Hands-on session 3: Jakarta EE - Part 1 Servlets

Distributed Systems and Middleware Technologies

2023/2024

By José Luis Corcuera Bárcena

Agenda

1. Jakarta EE evolution
2. Required software
3. Creating our first Maven Java Web Application
4. Running our first Maven Java Web Application
5. Structure of a Maven Java Web Application
6. The web.xml file
7. ServletContextListener interface
8. Servlets
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

Apache Tomcat Versions

Apache Tomcat 11.0.x

Apache Tomcat 11.0.x is the current focus of development. It builds on Tomcat 10.1.x and implements the **Servlet 6.1, JSP 4.0, EL 6.0, WebSocket 2.2** and **Authentication 3.1** specifications (the versions required by Jakarta EE 11 platform).

Apache Tomcat 10.1.x

Apache Tomcat 10.1.x builds on Tomcat 10.0.x and implements the **Servlet 6.0, JSP 3.1, EL 5.0, WebSocket 2.1** and **Authentication 3.0** specifications (the versions required by Jakarta EE 10 platform).

Apache Tomcat 10.0.x

Apache Tomcat 10.0.x builds on Tomcat 9.0.x and implements the **Servlet 5.0, JSP 3.0, EL 4.0, WebSocket 2.0** and **Authentication 2.0** specifications (the versions required by Jakarta EE 9 platform).

Users of Tomcat 10.0 should be aware that Tomcat 10.0 has now reached end of life. Users of Tomcat 10.0.x should upgrade to Tomcat 10.1.x or later.

Apache Tomcat 9.x

Apache Tomcat 9.x builds on Tomcat 8.0.x and 8.5.x and implements the **Servlet 4.0, JSP 2.3, EL 3.0, WebSocket 1.1** and **JASPIC 1.1** specifications (the versions required by Java EE 8 platform). In addition to this, it includes the following significant improvements:

- Adds support for HTTP/2 (requires either running on Java 9 (since Apache Tomcat 9.0.0.M18) or the Tomcat Native library being installed)
- Adds support for using OpenSSL for TLS support with the JSSE connectors (NIO and NIO2)
- Adds support for TLS virtual hosting (SNI)

Source: https://tomcat.apache.org/whichversion.html

Required software

For this hands-on session, it is required:

- Apache Tomcat 10.1.x
- OpenJDK 11
- Apache Maven 3.x >=
- IntelliJ IDEA
- A good terminal program

Creating our first Maven Java Web Application (1)

Select the following values:

- JDK: 11
- Archetype: org.eclipse.starter:jakartaee10-minimal
- Version: 1.0.0

You can also generate your project by using this tool
https://start.jakarta.ee/

Creating our first Maven Java Web Application (2)

Once your project is created:

- Remove the content of the folder **src/main/java**

- Create the file **/lab03/src/main/webapp/WEB-INF/web.xml** with the following content:

```xml
<?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.**

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

Creating our first Maven Java Web Application (3)

Also, modify the file **/lab03/pom.xml** by updating the maven compiler source/target to 11.

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

Why provided?

Because Apache Tomcat already includes this dependency in the ${TOMCAT_BASE}/lib/servlet-api.jar file.

```xml
<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <maven.compiler.source>11</maven.compiler.source>
 <maven.compiler.target>11</maven.compiler.target>
</properties>
```
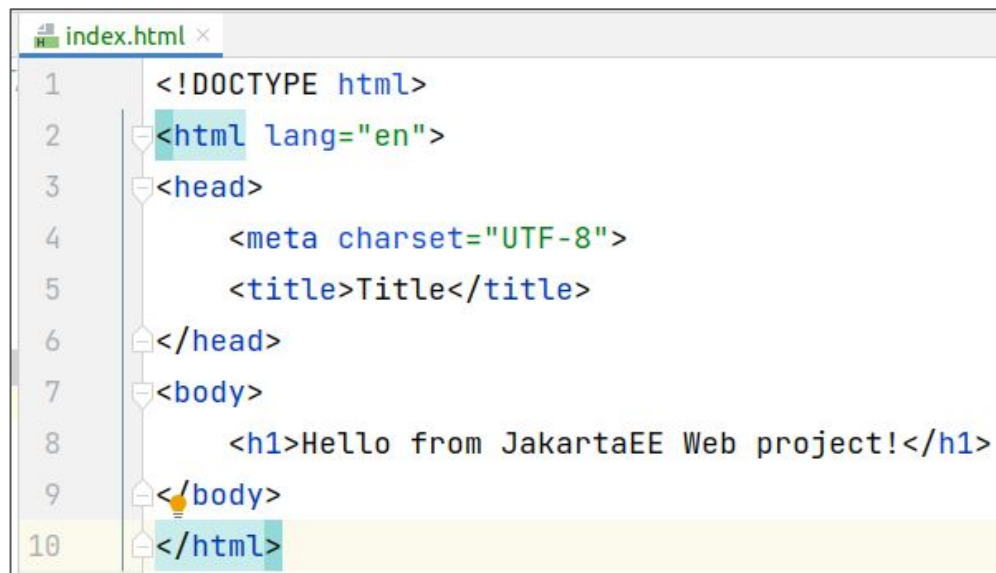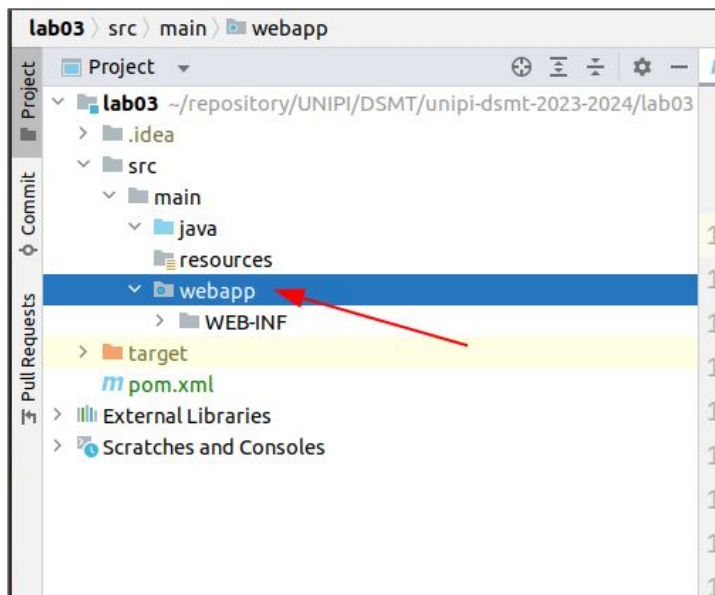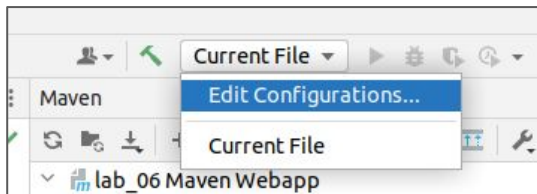
Creating our first Maven Java Web Application (4)

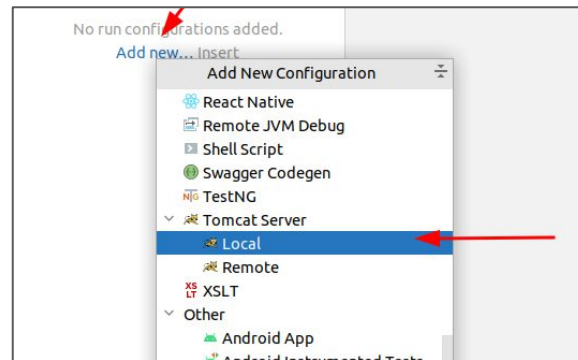Let's add an index.html web page

Running our first Maven Java Web Application (1)

To run our application, we have to create a **configuration**.

1. Select Edit Configurations

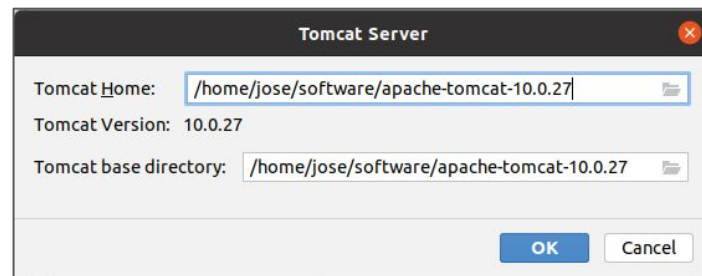2. In the opened window, click on "Add new…" and select Tomcat Server - Local.

Running our first Maven Java Web Application (2)

There is **no** Apache Tomcat configured so lets configure one.

1. Click on configure

2. Enter the same folder path on both input boxes.

Source: https://en.wikipedia.org/wiki/Jakarta_EE
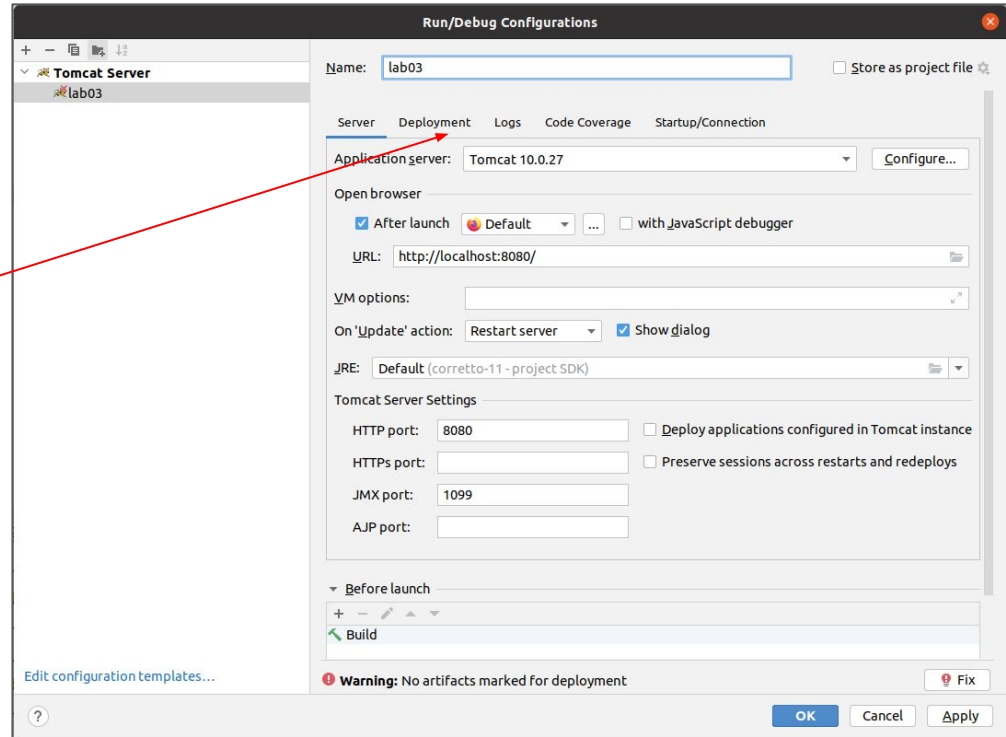
Running our first Maven Java Web Application (3)

After configuring Apache Tomcat, our Profile configuration looks like this:

As final step, we have to configure the deployment of our application. Click on the Deployment tab.
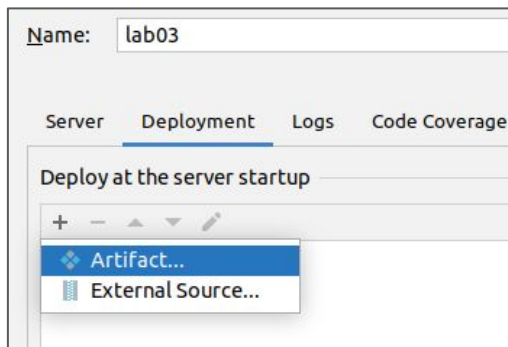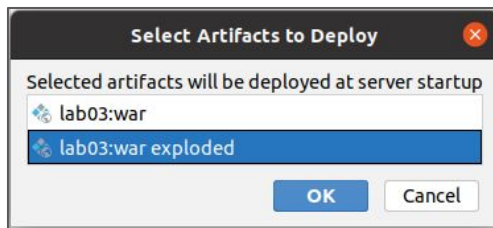
Running our first Maven Java Web Application (4)

In this Deployment tab, we have to select the exploded folder of our project.

1. Click on + and select "Artifact…"

2. Select lab_06:war_exploded.

3. By default, IntelliJ will append "_war_exploded" to the context of our application, get rid of it or change it.

Running our first Maven Java Web Application (5)

Go back to the Server tab and select the following values to enable hot updates when changes are made (_I prefer do the redeployment of the code when changes are made_) and click on OK to finish.

On 'Update' action:	Update classes and resources ▼
On frame deactivation:	Update classes and resources ▼
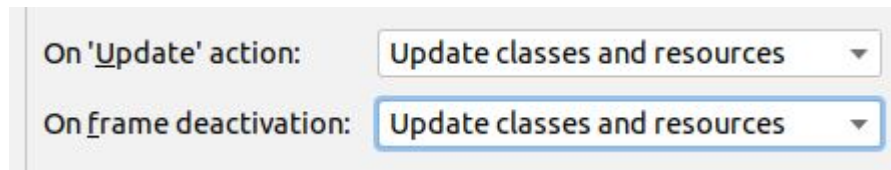
To run the project, click on the green arrow [🔧 lab03 ▼] ▶ . At this point, Apache Tomcat will start (in the image below the log is shown) and your application will be deployed.

Structure of a Maven Java Web Application

Three main folders:

- **java** - Java code goes here (Servlets, Filters, Listeners, POJOs, etc.).
- **resources** - Configuration/data files goes here (xml, properties, json, yml, etc.).
- **webapp** - Static resources goes here (jsp, html, css, js, etc.).

Resources defined inside the WEB-INF folder are not accessible.

The web.xml file (1)

- Web Application Deployment descriptor of your application.
- It defines listeners, servlets and their mapping, resources, error pages, etc.
- Thanks to Annotations, the amount of configuration lines here was reduced.
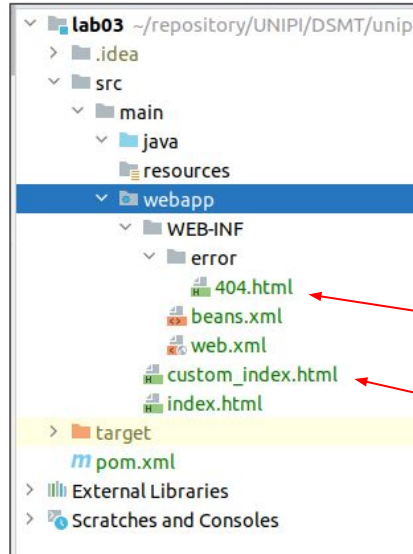

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="4.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_6_0.xsd">

  <display-name>DSMT: Lab 03</display-name>
  <error-page>
      <error-code>404</error-code>
      <location>/WEB-INF/error/404.html</location>
  </error-page>
  <welcome-file-list>
      <welcome-file>custom_index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

The web.xml file (2)

ServletContextListener interface

Sometimes, it is required to perform some actions at the startup of an Application (read configuration files, establishing connection to a database, calling a REST API, etc.). When the application/context is initialized the method contextInitialized is executed.

```java
package it.unipi.dsmt.javaee.lab03.conf ;

import jakarta.servlet. *;
import jakarta.servlet.annotation. *;

@WebListener
public class ApplicationListener implements ServletContextListener {

    public ApplicationListener () {
    }

    @Override
    public void contextInitialized (ServletContextEvent  sce) {
        System.out.println("App has been initialized." );
    }

    @Override
    public void contextDestroyed (ServletContextEvent  sce) {
        /* This method is called when the servlet Context is undeployed or Application Server shuts down. */
    }
}
```
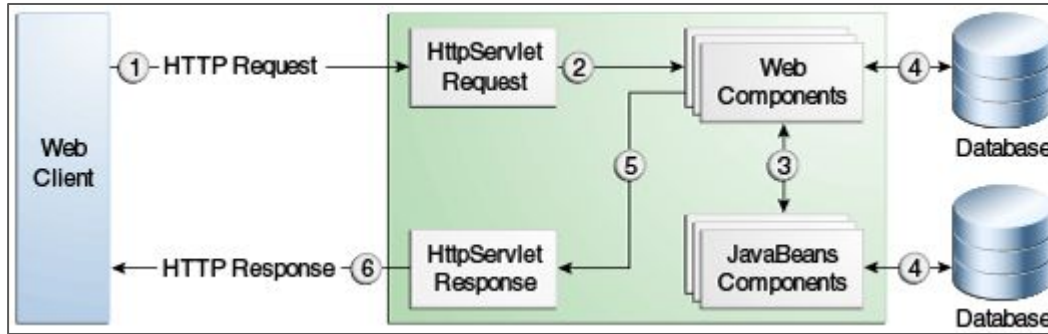
Here an implementation of this interface is implemented and annotated with @WebListener.

Servlets (1)

JAKARTA EE

A servlet is a Java programming language **class** used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

In this case:

Web Components = Servlets

Who implements these objects?

The Web Container = Apache Tomcat.

HTTP Request and Response are represented by **objects which implement the interfaces** jakarta.servlet.http.HttpServletRequest and javax.servlet.http.HttpServletResponse.

Image from: https://javaee.github.io/tutorial/webapp001.html

Servlets (2)

Depending on the HTTP Method used to send a request, we can have:

GET example:

http://localhost:8080/lab03/**HelloServlet**?**first-name**=Jose&**last-name**=Corcuera

ServletName

POST example:

```
<form action="http://localhost:8080/lab03/HelloServlet" method="POST">
    First Name: <input type = "text" name = "first-name"> <br/>
    Last Name: <input type = "text" name = "last-name" />
    <input type = "hidden" name = "action" value="action_hi"/>
    <input type = "submit" value = "Submit" />
</form>
```

Servlets (3)

On the Servlet side:

```java
package it.unipi.dsmt.javaee.lab03.servlets;


import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.io.IOException;

@WebServlet(name = "HelloServlet", value = "/HelloServlet")
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String firstName = request.getParameter("first-name");
        String lastName = request.getParameter("last-name");
        StringBuilder html = new StringBuilder();
        html.append("<html>").append("<body>");
        html.append("<h1>");
        html.append("Hi ").append(firstName).append(" ").append(lastName).append(" from doGet method.");
        html.append("</h1>");
        html.append("</body>").append("</html>");
        response.getWriter().write(html.toString());
        response.getWriter().flush();
        response.getWriter().close();
    }
}
```

Very important, do not forget to annotate your Servlet

URL path

This servlet only processes incoming GET requests.

The HttpServlet class allows to override the following methods: doGet, doPost, doDelete, doHead, doPut, doOptions and doTrace.

Servlets (4)

Code for doPost method:

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    String firstName = request.getParameter("first-name");
    String lastName = request.getParameter("last-name");
    String action = request.getParameter("action");
    StringBuilder html = new StringBuilder();
    html.append("<html>");
    html.append("<body>");
    html.append("<h1>");
    html.append("Hi ").append(firstName).append(" ").append(lastName).append(" from doPost method. <br>");
    html.append("Action: ").append(action);
    html.append("</h1>");
    html.append("</body>");
    html.append("</html>");
    response.getWriter().write(html.toString());
    response.getWriter().flush();
    response.getWriter().close();
}
```

Here, the value of action hidden input text is read.

The HttpServlet class allows to override the following methods: doGet, doPost, doDelete, doHead, doPut, doOptions and doTrace.
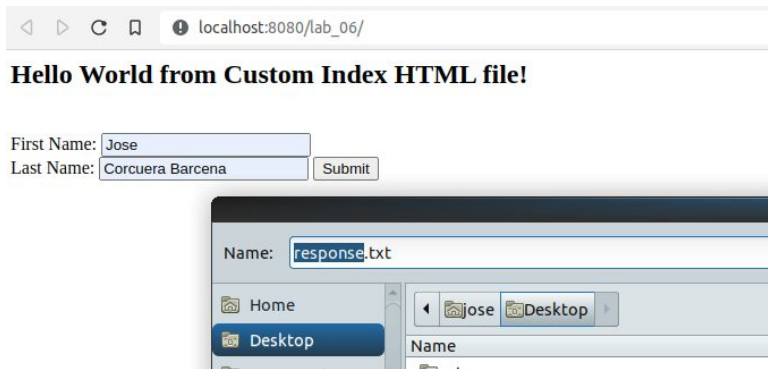
Servlets (5)

As a result:

Servlets (6)

As it was seen in the previous examples, we could write HTML on the response. In fact, we can write any type of content and set the content-type to be returned.

Example:

```
response.setContentType("plain/text");
response.setHeader("Content-disposition", "attachment; filename=response.txt");
```
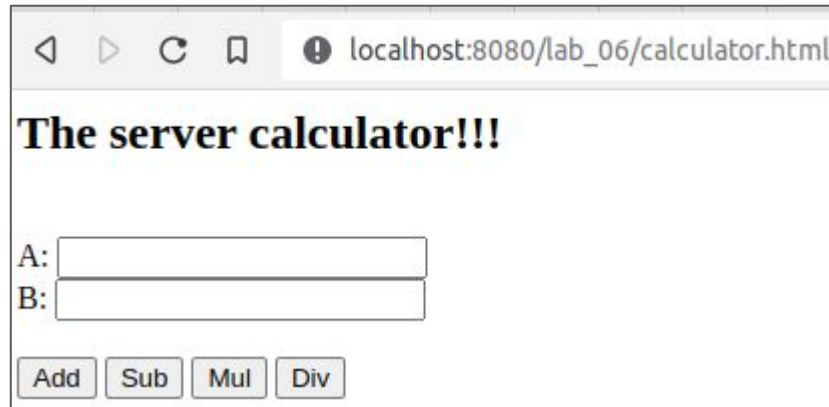
A full list of content type can be found here: https://www.geeksforgeeks.org/http-headers-content-type/

Exercise 01: The Calculator

Implement a HTML page that offers some basic functionalities of a calculator.

Below the design of this calculator. When a user click on one of the buttons Add,

Sub, Mul or Div, the data is sent to a Servlet which will display the result of the

operation.

Exercise 02: The Fortune Cookie Servlet

The **FortuneCookieServlet** is a Servlet that receives incoming HTTP GET calls and returns a random Fortune Cookie quote. The list of Fortune Cookie quotes are loaded once (when the context is created). The list of quotes can be found in the following in the file: **/lab03/src/main/resources/quotes.txt**

Exercise 03: Beers REST API

The **BeersRESTAPIServlet** is a Servlet that allows developers to search for beers. Incoming HTTP GET calls must include the "search" parameter. This parameter can be empty/null so in these cases, all beers are returned in JSON format. Instead, when a value is sent in the search parameter, only beers which contains that search value are going to be returned in JSON format too. Return only the name and image of each beer. The list of beers must be loaded once.

Beers JSON file: **/lab03/src/main/resources/beers.json**

Example of HTTP GET call: http://localhost:8080/lab_03/**BeersRESTAPIServlet**?**search**=ichnusa

You can make use of this tool to see the JSON structure: https://jsoncrack.com/editor

References

- https://javaee.github.io/tutorial/webapp001.html
- https://javaee.github.io/tutorial/webapp002.html
- https://javaee.github.io/tutorial/webapp004.html
- https://javaee.github.io/tutorial/webapp005.html
- https://javaee.github.io/tutorial/servlets.html
- https://www.jpassion.com/java-ee-programming/servlet-3-0-basics-java-ee-6
- https://www.jpassion.com/java-ee-programming/servlet-3-0-advanced-java-ee-6