Final Report – Group 26

November 30, 2014

Andrew Huang
1000000937

Ben Kovarsky
999557849

Design

Given the limited time between Labs 3 and 4, we decide not to improve the Lab 3 algorithms, but to focus on user experience. We implement a spelling corrector so that if a user's query returns no results, we suggest a possible spelling correction in case a typo was made, as well as a hyperlink to the results of the new word. Furthermore, we add the title of each page to the table of results, so that the user is more informed as to which result fits their specific needs.

Testing Strategy

For the frontend, we decided that since the functionality was fairly straightforward, we would test it manually rather than use a testing suite. We would input various queries that the server would expect to receive, and then checked to ensure that it works as required. Afterwards, we would input various corner case searches. Corner cases were usually determined by the following question: "Can the server handle unexpected input?" For example, we examined how the frontend deal with empty searches, multi-word searches, searches with special symbols, etc.

For the backend, corner cases were determined by brainstorming anything that could happen outside of the expected circumstances. Examples included: database file does not exist, database is missing tables, database has extra tables, database queries for columns that don't exist, etc. We generated a few tests for each of these situations in our unit tester. We then checked to make sure that in the expected circumstance, the web crawler functions as it should. That is, results were stored in the correct tables with the correct pageranks. We generated a few tests for these situations as well. We then ran the unit tester to ensure that all tests passed.

Lessons Learned

At the beginning of the project, we discussed how we would code collaboratively. We decided to use Git and Github, which ended up being a wise choice. We learned valuable lessons about how to work on a single repository simultaneously, lessons which we will definitely carry with us after this course. We learned about the importance of keeping code as clean and modulated as possible; otherwise, adding and removing features becomes a disastrous undertaking. Beyond that, most of the lessons we learned were simply not to make our mistakes twice: deleting key-pairs, deleting repositories, crashing servers, etc.

Reflections

We are quite satisfied with the way our project turned out, and would probably not do anything differently. If we had more time, there are a couple ideas we would implement. We would modify the web crawler to run an iterative depth search, such that after it finishes a depth 1 crawl, it begins depth 2 and so on, indefinitely. This would allow us to run the crawler in the background and continually update our database. We would also implement multi-word query.

Finally, we would spend some time making the site more aesthetically pleasing, possibly by integrating Twitter Bootstrap with our frontend.

Portions of the projects whose implementation heavily depended upon the way Bottle itself worked (e.g. Google Login, Lab 1) took much longer than we anticipated, due to poor documentation.

Material from the course
Other than Python, the programming language taught in the course and used to develop the project, it seemed that there was no relationship between course content and this project. The programming paradigms taught had no impact on the way we went about developing the project.

Time to complete
The labs took roughly 10-15 hours to complete outside of lab sessions.

Project Feedback
We learned an immense amount doing the project. We would consider it a wonderful experience were it not an undertaking of incredible frustration. Many hours working on labs were spent in frustration trying to implement features where neither documentation nor working examples exist. Often times, the lab handouts were unhelpful, as was the case with implementing Google Login. We understand the need to develop the project in Python, since that language is the focus of the course. But that does not necessitate the use of Bottle, a nearly unknown framework with abysmal documentation. Python web development frameworks such as Tornado, Flask, and Django are ubiquitous and well-documented, and would save us a lot of angst. If Python is not a requirement, then teaching frameworks such as Express.js or Ruby on Rails would allow us to develop skills that are in high demand in the workplace, whereas we expect never to develop in Bottle again.

Course Feedback
We have no objections with the content of the course, though we think the course is currently a misnomer and should instead be named “Programming Paradigms”. Our main objections are with the way the course has been structured. The course has no systematic way of displaying marks to students. There is no central reference with which we can follow up on lecture material; lecture notes are limited, and the textbooks have no real relation to the course content. This makes it difficult to check and improve understanding. Midterms seem to test fringe concepts rather than core ideas.

Responsibilities of each member
Andrew was in charge of all frontend development and testing. Ben was in charge of all backend development and testing. Benchmarking was a joint effort. Neither group member felt that the workload was distributed unequally.