Analysis of “Pynguin: Automated Unit Test Generation for Python”

Title: Pynguin: Automated Unit Test Generation for Python

DOI: 10.1145/3510454.3516829

Team Members: Oana-Andreea Ilie, Albert-Andrei Havirneanu, Iulia-Diana Groza

Installation and Setup

- Installation is straightforward via Python's package manager pip, though it requires Python version 3.10, necessitating an upgrade for users on older versions. Moreover, not only support for Python < 3.10 was dropped, but neither Python 3.11 is supported yet, so users should stick with Python 3.10, turning dependency management for most projects into a burden.
- Running Pynguin initially presented a challenge. It mandates setting an environment variable PYNGUIN_DANGER_AWARE to acknowledge the potential risks associated with executing arbitrary code. This step, while critical for safety, adds a layer of complexity and could be a stumbling block for less experienced users. Additionally, Pynguin's sensitivity to project structure required careful execution of the command with precise directory paths to successfully generate tests.
- The tool's documentation is comprehensive, yet there's a learning curve in understanding the appropriate configurations and command line options.

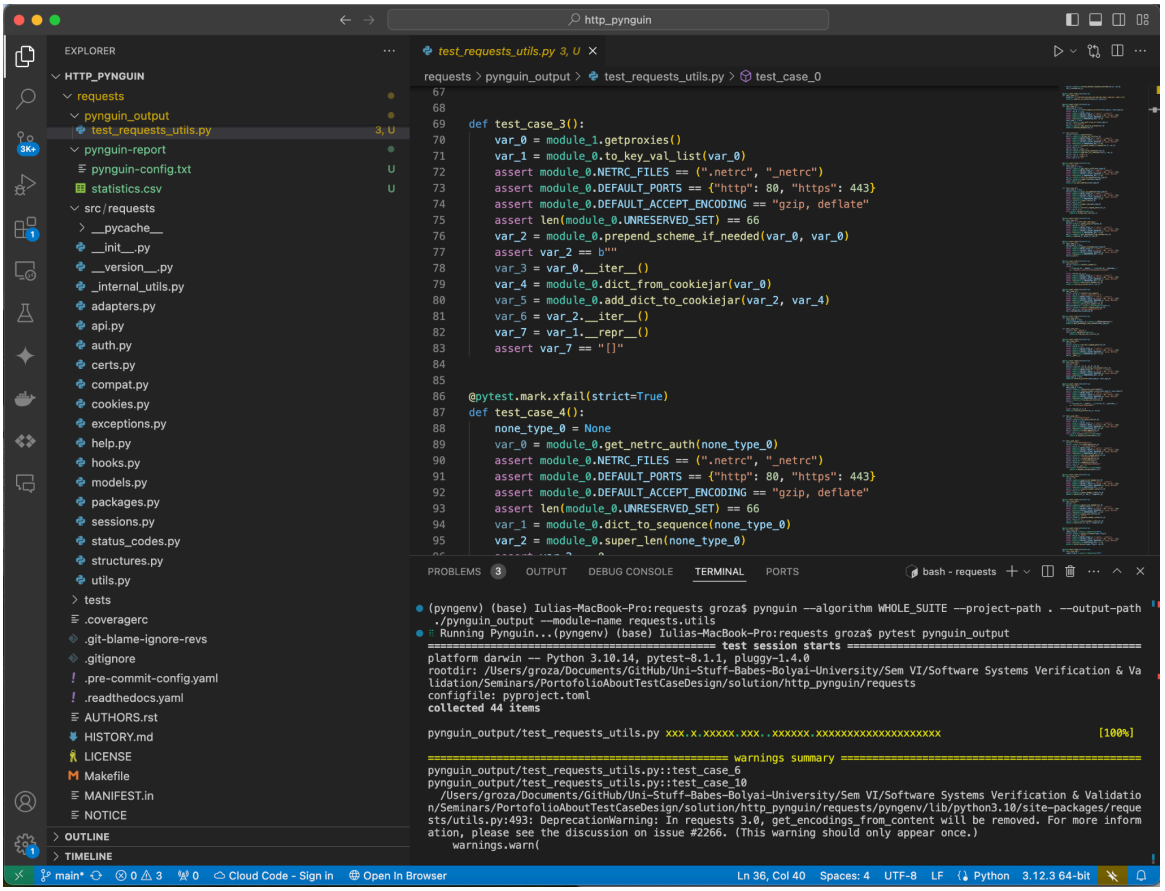
Usage

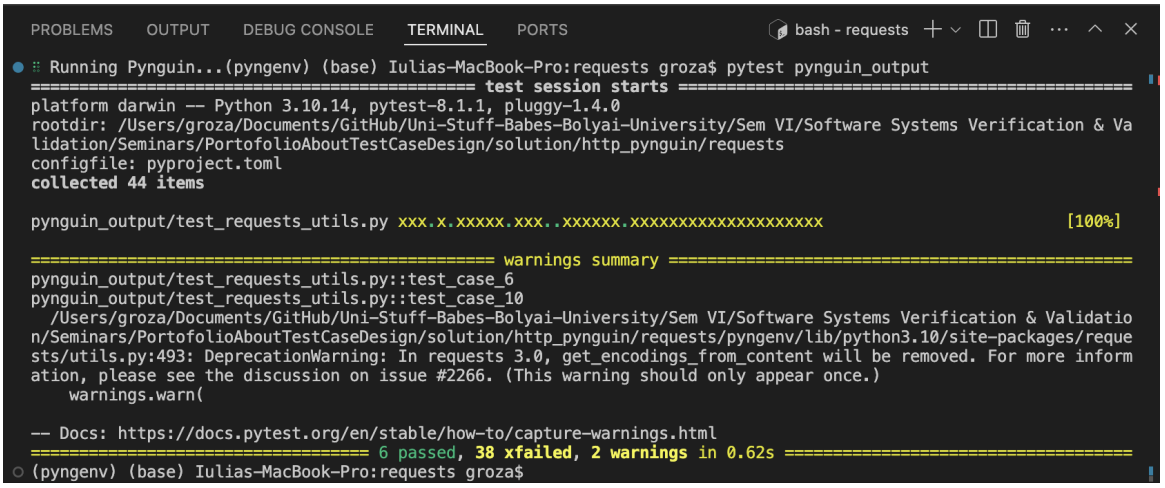
- We tested the tool on both a personal project, and the open source “*requests”* Python package. Overall, Pynguin performs its tasks with efficiency, automatically generating 36, respectively 43 tests. Moreover, it provided multiple generating algorithms that we were able to experiment with in creating our automated tests.
- Additionally, Penguin also generates a statistical report regarding coverage, that is essential in the testing process.
- Despite its ease to be utilised, Pynguin is an experimental project, therefore it is not very accurate in its generation. Multiple tests were completely wrong and required almost full refactoring for them being fixed. That would be a tedious process and it would ironically require the user to debug the tests.

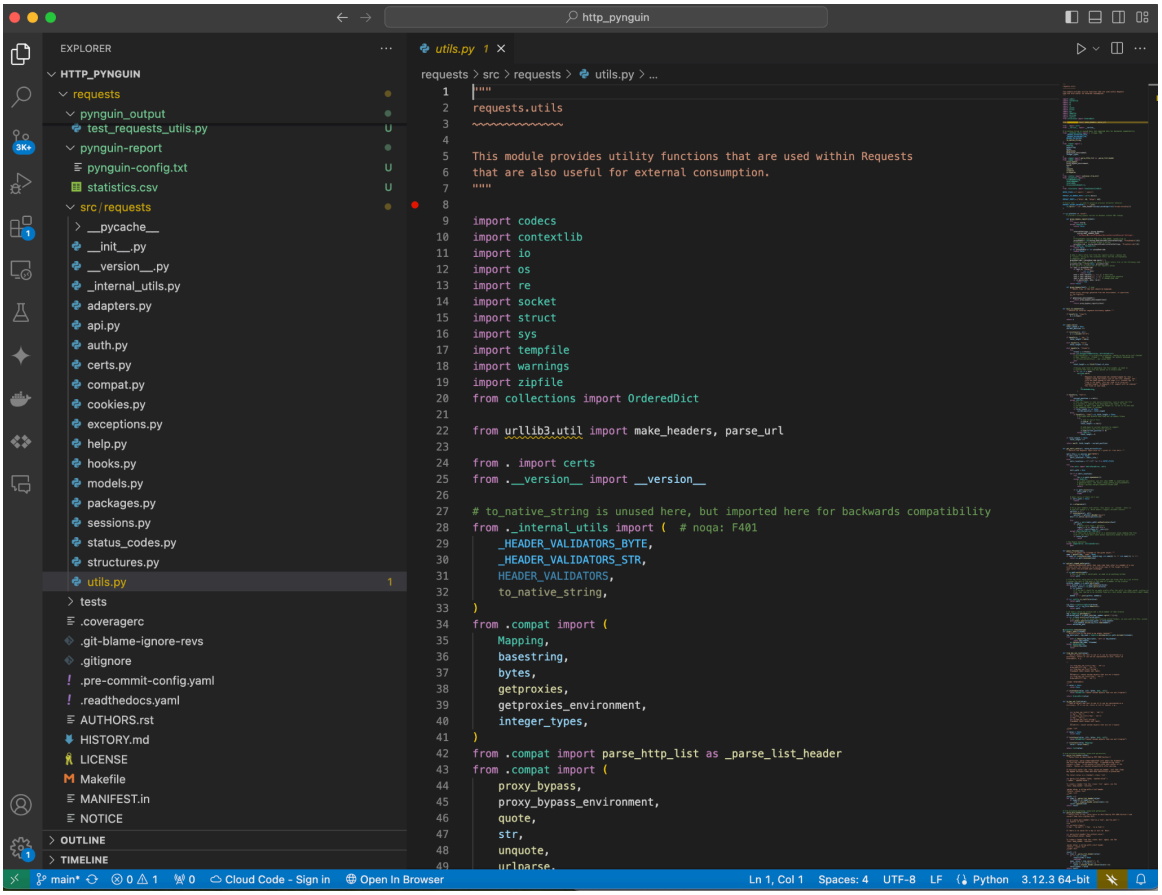
Benefits and Results

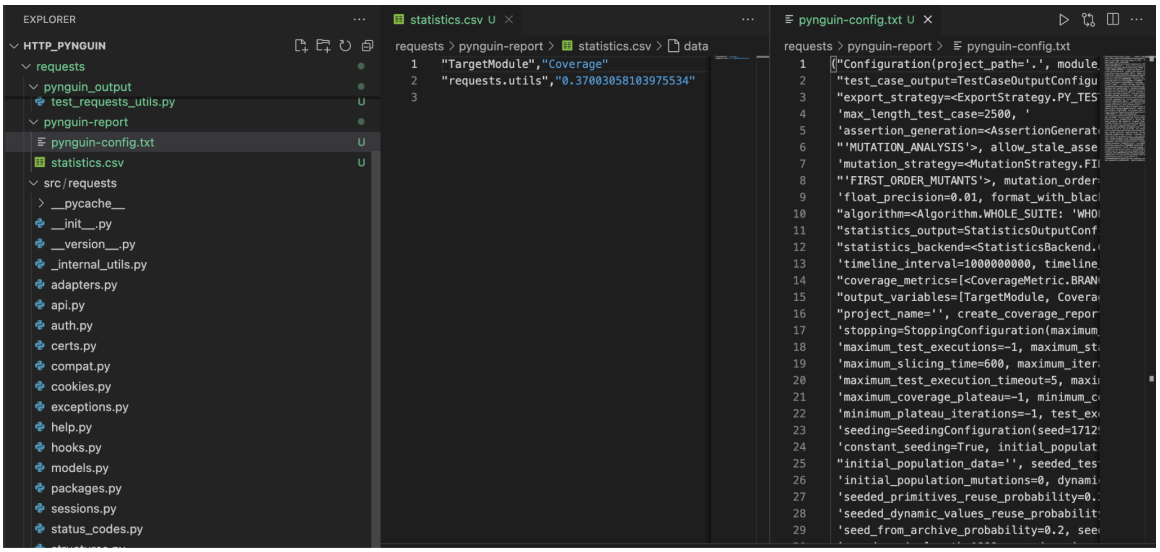
- **Code Coverage Enhancement:** Pynguin excels at creating tests that substantially improve code coverage, essential for thorough testing and quality assurance.
- **Ease of Integration:** It fits smoothly into existing Python projects, requiring no additional setup other than installing the package and running the tool from the command line.
- **Customizable Test Generation:** Pynguin offers various algorithms for test generation, providing flexibility in approaching test suite construction to match different project needs.

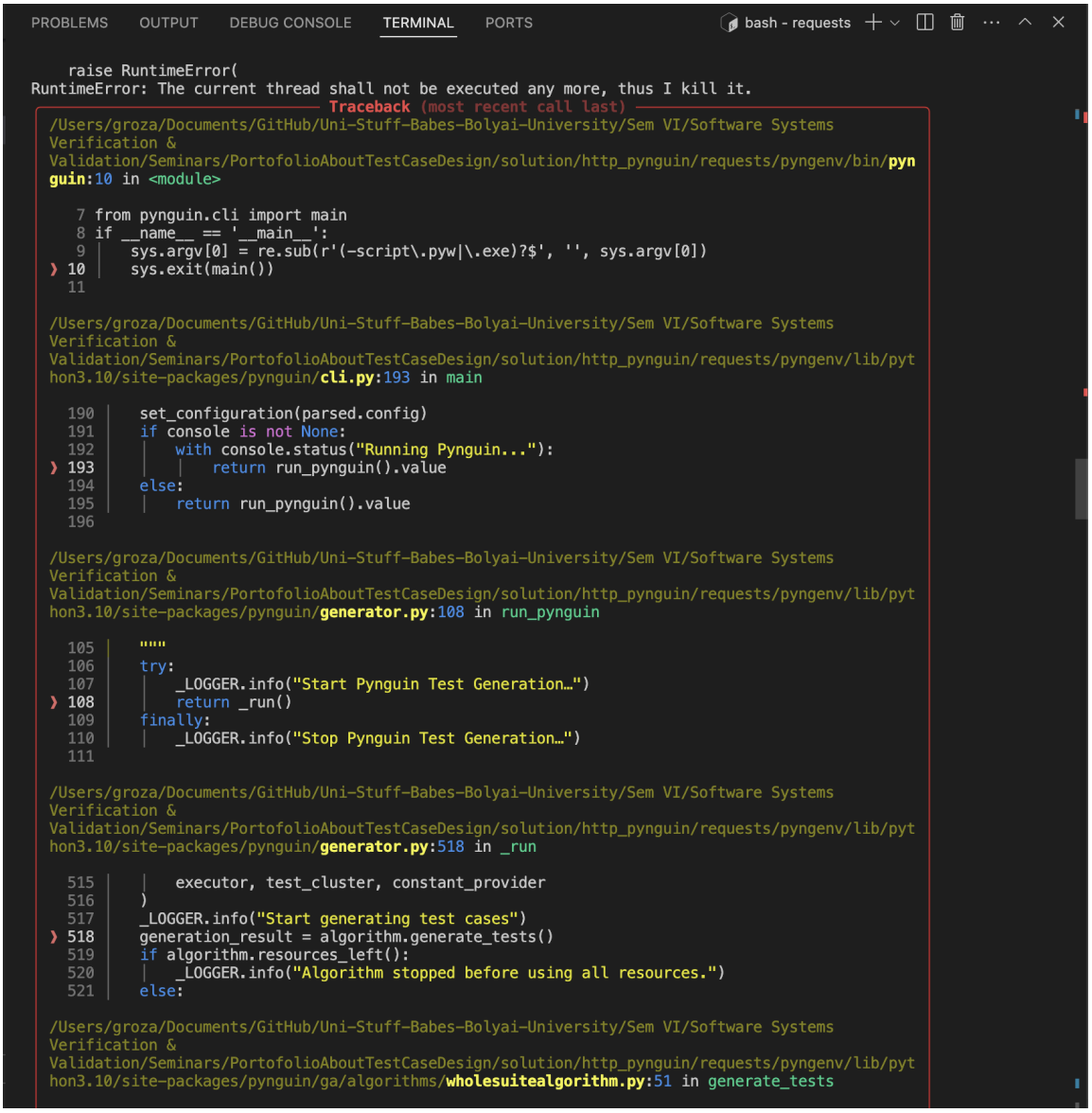
Screenshots






http_pynguin
EXPLORER
HTTP_PYNGUIN
requests
pynguin_output
test_requests_utils.py
pynguin-report
pynguin-config.txt
statistics.csv
src/requests
__pycache__
__init__.py
__version__.py
_internal_utils.py
adapters.py
api.py
auth.py
certs.py
compat.py
cookies.py
exceptions.py
help.py
hooks.py
models.py
packages.py
sessions.py
status_codes.py
structures.py
utils.py
tests
.coveragerc
.git-blame-ignore-revs
.gitignore
.pre-commit-config.yaml
.readthedocs.yaml
AUTHORS.rst
HISTORY.md
LICENSE
Makefile
MANIFEST.in
NOTICE
OUTLINE
TIMELINE
utils.py
requests > src > requests > utils.py > ...
"""
requests.utils
This module provides utility functions that are used within Requests
that are also useful for external consumption.
"""
import codecs
import contextlib
import io
import os
import re
import socket
import struct
import sys
import tempfile
import warnings
import zipfile
from collections import OrderedDict
from urllib3.util import make_headers, parse_url
from . import certs
from .__version__ import __version__
# to_native_string is unused here, but imported here for backwards compatibility
from ._internal_utils import (# noqa: F401
_HEADER_VALIDATORS_BYTE,
_HEADER_VALIDATORS_STR,
HEADER_VALIDATORS,
to_native_string,
)
from .compat import (
Mapping,
basestring,
bytes,
getproxies,
getproxies_environment,
integer_types,
)
from .compat import parse_http_list as _parse_list_header
from .compat import (
proxy_bypass,
proxy_bypass_environment,
quote,
str,
unquote,
urlparse,
main*
0 0 1
0
Cloud Code - Sign in
Open In Browser
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Python
3.12.3 64-bit


EXPLORER
HTTP_PYNGUIN
requests
pynguin_output
test_requests_utils.py
pynguin-report
pynguin-config.txt
statistics.csv
src/requests
__pycache__
__init__.py
__version__.py
_internal_utils.py
adapters.py
api.py
auth.py
certs.py
compat.py
cookies.py
exceptions.py
help.py
hooks.py
models.py
packages.py
sessions.py
status_codes.py
statistics.csv U
requests > pynguin-report > statistics.csv > data
"TargetModule","Coverage"
"requests.utils","0.37003058103975534"
pynguin-config.txt U
requests > pynguin-report > pynguin-config.txt
("Configuration(project_path='.', module
"test_case_output=TestCaseOutputConfigu
"export_strategy=<ExportStrategy.PY_TES
'max_length_test_case=2500, '
'assertion_generation=<AssertionGenerat
"'MUTATION_ANALYSIS'>, allow_stale_asse
'mutation_strategy=<MutationStrategy.FI
"'FIRST_ORDER_MUTANTS'>, mutation_order
'float_precision=0.01, format_with_blac
"algorithm=<Algorithm.WHOLE_SUITE: 'WHO
"statistics_output=StatisticsOutputConf
"statistics_backend=<StatisticsBackend.
'timeline_interval=1000000000, timeline_
"coverage_metrics=[<CoverageMetric.BRAN
"output_variables=[TargetModule, Covera
"project_name='', create_coverage_repor
'stopping=StoppingConfiguration(maximum_
'maximum_test_executions=-1, maximum_st
'maximum_slicing_time=600, maximum_iter
'maximum_test_execution_timeout=5, maxi
'maximum_coverage_plateau=-1, minimum_c
'minimum_plateau_iterations=-1, test_ex
'seeding=SeedingConfiguration(seed=1712
'constant_seeding=True, initial_populat
"initial_population_data='', seeded_tes
'initial_population_mutations=0, dynami
'seeded_primitives_reuse_probability=0.
'seeded_dynamic_values_reuse_probabilit
'seed_from_archive_probability=0.2, see


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash - requests
raise RuntimeError(
RuntimeError: The current thread shall not be executed any more, thus I kill it.
Traceback (most recent call last)
/Users/groza/Documents/GitHub/Uni-Stuff-Babes-Bolyai-University/Sem VI/Software Systems Verification & Validation/Seminars/PortofolioAboutTestCaseDesign/solution/http_pynguin/requests/pyngenv/bin/pynguin:10 in <module>
7 from pynguin.cli import main
8 if __name__ == '__main__':
9 sys.argv[0] = re.sub(r'(-script\.pyw|\.exe)?$', '', sys.argv[0])
10 sys.exit(main())
11
/Users/groza/Documents/GitHub/Uni-Stuff-Babes-Bolyai-University/Sem VI/Software Systems Verification & Validation/Seminars/PortofolioAboutTestCaseDesign/solution/http_pynguin/requests/pyngenv/lib/python3.10/site-packages/pynguin/cli.py:193 in main
190 set_configuration(parsed.config)
191 if console is not None:
192 with console.status("Running Pynguin..."):
193 return run_pynguin().value
194 else:
195 return run_pynguin().value
196
/Users/groza/Documents/GitHub/Uni-Stuff-Babes-Bolyai-University/Sem VI/Software Systems Verification & Validation/Seminars/PortofolioAboutTestCaseDesign/solution/http_pynguin/requests/pyngenv/lib/python3.10/site-packages/pynguin/generator.py:108 in run_pynguin
105 """
106 try:
107 _LOGGER.info("Start Pynguin Test Generation…")
108 return _run()
109 finally:
110 _LOGGER.info("Stop Pynguin Test Generation…")
111
/Users/groza/Documents/GitHub/Uni-Stuff-Babes-Bolyai-University/Sem VI/Software Systems Verification & Validation/Seminars/PortofolioAboutTestCaseDesign/solution/http_pynguin/requests/pyngenv/lib/python3.10/site-packages/pynguin/generator.py:518 in _run
515 executor, test_cluster, constant_provider
516)
517 _LOGGER.info("Start generating test cases")
518 generation_result = algorithm.generate_tests()
519 if algorithm.resources_left():
520 _LOGGER.info("Algorithm stopped before using all resources.")
521 else:
/Users/groza/Documents/GitHub/Uni-Stuff-Babes-Bolyai-University/Sem VI/Software Systems Verification & Validation/Seminars/PortofolioAboutTestCaseDesign/solution/http_pynguin/requests/pyngenv/lib/python3.10/site-packages/pynguin/ga/algorithms/wholesuitealgorithm.py:51 in generate_tests