

Moduł 3 – Software Testing

Zajęcia 1. Test Driven Development (TDD) oraz Test Doubles – Python

Dane prowadzącego:

Adam Krzysiek

adam_krzysiek@epam.com

Czas trwania zajęć: 1,5h.

Łączna liczba punktów do zdobycia na zajęciach: 4 pkt za sprawozdanie + 1 pkt extra za aktywność.

Warunki zaliczenia zajęć: warunkiem zaliczenia zajęć jest obecność na zajęciach, przesłanie kodu końcowego zadania oraz opracowanie sprawozdania.

Lista materiałów przygotowujących do zajęć:

- <https://sjsi.org/download/6351/>
- <https://www.samouczekprogramisty.pl/test-driven-development-na-przykladzie/>
- <https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da>
- <https://martinfowler.com/bliki/TestDouble.html>

Wymagania software'owe:

1. Python 3.8
2. Pip dla Python 3.8

Zadania do wykonania:

Zadanie 1

Zaimplementuj aplikację do tworzenia listy maili pracowników firmy na podstawie ich imion i nazwisk w duchu TDD. Wszystkie wymagania funkcjonalne muszą zostać pokryte przez odpowiednie przypadki testowe uwzględniając określone ograniczenia.

Wymagania funkcjonalne:

1. Aplikacja generuje adresy mailowe pracowników – imie_nazwisko@example.com.
2. Aplikacja zaczytuje dane (imiona i nazwiska) z pliku tekstowego (format CSV).

Wymagania niefunkcjonalne:

- Pokrycie kodu testami – 80%.

Ograniczenia:

- Brak duplikatów (imię i nazwisko) – w przypadku duplikatów dodaj sufix _2.
- Brak cyfr w pliku wejściowym.
- Plik wejściowy nie może być pusty.
- Imiona i nazwiska w pliku wejściowym oddzielone są przecinkiem.

Scenariusze testowe

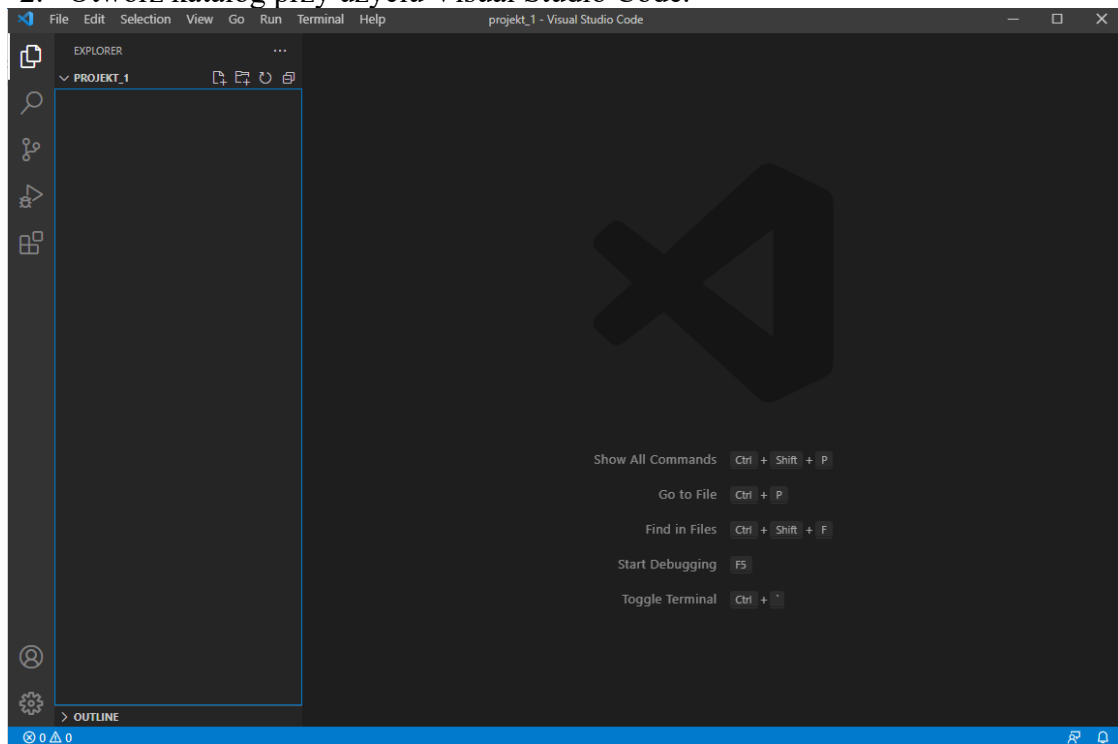
ID	Nazwa	Opis	Czynności przygotowawcze	Czynności końcowe
1	Generowanie adresów mailowych pracowników	Sprawdzenie poprawności działania funkcjonalności generowania adresów pracowników na podstawie imienia i nazwiska.	1.zainicjalizaowanie aplikacji	brak
2	Importowanie danych pracowników z pliku wejściowego.	Sprawdzenie poprawności importowania danych pracowników z pliku wejściowego.	1.zainicjalizaowanie aplikacji	brak

Przypadki testowe

ID	ID Scenariusza	Nazwa	Warunki wstępne	Kroki	Oczekiwany rezultat
1	2	Importowanie danych pracowników z pliku wejściowego.	1. dany jest plik wejściowy z danymi pracowników – imię i nazwisko w formacie CSV.	1. Zainicjalizuj pustą listę z wynikami. 2. Otwórz plik. 3. Dla każdego rekordu w pliku zapisz do listy pierwszy element jako imię a drugi jako nazwisko. 4. Zamknij plik.	Dla każdego rekordu z pliku wejściowego istnieje reprezentacja pracownika jako imię i nazwisko w liście.
2	1	Generowanie adresów mailowych pracowników	1. dana jest lista obiektów z imionami i nazwiskami pracowników	1. Zainicjalizuj pustą listę z wynikami. 2. Dla każdego elementu z listy danych wejściowych jako imię i nazwisko wygeneruj adres email.	Dla każdej pary imię i nazwisko wygenerowany jest adres mail.
3	2	Importowanie danych pracowników z pliku wejściowego – pusty plik.	1. dany jest pusty plik wejściowy.	1. Zainicjalizuj pustą listę z wynikami. 2. Otwórz plik. 3. Dla każdego rekordu w pliku zapisz do listy pierwszy element jako imię a drugi jako nazwisko. 4. Zamknij plik.	Zgłoś wyjątek informujący o braku rekordów.

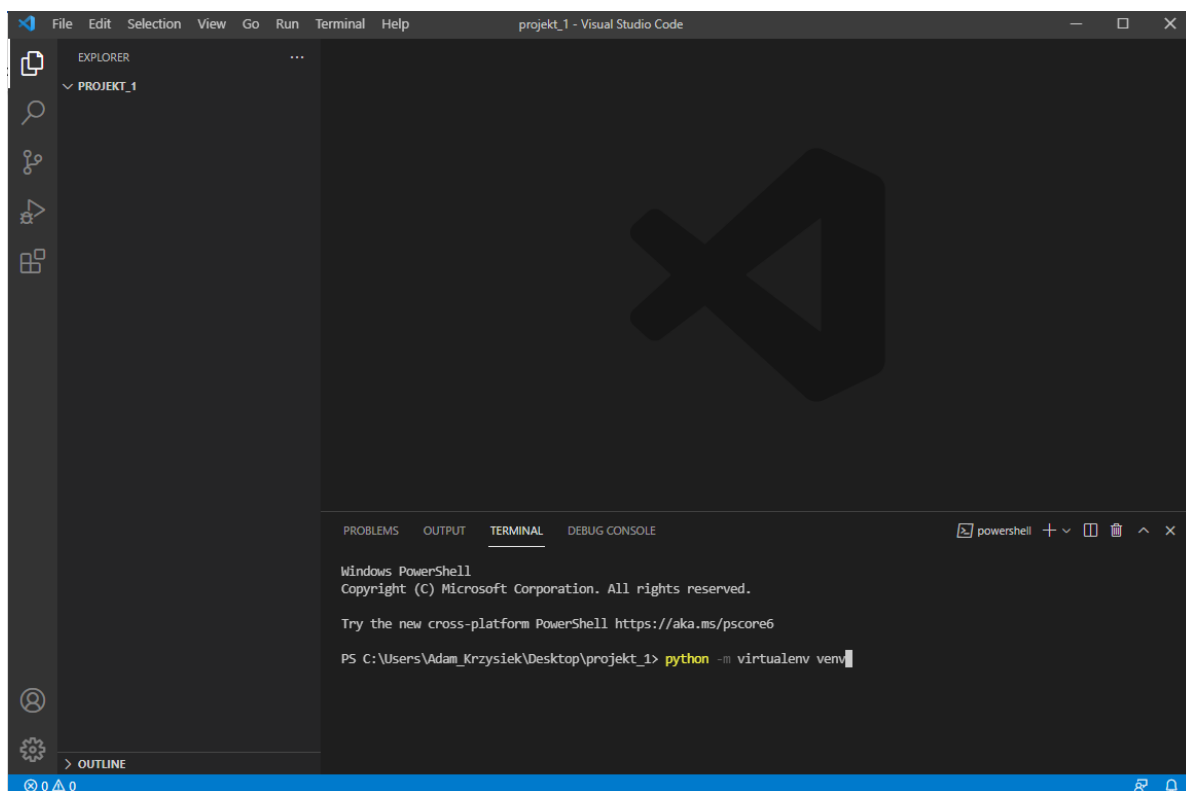
Przebieg zajęć:

1. Utwórz katalog na projekt.
2. Otwórz katalog przy użyciu Visual Studio Code.



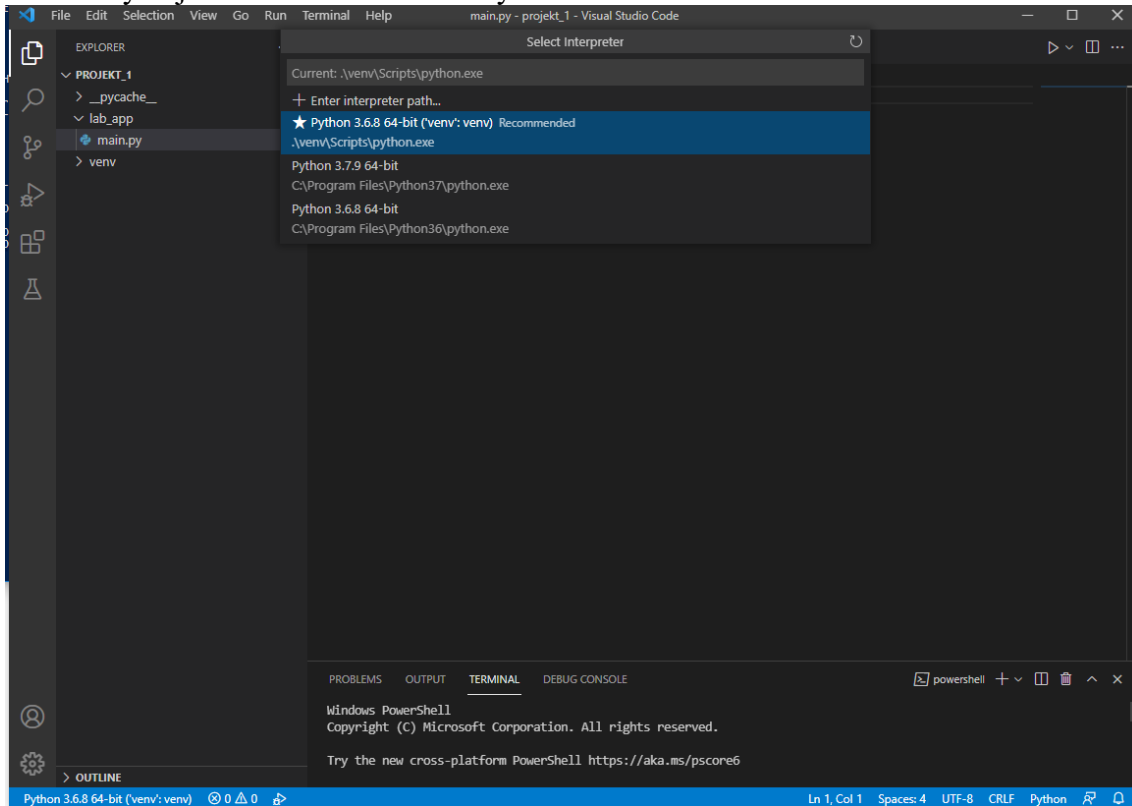
3. Utwórz lokalne, wirtualne środowisko Python 3.8 w katalogu na projekt w terminalu VS Code przy użyciu komendy

```
python -m venv venv
```

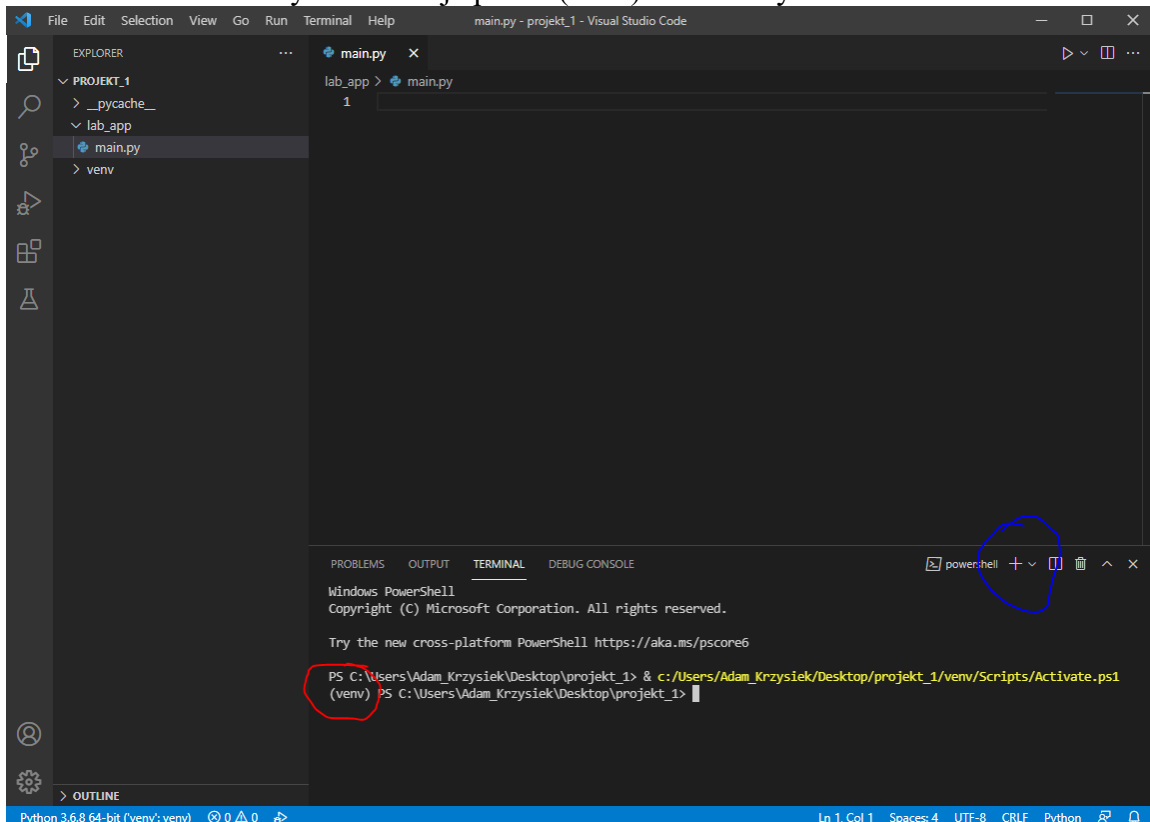


4. Stwórz katalog lab_app a następnie plik main.py w katalogu lab_app.

5. Aktywuj wirtualne środowisko Python w VS Code.



6. Otwórz nowy terminal w VS Code, tak aby w terminalu zainicjować wirtualne środowisko Python. Ważne jest, aby kolejne kroki wykonywane były w wirtualnym środowisku o czym informuje prefix (venv) - czerwonym kółku.

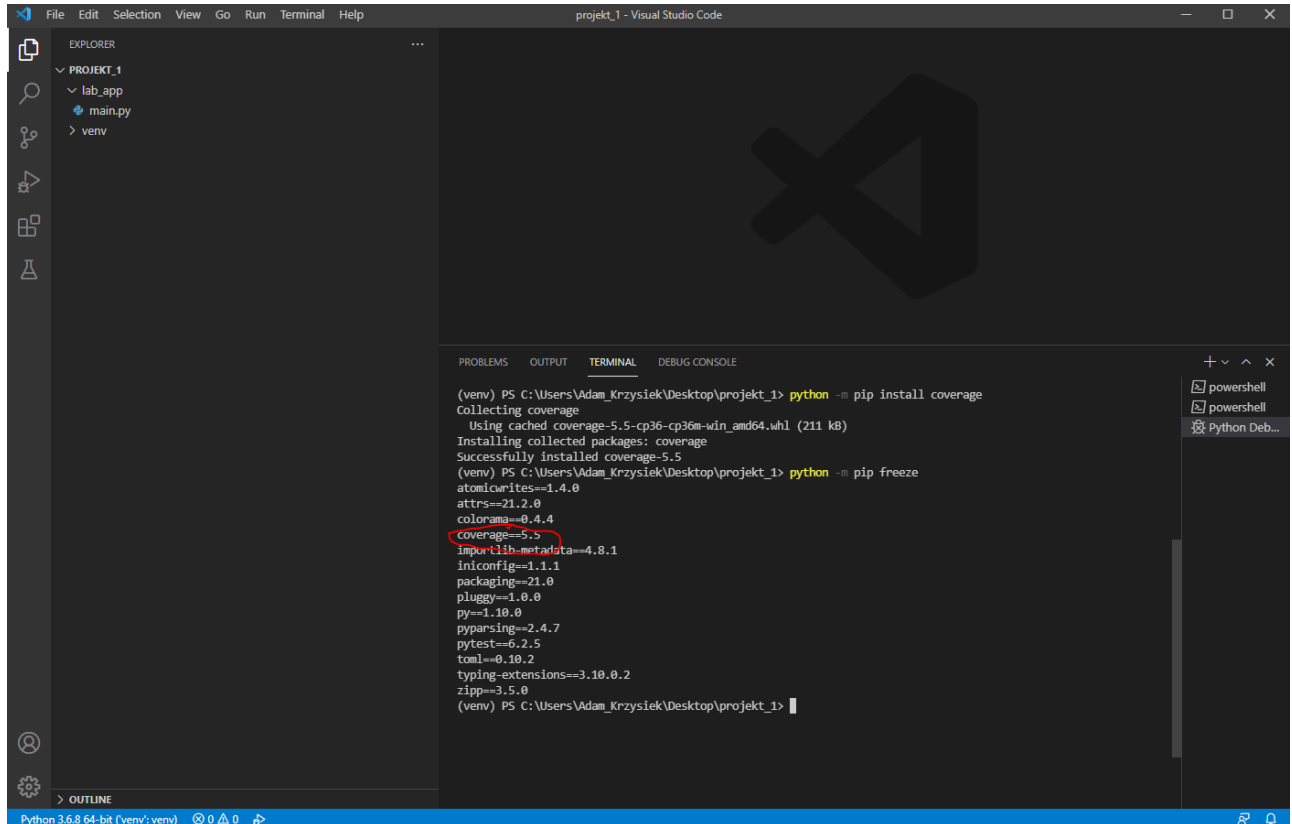


7. Zainstaluj bibliotekę Coverage w wirtualnym środowisku w terminalu VS Code przy użyciu komendy:

```
python -m pip install coverage
```

8. Zweryfikuj poprawność instalacji Coverage w wirtualnym środowisku w terminalu VS Code przy użyciu komendy:

```
python -m pip freeze
```



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal output shows the successful installation of coverage and the resulting list of installed packages. The package coverage==5.5 is highlighted with a red circle.

```
(venv) PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m pip install coverage
Collecting coverage
  Using cached coverage-5.5-cp36-cp36m-win_amd64.whl (211 kB)
Installing collected packages: coverage
Successfully installed coverage-5.5
(venv) PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m pip freeze
atomicwrites==1.4.0
attrs==21.2.0
colorama==0.4.4
coverage==5.5
importlib-metadata==4.8.1
iniconfig==1.1.1
packaging==21.0
pluggy==1.0.0
py==1.10.0
pyparsing==2.4.7
pytest==6.2.5
toml==0.10.2
typing-extensions==3.10.0.2
zipp==3.5.0
(venv) PS C:\Users\Adam_Krzysiek\Desktop\projekt_1>
```

9. Uzupełnij plik main.py o następujący kod.

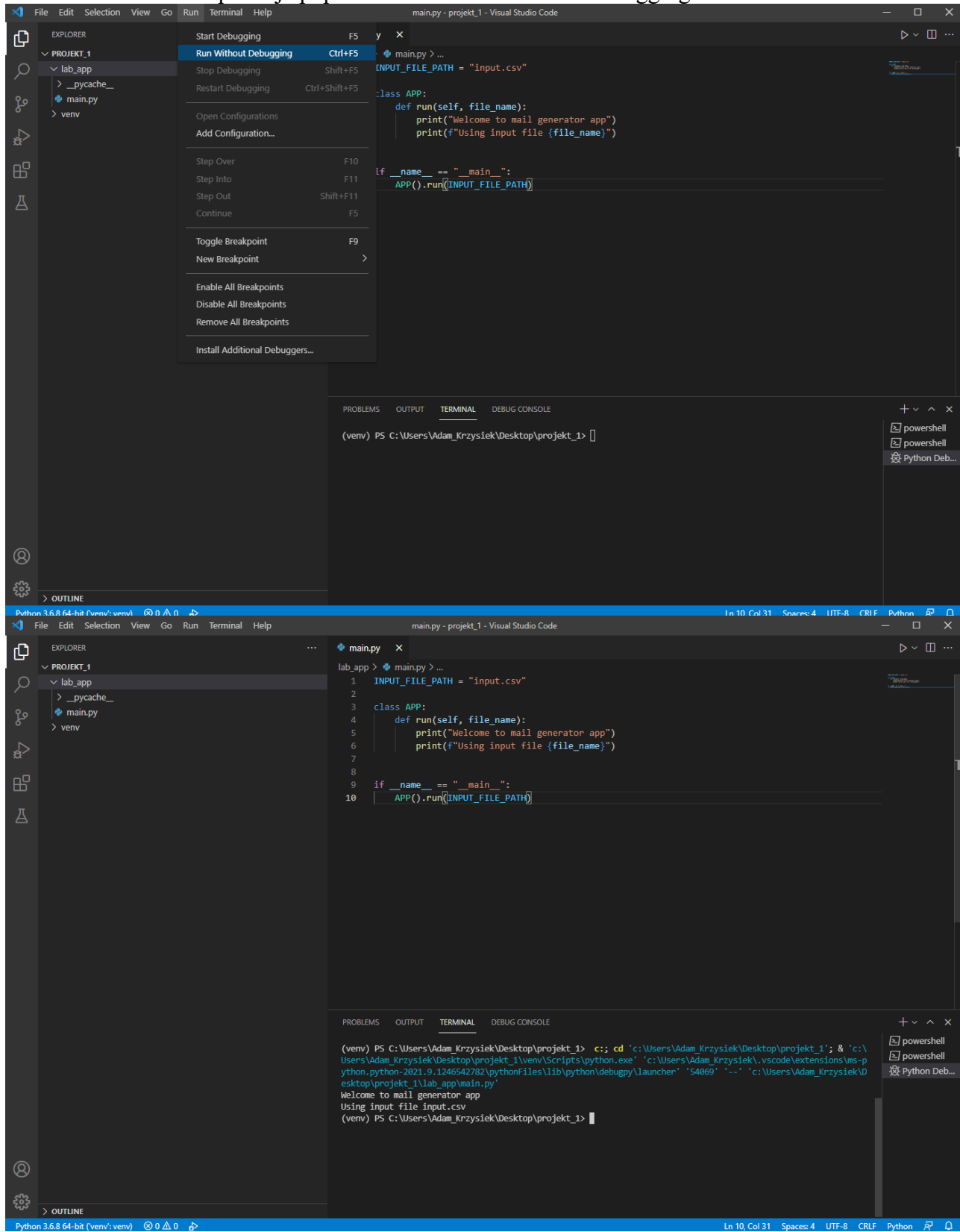
```
# lab_app/main.py
INPUT_FILE_PATH = "input.csv"

class APP:
    def run(self, file_name):
        print("Welcome to mail generator app")
        print(f"Using input file {file_name}")

if __name__ == "__main__":
    APP().run(INPUT_FILE_PATH)
```

10. Dodaj konfigurację uruchomieniową Run-> Add configuration -> module -> lab_app.main .

11. Uruchom aplikacje poprzez Run -> Run Without Debugging.



12. Stwórz klasę testową BasicTestCase. Ścieżka do pliku z testem -> tests/test_basic.py.

```
import unittest

from lab_app.main import APP

class BasicTestCase(unittest.TestCase):
    """This class represents the Basic test case"""

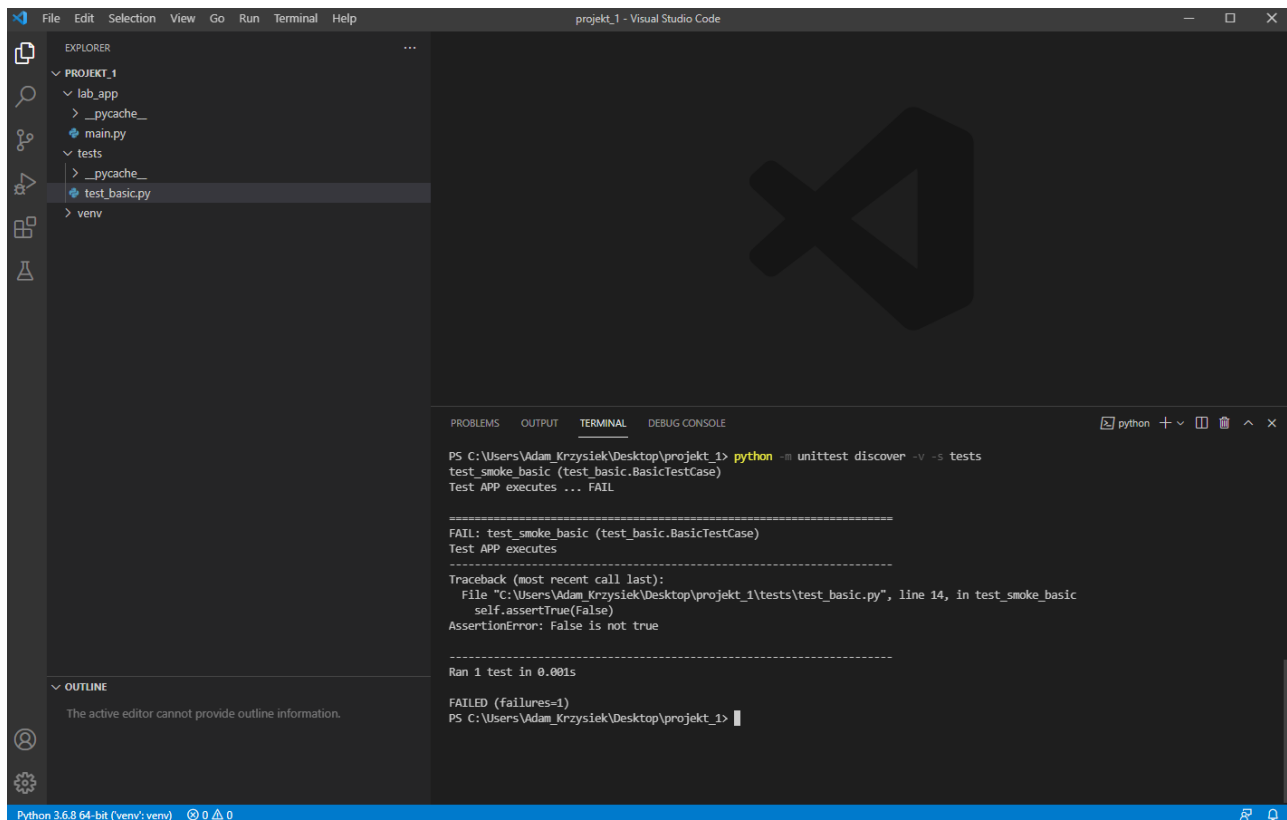
    def setUp(self):
        """Define test variables and initialize app."""
        self.app = APP()

    def test_smoke_basic(self):
        """ Test APP executes"""
        self.assertTrue(False)

    def tearDown(self):
        """Tear down all initialized variables and close app."""
        pass
```

13. Uruchom test manualnie w terminalu VS Code poprzez wywołanie komendy:

```
python -m unittest discover -v -s tests
```



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the project structure: `PROJEKT_1` containing `lab_app`, `__pycache__`, `main.py`, `tests`, and `venv`. The `tests` directory is expanded, showing `__pycache__` and `test_basic.py`. The main editor area is dark and contains a large, faint Visual Studio Code logo. At the bottom, the TERMINAL panel is active, showing the output of the command `python -m unittest discover -v -s tests`. The output indicates that the test `test_smoke_basic` failed with an `AssertionError: False is not true`. The terminal text is as follows:

```
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m unittest discover -v -s tests
test_smoke_basic (test_basic.BasicTestCase)
Test APP executes ... FAIL

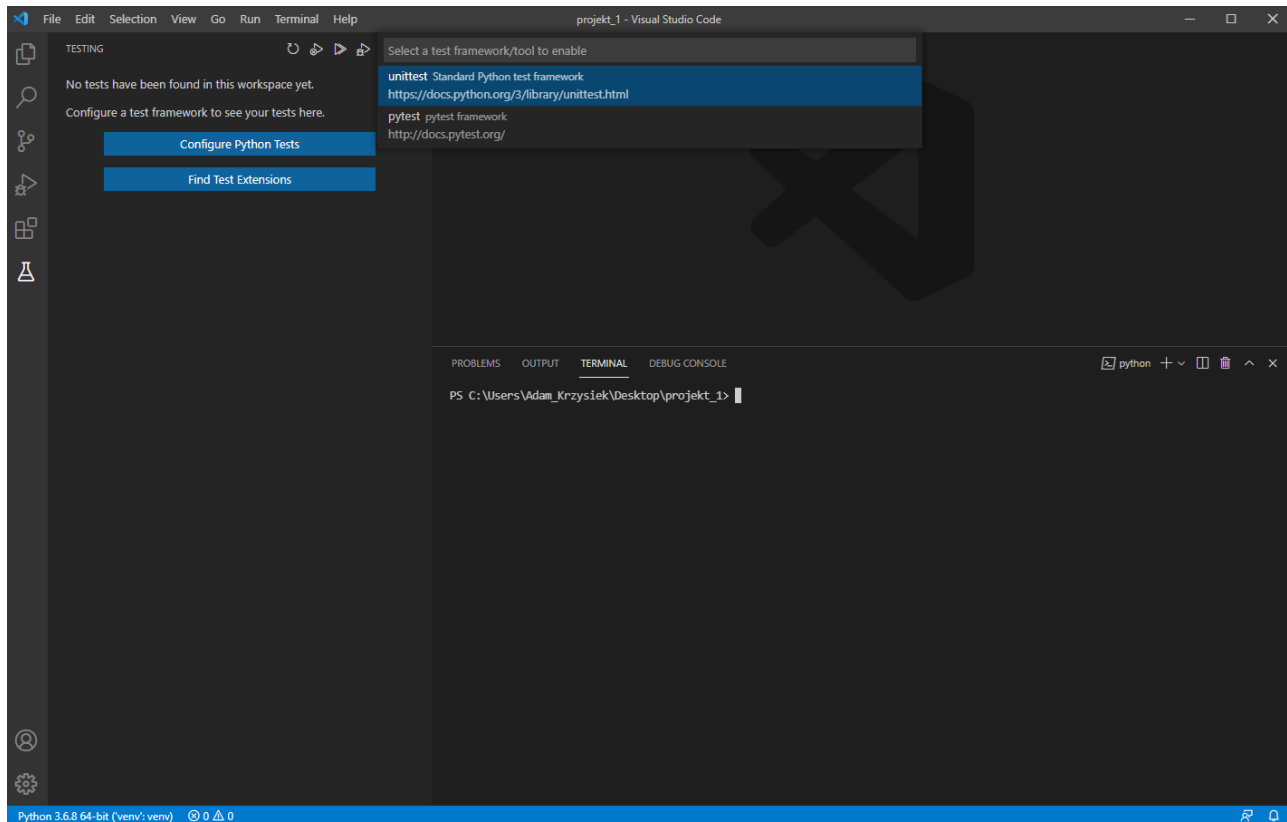
-----
FAIL: test_smoke_basic (test_basic.BasicTestCase)
Test APP executes
-----
Traceback (most recent call last):
  File "C:\Users\Adam_Krzysiek\Desktop\projekt_1\tests\test_basic.py", line 14, in test_smoke_basic
    self.assertTrue(False)
AssertionError: False is not true
-----
Ran 1 test in 0.001s

FAILED (failures=1)
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1>
```

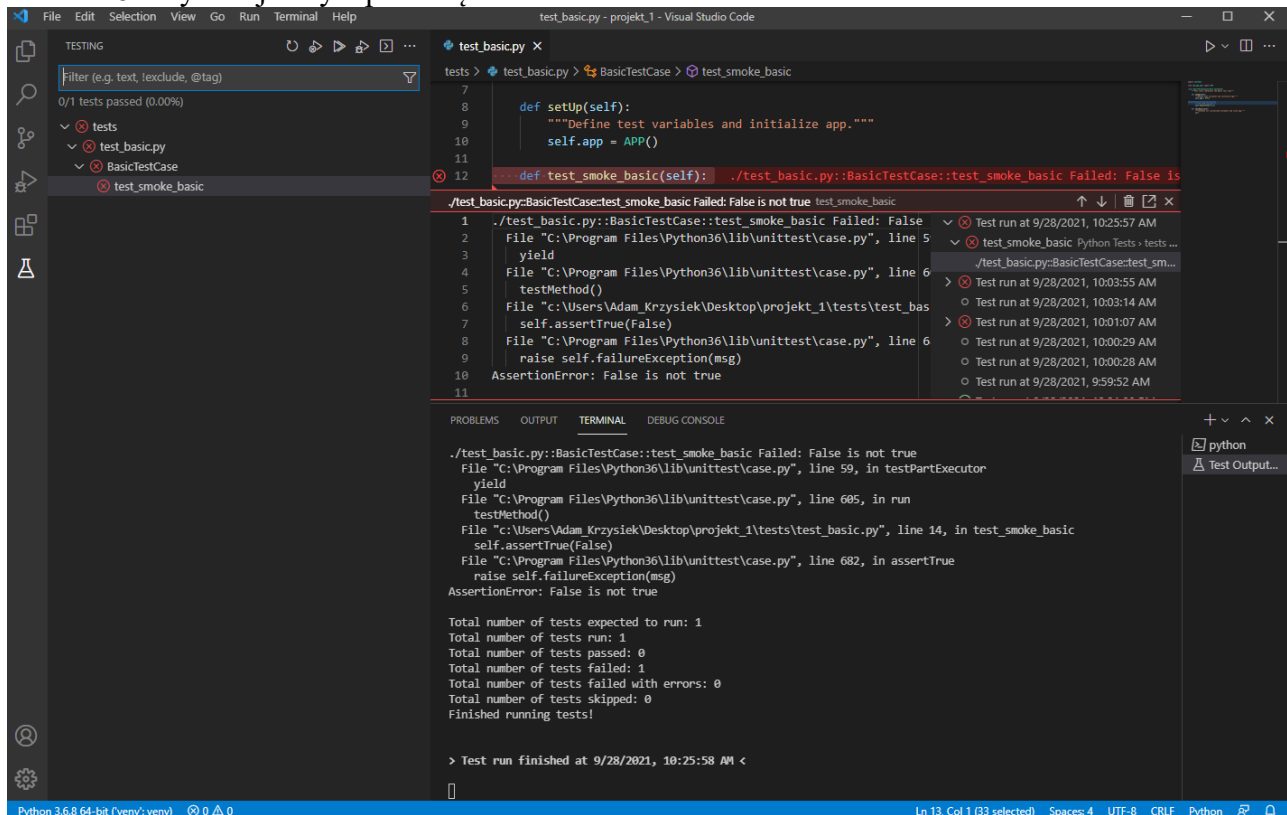
The status bar at the bottom of the window indicates "Python 3.6.8 64-bit (venv: venv)" and shows 0 errors, 0 warnings, and 0 info messages.

14. Stwórz konfigurację testową w VS Code.

Configure Python Tests -> unittest -> wybierz tests -> wybierz test_*.py



15. Wywołaj testy z pomocą VS Code.



16. Określ pokrycie testami kodu aplikacji w wirtualnym środowisku w terminalu VS Code przy użyciu komendy:

Uruchomienie testów z jednoczesnym pomiarem pokrycia kodu testami.

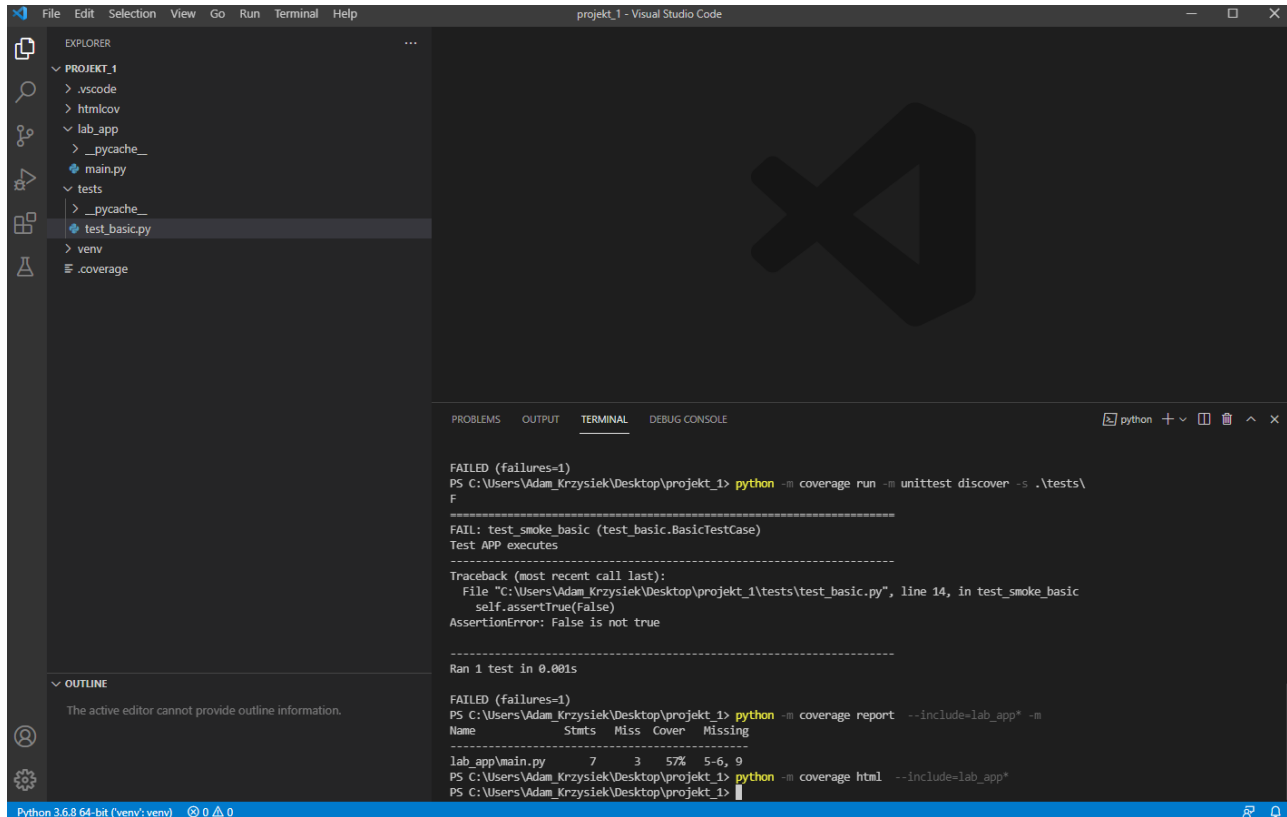
```
python -m coverage run -m unittest discover -s tests
```

Wygenerowanie raportu pokrycia kodu testami i wyświetlenie w terminalu.

```
python -m coverage report --include="lab_app*" -m
```

Wygenerowanie raportu html pokrycia kodu testami.

```
python -m coverage html --include="lab_app*" -m
```



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the output of running tests and generating coverage reports. The first command executed is `python -m coverage run -m unittest discover -s tests`, which results in a failure for `test_smoke_basic` due to an `AssertionError: False is not true`. The second command is `python -m coverage report --include=lab_app* -m`, which shows the coverage for `lab_app/main.py` as 57% (4 run, 3 missing, 0 excluded). The third command is `python -m coverage html --include=lab_app* -m`, which generates the HTML coverage report.

```
FAILED (failures=1)
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m coverage run -m unittest discover -s tests\
F
=====
FAIL: test_smoke_basic (test_basic.BasicTestCase)
Test APP executes
-----
Traceback (most recent call last):
  File "C:\Users\Adam_Krzysiek\Desktop\projekt_1\tests\test_basic.py", line 14, in test_smoke_basic
    self.assertTrue(False)
AssertionError: False is not true
-----
Ran 1 test in 0.001s

FAILED (failures=1)
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m coverage report --include=lab_app* -m
Name      Stats  Miss  Cover  Missing
-----
lab_app/main.py  7      3    57%  5-6, 9
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m coverage html --include=lab_app* -m
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1>
```

Coverage for **lab_app\main.py** : 57%

7 statements 4 run 3 missing 0 excluded

```
1 | INPUT_FILE_PATH = "input.csv"
2 |
3 | class APP:
4 |     def run(self, file_name):
5 |         print("Welcome to mail generator app")
6 |         print(f"Using input file {file_name}")
7 |
8 |
9 | if __name__ == "__main__":
10 |     APP().run(INPUT_FILE_PATH)
```

17. Popraw pierwszy test `test_1_basic` w klasie `BasicTestCase`, który będzie werifikował czy aplikacja wywołuje poprawnie metodę `run`. Ścieżka do pliku z testem -> `tests/test_basic.py`.

```
# tests/test_basic.py
import unittest

from lab_app.main import APP

class BasicTestCase(unittest.TestCase):
    """This class represents the Basic test case"""

    def setUp(self):
        """Define test variables and initialize app."""
        self.app = APP()

    def test_smoke_basic(self):
        """Test APP has 'run' attribute"""
        self.app.run("some_file")

    def tearDown(self):
        """Tear down all initialized variables and close app."""
        pass
```

18. Określ pokrycie testami kodu aplikacji w wirtualnym środowisku w terminalu VS Code przy użyciu komendy:

Uruchomienie testów z jednoczesnym pomiarem pokrycia kodu testami.

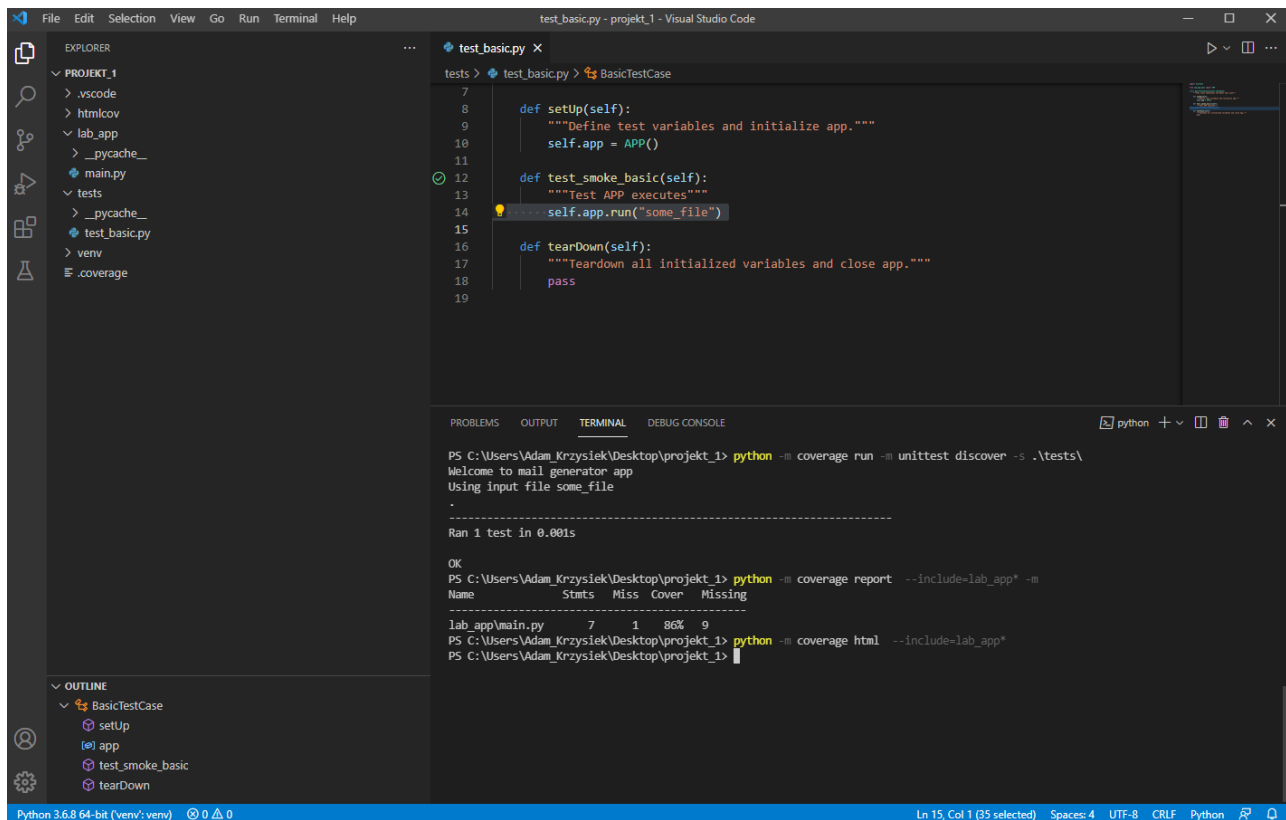
```
python -m coverage run -m unittest discover -s tests
```

Wygenerowanie raportu pokrycia kodu testami i wyświetlenie w terminalu.

```
python -m coverage report --include="lab_app*" -m
```

Wygenerowanie raportu html pokrycia kodu testami.

```
python -m coverage html --include="lab_app*" -m
```



Coverage for **lab_app\main.py** : 86%

7 statements

6 run

1 missing

0 excluded

```

1 | INPUT_FILE_PATH = "input.csv"
2
3 | class APP:
4 |     def run(self, file_name):
5 |         print("Welcome to mail generator app")
6 |         print(f"Using input file {file_name}")
7
8 | if __name__ == "__main__":
9 |     APP().run(INPUT_FILE_PATH)

```

19. Implementacja testów przypadku testowego 1.

- Utwórz plik testowy z danymi pracowników – input_test.csv

```

# input_test.csv
John,Smith
John,Doe

```

- Utwórz plik data_loader.py z szablonem klasy DataLoader.

```

# data_loader.py
class DataLoader:
    def read_data(self, file_name):
        pass

```

- Utwórz plik tests/test_data_loader.py z klasą testową DataLoaderTestCase i testem werifikującym poprawność wczytywania danych prakowników z pliku csv.

```
# tests/test_data_loader.py

import unittest

from lab_app.data_loader import DataLoader

class DataLoaderTestCase(unittest.TestCase):
    """This class represents the DataLoader test cases"""

    def setUp(self):
        """Define test variables and initialize data loader."""
        self.data_loader = DataLoader()

    def test_data_loader(self):
        """Test data loader loads data from file"""
        file_name = './input_test.csv'

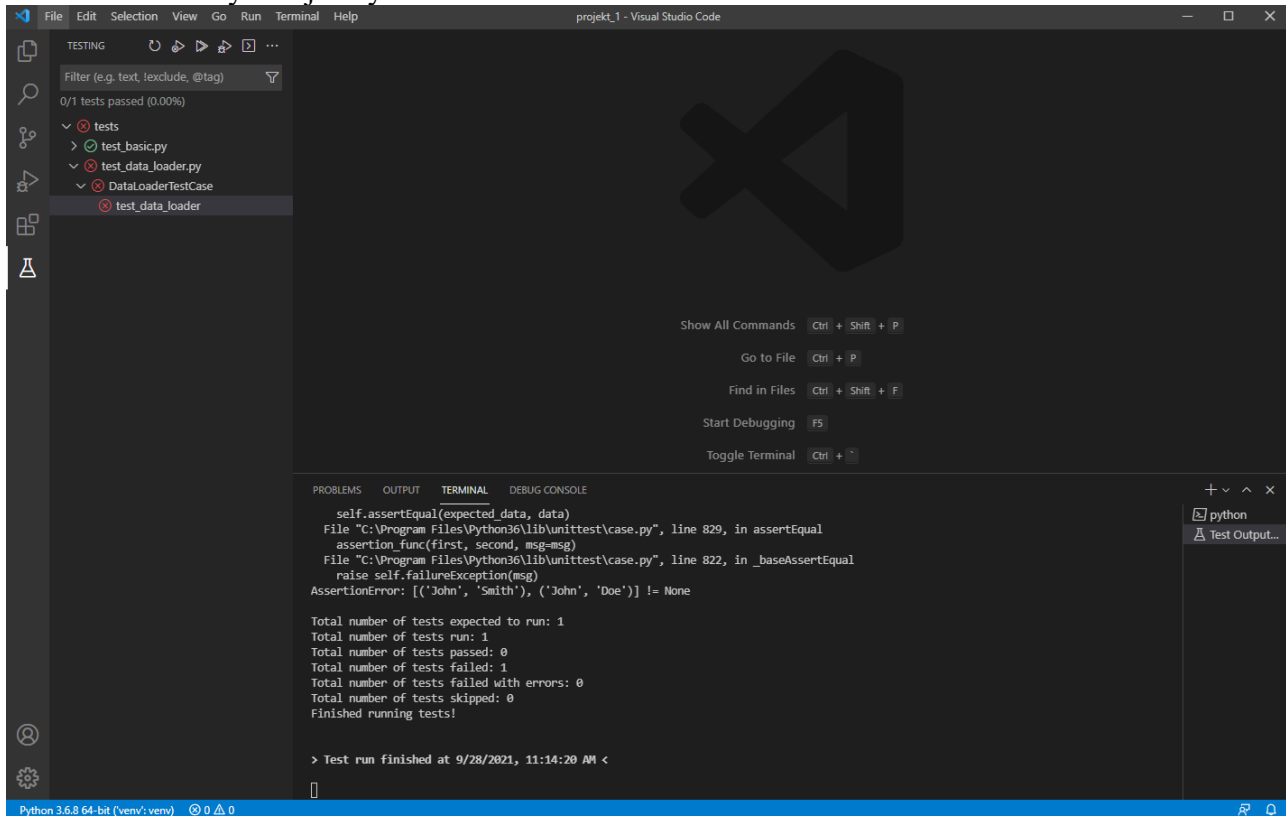
        expected_data=[
            ("John", "Smith"),
            ("John", "Doe")
        ]

        data = self.data_loader.read_data(file_name)

        self.assertEqual(expected_data, data)

    def tearDown(self):
        """Teardown all initialized variables and close data loader."""
        pass
```

- Wywołaj testy



- Zaimplementuj klasę DataLoader.

```
# data_loader.py

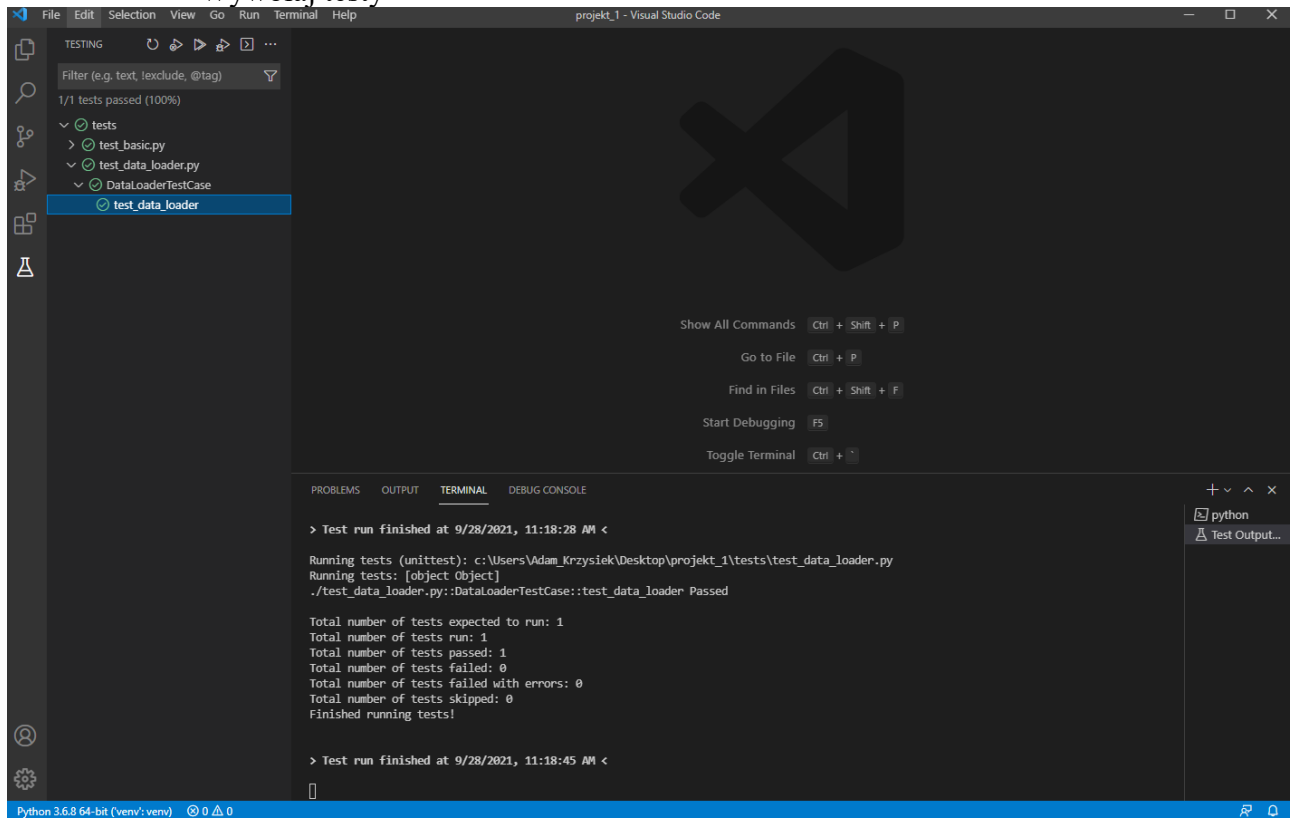
class DataLoader:

    def read_data(self, file_name):
        data = []
        try:
            file_handler = open(file_name, 'r')

            for line in file_handler.readlines():
                d = line.strip().split(',')
                data_tuple = name, surname = (d[0], d[1])
                data.append(data_tuple)
        finally:
            if file_handler:
                file_handler.close()

        return data
```

• Wywołaj testy



- Ulepsz implementację

```
# data_loader.py
class DataLoader:

    def __init__(self):
        self.data = []
        self.file_handler = None

    def open_file(self):
        self.file_handler = open(self.file_name, 'r')
        return self.file_handler

    def close_file(self):
        if self.file_handler:
            self.file_handler.close()

    def read_data(self, file_name):
        try:
            self.file_name = file_name
            self.file_handler = self.open_file()

            for line_no, line in enumerate(self.file_handler.readlines()):
                d = line.strip().split(',')
                data_tuple = name, surname = (d[0], d[1])
                self.data.append(data_tuple)
        finally:
            self.close_file()

        return self.data
```


- Testy dalej przechodzą

The screenshot shows the Visual Studio Code interface with a project named 'PROJEKT_1'. The Explorer sidebar on the left shows the file structure, including a 'tests' directory containing 'test_get_groups.py'. The main editor displays the code for 'test_get_groups.py', which defines a 'GetGroupsTestCase' class inheriting from 'unittest.TestCase'. The class includes a 'setUp' method to initialize the application and a 'test_get_group_basic' method to test a GET request. The bottom panel shows the 'TERMINAL' output, indicating that the test run was successful with 1 test passed and 0 failures. The status bar at the bottom indicates the Python 3.6.8 64-bit (venv) environment is active.

```
4
5
6 class GetGroupsTestCase(unittest.TestCase):
7     """This class represents the Basic test case"""
8
9     def setUp(self):
10         """Define test variables and initialize app."""
11         self.app = create_app()
12         self.client = self.app.test_client
13
14
15     def test_get_group_basic(self):
16         """Test API works (GET request)"""
17         res = self.client().get('/group')
18         self.assertEqual(res.status_code, 200)
19         self.assertEqual(type(res.data) == list)
20
21     def tearDown(self):
22         """Tear down all initialized variables and close app."""
23         pass
24
```

Terminal Output:

```
Total number of tests expected to run: 1
Total number of tests run: 1
Total number of tests passed: 1
Total number of tests failed: 0
Total number of tests failed with errors: 0
Total number of tests skipped: 0
Finished running tests!

> Test run finished at 9/23/2021, 10:49:31 AM <
```

20. Implementacja testów przypadku testowego 3.

- Dodaj test weryfikujący czy zgłaszany jest wyjątek, gdy plik jest pusty. Należy użyć obiektu Mock, tak aby nie było konieczności tworzenia nowych testowych plików wejściowych.

```
# tests/test_data_loader.py
import unittest
from unittest.mock import patch, Mock

from lab_app.data_loader import DataLoader

class DataLoaderTestCase(unittest.TestCase):
    """This class represents the DataLoader test cases"""

    def setUp(self):
        """Define test variables and initialize data loader."""
        self.data_loader = DataLoader()

    def test_data_loader(self):
        """Test data loader loads data from file"""
        file_name = './input_test.csv'
        expected_data=[
            ("John", "Smith"),
            ("John", "Doe")
        ]
        data = self.data_loader.read_data(file_name)

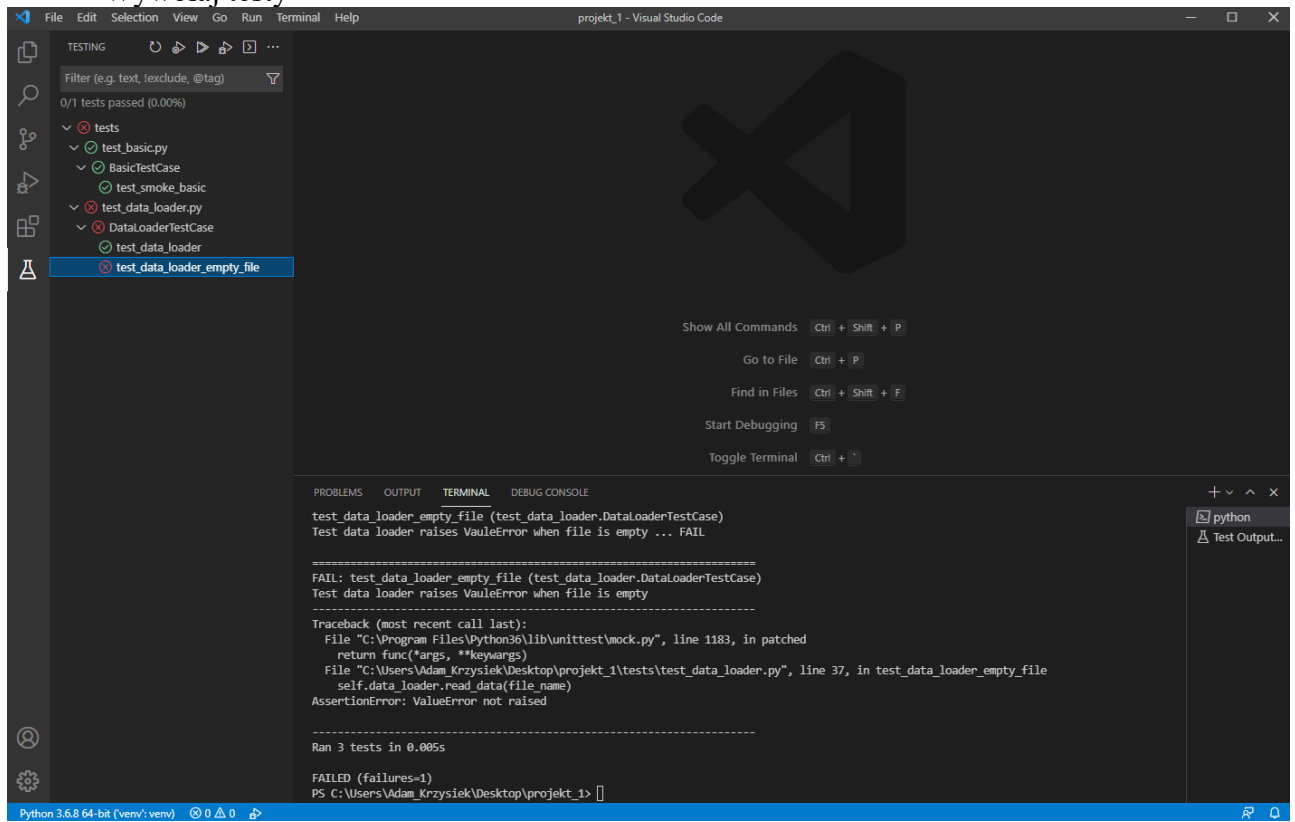
        self.assertEqual(expected_data, data)

    @patch('lab_app.data_loader.open')
    def test_data_loader_empty_file(self, open_mock):
        """Test data loader raises ValueError when file is empty"""
        # Mock jako Stub
        file_handler_mock = Mock()
        file_handler_mock.readlines = Mock(return_value="")
        open_mock.return_value = file_handler_mock

        file_name = 'some_nonexisting_file.csv'
        with self.assertRaises(ValueError):
            self.data_loader.read_data(file_name)
        # Mock jako Mock
        file_handler_mock.readlines.assert_called_once_with()
        open_mock.assert_called_once_with('some_nonexisting_file.csv', 'r')

    def tearDown(self):
        """Tear down all initialized variables and close data loader."""
        pass
```

• Wywołaj testy



- Zaimplementuj funkcjonalność – zgłaszanie wyjątku, gdy plik jest pusty tak, aby testy przechodziły.

```
# data_loader.py

class DataLoader:

    def __init__(self):
        self.data = []
        self.file_handler = None

    def open_file(self):

        self.file_handler = open(self.file_name, 'r')
        return self.file_handler

    def close_file(self):
        if self.file_handler:
            self.file_handler.close()

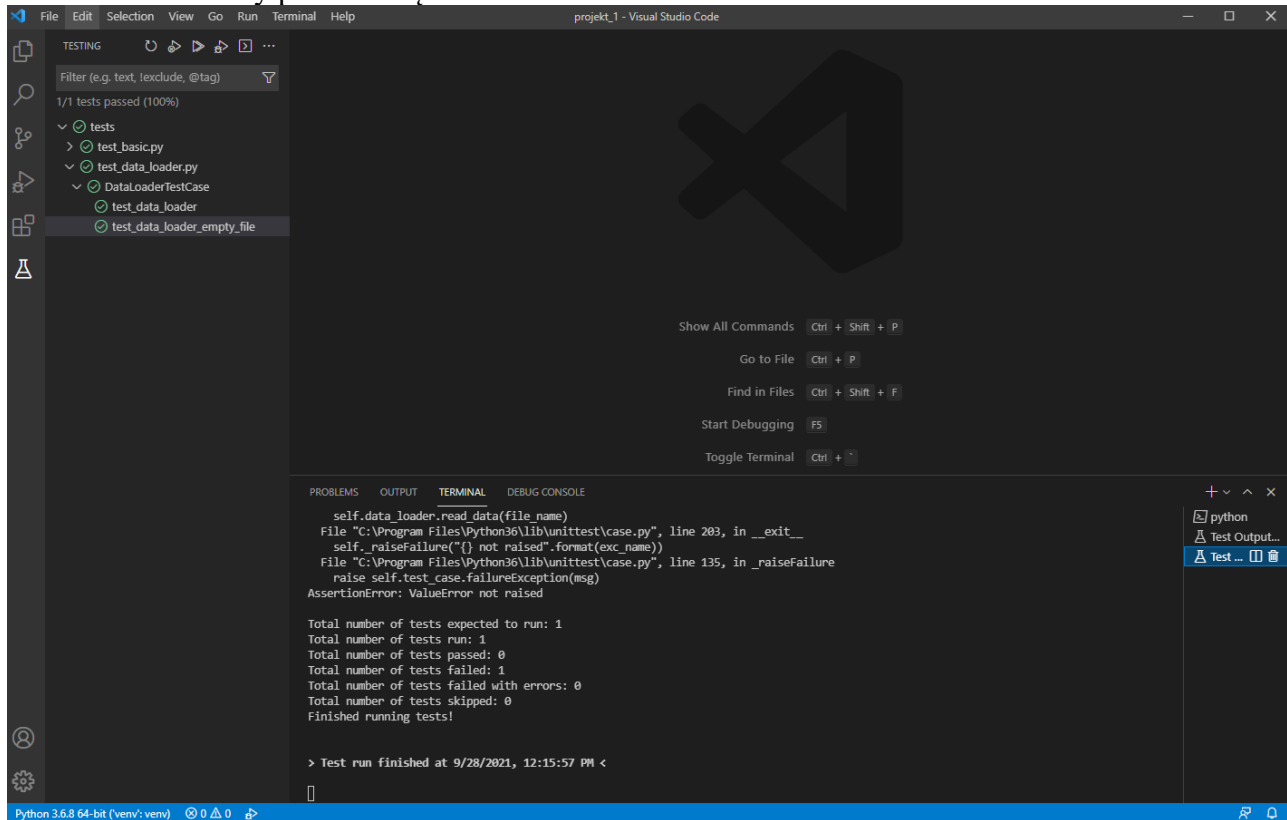
    def read_data(self, file_name):
        try:
            self.file_name = file_name
            self.file_handler = self.open_file()

            for line_no, line in enumerate(self.file_handler.readlines()):
                d = line.strip().split(',')
                data_tuple = name, surname = (d[0], d[1])
                self.data.append(data_tuple)
        finally:
            self.close_file()

        if not self.data:
            raise ValueError(f"File is empty: {self.file_name}")

        return self.data
```

- Testy przechodzą



21. Implementacja testów przypadku testowego 2.

- Utwórz plik `test_app.py` z klasą testową `AppTestCase` i dodaj test weryfikujący aplikacja poprawnie generuje adresy mail. Należy przygotować dwie wersje testu: z użyciem pliku wejściowego `input_test.csv` oraz z zamockowanym obiektem `data_loader` w instancji aplikacji.

```
# tests/test_app.py
import unittest
from unittest.mock import patch, Mock

from lab_app.main import APP

class AppTestCase(unittest.TestCase):
    """This class represents the App test case"""

    def setUp(self):
        """Define test variables and initialize app."""
        self.app = APP()

    @patch('lab_app.main.DataLoader')
    def test_app_basic(self, data_loader_mock):
        """Test APP generates mail adressess"""

        test_data = [
            ("John", "Smith"),
            ("Joe", "Doe")
        ]
```

```

data_loader_instance_mock = Mock()
data_loader_instance_mock.read_data = Mock(return_value=test_data)
data_loader_mock.return_value = data_loader_instance_mock

expected_mails=[
    "john_smith@example.com",
    "joe_doe@example.com"
]

# example of dummy
mails = APP().run(Mock())

self.assertEqual(expected_mails, mails)

def test_app_basic_spy(self):
    """Test APP generates mail adressess with spy"""
    file_name = './input_test.csv'
    expected_mails=[
        "john_smith@example.com",
        "john_doe@example.com"
    ]

    # Mock as spy
    with patch.object(self.app.data_loader, 'read_data', wraps=self.app.data_lo
ader.read_data) as wrapped_foo:

        mails = self.app.run(file_name)

        self.assertEqual(expected_mails, mails)

        wrapped_foo.assert_called_with('./input_test.csv')

def tearDown(self):
    """Tear down all initialized variables and close app."""
    pass

```

• Wywołaj testy

The screenshot displays the Visual Studio Code interface during a Python test run. The TESTING sidebar on the left shows a tree view of the test suite: tests (0/2 tests passed (0.00%)), test_app.py, AppTestCase, test_app_basic, test_app_basic.spy, test_data_loader.py, and DataLoaderTestCase. The code editor shows the test_app.py file with the following content:

```
9 def setUp(self):
10     """Define test variables and initialize app."""
11     self.app = APP()
12
13 @patch('lab_app.main.DataLoader')
14 def test_app_basic(self, data_loader_mock):
```

The PROBLEMS pane shows two errors:

- AssertionError: ['john_smith@example.com', 'joe_doe@example.com'] != None
- AssertionError: ['john_smith@example.com', 'john_doe@example.com'] != None

The TERMINAL pane shows the test output:

```
FAIL: test_app_basic (test_app.AppTestCase)
Test APP generates mail adressess with spy

Traceback (most recent call last):
  File "C:\Users\Adam_Krzysiek\Desktop\projekt_1\tests\test_app.py", line 49, in test_app_basic_spy
    self.assertEqual(expected_mails, mails)
AssertionError: ['john_smith@example.com', 'john_doe@example.com'] != None

Run 4 tests in 0.007s

FAILED (failures=2)
PS C:\Users\Adam_Krzysiek\Desktop\projekt_1>
```

The bottom status bar indicates the Python version (3.6.8 64-bit (venv: venv)), the current line and column (Ln 15, Col 1 (49 selected)), the number of spaces (4), the encoding (UTF-8), the line ending (CRLF), and the active language (Python).

- Zaimplementuj funkcjonalność generowania maili w klasie App, tak aby testy przechodziły.

```
# main.py
from lab_app.data_loader import DataLoader

INPUT_FILE_PATH = "input.csv"

class APP:

    def __init__(self):
        self.data_loader = DataLoader()

    def generate_mail(self, name, surname):
        arg_1 = [name.lower(), surname.lower()]
        m_str = "_".join(arg_1)
        return f"{m_str}@example.com"

    def run(self, file_name):
        print("Welcome to mail generator app")
        print(f"Using input file {file_name}")

        data = self.data_loader.read_data(file_name)
        mails = []
        for name, surname in data:
            _mail = self.generate_mail(name, surname)
            mails.append(_mail)
        return mails

if __name__ == "__main__":
    APP().run(INPUT_FILE_PATH)
```


• Testy przechodzą

The screenshot shows the Visual Studio Code interface with a Python project. The left sidebar displays the 'TESTING' panel, indicating that 4/4 tests passed (100%). The central editor shows the code for `test_app.py`, which includes a `setUp` method and a `test_app_basic` test function. The right sidebar shows the 'TERMINAL' panel with the output of the test run, confirming that all tests passed successfully.

```
def setUp(self):
    """Define test variables and initialize app."""
    self.app = APP()

    @patch('lab_app.main.DataLoader')
    def test_app_basic(self, data_loader_mock):
        """Test APP generates mail adressess"""

        test_data = [
            ("John", "Smith"),
            ("Joe", "Doe")]

        data_loader_instance_mock = Mock()
        data_loader_instance_mock.read_data = Mock(return_value=test_data)
        data_loader_mock.return_value = data_loader_instance_mock

        expected_mails=[
            "john_smith@example.com",
            "joe_doe@example.com"
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Ran 4 tests in 0.007s

FAILED (failures=2)

PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> python -m unittest discover -v -s tests

test_app_basic (test_app.AppTestCase)

Test APP generates mail adressess ... Welcome to mail generator app

Using input file <Mock id='2273952617584'>

ok

test_app_basic_spy (test_app.AppTestCase)

Test APP generates mail adressess with spy ... Welcome to mail generator app

Using input file ./input_test.csv

ok

test_data_loader (test_data_loader.DataLoaderTestCase)

Test data loader loads data from file ... ok

test_data_loader_empty_file (test_data_loader.DataLoaderTestCase)

Test data loader raises VauleError when file is empty ... ok

Ran 4 tests in 0.004s

OK

PS C:\Users\Adam_Krzysiek\Desktop\projekt_1> []

Python 3.6.8 64-bit (venv\venv) 0 0 0 Ln 19, Col 27 Spaces: 4 UTF-8 CRLF Python

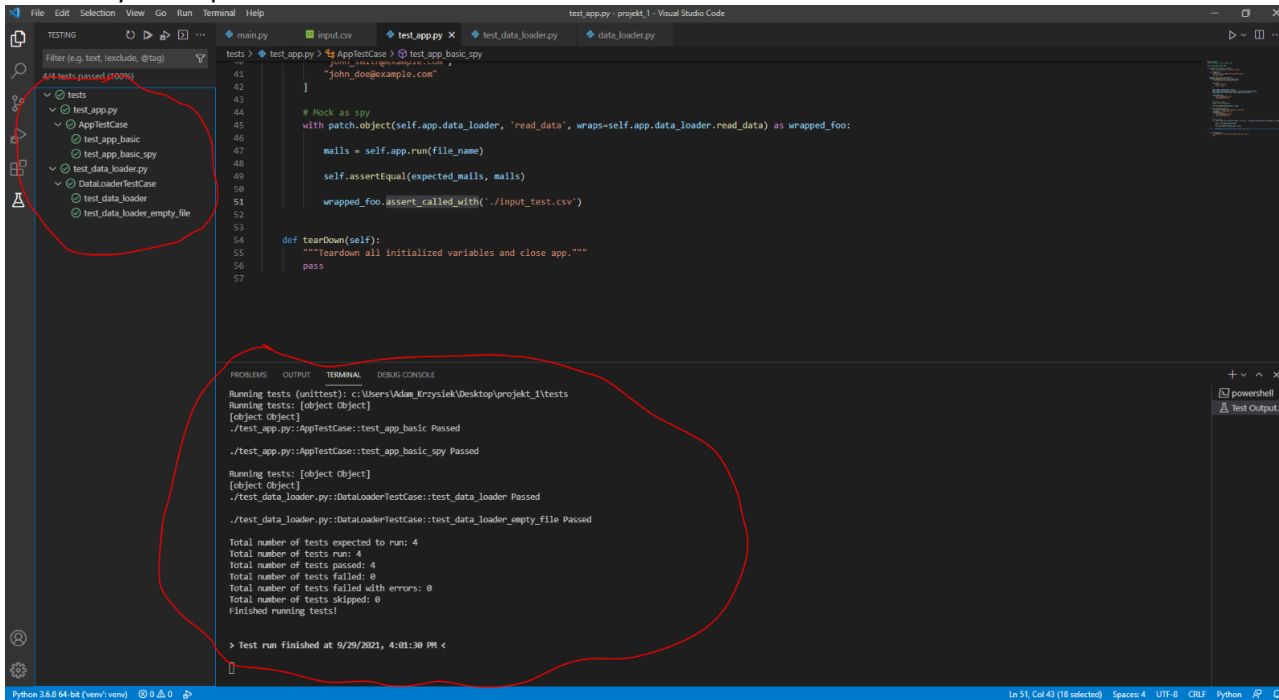
Praca własna – do sprawozdania

1. Podaj swoją definicję TDD. 1pkt

2. Zrealizuj polecenia z części wspólnej. (Obowiązkowe)

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem >

Przykład raportu:



```
File Edit Selection View Go Run Terminal Help
test_app.py - projekt_1 - Visual Studio Code

TESTING
Filter (e.g. test, !exclude, @tag)
4/4 tests passed (100%)
  tests
    test_app.py
      AppTestCase
        test_app_basic
        test_app_basic_spy
      DataLoaderTestCase
        test_data_loader
        test_data_loader_empty_file

main.py input.csv test_app.py x test_data_loader.py data_loader.py
tests > test_app.py > AppTestCase > test_app_basic_spy
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57

# Mock as spy
with patch.object(self.app.data_loader, 'read_data', wraps=self.app.data_loader.read_data) as wrapped_foo:
    mails = self.app.run(file_name)
    self.assertEqual(expected_mails, mails)
    wrapped_foo.assert_called_with('./input_test.csv')

def tearDown(self):
    """tearDown all initialized variables and close app."""
    pass

Running tests (unittest): c:\Users\Adam_Krzysiek\Desktop\projekt_1\tests
Running tests: [object Object]
[object Object]
./test_app.py::AppTestCase::test_app_basic Passed
./test_app.py::AppTestCase::test_app_basic_spy Passed
Running tests: [object Object]
[object Object]
./test_data_loader.py::DataLoaderTestCase::test_data_loader Passed
./test_data_loader.py::DataLoaderTestCase::test_data_loader_empty_file Passed

Total number of tests expected to run: 4
Total number of tests run: 4
Total number of tests passed: 4
Total number of tests failed: 0
Total number of tests failed with errors: 0
Total number of tests skipped: 0
Finished running tests!

> Test run finished at 9/29/2021, 4:01:30 PM <
```

3. Zaprojektuj oraz zaimplementuj zgodnie z TDD przypadek testowy 4 weryfikujący funkcjonalność - Importowanie danych pracowników z pliku wejściowego, w przypadku kiedy w pliku wejściowym użyto innego znaku niż przecinek jako separatora. Oczekiwanym zachowaniem jest zgłoszenie wyjątku ValueError z informacją nieprawidłowym formacie danych. Użyj przynajmniej jednego z test doubles. **1pkt**

Projekt przypadku testowego

ID	ID Scenariusza	Nazwa	Warunki wstępne	Kroki	Oczekiwany rezultat
4	2

Faza implementacji testu

<Screenshot kodu testu>

<Screenshot raportu z wywołania testów z widocznym negatywnym rezultatem>

Faza implementacji funkcjonalności

<Screenshot kodu funkcjonalności>

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem>

Faza refaktoringu implementacji funkcjonalności (jeśli potrzeba)

<Screenshot kodu funkcjonalności>

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem>

4. Zaprojektuj oraz zaimplementuj zgodnie z TDD przypadek testowy 5 weryfikujący funkcjonalność - Generowanie adresów mailowych pracowników, w przypadku kiedy pojawi się duplikat. Oczekiwanym zachowaniem jest zgłoszenie wyjątku ValueError z informacją o duplikacie wraz z numerem linii. Użyj przynajmniej jednego z test doubles. **1pkt**

Projekt przypadku testowego

ID	ID Scenariusza	Nazwa	Warunki wstępne	Kroki	Oczekiwany rezultat
5	1

Faza implementacji testu

<Screenshot kodu testu>

<Screenshot raportu z wywołania testów z widocznym negatywnym rezultatem>

Faza implementacji funkcjonalności

<Screenshot kodu funkcjonalności>

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem>

Faza refaktoringu implementacji funkcjonalności (jeśli potrzeba)

<Screenshot kodu funkcjonalności>

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem>

5. Zaprojektuj oraz zaimplementuj zgodnie z TDD przypadek testowy 6 weryfikujący dowolną „nieprzetestowaną” funkcjonalność. Użyj przynajmniej jednego z test doubles. **1pkt**

Projekt przypadku testowego

ID	ID Scenariusza	Nazwa	Warunki wstępne	Kroki	Oczekiwany rezultat
6

Faza implementacji testu

<Screenshot kodu testu>

<Screenshot raportu z wywołania testów z widocznym negatywnym rezultatem>

Faza implementacji funkcjonalności

<Screenshot kodu funkcjonalności>

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem>

Faza refaktoringu implementacji funkcjonalności (jeśli potrzeba)

<Screenshot kodu funkcjonalności>

<Screenshot raportu z wywołania testów z widocznym pozytywnym rezultatem>