# In-depth Analysis of Unit Testing and Coverage in JPacman - Tasks 2 & 3

**Author:** Ryan Amendala

**Github:** https://github.com/GNAR1ZARD/Group8/tree/master/jpacman

**Abstract:**
This report presents an exhaustive analysis of test coverage within the JPacman project using both IntelliJ IDEA's built-in coverage tool and the JaCoCo coverage analyzer. The investigation delves into the effectiveness of existing tests, introduces additional tests to improve coverage, and compares the insights gained from two different coverage tools.

## 1. Introduction:
The JPacman project, a Java-based implementation of the classic arcade game, serves as a practical platform for applying unit testing and analyzing code coverage. The initial phase employs IntelliJ's coverage plugin to establish a baseline of test coverage across the project. Subsequent efforts focus on enhancing test coverage by developing new test cases for previously untested methods.

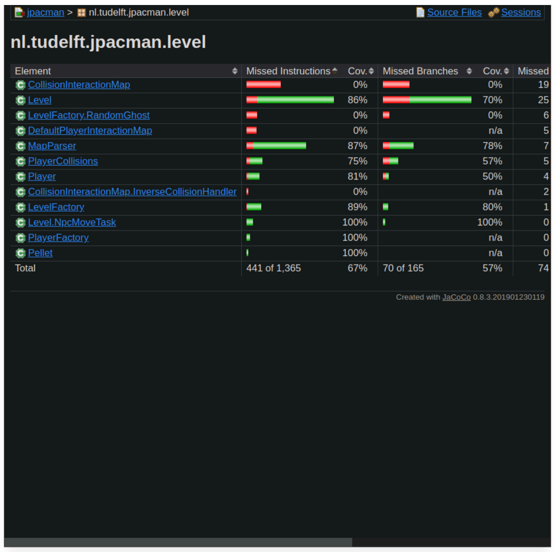
## 2. Methodology:
The process began with executing the existing test suite using the ./gradlew test command in IntelliJ's terminal, followed by a coverage analysis using IntelliJ's built-in tools. The next phase involved identifying methods with zero test coverage and writing new unit tests to cover those methods. The testing approach drew inspiration from existing test classes and employed object mocking where necessary to isolate test scenarios.

## 3. Results:
The initial coverage report revealed several packages with zero percent coverage. Subsequently, a new test case was created for the Player.isAlive() method, resulting in an increase in coverage. Additional unit tests targeted at least three methods, chosen in consultation among team members to avoid overlap.

## 4. Discussion:
Comparison of coverage results from IntelliJ and JaCoCo highlighted slight discrepancies, possibly due to differences in coverage metrics or configurations. JaCoCo's branch coverage visualization proved beneficial in identifying untested branches within methods. While IntelliJ's integration offers convenience, JaCoCo's detailed HTML report with highlighted source code provides a granular and shareable overview, making it the preferred tool for comprehensive coverage analysis.

## 5. Conclusion:
The concerted effort to write targeted unit tests based on coverage data led to measurable improvements in the JPacman project's test coverage. The comparative analysis of coverage tools underscored the importance of using dedicated tools for in-depth coverage analysis.

## 6. Appendix: Code Snippets

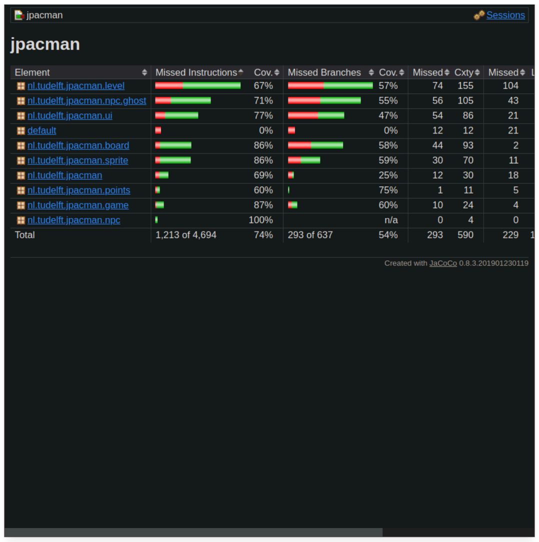


jpacman > nl.tudelft.jpacman.level    Source Files   Sessions

### nl.tudelft.jpacman.level

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed |
|---|---|---|---|---|---|
| CollisionInteractionMap | | 0% | | 0% | 19 |
| Level | | 86% | | 70% | 25 |
| LevelFactory.RandomGhost | | 0% | | 0% | 6 |
| DefaultPlayerInteractionMap | | 0% | | n/a | 5 |
| MapParser | | 87% | | 78% | 7 |
| PlayerCollisions | | 75% | | 57% | 5 |
| Player | | 81% | | 50% | 4 |
| CollisionInteractionMap.InverseCollisionHandler | | 0% | | n/a | 2 |
| LevelFactory | | 89% | | 80% | 1 |
| Level.NpcMoveTask | | 100% | | 100% | 0 |
| PlayerFactory | | 100% | | n/a | 0 |
| Pellet | | 100% | | n/a | 0 |
| Total | 441 of 1,365 | 67% | 70 of 165 | 57% | 74 |

Created with JaCoCo 0.8.3.201901230119

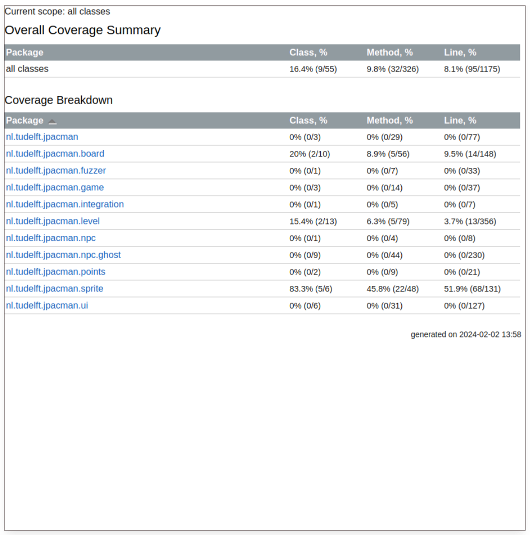


jpacman    Sessions

### jpacman

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | L |
|---|---|---|---|---|---|---|---|---|
| nl.tudelft.jpacman.level | | 67% | | 57% | 74 | 155 | 104 | |
| nl.tudelft.jpacman.npc.ghost | | 71% | | 55% | 56 | 105 | 43 | |
| nl.tudelft.jpacman.ui | | 77% | | 47% | 54 | 86 | 21 | |
| default | | 0% | | 0% | 12 | 12 | 21 | |
| nl.tudelft.jpacman.board | | 86% | | 58% | 44 | 93 | 2 | |
| nl.tudelft.jpacman.sprite | | 86% | | 59% | 30 | 70 | 11 | |
| nl.tudelft.jpacman | | 69% | | 25% | 12 | 30 | 18 | |
| nl.tudelft.jpacman.points | | 60% | | 75% | 1 | 11 | 5 | |
| nl.tudelft.jpacman.game | | 87% | | 60% | 10 | 24 | 4 | |
| nl.tudelft.jpacman.npc | | 100% | | n/a | 0 | 4 | 0 | |
| Total | 1,213 of 4,694 | 74% | 293 of 637 | 54% | 293 | 590 | 229 | 1 |

Created with JaCoCo 0.8.3.201901230119



Current scope: all classes

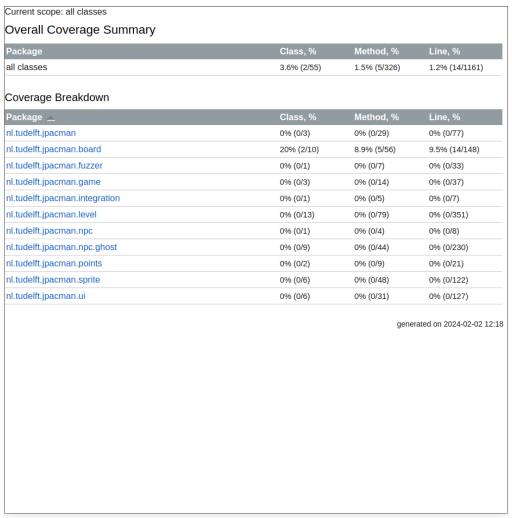
### Overall Coverage Summary

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| all classes | 16.4% (9/55) | 9.8% (32/326) | 8.1% (95/1175) |

### Coverage Breakdown

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| nl.tudelft.jpacman | 0% (0/3) | 0% (0/29) | 0% (0/77) |
| nl.tudelft.jpacman.board | 20% (2/10) | 8.9% (5/56) | 9.5% (14/148) |
| nl.tudelft.jpacman.fuzzer | 0% (0/1) | 0% (0/7) | 0% (0/33) |
| nl.tudelft.jpacman.game | 0% (0/3) | 0% (0/14) | 0% (0/37) |
| nl.tudelft.jpacman.integration | 0% (0/1) | 0% (0/5) | 0% (0/7) |
| nl.tudelft.jpacman.level | 15.4% (2/13) | 6.3% (5/79) | 3.7% (13/356) |
| nl.tudelft.jpacman.npc | 0% (0/1) | 0% (0/4) | 0% (0/8) |
| nl.tudelft.jpacman.npc.ghost | 0% (0/9) | 0% (0/44) | 0% (0/230) |
| nl.tudelft.jpacman.points | 0% (0/2) | 0% (0/9) | 0% (0/21) |
| nl.tudelft.jpacman.sprite | 83.3% (5/6) | 45.8% (22/48) | 51.9% (68/131) |
| nl.tudelft.jpacman.ui | 0% (0/6) | 0% (0/31) | 0% (0/127) |

generated on 2024-02-02 13:58



Current scope: all classes

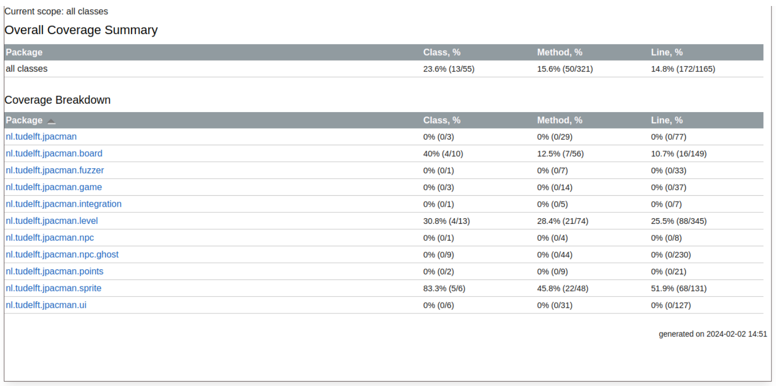
### Overall Coverage Summary

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| all classes | 3.6% (2/55) | 1.5% (5/326) | 1.2% (14/1161) |

### Coverage Breakdown

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| nl.tudelft.jpacman | 0% (0/3) | 0% (0/29) | 0% (0/77) |
| nl.tudelft.jpacman.board | 20% (2/10) | 8.9% (5/56) | 9.5% (14/148) |
| nl.tudelft.jpacman.fuzzer | 0% (0/1) | 0% (0/7) | 0% (0/33) |
| nl.tudelft.jpacman.game | 0% (0/3) | 0% (0/14) | 0% (0/37) |
| nl.tudelft.jpacman.integration | 0% (0/1) | 0% (0/5) | 0% (0/7) |
| nl.tudelft.jpacman.level | 0% (0/13) | 0% (0/79) | 0% (0/351) |
| nl.tudelft.jpacman.npc | 0% (0/1) | 0% (0/4) | 0% (0/8) |
| nl.tudelft.jpacman.npc.ghost | 0% (0/9) | 0% (0/44) | 0% (0/230) |
| nl.tudelft.jpacman.points | 0% (0/2) | 0% (0/9) | 0% (0/21) |
| nl.tudelft.jpacman.sprite | 0% (0/6) | 0% (0/48) | 0% (0/122) |
| nl.tudelft.jpacman.ui | 0% (0/6) | 0% (0/31) | 0% (0/127) |

generated on 2024-02-02 12:18



Current scope: all classes

### Overall Coverage Summary

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| all classes | 23.6% (13/55) | 15.6% (50/321) | 14.8% (172/1165) |

### Coverage Breakdown

| Package | Class, % | Method, % | Line, % |
|---|---|---|---|
| nl.tudelft.jpacman | 0% (0/3) | 0% (0/29) | 0% (0/77) |
| nl.tudelft.jpacman.board | 40% (4/10) | 12.5% (7/56) | 10.7% (16/149) |
| nl.tudelft.jpacman.fuzzer | 0% (0/1) | 0% (0/7) | 0% (0/33) |
| nl.tudelft.jpacman.game | 0% (0/3) | 0% (0/14) | 0% (0/37) |
| nl.tudelft.jpacman.integration | 0% (0/1) | 0% (0/5) | 0% (0/7) |
| nl.tudelft.jpacman.level | 30.8% (4/13) | 28.4% (21/74) | 25.5% (88/345) |
| nl.tudelft.jpacman.npc | 0% (0/1) | 0% (0/4) | 0% (0/8) |
| nl.tudelft.jpacman.npc.ghost | 0% (0/9) | 0% (0/44) | 0% (0/230) |
| nl.tudelft.jpacman.points | 0% (0/2) | 0% (0/9) | 0% (0/21) |
| nl.tudelft.jpacman.sprite | 83.3% (5/6) | 45.8% (22/48) | 51.9% (68/131) |
| nl.tudelft.jpacman.ui | 0% (0/6) | 0% (0/31) | 0% (0/127) |

generated on 2024-02-02 14:51

# Advancing Test Coverage in Python for the Test Coverage Lab - Task 4

**Abstract:**
This report details the progressive approach taken to enhance the test coverage of the Test Coverage Lab's Python project. Utilizing nosetests and coverage analysis, the task centered on identifying untested code segments and crafting tests to cover these lines, with the objective of achieving as close to 100% coverage as possible.

## 1. Introduction:

The Python Testing lab's test coverage project serves as an ideal case study for demonstrating the efficacy of systematic testing practices. This exercise involves evaluating existing test coverage, pinpointing code not covered by tests, and developing test cases to fill these gaps.

## 2. Methodology:

The project was cloned, and necessary dependencies were installed.
Initial test coverage was gauged using nosetests with coverage reporting, which revealed a starting coverage of 72%.
A detailed examination of the coverage report identified specific lines in models/account.py that lacked test coverage, which guided the creation of new tests.
Each newly written test was followed by rerunning nosetests to ensure coverage improvement and to identify the next line requiring a test case.

## 3. Results

Detailed descriptions of each test case:

- test_from_dict validated the functionality of updating account attributes from a dictionary.
- test_account_update confirmed the update operation's correctness for an existing account in the database.
- test_account_update_with_empty_id ensured the proper exception was raised when attempting to update an account without an ID.
- test_account_delete tested the deletion process and subsequent non-existence of an account.
- test_all_accounts checked the retrieval of all accounts, affirming the all method's effectiveness.
- test_find_account focused on locating an account by its ID, guaranteeing accurate retrieval functionality.

Post-integration of the new tests, the coverage report reflected a 100% coverage achievement. This represented a comprehensive validation of all code paths in the Account model.

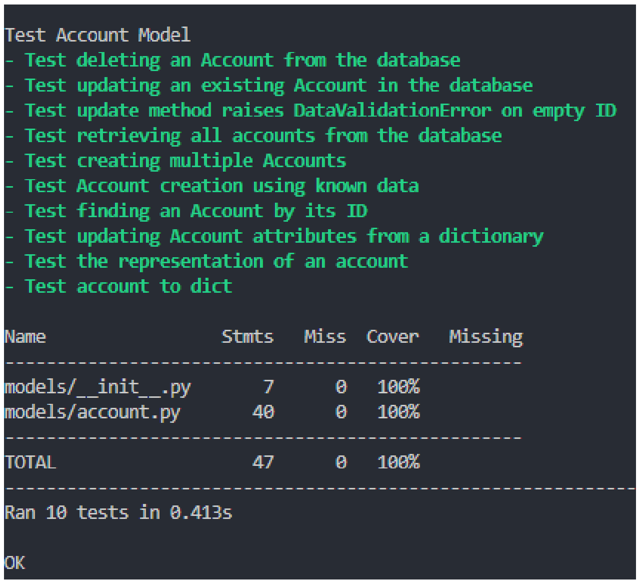
## 4. Discussion:

Each test case addressed specific functionalities within the codebase, showcasing a targeted approach to uncovering potential bugs and ensuring expected behavior. The incremental coverage increases demonstrated the value of each test in the overall suite.

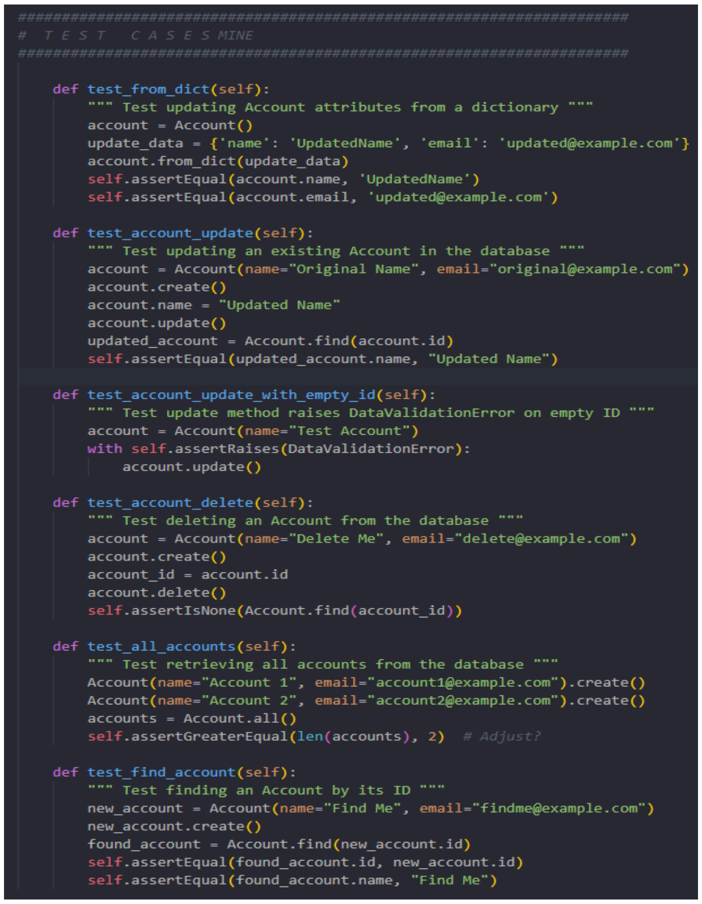
## 5. Conclusion:

The task exemplified the importance of thorough testing in software development. By achieving full test coverage, we have significantly reduced the risk of undetected issues in the codebase and bolstered the confidence in its reliability.

## 6. Appendix: Code Snippets



```
Test Account Model
- Test deleting an Account from the database
- Test updating an existing Account in the database
- Test update method raises DataValidationError on empty ID
- Test retrieving all accounts from the database
- Test creating multiple Accounts
- Test Account creation using known data
- Test finding an Account by its ID
- Test updating Account attributes from a dictionary
- Test the representation of an account
- Test account to dict

Name                    Stmts   Miss  Cover   Missing
-----------------------------------------------------
models/__init__.py         7      0   100%
models/account.py         40      0   100%
-----------------------------------------------------
TOTAL                     47      0   100%
-----------------------------------------------------------------

Ran 10 tests in 0.413s

OK
```



```
######################################################################
#  T E S T   C A S E S MINE
######################################################################

    def test_from_dict(self):
        """ Test updating Account attributes from a dictionary """
        account = Account()
        update_data = {'name': 'UpdatedName', 'email': 'updated@example.com'}
        account.from_dict(update_data)
        self.assertEqual(account.name, 'UpdatedName')
        self.assertEqual(account.email, 'updated@example.com')

    def test_account_update(self):
        """ Test updating an existing Account in the database """
        account = Account(name="Original Name", email="original@example.com")
        account.create()
        account.name = "Updated Name"
        account.update()
        updated_account = Account.find(account.id)
        self.assertEqual(updated_account.name, "Updated Name")

    def test_account_update_with_empty_id(self):
        """ Test update method raises DataValidationError on empty ID """
        account = Account(name="Test Account")
        with self.assertRaises(DataValidationError):
            account.update()

    def test_account_delete(self):
        """ Test deleting an Account from the database """
        account = Account(name="Delete Me", email="delete@example.com")
        account.create()
        account_id = account.id
        account.delete()
        self.assertIsNone(Account.find(account_id))

    def test_all_accounts(self):
        """ Test retrieving all accounts from the database """
        Account(name="Account 1", email="account1@example.com").create()
        Account(name="Account 2", email="account2@example.com").create()
        accounts = Account.all()
        self.assertGreaterEqual(len(accounts), 2)  # Adjust?

    def test_find_account(self):
        """ Test finding an Account by its ID """
        new_account = Account(name="Find Me", email="findme@example.com")
        new_account.create()
        found_account = Account.find(new_account.id)
        self.assertEqual(found_account.id, new_account.id)
        self.assertEqual(found_account.name, "Find Me")
```

# Implementing Test Driven Development for RESTful Services in Python - Task 5

**Abstract:**
This report encapsulates the application of Test-Driven Development (TDD) to implement a RESTful service in Python. Following the TDD mantra of "Red, Green, Refactor," I developed a counter service, rigorously tested by a suite of unit tests. The iterative approach not only validated the functionality of HTTP methods but also reinforced best practices in software development.

## 1. Introduction:
The practice of TDD was applied to a Python project structured to mimic a RESTful service. The task involved the creation, updating, and reading of a counter resource using HTTP methods, with the TDD cycle guiding the development process.

## 2. Methodology:
The TDD approach required the initial drafting of test cases that delineated the expected functionality of the RESTful service. These tests were executed to confirm their failure in the absence of implementation (Red phase). Corresponding service logic was then implemented to satisfy the test cases (Green phase). Upon successful test completion, the code was reviewed for potential refactoring opportunities to enhance maintainability and readability (Refactor phase).

## 3. Results:
Red Phase:

The initial addition of the test_update_a_counter method in test_counter.py results in a failing test because the functionality to update a counter is not yet implemented in counter.py. This is the expected outcome in the Red phase of TDD, where tests are written to fail initially.

Green Phase:

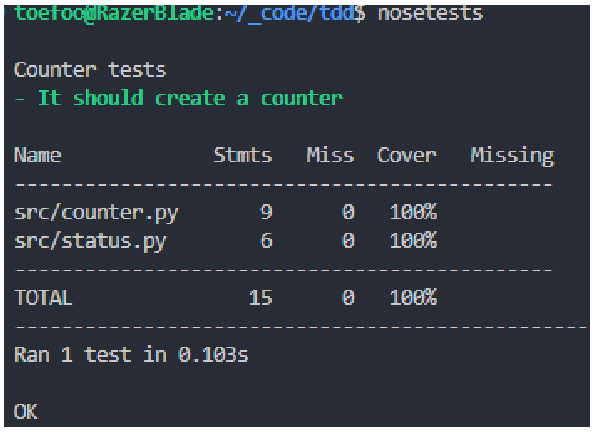
Following the Red phase, the update_counter function is implemented in counter.py. This function allows for the update of an existing counter's value via a PUT request. Running the tests again after this implementation resulted in the test_update_a_counter method passing, as the required endpoint and logic are now in place. This is the Green phase where the previously failing tests now pass due to the new code.

## 4. Discussion:
Initial `ModuleNotFoundError` and `ImportError` were resolved by creating the necessary files and scaffolding for the Flask application. The `AssertionError` encountered during the Red phase drove the development of the required RESTful endpoints.
This logic was then duplicated for the subsequent sections of the task.

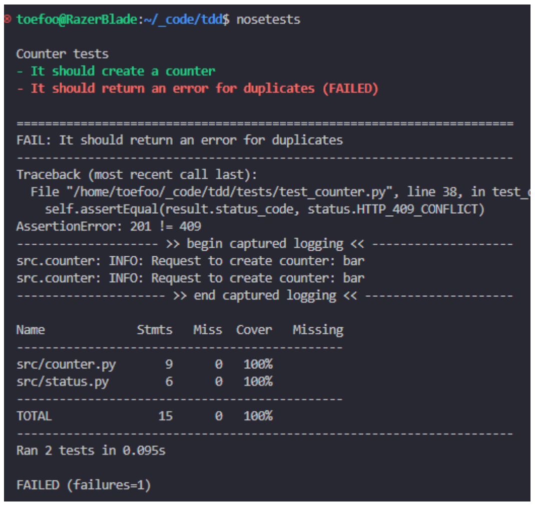
## 5. Conclusion:

The TDD methodology proved instrumental in developing a robust RESTful service by ensuring all code committed to the project was functional and tested. This led to an incrementally built, thoroughly tested, and well documented codebase.

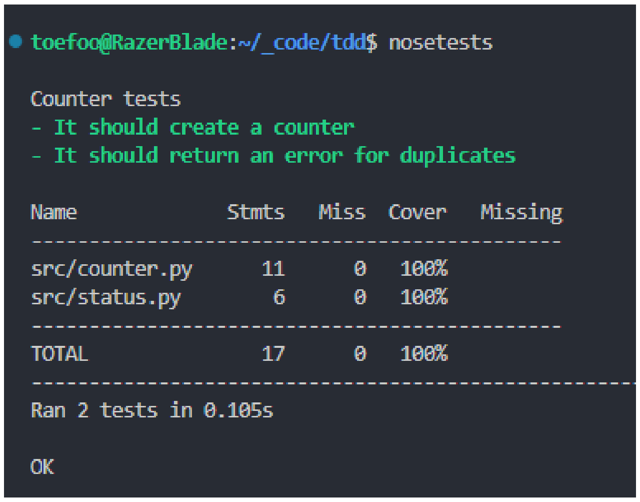
## 6. Code Snippets:



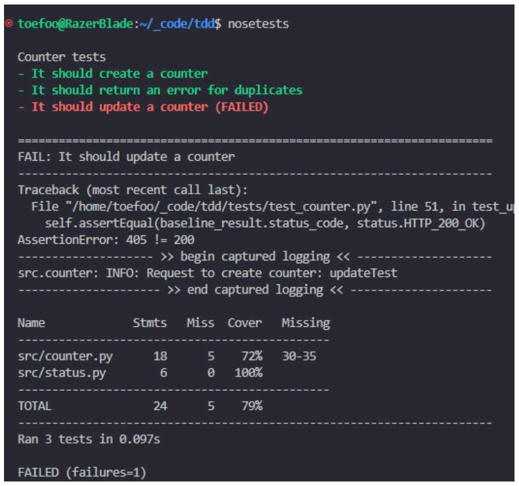
```
toefoo@RazerBlade:~/_code/tdd$ nosetests

Counter tests
- It should create a counter

Name               Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py         9      0   100%
src/status.py          6      0   100%
------------------------------------------------
TOTAL                 15      0   100%
----------------------------------------------------------------------
Ran 1 test in 0.103s

OK
```



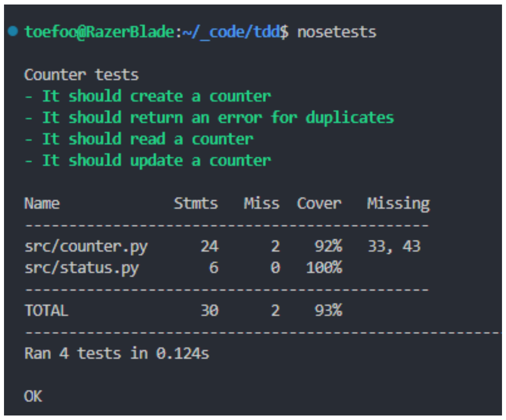
```
toefoo@RazerBlade:~/_code/tdd$ nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates (FAILED)

======================================================================
FAIL: It should return an error for duplicates
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/toefoo/_code/tdd/tests/test_counter.py", line 38, in test_
    self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
AssertionError: 201 != 409
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: bar
src.counter: INFO: Request to create counter: bar
--------------------- >> end captured logging << ---------------------

Name               Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py         9      0   100%
src/status.py          6      0   100%
------------------------------------------------
TOTAL                 15      0   100%
----------------------------------------------------------------------
Ran 2 tests in 0.095s

FAILED (failures=1)
```



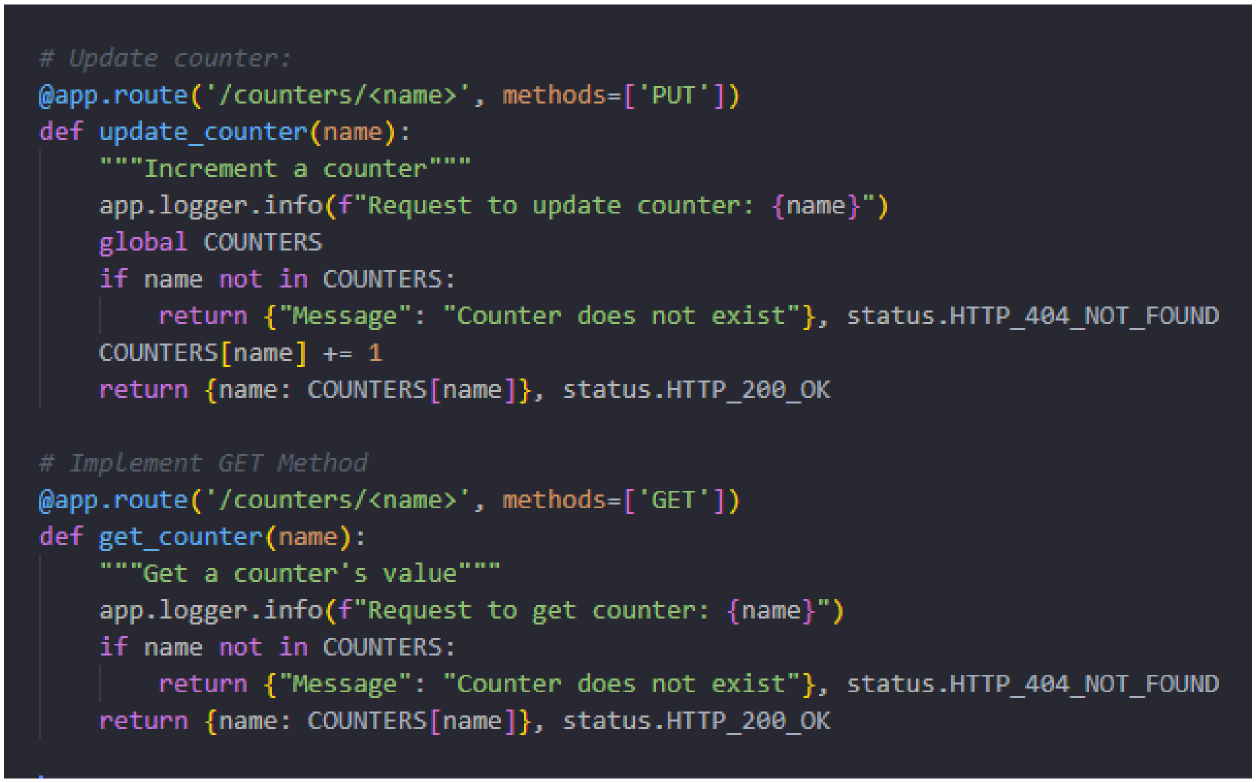
```
toefoo@RazerBlade:~/_code/tdd$ nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates

Name               Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py        11      0   100%
src/status.py          6      0   100%
------------------------------------------------
TOTAL                 17      0   100%
----------------------------------------------------------------------
Ran 2 tests in 0.105s

OK
```



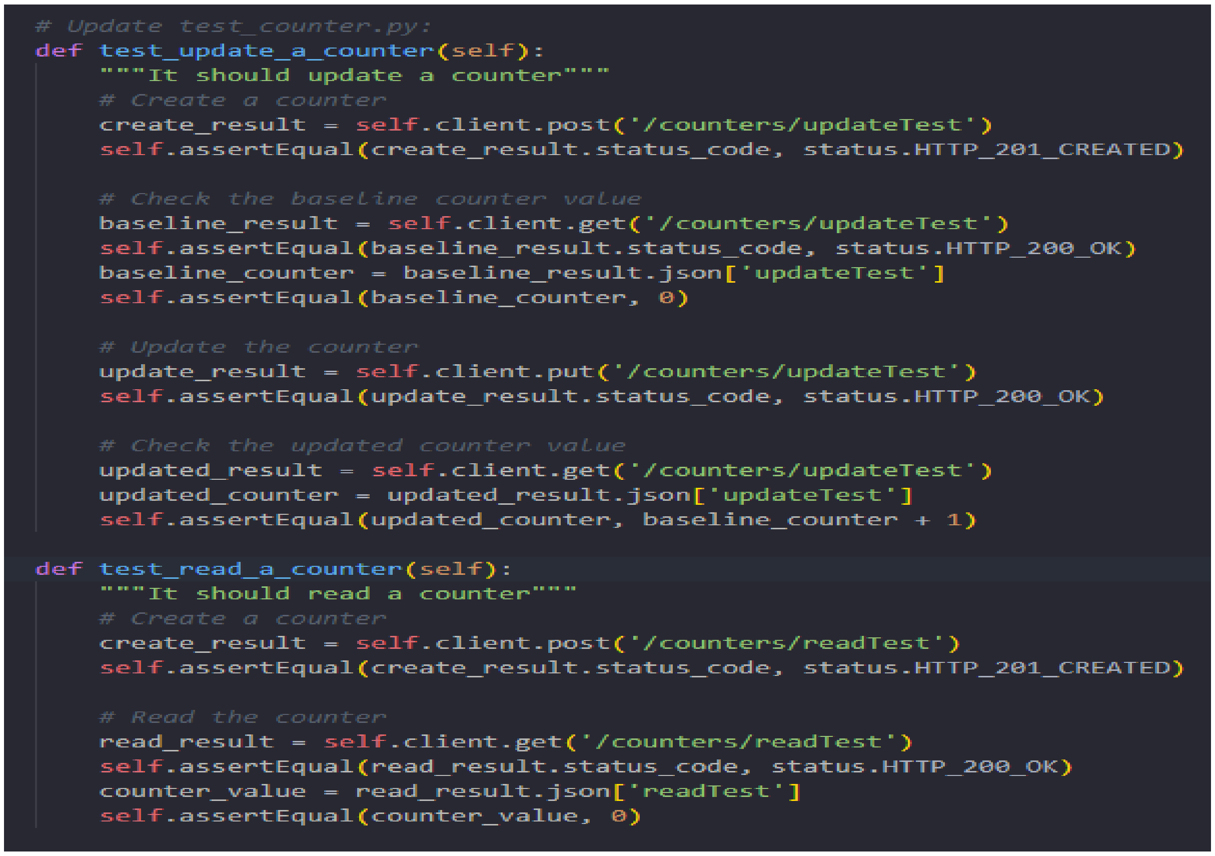
```
toefoo@RazerBlade:~/_code/tdd$ nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should update a counter (FAILED)

======================================================================
FAIL: It should update a counter
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/toefoo/_code/tdd/tests/test_counter.py", line 51, in test_u
    self.assertEqual(baseline_result.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
-------------------- >> begin captured logging << --------------------
src.counter: INFO: Request to create counter: updateTest
--------------------- >> end captured logging << ---------------------

Name               Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py        18      5    72%   30-35
src/status.py          6      0   100%
------------------------------------------------
TOTAL                 24      5    79%
----------------------------------------------------------------------
Ran 3 tests in 0.097s

FAILED (failures=1)
```



```
toefoo@RazerBlade:~/_code/tdd$ nosetests

Counter tests
- It should create a counter
- It should return an error for duplicates
- It should read a counter
- It should update a counter

Name               Stmts   Miss  Cover   Missing
------------------------------------------------
src/counter.py        24      2    92%   33, 43
src/status.py          6      0   100%
------------------------------------------------
TOTAL                 30      2    93%
----------------------------------------------------------------------
Ran 4 tests in 0.124s

OK
```



```python
# Update counter:
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Increment a counter"""
    app.logger.info(f"Request to update counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message": "Counter does not exist"}, status.HTTP_404_NOT_FOUND
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK

# Implement GET Method
@app.route('/counters/<name>', methods=['GET'])
def get_counter(name):
    """Get a counter's value"""
    app.logger.info(f"Request to get counter: {name}")
    if name not in COUNTERS:
        return {"Message": "Counter does not exist"}, status.HTTP_404_NOT_FOUND
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```



```python
# Update test_counter.py:
def test_update_a_counter(self):
    """It should update a counter"""
    # Create a counter
    create_result = self.client.post('/counters/updateTest')
    self.assertEqual(create_result.status_code, status.HTTP_201_CREATED)

    # Check the baseline counter value
    baseline_result = self.client.get('/counters/updateTest')
    self.assertEqual(baseline_result.status_code, status.HTTP_200_OK)
    baseline_counter = baseline_result.json['updateTest']
    self.assertEqual(baseline_counter, 0)

    # Update the counter
    update_result = self.client.put('/counters/updateTest')
    self.assertEqual(update_result.status_code, status.HTTP_200_OK)

    # Check the updated counter value
    updated_result = self.client.get('/counters/updateTest')
    updated_counter = updated_result.json['updateTest']
    self.assertEqual(updated_counter, baseline_counter + 1)

def test_read_a_counter(self):
    """It should read a counter"""
    # Create a counter
    create_result = self.client.post('/counters/readTest')
    self.assertEqual(create_result.status_code, status.HTTP_201_CREATED)

    # Read the counter
    read_result = self.client.get('/counters/readTest')
    self.assertEqual(read_result.status_code, status.HTTP_200_OK)
    counter_value = read_result.json['readTest']
    self.assertEqual(counter_value, 0)
```