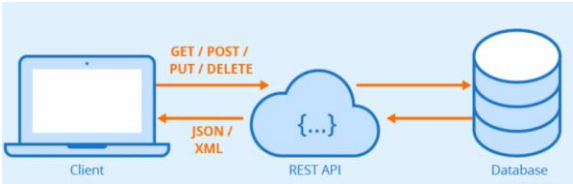
Test Automation using:

- Python
- PyTest
- Visual Studio Code
- Libraries (ie., requests, json, uuid, os, datetime)
- Git and GitHub repository

Document highlight:

- API Automation framework (GET method) using PyTest, Python, and Requests libraries (in Visual Studio Code)
- API Automation framework (POST method) using PyTest, Python, and Requests libraries (in Visual Studio Code)
- API Automation framework (PUT, DELETE method) plus Dynamic Data handling in POST method using PyTest, Python, and Requests libraries (in Visual Studio Code)
- GitHub setup (Create, Stage, Commit, and Publish with VS Code)
- CI with GitHub Actions

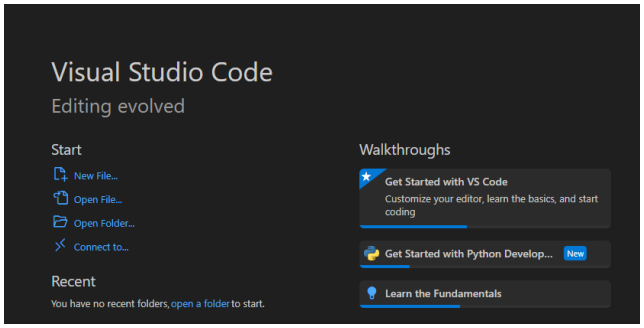
1. API Concept (interface between client and server (database), request and response)



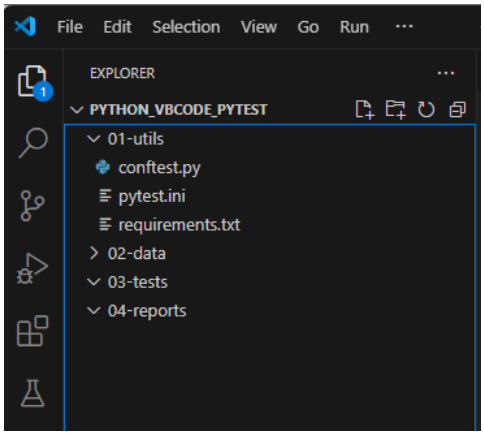
2. API Automation framework (GET method) using PyTest request libraries (in Visual Studio Code)

Ensure that the Visual Studio Code is setup

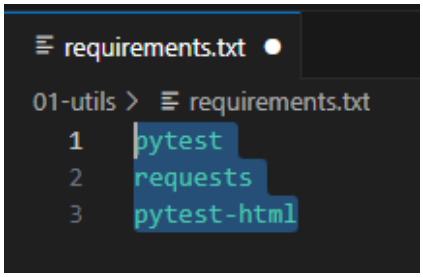
Download:
https://code.visualstudio.com/download



With the following files

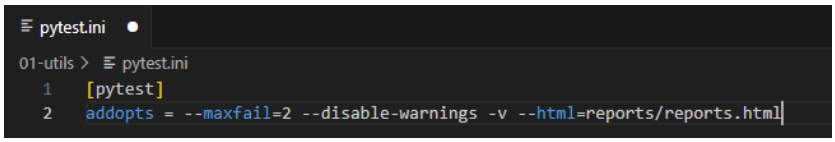


And part of the setup, the following libraries are installed (requirements.txt)

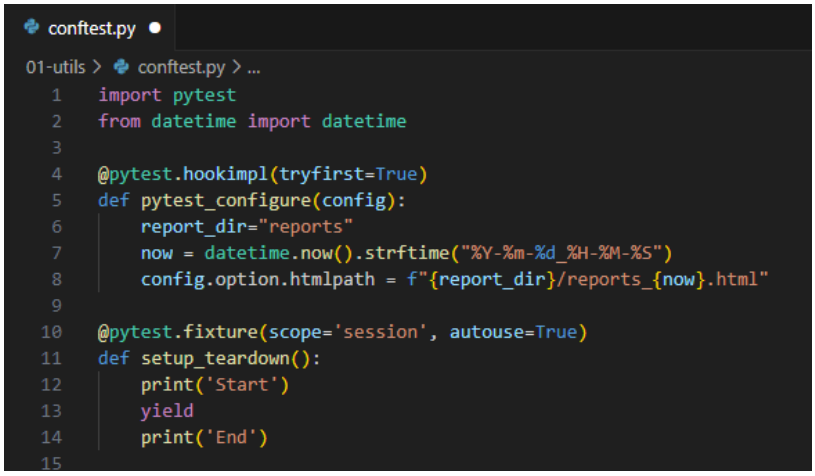


Further, the following are outlined in the following files:

pytest.ini



conftest.py



apis.py

```python
import requests

class APIs:
    BASE_URL= "https://jsonplaceholder.typicode.com"

    def __init__(self):
        self.header = {
            "Content-Type":"application/json"
        }

    def get(self,endpoint):
        url=f'{self.BASE_URL}/{endpoint}'
        getResponse = requests.get(url, headers=self.header)
        return getResponse
```

3. Sampling source of get users

url:
https://jsonplaceholder.typicode.com/

JSONPlaceholder Guide Sponsor this project Blog My JSON Server

{JSON} Placeholder

Free fake and reliable API for testing and prototyping.

Powered by JSON Server + LowDB.

Serving ~3 billion requests each month.

Resources

JSONPlaceholder comes with a set of 6 common resources:

/posts	100 posts
/comments	500 comments
/albums	100 albums
/photos	5000 photos
/todos	200 todos
/users	10 users

4. Run a test case in Visual Studio Code (get the same set of data (users))

5. Run the PyTest command to get expected response results and response code

Terminal:

```
(venv) C:\Users\njmlo\Desktop\Test Automation notes\Python_VBCode_PyTest>pytest -s
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

phone': '1-463-123-4447', 'website': 'ramiro.info', 'company': {'name': 'Romaguera-Jacobson', 'catchPhrase': 'Face to face bifurcated interface', 'bs': 'e-enable strategic applications'}}, {'id': 4, 'name': 'Patricia Lebsack', 'username': 'Karianne', 'email': 'Julianne.OConner@kory.org', 'address': {'street': 'Hoeger Mall', 'su
ite': 'Apt. 692', 'city': 'South Elvis', 'zipcode': '53919-4257', 'geo': {'lat': '29.4572', 'lng': '-164.2990'}}, 'phone': '493-170-9623 x156', 'website': 'kale.biz', 'company': {'name': 'Robel-Corkery', 'catchPhrase': 'Multi-tiered zero tolerance productivity', 'bs': 'transition cutting-edge web services'}}, {'id': 5, 'name':
'Chelsey Dietrich', 'username': 'Kamren', 'email': 'Lucio_Hettinger@annie.ca', 'address': {'street': 'Skiles Walks', 'suite': 'Suite 351', 'city': 'Roscoeview', 'zipcode': '33263', 'geo': {'lat': '-31.8129', 'lng': '62.5342'}}, 'phone': '(254)954-1289', 'website': 'demarco.info', 'company': {'name': 'Keebler LLC', 'catchPhrase'
: 'User-centric fault-tolerant solution', 'bs': 'revolutionize end-to-end systems'}}, {'id': 6, 'name': 'Mrs. Dennis Schulist', 'username': 'Leopoldo_Corkery', 'email': 'Karley_Dach@jasper.info', 'address': {'street': 'Norberto Crossing', 'suite': 'Apt. 950', 'city': 'South Christy', 'zipcode': '23505-1337', 'geo': {'lat': '-71
.4197', 'lng': '71.7478'}}, 'phone': '1-477-935-8478 x6430', 'website': 'ola.org', 'company': {'name': 'Considine-Lockman', 'catchPhrase': 'Synchronised bottom-line interface', 'bs': 'e-enable innovative applications'}}, {'id': 7, 'name': 'Kurtis Weissnat', 'username': 'Elwyn.Skiles', 'email': 'Telly.Hoeger@billy.biz', 'address
': {'street': 'Rex Trail', 'suite': 'Suite 280', 'city': 'Howemouth', 'zipcode': '58804-1099', 'geo': {'lat': '24.8918', 'lng': '21.8984'}}, 'phone': '210.067.6132', 'website': 'elvis.io', 'company': {'name': 'Johns Group', 'catchPhrase': 'Configurable multimedia task-force', 'bs': 'generate enterprise e-tailers'}}, {'id': 8, '
name': 'Nicholas Runolfsdottir V', 'username': 'Maxime_Nienow', 'email': 'Sherwood@rosamond.me', 'address': {'street': 'Ellsworth Summit', 'suite': 'Suite 729', 'city': 'Aliyaview', 'zipcode': '45169', 'geo': {'lat': '-14.3990', 'lng': '-120.7677'}}, 'phone': '586.493.6943 x140', 'website': 'jacynthe.com', 'company': {'name': '
Abernathy Group', 'catchPhrase': 'Implemented secondary concept', 'bs': 'e-enable extensible e-tailers'}}, {'id': 9, 'name': 'Glenna Reichert', 'username': 'Delphine', 'email': 'Chaim_McDermott@dana.io', 'address': {'street': 'Dayna Park', 'suite': 'Suite 449', 'city': 'Bartholomebury', 'zipcode': '76495-3109', 'geo': {'lat': '
24.6463', 'lng': '-168.8889'}}, 'phone': '(775)976-6794 x41206', 'website': 'conrad.com', 'company': {'name': 'Yost and Sons', 'catchPhrase': 'Switchable contextually-based project', 'bs': 'aggregate real-time technologies'}}, {'id': 10, 'name': 'Clementina DuBuque', 'username': 'Moriah.Stanton', 'email': 'Rey.Padberg@karina.bi
z', 'address': {'street': 'Kattie Turnpike', 'suite': 'Suite 198', 'city': 'Lebsackbury', 'zipcode': '31428-2261', 'geo': {'lat': '-38.2386', 'lng': '57.2232'}}, 'phone': '024-648-3804', 'website': 'ambrose.net', 'company': {'name': 'Hoeger LLC', 'catchPhrase': 'Centralized empowering task-force', 'bs': 'target end-to-end model
s'}}]
****Code 200 is passed!****

.

========================================================================== 1 passed in 0.69s ==========================================================================
```

6. Here is the code in text

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 01 - get all users

def test_get_user_validation(apis):
    response = apis.get('users')

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200
    assert len (response.json()) > 0

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")
```

7. API Automation framework (POST method) using PyTest request libraries (in Visual Studio Code)

As a continuation of the prior method framework, reuse and update some of the existing files:

apis.py

```
apis.py
venv > Lib > site-packages > utils > apis.py > APIs > post
import requests

class APIs:
    BASE_URL= "https://jsonplaceholder.typicode.com"

    def __init__(self):
        self.header = {
            "Content-Type":"application/json"
        }

#  **GET method**
    def get(self,endpoint):
        url=f'{self.BASE_URL}/{endpoint}'
        getResponse = requests.get(url, headers=self.header)
        return getResponse

#  **POST method**
    def post(self,endpoint, data):
        url=f'{self.BASE_URL}/{endpoint}'
        postResponse = requests.post(url, headers=self.header, json=data)
        return postResponse
```

8. Run the test case in Visual Studio Code (add a user)

Sampling of no error

```
test_postuser.py
tests > test_postuser.py > ...
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 02 - create (post) a user

def test_post_user_validation(apis):
    user_data ={
        "name": "test user1",
        "username":"test QA1",
        "email":"test@gmail.com"
    }
    response = apis.post('users', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 201

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is added in the database
    assert response.json()['name'] =='test user1'
    print("****User ID/Name " + str(response.json()['id']) + " and " + response.json()['username'] +" is added!****")
```

```
tests\test_postuser.py {'name': 'test user1', 'username': 'test QA1', 'email': 'test@gmail.com', 'id': 11}
****Code 201 is passed!****
****User ID/Name 11 and test QA1 is added!****
```

Sampling with error

```
test_postuser.py ×
tests > test_postuser.py > ...
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 02 - create (post) a user

def test_post_user_validation(apis):
    user_data ={
        "name": "test user1",
        "username":"test QA1",
        "email":"test@gmail.com"
    }
    response = apis.post('users', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 201

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is added in the database
    assert response.json()['name'] ==='test user12'
    print("****User ID/Name " + str(response.json()['id']) + " and " + response.json()['username'] +" is added!****")
```

```
        # ensure that the data key (username) is added in the database
>       assert response.json()['name'] ==='test user12'
E       AssertionError: assert 'test user1' == 'test user12'
E
E         - test user12
E         ?          -
E         + test user1

tests\test_postuser.py:29: AssertionError
========================================================================================================================== short test summary info ==
FAILED tests/test_postuser.py::test_post_user_validation - AssertionError: assert 'test user1' == 'test user12'
```

9. Here is the code in text

```
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 02 - create (post) a user

def test_post_user_validation(apis):
    user_data ={
        "name": "test user1",
        "username":"test QA1",
        "email":"test@gmail.com"
    }
    response = apis.post('users', user_data)

    # display the response
```

```
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 201

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is added in the database
    assert response.json()['name'] ='test user12'
    print("****User ID/Name " + str(response.json()['id']) + " and " +
response.json()['username'] +" is added!****")
```

10. API Automation framework (PUT method) using PyTest, Python, and Requests libraries (in Visual Studio Code)

As a continuation of the prior method framework, reuse and update some of the existing files:

apis.py

```
apis.py
venv > Lib > site-packages > utils > apis.py > APIs > put
import requests

class APIs:
    BASE_URL= "https://jsonplaceholder.typicode.com"

    def __init__(self):
        self.header = {
            "Content-Type":"application/json"
        }

#  **GET method**
    def get(self,endpoint):
        url=f'{self.BASE_URL}/{endpoint}'
        getResponse = requests.get(url, headers=self.header)
        return getResponse

#  **POST method**
    def post(self,endpoint, data):
        url=f'{self.BASE_URL}/{endpoint}'
        postResponse = requests.post(url, headers=self.header, json=data)
        return postResponse

#  **PUT method**
    def put(self,endpoint, data):
        url=f'{self.BASE_URL}/{endpoint}'
        putResponse = requests.put(url, headers=self.header, json=data)
        return putResponse
```

11. Run the test case in Visual Studio Code (update a user)

Sampling of no error

test_putuser.py

tests > test_putuser.py > ...

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 03 - update (put) a user

def test_update_user_validation(apis):
    user_data ={
        "name": "test user1",
        "username":"test QA2",
        "email":"test@gmail.com"
    }
    response = apis.put('users/1', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is updated in the database
    assert response.json()['username'] =='test QA2'
    print("****User ID/Name " + str(response.json()['id']) + " and " + response.json()['username'] +" is updated!****")
```

```
tests\test_putuser.py {'name': 'test user1', 'username': 'test QA2', 'email': 'test@gmail.com', 'id': 1}
****Code 200 is passed!****
****User ID/Name 1 and test QA2 is updated!****
```

Sampling with error

test_putuser.py

tests > test_putuser.py > test_update_user_validation

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 03 - update (put) a user

def test_update_user_validation(apis):
    user_data ={
        "name": "test user1",
        "username":"test QA2",
        "email":"test@gmail.com"
    }
    response = apis.put('users/1', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is updated in the database
    assert response.json()['username'] =='test QA3'
    print("****User ID/Name " + str(response.json()['id']) + " and " + response.json()['username'] +" is updated!****")
```

```
        # ensure that the data key (username) is added in the database
>       assert response.json()['username'] =='test QA3'
E       AssertionError: assert 'test QA2' == 'test QA3'
E
E         - test QA3
E         ?       ^
E         + test QA2
E         ?       ^

tests\test_putuser.py:29: AssertionError
========================================================================================================================================================== short test summary info
FAILED tests/test_putuser.py::test_update_user_validation - AssertionError: assert 'test QA2' == 'test QA3'
```

12. Here is the code in text

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 03 - update (put) a user

def test_update_user_validation(apis):
    user_data ={
        "name": "test user1",
        "username":"test QA2",
        "email":"test@gmail.com"
    }
    response = apis.put('users/1', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is updated in the database
    assert response.json()['username'] =='test QA2'
    print("****User ID/Name " + str(response.json()['id']) + " and " +
response.json()['username'] +" is updated!****")
```

13. API Automation framework (DELETE method) using PyTest, Python, and Requests libraries (in Visual Studio Code)

As a continuation of the prior method framework, reuse and update some of the existing files:

apis.py

```
apis.py
venv > Lib > site-packages > utils > apis.py > ...
import requests

class APIs:
    BASE_URL= "https://jsonplaceholder.typicode.com"

    def __init__(self):
        self.header = {
            "Content-Type":"application/json"
        }

#  **GET method**
    def get(self,endpoint):
        url=f'{self.BASE_URL}/{endpoint}'
        getResponse = requests.get(url, headers=self.header)
        return getResponse

#  **POST method**
    def post(self,endpoint, data):
        url=f'{self.BASE_URL}/{endpoint}'
        postResponse = requests.post(url, headers=self.header, json=data)
        return postResponse

#  **PUT method**
    def put(self,endpoint, data):
        url=f'{self.BASE_URL}/{endpoint}'
        putResponse = requests.put(url, headers=self.header, json=data)
        return putResponse

#  **DELETE method**
    def delete(self,endpoint):
        url=f'{self.BASE_URL}/{endpoint}'
        deleteResponse = requests.put(url, headers=self.header)
        return deleteResponse
```

14. Run the test case in Visual Studio Code (delete a user)

Sampling of no error

test_deleteuser.py

tests > test_deleteuser.py > ...

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 03 - delete a user

def test_delete_user_validation(apis):

    # select and delete the record/data
    response = apis.delete('users/1')

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed (data deleted!****")
```

```
(venv) C:\Users\njmlo\Desktop\Test Automation notes\Python_VBCode_PyTest>pytest -s
============================================================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\njmlo\Desktop\Test Automation notes\Python_VBCode_PyTest
plugins: html-4.1.1, metadata-3.1.1
collected 4 items

tests\test_deleteuser.py {'id': 1}
****Code 200 is passed (data deleted!****
```

Sampling with error

test_deleteuser.py

tests > test_deleteuser.py > test_delete_user_validation

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 03 - delete a user

def test_delete_user_validation(apis):

    # select and delete the record/data
    response = apis.delete('users/1')

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 201

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed (data deleted!****")
```

```
(venv) C:\Users\njmlo\Desktop\Test Automation notes\Python_VBCode_PyTest>pytest -s
==========================================================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\njmlo\Desktop\Test Automation notes\Python_VBCode_PyTest
plugins: html-4.1.1, metadata-3.1.1
collected 4 items

tests\test_deleteuser.py {'id': 1}
F
```

```
        # validate the response code if passed
>       assert response.status_code == 201
E       assert 200 == 201
E        +  where 200 = <Response [200]>.status_code

tests\test_deleteuser.py:19: AssertionError
==========================================================================================
FAILED tests/test_deleteuser.py::test_delete_user_validation - assert 200 == 201
```

15. Here is the code in text

```python
import pytest
from utils.apis import APIs

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 03 - delete a user

def test_delete_user_validation(apis):

    # select and delete the record/data
    response = apis.delete('users/1')

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed (data deleted!****")
```

16. API Automation framework using the dynamic data handling in POST method (in Visual Studio Code)

As a continuation of the prior method framework, reuse and update some of the existing files:

conftest.py

```
conftest.py ×
venv > Lib > site-packages > utils > conftest.py > ...
import pytest
from datetime import datetime
import json
import os


@pytest.hookimpl(tryfirst=True)

def pytest_configure(config):
    report_dir = "reports"
    now = datetime.now().strftime("%Y/%m/%d, %H:%M:%S")
    config.option.htmlpath = f"{report_dir}/report_{now}.html"

@pytest.fixture(scope='session',autouse=True)

def setup_teardown():
    print('Start')
    yield
    print('End')

# to load the data for the dynamic handling
@pytest.fixture
def load_user_data():
    json_file_path = os.path.join(os.path.dirname(__file__), "data", "test_data.json")
    with open(json_file_path) as json_file:
        data = (json.load(json_file))
    return data

```

Add a new file:

test_data.json

```
{} test_data.json ×
data > {} test_data.json > ...
{
    "users_data": {
        "name": "test user1",
        "username":"test QA1",
        "email":"test@gmail.com"
    }
}
```

17. Run the test case in Visual Studio Code (create a user)

Sampling of no error

test_postuser_dyna_handl.py

tests > test_postuser_dyna_handl.py > test_post_user_validation

```python
import pytest
from utils.apis import APIs
import uuid

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 02 - create (post) a user

def test_post_user_validation(apis, load_user_data):

# import test data from a file instead of getting locally from within this file
    user_data = load_user_data["users_data"]
    unique_email = f"{uuid.uuid4().hex[:8]}@yahoo.com"
    user_data["email"] = unique_email
    print("This is now the new email: " + unique_email)

    response = apis.post('users', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 201

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is added in the database
    assert response.json()['name'] =='test user1'
    print("****New Email " + str(response.json()['email']) + " for ID " + str(response.json()['id']) +" is added!****")
```

```
tests\test_postuser_dyna_handl.py This is now the new email: ffd1232f@yahoo.com
{'name': 'test user1', 'username': 'test QA1', 'email': 'ffd1232f@yahoo.com', 'id': 11}
****Code 201 is passed!****
****New Email ffd1232f@yahoo.com for ID 11 is added!****
```

Sampling with error

test_postuser_dyna_handl.py

tests > test_postuser_dyna_handl.py > ...

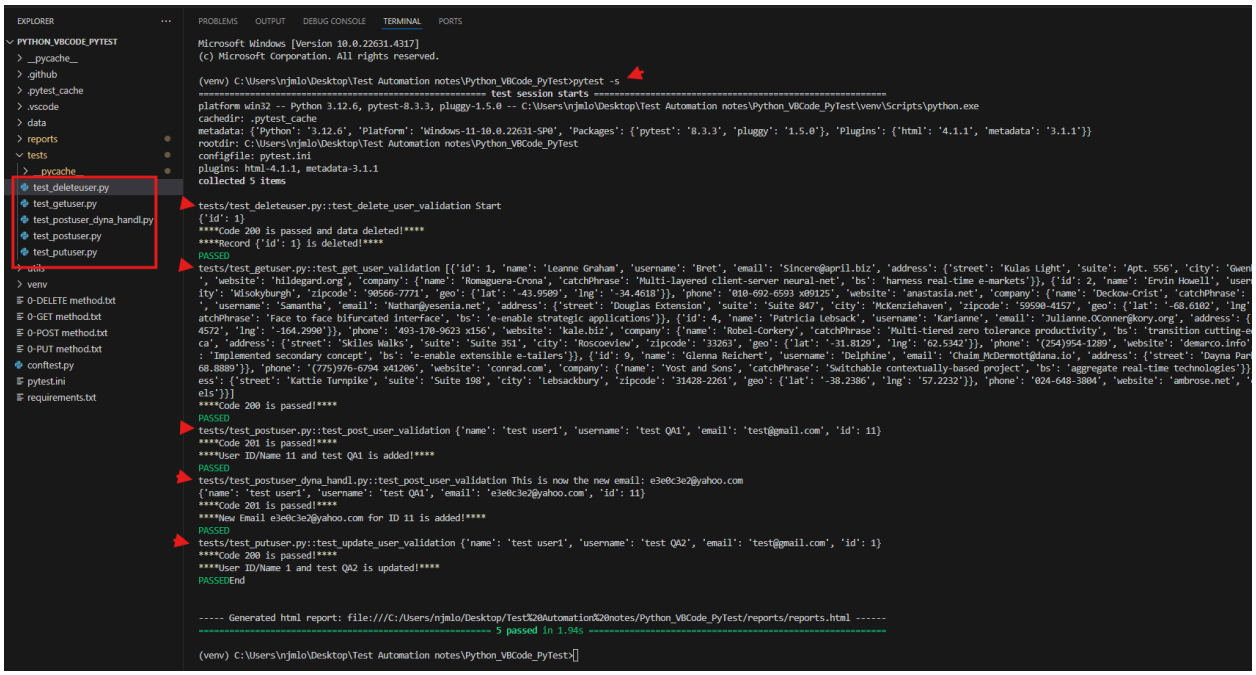
```python
import pytest
from utils.apis import APIs
import uuid

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 02 - create (post) a user

def test_post_user_validation(apis, load_user_data):

# import test data from a file instead of getting locally from within this file
    user_data = load_user_data["users_data"]
    unique_email = f"{uuid.uuid4().hex[:8]}@yahoo.com"
    user_data["email"] = unique_email
    print("This is now the new email: " + unique_email)

    response = apis.post('users', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 200

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is added in the database
    assert response.json()['name'] =='test user1'
    print("****New Email " + str(response.json()['email']) + " for ID " + str(response.json()['id']) +" is added!****")
```

```
        # validate the response code if passed
>       assert response.status_code == 200
E       assert 201 == 200
E        +  where 201 = <Response [201]>.status_code

tests\test_postuser_dyna_handl.py:25: AssertionError
========================================================================================================================== short test summary info ===
FAILED tests/test_postuser_dyna_handl.py::test_post_user_validation - assert 201 == 200
```

18. Here is the code in text

```
import pytest
from utils.apis import APIs
import uuid

@pytest.fixture(scope='module')
def apis():
    return APIs()

# test case 02 - create (post) a user

def test_post_user_validation(apis, load_user_data):

# import test data from a file instead of getting locally from within this file
    user_data = load_user_data["users_data"]
    unique_email = f"{uuid.uuid4().hex[:8]}@yahoo.com"
    user_data["email"] = unique_email
    print("This is now the new email: " + unique_email)

    response = apis.post('users', user_data)

    # display the response
    print(response.json())

    # validate the response code if passed
    assert response.status_code == 201

    # output returned response code and flag as passed
    str_code = str(response.status_code)
    print("****Code " + str_code +" is passed!****")

    # ensure that the data key (username) is added in the database
    assert response.json()['name'] ='test user1'
    print("****New Email " + str(response.json()['email']) + " for ID " +
str(response.json()['id']) +" is added!****")
```

19. Clean (end to end) 5 test cases run

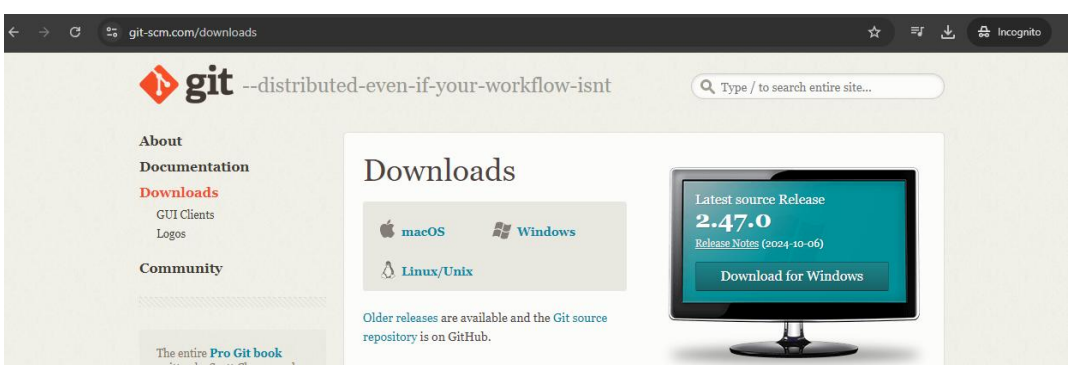


20. GitHub setup (Create, Stage, Commit, and Publish with VS Code)

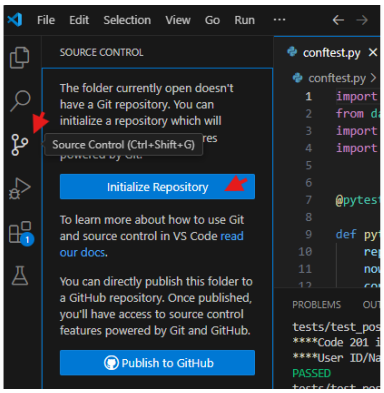
Have a GitHub account

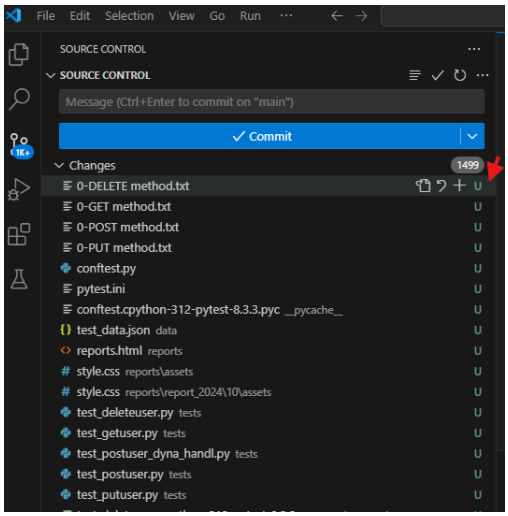
Download and install Git app

Download:
https://git-scm.com/downloads



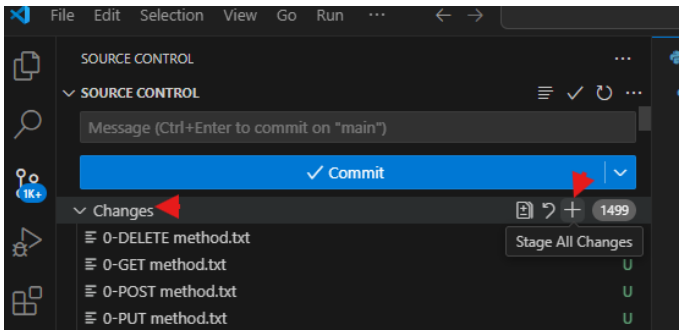
Initialize Repository



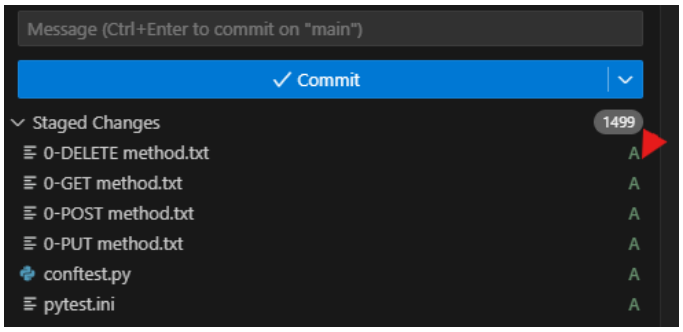


Stage all files

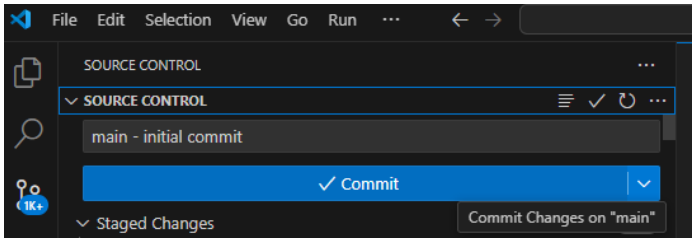
Before

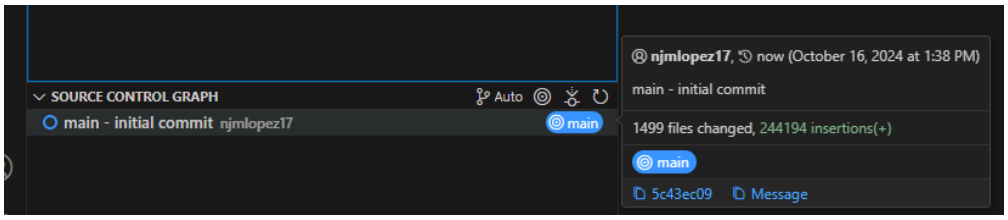


After

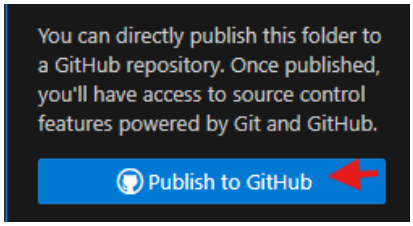


Commit all files

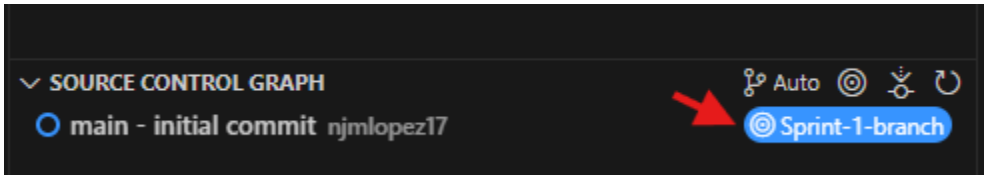




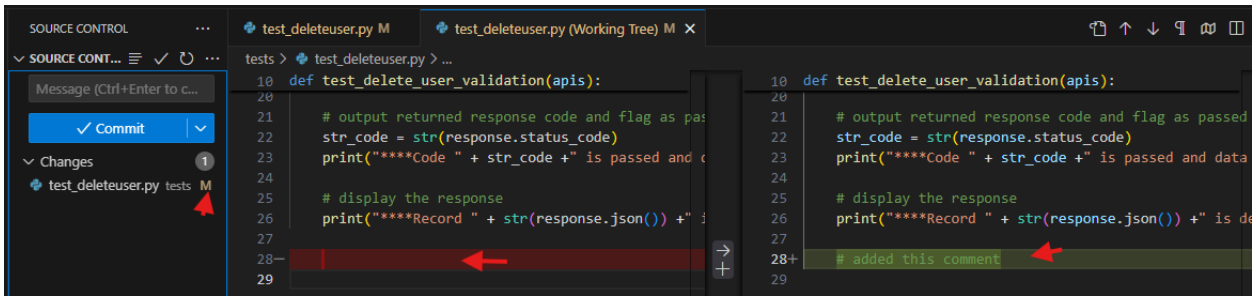
Publish GitHub



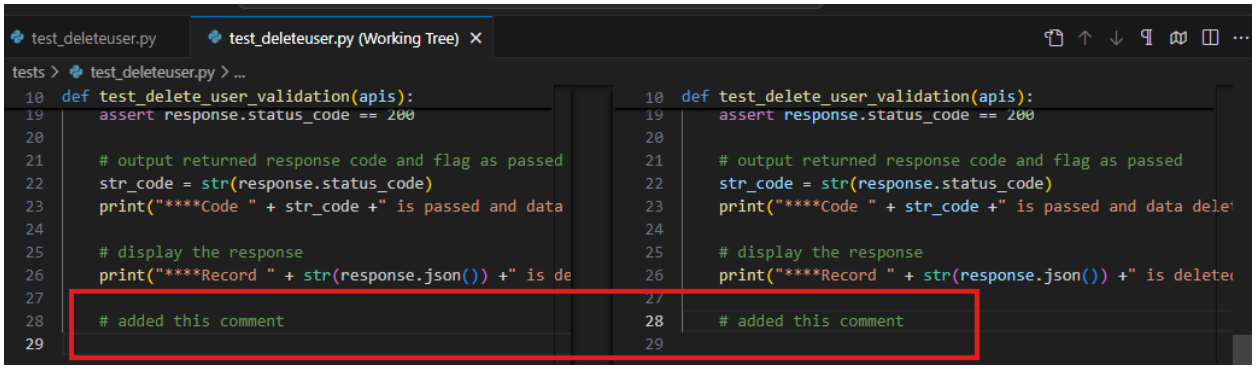
Other than the main branch, create a branch



Sampling code change that is in the branch



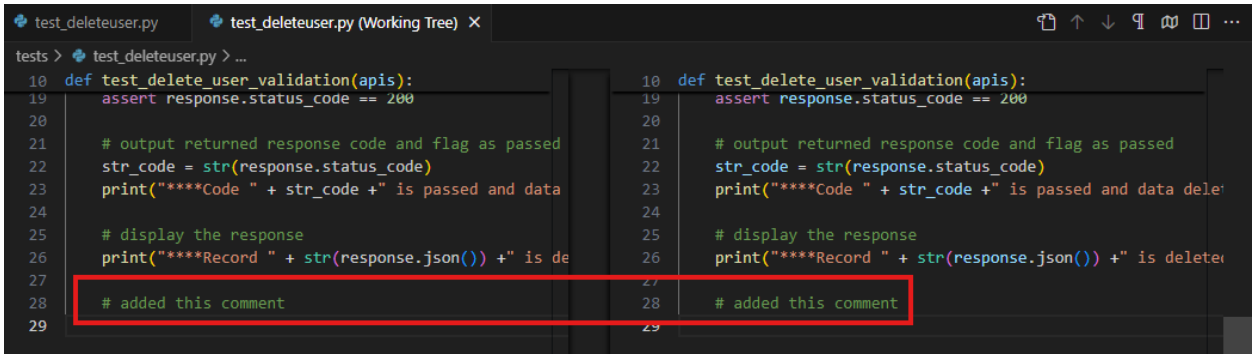
Stage and commit the change



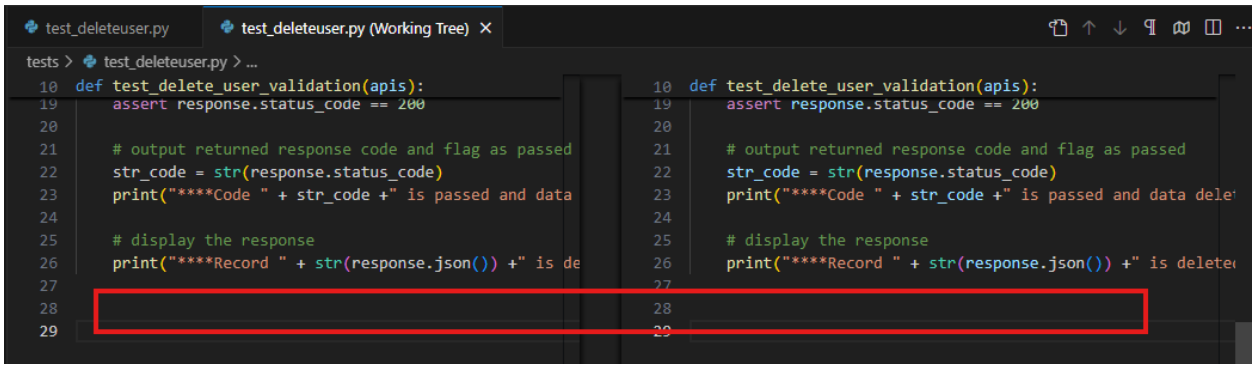
Merge the spring branch into that of the main branch

Before merging:

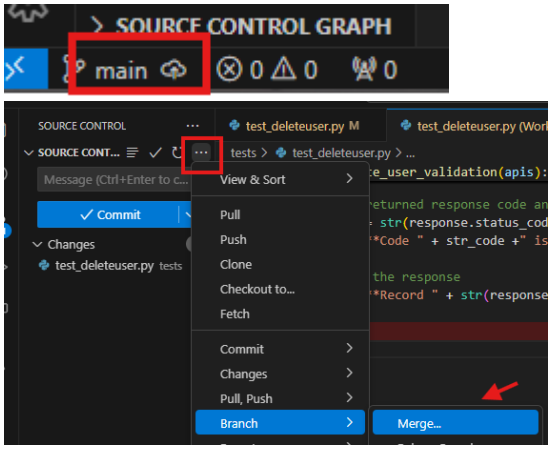
Sprint branch



Main branch

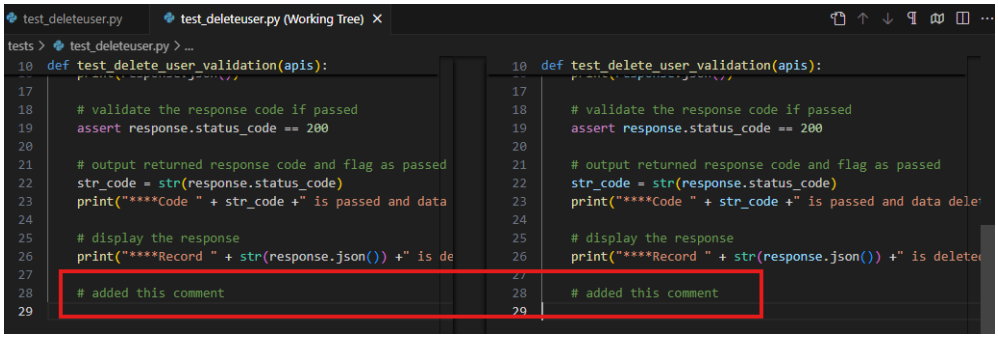


Merge the branch changes

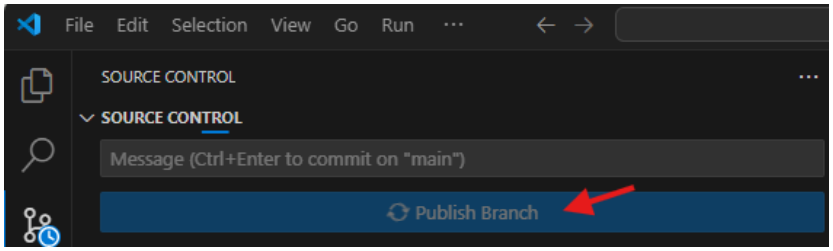


After merging:

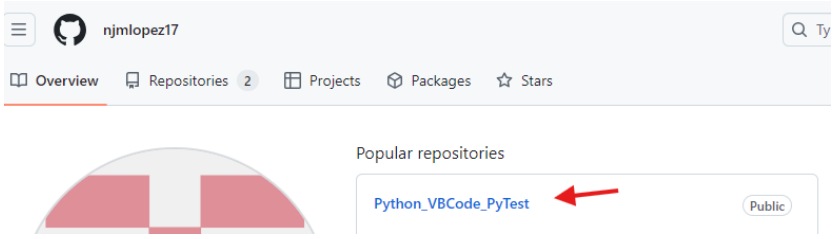
Main branch



Publishing the branch (for this sampling, it is public)

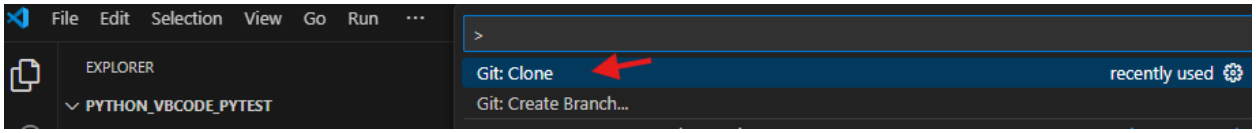


Added in GitHub



Public URL:
https://github.com/njmlopez17/Python_VBCode_PyTest.git

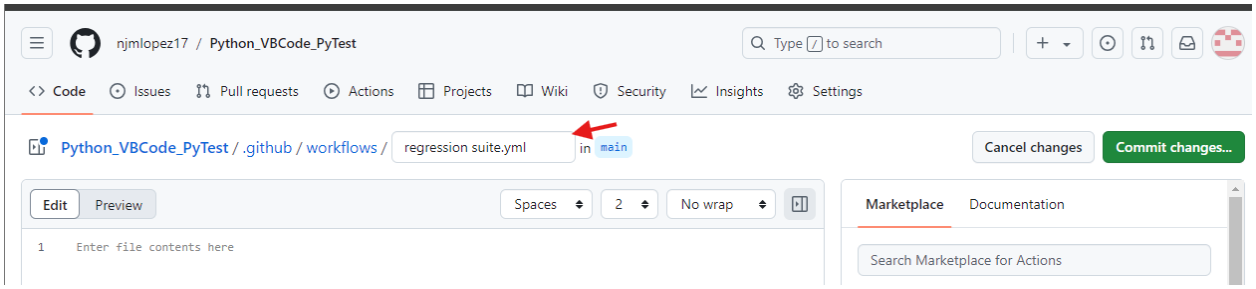
Note: to run this whole sampling of codes, simply clone in Visual Studio Code editor (using the URL above)



21. CI using GitHub Actions

Public URL:
https://github.com/njmlopez17/Python_VBCode_PyTest.git

Create a new file in GitHub (and commit changes)



Code context of the file

njmlopez17 / Python_VBCode_PyTest

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

Files

main

Go to file

- .github/workflows
  - regression suite.yml
- __pycache__
- data
- reports
- tests
- utils
- venv
- 0-DELETE method.txt
- 0-GET method.txt
- 0-POST method.txt
- 0-PUT method.txt
- conftest.py
- pytest.ini
- requirements.txt

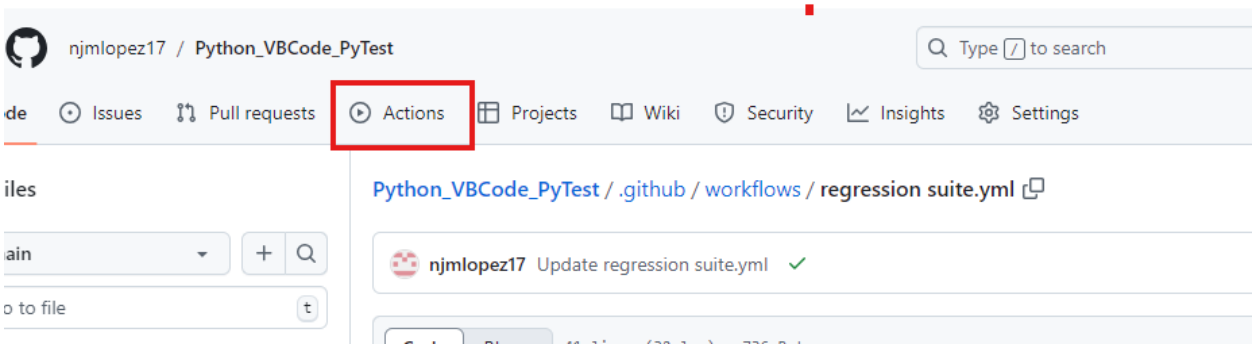
Python_VBCode_PyTest / .github / workflows / regression suite.yml

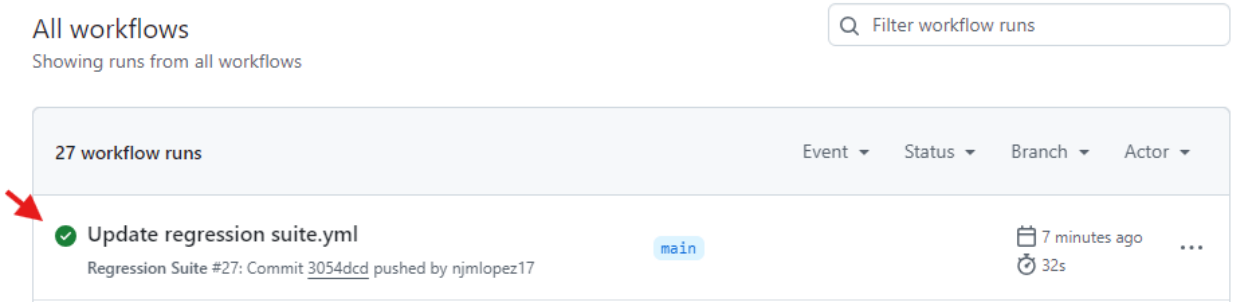
njmlopez17 Update regression suite.yml ✓ 3054dcd · 4 minutes ago History

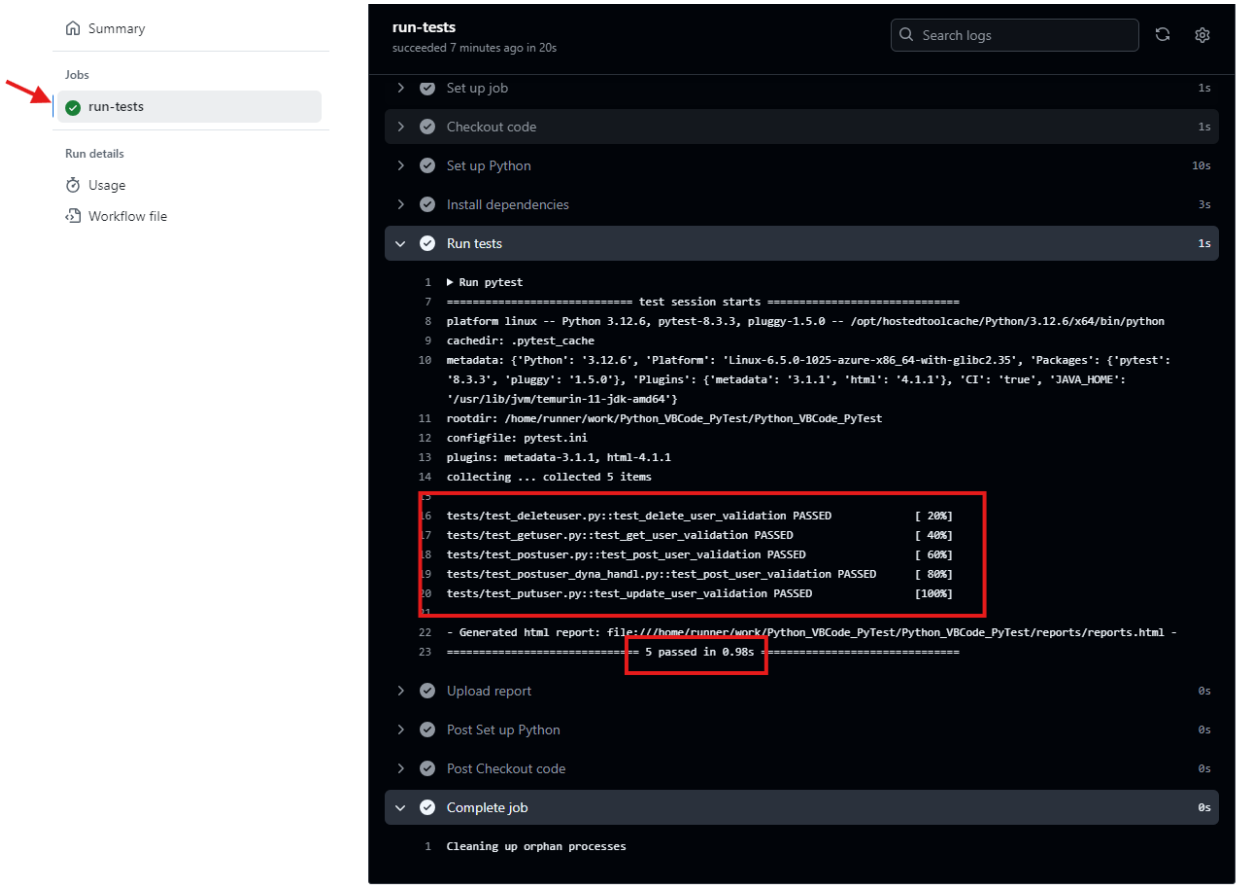
Code | Blame | 41 lines (32 loc) · 736 Bytes | Raw

```yaml
name: Regression Suite

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
  schedule:
    - cron: '30 2 * * *'

jobs:
  run-tests:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: '3.12.6'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run tests
        run: |
          pytest

      - name: Upload report
        uses: actions/upload-artifact@v3
        with:
          name: pytest-report
          path: report.html
```

After a successful run (via Actions button)







**********END**********