Timothy Machnacki (tmachnac)
EECS 481: Software Engineering W22
22 April 2022

HW6b: Contribution

1. **[*Name and Email Ids*]**
   Timothy Machnacki (tmachnac)

2. **[*Selected Project*]**
   Zulip is an opensource chat-messaging application similar to Slack, Discord, or Microsoft Teams. Zulip's primary website and available downloads can be found at https://zulip.com/ and the repositories can be found at https://github.com/zulip. Like the other aforementioned messaging apps, Zulip is designed for communities, companies, or organizations; these groups need a way for members to stay up to date and collaborate remotely. Zulip is unique in that it offers subdivided topics within streams (the equivalent of a channel or a team). The Zulip project is subdivided among smaller projects such as the main Zulip app, the Zulip API, and the Zulip mobile apps. Zulip utilizes a Python(Django) backend, Tornado for synchronous event updates, a JavaScript frontend, as well as a React Native mobile app. I was originally going to contribute by adding a feature in the main app that allowed a user to unsubscribe an entire group from a stream. When this proved to challenging, I decided to implement a cryptocurrency bot in the Zulip API instead.

3. **[*Social Good Indication*]**
   This project is **not** a contribution to Social Good.

4. **[*Project Context*]**
   Zulip is an open-source chat application that rivals apps like Slack and Discord. Zulip is mainly designed for larger businesses and organizations, similar to Microsoft Teams. As per their website, "Zulip combines the immediacy of real-time chat with an emil threading model. With Zulip, you can catch up on important conversations while ignoring irrelevant ones." Zulip aims to increase productivity by allowing users to quickly find relevant updates. Stream filtering is one main feature that is conducive for a more streamlined feed. Stream filtering breaks chat streams down into sub-categories and allows users to even search streams using keywords. This allows users to stay up to date with relevant information while ignoring the extra noise. It is features like these that separate Zulip from the competition.

5. **[*Project Governance*]**

The vast majority of communication among contributors and developers is done through the project's own zulip server at chat.zulip.org. There is a myriad of streams to help developers and discuss project management and development. Furthermore, there are stream specifically for helping new contributors such as "development help", "provision help", and "new members". Additionally, communication on the server is relatively informal; anyone is welcome to communicate their comments, questions, and concerns, and in my experience, everyone was friendly and open to helping answer any questions. There are many pre-existing sources to help new developer get up to speed with the project. For example, there is detailed documentation on setting up a development environment for testing and the directory structure of the project to help contributors locate relevant files. Additional communication occurs directly through github pull requests and open issues. Anyone can claim an issue by @-mentioning zulipbot under an open issue. However, you can always free up the issue, and if no pull request is submitted within 14 days, the issue is reopened for everyone else.

As far as QA structure and guidelines, the process is relatively straightforward. There are many pre-built tools to assist in the QA process. In my case, I was writing to the python-zulip-api repository. To set up a working environment, I built a clean virtual environment using the "tools/provision" script in the repository. I followed the guide for "writing bots" in the API documentation. Developers are encouraged to follow the existing framework for new bots and their relevant unit tests. I used the existing pytest framework to run my unit tests against my new code. After passing the test cases, I ran the provided linters on my code to adhere to Zulip's style conventions. Upon submitting a pull request, the code must be reviewed by at least one core developer / maintainer. Furthermore, the changes will be run through three Continuous Integration (CI) builds before the changes are accepted.

6. **[*Task Description*]**

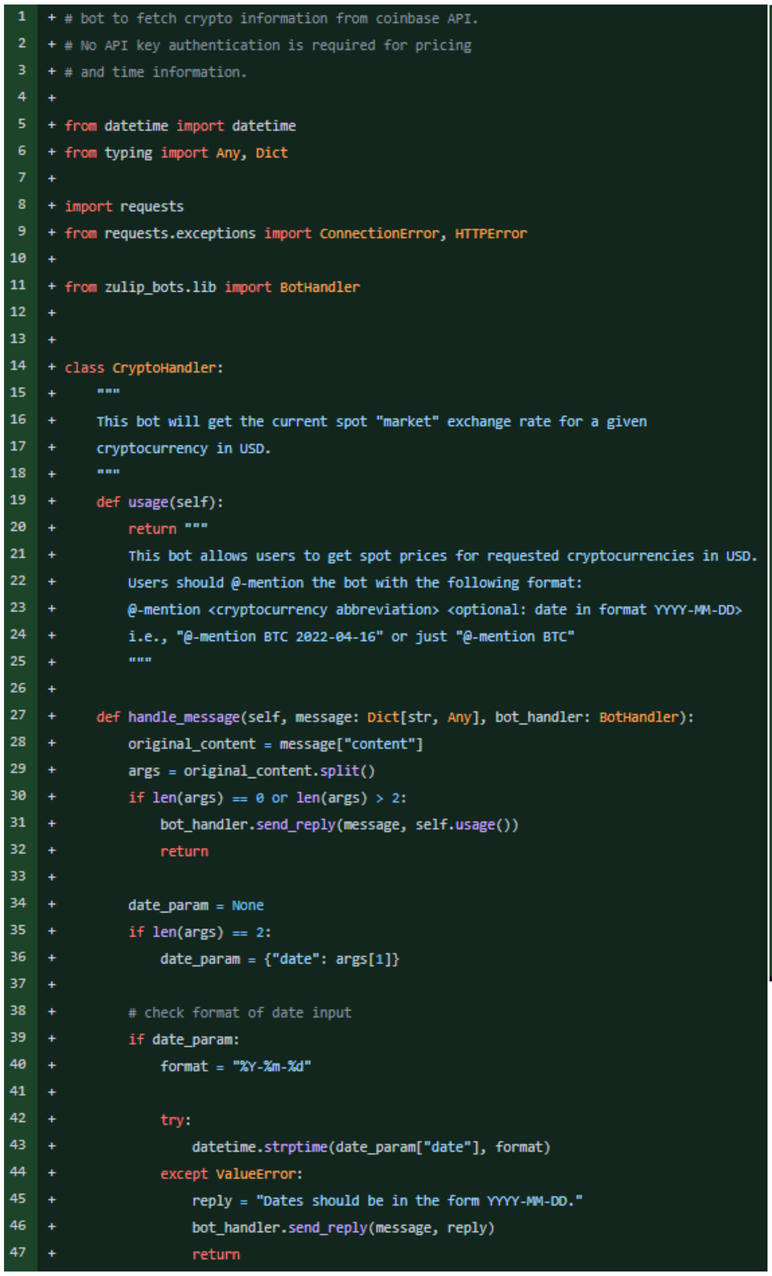
As mentioned earlier, I originally planned to implement a feature which allowed users to unsubscribe entire groups from stream – at the time of writing this, you must unsubscribe each member individually. After 6+ hours of research and documentation reading, I was not really sure where to even begin. I realized I would have to write additions to the database model, acquire special permissions, backend tests, frontend tests, and several files in the frontend, backend, and middleware. Thus, I deemed this project too daunting to undertake, and pivoted to implementing a bot in python-zulip-api repository. This repository was much less dense, and there were examples of bots and unit tests I could go off of. My bot fetches data from the Coinbase API to return market price information for a specified cryptocurrency. Additionally, users can supply a date argument to get back the market price for a cryptocurrency on said date.

Given the rising relevance of cryptocurrency, I thought this would be a neat bot to have during small-talk stand-ups or other conversations. I followed the existing "BotHandler" framework to implement my bot and followed the unit testing framework to write my test cases. Additionally, I had document the usage and purpose of the bot, and I had to write fixtures (JSON) that the testing modules could use to mock http requests and responses.

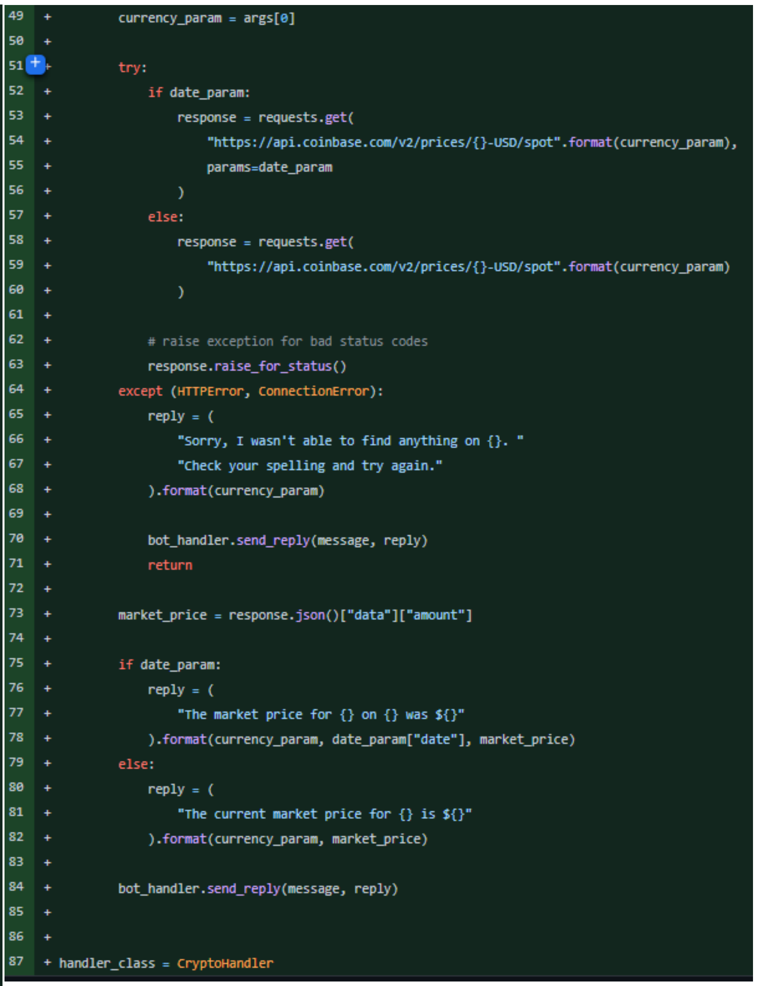
7. **[*Submitted Artifacts*]**

Below are screenshots of the artifacts I submitted @ https://github.com/zulip/python-zulip-api/pull/753/files:

Bot Implementation:

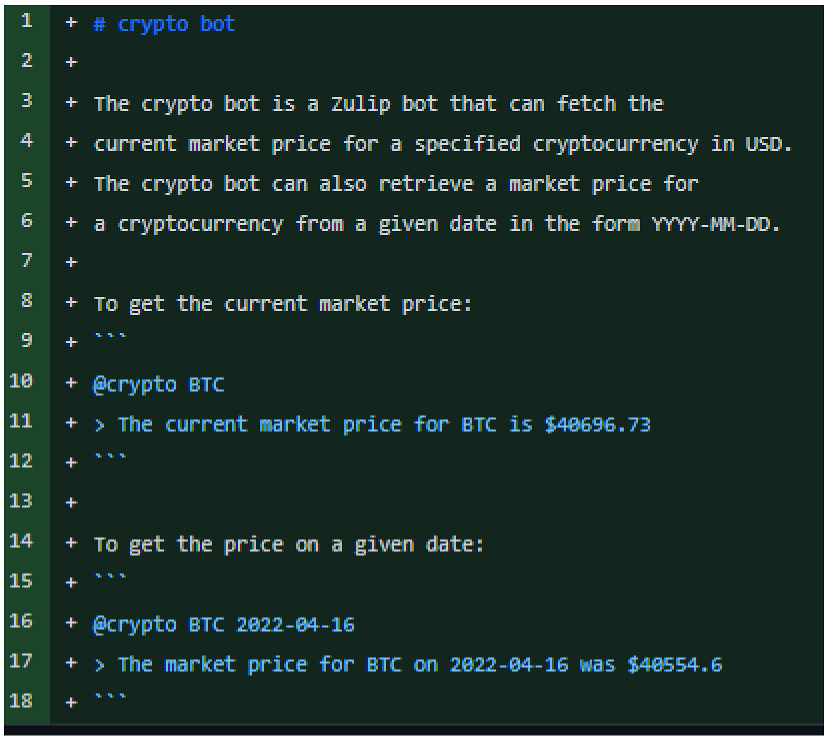


```
+ # bot to fetch crypto information from coinbase API.
+ # No API key authentication is required for pricing
+ # and time information.
+
+ from datetime import datetime
+ from typing import Any, Dict
+
+ import requests
+ from requests.exceptions import ConnectionError, HTTPError
+
+ from zulip_bots.lib import BotHandler
+
+
+ class CryptoHandler:
+     """
+     This bot will get the current spot "market" exchange rate for a given
+     cryptocurrency in USD.
+     """
+     def usage(self):
+         return """
+         This bot allows users to get spot prices for requested cryptocurrencies in USD.
+         Users should @-mention the bot with the following format:
+         @-mention <cryptocurrency abbreviation> <optional: date in format YYYY-MM-DD>
+         i.e., "@-mention BTC 2022-04-16" or just "@-mention BTC"
+         """
+
+     def handle_message(self, message: Dict[str, Any], bot_handler: BotHandler):
+         original_content = message["content"]
+         args = original_content.split()
+         if len(args) == 0 or len(args) > 2:
+             bot_handler.send_reply(message, self.usage())
+             return
+
+         date_param = None
+         if len(args) == 2:
+             date_param = {"date": args[1]}
+
+         # check format of date input
+         if date_param:
+             format = "%Y-%m-%d"
+
+             try:
+                 datetime.strptime(date_param["date"], format)
+             except ValueError:
+                 reply = "Dates should be in the form YYYY-MM-DD."
+                 bot_handler.send_reply(message, reply)
+                 return
```



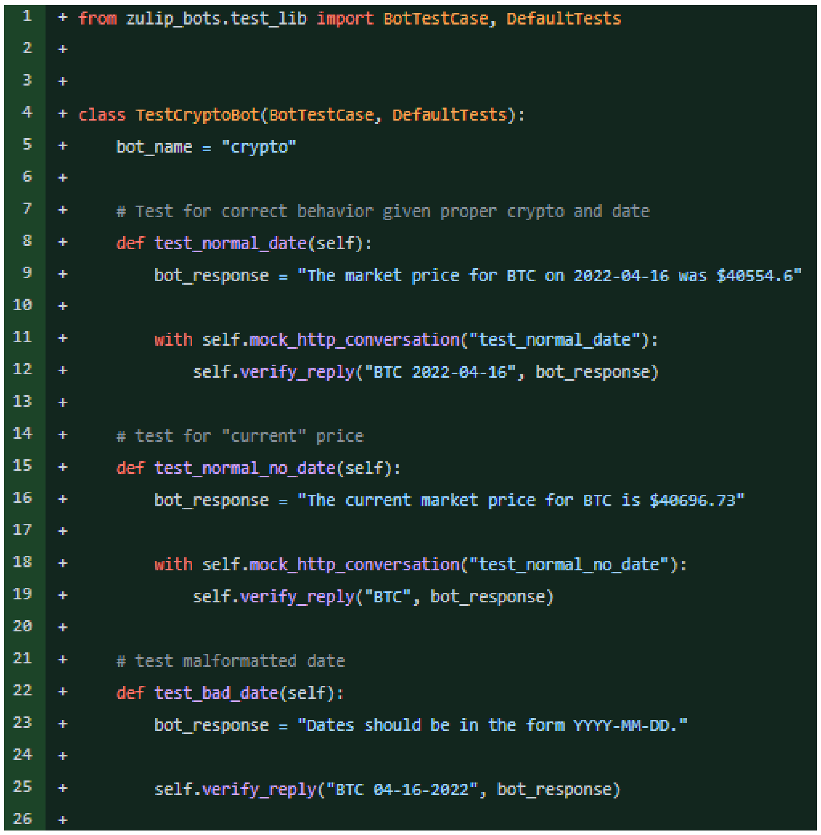
```
+         currency_param = args[0]
+
+         try:
+             if date_param:
+                 response = requests.get(
+                     "https://api.coinbase.com/v2/prices/{}-USD/spot".format(currency_param),
+                     params=date_param
+                 )
+             else:
+                 response = requests.get(
+                     "https://api.coinbase.com/v2/prices/{}-USD/spot".format(currency_param)
+                 )
+
+             # raise exception for bad status codes
+             response.raise_for_status()
+         except (HTTPError, ConnectionError):
+             reply = (
+                 "Sorry, I wasn't able to find anything on {}. "
+                 "Check your spelling and try again."
+             ).format(currency_param)
+
+             bot_handler.send_reply(message, reply)
+             return
+
+         market_price = response.json()["data"]["amount"]
+
+         if date_param:
+             reply = (
+                 "The market price for {} on {} was ${}"
+             ).format(currency_param, date_param["date"], market_price)
+         else:
+             reply = (
+                 "The current market price for {} is ${}"
+             ).format(currency_param, market_price)
+
+         bot_handler.send_reply(message, reply)
+
+
+ handler_class = CryptoHandler
```

Doc.md (documentation)

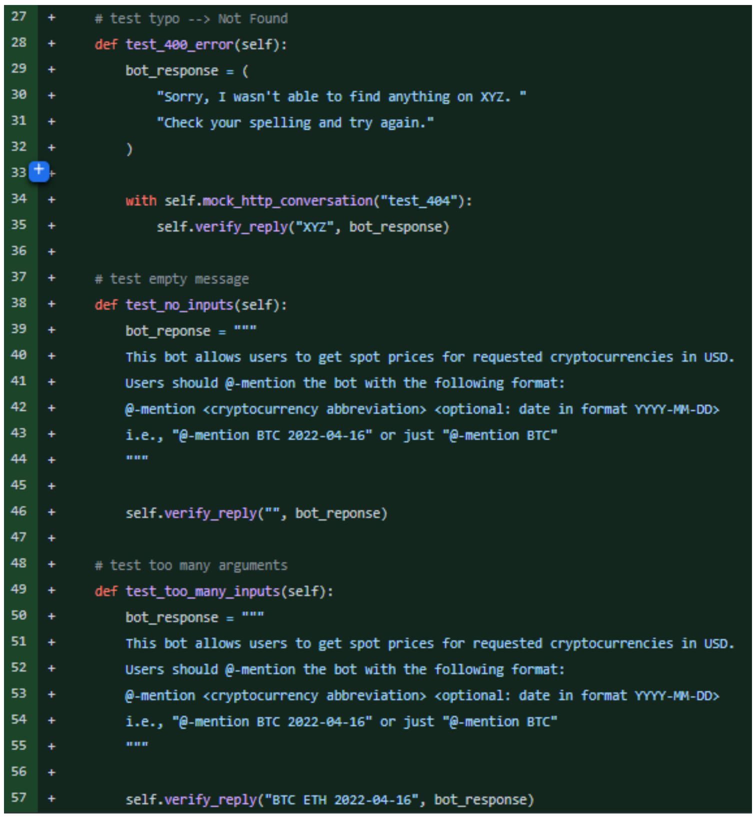


```
+ # crypto bot
+
+ The crypto bot is a Zulip bot that can fetch the
+ current market price for a specified cryptocurrency in USD.
+ The crypto bot can also retrieve a market price for
+ a cryptocurrency from a given date in the form YYYY-MM-DD.
+
+ To get the current market price:
+ ```
+ @crypto BTC
+ > The current market price for BTC is $40696.73
+ ```
+
+ To get the price on a given date:
+ ```
+ @crypto BTC 2022-04-16
+ > The market price for BTC on 2022-04-16 was $40554.6
+ ```
```

Unit Tests:

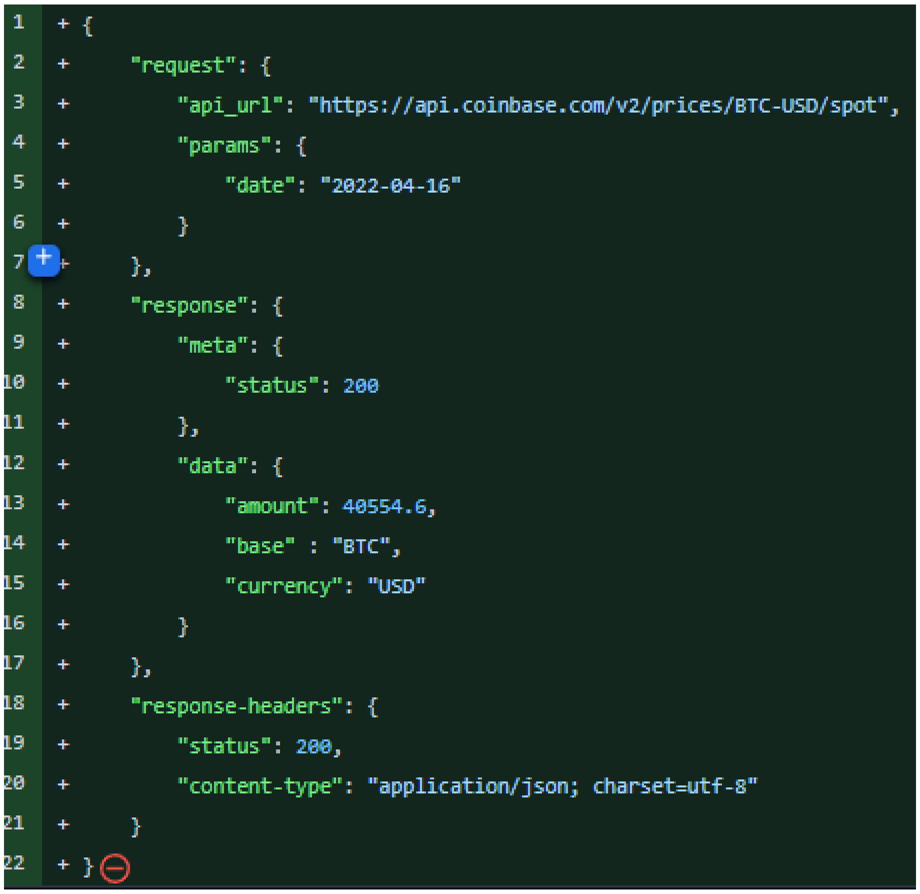


```
+ from zulip_bots.test_lib import BotTestCase, DefaultTests
+
+
+ class TestCryptoBot(BotTestCase, DefaultTests):
+     bot_name = "crypto"
+
+     # Test for correct behavior given proper crypto and date
+     def test_normal_date(self):
+         bot_response = "The market price for BTC on 2022-04-16 was $40554.6"
+
+         with self.mock_http_conversation("test_normal_date"):
+             self.verify_reply("BTC 2022-04-16", bot_response)
+
+     # test for "current" price
+     def test_normal_no_date(self):
+         bot_response = "The current market price for BTC is $40696.73"
+
+         with self.mock_http_conversation("test_normal_no_date"):
+             self.verify_reply("BTC", bot_response)
+
+     # test malformatted date
+     def test_bad_date(self):
+         bot_response = "Dates should be in the form YYYY-MM-DD."
+
+         self.verify_reply("BTC 04-16-2022", bot_response)
+
```



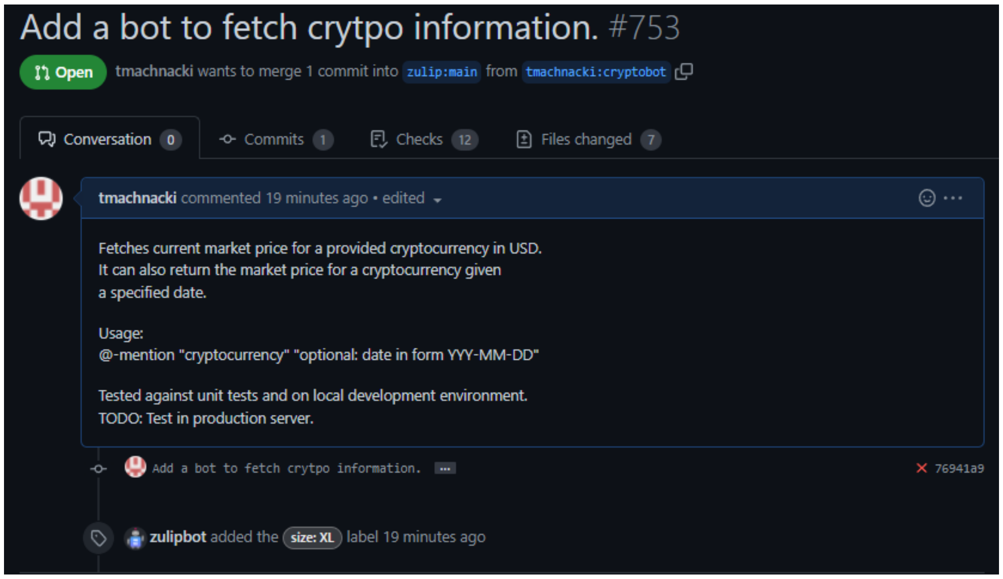
```
+     # test typo --> Not Found
+     def test_400_error(self):
+         bot_response = (
+             "Sorry, I wasn't able to find anything on XYZ. "
+             "Check your spelling and try again."
+         )
+
+         with self.mock_http_conversation("test_404"):
+             self.verify_reply("XYZ", bot_response)
+
+     # test empty message
+     def test_no_inputs(self):
+         bot_reponse = """
+         This bot allows users to get spot prices for requested cryptocurrencies in USD.
+         Users should @-mention the bot with the following format:
+         @-mention <cryptocurrency abbreviation> <optional: date in format YYYY-MM-DD>
+         i.e., "@-mention BTC 2022-04-16" or just "@-mention BTC"
+         """
+
+         self.verify_reply("", bot_reponse)
+
+     # test too many arguments
+     def test_too_many_inputs(self):
+         bot_response = """
+         This bot allows users to get spot prices for requested cryptocurrencies in USD.
+         Users should @-mention the bot with the following format:
+         @-mention <cryptocurrency abbreviation> <optional: date in format YYYY-MM-DD>
+         i.e., "@-mention BTC 2022-04-16" or just "@-mention BTC"
+         """
+
+         self.verify_reply("BTC ETH 2022-04-16", bot_response)
```

Example Fixture:



```
+ {
+     "request": {
+         "api_url": "https://api.coinbase.com/v2/prices/BTC-USD/spot",
+         "params": {
+             "date": "2022-04-16"
+         }
+     },
+     "response": {
+         "meta": {
+             "status": 200
+         },
+         "data": {
+             "amount": 40554.6,
+             "base" : "BTC",
+             "currency": "USD"
+         }
+     },
+     "response-headers": {
+         "status": 200,
+         "content-type": "application/json; charset=utf-8"
+     }
+ }
```

Pull Request: https://github.com/zulip/python-zulip-api/pull/753



Add a bot to fetch crytpo information. #753

Open tmachnacki wants to merge 1 commit into zulip:main from tmachnacki:cryptobot

Conversation 0 Commits 1 Checks 12 Files changed 7

tmachnacki commented 19 minutes ago • edited

Fetches current market price for a provided cryptocurrency in USD.
It can also return the market price for a cryptocurrency given
a specified date.

Usage:
@-mention "cryptocurrency" "optional: date in form YYY-MM-DD"

Tested against unit tests and on local development environment.
TODO: Test in production server.

Add a bot to fetch crytpo information. ... 76941a9

zulipbot added the size: XL label 19 minutes ago

8. **[*QA strategy*]**

Communication:

- Any questions or concerns about contributing to the project were addressed in the Zulip development server (development help specifically). I also found solutions to issues I was having by filtering channels.

Code Review

- Every pull request must be reviewed by a core developer / maintainer of the project. They provide feedback and make sure the code is up to the project's standards.

Maintainability and Readability:

- I spent a large portion of my time reading through existing code and documentation to ensure that my code followed desired structural and style guidelines. Modelling my implementation after existing frameworks would make it easier for future developers to understand and ensure that any additions integrated well within the existing project structure.
- While not explicitly required, it is highly recommended that new contributions be run against linters to enforce style guidelines. I was having trouble running the existing linting script but was able to run each linter in the script on its own. Some linters include pycodestyle, flake8, and mypy. Linting is a great static analysis tool that can improve code's readability.

Unit-Testing:

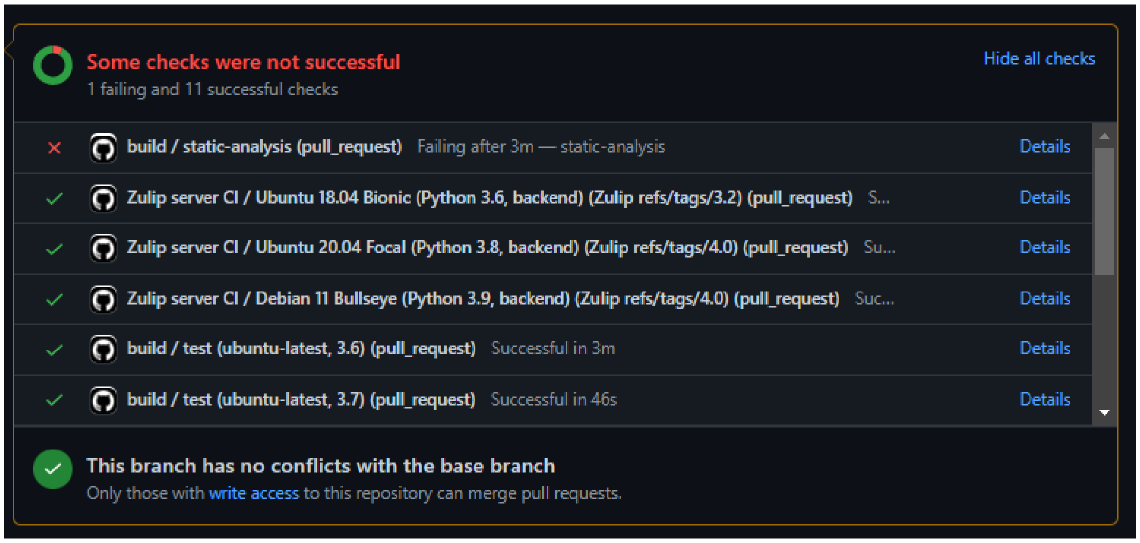
- As per the API documentation, bots should have their own self-contained unit tests. These unit tests follow a framework laid out in "test_lib.py". Each bot should have a corresponding test<botname> class containing unit test methods. Additionally, the framework utilizes JSON fixtures to supply mocking information to the mock_http_converstion method. This is done to avoid unnecessarily loading third-party APIs during testing.
- While not explicitly required, developers are encouraged to run static analysis coverage metrics to ensure the adequacy of the unit tests. I was able to achieve 100% statement coverage with my unit tests.

Integration Testing:

- After implementing code changes, developers should run the full test suite with pytest to ensure changes don't break existing code.
- Pull Requests must pass through Travis CI integration tests which runs a build of the entire project. This is done to ensure that nothing has broken before pushing code.

9. ***[QA Evidence]***

CI Build:



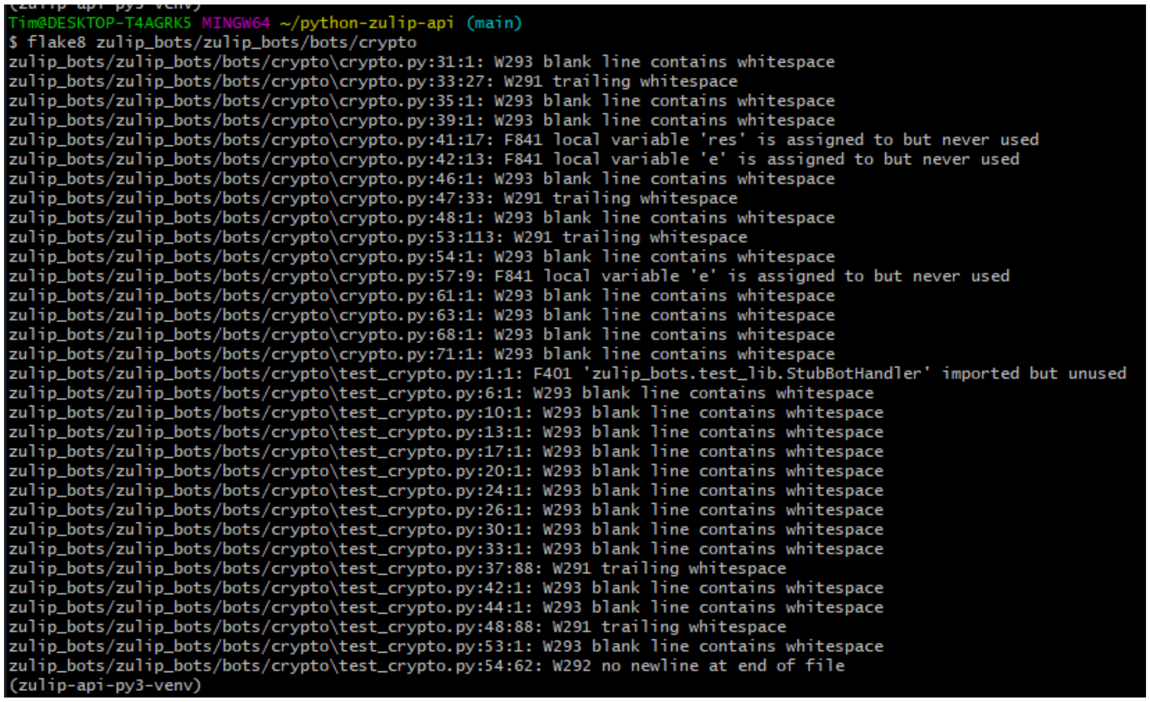
Some checks were not successful
1 failing and 11 successful checks

Hide all checks

✗	build / static-analysis (pull_request) Failing after 3m — static-analysis	Details
✓	Zulip server CI / Ubuntu 18.04 Bionic (Python 3.6, backend) (Zulip refs/tags/3.2) (pull_request) S...	Details
✓	Zulip server CI / Ubuntu 20.04 Focal (Python 3.8, backend) (Zulip refs/tags/4.0) (pull_request) Su...	Details
✓	Zulip server CI / Debian 11 Bullseye (Python 3.9, backend) (Zulip refs/tags/4.0) (pull_request) Suc...	Details
✓	build / test (ubuntu-latest, 3.6) (pull_request) Successful in 3m	Details
✓	build / test (ubuntu-latest, 3.7) (pull_request) Successful in 46s	Details

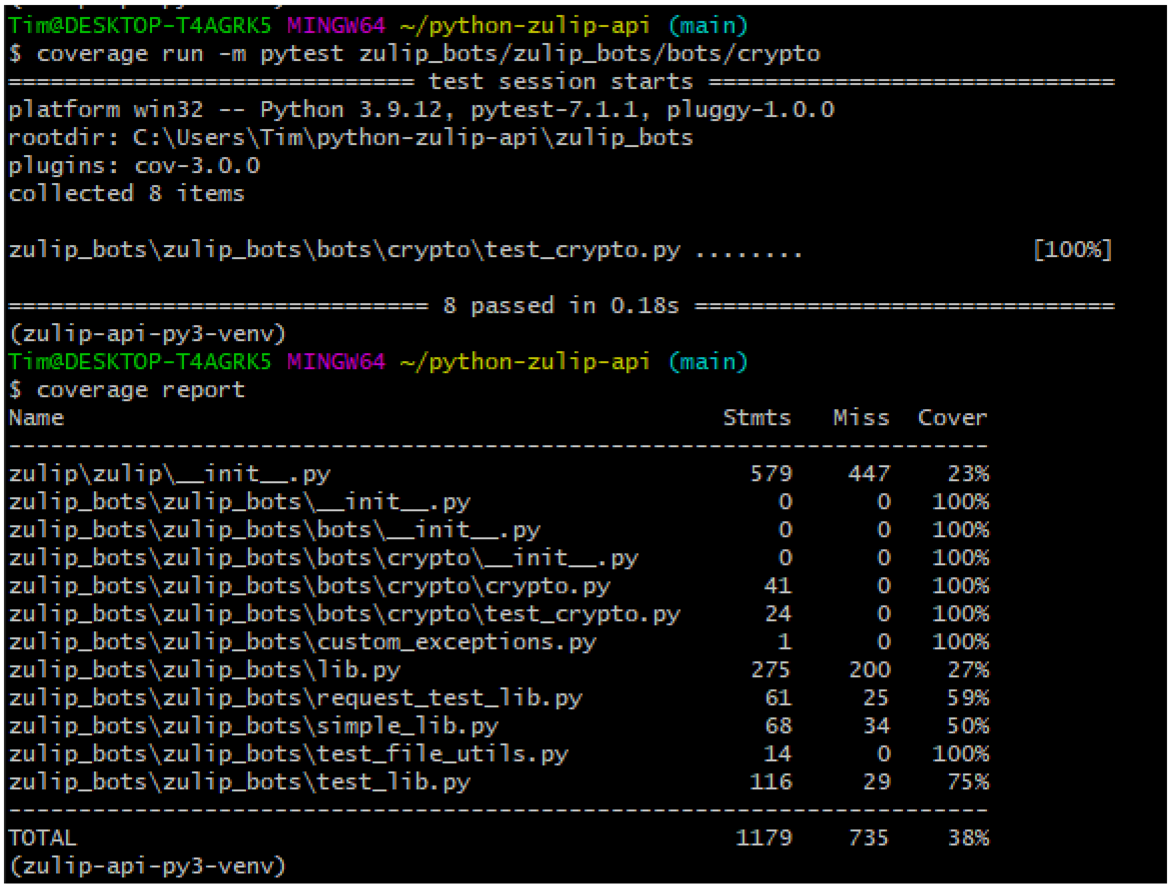
This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.

Linting:



```
Tim@DESKTOP-T4AGRK5 MINGW64 ~/python-zulip-api (main)
$ flake8 zulip_bots/zulip_bots/bots/crypto
zulip_bots/zulip_bots/bots/crypto\crypto.py:31:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:33:27: W291 trailing whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:35:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:39:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:41:17: F841 local variable 'res' is assigned to but never used
zulip_bots/zulip_bots/bots/crypto\crypto.py:42:13: F841 local variable 'e' is assigned to but never used
zulip_bots/zulip_bots/bots/crypto\crypto.py:46:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:47:33: W291 trailing whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:48:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:53:113: W291 trailing whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:54:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:57:9: F841 local variable 'e' is assigned to but never used
zulip_bots/zulip_bots/bots/crypto\crypto.py:61:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:63:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:68:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\crypto.py:71:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:1:1: F401 'zulip_bots.test_lib.StubBotHandler' imported but unused
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:6:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:10:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:13:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:17:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:20:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:24:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:26:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:30:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:33:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:37:88: W291 trailing whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:42:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:44:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:48:88: W291 trailing whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:53:1: W293 blank line contains whitespace
zulip_bots/zulip_bots/bots/crypto\test_crypto.py:54:62: W292 no newline at end of file
(zulip-api-py3-venv)
```

Coverage:

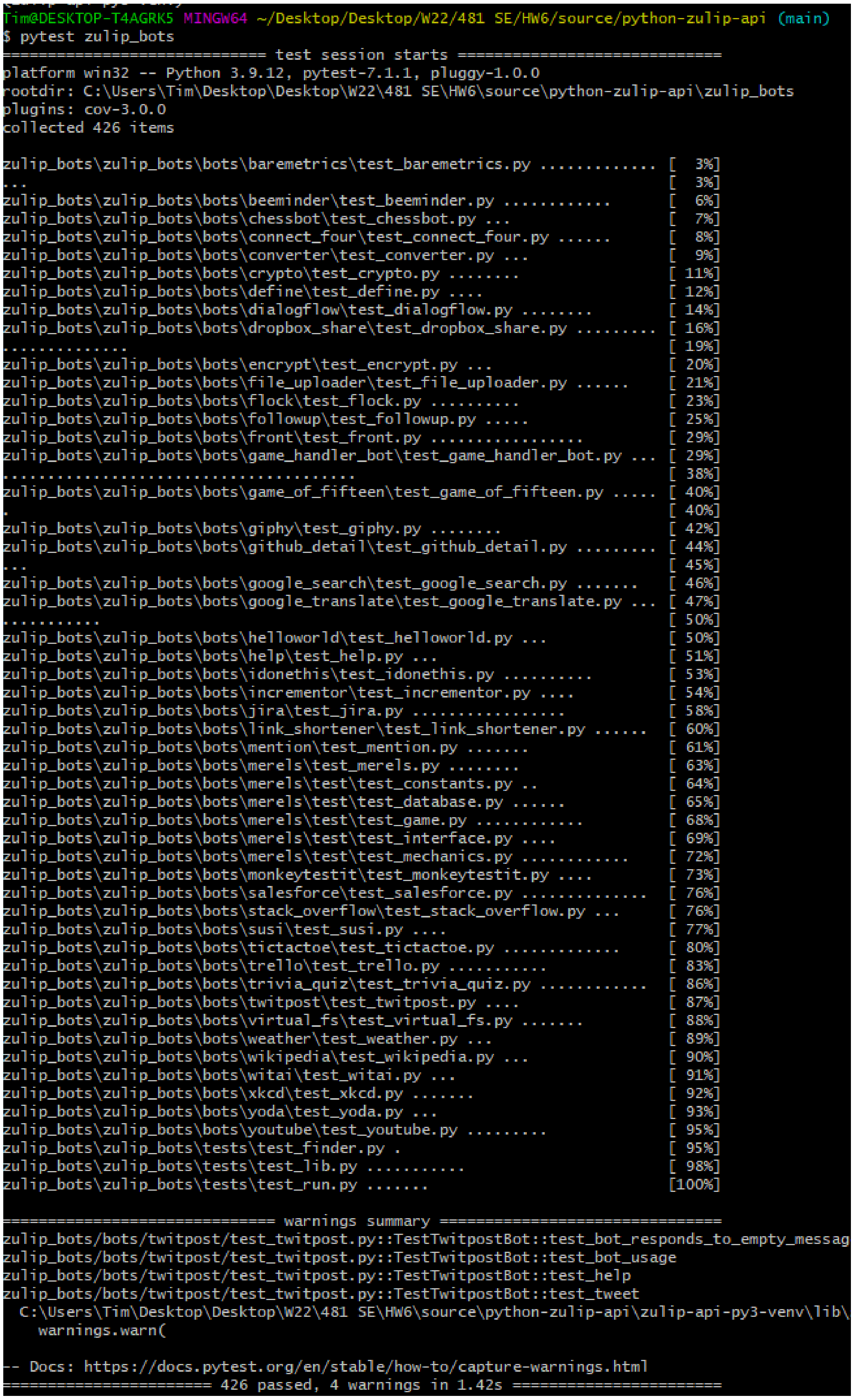


```
Tim@DESKTOP-T4AGRK5 MINGW64 ~/python-zulip-api (main)
$ coverage run -m pytest zulip_bots/zulip_bots/bots/crypto
============================= test session starts =============================
platform win32 -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: C:\Users\Tim\python-zulip-api\zulip_bots
plugins: cov-3.0.0
collected 8 items

zulip_bots\zulip_bots\bots\crypto\test_crypto.py ........                [100%]

============================== 8 passed in 0.18s ==============================
(zulip-api-py3-venv)
Tim@DESKTOP-T4AGRK5 MINGW64 ~/python-zulip-api (main)
$ coverage report
Name                                               Stmts   Miss  Cover
----------------------------------------------------------------------
zulip\zulip\__init__.py                              579    447    23%
zulip_bots\zulip_bots\__init__.py                      0      0   100%
zulip_bots\zulip_bots\bots\__init__.py                 0      0   100%
zulip_bots\zulip_bots\bots\crypto\__init__.py          0      0   100%
zulip_bots\zulip_bots\bots\crypto\crypto.py           41      0   100%
zulip_bots\zulip_bots\bots\crypto\test_crypto.py      24      0   100%
zulip_bots\zulip_bots\custom_exceptions.py             1      0   100%
zulip_bots\zulip_bots\lib.py                         275    200    27%
zulip_bots\zulip_bots\request_test_lib.py             61     25    59%
zulip_bots\zulip_bots\simple_lib.py                   68     34    50%
zulip_bots\zulip_bots\test_file_utils.py              14      0   100%
zulip_bots\zulip_bots\test_lib.py                    116     29    75%
----------------------------------------------------------------------
TOTAL                                               1179    735    38%
(zulip-api-py3-venv)
```

Pytest Suite:



```
Tim@DESKTOP-T4AGRK5 MINGW64 ~/Desktop/Desktop/W22/481 SE/HW6/source/python-zulip-api (main)
$ pytest zulip_bots
============================== test session starts ==============================
platform win32 -- Python 3.9.12, pytest-7.1.1, pluggy-1.0.0
rootdir: C:\Users\Tim\Desktop\Desktop\W22\481 SE\HW6\source\python-zulip-api\zulip_bots
plugins: cov-3.0.0
collected 426 items

zulip_bots\zulip_bots\bots\baremetrics\test_baremetrics.py ............. [  3%]
...                                                                      [  3%]
zulip_bots\zulip_bots\bots\beeminder\test_beeminder.py ............      [  6%]
zulip_bots\zulip_bots\bots\chessbot\test_chessbot.py ...                  [  7%]
zulip_bots\zulip_bots\bots\connect_four\test_connect_four.py ......      [  8%]
zulip_bots\zulip_bots\bots\converter\test_converter.py ...               [  9%]
zulip_bots\zulip_bots\bots\crypto\test_crypto.py ........                [ 11%]
zulip_bots\zulip_bots\bots\define\test_define.py ....                    [ 12%]
zulip_bots\zulip_bots\bots\dialogflow\test_dialogflow.py ........        [ 14%]
zulip_bots\zulip_bots\bots\dropbox_share\test_dropbox_share.py ......... [ 16%]
..............                                                           [ 19%]
zulip_bots\zulip_bots\bots\encrypt\test_encrypt.py ...                   [ 20%]
zulip_bots\zulip_bots\bots\file_uploader\test_file_uploader.py ......    [ 21%]
zulip_bots\zulip_bots\bots\flock\test_flock.py ..........                [ 23%]
zulip_bots\zulip_bots\bots\followup\test_followup.py .....               [ 25%]
zulip_bots\zulip_bots\bots\front\test_front.py .................         [ 29%]
zulip_bots\zulip_bots\bots\game_handler_bot\test_game_handler_bot.py ... [ 29%]
......................................                                   [ 38%]
zulip_bots\zulip_bots\bots\game_of_fifteen\test_game_of_fifteen.py ..... [ 40%]
.                                                                        [ 40%]
zulip_bots\zulip_bots\bots\giphy\test_giphy.py ........                  [ 42%]
zulip_bots\zulip_bots\bots\github_detail\test_github_detail.py ......... [ 44%]
...                                                                      [ 45%]
zulip_bots\zulip_bots\bots\google_search\test_google_search.py .......   [ 46%]
zulip_bots\zulip_bots\bots\google_translate\test_google_translate.py ... [ 47%]
...........                                                              [ 50%]
zulip_bots\zulip_bots\bots\helloworld\test_helloworld.py ...             [ 50%]
zulip_bots\zulip_bots\bots\help\test_help.py ...                         [ 51%]
zulip_bots\zulip_bots\bots\idonethis\test_idonethis.py ..........        [ 53%]
zulip_bots\zulip_bots\bots\incrementor\test_incrementor.py ....          [ 54%]
zulip_bots\zulip_bots\bots\jira\test_jira.py ..................          [ 58%]
zulip_bots\zulip_bots\bots\link_shortener\test_link_shortener.py ......  [ 60%]
zulip_bots\zulip_bots\bots\mention\test_mention.py .......               [ 61%]
zulip_bots\zulip_bots\bots\merels\test_merels.py ........                [ 63%]
zulip_bots\zulip_bots\bots\merels\test\test_constants.py ..              [ 64%]
zulip_bots\zulip_bots\bots\merels\test\test_database.py ......           [ 65%]
zulip_bots\zulip_bots\bots\merels\test\test_game.py ............         [ 68%]
zulip_bots\zulip_bots\bots\merels\test\test_interface.py ....            [ 69%]
zulip_bots\zulip_bots\bots\merels\test\test_mechanics.py ............    [ 72%]
zulip_bots\zulip_bots\bots\monkeytestit\test_monkeytestit.py ....        [ 73%]
zulip_bots\zulip_bots\bots\salesforce\test_salesforce.py ..............  [ 76%]
zulip_bots\zulip_bots\bots\stack_overflow\test_stack_overflow.py ...     [ 76%]
zulip_bots\zulip_bots\bots\susi\test_susi.py ....                        [ 77%]
zulip_bots\zulip_bots\bots\tictactoe\test_tictactoe.py .............     [ 80%]
zulip_bots\zulip_bots\bots\trello\test_trello.py ...........             [ 83%]
zulip_bots\zulip_bots\bots\trivia_quiz\test_trivia_quiz.py ............  [ 86%]
zulip_bots\zulip_bots\bots\twitpost\test_twitpost.py ....                [ 87%]
zulip_bots\zulip_bots\bots\virtual_fs\test_virtual_fs.py .......         [ 88%]
zulip_bots\zulip_bots\bots\weather\test_weather.py ...                   [ 89%]
zulip_bots\zulip_bots\bots\wikipedia\test_wikipedia.py ...               [ 90%]
zulip_bots\zulip_bots\bots\witai\test_witai.py ...                       [ 91%]
zulip_bots\zulip_bots\bots\xkcd\test_xkcd.py .......                     [ 92%]
zulip_bots\zulip_bots\bots\yoda\test_yoda.py ...                         [ 93%]
zulip_bots\zulip_bots\bots\youtube\test_youtube.py .........             [ 95%]
zulip_bots\zulip_bots\tests\test_finder.py .                             [ 95%]
zulip_bots\zulip_bots\tests\test_lib.py ...........                      [ 98%]
zulip_bots\zulip_bots\tests\test_run.py .......                          [100%]

=============================== warnings summary ===============================
zulip_bots/bots/twitpost/test_twitpost.py::TestTwitpostBot::test_bot_responds_to_empty_messag
zulip_bots/bots/twitpost/test_twitpost.py::TestTwitpostBot::test_bot_usage
zulip_bots/bots/twitpost/test_twitpost.py::TestTwitpostBot::test_help
zulip_bots/bots/twitpost/test_twitpost.py::TestTwitpostBot::test_tweet
  C:\Users\Tim\Desktop\Desktop\W22\481 SE\HW6\source\python-zulip-api\zulip-api-py3-venv\lib\
    warnings.warn(

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
======================= 426 passed, 4 warnings in 1.42s ========================
```

10. **[*Plan Updates*]**

It is safe to say that the reality of my open-source contribution did not align with my initial plans. I originally anticipated only spending around 6 hours in preliminary research for my original task which was to implement a feature that required extensive work in all parts of the full stack of Zulip's application. It took me around five hours just to realize that I would need way more time to be able to finish my initial task. Thus, my initial sprint board did not reflect the actual allocation of my time since I would no longer be implementing features across the full stack. Below is a more refined time log:

Time Logging (in man-hours)

Preliminary research: ~15 hours

- 3 hours familiarizing myself with Zulip
- 4 hours reading through new contributor documentation
- 5 hours reading through and studying architecture and directory structure for Zulip's main web application.
- 3 hours reading through and studying Zulip's python api repository.

Planning and Implementation: ~11 hours

- 2 hours configuring local development setup/environment
- 2 hours reading existing code to understand framework and expectations of new features
- 1 hour to plan out implementation using API documentation and existing code as a guide.
- 2 hours implementing bot
- 2 hours writing unit tests
- 1 hour testing and linting
- 1 hour finalizing code submission

Report:

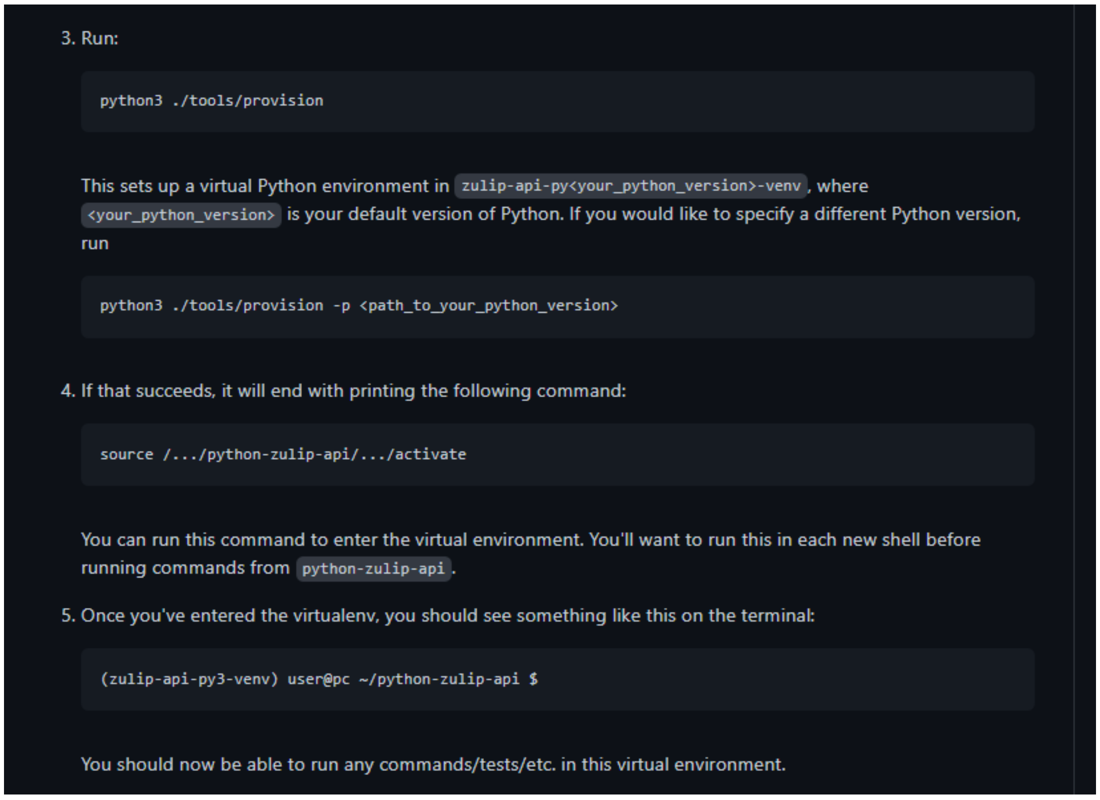
- 1 hour reviewing requirements and reviewing examples
- 4 hours writing report

11. **[*Experiences and Recommendations*]**

The learning curve of the real world.

There is no way to sugarcoat the fact that working with software at the industry level is vastly different from we students' experiences in most computer science courses. I can recall only one or two projects from academic career that exceeded over about 1,000 lines of code; furthermore, the largest group I ever had to work with was five people. On the other hand, a project like Zulip has *thousands of source files* and hundreds of thousands of lines of codes. This is still relatively small in comparison to tech giants like FANG companies who work with repositories containing millions of lines of code. Moreover, operating within a five-person group is much different than working on an open-source project which has essentially endless contributors; it also differs from a typical closed-source company in which there are hundreds of employees.

Thus, it follows that the hardest part of taking on an endeavor in the world of software engineering is often figuring out where to even begin. Zulip fortunately puts forth an earnest effort to provide useful documentation, and they include guides for writing new documentation in their contribution guidelines. Zulip's wiki docs contained hundreds of pages for everything from Git Flow to outreach programs. Thorough documentation is especially important for new onboarding hires or first-time contributors in this case. Thankfully, Zulip's documentation is detailed and easy to follow in this regard; they have several pages dedicated to helping new contributors. I found that I was able to work through local development setup with relative ease. Zulip's docs laid out examples of launching the dev environment on different operating systems. Furthermore, Zulip has put a lot of thought into their scripts and tools (See below). Installing prerequisites and setting up a proper virtual machine environment was as easy as running a couple executables in their tools/ directory. Subsequently, they have also put thought into explaining the architecture and directory structure of the project. It is important to understand how different components of the application work together before diving into writing code.



3. Run:

```
python3 ./tools/provision
```

This sets up a virtual Python environment in `zulip-api-py<your_python_version>-venv`, where `<your_python_version>` is your default version of Python. If you would like to specify a different Python version, run

```
python3 ./tools/provision -p <path_to_your_python_version>
```

4. If that succeeds, it will end with printing the following command:

```
source /.../python-zulip-api/.../activate
```

You can run this command to enter the virtual environment. You'll want to run this in each new shell before running commands from `python-zulip-api`.

5. Once you've entered the virtualenv, you should see something like this on the terminal:

```
(zulip-api-py3-venv) user@pc ~/python-zulip-api $
```

You should now be able to run any commands/tests/etc. in this virtual environment.

Working on a project with experienced developers can be intimidating. I find that I am always hesitant to reach out for help because in fear of wasting some one else's time. However, I've found that everyone is more than happy to help. The Zulip community development server was structured well so that I could ask for help in the right places to get a response faster.

Parallels with internship experience

My experience with contributing to Zulip was awfully similar to my past internship experience at Nexsys Technologies. To no surprise, the hardest part of my internship was getting everything set up and working properly. I spent almost two weeks working with developers to get my local development environment up and running. I found that Zulip had more useful documentation for new contributors; the wiki pages at Nexsys were slightly outdated and sometimes included deprecated requirements. Furthermore, working in a closed-source company requires the acquisition of many permissions. I found that I had to reach out to somebody new for just about every tool that I needed to install on my machine. Nevertheless, having to familiarize myself with the existing code base was a difficult and necessary preliminary step in both cases. One thing that Nexsys offered which other projects/companies should consider was "learn to code" sessions in which a senior developer delved into the details of the implementation of a particular section of the code base.

Course concepts in practice

Often in computer science courses, we students are taught concepts deemed fundamental to the field of computer science, but what we learned is only applied to passing a rigorous exam (cough 203 cough) or small group project. It was rewarding to see the concepts taught in this course are actually relevant in the world of software engineering beyond university. For example, this course stressed the fact that maintenance comprised the majority of the software engineering process. Zulip, too emphasized the importance of submitting code that was readable and maintainable. They wrote many guidelines to ensure that new contributors had style suggestions to follow to ensure that code changes were held to a high standard of readability. Furthermore, the Zulip api docs contained examples of existing models and frameworks so that new contributors could easily follow a similar structure. The api docs on "writing your own bots" detailed how to use the BotHandler and how to structure unit tests. These guidelines all contribute to a more maintainable code base. Furthermore, Zulip laid out recommendations for commit messages and submitting pull requests. These instructions also increase maintainability because future contributors could more easily understand why changes took place. Zulip also put quality metrics and static analysis tools covered in this class to use. Documentation suggested that unit tests should strive for %100 statement coverage and code changes were subject to linters as well as code review.

Recommendations
I would recommend abiding to a fifteen-minute-rule when delving into an open-source project for the first time; if you are stuck on something for more than fifteen minutes, go ask someone for help. This will save you a lot of time, so you can put your efforts to actually contributing. As far as selecting a task to undertake, it will inevitably be more work than you initially think. Thus, I would advise choosing a project that uses tools, languages, frameworks, etc. that you are familiar with.

12. **[*Advice for Future Students*]**
This class has a deceivingly heavy workload, though it is not the kind of work CS students are accustomed to. If you are taking this class because it is categorized as having a low-moderate workload, I would see 493. Other classes don't test your ability to comprehend technical papers nor do they have as open-ended of project specifications.

13. **[*Future Use of Material*]**
I am willing to allow future students to use my material.