

**DATE:** March 30th, 2019  
**TO:** Dr. Mary Eberlein  
**FROM:** Team Eagle - Hiep Nguyen, Michael Flanders, Kunpeng Qin, Nabil Khan, Nelson Levy, Wesley Klock  
**SUBJECT:** Team Eagles' BookBrain Phase 2 Report

We are team Eagles on Canvas, and our project is BookBrain. Our github can be found here: <https://github.com/Flandini/461L-Eagles>, and our webapp is located at <http://35.193.196.53/>. We have 6 members on our team:

- Name: Hiep Nguyen, Email: hiepnguyen8069@gmail.com, github username: hnguyen257
- Name: Michael Flanders, Email: flanders.michaelk@gmail.com, github username: flandini
- Name: Nelson Levy , Email: nlevy2807@gmail.com, github username: nrl538
- Name: Wesley Klock, Email: , github username: wesclock777
- Name: Kunpeng Qin, Email: , github username: qkpqkp
- Name: Nabil Khan, Email: nabilliban14@gmail.com, github username: Nabilliban14

## **TASKS, FEATURES, AND FUNCTIONALITY**

For the second phase, we promised four things:

- More tests and the output of all of our tests
- A scraper that has been extended to more sources than we previously had in phase 1
- A scraper that has been extended to use Selenium and BeautifulSoup4 to scrape dynamic web content
- A CSV file containing all of the scraped data containing at least hundreds of books

We have increased our number of tests from 10 to 32. We are still writing mostly just unit tests with a few integration tests, and we have still have no tests for dynamic web content as we promised these as a deliverable for the third phase. We have included the test output in the 'Tests' section below.

Included in the github repo is a scraper that has been extended to scrape additional sources at a much higher capacity. We have added Barnes & Noble and IDreamBooks to our previous scraper source, GoodReads. We have also started using BeautifulSoup4 to traverse the DOM of new sources in the scraper. The scrapers are now also capable of bulk scraping thousands of books without failure, and our team's github repo contains CSV files of all scraped data in its root directory as promised in our proposal (they are separated into a few CSV's because these CSV's will be fed into different databases). The different scrapers are stored in a scrapers folder located in the root directory of our github repo.

In this phase, we have also added extra features such as pagination and autocompletion to BookBrain that were not promised in our deliverables or proposal.

## **DESIGN**

We are using Python3 and Flask for our web application. Because there are no classes in our scraper, testing, or web app code, we have not included a UML class diagram as we felt that it would not properly describe the design of our application. Instead, we have included a UML sequence diagram below to describe the design of our web application for this second phase.

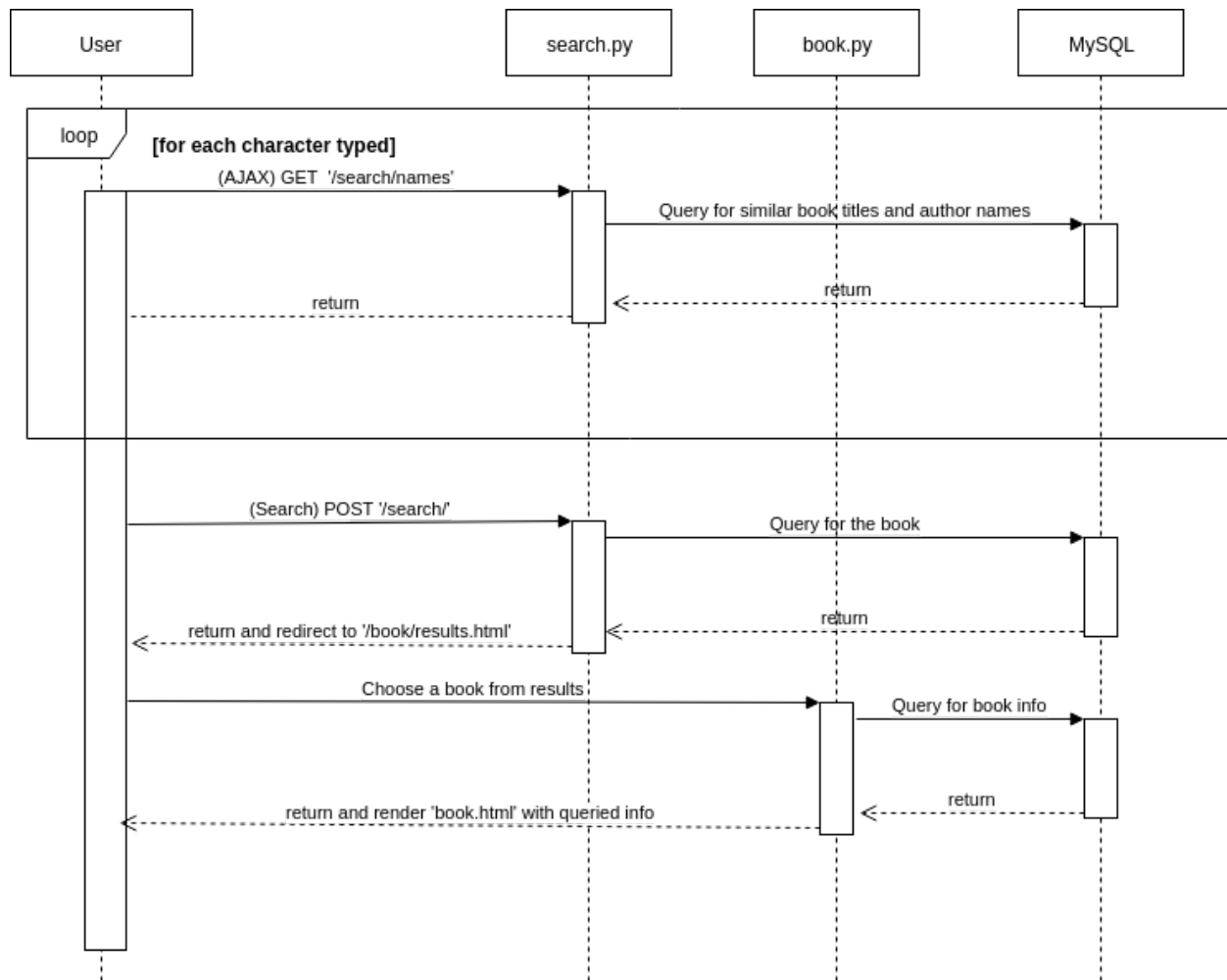


Figure 1. Sequence Diagram depicting autocomplete, search, and book page display

At this time, we have separated every piece of functionality of our webapp into modules that maximize our cohesion while minimizing coupling. For example, our `auth.py` module only handles authentication functionality such as logging in, logging out, and registering, our `search.py` module handles only search functionality including autocomplete, and our `book.py` module simply renders our book template with whatever data is in our database. As a result, our webapp is made up of only a handful of modules with methods that handle one thing and one thing only such as this logout function:

```
@bp.route('/logout')
def logout():
```

```
"""Clear the current session, including the stored user id."""  
session.clear()  
return redirect(url_for('index'))
```

We have also decided to keep our scrapers as small, simple python scripts instead of integrating them into our Flask application at this time; we are doing this mainly because we plan on using Google Cloud Functions to run our scrapers as it would be more cost-effective and simpler to automate while offloading work from our free VM instance running the webapp.

## REQUIREMENTS

We have added five more user stories to our github issues board located at <https://github.com/Flandini/461L-Eagles/issues/3>, and we have also listed them below. The requirements we made for this phase came directly from our deliverables and focus mainly on a better UI/UX when browsing our site and pulling more book data with our scraper. To provide users with a better, book-browsing experience, we have fulfilled the first two user stories by implementing pagination. To address the third user story, we have started scraping Barnes & Noble and IDreamBooks to pull critic and audience reviews. Finally, to address the last two user stories below, we have added a carousel to each individual book page listing similar books (currently does not list actually similar books as this scraped data comes from phase 3 deliverables) and fully implemented and tested our login and registration functionality. To further show our progress in fulfilling our web application's requirements, we have also included an updated use case diagram below (see Figure 2) to compare with our goal use case diagram (Figure 3).

- As a person looking for a book, I want to be able to search for a book based on different categories (author/title/isbn).
- As a person who may or may not know the full name of a book or author, I want my search query to autocomplete or suggest the book title or author as I type it.

- As a person looking for holistic reviews of a book, I want to be shown a general public/audience review and a critic's review so that I can get a feel of the overall public sentiment of the book.
- *As a person trying to find a new book to read, I want to be shown related books based on genre and an overall internet score of the book. (from phase 1)*
- As a person who looks for books often, I want to be able to register an account and log in to it to view personalized information (e.g. books I may be interested in).

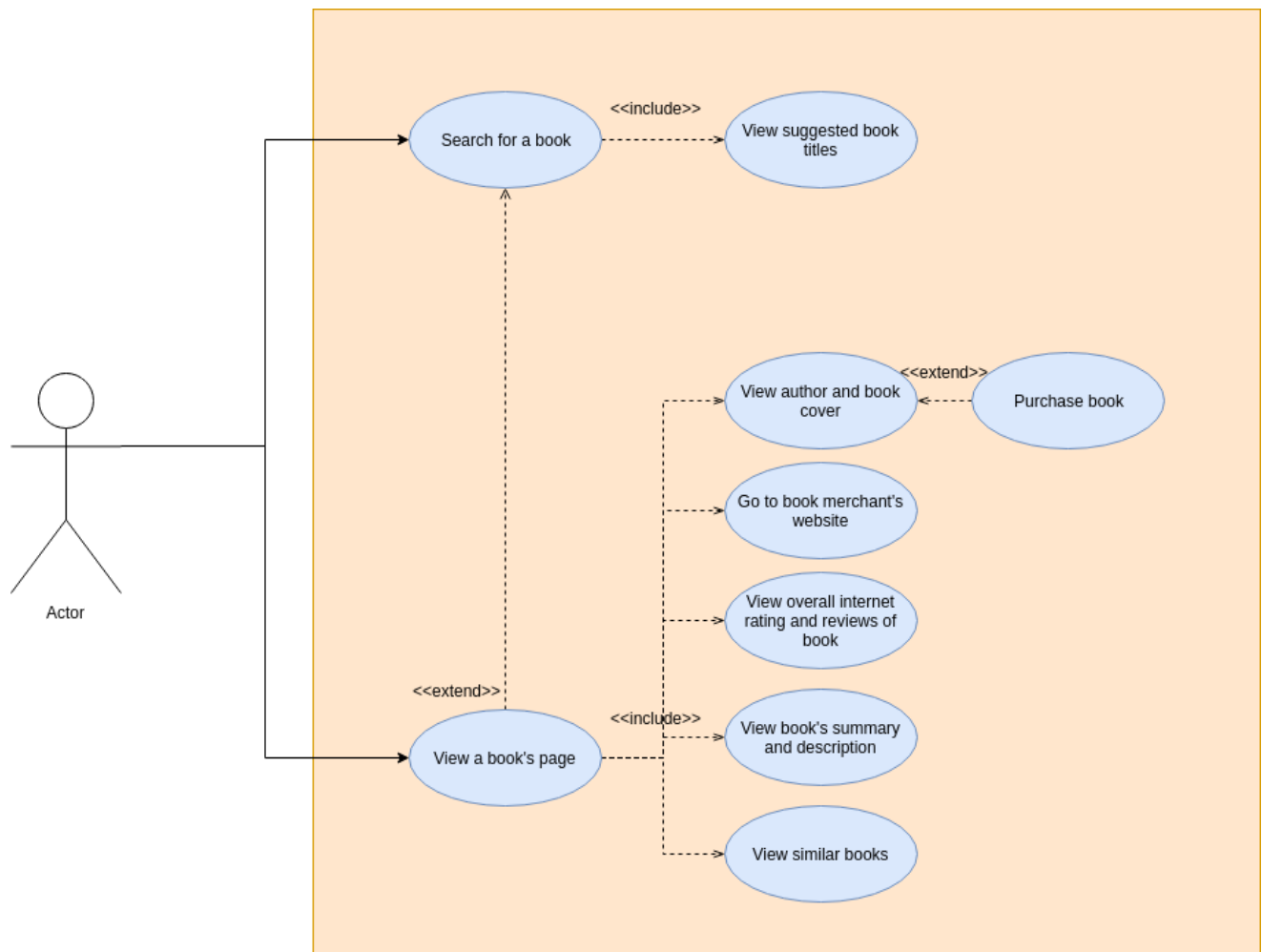


Figure 2. Phase 2 Updated Use Case Diagram

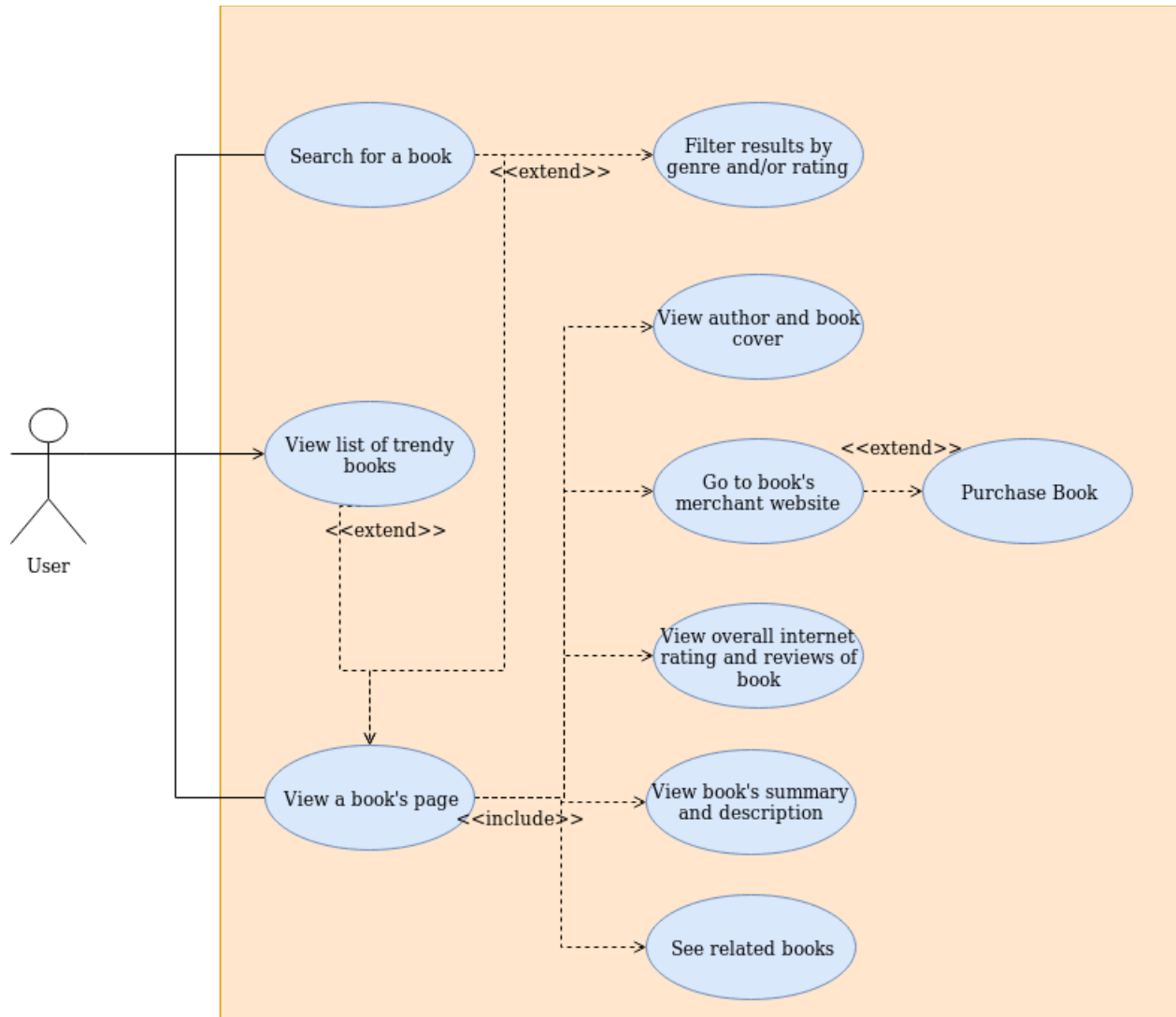


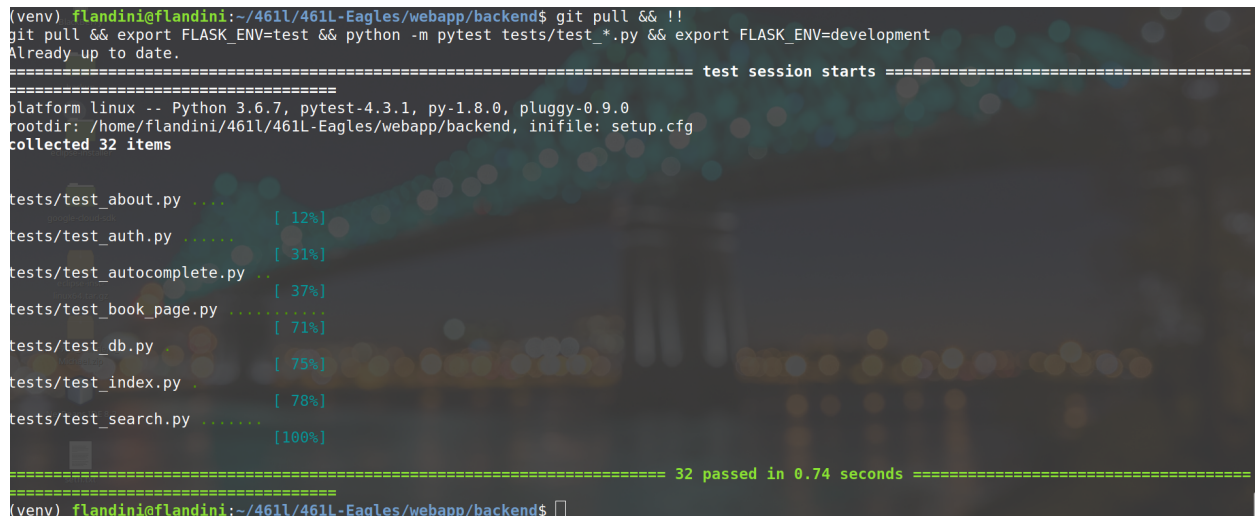
Figure 3. Goal Use Case Diagram/Use Case Diagram of BookBrain at Completion

## TOOLS, SOFTWARE, AND FRAMEWORKS

We have only changed the tools that we are using for three of our categories: back end, front end, and scraping. For the back end, we made the pagination ourselves using python, flask, MySQL, and Bootstrap. We have also added jQuery for the autocomplete of titles and authors for the search bar. For the front end, we have added no new tools and are still using HTML, CSS, and Bootstrap for the user interface. We have added beautifulsoup4 to our scraper and are still using python and Selenium to pull book information from other websites.

## TESTING

In terms of testing, phase two involved increasing test coverage as we increase our web application's functionality. We have increased our total number of tests to 32 tests. We have tests for every static feature of our web application currently (dynamic tests will be turned in for the third phases). The majority of our test suite focuses around testing the search functionality, book page rendering, and login and registration functionality; 22% of tests focus on testing our search, 34% of tests focus on individual book page rendering, and 19% of tests focus on authentication related functionality.



```
(venv) flandinigflandini:~/461L/461L-Eagles/webapp/backend$ git pull && !!
git pull && export FLASK_ENV=test && python -m pytest tests/test_*.py && export FLASK_ENV=development
Already up to date.
===== test session starts =====
platform linux -- Python 3.6.7, pytest-4.3.1, py-1.8.0, pluggy-0.9.0
rootdir: /home/flandini/461L/461L-Eagles/webapp/backend, inifile: setup.cfg
collected 32 items

tests/test_about.py ..... [ 12%]
tests/test_auth.py ..... [ 31%]
tests/test_autocomplete.py .. [ 37%]
tests/test_book_page.py ..... [ 71%]
tests/test_db.py ..... [ 75%]
tests/test_index.py . [ 78%]
tests/test_search.py ..... [100%]

===== 32 passed in 0.74 seconds =====
(venv) flandinigflandini:~/461L/461L-Eagles/webapp/backend$
```

Figure 4. Webapp Test Suite Output.

We are still only using PyTest to test our web application, but we have improved testing mechanisms from phase 1. For instance, we have updated the function `init_db()` in `db.py` to check the `FLASK_ENV` for the web application, and then connect to a development, test, or production database and seed the database depending on which one was selected.

A lot of the tests now also focus on testing the negative case in order to expand our path and branch test coverage. For instance, we have tests for the individual book page called

test\_book\_page.py (see Figure 2) that test that all the information we wish to display to the user is actually rendered, and we also test for the case that the supplied book id in the URL is an invalid book. Another example is our tests for our pagination code. We have three branches in our HTML template for search results, and we have tests to cover the HTML rendered to the client in all three cases through the following tests: test\_shows\_book\_title, test\_shows\_book\_description, test\_shows\_book\_author, test\_no\_results\_found, test\_pagination\_shows\_only\_10\_results, test\_pagination\_lists\_multiple\_page\_links, and test\_search\_results\_displays\_page\_one.