UNIVERSITÀ DI PISA

AI-DII

Hands-on session 4:
Jakarta EE - Part 2 Servlets & JSP

Distributed Systems and Middleware Technologies

2023/2024

By José Luis Corcuera Bárcena

Agenda

1. Jakarta EE evolution
2. Creating a Maven Java Web Application
3. The web.xml file for Jakarta EE 10
4. JSP - Why?
5. JSP - scriptlet tag
6. JSP - expression tag
7. JSP - dynamic HTML generation
8. Servlet & JSP
9. Exercises

Jakarta EE evolution

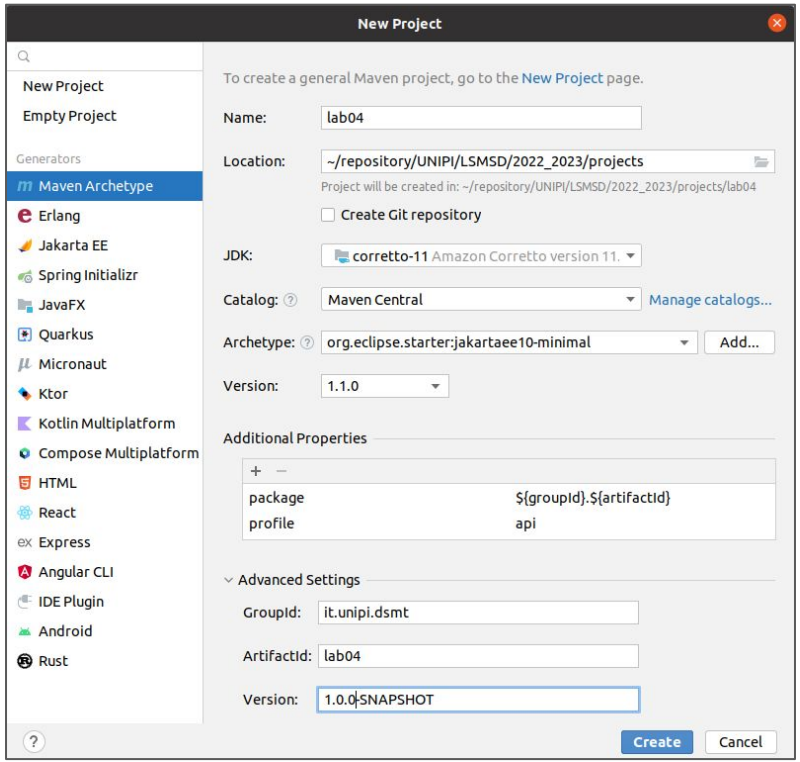
Java enterprise platform history

Platform version	Released	Specification	Java SE Support	Important Changes
Jakarta EE 10	2022-09-13[9]	10	Java SE 17 Java SE 11	Removal of deprecated items in Servlet, Faces, CDI and EJB (Entity Beans and Embeddable Container). CDI-Build Time.
Jakarta EE 9.1	2021-05-25[10]	9.1	Java SE 11 Java SE 8	JDK 11 support
Jakarta EE 9	2020-12-08[11]	9	Java SE 8	API namespace move from `javax` to `jakarta`
Jakarta EE 8	2019-09-10[12]	8	Java SE 8	Full compatibility with Java EE 8
Java EE 8	2017-08-31	JSR 366	Java SE 8	HTTP/2 and CDI based Security
Java EE 7	2013-05-28	JSR 342	Java SE 7	WebSocket, JSON and HTML5 support
Java EE 6	2009-12-10	JSR 316	Java SE 6	CDI managed Beans and REST
Java EE 5	2006-05-11	JSR 244	Java SE 5	Java annotations
J2EE 1.4	2003-11-11	JSR 151	J2SE 1.4	WS-I interoperable web services[13]
J2EE 1.3	2001-09-24	JSR 58	J2SE 1.3	Java connector architecture[14]
J2EE 1.2	1999-12-17	1.2	J2SE 1.2	Initial specification release

In this lab session, we are going to work with this version

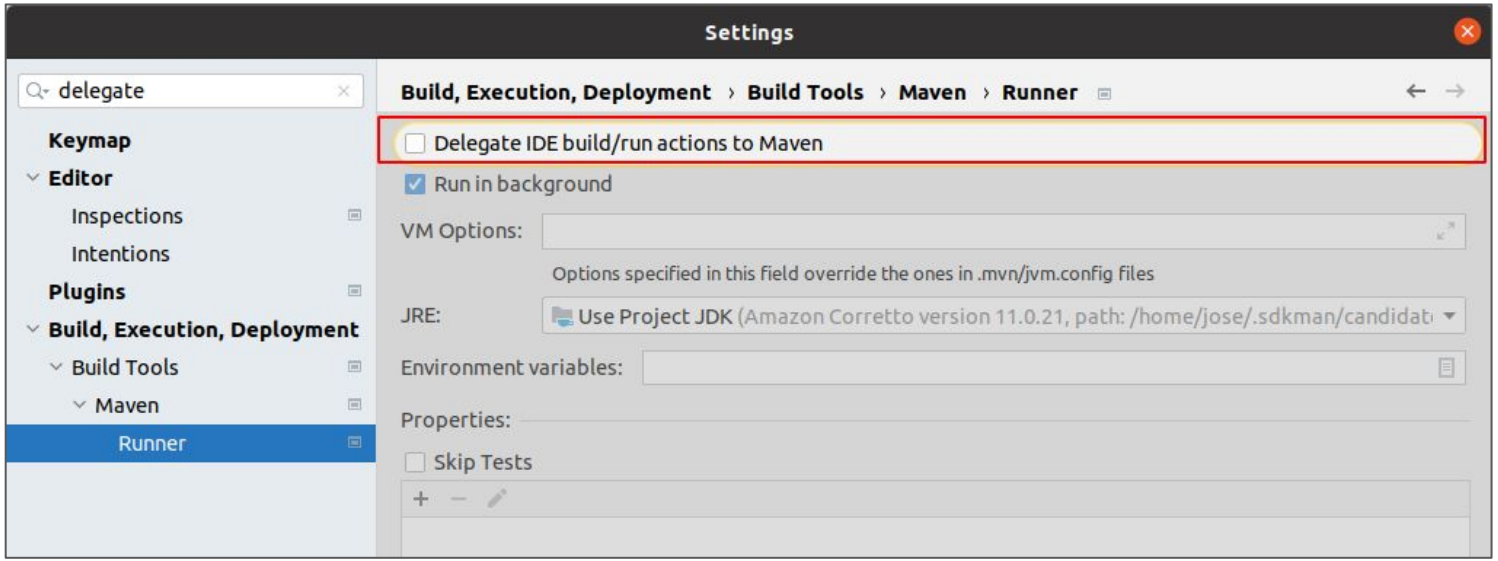
Source: https://en.wikipedia.org/wiki/Jakarta_EE

Creating a Maven Java Web Application (1)



- **Same steps as in the previous hands-on session**

Creating a Maven Java Web Application (2)



Do not forget to delegate build/run actions to Apache Maven

Creating a Maven Java Web Application (3)

Once your project is created, create a file **src/main/webapp/WEB-INF/web.xml** with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_6_0.xsd">

</web-app>
```

Servlet 6.0 Specification.

In addition to this, please add a new index.jsp file.

Jakarta Specifications for each version: https://en.wikipedia.org/wiki/Jakarta_EE#Web_profile

Creating a Maven Java Web Application (4)

Also, modify the file **pom.xml** by updating the maven compiler source/target to 11.

```
<dependencies>
   <dependency>
       <groupId>jakarta.platform</groupId>
       <artifactId>jakarta.jakartaee-api</artifactId>
       <version>10.0.0</version>
       <scope>provided</scope>
   </dependency>
</dependencies>
```

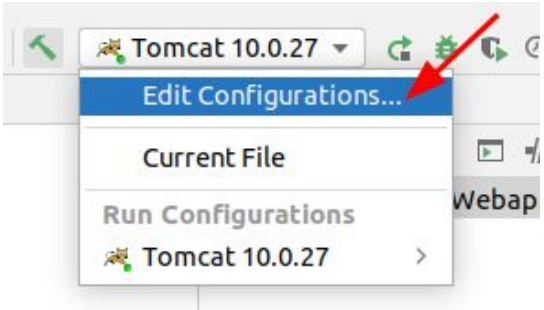
```
<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <maven.compiler.source>11</maven.compiler.source>
 <maven.compiler.target>11</maven.compiler.target>
</properties>
```

Running our first Maven Java Web Application

Please download Apache Tomcat 10.0.27. After downloading and uncomprezing Apache Tomcat 10.0.27 zip file, grant execution permission to all sh files inside the bin directory:

```
cd bin
chmod +x *.sh
```

Finally, create a “configuration” to run this new Web Application and make sure to register and select Apache Tomcat 10.0.27 with this configuration (same steps as we did in our previous hands-on session).



JSP - Why? (1)

So far in the previous exercises we generated the HTML content inside the Servlet code. Handling the HTML code generation at that level is difficult to maintain.

```java
@WebServlet(name = "CalculatorServlet", value = "/CalculatorServlet")
public class CalculatorServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int a = Integer.parseInt(request.getParameter("a"));
        int b = Integer.parseInt(request.getParameter("b"));
        String action = request.getParameter("action");
        double result = 0;
        switch(action){
            case "add":
                result = a + b;
                break;
            case "sub":
                result = a - b;
                break;
            case "mul":
                result = a * b;
                break;
            case "div":
                result = a * 1.0 / b;
                break;
        }
        StringBuilder html = new StringBuilder();
        html.append("<html>");
        html.append("<body>");
        html.append("<h1>");
        html.append("You sent: a = ").append(a).append(", b = ").append(b).append("<br>");
        html.append("Action: ").append(action).append("<br>");
        html.append("Result: ").append(result);
        html.append("</h1>");
        html.append("</body>");
        html.append("</html>");
        response.setContentType("text/html");
        response.getWriter().write(html.toString());
        response.getWriter().flush();
        response.getWriter().close();
    }
}
```

Small piece of HTML code generation.

JSP - Why? (2)

JAKARTA EE

JavaServer Pages technology allow us to combine Java code, HTML and JSP tags to generate HTML dynamically. An example of JSP:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <%
        String message = "Hello World!";
    %>
    <h1>Message from Java: <%= message %></h1>
</body>
</html>
```

JSP Tag

JSP scriptlet tag

JSP expression tag

JSP - scriptlet tag

This tag is used to include Java code inside a JSP file.

Each instruction is delimited by a ;

```
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="java.util.Date" %>
```

Begin

```
<%
    int n = 10;
    SimpleDateFormat SDF = new SimpleDateFormat("dd/MM/yyyy hh:mm:ss");
    String currentDateTime = SDF.format(new Date());
```

End

```
%>
```

JSP - expression tag

This tag is used to “**print**” a string value into the HTML content.

Begin End

```
<h1>Welcome, the current time is <%= currentDateTime %></h1>
```

There is no need to end the expression with ;
Very used operator: ternary operator -> condition ? expression1 : expression2

JSP - dynamic HTML generation (1)

We can combine Java code with HTML to generate content dynamically.

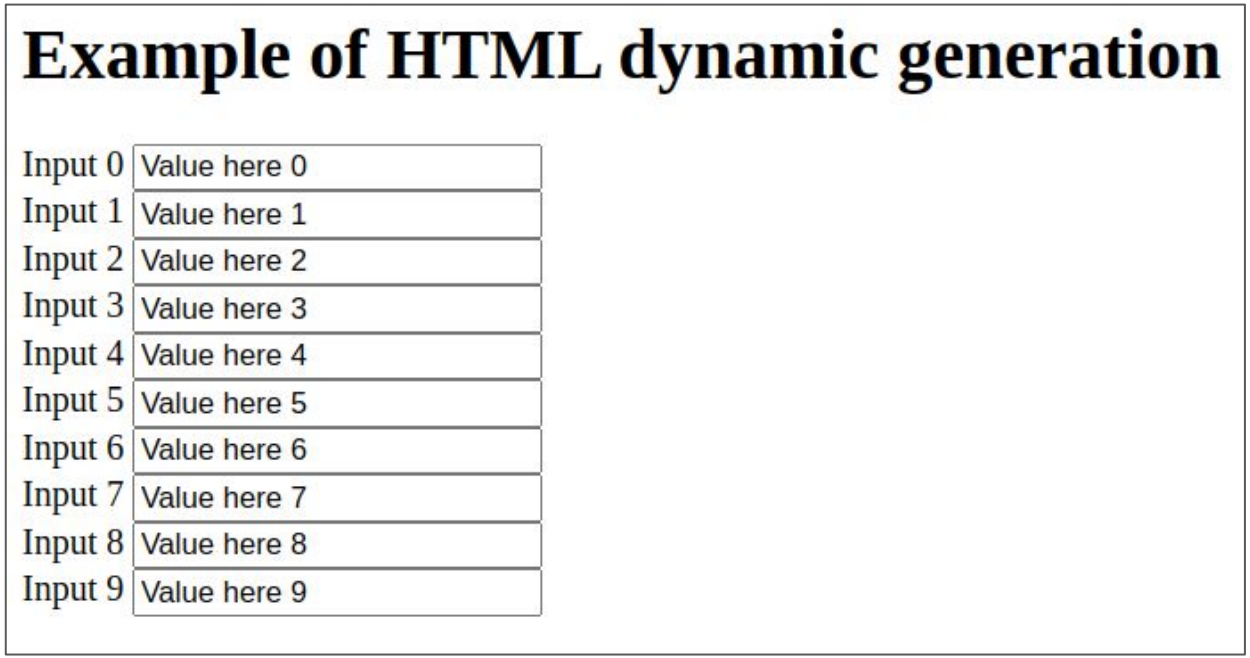
```
<%
   int n = 100;
   for (int i = 1; i < n; i++) {
%>
   <p> Current number = <%= i %>.</p>

<% } %>
```

JSP - Coding time! - 10 min

Creates a JSP that prints “**Benvenuto al corso di DSMT!**” when the minutes of the current clock is odd or “**¡Bienvenido al curso de DSMT!**” when it is even.

JSP - Coding time! - 10 min

Create a JSP page that prints N times an Input text as it is show in the next image:



JSP - Coding time! - 3 min

Modify the previous example to receive the number of times the HTML Input element is printed as HTTP request parameter. In case this value is not sent, print 10 times this element.

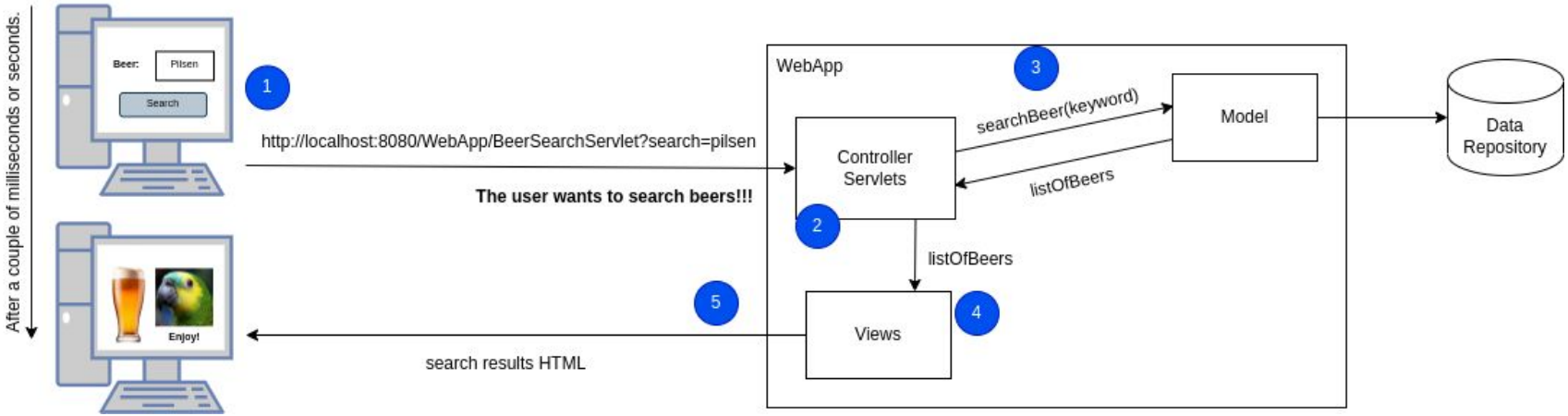
JSP - dynamic HTML generation (2)

We can hide/show content based on some conditions.

```
<%
   List<String> people = (List<String>)session.getAttribute("people");
   boolean notEmptyPeople = people != null && !people.isEmpty();
%>
<%
   if (notEmptyPeople) {
%>
   <p> There are <%= people.size() + "" %> people registered.</p>
<% } else { %>
   <p> No people found.</p>
<% } %>
```

Servlets & JSP (1)

In the MVC model, a request **(1)** is sent and received by a **Controller**. This is in charge of determining what functionality a user wants to execute **(2)**. Usually, it is required to perform some operations (i.e. CRUD) on the **Model (DAO, Entities, DTOs, etc.) (3)**. Finally, a **view** is used to generate the response **(4)** to be sent to the requester **(5)**.



Servlets & JSP (2)

So far we know that in the JSP, we have access to the following variables: request, session, response, application, pageContext, etc. Once a request arrives to the Controller, we can determine what functionality a user wants. In the code below, it is obvious that the user want to perform a search on beers. As next step, it is required to fetch the list of beers and forward that list to a JSP page in order to generate the final HTML to be sent as response.

```
@WebServlet(name = "BeerSearchServlet", value = "/BeerSearchServlet")
public class BeerSearchServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
        List<BeerDTO> beerDTOList = BeersDAO.search("");
        request.setAttribute("beers", beerDTOList);
        String targetJSP = "/WEB-INF/jsp/search.jsp";
        RequestDispatcher requestDispatcher = req.getRequestDispatcher(targetJSP);
        requestDispatcher.forward(req, resp);
    }

}
```

Getting list of beers.

Adding the list of beers in the request as "beers".

Forwarding the request to the search.jsp.

Servlets & JSP (3)

JAKARTA EE

In the JSP side, we just need to get from the request the list of beers and generate the HTML content by iterating that list.

```
<%
        List<BeerDTO> beerDTOList = (List<BeerDTO>) request.getAttribute( "beers");
%>
...

<% for(BeerDTO beer: beerDTOList) { %>
 <div class="col">
   <div class="card shadow-sm">
     <img class="product_img" src="<%= beer.getImageUrl()%>" />
     <div class="card-body">
       <p class="card-text"><%= beer.getName() %></p>
       <div class="d-flex justify-content-between align-items-center">
         <div class="btn-group">
           <button type="button" class="btn btn-sm btn-outline-secondary">View</button>
           <button type="button" class="btn btn-sm btn-outline-secondary">Add to cart</button>
         </div>
         <small class="text-muted">Rating: <%= beer.getRating() %></small>
       </div>
     </div>
   </div>
 </div>
 <% } %>
...
```

Retrieve the list of beers from the request.

Generating the HTML content.

Exercise 01: Search beer filter

Implement the logic to support the search beer functionality. Currently there exists a method in the model that allows filtering by a keyword, modify it. In case no results found, a message “No results found.” should be displayed in the web page.

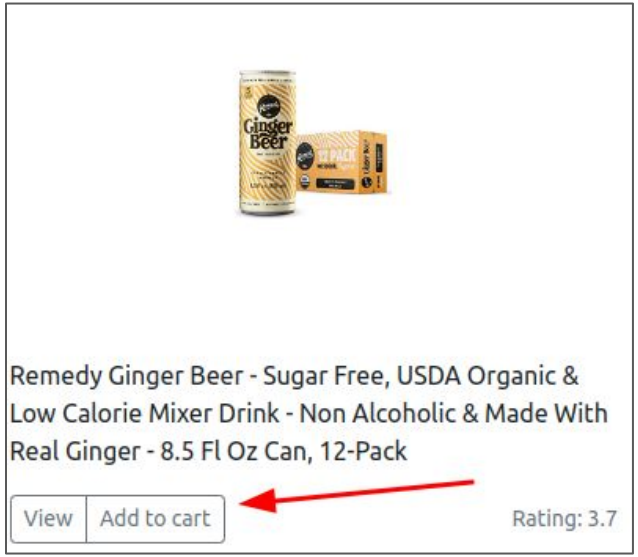
Exercise 02: Registering a new beer

Implement the logic to register a new beer. When a user clicks on the “Add new beer” button, a new web page is displayed and a user can enter the information of the following fields: name, imageUrl and rating. After submitting that information, the new beer is registered. Make the needed changes to complete this functionality.

Note that, this new beer will be displayed in the search page.

Exercise 03: Adding a beer to a shopping cart

Implement the logic to support the functionality of adding a beer to a shopping cart. When you click on the button “Add to cart”, one unit of that product is added. When the user navigates to the “Shopping cart web page”, all the added products must be displayed. Make the needed changes to complete this functionality.



References

- https://www.javatpoint.com/jsp-tutorial
- https://tomcat.apache.org/tomcat-10.0-doc/index.html
- https://stackoverflow.com/questions/72845243/the-cause-of-resubmit-form-in-browser