Project 1. Provide a Computing Service

(Due 11/13/2022 Sun)

Description

In this project, you will practice some basic features of the servlet technology and the JSP technology. Since this is the first project of this semester, we use a simple structure for this web application. The structure of the web application is as follows:

- A JSP page is used for the user's interface (two versions: Windows and Mac).
- A servlet is used to provide the core computing service.
- Another JSP page is used to display the computation result.
- A preparation servlet may be needed to retrieve some initial data from the `web.xml`, so that the first JSP page can display it as a UI component.
- A JavaBean is used to send data to the front end.
- When you display individual pieces of data, use EL as much as you can.
- Use JSTL to display two dropdown lists using the data in the JavaBean.

The web application is used to run a business to sell a product in several cities. So the city information including the tax rates will be used to calculate the final prices of the product. The city information can be configured in an XML file, so that a non-Java programmer can change it easily.

Requirements

1. When the web application starts, the welcome file *redirects* the request to a servlet to prepare for the initial data. This welcome page does not display any data. It is only used to invoke the preparation servlet.
2. In the preparation servlet, you retrieve the user's operating system information from the request headers. Based on that information, you redirect the request to either Windows' version or Mac version of the input page.

   Note: If you detect that the user uses the Windows' system, then you treat the remaining cases as the Mac system; similarly, if you detect that the user uses the Mac system, you treat the remaining cases as the Windows' system.

3. In the preparation servlet, you retrieve the initial parameters from the `web.xml` for the city data and the tax rate data, so as to calculate the product price later. You use a JavaBean to store the city data and the tax rate data. Specifically, the city data is stored as a list of strings (**`List<String>`**); and the tax rate data is stored as an array of **`double`**s. Then you place the JavaBean in the request scope.
4. The user's input page will display two dropdown lists: one for the city options and the other for the tax rates for the corresponding cities, so that the user can select the suitable city for the product. You will use the JSTL technology to display the two dropdown lists.
5. (*Product Simulation*) In order to practice data processing in a servlet, we use a computation task which has its own value to simulate the product. The computation task is to factorize a big integer as the product of its prime factors with the following details:
    - You put a **`Generate`** button on the input page. When the user press this button, you call a JavaScript function to generate a random `100`-digit integer as a string displayed on the input page in an input field whose value cannot be editted. You also put a **`Decompose`** button on the input page. After the user clicks the **`Decompose`** button, the big integer as a string is sent to our data processing servlet for decomposition.
    - In the servlet, you can use Java's `BigInteger` class to do operations for big integers. (*An example for calculating Java BigIntegers is included with this project*.) The output is one of the two cases:
        1. If the input number `n` is a prime, just output a pair `(n,1)`, which means that a prime number is repeated only once;
        2. If the input number is not a prime, you need to do the prime factorization on this big integer. Although this `100`-digit integer is relatively big, we expect to complete the factorization in a reasonable amount of time.
        3. The format of decomposition can be written using the following example:

            ```
            63000 = (2,3)-(3,2)-(5,3)-(7,1),
            ```

            where for each pair `(p,m)`, `p` is a prime, and `m` means the exponent number for this prime.

    - Now we need to *simulate* the service cost. We simply take the last two digits of the largest factor in your factorization as the service cost, and the service charge (before adding the tax) is just that two-digit number of dollars.
6. (*Display Result*) You forward the page to another JSP to display the factorization result with the decomposition result stored as another JavaBean in the request scope.
7. (*Display Invoice*) Below the computation result, you also display the invoice of this computing task by adding the state tax on top of the service charge. (Note here we use the service to simulate the product.)
8. Put a **Home** link on the result page for the user to go back to the input page to purchase another product.

==========The End==========