# Implementació CRUD amb servei REST i JQuery

## A qui va dirigit

Arquitectes i desenvolupadors d'aplicacions basades en Canigó.

## Versió de Canigó

A partir de la versió 1.0.0 de Canigó 3.

## Introducció

El propòsit del document és proporcionar un exemple d'implementació d'un manteniment CRUD (Create-Read-Update-Delete) d'una entitat de negoci utilitzant un servei REST expossat amb Spring MVC i consumit amb Javascript (JQuery) en una aplicació Canigó 3. El format de les dades que intercanviaran client i servidor serà JSON.

També s'explicarà el concepte JSONP (JSON with Padding) que permetrà fer crides AJAX entre diferents dominis.

## Instal·lació

Cal afegir les següents dependències al fitxer pom.xml del projecte:

**pom.xml**

```
<!-- MVC -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${org.springframework.version}</version>
</dependency>

<!-- Transaccionalitat -->
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${org.springframework.version}</version>
</dependency>

<!-- Processador JSON -->
<dependency>
        <groupId>org.codehaus.jackson</groupId>
        <artifactId>jackson-mapper-asl</artifactId>
        <version>1.9.12</version>
</dependency>

<!-- ArrayUtils a Spring MVC+JSONP -->
<dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.6</version>
</dependency>
```

## Configuració

En el fitxer web.xml descriptor de l'aplicació web ha de definir-se el servlet que rebrà les peticions dels serveis REST expossats:

**web.xml**

```
<servlet>
	<servlet-name>dispatcher</servlet-name>
	<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
	<servlet-name>dispatcher</servlet-name>
	<url-pattern>/restServices/*</url-pattern>
</servlet-mapping>
```

Un cop afegida la definició del servlet cal configurar-lo. Per això crearem el fitxer dispachter-servlet.xml (<servlet-name>-servlet.xml) amb el següent contingut:

**dispatcher-servlet.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
	xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

	<context:component-scan base-
package="cat.gencat.canigo.aplicaciodemorestjs.endpoints" />

	<tx:annotation-driven />
	<mvc:annotation-driven />

	<bean
class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
		<property name="order" value="1" />
		<property name="mediaTypes">
			<map>
				<entry key="json" value="application/json" />
			</map>
		</property>
		<property name="defaultViews">
			<list>
				<bean
class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />
			</list>
		</property>
	</bean>

	<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<property name="order" value="2" />
		<property name="prefix" value="/WEB-INF/views/" />
		<property name="suffix" value=".jsf" />
	</bean>
```

```
</beans>
```

## Publicació dels mètodes de negoci

Per la publicació es fà un mapeig dels mètodes per la realització del manteniment de l'entitat, en aquest d'exemple l'entitat User, i els diferents mètodes HTTP:

- GET per obtenir i cercar dades
- POST per afegir dades
- PUT per actualitzar dades
- DELETE per eliminar dades

Aquest és el codi corresponent al controlador on es realitza el mapeig mitjançant anotacions i s'invoca al servei que conté el negoci per l'alta, modificació, baixa i cerca de l'entitat:

**UserServiceController.java**

```
package cat.gencat.canigo.aplicaciodemorestjs.endpoints;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import cat.gencat.canigo.aplicaciodemorestjs.model.User;
import cat.gencat.canigo.aplicaciodemorestjs.service.UserService;

@Controller
@RequestMapping("/userService")
public class UserServiceController {

	@Autowired
	UserService userService;

	@RequestMapping(method = RequestMethod.GET)
	@ResponseBody
	public List<User> findAll() {
		return userService.findAll();
	}

	@RequestMapping(value = "/{id}", method = RequestMethod.GET)
	@ResponseBody
	public User getUser(@PathVariable("id") Long userId) {
		return userService.getUser(userId);
	}

	@RequestMapping(method = RequestMethod.POST)
	@Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)
	public void saveUser(@RequestBody User user)
			throws Exception {
		userService.saveUser(user);
	}
```

```
        @RequestMapping(value = "/{id}", method = RequestMethod.PUT)
        @Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)
        public void updateUser(@RequestBody User user)
                        throws Exception {
                userService.updateUser(user);
        }

        @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
        @Transactional(propagation = Propagation.REQUIRED, rollbackFor = Exception.class)
        @ResponseBody
        public void deleteUser(@PathVariable("id") Long userId)
                        throws Exception {
                userService.deleteUser(userId);
        }
}
```

El servei ha estat mapejat en el context /userService i els mètodes de negoci segons el criteri especificat anteriorment també amb l'anotació **@RequestMapping**.

Les dades de la petició es poden obtenir tant de la URL (**@PathVariable**) com del cos de la petició **(@RequestBody**). El binding de dades JSON <-> POJO és realitzat per la llibreria de processament JSON Jackson. Qualsevol incongruència entre els camps definits a la classe Usuari i el JSON enviat per el client produirà una excepció.

## Transaccionalitat

La transaccionalitat és un aspecte fundamental per garantir la integritat de les dades. Els mètodes que impliquen alteració de les dades es marquen amb l'anotació **@Transactional**. Qualsevol excepció a l'execució d'aquests mètodes produïrà un rollback a nivell de base de dades.

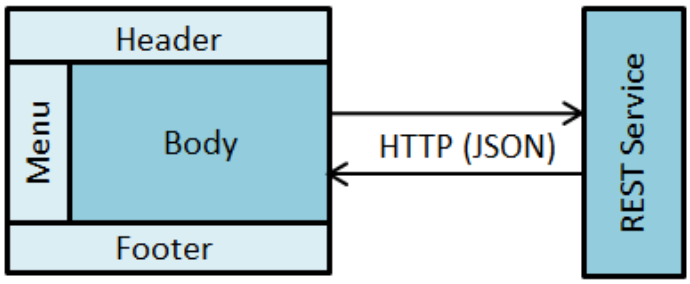
# Consum del servei REST

Un cop expossat el servei, aquest pot ser consumit per diferents tipologies de clients degut al desacoblament que ofereix la publicació mitjançant REST entre el servidor i el client.

## Canigó 3 - JSF

A continuació veurem el detall d'un client basat en una aplicació Canigó 3 amb plantillatge JSF. Les dues pàgines implementades per el manteniment de l'entitat Usuari, userList.xhtml i userForm.xhtml es basen en una plantilla template.jsf:

# Implementació CRUD amb servei REST i JQuery




Com es pot observar en la figura, el contingut de les pàgines (body) on es visualitza el llistat i el detall de l'usuari interactua mitjançant peticions HTTP amb el servei REST. Aquesta interacció és realitza gràcies a JQuery realitzant peticions Ajax i a la serialització/deserialització de les dades JSON amb Jackson.

**userList.xhtml** (per brevetat hem omitit el codi javascript corresponent a la funció createTableView)

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:rich="http://richfaces.org/rich"
        xmlns:c="http://java.sun.com/jstl/core">

        <f:view contentType="text/html" />

        <ui:composition template="layouts/template.jsf">
                <ui:define name="body">

                        <script>
                        //<![CDATA[

                        $(document).ready(function(){
                                $.ajax({
                                        type: "GET",
                                        url:
"/canigo.aplicaciodemorestjs/AppJava/restServices/userService",
                                        dataType:"json",
                                        success: function(data){
                                                if(data.length > 0){

        $('#content').append(createTableView(data)).fadeIn();
                                                }
                                                else
                                                {
                                                        $('#content').html('No existeixen
usuaris');
                                                }
                                        },
                                 error: function (result) {
                                        alert('ERROR ' + result.status + ' ' +
result.statusText);
                                        }
                                });

                                $('#enterAddUser').click(function() {

        $(window).attr('location','/canigo.aplicaciodemorestjs/AppJava/views/userForm.xht
ml');
                                        });
                        });
                        //]]>
                        </script>

                        <div data-role="page">
```

# Implementació CRUD amb servei REST i JQuery

```
                        <div data-role="header" data-theme="e">
                                <h1>Llista d'usuaris</h1>
                        </div>
                                <div data-role="content" id="content">
                            </div>

                                <div id="toolbar">
                                    <input type="button" id="enterAddUser"
name="enterAddUser" value="Nou" />
                                </div>
                        </div>
                </ui:define>
        </ui:composition>
</html>
```

**userForm.xhtml**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:rich="http://richfaces.org/rich"
        xmlns:c="http://java.sun.com/jstl/core">

        <f:view contentType="text/html" />

        <ui:composition template="layouts/template.jsf">
                <ui:define name="body">

                        <script>
                        //<![CDATA[
                                $.urlParam = function(name){
                                        var results = new RegExp('[\\?&]' + name +
'=([^&#]*)').exec(window.location.href);

                                        if (results==null){
                                                return null;
                                        }else{
                                                return results[1] || 0;
                                        }
                                }

                                $(document).ready(function() {

                                        if($.urlParam('userId') != null) {

        $('#userId').val(decodeURIComponent($.urlParam('userId')));

                                                $.ajax({
                                                        type: "GET",
                                                        url:
"/canigo.aplicaciodemorestjs/AppJava/restServices/userService/" + $("#userId").val(),
                                                        dataType:"json",
                                                        success: function(data){
                                                                $.each(data, function(key,
value){

$("#"+key).val(value);

                                                                });

                                                        },
                                                 error: function (result) {
                                                        alert('ERROR ' + result.status + ' '
+ result.statusText);

                                                        }
                                                });
                                        }
```

# Implementació CRUD amb servei REST i JQuery

```
                                $('#saveOrUpdateUser').click(function() {

                                        var userForm =
$('#userForm').serializeArray();
                                        var userFormObject = {};
                                        $.each(userForm,
                                            function(i, v) {
                                                if(v.name == 'enabled') {
                                                        userFormObject[v.name] =
(v.value == "on") ? "true" : "false";
                                                        }
                                                else {
                                                                userFormObject[v.name]
= v.value;
                                                }
                                            });

                                        $.ajax({
                                           type: ($('#userId').val() != '') ? "PUT"
: "POST",
                                           url:
"/canigo.aplicaciodemorestjs/AppJava/restServices/userService/"
+userFormObject['userId'],
                                           data: JSON.stringify(userFormObject),
                                           contentType: "application/json;
charset=utf-8",
                                           dataType: "json",
                                           success: function (response) {

$(window).attr('location','/canigo.aplicaciodemorestjs/AppJava/views/userList.xhtml');
                                           },
                                           error: function (result) {
                                             alert('ERROR ' + result.status + ' ' +
result.statusText);
                                           }
                                        });
                                });

                                $('#deleteUser').click(function() {
                                        if($.urlParam('userId') != null) {
                                                $.ajax({
                                                   type: "DELETE",
                                                   url:
"/canigo.aplicaciodemorestjs/AppJava/restServices/userService/" + $("#userId").val(),
                                                   dataType: "json",
                                                   success: function (response) {

$(window).attr('location','/canigo.aplicaciodemorestjs/AppJava/views/userList.xhtml');
                                                   },
                                                   error: function (result) {
                                                     alert('ERROR ' + result.status +
' ' + result.statusText);
                                                   }
                                            });
                                        }
                                });

                        });

                //]]>
                </script>

                <div data-role="page">
                 <div data-role="header" data-theme="e">
                        <h1>Usuari</h1>
                 </div>
                 <div data-role="content">
                     <form id="userForm">
                        <div id="usuari" data-role="fieldcontain">
```

```
                                    <input type="hidden" name="userId" id="userId"
value=""  />
                                    <label for="nom">Username</label>
                                <input type="text" name="username" id="username"
value=""  />
                                <label for="nom">Name</label>
                                <input type="text" name="name" id="name" value=""  />
                                <label for="cognoms">Surname</label>
                                <input type="text" name="surname" id="surname" value=""
/>
                                <label for="cognoms">Enabled</label>
                                <input type="checkbox" name="enabled" id="enabled"
checked="checked" />
                            </div>
                            <div data-role="fieldcontain">
                                    <input type="button" id="saveOrUpdateUser"
name="saveOrUpdateUser" value="Guardar" />
                                    <input type="button" id="deleteUser"
name="deleteUser" value="Eliminar" />
                            </div>
                        </form>
                    </div>
                </div>

                </ui:define>
        </ui:composition>
</html>
```

*Nota: en aquest exemple no s'ha realitzat cap tasca de maquetació ni d'internacionalització de literals*

## Java

Des de codi Java també és possible consumir serveis REST. Aquest és un exemple per obtenir el llistat d'usuaris:

**RESTServiceClient.java**

```
package cat.gencat.canigo.aplicaciodemorestjs.client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;

public class RESTServiceClient {

        public static final String REST_ENDPOINT =
        "http://localhost:8080/canigo.aplicaciodemorestjs/AppJava/restServices";
        public static final String SERVICE_INFO_PATH = "/userService";

        public static void main(String[] args) throws IOException {

                String endpoint = REST_ENDPOINT + SERVICE_INFO_PATH;

                HttpURLConnection request = null;
                BufferedReader rd = null;
                StringBuilder response = null;

                try {
                        URL endpointUrl = new URL(endpoint);
                        request = (HttpURLConnection) endpointUrl.openConnection();
                        request.setRequestMethod("GET");
```

```
                        //findAll
                        request.connect();

                        rd = new BufferedReader(new InputStreamReader(
                                        request.getInputStream()));
                        response = new StringBuilder();
                        String line = null;
                        while ((line = rd.readLine()) != null) {
                                response.append(line + '\n');
                        }
                } catch (MalformedURLException e) {
                        System.out.println("Exception: " + e.getMessage());
                        // e.printStackTrace();
                } catch (ProtocolException e) {
                        System.out.println("Exception: " + e.getMessage());
                        // e.printStackTrace();
                } catch (IOException e) {
                        System.out.println("Exception: " + e.getMessage());
                        // e.printStackTrace();
                } catch (Exception e) {
                        System.out.println("Exception: " + e.getMessage());
                        // e.printStackTrace();
                } finally {
                        try {
                                request.disconnect();
                        } catch (Exception e) {
                        }

                        if (rd != null) {
                                try {
                                        rd.close();
                                } catch (IOException ex) {
                                }
                                rd = null;
                        }
                }

                //JSON to POJO
                ObjectMapper mapper = new ObjectMapper();

                try {
                        List<User> userList = mapper.readValue(response.toString(), new
TypeReference<List<User>>(){});
                        // display to console
                        System.out.println(userList);
                } catch (JsonGenerationException e) {
                        e.printStackTrace();
                } catch (JsonMappingException e) {
                        e.printStackTrace();
                }
        }
}
```

## Aplicacions mòbils

També es podrien incloure d'altres exemples de clients com podrien ser aplicacions mòbils per a diferents dispositius Android, iPhone, etc.

# JSONP (JavaScript Object Notation with Padding)

# Implementació CRUD amb servei REST i JQuery

Tal com hem vist, l'ús de les tecnologies REST i JSON permet construir serveis als quals es poden connectar multitud de clients: des d' aplicacions lleugeres basades en HTML+Javascript a sistemes d'informació que en facin ús d'aquestes dades a la banda de servidor (Ex. des de codi Java).

En el cas de clients web, aquest desacoblament "tecnològic" no permet però realitzar crides AJAX entre diferents dominis. És a dir, si el servei REST està desplegat en un domini X.com, el client web ha d'estar desplegat en aquest mateix domini. L'accés des d'un altre domini Y.com produiria un error, com per exemple el següent amb Firefox:

```
Access to restricted URI denied" code: "1012
```

Per tal que un client web pugui realitzar crides AJAX a dominis diferents al que pertany, és a dir, crides entre dominis, pot utilitzar-se una tècnica coneguda com a JSONP. Mitjançant l'ús d'aquesta tècnica clients web desplegats a dominis Y.com, Z.com, etc. poden accedir a serveis REST que pertanyen a un altre domini X.com. D'aquest manera s'aconsegueix a més d'un desacoblament "tecnològic", un desacoblament de domini i fins i tot d'infraestructura entre les aplicacions client i el servidor.

JSONP és una extensió de JSON per suportar crides entre dominis. A mode de resum, aquesta tècnica es basa en que sí es pot carregar un script d'un domini remot, de forma que si tenim el nostre domini Y.com, podem fer el següent:

```
<script type="text/javascript" src="http://X.com/dades.js"></script>
```

Però d'aquesta manera no podem accedir a la informació retornada. Amb JSONP, el servidor s'haurà d'encarregar de posar aquestes dades com a paràmetre d'una funció existent al nostre codi Javascript.

## Suport JSONP amb Spring MVC

Per tal d'implementar un servei REST amb Spring MVC (v3.0.3.FINAL) que retorni JSONP enlloc de JSON caldrà definir el següent MessageConverter:

**MappingJackson2JsonpHttpMessageConverter.java**

```
package cat.gencat.canigo.aplicaciodemorestjs.endpoints.converter;

import java.io.IOException;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpServletRequest;

import org.codehaus.jackson.JsonEncoding;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.springframework.http.HttpOutputMessage;
import org.springframework.http.MediaType;
import org.springframework.http.converter.HttpMessageNotWritableException;
import org.springframework.http.converter.json.MappingJacksonHttpMessageConverter;
import org.springframework.util.Assert;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

/**
 * Converter class that we are going to use to create json-p messages.
```

```
 *
 * cscanigo
 *
 */
public class MappingJackson2JsonpHttpMessageConverter extends
MappingJacksonHttpMessageConverter {

    private static final String DEFAULT_CALLBACK_PARAMETER = "callback";

    private ObjectMapper objectMapper = new ObjectMapper();

    private static final List<MediaType> SUPPORTED_MEDIA_TYPES = new
ArrayList<MediaType> () { {
        add( new MediaType("application", "x-javascript") );
        add( new MediaType("application", "javascript") );
        add( new MediaType("text", "javascript") );
    }};


    public MappingJackson2JsonpHttpMessageConverter() {
        setSupportedMediaTypes(SUPPORTED_MEDIA_TYPES);
    }


    @Override
    protected void writeInternal( Object object, HttpOutputMessage outputMessage )
        throws IOException, HttpMessageNotWritableException {
        JsonGenerator jsonGenerator = getJsonGenerator(outputMessage);

        try {
            String callbackParam = getRequestParam( DEFAULT_CALLBACK_PARAMETER );

            if ( callbackParam == null || "".equals(callbackParam.trim()) ) {
                //if the callback parameter doesn't exists, use the default one...
                callbackParam = DEFAULT_CALLBACK_PARAMETER;
            }

            jsonGenerator.writeRaw(callbackParam);
            jsonGenerator.writeRaw(" (");
            this.getObjectMapper().writeValue(jsonGenerator, object);
            jsonGenerator.writeRaw(");");
            jsonGenerator.flush();
        } catch (JsonProcessingException ex) {
            throw new HttpMessageNotWritableException("Could not write JSON:"  +
ex.getMessage(), ex);
        }
    }


    private JsonGenerator getJsonGenerator( HttpOutputMessage outputMessage ) throws
IOException {
        JsonEncoding encoding =
getJsonEncoding(outputMessage.getHeaders().getContentType());
        return
getObjectMapper().getJsonFactory().createJsonGenerator(outputMessage.getBody(),
encoding);
    }

    /**
     * Returns given parameter from servlet request.
     *
     * @param paramName
     *          Name of the param
     */
    private String getRequestParam( String paramName ) {
        return getServletRequest().getParameter( paramName );
    }

    /**
     * Returns current servlet request.
```

```
     */
    private HttpServletRequest getServletRequest() {
        return ((ServletRequestAttributes)
RequestContextHolder.currentRequestAttributes()).getRequest();
    }

    /**
         * Determine the JSON encoding to use for the given content type.
         * @param contentType the media type as requested by the caller
         * @return the JSON encoding to use (never <code>null</code>)
         */
        protected JsonEncoding getJsonEncoding(MediaType contentType) {
                if (contentType != null && contentType.getCharSet() != null) {
                        Charset charset = contentType.getCharSet();
                        for (JsonEncoding encoding : JsonEncoding.values()) {
                                if (charset.name().equals(encoding.getJavaName())) {
                                        return encoding;
                                }
                        }
                }
                return JsonEncoding.UTF8;
        }


        /**
         * Set the {@code ObjectMapper} for this view. If not set, a default
         * {@link ObjectMapper#ObjectMapper() ObjectMapper} is used.
         * <p>Setting a custom-configured {@code ObjectMapper} is one way to take further
control of the JSON
         * serialization process. For example, an extended {@link
org.codehaus.jackson.map.SerializerFactory}
         * can be configured that provides custom serializers for specific types. The
other option for refining
         * the serialization process is to use Jackson's provided annotations on the
types to be serialized,
         * in which case a custom-configured ObjectMapper is unnecessary.
         */
        public void setObjectMapper(ObjectMapper objectMapper) {
                Assert.notNull(objectMapper, "ObjectMapper must not be null");
                this.objectMapper = objectMapper;
        }

        /**
         * Return the underlying {@code ObjectMapper} for this view.
         */
        public ObjectMapper getObjectMapper() {
                return this.objectMapper;
        }

}
```

També caldria modificar el fitxer de configuració dispatcher-servlet.xml per afegir el nou MessageConverter:

**dispatcher-servlet.xml**

```
…
        <bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapterConf
igurer" init-method="init">
          <property name="messageConverters">
            <list>
              <bean id="jacksonMessageConverter"

class="cat.gencat.canigo.aplicaciodemorestjs.endpoints.converter.MappingJackson2JsonpHtt
pMessageConverter" />
            </list>
          </property>
        </bean>
```

...

Per tal de registrar el nou MessageConverter customitzat ha estat necessari afegir la següent classe al projecte:

**AnnotationMethodHandlerAdapterConfigurer.java**

```
package org.springframework.web.servlet.mvc.annotation;

import org.apache.commons.lang.ArrayUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.util.PathMatcher;
import org.springframework.web.bind.support.WebArgumentResolver;
import org.springframework.web.bind.support.WebBindingInitializer;
import org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter;
import org.springframework.web.servlet.mvc.annotation.ModelAndViewResolver;
import org.springframework.web.servlet.mvc.multiaction.MethodNameResolver;
import org.springframework.web.util.UrlPathHelper;


public class AnnotationMethodHandlerAdapterConfigurer {

    @Autowired
    private AnnotationMethodHandlerAdapter adapter;

    private WebBindingInitializer webBindingInitializer;

    private HttpMessageConverter[] messageConverters;
    private PathMatcher pathMatcher;
    private UrlPathHelper urlPathHelper;
    private MethodNameResolver methodNameResolver;
    private WebArgumentResolver[] customArgumentResolvers;
    private ModelAndViewResolver[] customModelAndViewResolvers;

    private boolean replaceMessageConverters = false;

    public void init() {
        if (webBindingInitializer != null) {
            adapter.setWebBindingInitializer(webBindingInitializer);
        }

        if (messageConverters != null) {
            if (replaceMessageConverters) {
                adapter.setMessageConverters(messageConverters);
            } else {
                adapter.setMessageConverters(mergeMessageConverters());
            }
        }

        if (pathMatcher != null) {
            adapter.setPathMatcher(pathMatcher);
        }

        if (urlPathHelper != null) {
            adapter.setUrlPathHelper(urlPathHelper);
        }

        if (methodNameResolver != null) {
            adapter.setMethodNameResolver(methodNameResolver);
        }

        if (customArgumentResolvers != null) {
            adapter.setCustomArgumentResolvers(customArgumentResolvers);
        }

        if (customModelAndViewResolvers != null) {
            adapter.setCustomModelAndViewResolvers(customModelAndViewResolvers);
        }
```

## Implementació CRUD amb servei REST i JQuery

```
    }

    private HttpMessageConverter[] mergeMessageConverters() {
        return (HttpMessageConverter[]) ArrayUtils.addAll(messageConverters,
adapter.getMessageConverters());
    }

    public void setWebBindingInitializer(WebBindingInitializer webBindingInitializer) {
        this.webBindingInitializer = webBindingInitializer;
    }

    public void setPathMatcher(PathMatcher pathMatcher) {
        this.pathMatcher = pathMatcher;
    }

    public void setUrlPathHelper(UrlPathHelper urlPathHelper) {
        this.urlPathHelper = urlPathHelper;
    }

    public void setMethodNameResolver(MethodNameResolver methodNameResolver) {
        this.methodNameResolver = methodNameResolver;
    }

    public void setCustomArgumentResolver(WebArgumentResolver argumentResolver) {
        this.customArgumentResolvers = new WebArgumentResolver[] { argumentResolver };
    }

    public void setCustomArgumentResolvers(WebArgumentResolver[] argumentResolvers) {
        this.customArgumentResolvers = argumentResolvers;
    }

    public void setCustomModelAndViewResolver(ModelAndViewResolver
customModelAndViewResolver) {
        this.customModelAndViewResolvers = new ModelAndViewResolver[] {
customModelAndViewResolver };
    }

    public void setCustomModelAndViewResolvers(ModelAndViewResolver[]
customModelAndViewResolvers) {
        this.customModelAndViewResolvers = customModelAndViewResolvers;
    }

    public void setMessageConverters(HttpMessageConverter[] messageConverters) {
        this.messageConverters = messageConverters;
    }

    public void setReplaceMessageConverters(boolean replaceMessageConverters) {
        this.replaceMessageConverters = replaceMessageConverters;
    }
}
```

Un cop realitzada la configuració del servei REST per a donar suport a JSONP, es pot realitzar la següent crida AJAX amb JQuery per accedir a aquest servei REST desplegat al domini X.com des d'un client web del domini Y.com:

```
        $.ajax({
                url:
"http://X.com:8080/canigo.aplicaciodemorestjs/AppJava/restServices/userService?callback=
?",
                        dataType: "jsonp",
                success: function (data) {
                if(data.length > 0){

        $('#content').append(createTableView(data)).fadeIn();
                                                }
                                                else
```

## Implementació CRUD amb servei REST i JQuery

```
                                        {
                                                $('#content').html('No existeixen
usuaris');
                                        }                               },
                                error: function (result) {
                                        alert('ERROR ' + result.status + ' ' +
result.statusText);
                                        }
                        });
```