IT3030 Programming Applications and Frameworks

Practical 10 – Advanced Client-side Development 3 - RiWAs

In this lab sheet, you will learn how to use AJAX for the thick-client application you developed in practical 09 and convert it into a Rich Web-based Application.

This practical is based on an architectural style for the Rich Web-based Applications named the RiWAArch style, which is shown on the figure below.

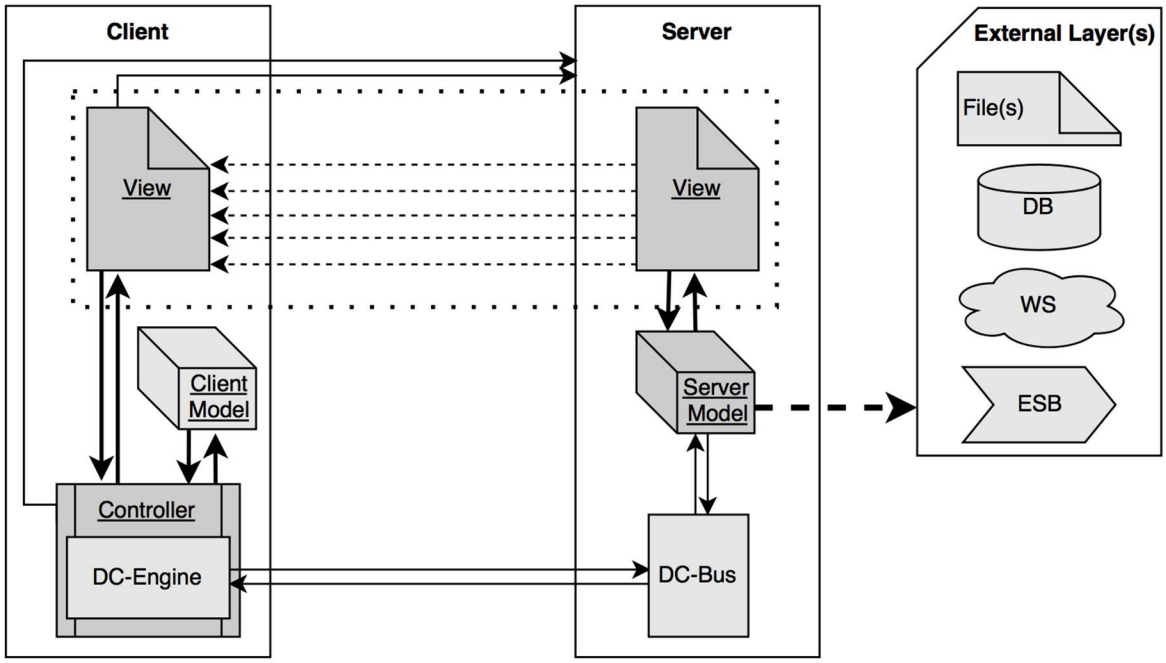


Figure 1: The RiWAArch style

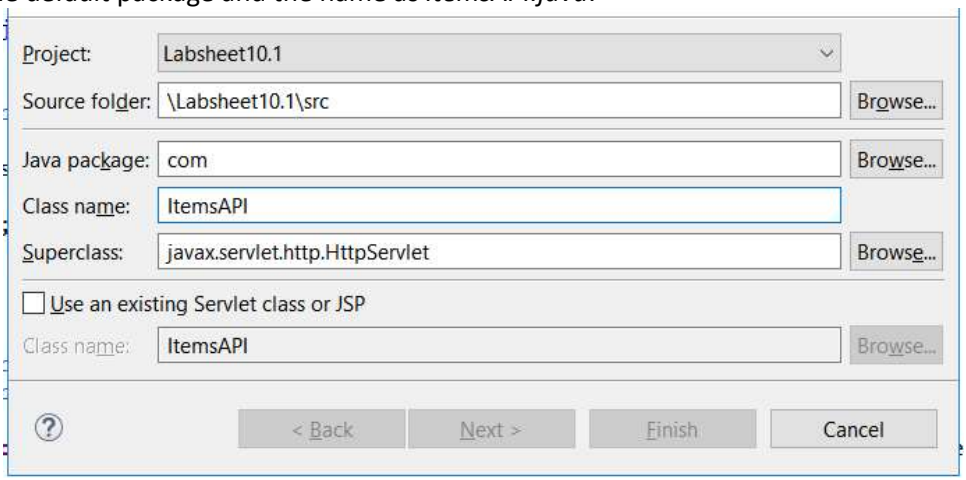
Section 1: Project Structure

It is recommended to get a copy of the completed project for the lab sheet 09.

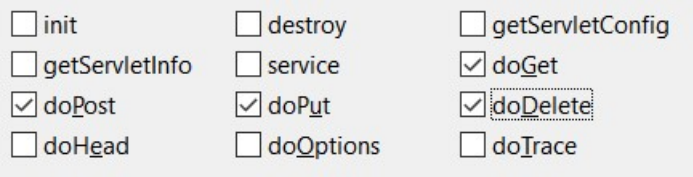
Step 1.1: Add a servlet

Let's add a servlet to the server-side, to act as a RESTful API for the rich client. The client can make AJAX calls to the server using this API.

Provide the default package and the name as *ItemsAPI.java*.

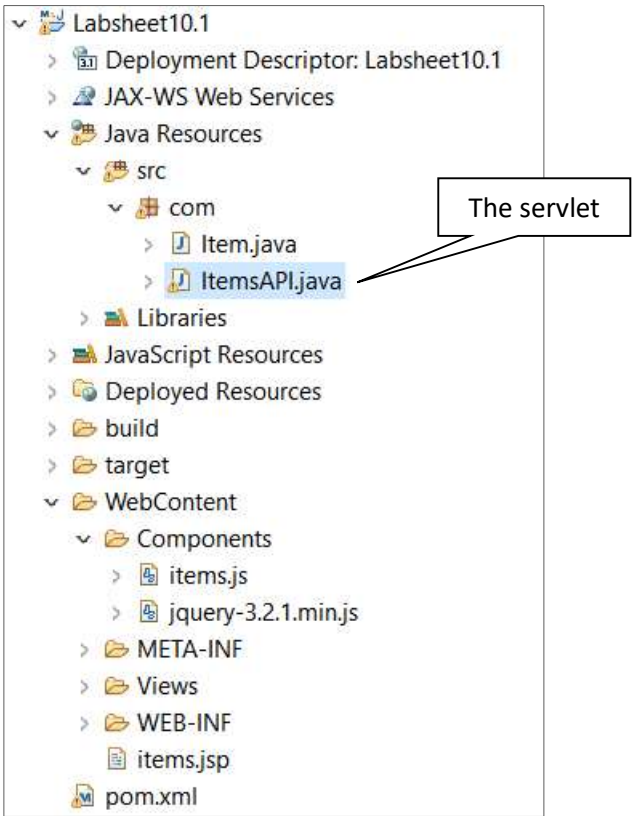


Go to the end of the wizard and make sure to tick all doGet, doPost, doPut, and doDelete options.



These methods will be triggered when the request uses the relevant form method.

The complete project structure is shown below.



We may need to update some code as discussed in the sections below.

Step 1.2: Update View

- Remove the java code for the connector in *items.jsp* page. We will not submit the forms using traditional form submissions, instead we will use AJAX calls.
- Remove the java code in *alertSuccess*. Since the page is not refreshed, we will not update the alert using server-side code.

The complete code of *items.jsp* will look like below.

```
<%@page import="com.Item"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Items Management</title>
<link rel="stylesheet" href="Views/bootstrap.min.css">
<script src="Components/jquery-3.2.1.min.js"></script>
<script src="Components/items.js"></script>
</head>
<body>

<div class="container"><div class="row"><div class="col-6">

	<h1>Items Management V10.1</h1>

	<form id="formItem" name="formItem">
		Item code:
		<input id="itemCode" name="itemCode" type="text"
			class="form-control form-control-sm">

		<br> Item name:
		<input id="itemName" name="itemName" type="text"
			class="form-control form-control-sm">

		<br> Item price:
		<input id="itemPrice" name="itemPrice" type="text"
			class="form-control form-control-sm">

		<br> Item description:
		<input id="itemDesc" name="itemDesc" type="text"
			class="form-control form-control-sm">

		<br>
		<input id="btnSave" name="btnSave" type="button" value="Save"
			class="btn btn-primary">
		<input type="hidden" id="hidItemIDSave"
			name="hidItemIDSave" value="">
	</form>

	<div id="alertSuccess" class="alert alert-success"></div>
	<div id="alertError" class="alert alert-danger"></div>

	<br>
	<div id="divItemsGrid">
		<%
			Item itemObj = new Item();
			out.print(itemObj.readItems());
		%>
	</div>

</div> </div> </div>
</body>
</html>
```

Step 1.3: Update the Server-model – readItems()

- Remove the hidden element in rows.
- Remove the form used for the Remove/Delete button and convert it into a regular button.
- Also give the remove button a class named `btnRemove` similar to update button.

NOTE: Delete operation will be handled by JS.

To store the *itemID* for the remove operation, let's use the data attribute of jQuery, with the attribute name `data-itemid`.

```
"<td><input name='btnRemove' type='button' value='Remove'
      class='btnRemove btn btn-danger' data-itemid='" + itemID + "'>" +
"</td>"
```

Do the same for the update button.

```
"<td><input name='btnUpdate' type='button' value='Update' "
+ "class='btnUpdate btn btn-secondary' data-itemid='" + itemID + "'></td>"
```

The complete updated code for the `readItems()` method is given below.

```
public String readItems()
{
	String output = "";

	try
	{
		Connection con = connect();

		if (con == null)
		{
			return "Error while connecting to the database for reading.";
		}

		// Prepare the html table to be displayed
		output = "<table border='1'><tr><th>Item Code</th>"
				+ "<th>Item Name</th><th>Item Price</th>"
				+ "<th>Item Description</th>"
				+ "<th>Update</th><th>Remove</th></tr>";

		String query = "select * from items";
		Statement stmt = con.createStatement();
		ResultSet rs = stmt.executeQuery(query);

		// iterate through the rows in the result set
		while (rs.next())
		{
			String itemID = Integer.toString(rs.getInt("itemID"));
			String itemCode = rs.getString("itemCode");
			String itemName = rs.getString("itemName");
			String itemPrice = Double.toString(rs.getDouble("itemPrice"));
			String itemDesc = rs.getString("itemDesc");
```

```
                    // Add into the html table
                    output += "<tr><td>" + itemCode + "</td>";
                    output += "<td>" + itemName + "</td>";
                    output += "<td>" + itemPrice + "</td>";
                    output += "<td>" + itemDesc + "</td>";


      // buttons
      output += "<td><input name='btnUpdate' type='button' value='Update' "
+ "class='btnUpdate btn btn-secondary' data-itemid='" + itemID + "'></td>"
+ "<td><input name='btnRemove' type='button' value='Remove' "
+ "class='btnRemove btn btn-danger' data-itemid='" + itemID + "'></td></tr>";
            }

            con.close();

            // Complete the html table
            output += "</table>";
      }
      catch (Exception e)
      {
            output = "Error while reading the items.";
            System.err.println(e.getMessage());
      }

      return output;
}
```

Step 1.4: Update server-model – insertItem()/updateItem()/deleteItem()

In the successful completion of these operations, we are going to submit the grid with the updated data. We have to handle the criteria below.

If (success) – send the grid with new/updated data
If (error) – send an error message

Let's use a JSON object to send the respond to the client. The structure of the JSON object will be the below.

```
{
      "status" : "<success/error>",
      "data" :  "<HTML for grid or error message>"
}
```

By the end of the try block of the *insertItem()* method, let's call the *readItems()* method and get the updated grid and embed the HTML into the JSON object.

```
String newItems = readItems();
output = "{\"status\":\"success\", \"data\": \"" + newItems + "\"}";
```

For the errors, let's modify the catch block as below.

```
catch (Exception e)
{
 output = "{\"status\":\"error\", \"data\": \"Error while inserting the item.\"}";
 System.err.println(e.getMessage());
}
```

Do the same for the *updateItem()* and *deleteItem()* methods.

The complete *Item* class is given below.

```
package com;

import java.sql.*;

public class Item
{
	private Connection connect()
	{
		Connection con = null;

		try
		{
			Class.forName("com.mysql.jdbc.Driver");
			con =
				DriverManager.getConnection(
				"jdbc:mysql://127.0.0.1:3306/test", "root", "");
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}

		return con;
	}

	public String readItems()
	{
		String output = "";

		try
		{
			Connection con = connect();

			if (con == null)
			{
				return "Error while connecting
						to the database for reading.";
			}

			// Prepare the html table to be displayed
			output = "<table border='1'><tr><th>Item Code</th>
				<th>Item Name</th><th>Item Price</th>"
				+ "<th>Item Description</th>
				<th>Update</th><th>Remove</th></tr>";




			String query = "select * from items";
			Statement stmt = con.createStatement();
			ResultSet rs = stmt.executeQuery(query);

			// iterate through the rows in the result set
			while (rs.next())
			{
				String itemID = Integer.toString(rs.getInt("itemID"));
				String itemCode = rs.getString("itemCode");
```

```
                        String itemName = rs.getString("itemName");
                        String itemPrice = Double.toString(
                                        rs.getDouble("itemPrice"));
                        String itemDesc = rs.getString("itemDesc");

                        // Add into the html table
                        output += "<tr><td><input id='hidItemIDUpdate'
                                        name='hidItemIDUpdate'
                                        type='hidden' value='" + itemID
                                        + "'>" + itemCode + "</td>";
                        output += "<td>" + itemName + "</td>";
                        output += "<td>" + itemPrice + "</td>";
                        output += "<td>" + itemDesc + "</td>";

                        // buttons
                        output += "<td><input name='btnUpdate'
                                        type='button' value='Update'
                                        class='btnUpdate btn btn-secondary'></td>"
                                        + "<td><input name='btnRemove'
                                        type='button' value='Remove'
                                        class='btnRemove btn btn-danger'
                                        data-itemid='"
                                        + itemID + "'>" + "</td></tr>";
                }

                con.close();

                // Complete the html table
                output += "</table>";
        }
        catch (Exception e)
        {
                output = "Error while reading the items.";
                System.err.println(e.getMessage());
        }

        return output;
}



public String insertItem(String code, String name,
                        String price, String desc)
{
        String output = "";

        try
        {
                Connection con = connect();

                if (con == null)
                {
                        return "Error while connecting
                                        to the database for inserting.";
                }

                // create a prepared statement
                String query = " insert into items
                (`itemID`,`itemCode`,`itemName`,`itemPrice`,`itemDesc`)"
```

```
                                + " values (?, ?, ?, ?, ?)";

                PreparedStatement preparedStmt = con.prepareStatement(query);

                // binding values
                preparedStmt.setInt(1, 0);
                preparedStmt.setString(2, code);
                preparedStmt.setString(3, name);
                preparedStmt.setDouble(4, Double.parseDouble(price));
                preparedStmt.setString(5, desc);

                // execute the statement
                preparedStmt.execute();
                con.close();

                String newItems = readItems();
                output = "{\"status\":\"success\", \"data\": \"" +
                            newItems + "\"}";
        }
        catch (Exception e)
        {
                output = "{\"status\":\"error\", \"data\":
                            \"Error while inserting the item.\"}";
                System.err.println(e.getMessage());
        }

        return output;
}



public String updateItem(String ID, String code, String name,
                                String price, String desc)
{
        String output = "";

        try
        {
                Connection con = connect();

                if (con == null)
                {
                        return "Error while connecting
                                    to the database for updating.";
                }

                // create a prepared statement
                String query = "UPDATE items SET
                itemCode=?,itemName=?,itemPrice=?,itemDesc=? WHERE itemID=?";

                PreparedStatement preparedStmt = con.prepareStatement(query);

                // binding values
                preparedStmt.setString(1, code);
                preparedStmt.setString(2, name);
                preparedStmt.setDouble(3, Double.parseDouble(price));
                preparedStmt.setString(4, desc);
                preparedStmt.setInt(5, Integer.parseInt(ID));
```

```
                // execute the statement
                preparedStmt.execute();
                con.close();

                String newItems = readItems();
                output = "{\"status\":\"success\", \"data\": \"" +
                        newItems + "\"}";
            }
            catch (Exception e)
            {
                output = "{\"status\":\"error\", \"data\":
                        \"Error while updating the item.\"}";
                System.err.println(e.getMessage());
            }

            return output;
        }

        public String deleteItem(String itemID)
        {
            String output = "";

            try
            {
                Connection con = connect();

                if (con == null)
                {
                    return "Error while connecting
                            to the database for deleting.";
                }

                // create a prepared statement
                String query = "delete from items where itemID=?";

                PreparedStatement preparedStmt = con.prepareStatement(query);

                // binding values
                preparedStmt.setInt(1, Integer.parseInt(itemID));

                // execute the statement
                preparedStmt.execute();
                con.close();

                String newItems = readItems();
                output = "{\"status\":\"success\", \"data\": \"" +
                        newItems + "\"}";
            }
            catch (Exception e)
            {
                output = "{\"status\":\"error\", \"data\":
                        \"Error while deleting the item.\"}";
                System.err.println(e.getMessage());
            }

            return output;
        }
}
```

Section 2: Implement Delta-Communication

The *ItemsAPI* servlet will act as a RESTful API, which implements the DC-Bus of the DC connector. The DC-engine will be implemented using jQuery's AJAX function.

Step 2.1: Set up the DC-Bus

Similar to RESTful web service, we can map a URL segment to a Servlet. Let's map `/ItemsAPI` to the *ItemsAPI* servlet.

```
@WebServlet("/ItemsAPI")
public class ItemsAPI extends HttpServlet
{
	//Your code
}
```

The grid is loaded to the View at the server at the initial loading time. Thereafter, the grid will be updated after Insert, Update, and Delete operations. Therefore, we won't use the *doGet()* method of the servlet.

We will use *doPost()* for Insert operation. For Insert operation, similar to web application, we will read the values passed by the client as an XHR request and pass them to the model by calling the *insertItem()* method in the *Item* class then return the output of the *insertItem()* method to the client.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException
{
	String output = itemObj.insertItem(request.getParameter("itemCode"),
                        request.getParameter("itemName"),
                        request.getParameter("itemPrice"),
                        request.getParameter("itemDesc"));

	response.getWriter().write(output);
}
```

To read the request parameters in *doPut*() and *doDelete*(), we will use a custom method. This method reads the request parameters, store them in a Map, and returns.

You may need the dependencies below to implement this method.

```
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
```

The code for the custom method is given below.

```
// Convert request parameters to a Map
private static Map getParasMap(HttpServletRequest request)
{
	Map<String, String> map = new HashMap<String, String>();
	try
	{
		Scanner scanner = new Scanner(request.getInputStream(), "UTF-8");
		String queryString = scanner.hasNext() ?
                            scanner.useDelimiter("\\A").next() : "";
		scanner.close();

		String[] params = queryString.split("&");
		for (String param : params)
		{
```

```
                    String[] p = param.split("=");
                    map.put(p[0], p[1]);
                }
        }
        catch (Exception e)
        {
        }
        return map;
}
```

Let's use the *getParasMap()* method in *doPut()* and *doDelete()* methods.

```
protected void doPut(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException
{
        Map paras = getParasMap(request);

        String output = itemObj.updateItem(paras.get("hidItemIDSave").toString(),
                            paras.get("itemCode").toString(),
                            paras.get("itemName").toString(),
                            paras.get("itemPrice").toString(),
                            paras.get("itemDesc").toString());

        response.getWriter().write(output);
}

protected void doDelete(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException
{
        Map paras = getParasMap(request);

        String output = itemObj.deleteItem(paras.get("itemID").toString());

        response.getWriter().write(output);
}
```

Step 2.2: Implement DC-Engine

Let's implement the elements of DC-engine in *items.js*.

The *Save* button works for both Insert and Update operations.
In the *Save* button's click event handler, after form validation, we have to check the value of *hidItemIDSave* to identify if it is insert or update and set the form method type of the Ajax communication.

```
var type = ($("#hidItemIDSave").val() == "") ? "POST" : "PUT";
```

Then we can perform Ajax communication using that type.

```
$.ajax(
{
        url : "ItemsAPI",
        type : type,
        data : $("#formItem").serialize(),
        dataType : "text",
        complete : function(response, status)
        {
                onItemSaveComplete(response.responseText, status);
        }
});
```

The complete code the save button is given below.

```
$(document).on("click", "#btnSave", function(event)
{
	// Clear alerts---------------------
	$("#alertSuccess").text("");
	$("#alertSuccess").hide();
	$("#alertError").text("");
	$("#alertError").hide();

	// Form validation-------------------
	var status = validateItemForm();
	if (status != true)
	{
		$("#alertError").text(status);
		$("#alertError").show();
		return;
	}

	// If valid-------------------------
	var type = ($("#hidItemIDSave").val() == "") ? "POST" : "PUT";

	$.ajax(
	{
		url : "ItemsAPI",
		type : type,
		data : $("#formItem").serialize(),
		dataType : "text",
		complete : function(response, status)
		{
			onItemSaveComplete(response.responseText, status);
		}
	});
});
```

NOTE: The URL is set to the URL segment mapped to the *ItemsAPI* servlet.

You also have to implement the complete function *onItemSaveComplete()*.

```
function onItemSaveComplete(response, status)
{
	//Your code
}
```

If the status is *success*, we can process the response. Since the server responses with JSON data object, we have to parse it.

In the case of *success*, we can obtain the data in the JSON object, which is the HTML for the updated grid, and set into the *divItemsGrid*.

If there is an error generated by the server logic, then we can display the error message in the relevant alert.

```
var resultSet = JSON.parse(response);

if (resultSet.status.trim() == "success")
{
	$("#alertSuccess").text("Successfully saved.");
	$("#alertSuccess").show();

	$("#divItemsGrid").html(resultSet.data);
} else if (resultSet.status.trim() == "error")
{
	$("#alertError").text(resultSet.data);
	$("#alertError").show();
}
```

For other error status, we can display a relevant error messages on alerts.

```
else if (status == "error")
{
	$("#alertError").text("Error while saving.");
	$("#alertError").show();
} else
{
	$("#alertError").text("Unknown error while saving..");
	$("#alertError").show();
}
```

Finally, we can reset the form.

```
$("#hidItemIDSave").val("");
$("#formItem")[0].reset();
```

The complete code is given below.

```
function onItemSaveComplete(response, status)
{
	if (status == "success")
	{
		var resultSet = JSON.parse(response);

		if (resultSet.status.trim() == "success")
		{
			$("#alertSuccess").text("Successfully saved.");
			$("#alertSuccess").show();

			$("#divItemsGrid").html(resultSet.data);
		} else if (resultSet.status.trim() == "error")
		{
			$("#alertError").text(resultSet.data);
			$("#alertError").show();
		}

	} else if (status == "error")
	{
		$("#alertError").text("Error while saving.");
		$("#alertError").show();
	} else
	{
		$("#alertError").text("Unknown error while saving..");
		$("#alertError").show();
	}
```

```
        $("#hidItemIDSave").val("");
        $("#formItem")[0].reset();
}
```

NOTE: You can run and test the Insert and Update operations now.

NOTE: You may need to update the event handler for the update button, to read from the data attribute.

```
$(document).on("click", ".btnUpdate", function(event)
{
        $("#hidItemIDSave").val($(this).data("itemid"));
        $("#itemCode").val($(this).closest("tr").find('td:eq(0)').text());
        $("#itemName").val($(this).closest("tr").find('td:eq(1)').text());
        $("#itemPrice").val($(this).closest("tr").find('td:eq(2)').text());
        $("#itemDesc").val($(this).closest("tr").find('td:eq(3)').text());
});
```

For the Delete operation, we can get the item ID from the data attribute of the Remove button.

```
$(document).on("click", ".btnRemove", function(event)
{
        $.ajax(
        {
                url : "ItemsAPI",
                type : "DELETE",
                data : "itemID=" + $(this).data("itemid"),
                dataType : "text",
                complete : function(response, status)
                {
                        onItemDeleteComplete(response.responseText, status);
                }
        });
});
```

The complete handler is similar to Save operation's complete handler.

```
function onItemDeleteComplete(response, status)
{
	if (status == "success")
	{
		var resultSet = JSON.parse(response);

		if (resultSet.status.trim() == "success")
		{
			$("#alertSuccess").text("Successfully deleted.");
			$("#alertSuccess").show();

			$("#divItemsGrid").html(resultSet.data);
		} else if (resultSet.status.trim() == "error")
		{
			$("#alertError").text(resultSet.data);
			$("#alertError").show();
		}

	} else if (status == "error")
	{
		$("#alertError").text("Error while deleting.");
		$("#alertError").show();
	} else
	{
		$("#alertError").text("Unknown error while deleting..");
		$("#alertError").show();
	}
}
```