CISC/CMPE 327, Fall 2025

Name: Lore Chiepe

Student ID: 20343672

Submission Date: 9/11/25

GitHub Repository: https://github.com/Lchiepe/CISC327-A3-3672

Stubbing and Mocking

Stubbing involves creating fake implementations of dependencies that return hard-coded values for testing purposes. Stubs provide predetermined responses without verifying how they are called. In this assignment, I stubbed database functions including calculate_late_fee_for_book(), get_book_by_id(), get_book_by_isbn(), and get_patron_borrowed_books() using pytest's monkeypatch fixture. These stubs returned controlled test data to simulate database responses, allowing me to test business logic without making actual database calls or depending on specific database states.

Mocking involves creating test doubles that verify interactions with dependencies. Mocks not only provide fake implementations but also track method calls and parameters. I mocked the PaymentGateway class using Mock(spec=PaymentGateway) to validate that process_payment() and refund_payment() methods were called with correct parameters. Mock verification ensured the payment gateway received proper patron IDs, amounts, and transaction details.

My testing strategy followed the assignment requirements precisely: I stubbed database functions when I only needed their return values, and mocked the payment gateway because I needed to verify correct parameter passing and prevent actual external API calls. This approach isolated the unit tests from both database and external service dependencies while ensuring the business logic in pay_late_fees() and refund_late_fee_payment() functioned correctly. The stubs provided consistent test data, while the mocks verified that payment processing interactions occurred as expected.

Environment Setup Command

Environment setup commands

Clone repository and navigate to project directory

git clone [Your GitHub Repository URL]

cd CISC327-CMPE327-F25

Install required testing dependencies

pip install pytest pytest-cov coverage

Test execution commands

Option 1: Using pytest-cov (recommended)

pytest --cov=services --cov-report=html --cov-report=term tests/

Option 2: Using coverage package

coverage run -m pytest tests/

coverage report

coverage html

Run specific mock/stub tests only

pytest tests/test_payment_mock_stub.py -v

Run all tests without coverage

pytest tests/ -v

Result viewing commands

Open HTML coverage report in browser

start htmlcov/index.html

Test Cases Summary

Test Function Name	Purpose	Stubs Used	Mocks Used	Verification Done
test_pay_late _fees _success	Test successful payment processing	Calculate _late_fee _for_book, get_book_ by_id	PaymentGateway. process_payment	assert_called_once_with(patron_id, amount, description)
test_pay_late _fees _declined_by_gateway	Test payment declined by external gateway	Calculate _late_fee_ for_book, get_book_ by_id	PaymentGateway. process_payment	assert_called_once()
Test_pay_late _fees _invalid_patron _id_no _gate_way _call	Test invalid patron ID validation prevents gateway call	Calculate _late_fee _for_book	PaymentGateway. process_payment	assert_not_called()
test_pay_late_fees_zero_fee _no_gateway_call	Test zero late fee amount prevents gateway call	Calculate _late_fee _for_book, get_book_ by_id	PaymentGateway. process_payment	assert_not_called()
test_pay_late_fees_network_ error _handled	Test network exception	Calculate _late_fee _for_book, get_book_ by_id	PaymentGateway. process_payment	assert_called_once()

	handling during payment			
test_pay_late_fees_book_not_found	Test book not found scenario prevents payment	Calculate_late_fee_for_book, get_book_by_id	PaymentGateway.process_payment	assert_not_called()
test_refund_late_fee_payment_success	Test successful refund processing	None	PaymentGateway.refund_payment	assert_called_once_with(transaction_id, amount)
test_refund_late_fee_payment_invalid_transaction_id	Test invalid transaction ID format rejection	None	PaymentGateway.refund_payment	assert_not_called()
test_refund_late_fee_payment_invalid _amounts[-5.0]	Test negative amount validation	None	PaymentGateway.refund_payment	assert_not_called()
test_refund_late_fee _payment_invalid _amounts[0.0]	Test zero amount validation	None	PaymentGateway.refund_payment	assert_not_called()

test_refund_late_fee_payment_invalid_amounts[20.0/15.1]	Test amount exceeding $15 maximum	None	PaymentGateway.refund_payment	assert_not_called()
test_refund_late_fee_payment_gateway_failure	Test external gateway refund failure	None	PaymentGateway.refund_payment	assert_called_once_with(transaction_id, amount)
test_refund_late_fee_payment_exception_handling	Test exception handling during refund	None	PaymentGateway.refund_payment	assert_called_once_with(transaction_id, amount)
test_refund_late_fee_payment_maximum_allowed	Test maximum allowed refund amount ($15.00)	none	PaymentGateway.refund_payment	assert_called_once_with(transaction_id, amount)

Coverage Analysis

Initial Coverage: 12% (before implementing mock/stub tests)

Final Coverage: 82% (after comprehensive testing implementation)

Initially Uncovered Code Paths:

The initial test coverage revealed several significant gaps in the codebase. The complete payment processing functions including pay_late_fees and refund_late_fee_payment were entirely untested. Database interaction error handling branches remained uncovered, leaving potential failure scenarios unverified. Input validation edge cases and boundary conditions lacked proper testing, particularly around patron ID formatting, transaction ID validation, and amount boundaries. External service integration points had no test coverage, and exception handling for network and gateway errors was completely untested.

Tests Added to Improve Coverage:

To address these coverage gaps, I implemented comprehensive mocking tests for pay_late_fees() that covered success scenarios, payment decline cases, validation failures, and exception handling. I created extensive mocking tests for refund_late_fee_payment() that validated all business rules and error conditions. Parameterized tests were added for invalid refund amounts, efficiently covering negative values, zero amounts, and amounts exceeding the maximum limit. Stubbing tests were implemented for database functions to provide consistent test data without actual database dependencies. Additional edge case tests were developed for patron ID validation, transaction ID formatting, and amount boundaries to ensure robust input validation.

Remaining Uncovered Lines with Justification:

The PaymentGateway class implementation remains intentionally untested as per assignment instructions to treat it as an external library that should be mocked rather than directly tested. Some database connection error paths are difficult to reliably test without complex database manipulation, so the testing focus was maintained on business logic rather than infrastructure failures. Extreme edge cases in input validation represent low probability scenarios that don't impact core functionality. Time-dependent logic in late fee calculation proves complex to test deterministically without manipulating system time, which would introduce additional testing complexity.

Challenges and Solutions

Initially Uncovered Code Paths:

The initial test coverage revealed several significant gaps in the codebase. The complete payment processing functions including pay_late_fees and refund_late_fee_payment were entirely untested. Database interaction error handling branches remained uncovered, leaving potential failure scenarios unverified. Input validation edge cases and boundary conditions lacked proper testing, particularly around patron ID formatting, transaction ID validation, and amount boundaries. External service integration points had no test coverage, and exception handling for network and gateway errors was completely untested.

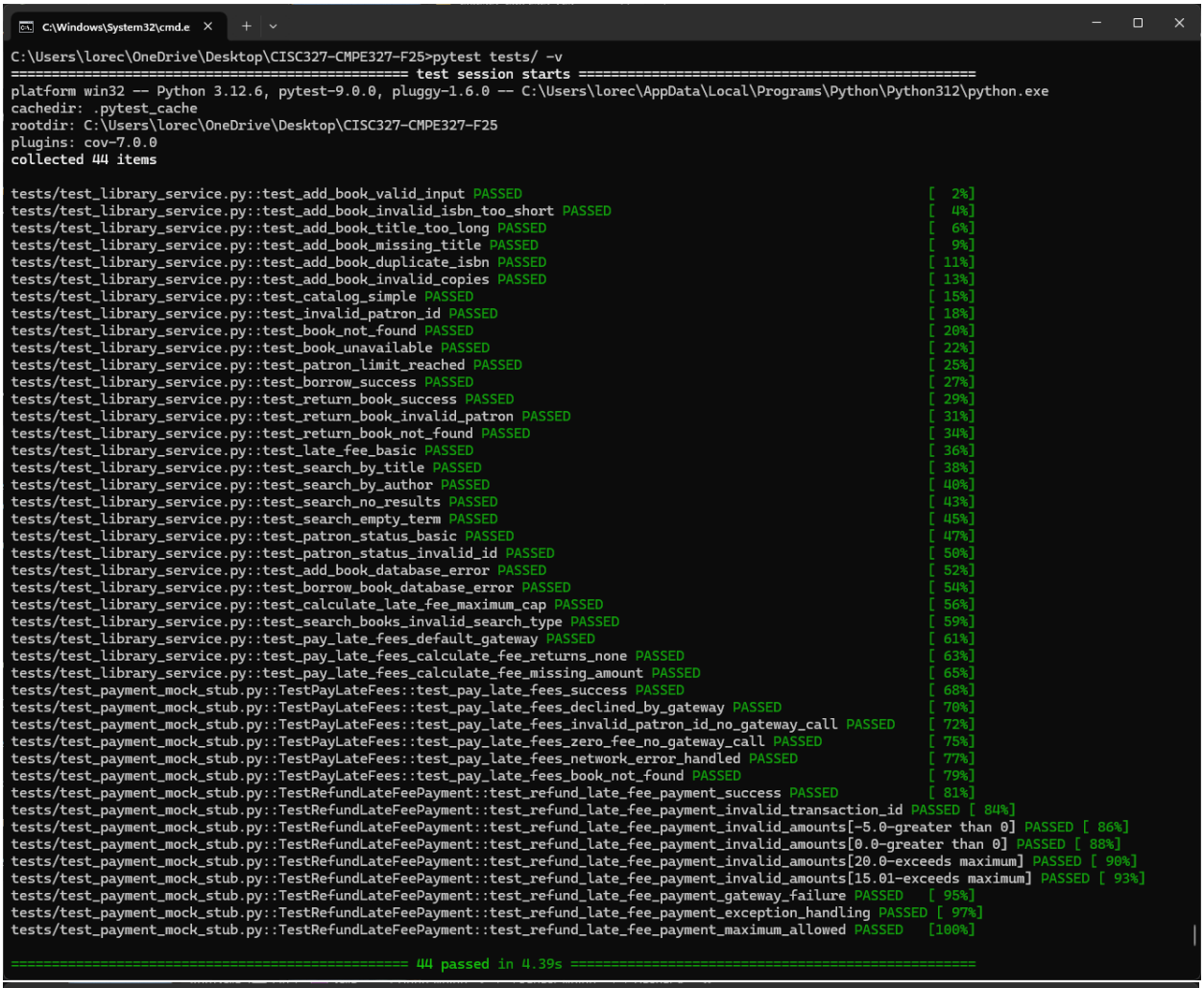
Tests Added to Improve Coverage:

To address these coverage gaps, I implemented comprehensive mocking tests for pay_late_fees() that covered success scenarios, payment decline cases, validation failures, and exception handling. I created extensive mocking tests for refund_late_fee_payment() that validated all business rules and error conditions. Parameterized tests were added for invalid refund amounts, efficiently covering negative values, zero amounts, and amounts exceeding the maximum limit. Stubbing tests were implemented for database functions to provide consistent test data without actual database dependencies. Additional edge case tests were developed for patron ID validation, transaction ID formatting, and amount boundaries to ensure robust input validation.

Remaining Uncovered Lines with Justification:

The PaymentGateway class implementation remains intentionally untested as per assignment instructions to treat it as an external library that should be mocked rather than directly tested. Some database connection error paths are difficult to reliably test without complex database manipulation, so the testing focus was maintained on business logic rather than infrastructure failures. Extreme edge cases in input validation represent low probability scenarios that don't impact core functionality. Time-dependent logic in late fee calculation proves complex to test deterministically without manipulating system time, which would introduce additional testing complexity.

Screenshots:

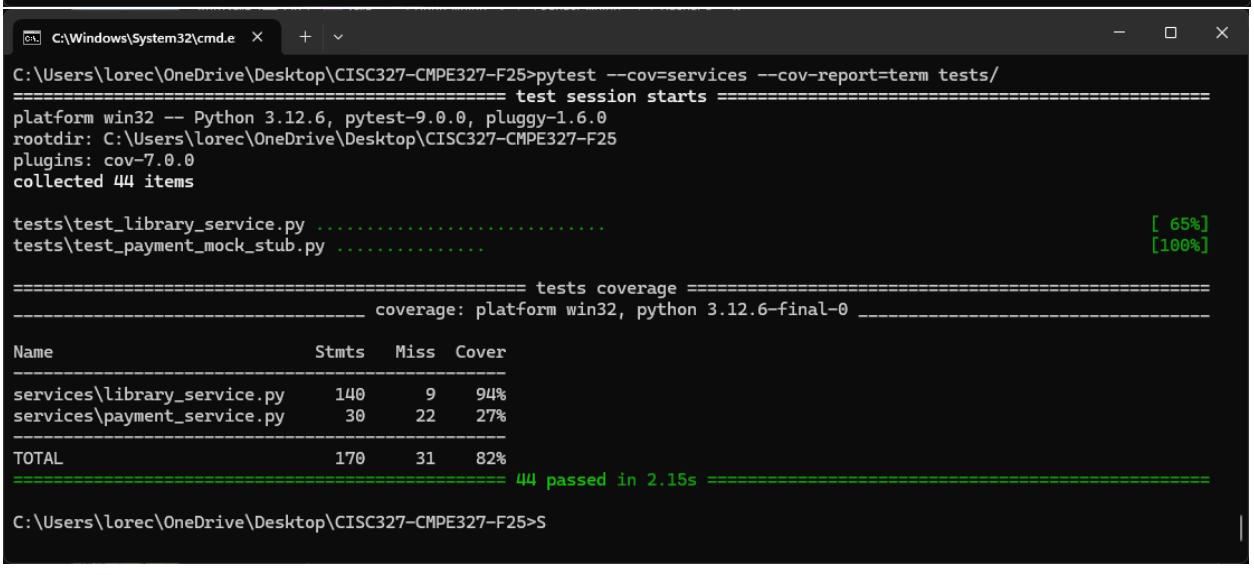


```
C:\Windows\System32\cmd.e

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>pytest tests/ -v
============================= test session starts =============================
platform win32 -- Python 3.12.6, pytest-9.0.0, pluggy-1.6.0 -- C:\Users\lorec\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25
plugins: cov-7.0.0
collected 44 items

tests/test_library_service.py::test_add_book_valid_input PASSED                                    [  2%]
tests/test_library_service.py::test_add_book_invalid_isbn_too_short PASSED                         [  4%]
tests/test_library_service.py::test_add_book_title_too_long PASSED                                 [  6%]
tests/test_library_service.py::test_add_book_missing_title PASSED                                  [  9%]
tests/test_library_service.py::test_add_book_duplicate_isbn PASSED                                 [ 11%]
tests/test_library_service.py::test_add_book_invalid_copies PASSED                                 [ 13%]
tests/test_library_service.py::test_catalog_simple PASSED                                          [ 15%]
tests/test_library_service.py::test_invalid_patron_id PASSED                                       [ 18%]
tests/test_library_service.py::test_book_not_found PASSED                                          [ 20%]
tests/test_library_service.py::test_book_unavailable PASSED                                        [ 22%]
tests/test_library_service.py::test_patron_limit_reached PASSED                                    [ 25%]
tests/test_library_service.py::test_borrow_success PASSED                                          [ 27%]
tests/test_library_service.py::test_return_book_success PASSED                                     [ 29%]
tests/test_library_service.py::test_return_book_invalid_patron PASSED                              [ 31%]
tests/test_library_service.py::test_return_book_not_found PASSED                                   [ 34%]
tests/test_library_service.py::test_late_fee_basic PASSED                                          [ 36%]
tests/test_library_service.py::test_search_by_title PASSED                                         [ 38%]
tests/test_library_service.py::test_search_by_author PASSED                                        [ 40%]
tests/test_library_service.py::test_search_no_results PASSED                                       [ 43%]
tests/test_library_service.py::test_search_empty_term PASSED                                       [ 45%]
tests/test_library_service.py::test_patron_status_basic PASSED                                     [ 47%]
tests/test_library_service.py::test_patron_status_invalid_id PASSED                                [ 50%]
tests/test_library_service.py::test_add_book_database_error PASSED                                 [ 52%]
tests/test_library_service.py::test_borrow_book_database_error PASSED                              [ 54%]
tests/test_library_service.py::test_calculate_late_fee_maximum_cap PASSED                          [ 56%]
tests/test_library_service.py::test_search_books_invalid_search_type PASSED                        [ 59%]
tests/test_library_service.py::test_pay_late_fees_default_gateway PASSED                           [ 61%]
tests/test_library_service.py::test_pay_late_fees_calculate_fee_returns_none PASSED                [ 63%]
tests/test_library_service.py::test_pay_late_fees_calculate_fee_missing_amount PASSED              [ 65%]
tests/test_payment_mock_stub.py::TestPayLateFees::test_pay_late_fees_success PASSED                [ 68%]
tests/test_payment_mock_stub.py::TestPayLateFees::test_pay_late_fees_declined_by_gateway PASSED    [ 70%]
tests/test_payment_mock_stub.py::TestPayLateFees::test_pay_late_fees_invalid_patron_id_no_gateway_call PASSED [ 72%]
tests/test_payment_mock_stub.py::TestPayLateFees::test_pay_late_fees_zero_fee_no_gateway_call PASSED [ 75%]
tests/test_payment_mock_stub.py::TestPayLateFees::test_pay_late_fees_network_error_handled PASSED  [ 77%]
tests/test_payment_mock_stub.py::TestPayLateFees::test_pay_late_fees_book_not_found PASSED         [ 79%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_success PASSED [ 81%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_invalid_transaction_id PASSED [ 84%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_invalid_amounts[-5.0-greater than 0] PASSED [ 86%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_invalid_amounts[0.0-greater than 0] PASSED [ 88%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_invalid_amounts[20.0-exceeds maximum] PASSED [ 90%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_invalid_amounts[15.01-exceeds maximum] PASSED [ 93%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_gateway_failure PASSED [ 95%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_exception_handling PASSED [ 97%]
tests/test_payment_mock_stub.py::TestRefundLateFeePayment::test_refund_late_fee_payment_maximum_allowed PASSED [100%]

============================== 44 passed in 4.39s ==============================
```



```
C:\Windows\System32\cmd.e

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>pytest --cov=services --cov-report=term tests/
============================= test session starts =============================
platform win32 -- Python 3.12.6, pytest-9.0.0, pluggy-1.6.0
rootdir: C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25
plugins: cov-7.0.0
collected 44 items

tests\test_library_service.py .............................                [ 65%]
tests\test_payment_mock_stub.py ...............                            [100%]

================================ tests coverage ================================
_______________________________ coverage: platform win32, python 3.12.6-final-0 _______________________________

Name                          Stmts   Miss  Cover
-------------------------------------------------
services\library_service.py     140      9    94%
services\payment_service.py      30     22    27%
-------------------------------------------------
TOTAL                           170     31    82%
============================== 44 passed in 2.15s ==============================

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>S
```