tY-%1$tm-%1$td", cuenta.getFechaRenovacion());
            entrada.setId(String.format("tag:bolsadeideas.com,%s:%d", fecha, cuenta.getId()));
            entrada.setTitle(String.format("On %s, escrito por %s", fecha, cuenta.getNombre()));
            entrada.setUpdated(cuenta.getFechaRenovacion());
            Content resumen = new Content();
            resumen.setValue("Algún resumen ...");
            entrada.setSummary(resumen);
            entradas.add(entrada);
        }

        return entradas;
    }
}

```

## 5. Abrir y estudiar el archivo de configuración del contexto de Spring: mvc-config.xml. /src/main/webapp/WEB-INF/config/mvc-config.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:mvc="http://www.springframework.org/schema/mvc"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd">

  <!-- Habilita la anotacion de Spring MVC @Controller -->
  <mvc:annotation-driven />

  <!-- Views configuradas y mapeadas en un bean, ej: id="cuentas" (clases JSON, PDF, XLS, etc) -->
  <bean
    class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
    <property name="mediaTypes">
      <map>
        <entry key="atom" value="application/atom+xml" />
        <entry key="html" value="text/html" />
        <entry key="pdf" value="application/pdf" />
        <entry key="xsl" value="application/vnd.ms-excel" />
        <entry key="xml" value="application/xml" />
        <entry key="json" value="application/json" />
      </map>
    </property>
    <property name="viewResolvers">
      <list>
        <bean class="org.springframework.web.servlet.view.BeanNameViewResolver" />
        <bean
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
          <property name="prefix" value="/WEB-INF/views/" />
          <property name="suffix" value=".jsp" />
        </bean>
      </list>
    </property>
  </bean>

  <!-- La vista "cuentas" manejada por la clase "CuentaAtomView" -->
  <bean id="cuentas" class="com.bolsadeideas.ejemplos.cuenta.views.CuentaAtomView" />
</beans>
```

## 8. Finalmente hay que tener configuradas las dependencias **maven** de **Rome RSS** en el pom.xml:

```
<dependency>
  <groupId>com.rometools</groupId>
  <artifactId>rome</artifactId>
  <version>1.5.1</version>
</dependency>
```

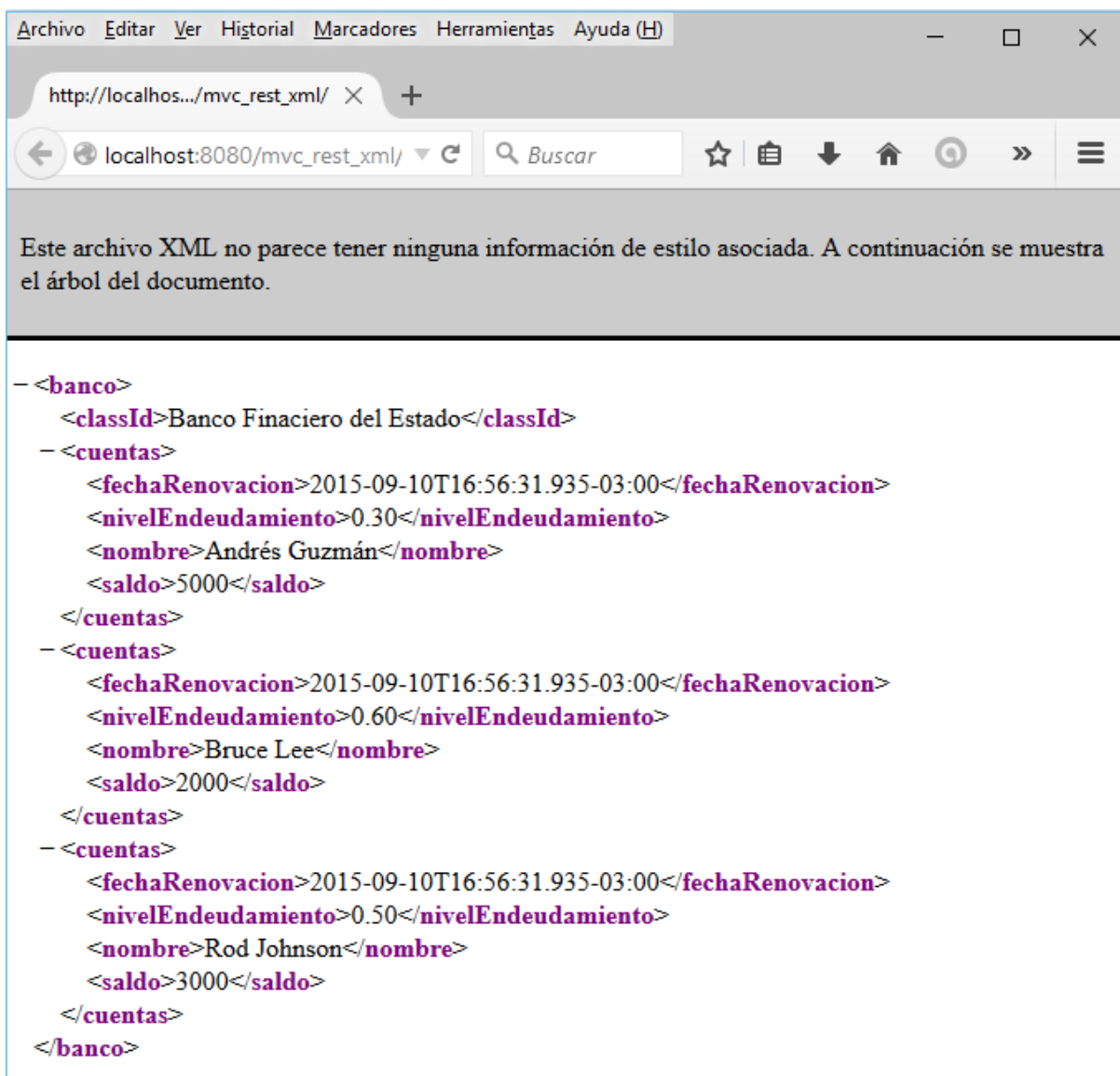


## Ejercicio 5: Generar y ejecutar el ejemplo "REST con salida XML"

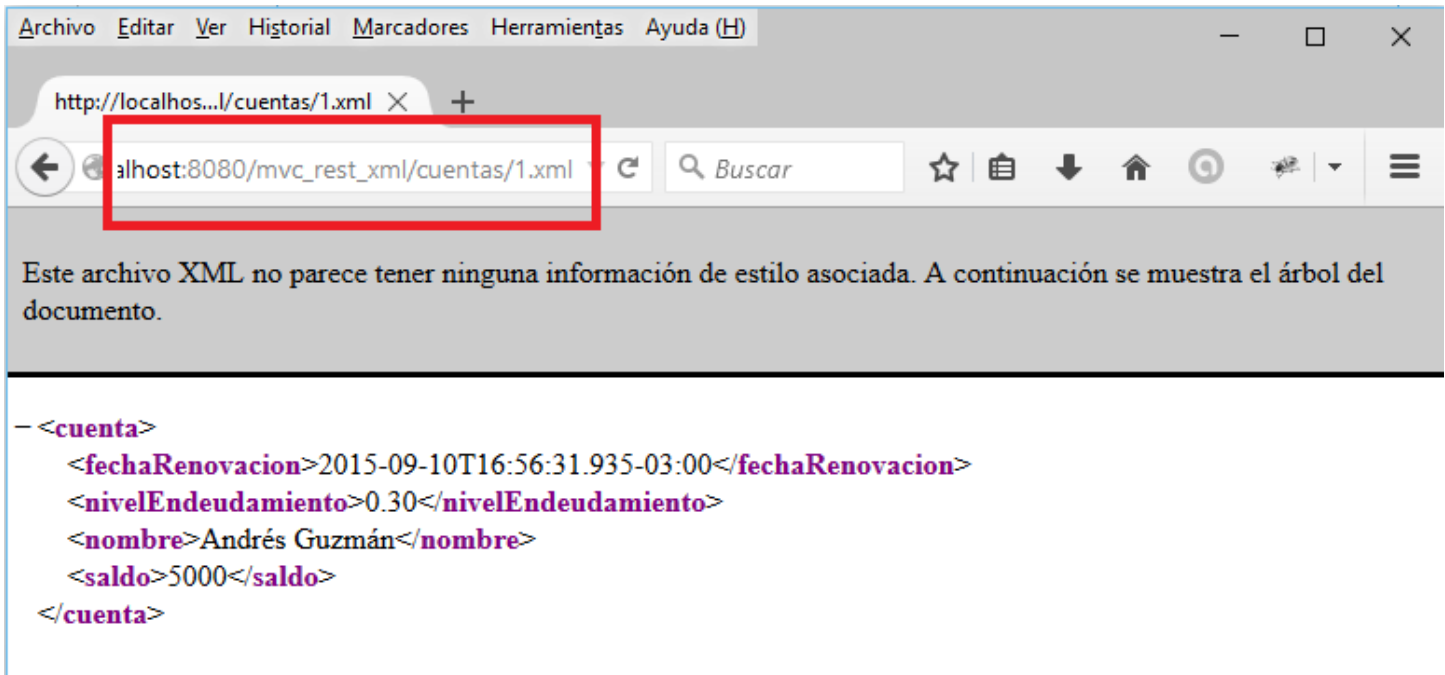
En el ejemplo veremos la configuración del **spring content negotiation** para que en la respuesta soporte el formato XML, aprenderemos a configurar como vista XML la clase de Spring `org.springframework.xml.jaxb.Jaxb2Marshaller`.

### 1. Ejecutamos el proyecto

- Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
- Clic derecho sobre el proyecto y **Maven->Update Project...**
- Ejecutamos la aplicación: clic derecho sobre **mvc\_rest\_xml-> Run As on Server**
- Observe que la respuesta en el browser:

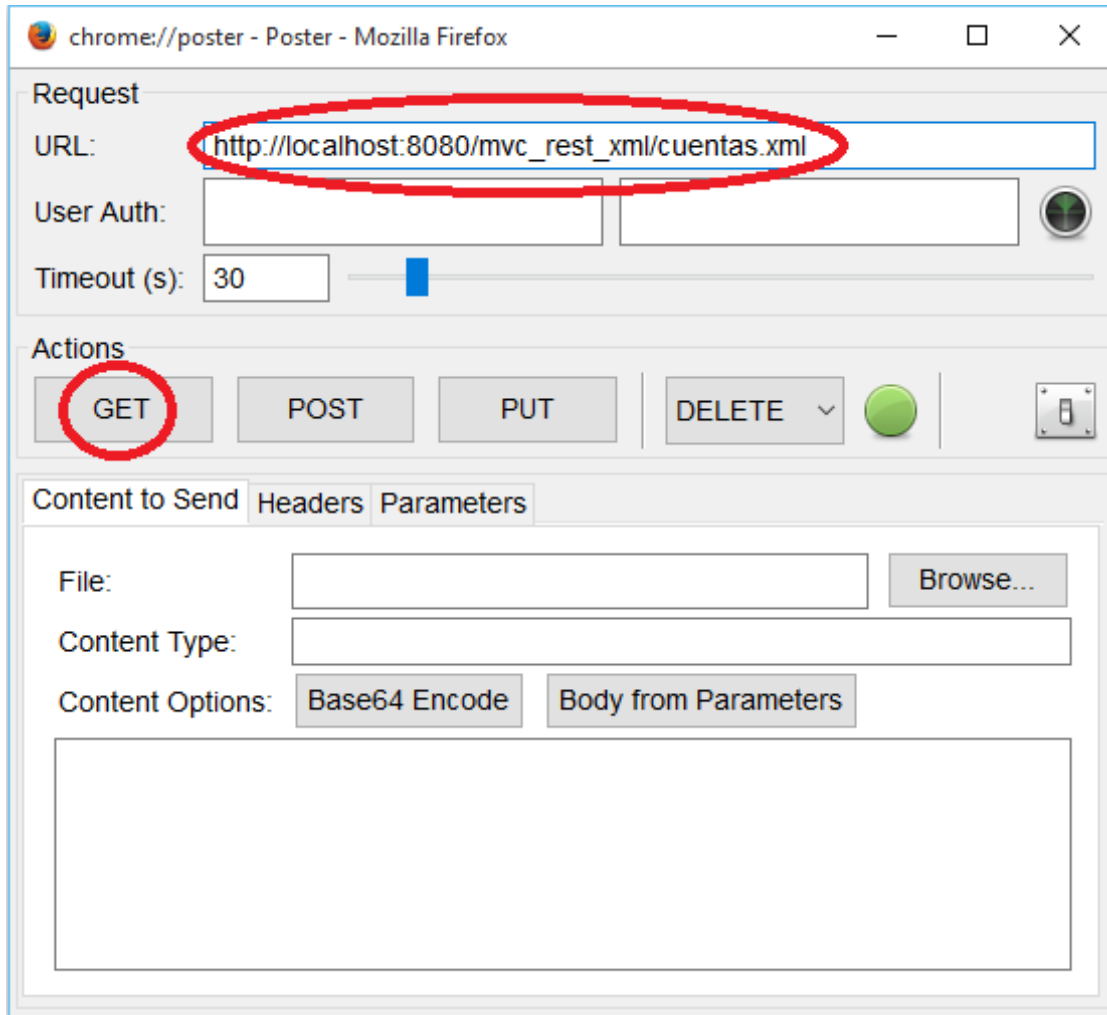


- Accedemos a una cuenta específica, con el id=1  
[http://localhost:8080/mvc\\_rest\\_xml/cuentas/1](http://localhost:8080/mvc_rest_xml/cuentas/1)



## 2. Accedemos al servicio a través de Poster

- Para el campo URL, ingresamos [http://localhost:8080/mvc\\_rest\\_xml/cuentas.xml](http://localhost:8080/mvc_rest_xml/cuentas.xml)
- Clic GET



Response

GET on http://localhost:8080/mvc\_rest\_xml/cuentas.xml  
Status: 200 OK

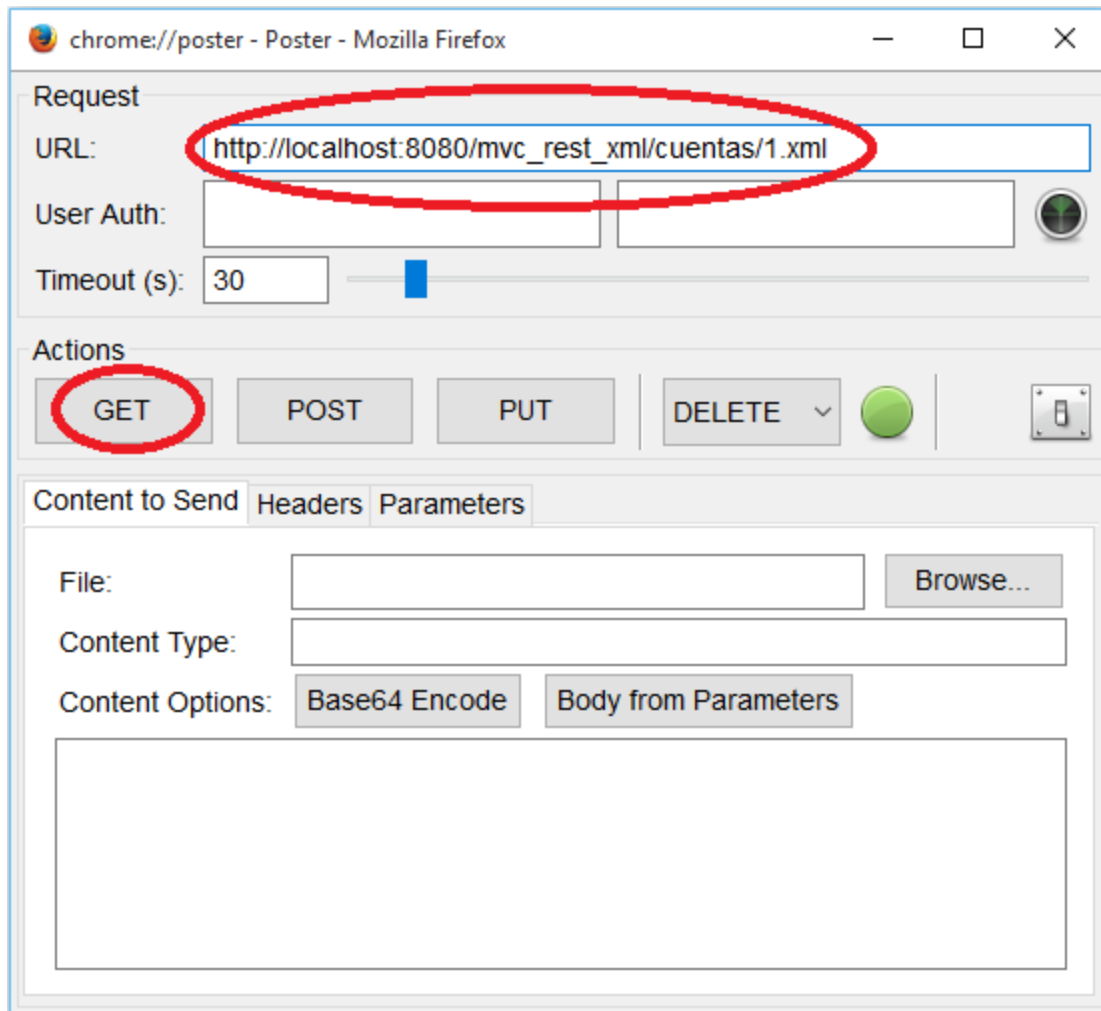
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<banco><classId>Banco Finaciero del Estado</classId>
<cuentas>
<fechaRenovacion>2015-09-10T16:59:07.176-03:00</fechaRenovac
ion><nivelEndeudamiento>0.30</nivelEndeudamiento>
<nombre>Andrés Guzmán</nombre><saldo>5000</saldo></cuentas>
<cuentas>
<fechaRenovacion>2015-09-10T16:59:07.176-03:00</fechaRenovac
ion><nivelEndeudamiento>0.60</nivelEndeudamiento>
<nombre>Bruce Lee</nombre><saldo>2000</saldo></cuentas>
<cuentas>
<fechaRenovacion>2015-09-10T16:59:07.176-03:00</fechaRenovac
ion><nivelEndeudamiento>0.50</nivelEndeudamiento>
<nombre>Rod Johnson</nombre><saldo>3000</saldo></cuentas>
</banco>
```

Headers:

Server	Apache-Coyote/1.1
Content-Type	application/xml
Content-Language	es-CL
Content-Length	642
Date	Thu, 10 Sep 2015 19:59:07 GMT

Close

- Accedemos a una cuenta específica, con el id=1



Response

GET on http://localhost:8080/mvc\_rest\_xml/cuentas/1.xml  
Status: 200 OK

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cuenta>
<fechaRenovacion>2015-09-10T16:59:07.176-03:00</fechaRenovacion>
<nivelEndeudamiento>0.30</nivelEndeudamiento>
<nombre>Andrés Guzmán</nombre>
<saldo>5000</saldo>
</cuenta>
```

Headers:

Server

Apache-Coyote/1.1

Content-Type

application/xml

Content-Language

es-CL

Content-Length

232

Date

Thu, 10 Sep 2015 19:59:50 GMT

Close

3. Abrir y estudiar la clase controladora **CuentaController****/src/main/java/com/bolsadeideas/ejemplos/cuenta/controllers/CuentaController.java**

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.bolsadeideas.ejemplos.cuenta.domain.Banco;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value="/cuentas")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    @RequestMapping(method=RequestMethod.GET)
    public String getAccounts(Model model) {

        // Agregamos algunos datos de ejemplo
        Cuenta account = new Cuenta( new Long(0001), "Andres Guzmán",
            new BigDecimal("5000"), new BigDecimal("0.30"), new Date());

        cuentas.put(new Long(0001), account);

        account = new Cuenta( new Long(0002), "Bruce Lee",
            new BigDecimal("2000"), new BigDecimal("0.60"), new Date());

        cuentas.put(new Long(0002), account);

        account = new Cuenta( new Long(0003), "Rod Johnson",
            new BigDecimal("3000"), new BigDecimal("0.50"), new Date());

        cuentas.put(new Long(0003), account);

        // Agregamos el map de cuentas al atributo del model "cuentas",
        // para desplegarlos en el reporte de la vista json
        model.addAttribute("cuentas", new Banco("Banco Finaciero del Estado",
            new ArrayList<Cuenta>(cuentas.values())));

        return "cuentas";
    }
}
```

```

@RequestMapping(value="{id}", method=RequestMethod.GET)
public String getView(@PathVariable Long id, Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }

    model.addAttribute(cuenta);
    return "cuenta";
}
}

```

4. Abrir y estudiar la clase Banco que contiene las cuentas, mapeada al XML usando la anotación `@XmlRootElement`.

**`/src/main/java/com/bolsadeideas/ejemplos/cuenta/domain/Banco.java`**

```

package com.bolsadeideas.ejemplos.cuenta.domain;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "banco")
public class Banco {

    private String classId = null;
    private List<Cuenta> cuentas = null;

    public Banco() {}

    public Banco(String id, List<Cuenta> cuentas) {
        this.classId = id;
        this.cuentas = cuentas;
    }

    public String getClassId() {
        return classId;
    }

    public void setClassId(String classId) {
        this.classId = classId;
    }

    @XmlElement(name="cuentas")
    public List<Cuenta> getCuentas() {
        return cuentas;
    }

    public void setCuentas(List<Cuenta> cuentas) {
        this.cuentas = cuentas;
    }
}

```



## 5. Abrir y estudiar la clase Cuenta mapeada al XML.

**/src/main/java/com/bolsadeideas/ejemplos/cuenta/domain/Cuenta.java**

```
package com.bolsadeideas.ejemplos.cuenta.domain;

import java.math.BigDecimal;
import java.util.Date;

import javax.validation.constraints.Future;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;

import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.NumberFormat;
import org.springframework.format.annotation.NumberFormat.Style;

@XmlRootElement(name = "cuenta")
public class Cuenta {

    private Long id;

    @NotNull
    @Size(min = 1, max = 25)
    private String nombre;

    @NotNull
    @NumberFormat(style = Style.CURRENCY)
    private BigDecimal saldo = new BigDecimal("5500");

    @NotNull
    @NumberFormat(style = Style.PERCENT)
    private BigDecimal nivelEndeudamiento = new BigDecimal(".05");

    @DateTimeFormat(style = "S-")
    @Future
    private Date fechaRenovacion = new Date(new Date().getTime() + 21425000000L);

    public Cuenta() {}

    public Cuenta(Long id, String nombre, BigDecimal saldo,
        BigDecimal nivelEndeudamiento, Date fechaRenovacion) {
        super();
        this.id = id;
        this.nombre = nombre;
        this.saldo = saldo;
        this.nivelEndeudamiento = nivelEndeudamiento;
        this.fechaRenovacion = fechaRenovacion;
    }

    public Long getId() {
        return id;
    }

    void setId(Long id) {
        this.id = id;
    }
}
```

```
public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public BigDecimal getSaldo() {
    return saldo;
}

public void setSaldo(BigDecimal saldo) {
    this.saldo = saldo;
}

public BigDecimal getNivelEndeudamiento() {
    return nivelEndeudamiento;
}

public void setNivelEndeudamiento(BigDecimal nivelEndeudamiento) {
    this.nivelEndeudamiento = nivelEndeudamiento;
}

public Date getFechaRenovacion() {
    return fechaRenovacion;
}

public void setFechaRenovacion(Date fechaRenovacion) {
    this.fechaRenovacion = fechaRenovacion;
}
}
```

---

## 6. Abrir y estudiar el archivo de configuración del contexto de Spring: servlet-context.xml. /src/main/webapp/WEB-INF/spring/appServlet/servlet-context.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:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        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.xsd">

<!-- Escanea o busca en el package base de la aplicación clases beans anotados
    con @Components, @Controller, @Service -->
<context:component-scan base-package="com.bolsadeideas.ejemplos.cuenta.controllers" />

<!-- Habilita la anotacion de Spring MVC @Controller -->
<mvc:annotation-driven />

<!-- Views configuradas y mapeadas en un bean, ej: id="cuentas_xls" (clases PDF, XLS, etc) -->
<bean id="contentNegotiatingResolver"
    class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
    <property name="order"
        value="#{T(org.springframework.core.Ordered).HIGHEST_PRECEDENCE}" />
</bean>

<!-- BeanNameViewResolver -->
<bean id="beanNameViewResolver"
    class="org.springframework.web.servlet.view.BeanNameViewResolver">
    <property name="order" value="#{contentNegotiatingResolver.order+1}" />
</bean>

<!-- "cuentas y cuenta" nombre de la vista en formato XML -->
<bean id="cuentas"
    class="org.springframework.web.servlet.view.xml.MarshallingView">
    <constructor-arg ref="jaxbMarshaller" />
</bean>
<bean id="cuenta"
    class="org.springframework.web.servlet.view.xml.MarshallingView">
    <constructor-arg ref="jaxbMarshaller" />
</bean>

<!-- JAXB2 marshaller. Automaticamente convierte un beans en salida xml -->
<bean id="jaxbMarshaller" class="org.springframework.xml.jaxb.Jaxb2Marshaller">
    <property name="classesToBeBound">
        <list>
            <value>com.bolsadeideas.ejemplos.cuenta.domain.Banco</value>
            <value>com.bolsadeideas.ejemplos.cuenta.domain.Cuenta</value>
        </list>
    </property>
</bean>
```

```
<!-- Resuelve la ubicacion de las vistas .jsp de @Controllers en la ruta /WEB-INF/views -->
<bean id="internalResourceResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/views/" />
  <property name="suffix" value=".jsp" />
  <property name="order" value="#{beanNameViewResolver.order+1}" />
</bean>
</beans>
```

7. Finalmente hay que tener configuradas las dependencias **maven** de **OXM de Spring** en el pom.xml:

ETC...

```
<!--Spring OXM -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-oxm</artifactId>
  <version>${spring.version}</version>
</dependency>
```

ETC...

## Ejercicio 6: Generar y ejecutar el ejemplo "mvc\_rest\_controller\_json"

En el ejemplo aprenderemos a implementar un Controlador REST JSON con la anotación **@RestController** de Spring, que combina las anotaciones **@ResponseBody** y **@Controller**.

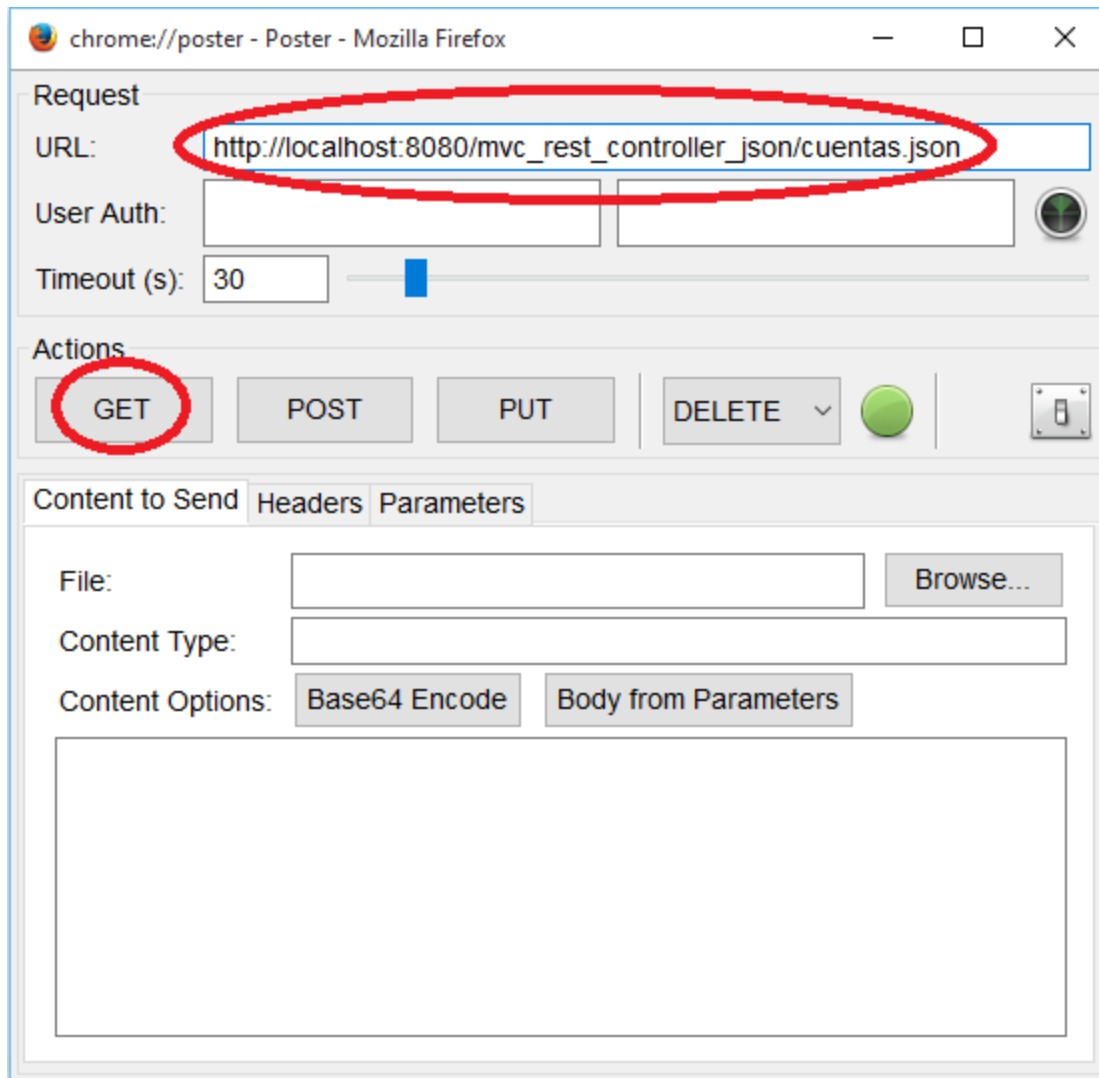
### 1. Ejecutamos el proyecto

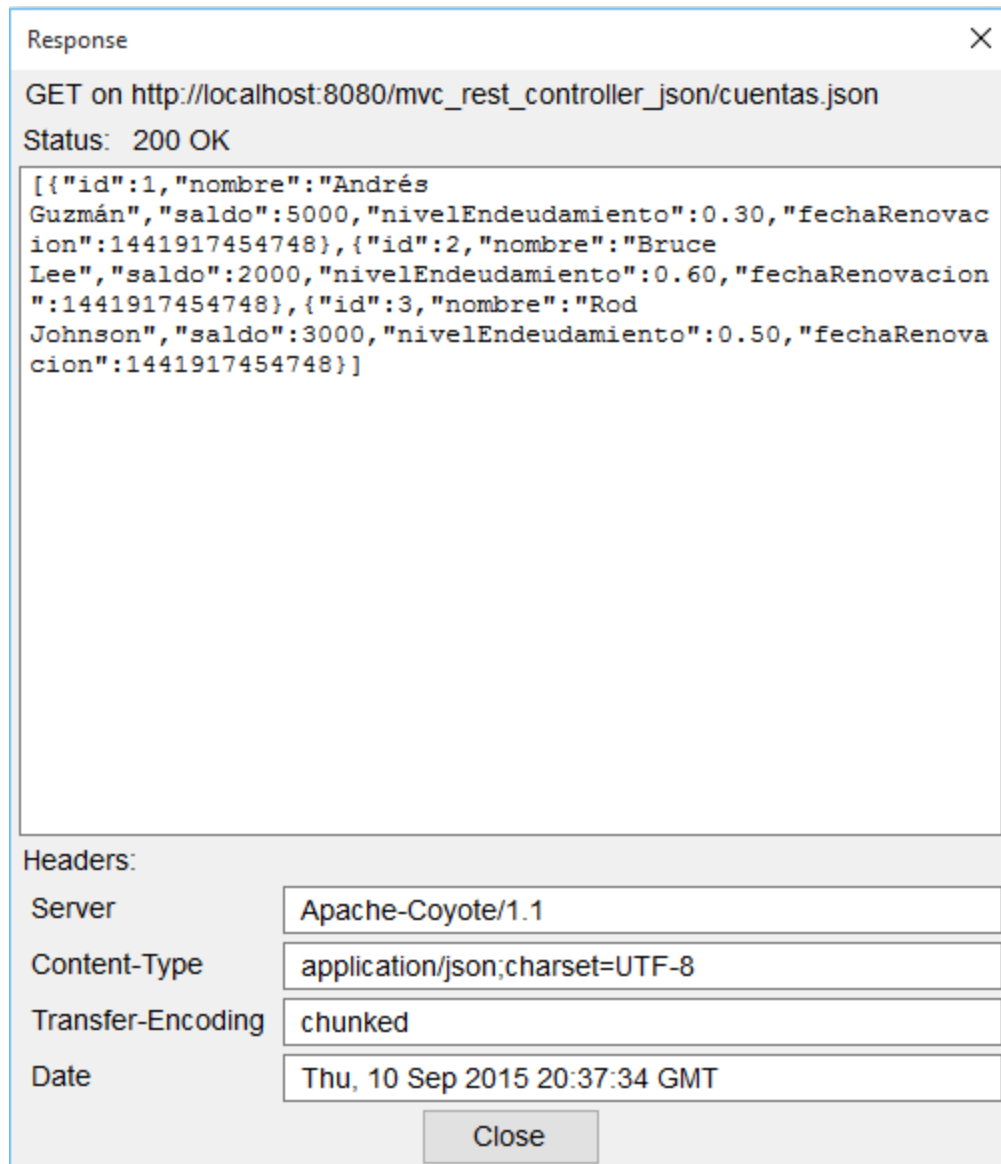
- Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
- Clic derecho sobre el proyecto y **Maven->Update Project...**
- Ejecutamos la aplicación: clic derecho sobre **mvc\_rest\_controller\_json-> Run As on Server**
- Observe que la respuesta JSON se despliega en pantalla



## 2. Accedemos al servicio a través de Poster

- Para el campo URL, ingresamos  
`http://localhost:8080/mvc_rest_controller_json/cuentas.json`
- Clic GET





3. Abrir y estudiar la clase controladora **CuentaController****/src/main/java/com/bolsadeideas/ejemplos/cuenta/controllers/CuentaController.java**

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@RestController
@RequestMapping(value="/cuentas")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    @RequestMapping(method=RequestMethod.GET)
    public ArrayList<Cuenta> getAccounts() {
        // Agregamos algunos datos de ejemplo
        Cuenta account = new Cuenta( new Long(0001), "Andrés Guzmán",
            new BigDecimal("5000"), new BigDecimal("0.30"), new Date());
        cuentas.put(new Long(0001), account);

        account = new Cuenta( new Long(0002), "Bruce Lee",
            new BigDecimal("2000"), new BigDecimal("0.60"), new Date());
        cuentas.put(new Long(0002), account);

        account = new Cuenta( new Long(0003), "Rod Johnson",
            new BigDecimal("3000"), new BigDecimal("0.50"), new Date());
        cuentas.put(new Long(0003), account);

        // Agregamos el map de cuentas al atributo del model "cuentas",
        // para desplegarlos en el reporte de la vista json
        return new ArrayList<Cuenta>(cuentas.values());
    }

    @RequestMapping(value="{id}", method=RequestMethod.GET)
    public Cuenta getView(@PathVariable Long id) {
        Cuenta cuenta = this.cuentas.get(id);
        return cuenta;
    }
}
```



- Es un caso de uso muy común tener controladores que necesiten implementar el API REST, sirviendo así solamente JSON, XML o contenido personalizado MediaType.
  - Para mayor comodidad, en vez de anotar todos nuestros métodos **@RequestMapping** con **@ResponseBody** o usar vistas especiales del tipo **MappingJackson2JsonView** (vista en el primer ejemplo), podemos anotar nuestro Controlador con **@RestController** y por defecto y de forma transparente asume como respuesta **@ResponseBody** para todos los métodos handler del controlador.
  - Entonces para resumir, la anotación **@RestController** es un estereotipo que combina las anotaciones **@ResponseBody** y **@Controller** para darle más sentido y simplicidad a nuestros controladores REST.
4. Abrir y estudiar el archivo de configuración del contexto de Spring: `src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml`.
- Observamos que ya no es necesario configurar la vista **JSON** del tipo **MappingJackson2JsonView** como en el primer ejemplo.

```
<?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:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc/spring-mvc.xsd
                           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.xsd">

    <!-- Escanea o busca en el package base de la aplicación clases beans anotados
         con @Components, @Controller, @Service -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos.cuenta.controllers" />

    <!-- Habilita la anotacion de Spring MVC @Controller -->
    <mvc:annotation-driven />

    <!-- View Resolvers -->
    <!-- Resuelve la ubicacion de las vistas .jsp de @Controllers en la ruta /WEB-INF/views -->
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

## Ejercicio 7: Generar y ejecutar el ejemplo "mvc\_rest\_controller\_xml"

En el ejemplo aprenderemos a implementar un Controlador REST XML con la anotación **@RestController** de Spring, que combina las anotaciones **@ResponseBody** y **@Controller**.

### 1. Ejecutamos el proyecto

- Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
- Clic derecho sobre el proyecto y **Maven->Update Project...**
- Ejecutamos la aplicación: clic derecho sobre **mvc\_rest\_controller\_xml-> Run As on Server**
- Observe que la respuesta XML se despliega en pantalla

```

- <banco>
  <classId>Banco Finaciero del Estado</classId>
  - <cuentas>
    <fechaRenovacion>2015-09-10T18:04:32.225-03:00</fechaRenovacion>
    <nivelEndeudamiento>0.30</nivelEndeudamiento>
    <nombre>Andrés Guzmán</nombre>
    <saldo>5000</saldo>
  </cuentas>
  - <cuentas>
    <fechaRenovacion>2015-09-10T18:04:32.225-03:00</fechaRenovacion>
    <nivelEndeudamiento>0.60</nivelEndeudamiento>
    <nombre>Bruce Lee</nombre>
    <saldo>2000</saldo>
  </cuentas>
  - <cuentas>
    <fechaRenovacion>2015-09-10T18:04:32.225-03:00</fechaRenovacion>
    <nivelEndeudamiento>0.50</nivelEndeudamiento>
    <nombre>Rod Johnson</nombre>
    <saldo>3000</saldo>
  </cuentas>
</banco>

```

## 2. Abrir y estudiar la clase controladora **CuentaController**

**/src/main/java/com/bolsadeideas/ejemplos/cuenta/controllers/CuentaController.java**

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@RestController
@RequestMapping(value="/cuentas")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    @RequestMapping(method=RequestMethod.GET)
    public ArrayList<Cuenta> getAccounts() {
        // Agregamos algunos datos de ejemplo
        Cuenta account = new Cuenta( new Long(0001), "Andrés Guzmán",
            new BigDecimal("5000"), new BigDecimal("0.30"), new Date());
        cuentas.put(new Long(0001), account);

        account = new Cuenta( new Long(0002), "Bruce Lee",
            new BigDecimal("2000"), new BigDecimal("0.60"), new Date());
        cuentas.put(new Long(0002), account);

        account = new Cuenta( new Long(0003), "Rod Johnson",
            new BigDecimal("3000"), new BigDecimal("0.50"), new Date());
        cuentas.put(new Long(0003), account);

        // Agregamos el map de cuentas al atributo del model "cuentas",
        // para desplegarlos en el reporte de la vista XML
        return new ArrayList<Cuenta>(cuentas.values());
    }

    @RequestMapping(value="{id}", method=RequestMethod.GET)
    public Cuenta getView(@PathVariable Long id) {
        Cuenta cuenta = this.cuentas.get(id);
        return cuenta;
    }
}
```

- Es un caso de uso muy común tener controladores que necesiten implementar el API REST, sirviendo así solamente JSON, XML o contenido personalizado MediaType.
  - Para mayor comodidad, en vez de anotar todos nuestros métodos **@RequestMapping** con **@ResponseBody** o usar vistas especiales del tipo **MarshallingView** (vista en el ejemplo `mvc_rest_xml`), podemos anotar nuestro Controlador con **@RestController** y por defecto y de forma transparente asume como respuesta **@ResponseBody** para todos los métodos handler del controlador.
  - Entonces para resumir, la anotación **@RestController** es un estereotipo que combina las anotaciones **@ResponseBody** y **@Controller** para darle más sentido y simplicidad a nuestros controladores REST.
3. Abrir y estudiar el archivo de configuración del contexto de Spring: `src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml`.
- Observamos que ya no es necesario configurar la vista **XML** del tipo **MarshallingView** como en el ejemplo `mvc_rest_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:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc/spring-mvc.xsd
                           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.xsd">

    <!-- Escanea o busca en el package base de la aplicación clases beans anotados
         con @Components, @Controller, @Service -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos.cuenta.controllers" />

    <!-- Habilita la anotacion de Spring MVC @Controller -->
    <mvc:annotation-driven />

    <!-- View Resolvers -->
    <!-- Resuelve la ubicacion de las vistas .jsp de @Controllers en la ruta /WEB-INF/views -->
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

## Resumen

En este capítulo vimos cómo trabajar con Web Service REST, Atom y Feed RSS con Spring Framework, tanto como servidor (para crear servicios web) y cómo clientes, accediendo a los servicios y desplegar los datos. Abarcamos lo suficiente para empezar a desarrollar aplicaciones que acceda y desplieguen Servicios Web basados en REST.

¡Dudas, a los foros! ;-)

**FIN.**

**Envía tus consultas a los foros!**

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes

## Lectura Recomendada y Bibliografía

- [Rest Controller](#): recomendable lectura del manual oficial para complementar con este workshop.
- [RestTemplate - Accessing RESTful services on the Client](#): recomendable lectura del manual oficial para complementar con este workshop.
- [REST in Spring](#): tutorial escrito por Arjen Poutsma.
- [Adding an Atom view to an application using Spring's REST support](#): tutorial escrito por Alef Arendsen.
- [REST in Spring: RestTemplate](#) escrito por Alef Arendsen.