CSCC01 PHASE 4 REPORT

Team Chihuahuas (#16)

Table of Contents

State of the Project

With Phase 3 ending there was only two weeks left before the entire project was to be completed. While most of the important pieces were there, there was still a lot more work that was needed to meet the user's satisfaction.

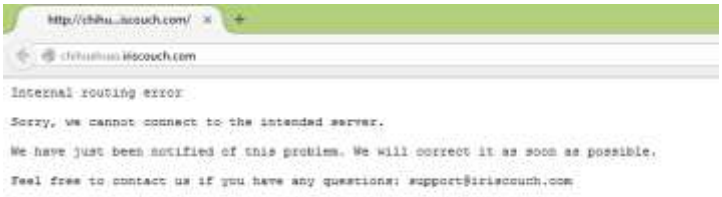
With the feed crawl already completed from the previous phase, our next priority was to create a Twitter crawl. While the processing of the data was generally easy, actually getting the tweets proved harder than expected. For one, we were only allowed to extract the 200 most recent tweets, and there was also a timeout when the extract limit was exceeded. There was a quick workaround of creating more accounts to be able to extract, but in the end we settled for extract less tweets.

While the feed crawl is useful for its simplicity to crawl (just looking for link tags), the user would like to crawl an arbitrary page. This prompted the creation of ArticleCrawler, which is able to look for hyperlinks and recursively call itself to continue the crawl. However, this can end up in an infinite loop if it calls the same header over again, so we used a Python library boilerpipe, which is a Java implementation wrapped in Python. With boilerpipe, we were able to extract only the article text/HTML of the page, which reduced the number links needed to crawl to a reasonable amount.

The website also needed to be redesigned for simplicity and usability. Aside from the login page, it was divided into three sections:

1) Keywords: All our crawls use the same list of keywords that determine if the article/tweets would be stored from the database.
2) Web Crawl: Feed Crawl and Article Crawl were grouped together since the results for both crawls are save to the same database.
3) Twitter Crawl: Twitter crawl has a different set up from the webcrawl, and its results and process in real time.

With the deadline coming closer and closer, a few changes were made to the release plan. First, the items that were originally planned for Week 8 were ultimately abandoned as their priority was not as high as the others. We did not have the time to implement permissions, so it was not possible to do User Story 2, where the grad student could request a list of sources to the researcher to add. Most of the graphs were created in this phase, as well as minor look-and-feel changes to the website. We could not create a WARC file for the librarian as the specification was complex, so we settled on exporting the results table into an XML file.



We encountered a big issue in regards to our database in that the server where we host it encountered an internal server error. This issue only started in the afternoon of Wednesday, November 26th, a few hours before we were to submit the final product. After a few hours, we came up with a solution to host our database to a different server, but in the middle of migrating (under branch db_issue), our database went back online. It seemed to be a onetime issue until it happened again on the day of the presentation. There was a backup database, but it wasn't fully operational, there were changes from our master branch that was not implemented in this version and the database was not fully setup. Our team decided that it was not a good idea to

present a partial product. As of writing this report, our original database is offline. It seems the best time for access is in the morning.

If we were given more time on this project, we would mostly likely start incorporating the user stories that we abandoned, as well as creating more insightful graphs that could help the user. The graphs we currently have implemented are very basic at best (it only shows the count of hyperlinks/quotes from a given article).

Looking back from this project, we would definitely do some things differently, if we were to start over.

- Start the website from the backend. For this project we started from the front end, so when the time came to incorporate the backend we needed to migrate all the current pages and the functionality.
- Creating the system model and the UML class diagram would have been done in the beginning before the implementation was to begin, not the other way around.
- Code reviews should be done more frequently as more members of our team can understand as much code as possible.
- Experiment with more libraries/find different options to solve our tasks (e.g. How to extract hyperlinks/quotes/mentions). Due to limited time, the research/testing the libraries was not very thorough and the task was either done by ourselves or through the very available library. This could explain the fact that we needed to install a new library to mathlab every week.
- Or, instead of experimenting, we could use a familiar language/library that the team knows and sticks with it. While it is a great programming experience to use more technologies, our project has a tight deadline and time would not be well spent trying to determine how the library worked.
- More pair programming. This way changing a particular functionality would not depend on only one person alone.

Release Plan

Please see our Trello Board ("CSCC01 Team 16: Release Plan"). If you do not have access to it, please contact one of the team members. Each week contains the user stories to be implemented for that week.

Iteration Plan

The iteration plans for week 7 to week 8 are included in the master branch in "Project Management/Iteration Planning"

Task Board

Please see our Trello page ("CSCC01 Team 16 Task Board") for the Task Board. If you do not have access to it, please contact one of the team members. The instructions to understand the task board are in the first list by the name "*How to understand the Task Board (Click on each card to see the description)*"

Burndown Chart

The burndown charts for all the weeks in included in Burndown_Charts.xlsx at the same place as the iteration plans i.e. "Project Management/Iteration Planning"

Code Reviews

Code Reviews from each team member is located in "Project Management/Code Reviews". All the reviews follow the format mentioned in the handout.

Code Inspection video is located in the same folder by the name Code_Review.mov

Testing

The unit tests and selenium tests for the backend and frontend are included in the "flask" branch. The instructions on how to run these have been included in the README.

For unit tests, a new test suite **GetTweetsTestCase** was created to test whether the correct JSON data was sent to the front end to generate the bar graph using a Javascript library chart.js. The frontend tests were updated to reflect the current website design.

System Modeling and Design

UML Diagram

Our UML diagram is located in (Course Related Items/UML and Sequence Diagrams/UML Diagram.pdf).

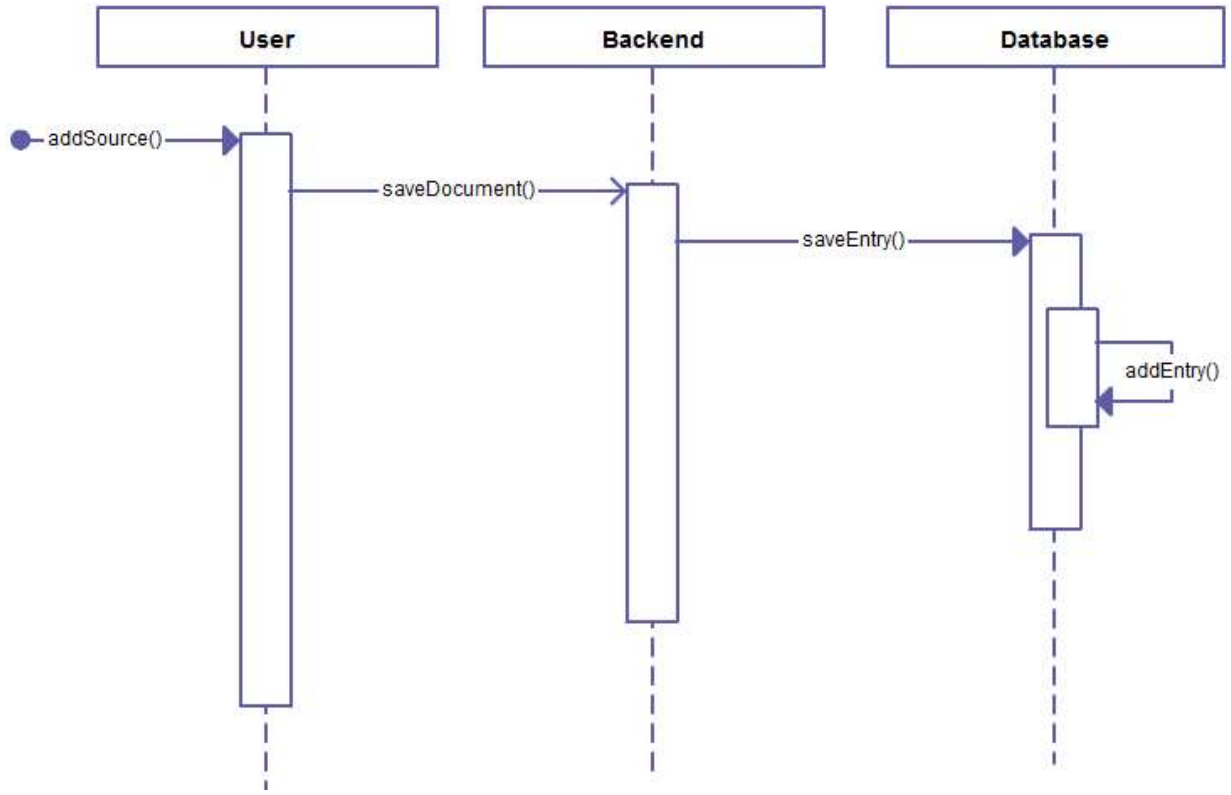
The diagram has been organized to three sections:

- Top: Website in perspective to the user (e.g. Login, add/view/delete sources/keywords/etc.)
- Middle Right: Backend & Crawler
- Bottom Left: Database & related document object stored in the database.

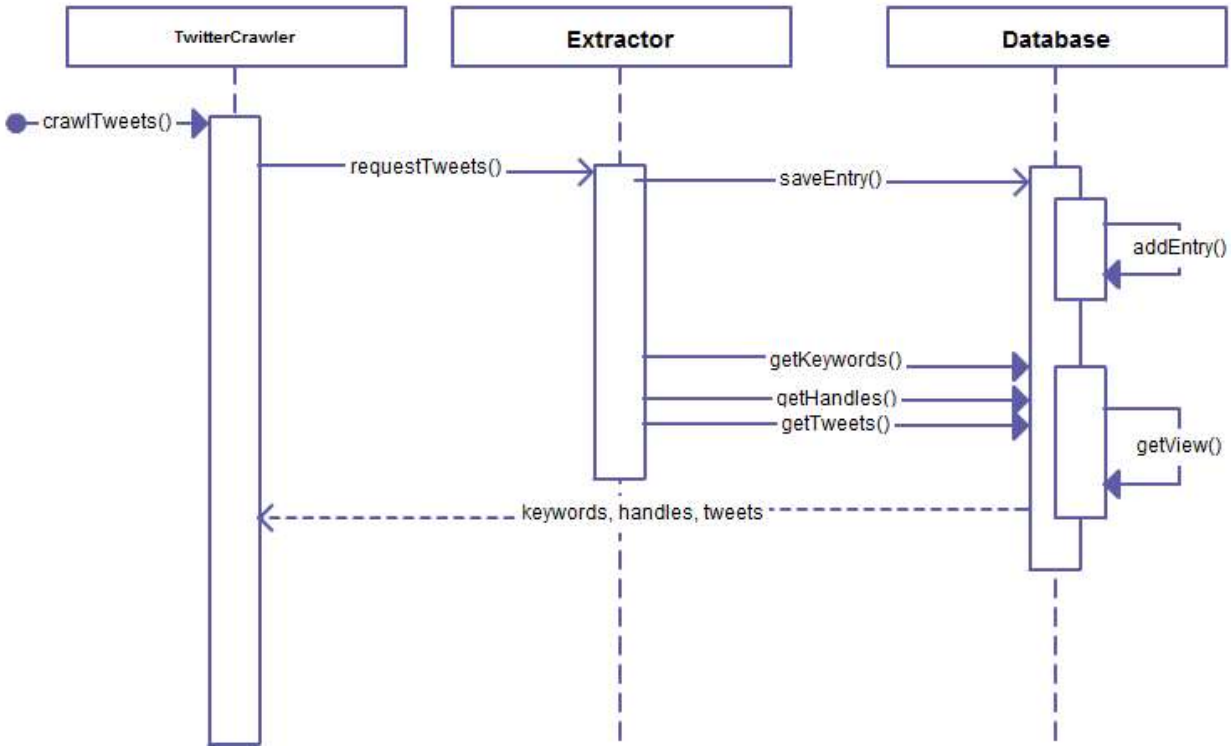
Sequence Diagram

These sequence diagrams are also located in (Course Related Items/UML and Sequence Diagrams) if it is not legible in this report.

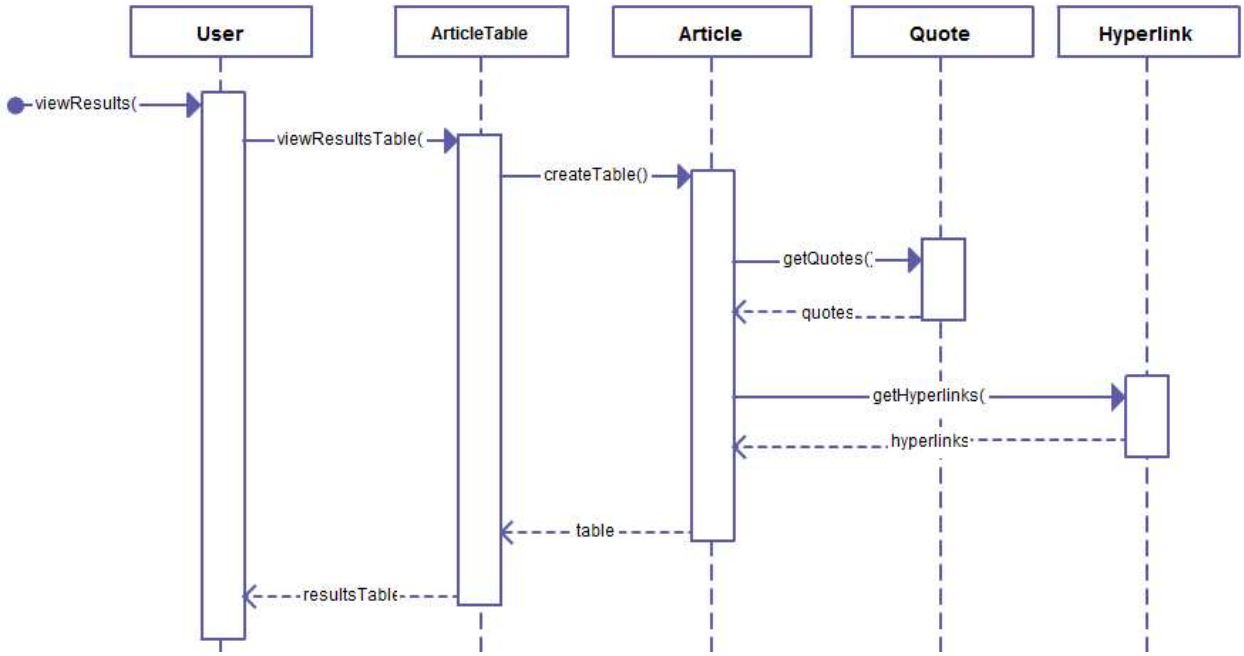
Sequence Diagram #1 – Add News Source to Database



Sequence Diagram #2 – Tweet Extraction



Sequence Diagram #3 – View Article/Feed Crawl Results



Design Pattern

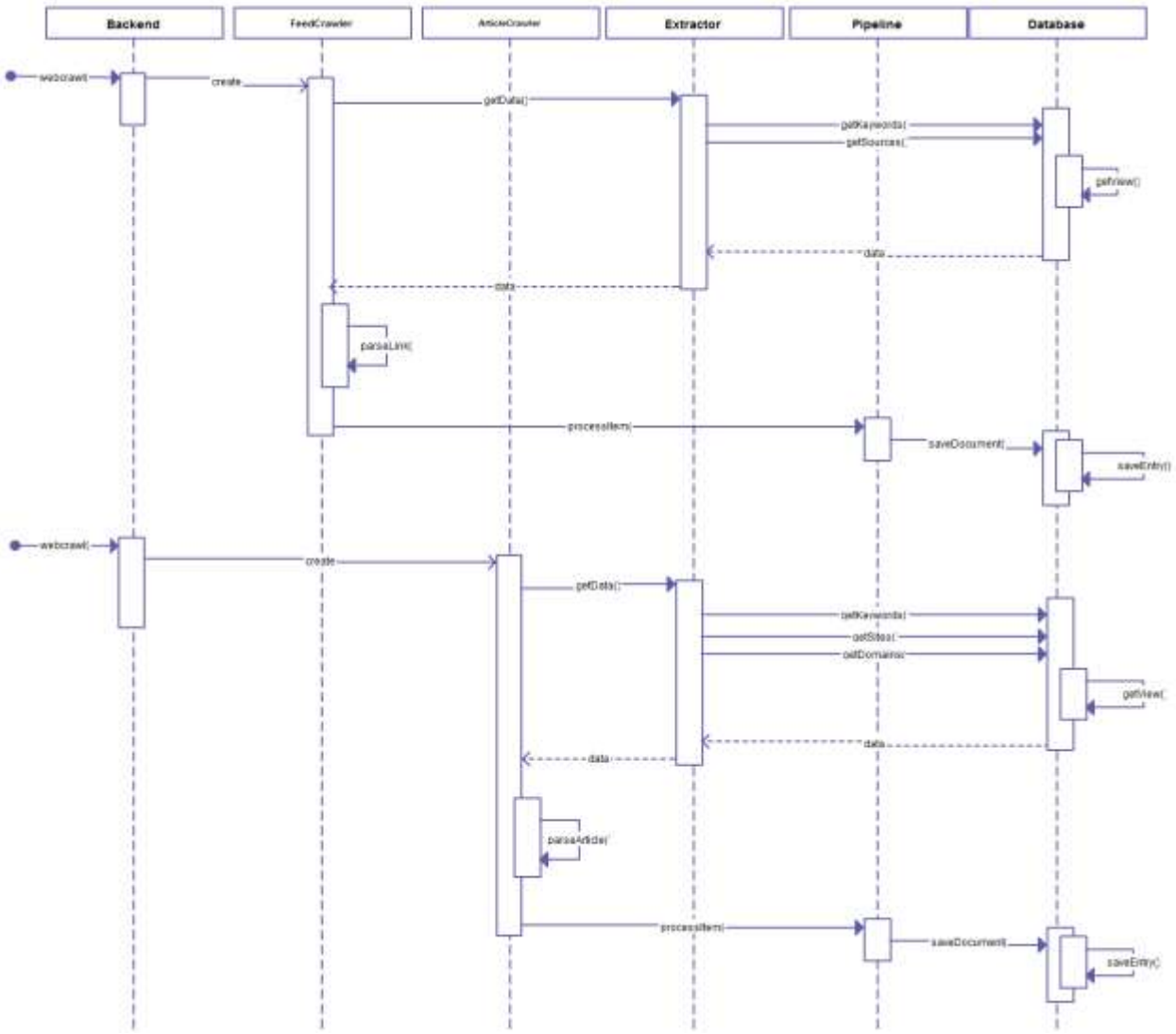
For our project we chose to use the Strategy design pattern. For this project, the user would like different ways to crawl, given their current data. Our first implementation was to use RSS feeds to crawl the data; this spawned the first crawler FeedCrawler. However, the user would also like to crawl arbitrary sites, and possible continue searching the links contained in the articles, so an ArticleCrawler was created. Since both crawlers only differ in how links are crawled, it made sense to use the Strategy design pattern.

UML diagram

The UML diagram for the Strategy design pattern is located in the middle right portion in the overall UML diagram (Course Related Items/UML and Sequence Diagrams/UML Diagram.pdf). In particular, the base class is the Crawler interface, and there are three different versions to crawl, which implements the Crawler: ArticleCrawler, FeedCrawler and TwitterCrawler. After the crawlers have found an item, it gets sent to Pipeline for further processing and the data is saved to the database when the keyword is found.

Sequence Diagram

The following sequence diagram shows the backend calling webcrawl on two different classes; one with FeedCrawler and one with ArticleCrawler. The FeedCrawler only parses links found on the page while the ArticleCrawler is will continue to parse links within the specified domains.



Improving on Phase 3

Unit Tests

More test cases and documentation have been added to **GetResultsTestCase**, located in Website/flaskVersion/app/tests.py in the master branch. In particular, both hyperlinks and quotes found in the article need to be sanitized to removed un-exportable characters.