# Information Retrieval

## WS 2013 / 2014

### Lecture 6, Tuesday November 26th, 2013

(How to build a web application)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

# Overview of this lecture

UNI FREIBURG

- Organizational
  - Your experiences with ES#5 (error-tolerant search)
- How to build a web application
  - Everything you need to know about:

    HTML, HTTP, MIME Types, Sockets, CSS, JavaScript, DOM, AJAX, JSON, jQuery, jQuery UI, CORS

    I will explain all these by the example of a toy web app, which we will build together live today

    Exercise Sheet 6: build a web app that displays error-tolerant prefix matches (ES5) as you type your query

# Experiences with ES#5 (error-tolerant search)

UNI FREIBURG

- Summary / excerpts last checked November 26, 16:00
  - Most: again time-consuming, but easier than Gollum
  - Many of the Java people had RAM problems

    Solution 1: ArrayList → Björn's dynamic int array

    Solution 2: work with only a part of the input
  - "Never program on a Friday"
  - Not clear how to implement multi-way union using PQ

    Code suggestions on the Forum + simpler alternatives
  - Gap-encoding the q-gram index reduces space to half
  - Would make sense to incorporate PED into ranking ... yes!

    e.g. for query uniwers rank uniwerse before universe

# How to build a web application

UNI FREIBURG

- Components
  - Server that delivers the web pages
  - Server that answers the queries
  - The contents of the web pages
  - The code that runs as part of the web pages and communicates with the server that answers queries
- Implementation
  - Many technologies behind this, each quite complex
  - But the basic principle behind each is easy to understand

  In the following, brief motivation + example for each

  Along with that we will code a toy web application **live**

# HTML = HyperText Markup Language

UNI FREIBURG

- Motivation
  - Language for specifying the content of a web page

    Including style information (layout, font, colors)

    Including code that dynamically changes the content
- Implementation
  - XML-like language, example tags:

| | |
|---|---|
| <h1>...</h1> | Level-1 heading |
| <p> ... <p> | A paragraph of text |
| <input> ... </input> | Input field |
| <script> ... </script> | Include JavaScript code |

# HTTP = Hypertext Transfer Protocol

UNI FREIBURG

- Motivation
  - How to transfer data between a web server (e.g. serving web pages) and a browser (e.g. requesting web pages)
- Implementation
  - Protocol for basic requests quite simple, e.g. HTTP **GET**

    Request line: GET /search.html HTTP/1.1 ...

    Answer line**s**:
    HTTP/1.1 200 OK
    Content-Length: 653
    Content-Type: text/html

    ... the 653 bytes of the content ...
  - There are MANY more request types and headers

    For ES6, just implement enough to make the browser happy

# Content Types aka MIME Types

UNI FREIBURG

- Motivation
  - Standard names for the different types of content sent across the internet
  - MIME = Multipurpose Internet Mail Extensions
- Examples
  - Plain text: text/plain
  - HTML: text/html
  - CSS: text/css
  - JavaScript: application/javascript
  - JSON: application/json

# Socket communication

UNI FREIBURG

- Motivation
  - Two programs communicating with each other
  - Possibly (and often) on two different machines
  - In particular: a program running as part of a web page with a program answering complex queries
- Implementation
  - Socket = machine + port ... different comm, different ports
  - In C++ easy with boost::asio (asio = asynchronous IO)
  - In Java easy with java.net.Socket / java.net.ServerSocket

  Code for socket communication is provided on the Wiki, in both Java and C++ ... yes, we are very nice

# CSS = Cascading Style Sheets

UNI FREIBURG

- Motivation
  - Specify style information (layout, font, color, etc) independent from the contents of the page
- Implementation
  - For example, all level-1 headings in blue and boldface

    h1 { color : blue; font-weight: bold }
  - Well-defined priority of rules, in case several of them apply to the same element of a page

    Often the case, hence the "cascading" in the name

    Intuitively: the most "specific" rule wins

# JavaScript

UNI FREIBURG

- Motivation
  - A language that runs as part of a web page

    For dynamically changing its contents (textual or otherwise) in response to user actions
- Implementation
  - Syntax similar to Java, hence the name
  - A script language = interpreted line by line = slow
  - Variables are untyped
  - Supports object orientation

# DOM = Document Object Model

UNI FREIBURG

- Motivation
  - Well-defined scheme for how to address elements in a web page, in particular by JavaScript code
- Implementation
  - For example: get the contents of an element with a particular id on the web page

    In the HTML:

    ```
    <div id="result">NO RESULT YET</div>
    ```

    In the JavaScript:

    ```
    document.getElementById("result").innerHTML = "42";
    ```

# AJAX

UNI FREIBURG

- Explanation
  - AJAX = Asynchronous JavaScript and XML
  - General name for communication between the JavaScript running within a browser and some server elsewhere, e.g.

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
   if (xhr.readyState == 4 && xhr.status == 200) {
      response = xhr.responseText;
      ... process the response ... }}
   xhr.open("GET", "<url>", true);
xhr.send();
```

  - In **jQuery** (see slide 14), this is a nice and simple two-liner

# JSON

UNI FREIBURG

- Motivation
  - The result from a computation is often a complex object, e.g. an array or associative array
  - If send as a mere string, we need code to parse that string on the JavaScript side
  - JSON is content in the form of ready-to-use JavaScript code
- Example
  - An associative array with two keys (from our toy web app)

    { "numVowels" : 5, "numConsonants" : 13 }

# jQuery

UNI FREIBURG

- Motivation
  - Raw JavaScript can be quite cumbersome and ugly
  - jQuery is a JavaScript library with convenient functions for all the common stuff
- Example: do something after each keypress
  - With raw JavaScript, you need something like this:

    HTML: `<input id="query" onkeypress="myFunction()"/>`

    JavaScript: `myFunction() { /* ... code here ... */ }`
  - With jQuery, nice separation of contents and code:

    HTML: `<input id="query">`

    JavaScript: `$("#query").keypress(function() { ... })`

# jQuery UI

UNI FREIBURG

- Motivation
  - Realizing (in particular: drawing) complex UI elements in raw JavaScript is (again) very cumbersome and ugly
  - jQuery UI is a library with convenient and easy-to-use functions for all the typical UI elements
- Example: autocompletion from fixed set of strings
  - HTML: `<input id="query">`
  - JavaScript:

```
$("query").autocomplete({
  source: [ ... array of strings from
                which to autocomplete ... ]
});
```

# Implementation Advice

UNI FREIBURG

- Debugging web apps is complex
  - Asynchronicity is a bitch
  - Use the "JavaScript console" from your browser

    Google Chrome: just type **Ctrl+Shift+J**

    Firefox: install the **Firebug** addon

    Internet Explorer: **don't** use it

    Very useful information about: which requests are launched when with which headers, responses and their headers, JavaScript errors, a console for logging, etc.

# References

- Relevant Wikipedia articles (in order of appearance)

  http://en.wikipedia.org/wiki/HTML

  http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

  http://en.wikipedia.org/wiki/Internet_media_type

  http://en.wikipedia.org/wiki/Network_socket

  http://en.wikipedia.org/wiki/Cascading_Style_Sheets

  http://www.w3schools.com/js

  http://en.wikipedia.org/wiki/Document_Object_Model

  http://en.wikipedia.org/wiki/Ajax_(programming)

  http://jquery.com/ http://jqueryui.com/