60-538 Project Phase 1 Overview

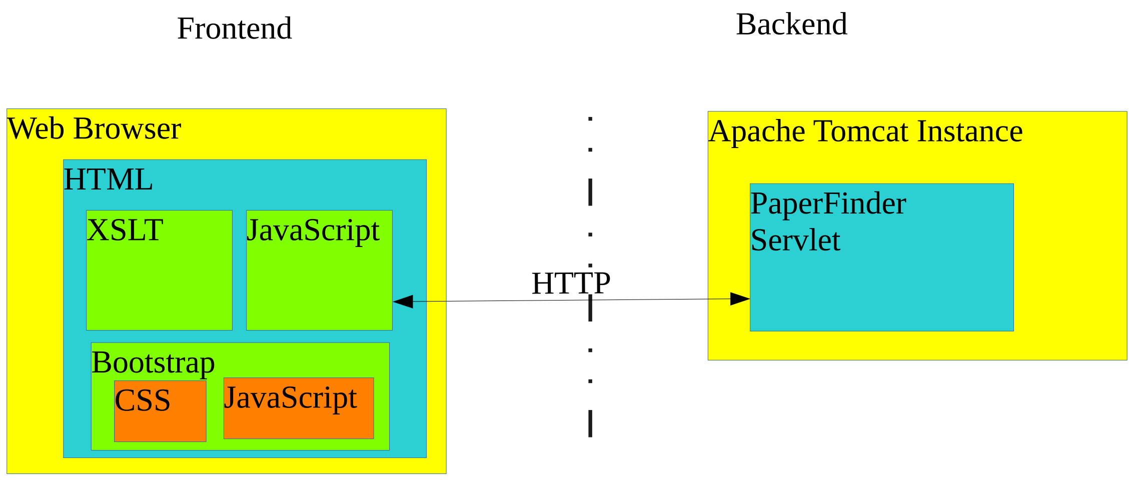
Shane Peelar

Project Name: PaperFinder

URL: http://speechweb2.cs.uwindsor.ca:8080/PaperFinder/

Architectural Overview

I chose to implement my search engine with a separate frontend and backend.



Backend

The backend of the search engine is a Java Servlet running on an Apache Tomcat instance. This servlet performs all communication over HTTP, and has identical behaviour for both GET and POST requests. It accepts one argument, “query”, which is expected to be the query string of the request. Apache Lucene is used here to fulfill query requests.

In order to improve query performance, the content index is only loaded once. This happens in the constructor of the servlet itself. Since the servlet is intended to be a long-running process, there is no need to manually re-parse the content index on each query request, and significant speed improvements can be obtained by caching the reader, searcher, analyzer, and query parser objects in this fashion. A simple check exists to ensure all objects were initialized correctly before attempting to fulfill any query request.

Results are returned in XML format using UTF-8. The XML format chosen is intended to be both easy to understand and also easy to extend. There are several reasons for this design choice:

1. Separation of Concerns: The backend is concerned only with efficiently obtaining query results and responding to queries in a timely manner. The frontend is responsible for formatting these results into a human readable form.
2. Extensibility: In the course, we will be incorporating multiple features into our search engine. These may require adding extra metadata to results. In the XML format chosen, additional result data is easily accommodated by adding extra children to each "result" tag.
3. Speed: Sending XML over the network is faster and less resource intensive than sending full blown HTTP, complete with Javascript, CSS, and other associated data
4. Scalability: Since the frontend and backend are separate, the infrastructure is in place to have multiple possible backend databases for queries. This could help later if we choose to add parallelism to the search engine
5. Compatibility: Using UTF-8 allows any character to be represented in the result lists. This is important because papers could potentially be written in any language, and the Latin character set is insufficient to represent characters in these languages.

As an example, this is a query that is sent directly to the backend:

http://speechweb2.cs.uwindsor.ca:8080/PaperFinder/PaperFinder?query=info

If you open that URL and click "view source" in your browser, you'll see the resultant XML that this backend request produces.

Frontend

The frontend is written entirely in HTML, XSLT, and Javascript. The Bootstrap library is used to style the user interface and provide flexibility across a variety of screen sizes. Essentially, the "index.html" file contained in the servlet is served to the client upon the initial connection, sending the necessary client logic over to the web browser. After this, the web browser takes over and becomes the frontend of the application. The "index.html" file could theoretically be hosted anywhere, but I've chosen to include it in the servlet itself for deployment convenience for now.

When queries responses are received, they are formatted as XML data. A few basic tags are defined, and have meanings as follows:

- search: The top level XML element that describes a set of search results and associated metadata
  - total: Expected to be a non-negative integer that describes the total number of search results. This is an example of including metadata.
  - results: The actual list of results
    - result: An individual query result
      - title: The title of the paper that the result references
      - path: The path relative to the servlet to where the paper is located. This can be used to obtain a URL to the paper itself.

The nesting in the above list indicates lineage: If a tag X1 is nested under another tag X2, then X1 is expected to be a child of X2 in the resultant XML.

An XSLT stylesheet is included in the HTML that describes how to transform these XML results into HTML, such that the results can be presented on the web page. A nice benefit of this approach is that queries do not involve a refresh of the page, which improves user experience and also lessens load on the server, as it does not have to potentially send the index.html file again for subsequent requests. Using XSLT here allows the web browser itself to decide how to best render the XML data rather than making these decisions on the server. In the future, this could become important if a wide variety of devices are to be supported in the frontend.

Additional features

* Search suggestions if no results

If no result is found, the user's query is compared against a dictionary and similar matches are suggested. This is implemented in Lucene with the SpellChecker module.

* Context shown

In addition to the titles being shown, context is shown as well about individual items in the search results. This is implemented using the Highlighter module.

* Pagination

Search results are displayed on different pages in a convenient interface

* Speed

Query objects are cached so that they are not regenerated on every request. This helps speed dramatically.

* XML – the context and search suggestion features are exposed in the XML output

Source code can be found here: http://speechweb2.cs.uwindsor.ca/~peelar/phase1source.zip

Next steps

The architecture presented here lays the groundwork for future work on the system. With separation of concerns being a design feature, it should be possible to easily incorporate modular code into the system, and also produce reusable code that other students could use in their own search engines. Next, I'd like to look at how to incorporate PageRank into this system, as well as provide a method to refresh the index on the backend in case the document corpus changes.