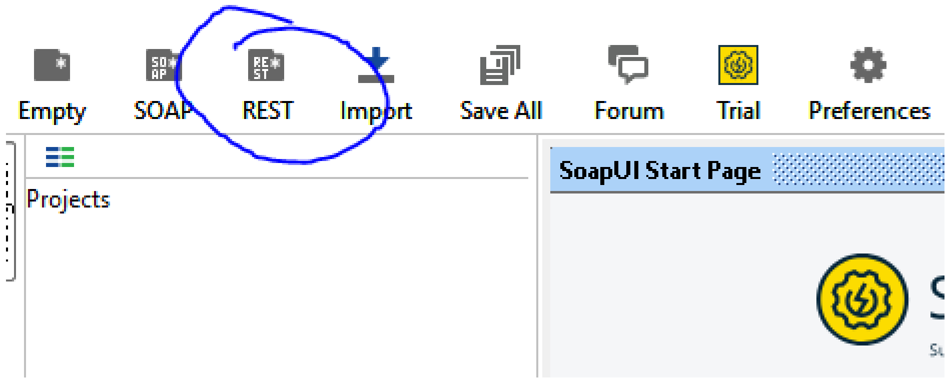
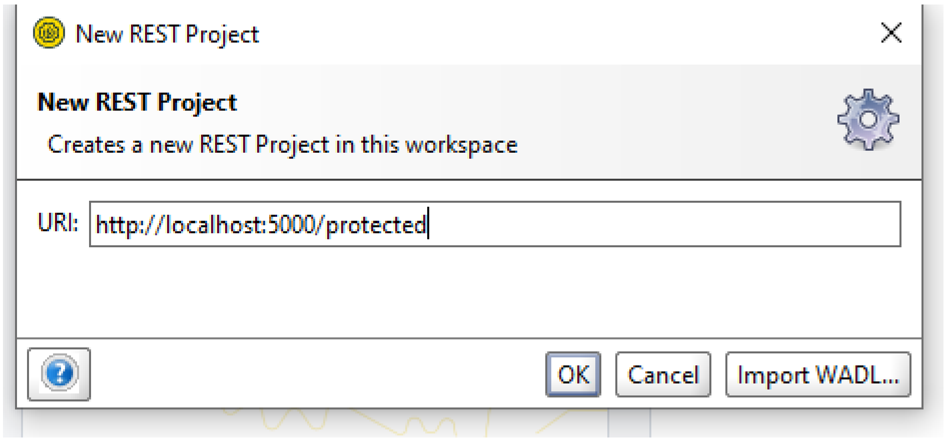
```
Install soapUI from https://www.soapui.org/downloads/soapui/

Install python dependencies
`pip install -r requirements.txt`



Run the flask server
`python3 api_service/app.py `

Generate jwts using the pyJWT library
`python jwt_generator/jwt_utils.py`

These tokens will be written a text file called 'jwt_tokens.txt' and be valid for
an hour
```

When you open soapUI click rest



Then go ahead and enter our protected endpoint being served by our local app server.

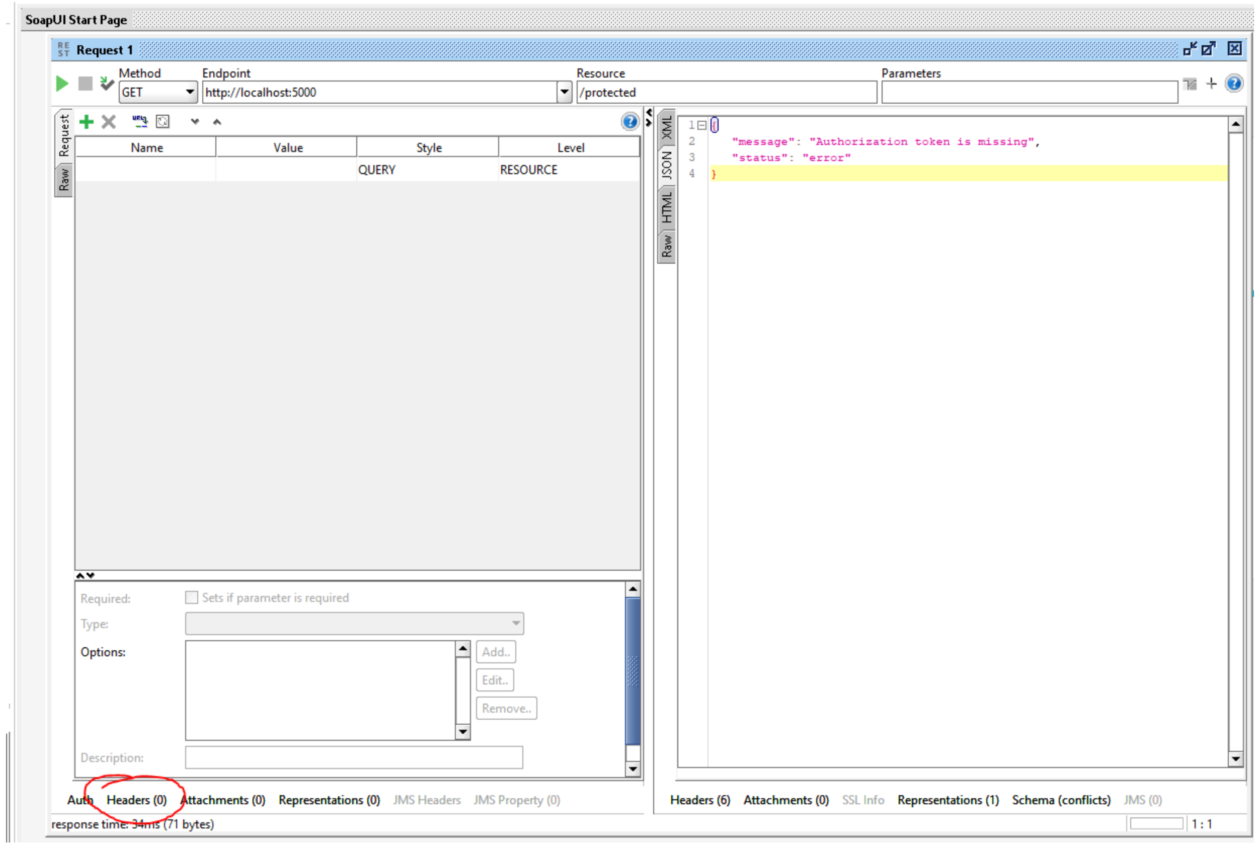


New REST Project

Creates a new REST Project in this workspace

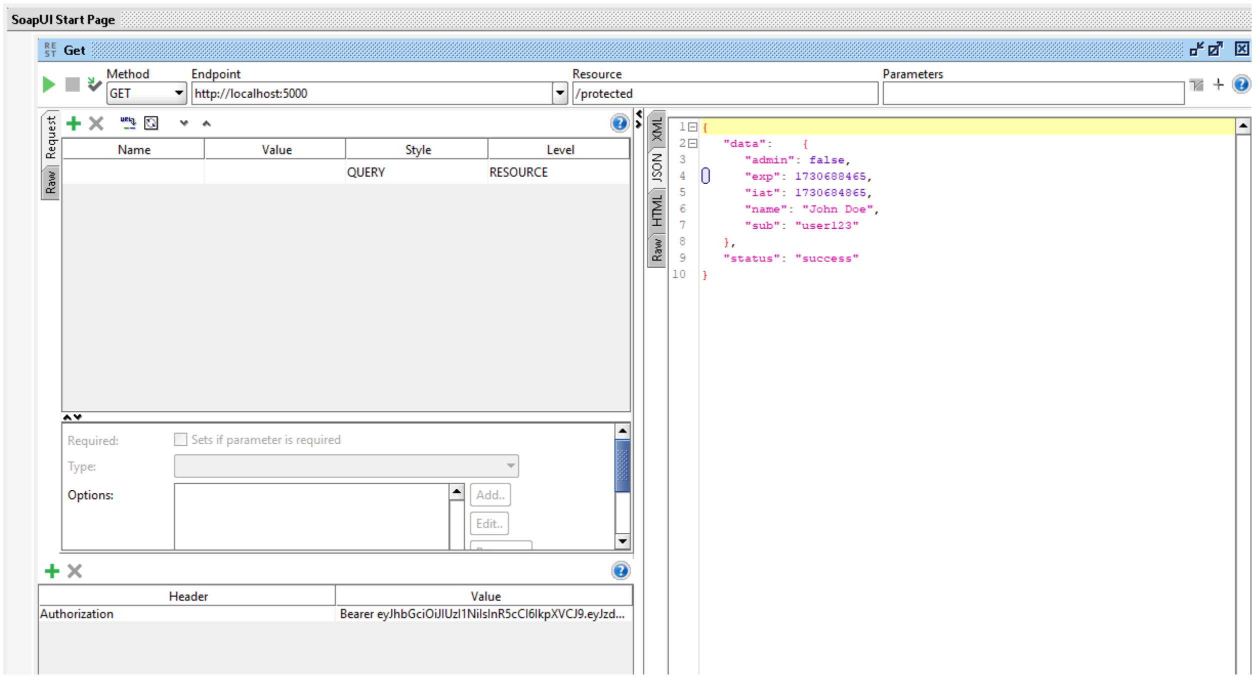
URI: http://localhost:5000/protected

For testing's sake, if you try to run a get request on this endpoint without adding a header you will get an “authorization token missing error.”



```
{
   "message": "Authorization token is missing",
   "status": "error"
}
```

If you add the authorization token, and send another get request we can see that pyJWTs created token was successfully authorized.



Name	Value	Style	Level
		QUERY	RESOURCE

Header	Value
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzd...

```
{
   "data":    {
      "admin": false,
      "exp": 1730688465,
      "iat": 1730684865,
      "name": "John Doe",
      "sub": "user123"
   },
   "status": "success"
}
```

Steps: Ensure that no whitespace is in the header or values or you will get an incorrect token value.

If you are having trouble with this, take a look at this curl request

curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyMTIzIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOmZhbHNlLCJpYXQiOjE3MzA2ODQ4NjUsImV4cCI6MTczMDY4ODQ2NX0.nBXE6Wyzn-NXUmHaYZaLajEtfLA1lG5tdpPrPWVnPGE" http://127.0.0.1:5000/protected

Everything highlighted in green is what is set in our value parameter, including all white space.

Next steps:

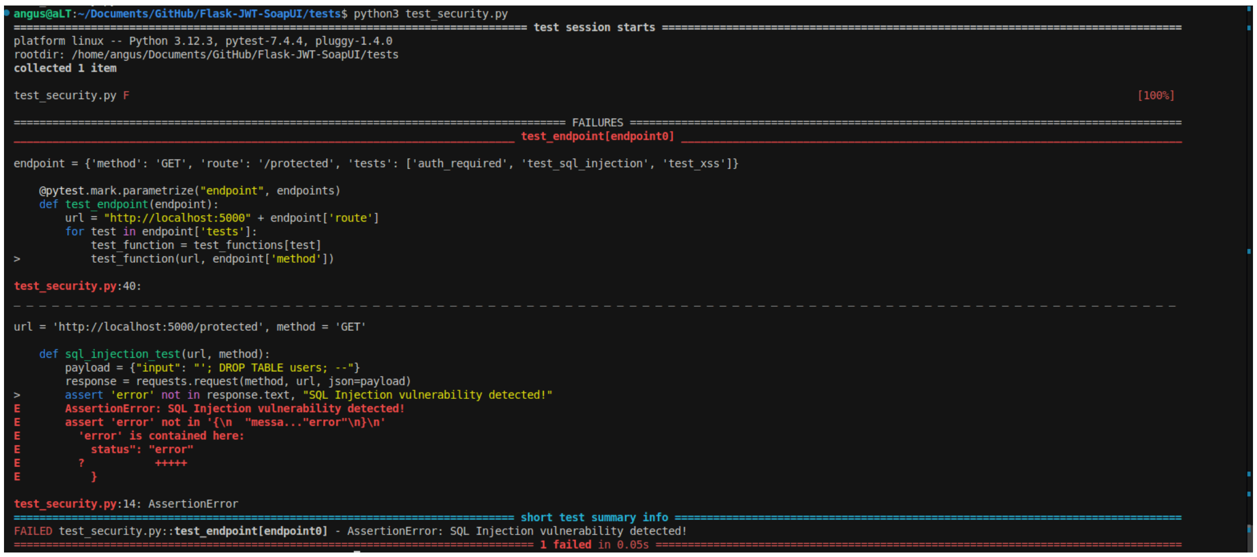
- Security Testing: Leverage SoapUI to conduct thorough security assessments, including testing for common vulnerabilities outlined by the OWASP Top 10, such as SQL injection, XSS, and improper authentication handling.

- Functional Testing: Validate that all API endpoints perform expected functions under various conditions and handle errors gracefully.

- Load and Performance Testing: Analyze how the application performs under stress, using an attempt to check if SoapUI can simulate high traffic and data load for a local server

- Mocking and Service Virtualization: Implement mocks for external services to test the API in isolation, ensuring functionality is preserved even when external dependencies are not available.

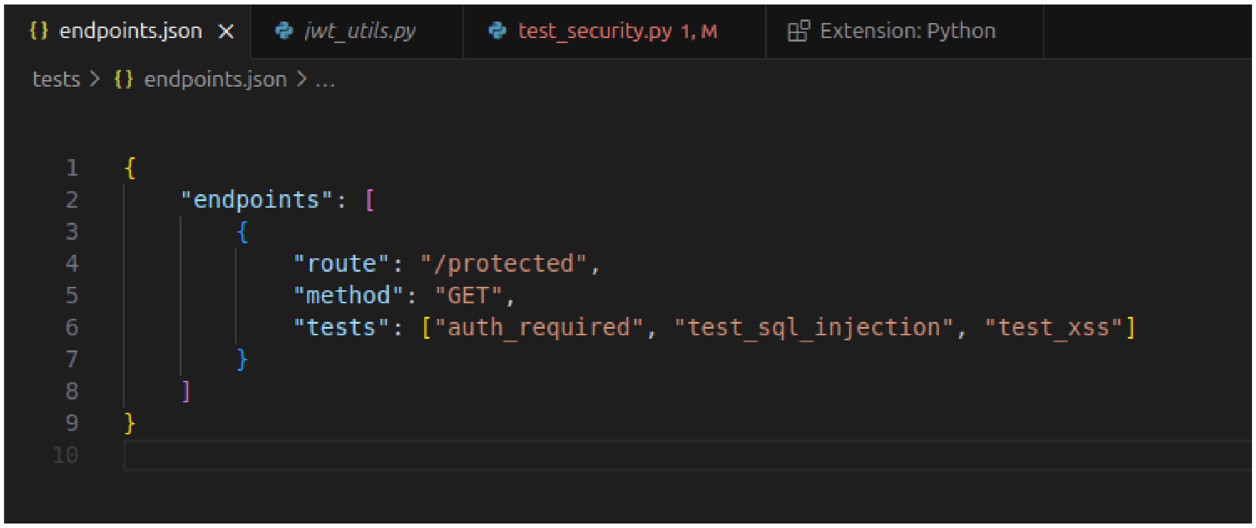
Currently to run endpoint tests you can go to test/security_tests and run it.



```
angus@aLT:~/Documents/GitHub/Flask-JWT-SoapUI/tests$ python3 test_security.py
============================================ test session starts ============================================
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
rootdir: /home/angus/Documents/GitHub/Flask-JWT-SoapUI/tests
collected 1 item

test_security.py F                                                                                    [100%]

================================================= FAILURES ==================================================
_________________________________________ test_endpoint[endpoint0] __________________________________________

endpoint = {'method': 'GET', 'route': '/protected', 'tests': ['auth_required', 'test_sql_injection', 'test_xss']}

    @pytest.mark.parametrize("endpoint", endpoints)
    def test_endpoint(endpoint):
        url = "http://localhost:5000" + endpoint['route']
        for test in endpoint['tests']:
            test_function = test_functions[test]
>           test_function(url, endpoint['method'])

test_security.py:40:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

url = 'http://localhost:5000/protected', method = 'GET'

    def sql_injection_test(url, method):
        payload = {"input": "'; DROP TABLE users; --"}
        response = requests.request(method, url, json=payload)
>       assert 'error' not in response.text, "SQL Injection vulnerability detected!"
E       AssertionError: SQL Injection vulnerability detected!
E       assert 'error' not in '{\n  "messa..."error"\n}\n'
E         'error' is contained here:
E           status": "error"
E         ?         +++++
E           }

test_security.py:14: AssertionError
========================================== short test summary info ==========================================
FAILED test_security.py::test_endpoint[endpoint0] - AssertionError: SQL Injection vulnerability detected!
============================================= 1 failed in 0.05s =============================================
```

As a demonstration this endpoint fails against our sql injection test.

Furthermore test cases will be run and specified on any endpoints included in endpoints.json



```
{
    "endpoints": [
        {
            "route": "/protected",
            "method": "GET",
            "tests": ["auth_required", "test_sql_injection", "test_xss"]
        }
    ]
}
```

As shown this currently is running a test on the protected endpoint in which we secured with pyJWT. In future iterations I would like to add dynamic endpoint discovery.