Web Technologies - Lab 1

Credits to Mohamed Sellami

Contents

1 Java Servlet

1.1 Handling a POST request

1. Create a simple HTML page that sends a POST request to a servlet.
   - the POST request includes a first name, a last name and a birth date (mm/dd/yyy) (Use an HTML `<form>`).
   - the servlet retrieves these elements and displays them using an HTML table.

2. Update the (i) HTMl form by adding two radio buttons to specify the sex of a person and (ii) the displayed HTML table.

3. Customize your servlet's URL so it can be accessed by the following URLs:

   - http://localhost:8080/<project-name>/FormServlet
   - http://localhost:8080/<project-name>/Form_Servlet

1.2 Handling cookies

Cookies are text files stored on the client's computer. They store different information which can be used for tracking-purpose such as identifying a returned user. In this part, we will create, read and destroy cookies using Java servlet.

1. Modify the servlet from the previous part. The new servlet should add cookies for firstName, LastName, birthDay and Sex.

2. Test the new servlet from your favourite browser and check if the created cookies were created (go to your browser's parameters panel, confidentiality and look for the cookies tab).

3. Create a Servlet (called `ReadCookie`) that reads the four cookies and dispalys their values. Add a link under your HTML form to invoke the `ReadCookie` Servlet.(hint: doGet/doPost)

4. **Extra**: Create another servlet that count the number of visits using cookies.

1.3 Tracking HTTP sessions

Sessions are used to track client accesses and interactions. They answer user-access related questions, such as: How does a server know whether a client logged in? How does an on-line store keep tracking items that a user selected in his cart? In this part, we will see how to handle sessions using servlet.

1.3.1 Storing simple values.

Create a new Servlet `ShowSession` that:

- displays **Welcome on my site** for new comers and **Welcome back** for returning visitors.

- counts the number of visits.

Extra: Modify the `ShowSession` servlet to ask a user to input his [full name] and [date of birth] if he access the first time. If he is a returned user, compute the number of days to his birthday, and show him a message: "`Hi, [full name]. There are [number of days] days to your birthday.`"

1.3.2 Storing complex values.

- Create a new HTML file, name it `items.html` and put the following codes between the <body> tags :

```
<form action="ListItems" method="POST">
  Item: <input type="text" name="newItem"><p>
  <input type="submit">
</form>
```

- Create a new Servlet `ListItems` that stores the items passed from the form in the user's session (using an `ArrayList<String>`) and displays the content of this list each time the servlet is called.
- **Extra**: Modify the ListItems servlet to display also the number of times that an item was input.

2 Java Server Page (JSP)

2.1 Your first JSPs

Java Server Pages are web page documents that integrate Java codes to generate a dynamic content. In this first part, we will practice with the JSP expressions, scriptlets and declarations. Expressions are placed between <%= and %>, scriptlets are placed between <% and %> and declarations are placed between <%! and %>.

1. **JSP expression**. Create a JSP page (`expression.jsp`) that displays a random number (using `Math.random()`).
2. **JSP scriptlet**. Create a JSP page (`scriptlet.jsp`) that displays the following messages:
   - the current time (using `java.util.Date`),

- and the server's IP address (using `request.getRemoteHost()`).

3. **JSP declaration**. Create a JSP page (`declaration.jsp`) and insert the following codes between the <body> tags:

```
<%! private int numEntries = 10;
    private int randomInt(int range) {
    return(1 + ((int)(Math.random() * range)));
    } %>

<h1>A random list from 1 to 100:</h1>
<ul>
<% for(int i=0; i<numEntries; i++) {
out.println("<li>" + randomInt(100));
} %>
</ul>
```

2.2 Form, JSP and JavaBean

In this part, we will see how a JSP interacts with a JavaBean. This helps to easily program and flexibly manage data at the server side.

1. Modify the HTML file created in the first part to submit the request to a JSP file named `bean.jsp`. Save the HTML file as `form-jsp.html`.
2. Create a Java bean `Person` (its just a special particular Java class, see lecture 1) in a package `isep.lab1.bean`.
3. `Peerson` has a `firstName`, a `lastName` , a `birthDay` and a `sex`. Create these private properties (as `String` and using the same names as in the HTML form) and the associated getters and setters.
4. Create a JSP (`bean.jsp`) that gets the request from the HTML form and uses the bean `Person` to save the request's parameters using the bean and then **retrieves them from the bean** and displays them.

3 Model, View, Controller (MVC) architecture

In the MVC architecture, Java classes/Beans are developed as models, servlets act as controllers and JSP as views. In this part, we will develop a simple MVC application based on Java servlet and JSP.

3.1 Application description

In this part, we apply the MVC pattern to create a simple application that redirect users to adequate Web pages depending on their age:

1. A servlet (Controller) reads a person's information passed through an HTML form (View).
2. The controller uses these data to populate a bean called `Person` (Model).
3. The controller stores a reference to the bean in a request.
4. The controller forwards the request the appropriate JSP page:
    - If the person is born before 1990 to a dedicated page (View - JSP),
    - Otherwise (born after 1990) the request is forwarded to another page (View - JSP).
5. The JSP pages access the beans and display the person details
6. Launch eclipse and create a new **Dynamic Web Project**.
7. Follow the steps described below (Model, Controller and View) to develop this application.

3.2 Creating Model

1. Create a new package, name it lab.mvc.*model*.
2. In this package, create a Java bean `Person` representing your Model:
    - define these attributes: firstName, lastName , birthDay and sex.
    - define their getters and setters.
    - define a constructor with the four attributes.

3.3 Creating Controller

1. Create a new package, name it lab.mvc.*controller*.
2. In this package, create a Servlet and name it `Controller`.

3. We consider that our controller receives requests only from a form (will be developed later) and hence we implement only the doPost() method. Add the adequate code in the doPost() body such that the servlet:
    - reads a person's information passed through the HTML form,
    - uses these information to create an instance of the bean `Person`,
    - adds the created bean to the `request` (using `request.setAttribute(..,..)`),
    - forwards the request the appropriate JSP page using:

```
RequestDispatcher dispatcher=getServletContext().getRequestDispatcher("/jspPage.jsp");
  dispatcher.include(request, response);
```

3.4 Creating Views

In the **WebContent** folder, create two JSP pages `old.jsp` and `young.jsp` and one html page `index.html`.

1. Create the JSP pages to access the beans and display the person details:
    - The JSP page will not create objects, you should use <jsp:useBean ... **type** ="package.Class" /> instead of <jsp:useBean ... **class** ="package.Class" />.
    - The JSP page should not modify the objects, you should only use `jsp:getProperty` but not `jsp:setProperty`.
2. Create the HTML page that contains the form to submit a person's information.