REST alkalmazás jQuery klienssel és Jersey szerverrel

Áttekintés

Ennek a leírásnak az a célja, hogy bemutassa, miként lehet egy szerver oldali Java-REST alkalmazáshoz kliensoldalon JavaScript-jQuery segítségével csatlakozni. Java oldalon a REST szolgáltatást a Jersey keretrendszer segítségével valósítottam meg. A JavaSript és a Java oldal közti adatátvitelt elször XML-lel, majd JSON-nal mutatom be. Az így kialakított architektúra az alábbi elnyökkel rendelkezik:

- A kliens és a szerveroldal technológia független egymástól, egyedül abban kell megegyezniük, hogy XML-t vagy JSON-t használnak az adatok átadására. A szerver oldalhoz akár több, különböz technológiával megírt kliens is kapcsolódhat.
- Az architektúra felépítése kevés kódolással jár, mivel a JavaScript és a Java oldali technológiák is nagymértékben támogatják a megvalósítását.
- Mivel a kliens AJAX-ot, a szerver oldal REST-et használ, a hálózaton kevés adat utazik.
- A jQuery jelenleg a legnépszerbb JavaScript keretrendszer, sok más keretrendszer is erre épül. Jelenleg több JavaScript feljeszt használ jQuery-t, mint aki nem: http://w3techs.com/technologies/overview/javascript_library/all
- A REST is egyre népszerbb a Java programozók között, része a Jave EE 6 szabványnak.

A leírásban szerepl kódokat Windows7-en, JDK7-en, Tomcat8 webszerveren teszteltem, de más Java Servlet kompatibilis webszerver is alkalmas erre. A telepítéshez Maven-t használtam, mert a Jersey felhasználói segédlete is azt használja: https://jersey.java.net/documentation/latest/index.html

Same-origin policy

Bár a jelen cikket nem érinti, fontosnak tartom megemlíteni ezt a biztonsági megkötést. Egy JavaScript program csak olyan fájlokat és szolgáltatásokat érhet el, amik ugyanabból az internet domain-bl származnak, mint ahonnan a JavaScript forrást letöltöttük. Például a saját weboldalunkon lév JavaScript nem használhat egy idegen weboldalon lév JavaScript könyvtárat, ha használni akarjuk, át kell másolnunk a saját domain-ünkbe.

Mi micsoda

jQuery: Egy nyílt forrású JavaScript könyvtár, mely segíti böngész független JavaScript kód írását és könnyebbé teszi a HTML elemek elérését és az AJAX hívások megvalósítását.

AJAX: (Asynchronous JavaScript and XML): Segítségével a weboldalnak nem kell újra betöltdnie egy szerver oldali metódus meghívása után, csak a kívánt tartalom frissül rajta.

XML (Extensible Markup Language): Egy jelöl nyelv, mely segítségével összetett adatokat jeleníthetünk meg és továbbíthatunk mind az ember, mind a számítógép programok számára érthet formában Pl.: <todo><priority>1</priority><task>do something</task></todo>

JSON (JavaScript Object Notation): Az XML versenytársa, JavaScript nyelven ír le összetett adatot. Pl.: { 'priority': 1, 'task': 'do something'}

REST: (Representational State Transfer): Egy állapotmentes kliens - szerver architektúra, melyben a kliens HTTP üzenetek segítségével kommunikál a szerverrel.

Jersey: Nyílt forráskódú, a REST-et megvalósító Java keretrendszer. A JAX-RS szabvány referencia implementációja.

Tomcat: Nyílt forráskódú webszerver, mely képes Java Servlet alapú alkalmazások futtatására.

Maven: Szoftver project management eszköz, mely Java alkalmazások létrehozását, telepítését, tesztelését és dokumentálását segíti. Jellegzetessége, hogy a Java alkalmazás által igényelt könyvtárakat (jar) saját és megosztott repository-kban tárolja.

JAXB: Java objektumot tud XML-lé alakítani és XML-t Java objektummá.

Maven telepítése

Töltsük le a Maven-t a http://maven.apache.org/download.cgi#Installation weboldalról (pl.: apache-maven-3.1.1-bin.zip).

Csomagoljuk ki a letöltött zip fájlt egy általunk létrehozott könyvtárba vagy a gyökérkönyvtárba (pl.: c:\apache-maven-3.1.1).

Állítsuk be a szükséges környezeti változókat (pl.: Windows 7 alatt: Start > jobb klikk a Computer-en > Advanced system settings > Environment variables > User variables > Path > Edit):

- JAVA_HOME (pl.: JAVA_HOME=C:\Java\jdk1.7.0_13)
- PATH-hoz adjuk hozzá a JDK bin könyvtárát (pl.: Path=...;%JAVA_HOME%\bin)
- M2_HOME (pl.: M2_HOME= C:\apache-maven-3.1.1)
- PATH-hoz adjuk hozzá a Maven bin könyvtárát (pl.: Path=...;%M2_HOME%\bin)

Telepítés ellenrzése command promptból:

```
mvn --version
```

Tomcat telepítése

Töltsük le a http://tomcat.apache.org/download-80.cgi weboldalról a legfrissebb verziót (pl.: apache-tomcat-8.0.0-RC3.zip).

Csomagoljuk ki a letöltött zip fájlt egy általunk létrehozott könyvtárba vagy a gyökérkönyvtárba (pl.: C:\apache-tomcat-8.0.0-RC3)

Állítsuk be a szükséges környezeti változókat, ha még nem tettük meg (pl.: Windows 7 alatt: Start > jobb klikk a Computer-en > Advanced system settings > Environment variables > User variables > Path > Edit):

- JAVA_HOME (pl.: JAVA_HOME=C:\Java\jdk1.7.0_13)
- PATH-hoz adjuk hozzá a JDK bin könyvtárát (pl.: Path=...;%JAVA_HOME%\bin)

Indítsuk el a Tomcat-et: C:\apache-tomcat-8.0.0-RC3\bin\startup.bat

Próbáljuk ki, hogy a böngészben a http://localhost:8080/ visszaadja a Tomcat üdvözloldalát. A Tomcat-et az alábbi paranccsal lehet majd leállítni: C:\apache-tomcat-8.0.0-RC3\bin\shutdown.bat

Maven mintaalkalmazás létrehozása

Menjünk abba az alkönyvtárba, ahol a projectet létre akarjuk hozni és írjuk be:

```
mvn archetype:generate -DarchetypeGroupId=org.glassfish.jersey.archetypes \
    -DarchetypeArtifactId=jersey-quickstart-webapp -DarchetypeVersion=2.3.1
```

Az archetypeVersion-t igazítsuk a legfrissebb Jersey verziószámhoz. További információ: https://jersey.java.net/documentation/latest/getting-started.html. A projekt létrehozása során a Maven az alábbiakat fogja kérdezni:

- groupId: ez lesz az alkalmazás csoport neve (pl.: jersey1)
- artifactId: ez lesz a war fájl neve (pl.: jersey1)
- version: maradhat amit felajánl
- package: ez lesz a Java package neve (pl.: jersey1)

Ha sikerült létrehoznunk a projekt-et, menjünk bele a létrejött jersey1 könyvtárba, ahol a pom.xml is van, és adjuk ki a következ parancsot:

```
mvn install
```

Létrejön egy target könyvtár és benne egy jersey1.war fájl. A jersey1.war fájt másoljuk a Tomcat webapps könyvtárába (pl.: C:\apache-tomcat-8.0.0-RC3\webapps\jersey1.war). Ha a webapps könyvtárban már volt egy kicsomagolt jersey1 könyvtár, azt töröljük le. Indítsuk el a Tomcat-et ha nincs elindítva. Írjuk be a böngészbe: http://localhost:8080/jersey1/

A telepítés gyorsítására írtam két rövid script-et, melyeket a pom.xml-lel azonos könyvtárban helyeztem el.

deploy.bat

```
set CATALINA_HOME=c:\apache-tomcat-8.0.0-RC3
set APP_NAME=jersey1
call mvn clean install
copy target\%APP_NAME%.war %CATALINA_HOME%\webapps
call %CATALINA_HOME%\bin\startup.bat
```

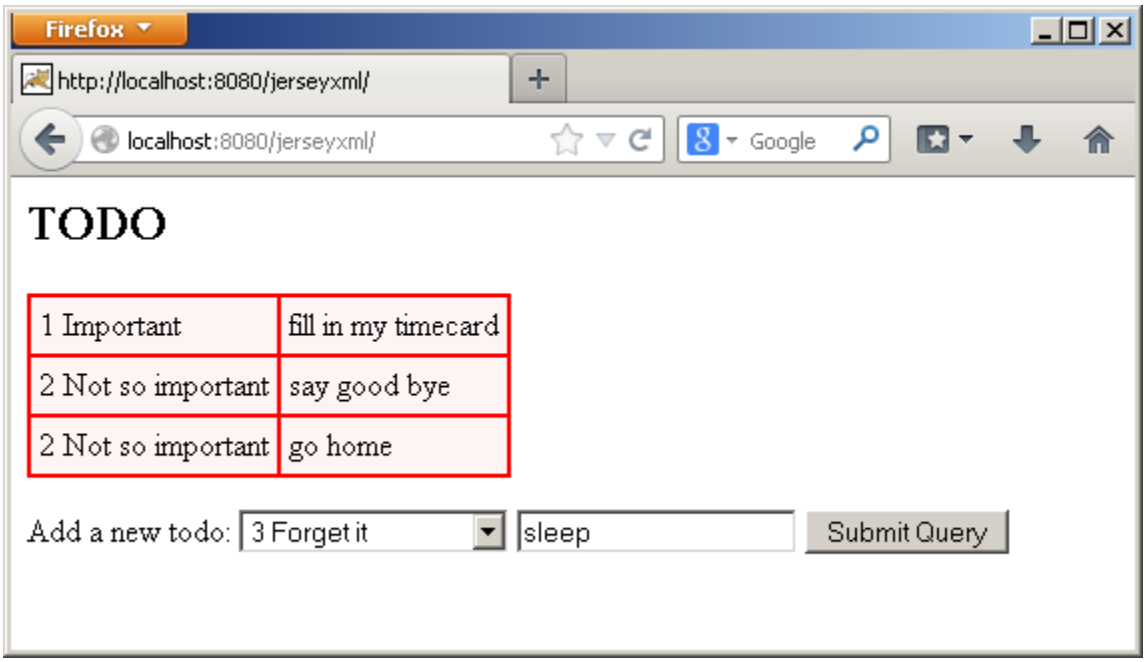
undeploy.bat

```
set CATALINA_HOME=c:\apache-tomcat-8.0.0-RC3
set APP_NAME=jersey1
rd /S /Q %CATALINA_HOME%\webapps\%APP_NAME%
del %CATALINA_HOME%\webapps\%APP_NAME%.war
call %CATALINA_HOME%\bin\shutdown.bat
```

Futtatás eltt írjuk át a CATALINA_HOME és az APP_NAME értékeit a nekünk megfelel értékekre. Az undeploy.bat-ot néha kétszer is le kell futtatni, lehet, hogy szerencsésebb lenne elször a shutdown.bat-ot meghívni, utána várakozni egy kicsit és utána letörölni a könyvtárat és a war-t.

Továbbfejlesztett mintaalkalmazás

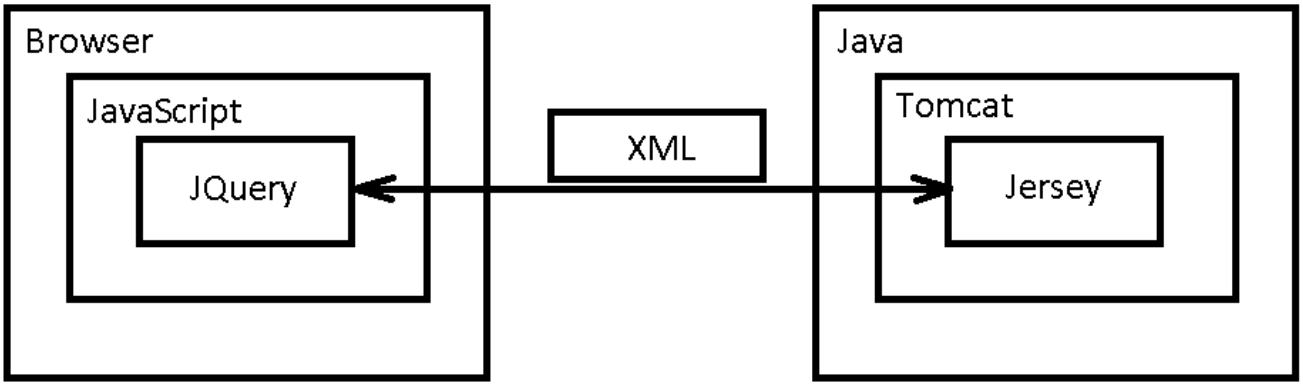
A Maven mintaalkalmazás két továbbfejlesztését mutatom be. Mindkét továbbfejlesztett mintaalkalmazás egy Todo objektumokból álló listát tud megmutatni és a listához hozzá adni egy új Todo objektumot, mint azt az alábbi ábra is mutatja:



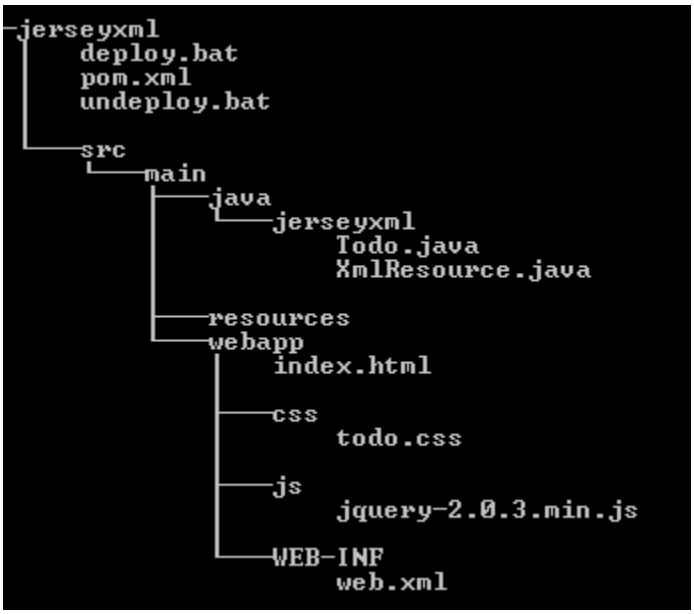
A következ két fejezet a továbbfejlesztett mintaalkalmazások implementációját mutatja be. Mindkét esetben Jersey (Java szerver oldal) és jQuery (JavaScript kliens oldal) használatával. A különbség az, hogy az els esetben az adatok XML, a másodikban JSON formában utaznak a Java szerver oldal és a JavaScript kliens oldal között. A továbbiakban szükségünk lesz jQuery-re is, amit a http://jquery.com/ weboldalról tölthetünk le.

XML alapú üzenetküldés megvalósítása

Az alkalmazás felépítése:



Fájlok:



Az ábra az alábbi MS-DOS paranccsal készült:

```
tree /f
```

Todo.java

```
package jerseyxml;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Todo {

    private int priority;

    String task;

    public Todo() {
    }

    public Todo(int priority, String task) {
        setPriority(priority);
        setTask(task);
    }

    public int getPriority() {
        return priority;
    }

    public void setPriority(int priority) {
        this.priority = priority;
    }

    public String getTask() {
        return task;
    }

    public void setTask(String task) {
        this.task = task;
    }

    @Override
    public String toString() {
        return "Todo[" + priority + "," + task + "]";
    }
}
```

A Todo.java egy sima POJO (Plain Old Java Object), kiegészítve egy @XmlRootElement annotációval, ami lehetvé teszi hogy a Jersey a JAXB segítségével automatikusan XML formátumú szöveggé alakítsa az objektumot. Ez az átalakítás automatikusan mködik visszafelé is XML-bl POJO-ba. A POJO-nak kell tartalmaznia egy üres konstruktort is.

XmlResource.java

```
package jerseyxml;

import java.util.ArrayList;
import java.util.List;
import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Consumes;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;

@Path("xmlresource")
public class XmlResource {

    @Context
    UriInfo uriInfo;

    private static List<Todo> todoList = new ArrayList<Todo>() {{
        add(new Todo(1, "fill in my timecard"));
        add(new Todo(2, "say good bye"));
        add(new Todo(2, "go home"));
    }};

    @GET
    @Produces({ MediaType.APPLICATION_XML })
    public List<Todo> getTodoList() {
        System.out.println("getTodoList()");
        return todoList;
    }

    @PUT
    @Consumes({ MediaType.APPLICATION_XML })
    public Response addTodo(Todo todo) {
        System.out.println("addTodo(" + todo + ")");
        todoList.add(todo);
        return Response.created(uriInfo.getAbsolutePath()).build();
    }
}
```

Az XmlResource.java tartalmaz egy todoList nev listát, amihez rögtön adunk is három Todo.java objektumot. A listát a getTodoList metódussal olvashatjuk, és az addTodo metódussal adhatunk hozzá egy új elemet. A Path annotáció segítségével adhatjuk meg a REST szolgáltatás eléréséhez szükséges URL darabot, ami most xmlresource. A Path annotációt nem csak osztályszinten, hanem metódus szinten is használhatjuk.

A getTodoList metódus annotációi azt jelzik, hogy a metódust HTTP GET mvelettel hívhatjuk meg, és a metódus a HTTP GET-tel történ meghívás hatására XML formátumban adja vissza a todoList-et. A Produces annotációban több formátumot is megadhatunk, a Jersey a hívó által várt formátumot választja a felsoroltak közül.

Az addTodo metódus annotációi azt jelzik, hogy a metódust HTTP PUT mvelettel hívhatjuk meg, és a metódus XML formátumban várja a todo paramétert. A Consumes annotációban több formátumot is megadhatunk, a Jersey a hívó által küldött formátumot választja a felsoroltak közül. A metódus egy HTTP Created üzenettel tér vissza.

index.html:

Az index.html-ben található az alkalmazás kliens oldala, ami tartalmaz HTML, JavaScript és jQuery kódot. Összetettebb weboldal esetén célszer a JavaScript és jQuery kódokat kiszervezni a html kiterjesztés fájlokból js kiterjesztés fájlokba.

index.html

```
<html>
<head>
    <link rel="stylesheet" HREF="css/todo.css" TYPE="text/css">
    <script type="text/javascript" src="js/jquery-2.0.3.min.js"></script>
</head>
```

```
<body>
    <h2>TODO</h2>

    <div id="resultsDiv"></div>

    <form id="form1" action="#">
        <p>
            Add a new todo:
            <select id="prioritySelect"></select>
            <input id="taskInput" type="text" />
            <input id="submit1" type="submit" />
        </p>
    </form>

    <script type="text/javascript">

        // possible task priorities
        var priorities = {
            1: "1 Important",
            2: "2 Not so important",
            3: "3 Forget it"
        }

        // add priorities to the prioritySelect drop-down
        $.each(priorities, function(key, value) {
            $('#prioritySelect').append($("<option></option>").attr("value", key).text(value));
        });

        // this function is running when the page is loaded
        $(document).ready(function() {

            getResults();

            $("#form1").submit(function() {
                addTodo();
            });
        });

        // reads xml from server side, adds it to the div element as a table
        function getResults() {
            $.ajax({
                type: "GET",
                url: "rest/xmlresource",
                cache: false,
                dataType: "xml",
                success: function(xml) {
                    $("#resultsDiv").empty();
                    $("#resultsDiv").append("<table id='resultsTable'></table>");
                    $(xml).find("todoes").find("todo").each(function() {
                        $("#resultsTable").append(
                            "<tr><td>" +
                            priorities[$(this).find("priority").text()] +
                            "</td><td>" +
                            $(this).find("task").text() +
                            "</td></tr>"
                        );
                    });
                },
                error: showAjaxError
            });
        }

        // creates an xml of the form field contents, sends it to server side
        function addTodo() {
            // xml is case-sensitive
            var xml = "<todo><priority>" +
                $("#prioritySelect").val() +
                "</priority><task>" +
                $("#taskInput").val() + "</task></todo>";

            $.ajax({
```

```
                type: "PUT",
                url: "rest/xmlresource",
                dataType: "xml",
                contentType: "application/xml",
                data: xml,
                success: function() {
                    getResults();
                },
                error: showAjaxError
            });
        }

        // shows error response from server side
        function showAjaxError(xhr) {
            alert("Error\nstatus: " + xhr.status + "\nresponseText: " + xhr.responseText);
        }

    </script>

</body>
</html>
```

Az index.html elször beolvassa todo.css-t és a jQuery könyvtárat. A resultsDiv id-vel megjelölt div elembe kerül majd a HTTP GET hívás eredménye. A form1 id-vel jelölt form tartalmazza a HTTP PUT metódusnak átadandó adatok beviteli mezit. Ezután kezddik a JavaScript kód, ami elször a prioritySelect id-j select lenyíló lista kiválaszthaó értékeit definiálja, majd hozzáadja a select-hez. A JavaScript kódban a $ jel a jQuery könyvtár használatát jelöli. A # jel jQuery selector-okban használható egy adott id-vel rendelkez HTML elem kiválasztására, pl.: $('#prioritySelect').

A $(document).ready-ben található kód a teljes HTML oldal betöltése után fog meghívódni. A getResults metódus azonnal meghívódik az oldal betöltdését követen. Az addTodo metódus pedig a form submit-tálását követen hívódik meg.

A getResults metódus egy HTTP GET hívást küld a szerver oldal felé. Ha a hívás sikeres, elször kiüríti a resultsDiv id-j div elem tartalmát, majd beletesz egy táblázatot, amit a visszakapott XML-ben lév adatokkal tölt ki. A szerver oldalon a Jersey a List<Todo> típusból egy ilyesmi XML-t generál: <todos> <todo>...</todo> <todo>...</todo>... </todos>. A priorities-nél felhasználjuk a korábban létrehozott priorities JavaScript tömböt, hogy a számok helyett a select lenyílóban szerepl szövegek kerüljenek a táblázatba. Ha a HTTP GET hívás hibával tér vissza, meghívjuk a showAjaxError metódust.

Az addTodo metódus elször szöveg összefzéssel egy XML szöveget készít a létrehozandó todo bejegyzés adataiból. Majd egy HTTP PUT hívással ezt elküldi a szerver oldal felé. A hívásban az XML szöveget a "data: xml" sorban adjuk át. Siker esetén a getResults metódus segítségével frissítjük a táblázatot, hiba estén a showAjaxError hívódik meg. A showAjaxError megmutatja a kapott hiba status kódját és a szövegét.

todo.css

```
table {
    border: none;
    background: #FF0000;
    padding: 0px;
}
td {
    border: none;
    background: #FFF4F4;
    padding: 4px;
}
```

web.xml

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>jerseyxml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Az url-pattern érdekes számunkra, azt mondja, hogy JavaScript oldalon rest/* formában kell az AJAX hívások url-ét megadnunk.

A pom.xml-ben a jersey-quickstart-webapp -hoz képest csak az alkalmazás nevét írtam át jersey1-rl jerseyxml-re:

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>jerseyxml</groupId>
    <artifactId>jerseyxml</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>jerseyxml</name>

    <build>
        <finalName>jerseyxml</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

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

    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey.containers</groupId>
            <artifactId>jersey-container-servlet-core</artifactId>
            <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
            <!-- artifactId>jersey-container-servlet</artifactId -->
        </dependency>
        <!-- uncomment this to get JSON support
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-moxy</artifactId>
        </dependency>
        -->
    </dependencies>
    <properties>
        <jersey.version>2.3.1</jersey.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

A két telepít fájlban csak az alkalmazás nevét írtam át:

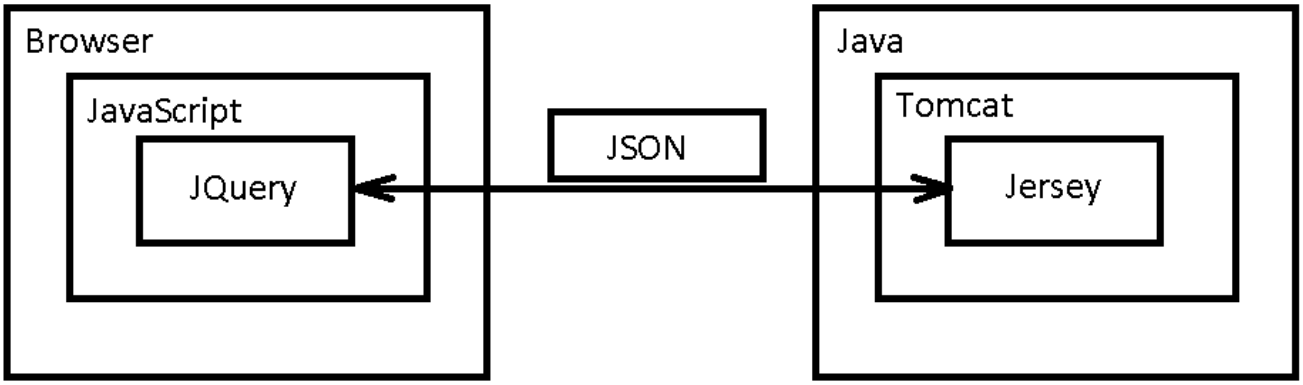
deploy.bat és undeploy.bat

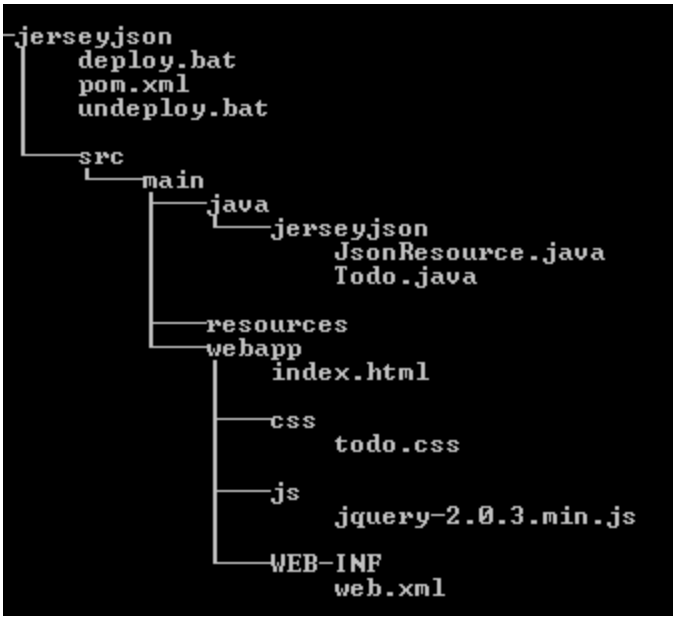
```
set APP_NAME=jerseyxml
```

JSON alapú üzenetküldés megvalósítása

Az alkalmazás felépítése:

Fájlok:





Todo.java

```
package jerseyjson;

public class Todo {

    private int priority;

    String task;

    public Todo() {
    }

    public Todo(int priority, String task) {
        setPriority(priority);
        setTask(task);
    }

    public int getPriority() {
        return priority;
    }

    public void setPriority(int priority) {
        this.priority = priority;
    }

    public String getTask() {
        return task;
    }

    public void setTask(String task) {
        this.task = task;
    }

    @Override
    public String toString() {
        return "Todo[" + priority + "," + task + "]";
    }
}
```

A Todo.java egy sima POJO (Plain Old Java Object), az üres konstruktoron kívül semmi különös megkötéssel.

JsonResource.java

```
package jerseyjson;

import java.util.ArrayList;
import java.util.List;
import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Consumes;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;

@Path("jsonresource")
public class JsonResource {

    @Context
    UriInfo uriInfo;

    private static List<Todo> todoList = new ArrayList<Todo>() {{
        add(new Todo(1, "fill in my timecard"));
        add(new Todo(2, "say good bye"));
        add(new Todo(2, "go home"));
    }};

    @GET
    @Produces({ MediaType.APPLICATION_JSON })
    public List<Todo> getTodoList() {
        System.out.println("getTodoList()");
        return todoList;
    }

    @PUT
    @Consumes({ MediaType.APPLICATION_JSON })
    public Response addTodo(Todo todo) {
        System.out.println("addTodo(" + todo + ")");
        todoList.add(todo);
        return Response.created(uriInfo.getAbsolutePath()).build();
    }
}
```

A JsonResource.java tartalmaz egy todoList nev listát, amihez rögtön adunk is három Todo.java objektumot. A listát a getTodoList metódussal olvashatjuk, és az addTodo metódussal adhatunk hozzá egy új elemet. A Path annotáció segítségével adhatjuk meg a REST szolgáltatás eléréséhez szükséges URL darabot, ami most jsonresource. A Path annotációt nem csak osztályszinten, hanem metódus szinten is használhatjuk.

A getTodoList metódus annotációi azt jelzik, hogy a metódust HTTP GET mvelettel hívhatjuk meg, és a metódus a HTTP GET-tel történ meghívás hatására JSON formátumban adja vissza a todoList-et. A Produces annotációban több formátumot is megadhatunk, a Jersey a hívó által várt formátumot választja a felsoroltak közül.

Az addTodo metódus annotációi azt jelzik, hogy a metódust HTTP PUT mvelettel hívhatjuk meg, és a metódus JSON formátumban várja a todo paramétert. A Consumes annotációban több formátumot is megadhatunk, a Jersey a hívó által küldött formátumot választja a felsoroltak közül. A metódus egy HTTP Created üzenettel tér vissza.

Az index.html-ben található az alkalmazás kliens oldala, ami tartalmaz HTML, JavaScript és jQuery kódot. Összetettebb weboldal esetén célszer a JavaScript és jQuery kódokat kiszervezni a html kiterjesztés fájlokból js kiterjesztés fájlokba.

index.html

```
<html>
<head>
    <link rel="stylesheet" HREF="css/todo.css" TYPE="text/css">
    <script type="text/javascript" src="js/jquery-2.0.3.min.js"></script>
</head>
<body>
    <h2>TODO</h2>
```

```
<div id="resultsDiv"></div>

<form id="form1" action="#">
    <p>
        Add a new todo:
        <select id="prioritySelect"></select>
        <input id="taskInput" type="text" />
        <input id="submit1" type="submit" />
    </p>
</form>

<script type="text/javascript">

    // possible task priorities
    var priorities = {
        1: "1 Important",
        2: "2 Not so important",
        3: "3 Forget it"
    }

    // add priorities to the prioritySelect drop-down
    $.each(priorities, function(key, value) {
        $('#prioritySelect').append($("<option></option>").attr("value", key).text(value));
    });

    // this function is running when the page is loaded
    $(document).ready(function() {

        getResults();

        $("#form1").submit(function() {
            addTodo();
        });
    });

    // reads json from server side, adds it to the div element as a table
    function getResults() {
        $.ajax({
            type: "GET",
            url: "rest/jsonresource",
            cache: false,
            dataType: "json",
            success: function(json) {
                $("#resultsDiv").empty();
                $("#resultsDiv").append("<table id='resultsTable'></table>");
                for (var i = 0; i < json.length; i++) {
                    $("#resultsTable").append(
                        "<tr><td>" +
                        priorities[json[i].priority] +
                        "</td><td>" +
                        json[i].task +
                        "</td></tr>"
                    );

                }
            },
            error: showAjaxError
        });
    }

    // creates an json of the form field contents, sends it to server side
    function addTodo() {
        var json = JSON.stringify({
            "priority": $("#prioritySelect").val(),
            "task": $("#taskInput").val()
        });

        $.ajax({
            type: "PUT",
            url: "rest/jsonresource",
            dataType: "json",
```

```
                contentType: "application/json",
                data: json,
                success: function() {
                    getResults();
                },
                error: function(xhr) {
                    if (201 == xhr.status) {
                        getResults();
                    } else {
                        showAjaxError(xhr);
                    }
                }
            });
        }

        // shows error response from server side
        function showAjaxError(xhr) {
            alert("Error\nstatus: " + xhr.status + "\nresponseText: " + xhr.responseText);
        }

    </script>

</body>
</html>
```

Az index.html elször beolvassa todo.css-t és a jQuery könyvtárat. A resultsDiv id-vel megjelölt div elembe kerül majd a HTTP GET hívás eredménye. A form1 id-vel jelölt form tartalmazza a HTTP PUT metódusnak átadandó adatok beviteli mezit. Ezután kezddik a JavaScript kód, ami elször a prioritySelect id-j select lenyíló lista kiválaszthatóértékeit definiálja, majd hozzáadja a select-hez. A JavaScript kódban a $ jel a jQuery könyvtár használatát jelöli. A # jel jQuery selector-okban használható egy adott id-vel rendelkez HTML elem kiválasztására, pl.: $('#prioritySelect').

A $(document).ready-ben található kód a teljes HTML oldal betöltése után fog meghívódni. A getResults metódus azonnal meghívódik az oldal betöltdését követen. Az addTodo metódus pedig a form submit-tálását követen hívódik meg. A getResults metódus egy HTTP GET hívást küld a szerver oldal felé. Ha a hívás sikeres, elször kiüríti a resultsDiv id-j div elem tartalmát, majd beletesz egy táblázatot, amit a visszakapott JSON-ban lév adatokkal tölt ki. A szerver oldalon a Jersey a List<Todo> típusból egy JSON objektumokból álló JSON tömböt generál. A JSON tömb és objektum a JavaScript-ben közvetlenül használható, implicit módon JavaScript tömb és objektum lesz belle. A priorities-nél felhasználjuk a korábban létrehozott priorities JavaScript tömböt, hogy a számok helyett a select lenyílóban szerepl szövegek kerüljenek a táblázatba. Ha a HTTP GET hívás hibával tér vissza, meghívjuk a showAjaxError metódust.

Az addTodo metódus elször a JSON.stringify parancs segítségével JSON szöveget készít a létrehozandó todo bejegyzés adataiból. Majd egy HTTP PUT hívással ezt elküldi a szerver oldal felé. A hívásban az JSON szöveget a „data: json” sorban adjuk át. Siker esetén a getResults metódus segítségével frissítjük a táblázatot, hiba estén a showAjaxError hívódik meg. A jQuery $.ajax hívás JSON adattípus esetén sajnos a szerver oldal által visszaadott 201-CREATED HTTP státuszt hibának tekinti, ezért külön vizsgálnunk kell a visszakapott hiba státusz kódját. A showAjaxError megmutatja a kapott hiba status kódját és a szövegét.

todo.css

```
table {
    border: none;
    background: #FF0000;
    padding: 0px;
}
td {
    border: none;
    background: #FFF4F4;
    padding: 4px;
}
```

web.xml

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>jerseyjson</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Az url-pattern érdekes számunkra, azt mondja, hogy JavaScript oldalon rest/* formában kell az AJAX hívások url-ét megadnunk. A pom.xml-ben a jersey-quickstart-webapp -hoz képest az alkalmazás nevét írtam át jersey1-rl jerseyjson-ra, és a Json-t aktiváló, eredetileg kikommentezett részrl levettem a kommenteket:

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>jerseyjson</groupId>
    <artifactId>jerseyjson</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>jerseyjson</name>

    <build>
        <finalName>jerseyjson</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

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

    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey.containers</groupId>
            <artifactId>jersey-container-servlet-core</artifactId>
            <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
            <!-- artifactId>jersey-container-servlet</artifactId -->
        </dependency>
        <!-- uncomment this to get JSON support -->
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-moxy</artifactId>
        </dependency>
    </dependencies>
    <properties>
        <jersey.version>2.3.1</jersey.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

A két telepít fájlban csak az alkalmazás nevét írtam át:

deploy.bat és undeploy.bat

```
set APP_NAME=jerseyjson
```

Irodalom

1. Maven: http://maven.apache.org/
2. Tomcat: http://tomcat.apache.org/
3. Jersey: https://jersey.java.net/
4. jQuery: http://jquery.com/
5. REST with Java (JAX-RS) using Jersey – Tutorial: http://www.vogella.com/articles/REST/article.html
6. XML Parsing with jQuery: http://tech.pro/tutorial/877/xml-parsing-with-jquery
7. Steve Suehring: JavaScript Step by Step 3rd Edition (Microsoft Press): http://www.amazon.co.uk/JavaScript-Step-3rd-Edition-Inside/dp/0735665931
8. Usage of JavaScript libraries for websites: http://w3techs.com/technologies/overview/javascript_library/all
9. RESTful services with jQuery and Java using JAX-RS and Jersey: http://coenraets.org/blog/2011/12/restful-services-with-jquery-and-java-using-jax-rs-and-jersey
10. Same-origin policy : http://en.wikipedia.org/wiki/Same-origin_policy