Section 1:
Name: Sandy Mourad
Student ID: 20406862
Submission Date: Nov 10th

Section 2: Stubbing VS Mocking explanation (200-300) words: Define stubbing (fake implementations returning hard-coded values) and mocking (test doubles that verify interactions). Explain your strategy: which functions you stubbed, which you mocked, and why.

In testing, stubbing and mocking are two different ways of replacing real dependencies so that we can test a function in isolation. Stubbing replaces a collaborators behaviour with a fixed, deterministic response so the unit under test runs a known state. But, stubs don't make assertions; they only really provide data.
Mocking replaces a collaborator object and also lets us verify interactions like call counts, arguments etc, and force error paths.
In my tests, I used stubs where calculate_late_fee_for_book and get_book_by_id were stubbed using mocker.patch("services.library_services.<fn>", return_value=...) This allowed me to simulate late-fee values and book lookups without touching the actual database. Because these helpers only return data and don't perform behavior, stubs were the appropriate choice.

For my mocks which replaced an entire collaborator object and allowed me to verify how it was used, including how many times it was called, what arguments it received, and how it behaved under different conditions. My PaymentGateway was mocked with Mock(spec=PaymentGateway) to simulate a successful charge, declined charge, exceptions, refund success/failure/exceptions. I asserted both outcomes and where the gateway was or wasn't called with the correct arguments. My overall strategy was, stubbing internal data lookups to isolate logic and keep the tests lightweight, mock the external payment gateway to verify interactions and simulate different response (successful, failure, exceptions) and then finally review coverage results and add direct PaymentGateway tests only where needed to push coverage past 80% (and a few extras). This separation helped my tests stay fast, deterministic, and unit-level while also covering success, failure, and exception branches

Section 3: Test execution instructions:
Environment:
```
python -V
pip install pytest pytest-mock pytest-cov
```
Run tests:
```
python -m pytest -q
```
Run with coverage (thru my terminal and HTML):
```
python -m pytest --cov=services --cov-report=term --cov-report=html
open htmlcov/index.html
```

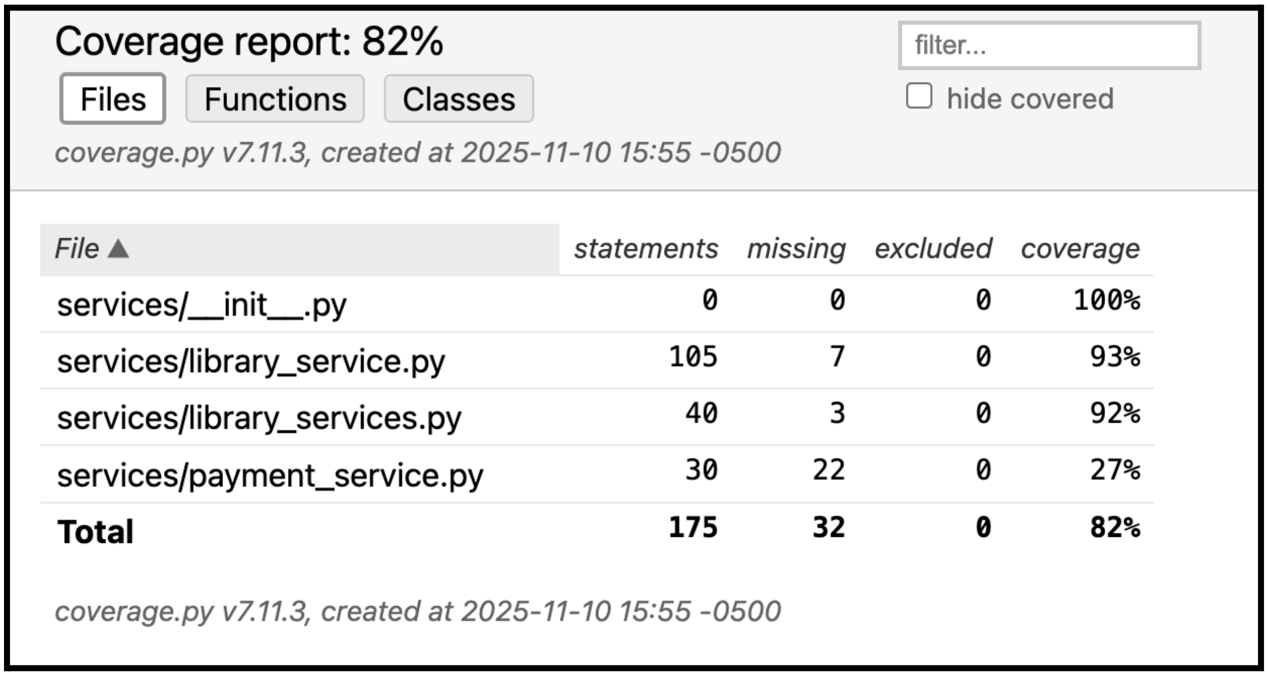
Section 4: Test cases summary for the new tests: table with columns (func name, purpose, stubs used, mocks used, verification done)

Test Function	Purpose	Stubs used	Mocks used	Key verification
`test_pay_late_fees_successful_payment`	To validate successful late-fee payment flow	get_book_by_id, calculate_late_fee _for_book	process_payment	ok=True, correct desc/amt/ID, assert_called_once_with
`test_pay_late_fees_decline_message`	Gateway declines payment	get_book_by_id, calculate_late_fee _for_book	Process_payment which returns false, none, or "card declined"	ok=false, msg contained failed/declined, no txn
`test_invalid_patron_id_skips_gateway`	Input validation stops early	None	Gateway mock present but it shouldn't be called	assert_not_called()
`test_zero_fee_no_gateway_call`	No fees = no gateway call	calculate_late_fee _for_book	None	Early return, assert_not_called()
`test_book_not_found_no_payment`	Missing book = no gateway call	Calculate_late_fee_for_book, get_book_by_id returns none	None	Early return, assert_not_called()
`test_gateway_exception_is_handled`	Simulated network error	Calculate_late_fee_for_book, get_book_by_id	process_payment.side_effect= RuntimeError	Graceful error msg, no txn
`test_refund_successful_case`	Refund successful path	None	Refund_payment returns (True, "...")	ok=true, assert_called_one_with(txn, amount)
`test_refund_invalid_txid_no_call`	Reject bad transaction ids	None	Gateway shouldnt be called	Assert_not_called()
`test_refund_amount_zero_not_allowed`	Amount == 0 is rejected	None	assert_not_called()	Error message includes "greater than 0"
`test_refund_negative_amount_not_allowed`	Amount < 0 is rejected	None	assert_not_called()	Error message includes "greater

				than 0”
`test_refund_amount_too_high_blocked`	Amount > $15 is rejected	None	assert_not_called()	Error mentions exceeds
`test_refund_gateway_failure_message`	Gateway returns failure	None	Refund_payment return false, gateway error	Message includes “refund failed”
`test_refund_gateway_exception_handled`	Refund raises exception	None	refund_payment.side_effect=RuntimeError	Graceful error message
`test_pg_process_payment_success`	Direct PaymentGateway success branch	None	Real object, time.sleep is patched	ok=true, txn format, msg
`test_pg_process_payment_invalid_amount_zero`	Reject amount <= 0	None	Real object, time.sleep is patched	ok=true, empty txn, msg
`test_pg_process_payment_exceeds_limit`	Reject amount > 1000	None	Real object, time.sleep is patched	ok=false, msg “exceeds limit”
`test_pg_refund_payment_success`	Refund success branch	None	Real object, time.sleep is patched	Success message contains formatted amount
`test_pg_verify_payment_status_invalid`	Status check for bad ID	None	Real object, time.sleep is patched	Status == “not_found”

Section 5: Coverage analysis:

Initial: My initial test run produced 82% overall statement coverage (175 total statements, 32 missing) this reflected that most a2 logic was already tested while all A3-related functionality was still untouched, especially the PaymentGateway. Most of my a2 functions were highly covered but my weakest area was the payment_service which contributed to 22/32 missing lines. Every method in PaymentGateway(process_payment, refund_payment, verify_payment_status) was 0% since no prior tests actually invoked those paths. The branches inside also included success vs decline, invalid inputs and exception handling that wasn't actually required until a3.



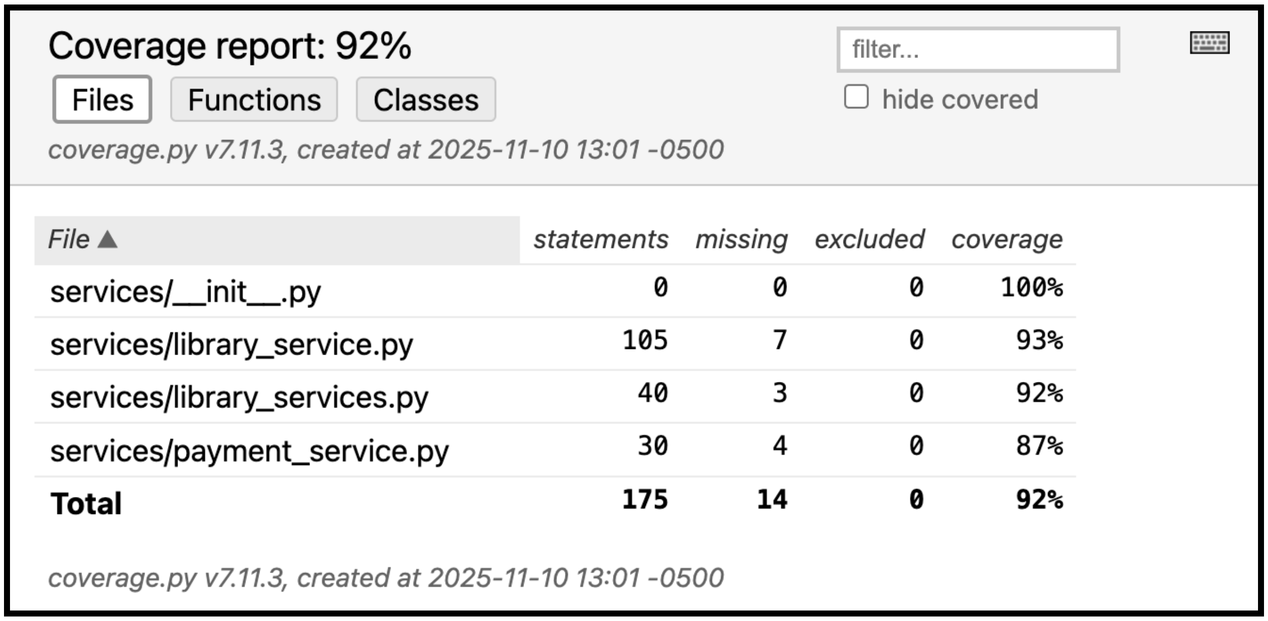
Coverage report: 82%

coverage.py v7.11.3, created at 2025-11-10 15:55 -0500

File ▲	statements	missing	excluded	coverage
services/__init__.py	0	0	0	100%
services/library_service.py	105	7	0	93%
services/library_services.py	40	3	0	92%
services/payment_service.py	30	22	0	27%
Total	**175**	**32**	**0**	**82%**

coverage.py v7.11.3, created at 2025-11-10 15:55 -0500

Final: As for my final coverage, after implementing new targeted tests for both library_services and payment_service, my overall coverage increased to 92% and payment_service rose from 27% to 87%. I added new payment gateway unit tests covering every branch in process_payment(), refund_payment(), verify_payment_status() and patched time.sleep to eliminate delays while still preserving the logic. I also extended sub/mock coverage in library_services so I stubbed calculate_late_fee_for_book and get_book_by_id to simulate DB returns. And I also mocked the PaymentGateway object to simulate charges, declines, and exceptions. Additionally, I made Assertions which validated expected interactions like assert_called_once_with and assert_not_called. Finally I also added edge path coverage for invalid patron IDs, zero fees, and book-not-found cases. Those actually helped ensure the functions handle graceful termination without gateway calls.


Coverage report: 92%

coverage.py v7.11.3, created at 2025-11-10 13:01 -0500

File ▲	statements	missing	excluded	coverage
services/__init__.py	0	0	0	100%
services/library_service.py	105	7	0	93%
services/library_services.py	40	3	0	92%
services/payment_service.py	30	4	0	87%
Total	**175**	**14**	**0**	**92%**

coverage.py v7.11.3, created at 2025-11-10 13:01 -0500

Remaining uncovered branches: process_payment, verify_payment_status, rare fee_amount missing-key branch in pay_late_fees().

Uncovered paths and final summary:
The initial 82% was already alright but PaymentGateway remained untested so I introduced the 5 extra direct gateway tests, this approach ensured that most a3 payment logic was verified without modifying db files or any a2 code paths.

Statement coverage: Initial (82%), Final (92%)
Branch coverage: Initial (-), Final (=90ish%)
PaymentGateway Coverage: Initial (27%), Final (87%)

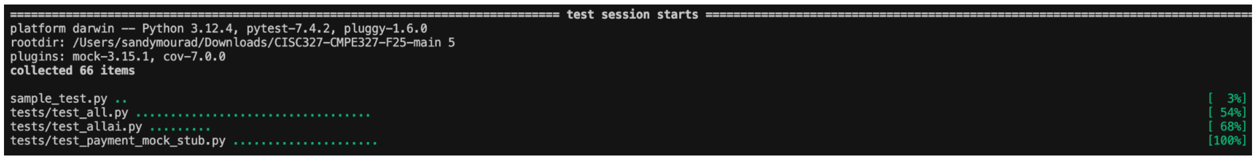
Section 6: Challenges and solution: Describe problems encountered (mock setup issues, coverage difficulties, etc.) and how you solved them. Reflect on what you learned about mocking/stubbing.

During this assignment I encountered several technical issues related to import paths, package discovery and coverage configuration. My first challenge was when I had import errors where I got “ModuleNotFoundError: services) since I was running pytest from a directory that didn't put my project root on sys.path so I then ran it using python -m pytest.
Next, I also initially patched services.library_service.<fn> but the code imported those names into services.library_services so I patched where the names were looked up.
Another issue was missing the requests dependency. Even though I never called it directly during tests, it was imported inside payment_service.py which caused test collection to fail. Installing the requests package fixed that warning.
Lastly i also had an issue with my coverage plugin flag because they weren't being recognized so i installed pytest-cov. And in terms of coverage improvement, the main challenge was raising

payment_service.py from 27% to above 80% so I just added 5 focused unit tests that directly called PaymentGateway methods, these covered success , decline, invalid input, and exception paths, along with verifying mock interactions.
Reflecting on the process , I learned that stubbing worked best for static, predictable data (like fee lookups) while mocking is actually ideal for verifying call behaviour and handling external API's. More importantly, I learned to isolate dependencies cleanly and use coverage feedback iteratively to fill in gaps without touching production logic.

Section 7: screenshots: include (1) all test passing (both positive and negative), (2) coverage terminal output

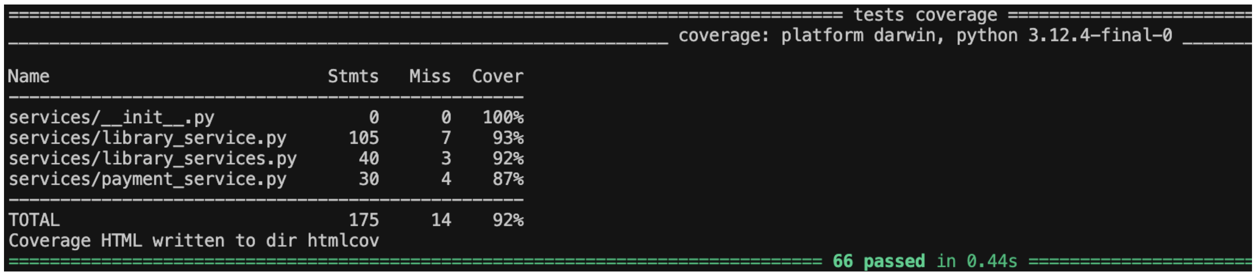
(1)



```
==================================================================== test session starts ====================================================================
platform darwin -- Python 3.12.4, pytest-7.4.2, pluggy-1.6.0
rootdir: /Users/sandymourad/Downloads/CISC327-CMPE327-F25-main 5
plugins: mock-3.15.1, cov-7.0.0
collected 66 items

sample_test.py ..                                                                                                                                   [  3%]
tests/test_all.py ..................................                                                                                                [ 54%]
tests/test_allai.py .........                                                                                                                       [ 68%]
tests/test_payment_mock_stub.py .....................                                                                                               [100%]
```

(2)



```
================================================================================ tests coverage ========================
________________________________________________________________ coverage: platform darwin, python 3.12.4-final-0 ______

Name                              Stmts   Miss  Cover
-----------------------------------------------------
services/__init__.py                  0      0   100%
services/library_service.py         105      7    93%
services/library_services.py         40      3    92%
services/payment_service.py          30      4    87%
-----------------------------------------------------
TOTAL                               175     14    92%
Coverage HTML written to dir htmlcov
============================================================================== 66 passed in 0.44s ======================
```