Distributed Systems Project, Spring 2017

Assignment: Multitier architectures and the Web

In this assignment, you need to write a web-based calculator both in a server-based version and in a version where some of the functionality is migrated on the client side.

Tasks

1. Write a web-based calculator which uses a simple input form and performs all the calculation on the server side.
2. Migrate some of the functionality of the calculator to the client side using Javascript.
3. Implement caching of results on the client side.

Specifications

The server-based application should follow these specifications. The client-side implementation has three steps, which are defined below.

Write a simple web form which contains two text input fields, one selector for the operation (supported operations +, -, *, and /), and one button for submitting the calculation. The form must use the GET-method. The server will perform the operation and return a line with the calculation, for example "1 + 2 = 3", which the browser displays (along with a new input form). You need to keep the history of all calculations visible on the page. You are free to choose the mechanism for keeping the history.

NOTE: The names of the parameters MUST be arg1, arg2, and op, for the two arguments and the operator, respectively.

Step 1

You need to add functionality on the client side with Javascript as follows. Replace the input form with a single text input field and the submit button. The client-side script should parse the input given in the text field and convert it to the "atomic" operations supported by the server (+, -, *, and /). Precedence of operators goes from left to right, i.e., it is not the usual order of precedence.

Example: Suppose the input form has the text "1 + 2 * 3 / 4". Your script should break this up to 3 operations which are submitted individually to the server. Given the way the server works, all the intermediate results should also be shown on the browser. In this case the output looks like this:

1 + 2 = 3
3 * 3 = 9
9 / 4 = 2.25

Step 2

Add functionality to plot sine functions. The functions are given in the form of "a*sin(x)", and you need to plot the function from [-π, π]. You need to recognize multipliers in front of the function but no multipliers are used for x. You need to implement three variants of plotting. For plotting, use step size 0.1 (or smaller) on the x-axis.

In the first variant, you send the whole line to the server (you need to write a new server-side script to handle this), the server parses the line, plots it in a figure, and sends back an image file (e.g., PNG-format) which the browser displays. You can use for example gnuplot to generate the figure.

In the second variant, plot the figure locally on the client. For implementing the plotting, you are allowed to use the standard JQuery-library (see below for URL). You do not need to contact the server at all.

In the third variant, the server does the calculations and the client does the plotting. On the server side, you are allowed to use only the basic operations +, -, *, and /. The client must send the appropriate calculations to the server to get the approximately correct values to plot. Keep the maximum approximation error to 1% of the true value. You can/should modify the server from step 1 slightly to be more useful. There is no need to keep the history of operations visible on the client side. For implementing the plotting, you are allowed to use the JQuery-library (see below).

Step 3

Implement caching of results on the client side. The client should keep a cache of N most recent computations and their results and should be able to re-use those whenever needed in a subsequent calculation. The value of N, i.e., the cache size, should be configurable from the interface and value of 0 (zero) must be possible to enter and it should disable caching.

In addition, implement a button "Simplify", which processes the expression once and replaces any operation with a cached result if available. You should only implement one left-to-right pass for one pressing of "Simplify"; repeatedly pressing "Simplify" should process the simplified expression, i.e., eventually simplifying everything to the minimum calculations needed to be done on the server.

You should change the behavior of the "Submit" button such that it first does all the simplifications possible (repeatedly, if needed) and after every operation on the server it checks for possible simplifications. In other words, you should only send the minimum number of operations to the server in the final version. Play around with various cache sizes and see the effect they have on the number of operations in the third variant of step 2. In your document, plot the number of sent messages as a function of the cache size for at least 10 different cache sizes.

Grading

Completing all three steps is mandatory to pass the assignment. Weak implementations can and will receive lower grades even if they implement all steps.

Grading is based on correctly implemented functionality, documentation, and coding style.

Guidelines

The assignment is individual work. You can of course discuss any problems you encounter with other students, but sharing code is not allowed and if found, will be considered as plagiarism. Copying code from any source, other students or the net, is plagiarism.

Use users.cs.helsinki.fi for testing your scripts. Instructions can be found at http://www.cs.helsinki.fi/en/compfac/running-cgi-and-php-scripts-and-use-tomcat-containers. Alternatively, you can install your own web server on your own machine, but this is not recommended. Running own web servers on the normal department machines is NOT allowed.

No external libraries are allowed, unless otherwise specified in the description of the task. The JQuery library that you are allowed to use is version 1.12.4 (or earlier version in the 1.x branch). Note that use of for example, the flot library (http://www.flotcharts.org/) or other libraries is NOT allowed.

You can freely select the language in which you implement the code on the server side. You do not need to implement all steps with the same client or server code; including three versions in your return is ok.

Deliverables

Program source code with documentation and a document describing your implementation. Document should contain the plot of messages vs. cache size from step 3.

Timeline

The assignment is due on February 26th at 20:00. No extensions will be given.

Return

Store all the files in a directory that has same name as your username. Zip this directory, name the zip-file "username_DSP17.zip", and return the zip-file via Moodle. Please indicate clearly your name and student ID in every source code file.

Demo Session

You need to demonstrate your solution in person to the teachers in the course demo sessions on February 28 or March 2. You can demonstrate the solution either from your own laptop or in the computer lab at the department. You only need to come to one of the demo sessions and the demo sessions will be assigned later.