Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Claudius Korzen

Information Retrieval
WS 2017/2018

http://ad-wiki.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Exercise Sheet 7

Submit until Tuesday, December 12 at **12:00pm (noon)**

This exercise sheet (ES7) is about making the static web application from ES6 dynamic and dealing properly with encoding issues and make use of cookies.

If for whatever reason you did not do or complete ES6, you may use the master solution (which is available in Java and C++). In any case, do not link to or include code from other folders but copy everything to the *sheet-07* folder so that everything there is self-contained.

Exercise 2 (20 points)

Extend your web application from ES6 by the following components and functionality.

1. (4 points) Extend your server code from ES6 such that it can also serve JSON and JavaScript properly and such that for a GET request of form */api?q=<query>*, the matches for *<query>* are now returned as a valid JSON object. You should not use a library for this (it's easy to do it yourself).

2. (8 points) Make your web application dynamic using JavaScript and the jQuery library. The URL of the web page should be *http://<host>:<port>/search.html*, just like for ES6. The matches should now be displayed automatically after each keystroke (so that an explicit search button is no longer necessary), and they should be obtained via your own API from item 1. As for ES6, you should display up to 5 matches and you should display them in a nice way, using the additional info from the input file as you see fit. Take care that the host name and port are obtained automatically (from the URL) by your JavaScript, and are not hard-coded anywhere.

Alternatively, you can (but do not have to) let the matches appear dynamically in a nice drop-down box (below the search field), and when one of the matches is selected (with the arrows or with the mouse) do something that is related to that match (like showing information about that match or going to the page or whatever comes to your mind). The variant described in the first paragraph is slightly less work, the variant described in this paragraph is more like the typical search interface with suggestions which you find in most search engines today.

[please t%C3%B6rn over]

3. (4 points) Make sure that your web app deals properly with *all* characters in the query. The TIP file on the Wiki provides a number of test cases, which must work. This requires that you properly decode and encode URLs in your communication between the web app and the server. For example, *z%C3%BCrich* should be decoded to *zürich*. In your server code, choose a string representation such that the edit distance between any two characters (for example, *ü* and *u*) is 1. In Java, this is trivial when you use standard strings. In C++, you can use *std::wstring*.

4. (2 points) Add one feature to your web app that makes good use of cookies.

5. (2 points) Make sure that your web app is protected against code injection: there should be no way that the web app can be made to execute arbitrary code by typing something in the input field or feeding a modified input file to the server. Both of these were demonstrated by example in the lecture. In particular, make sure that the queries "*the mätrix*", "*nirwana*", "*gorilas*", "*harlem*", "*Mikrösoft Windos*", "*snow*", "*Turn Around*" and "*asteroids*" work as expected.

Add your code to a new sub-directory *sheet-07* of your folder in the course SVN, and commit it. Make sure that *compile*, *test*, and *checkstyle* run through without errors on Jenkins. Also commit the usual *experiences.txt* with your much appreciated brief and concise feedback.