CPEN 321 Team 28
November 29, 2015
Hamza Faran//12563110//hfaran
Mesbah Mowlavi//12984134//PowerOfM
Rosa Mohammadi//31856131//rosam03

slack-overflow Project Report

Introduction

slack-overflow (our project fork at https://github.com/hfaran/slack-overflow) is a web application that implements a simple REST HTTP server, acting as a plugin for slackbot on any Slack project site. It implements the `/overflow` command, which allows users to perform a Stack Overflow query inside Slack itself. Instead of the user unknowingly asking a question which has been answered before, they will be given inline links to the top questions related to their query, without wasting anyone's time or looking foolish!

slack-overflow is written in Python. It uses a popular Python web microframework, called *Flask*, and implements only two endpoints. One is `/overflow` which receives a request from Slack on behalf of a user and returns the appropriate response that is relayed back to the user on Slack. The other is simply the root `/` endpoint which redirects to the project's GitHub repository.

Our objectives for this project were to add two key features to *slack-overflow.* One is a much more useful and accurate search, by replacing the search backend to be Google as opposed to Stack Overflow's search engine to get more relevant results. The other feature was to add an /overflow-inline command (since renamed to /soi, short for stack-overflow-inline, to save keystrokes for users) that displays the top answer for the top question in the response itself, in a similar manner to having a howdoi-like tool within Slack itself.

We were able to successfully add these features, and along the way, much improve the structure of the application by refactoring it completely, facilitating further growth in the future, as well, by adding tests to achieve **96% code coverage** (up from 0**%** as this project was previously completely untested by the project owner!).

We were motivated to choose this project since we consistently use Slack for software team projects, hence, we wanted to reap the benefits of this functionality but could not always make use of it since it was limited in its existing state. With the new features that we have added, the user experience for the `/overflow` command with better search results and the newly introduced `/soi` command, is much improved.

Work Accomplished

Restructuring for Expandability

On first examination of the source code, we found that the application was all written in one file app.py, and was not necessarily written in a way to facilitate future expansion. Additional indicators of this were: using sys.argv for arguments rather than using a library like argparse, and a config.py.example file that didn't really encourage good usability as it required forking of the app and changing the code rather than providing the one key as a command-line argument. Given this, we decided to use the click library and wrote an envvar helper method to create a framework for adding command-line options. These options **could also be provided by environment variables** (to allow for easy deployment on a PaaS like Heroku) by providing default values from a callable that either fetched from the environment or provided a default. Explicit options provided on the command line could then override these. Thus, we removed config.py as this new implementation replaced it. We then restructured the application by creating a slack_overflow package, and modularizing and moving code into the package. The only code to remain in the top-level app.py was the main function that handled command-line arguments and started the app.

Google as the Search Backend

After the restructuring, we moved onto adding search backed by Google. In order to add this, we decided to use the Google-Search-API library to simplify retrieval of search results from Google. In order to successfully search from Google for relevant results, we only wanted Stack Overflow questions to be returned from the search. For this, we looked at the URLs of the results and noticed that they were all under stackoverflow.com/questions so we appended the **site:stackoverflow.com/questions/*** filter to our Google search in order to filter results. Now here is where we ran into an interesting issue: our filter was not working. We discovered after digging into the library that the search function was normalizing the query string (with `.replace(":", "%3A")` being the particularly problematic bit). In order to fix this without having to patch the external library itself, we leveraged Python's dynamicity to patch the function during runtime so that it would not perform any normalizing. After this patch, the library returned relevant results as we desired. Following this, we wrote a function that would extract question IDs from the retrieved results and then use the Stack Overflow API to gather more information about the question that we needed to present to the user (as the Google query did not provide all of this information). Adding some additional error-checking code onto this, our Google search implementation was complete! The final result, with respect to the UI, was seamless to the user; the only change they saw were the much better search results! We tested the example that was reported on the GitHub issue by user wkoffel and the search results were indeed better.

Inline Answers

Next, we added the inline answer functionality as the /soi route. We were able to **modularize and re-use a lot of the code** from the earlier /overflow route. The important additional functionality required for implementation here was fetching answers of the top questions, and displaying the top answer body in a **Slack-compatible format**. Unfortunately, while Slack has formatting markup that is **similar** to Markdown, Slack insists on being difficult as it differs from Markdown in certain areas. To fetch the answer, we used the Stack Overflow API and then sorted the answers based on their scores to find the top answer. The next challenge was to convert the HTML bodies of the answers to Slack format.

Since no one else has written an HTML-to-Slack converter, we had to write one ourselves. However, we leveraged the fact that Slack formatting was **close** to Markdown and decided we would be able to perform the conversion nicely if we first used the excellent html2text library to first convert the body to Markdown. After doing this, we were able to look at the Slack formatting documentation (1, 2) to determine which tags still needed to be replaced. Using a combination of substring replacements, custom HTML2Text parser settings, and regex substitutions, we were able to implement a formatter that performed decently at rendering HTML answer bodies in Slack formatting.

Documentation

When writing our code, we wrote all of it following **Sphinx-style documentation** which gives the benefit of the code being **self-documenting**, i.e., we can use Sphinx to generate API documentation for the code if we wanted, and it gives us the additional benefit of **type-hinting** provided our IDE, PyCharm, which can read structured Sphinx docstrings and provide linting of the code through those. Additionally, all of ours is written using the standard Python PEP 8 style guide so that the code is easily readable by both us and new eyes.

Testing

When we started the project, there were **no tests implemented**. So we wrote **unit tests** for all of the modular, unit-testable code, and **functional tests** for the live routes to ensure the service is functionally performing correctly (going against Stack Overflow and Google). Additionally, we performed **acceptance testing** by manually running /overflow and /soi commands on Slack to assert that we get the accepted user-facing output (and if Slack is formatting our responses as it claims it should). With our **16 new test cases**, we improved **coverage** from **0% to 96%**. For our unit-test runner, we used pytest and additionally made use of the pytest-cov library for coverage and mock library for mocking external objects in our tests.

Continuous Integration

In the original project by the maintainer, **no CI was used,** so we set up CI ourselves. We used Travis-CI for continuous integration; it would immediately run tests and tell us if pushed code was problematic by emailing upon error or failure in test runs. Additionally, coverage results from Travis-CI runs are pushed to Coveralls, a tool that records and reports changes in test code coverage for developers.

Continuous Deployment

We deployed our application on Heroku and configured it so that a new version of our application would be deployed continuously when the *develop* branch changed. This way, we could see if the latest *develop* copy was functionally performing as expected.

Statistics

- **19 changed files** with **540 additions** and **72 deletions**.
- Project Contributors Graph
- 16 new tests added w/ 96% Code Coverage (up from 0% as the project originally no tests)

Project Management

Version Control

For version control, we of course used git with GitHub. We also made liberal use of pull requests for adding all of our features. Also, we followed the git flow model by nvie, so each feature was developed on its own branch. The branch would then be merged into *develop* when it was ready. When we are ready to release a new version, we first merge *develop* into *master* for that version and then make a new release (which can create a tag for us) on GitHub.

Task Management

We used waffle.io (which is backed by GitHub issues) to manage what tasks we were working on. Items not ready to be worked on stayed in the backlog, and items that were ready to be worked on were in ready, and so on. We also used **story points** (which is a feature of Waffle) to estimate the difficulty of each task. We also used the feature of GitHub of closing issues automatically with pull requests (PRs) to match issues with PRs in waffle.

Conclusion

We implemented two feature additions to slack-overflow: Google backend for query search for /overflow, and inline answer responses with /soi. Along the way, we also restructured the application for facilitating any future expansion along with a design that makes it easy to extend as well as deploy on PaaS' (like Heroku) while also keeping it easy to develop. We additionally added 16 new test cases to improve coverage to 96% of the code, and set up continuous integration and deployment.

We also offered to merge our work upstream with a clear and detailed Pull Request of the features added. Unfortunately, it seems as if the maintainer is on a GitHub hiatus, so it seems he will not be able to merge this soon, however, as we have demonstrated in this report, the improvements we have made to slack-overflow will benefit any and all new contributors to come, should they choose to use our fork (or the original author accept our changes to the upstream).