REST-API Tutorial
7 months ago

How to Mock a Rest API in Python with request-mock

We will build upon the sources of the Jira time report generator. We are using Python 3.7 and PyCharm as IDE. First, let's create a `test` directory and right-click the directory in PyCharm. Choose `New - Python File` and `Python unit test`. This creates the following default file:

```
import unittest

class MyTestCase(unittest.TestCase):
    def test_something(self):
        self.assertEqual(True, False)


if __name__ == '__main__':
    unittest.main()
```

Running this unit test obviously fails (True does not equals False), but we do have set up the basics for writing our own unit tests now.

Mocking a Rest API

contains a call to the Jira Rest API. We do not want our unit test to be dependent of a third party service and therefore we need a way to mock the Rest API. There are several options to mock a Rest API, but we will make use of the requests-mock Python library which fits our needs.

Install the `requests-mock` Python library:

```
pip install requests_mock
```

Test Single Page Response

The `get_updated_issues` function will request the issues which are updated in a certain time period. In our unit test, we will verify the behavior when one page with results is retrieved (the Rest API supports pagination, but that is something for a next unit test):

```
def test_get_updated_issues_one_page(self):
    with open("issues_one_page.json", "r") as issues_file:
        mock_response = issues_file.read()

    expected_result = [
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10005',
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10004',

    with requests_mock.Mocker() as m:
        m.register_uri('GET', '/rest/api/2/search', text=moc
        response = jiratimereport.get_updated_issues("https:
```

Let’s take a closer look at what is happening here. We have defined the Jira JSON response in file `issues_one_page.json`. At line 2 and 3, we read the contents of the file into variable `mock_response`. In line 5, we define the expected result with variable `expected_result` when the function `get_updated_issues` returns. At lines 11 up to 13 the magic happens. We register the URI we call from within the `get_updated_issues` function with the `Mocker` we defined. The third parameter of `register_uri` defines the response which should be returned from the mocked API call. At line 15 we call the `get_updated_issues` function and at the end we verify whether the response equals the expected result.

Test Paginated Response

The Jira API supports pagination. We added some functionality to the `get_updated_issues` function in order to handle paginated responses. The JSON response contains three fields for this:

- `startAt`: indicates from which result the page should be retrieved;
- `maxResults`: the number of maximum results which are returned within one response;
- `total`: the total number of results.

The `maxResults` for retrieving issues is, at the time of writing, 50. If we would like to test this against a real Jira server, we would have to create at least 51 issues. But for testing with our unit test, we paginated unit test looks as follows:

```
def test_get_updated_issues_multiple_pages(self):
    with open("issues_multiple_first_page.json", "r") as iss
        mock_response_first_page = issues_first_file.read()

    with open("issues_multiple_second_page.json", "r") as is
        mock_response_second_page = issues_second_file.read(

    expected_result = [
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10005',
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10004',
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10006',

    with requests_mock.Mocker() as m:
        m.register_uri('GET', '/rest/api/2/search', [{'text'
                                                     {'text'
        response = jiratimereport.get_updated_issues("https:
                                                     "2020-0

    self.assertEqual(expected_result, response)
```

The unit test looks quite similar to the one for the single page response. We defined two mock responses this time, because the Jira API will be called twice and we want to return two different responses. The `expected_result` variable contains the combined result of both API calls. The real difference can be seen at line 17.

responses, the latest defined mock response is returned again.

Test Failed

The unit tests above all pass. But do they fail when something is wrong? We can therefore change e.g. the following line in the `get_updated_issues` function:

```
issues_json.extend(response_json['issues'])
```

Change it with:

```
issues_json = response_json['issues']
```

This will ensure that only the response of the first API call will be added to the returned issues list, but not the response of succeeding API calls. Run both tests again. The `test_get_updated_issues_one_page` passes, but the `test_get_updated_issues_multiple_pages` fails:

```
AssertionError: Lists differ
```

Test Multiple URI's

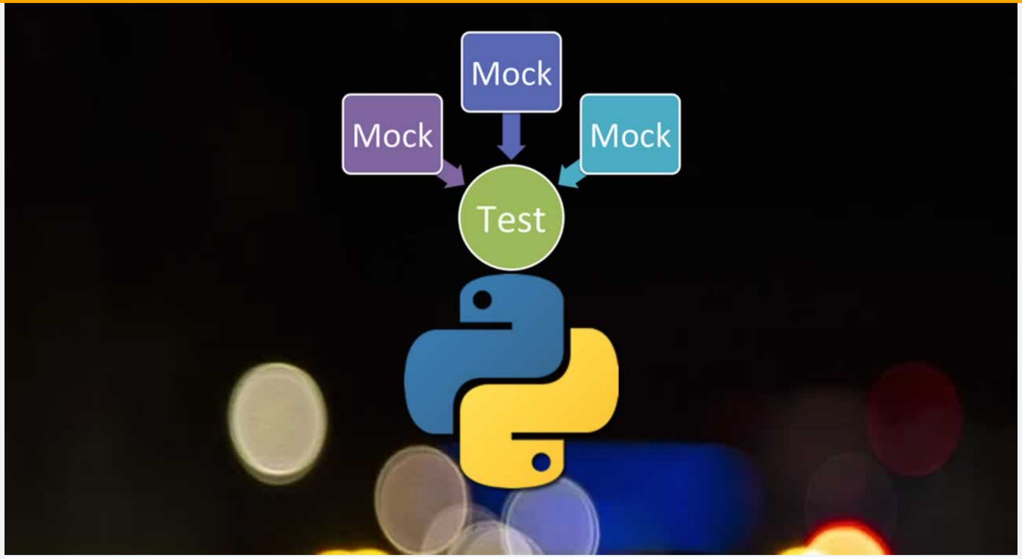
The Jira work logs are to be retrieved per issue. We therefore need to register more than one URI with different responses for each issue. The response of the `get_work_logs` function returns a list of `WorkLog` objects which we can assert.

```
        mock_response_first_issue = first_issue_file.read()

    with open("work_logs_second_issue_one_page.json", "r") a
        mock_response_second_issue = second_issue_file.read(

    issues_json = [
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10005',
        {'expand': 'operations,versionedRepresentations,edit
         'self': 'https://jira_url/rest/api/2/issue/10004',

    with requests_mock.Mocker() as m:
        m.register_uri('GET', '/rest/api/2/issue/MYB-5/workl
        m.register_uri('GET', '/rest/api/2/issue/MYB-4/workl
        work_logs = jiratimereport.get_work_logs("https://ji
                                                 "2020-01-1(

    self.assertEqual(work_logs[0], WorkLog("MYB-5", datetime
    self.assertEqual(work_logs[1], WorkLog("MYB-5", datetime
    self.assertEqual(work_logs[2], WorkLog("MYB-4", datetime
```

Conclusion

Writing unit tests is absolutely necessary when you want to develop software in a professional manner. In this post, we took a look at how to unit test a Rest API by means of the `requests-mock` Python library. We only scratched the surface of what this library has to offer, but our first impressions are very good.

Thank you for reading!



9.85 GEEK

Welcome to Morioh!

Let's write, share knowledge and earn GeekCash.

Sign up!

Complete Python Course: Zero to Mastery

739 students enrolled

FREE

Python Django Full Stack Web Developer

3,569 students enrolled

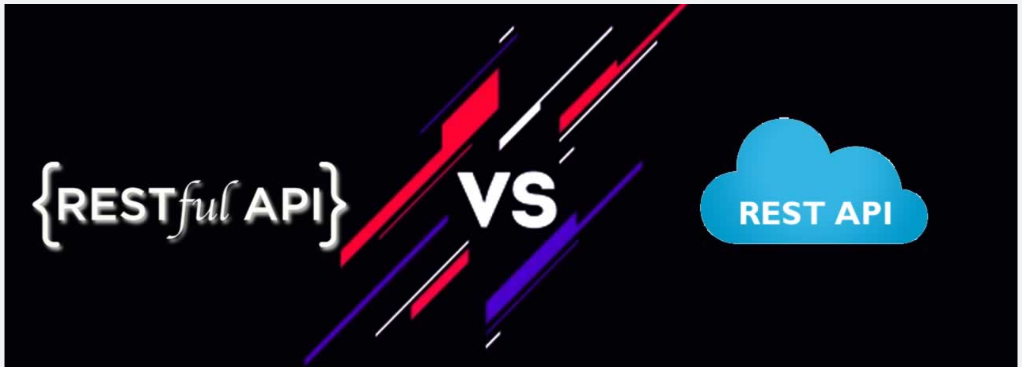
FREE

Python Programming : Object Oriented Programming

4,248 students enrolled FREE

Python For Data Science

21,825 students enrolled FREE



API?

1 164.15 GEEK

What the difference between REST API and RESTful API?

1 608.75 GEEK

Deploying a Serverless REST API with Python, Lambda, and DynamoDB

18.55 GEEK

A Quick Introduction to REST API

36.75 GEEK

Build Face Recognition as a REST API on Linux servers using Python Flask

An Introduction to REST and RESTful APIs

4.90 GEEK

MANSCAPED
Black Friday
25
SHOP NOW