GitHub Link:

https://github.com/SY64I/cisc327-library-management-a3-6063

Section 1

Arlen Smith, 20386063, November 10th 2025

Section 2

Stubbing and Mocking are both unit testing strategies that allow test cases to be executed without specified functions involved to truly be run. Both these methods are useful if the tester wishes to test a specific function that involves other functions that either reject important data values to test, or perform actions in the code that the tester does not want to execute while testing.

Stubbing in testing refers to when a tester overrides specific functions used by the code they want to test to ensure they return with a specific value. This is used if the tester wishes to test a function using data values that other functions shouldn't be expected to try and verify. In this assignment, I stubbed the `calculate_late_fee_for_book` and `get_book_by_id` functions to force them to return values I needed to test the `pay_late_fees` function. This was done to ensure that I could pass fake data values into both functions and have them return what I needed them to, since normally they would have run into errors trying to verify the fake data values.

Mocking in testing refers to when a tester creates a fake object in the code to ensure it returns specific values if functions within that class are run. This is useful if the tester wishes to test a function requiring a certain class that should return specific values when functions within that class are run. If a tester mocks a class, the mocked class can be forced to return specific values after running functions from it, similar to how stubbing works. In this assignment, I mocked the `PaymentGateway` class while testing the `pay_late_fee` and `refund_late_fee_payment` functions. This was to ensure that the `PaymentGateway` class's `process_payment` and `refund_payment` functions would return specific values without ever establishing a connection to a payment network.

Section 3

To run the tests provided for the first time, the project's files and external packages must first be installed if the user has not already done so by running the commands below in order:

```
python -m venv venv
source venv/bin/activate
```

```
pip install -r requirements.txt
pip install pytest pytest-cov pytest-mock
pip install requests
```

After the user has setup the project, to run the tests and generate a report for the tests, run the following commands:

```
source venv/bin/activate

pytest --cov=services --cov-report=html:tests/coverage_reports
--cov-report=term
```

The test results can viewed either the terminal of where the tests were ran, or through the generated report by navigating to tests/coverage_reports in the user computer's File Explorer (this can be done in VSCode by locating `index.html` in `tests/coverage_reports`, right clicking the html file and clicking 'Reveal in File Explorer'), and opening `index.html`.

Section 4

Function Purpose, Stubs and Mocks used

Test Function Name	**Purpose**	**Stubs Used**	**Mocks Used**
`add_book_to_catalog`	Adds new books to the book catalog.	- get_book_by_isbn - insert_book	N/A
`borrow_book_by_patron`	Handles patron book borrowing.	- get_book_by_id - get_patron_borrowed_books - get_patron_borrow_count - insert_borrow_record - update_book_availability	N/A
`return_book_by_patron`	Handles patron book returning	- get_book_by_id - get_patron_borrowed_books - update_borrow_record_return_date - update_book_availability	N/A
`calculate_late_fee_for_book`	Calculates late fees for books.	- get_book_by_id - get_patron_full_borrow_record	N/A
`search_books_in_catalog`	Searches for books in the catalog.	- get_all_books	N/A
`get_patron_status_report`	Gets status	- get_patron_full_borrow_record	N/A

	reports for patrons.	- get_patron_borrow_count - calculate_late_fee_for_book	
`pay_late_fees`	Processes payments for late fees.	- calculate_late_fee_for_book - get_book_by_id	- PaymentGateway
`refund_late_fee_payment`	Refunds late payment fees if books were wrongfully charged.	N/A	- PaymentGateway

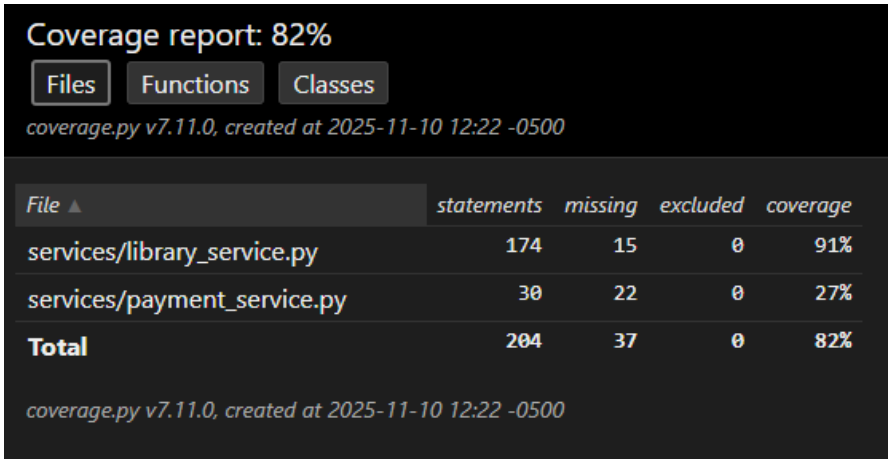
Test Cases per each function:

Test Function Name	**Verification Done**	**List of Test Cases**
`add_book_to_catalog`	Tested for: - Valid input - No title - Too long a title - No author - Too long an author - Incorrect ISBN length - ISBN not digits - Negative copies - Existing ISBN - Database error	- `test_add_valid_input` - `test_add_invalid_no_title` - `test_add_invalid_title_too_long` - `test_add_invalid_no_author` - `test_add_invalid_author_too_long` - `test_add_invalid_isbn_length` - `test_add_invalid_isbn_not_digits` - `test_add_invalid_negative_copies` - `test_add_invalid_existing_isbn` - `test_add_invalid_database_error`
`borrow_book_by_patron`	Tested for: - Valid input - Invalid patron ID - Invalid book ID - No availability - A book already borrowed - Exceeded borrow limit - Borrow record error - Update availability error	- `test_borrow_valid_input` - `test_borrow_invalid_patron` - `test_borrow_invalid_book_id` - `test_borrow_invalid_availability` - `test_borrow_invalid_already_borrowed` - `test_borrow_invalid_borrow_limit` - `test_borrow_invalid_invalid_record_error` - `test_borrow_invalid_availability_error`
`return_book_by_patron`	Tested for: - Valid input - Invalid patron ID - Invalid book ID - A book that's not borrowed - Borrow record error - Update availability error	- `test_return_valid_input` - `test_return_invalid_patron` - `test_return_invalid_book_id` - `test_return_invalid_not_borrowed` - `test_return_invalid_record_error` - `test_return_invalid_availability_error`

`calculate_late_fee_for_book`	Tested for: - Valid input - Valid input with max fee - Valid input without any fee - Invalid patron ID - Invalid book ID - A book that isn't borrowed	- `test_late_fee_valid_input` - `test_late_fee_valid_max_fee` - `test_late_fee_valid_not_overdue` - `test_late_fee_invalid_patron` - `test_late_fee_invalid_book_id` - `test_late_fee_invalid_not_borrowed`
`search_books_in_catalog`	Tested for: - Valid title search - Valid author search - Valid ISBN search - Invalid ISBN search - Invalid patron ID	- `test_search_valid_title` - `test_search_valid_author` - `test_search_valid_isbn` - `test_search_invalid_isbn` - `test_search_invalid_search`
`get_patron_status_report`	Tested for: - Valid report with no books - Valid report with one book - Valid report with multiple books - Valid report with a book that was returned late - Invalid patron ID	- `test_report_valid_no_books` - `test_report_valid_one_book` - `test_report_valid_multiple_books` - `test_report_valid_returned_late_book` - `test_report_invalid_patron`
`pay_late_fees`	Tested for: - Valid input - Invalid patron ID - Late fee calculation error - No late fee - Invalid book ID - Payment failure - Payment processing error	- `test_payment_valid` - `test_payment_invalid_patron` - `test_payment_invalid_calculation_error` - `test_payment_invalid_no_late_fee` - `test_payment_invalid_book_id` - `test_payment_invalid_failed_payment` - `test_payment_invalid_processing_error`
`refund_late_fee_payment`	Tested for: - Valid input - Invalid transaction ID - Negative refund amount - Zero refund amount - Refund amount above max - Refund failure - Refund processing error	- `test_refund_valid` - `test_refund_invalid_transaction_id` - `test_refund_invalid_amount_negative` - `test_refund_invalid_amount_zero` - `test_refund_invalid_amount_exceed_max` - `test_refund_invalid_failed_refund` - `test_refund_invalid_processing_error`

Section 5

Initial Test Coverage Screenshots (taken after completing Task 2.1 with test cases created only for the specified required test scenarios):



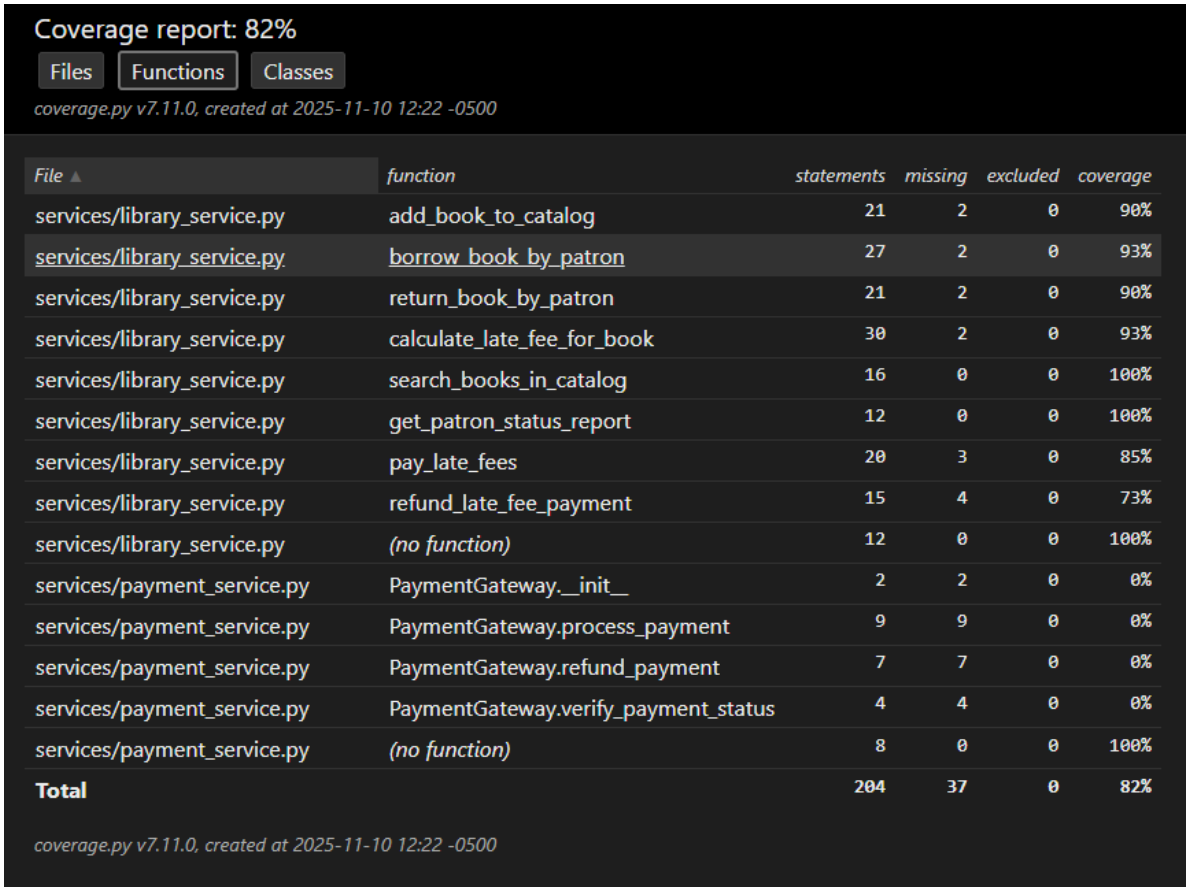
Coverage report: 82%

Files | Functions | Classes

coverage.py v7.11.0, created at 2025-11-10 12:22 -0500

File ▲	statements	missing	excluded	coverage
services/library_service.py	174	15	0	91%
services/payment_service.py	30	22	0	27%
Total	204	37	0	82%

coverage.py v7.11.0, created at 2025-11-10 12:22 -0500



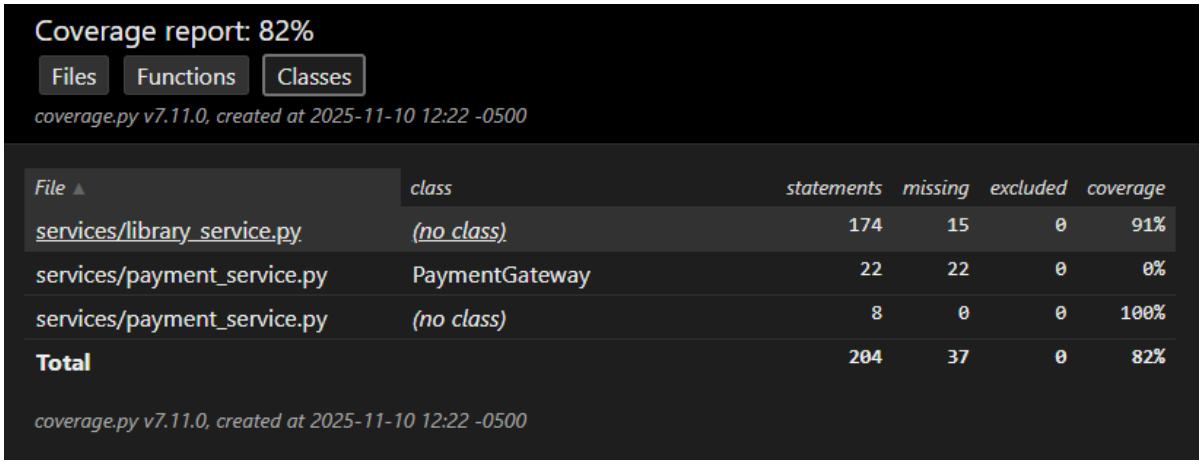
Coverage report: 82%

Files | Functions | Classes

coverage.py v7.11.0, created at 2025-11-10 12:22 -0500

File ▲	function	statements	missing	excluded	coverage
services/library_service.py	add_book_to_catalog	21	2	0	90%
services/library_service.py	borrow_book_by_patron	27	2	0	93%
services/library_service.py	return_book_by_patron	21	2	0	90%
services/library_service.py	calculate_late_fee_for_book	30	2	0	93%
services/library_service.py	search_books_in_catalog	16	0	0	100%
services/library_service.py	get_patron_status_report	12	0	0	100%
services/library_service.py	pay_late_fees	20	3	0	85%
services/library_service.py	refund_late_fee_payment	15	4	0	73%
services/library_service.py	*(no function)*	12	0	0	100%
services/payment_service.py	PaymentGateway.__init__	2	2	0	0%
services/payment_service.py	PaymentGateway.process_payment	9	9	0	0%
services/payment_service.py	PaymentGateway.refund_payment	7	7	0	0%
services/payment_service.py	PaymentGateway.verify_payment_status	4	4	0	0%
services/payment_service.py	*(no function)*	8	0	0	100%
Total		204	37	0	82%

coverage.py v7.11.0, created at 2025-11-10 12:22 -0500



Coverage report: 82%

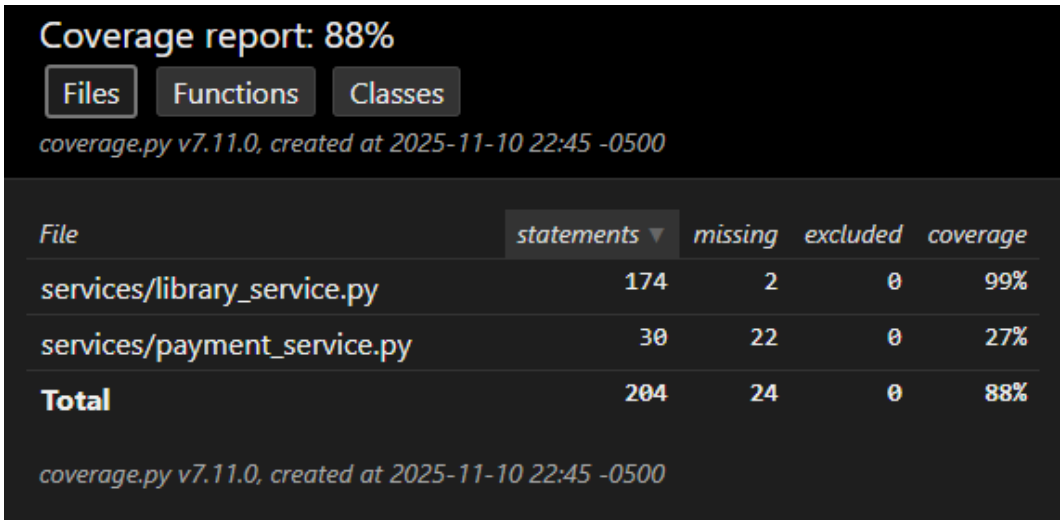
Files | Functions | Classes

coverage.py v7.11.0, created at 2025-11-10 12:22 -0500

File ▲	class	statements	missing	excluded	coverage
services/library_service.py	*(no class)*	174	15	0	91%
services/payment_service.py	PaymentGateway	22	22	0	0%
services/payment_service.py	*(no class)*	8	0	0	100%
Total		204	37	0	82%

coverage.py v7.11.0, created at 2025-11-10 12:22 -0500

Final Test Coverage Screenshots (taken after completing Task 2.2):



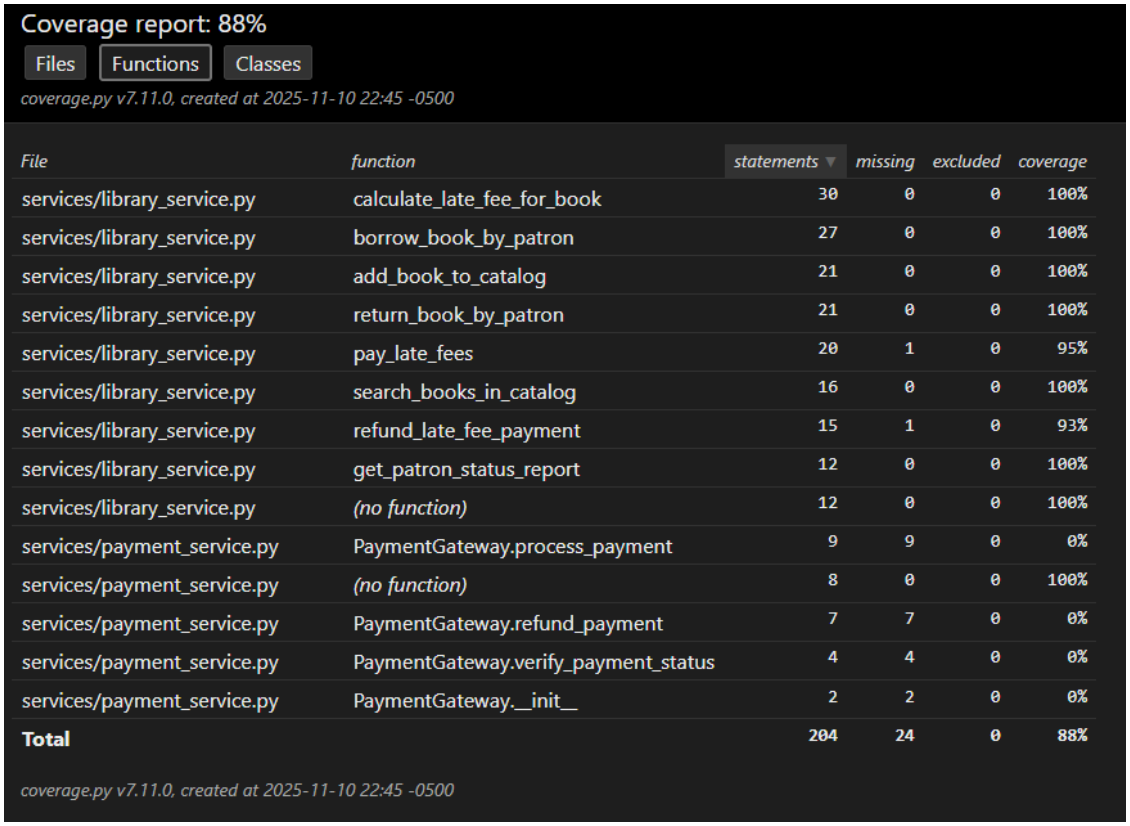
Coverage report: 88%

Files | Functions | Classes

coverage.py v7.11.0, created at 2025-11-10 22:45 -0500

File	statements ▼	missing	excluded	coverage
services/library_service.py	174	2	0	99%
services/payment_service.py	30	22	0	27%
Total	204	24	0	88%

coverage.py v7.11.0, created at 2025-11-10 22:45 -0500



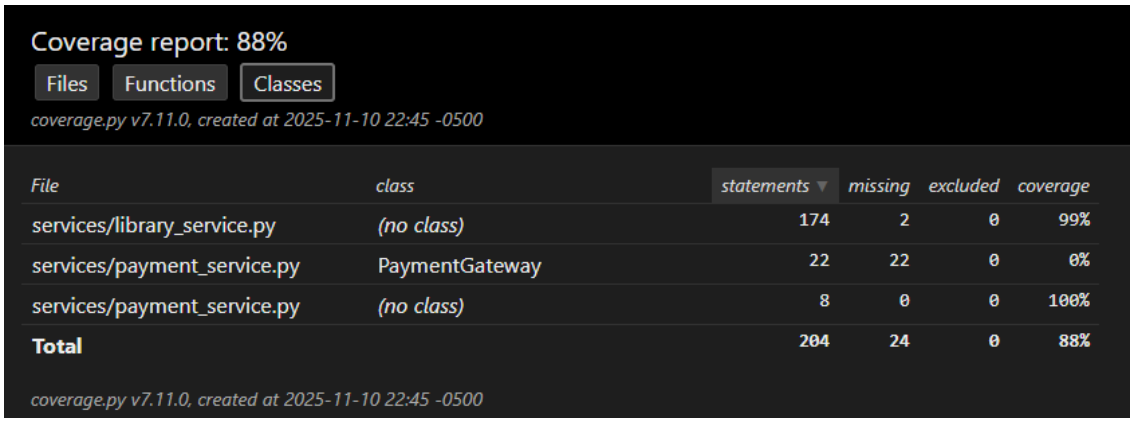
Coverage report: 88%

Files | Functions | Classes

coverage.py v7.11.0, created at 2025-11-10 22:45 -0500

File	function	statements ▼	missing	excluded	coverage
services/library_service.py	calculate_late_fee_for_book	30	0	0	100%
services/library_service.py	borrow_book_by_patron	27	0	0	100%
services/library_service.py	add_book_to_catalog	21	0	0	100%
services/library_service.py	return_book_by_patron	21	0	0	100%
services/library_service.py	pay_late_fees	20	1	0	95%
services/library_service.py	search_books_in_catalog	16	0	0	100%
services/library_service.py	refund_late_fee_payment	15	1	0	93%
services/library_service.py	get_patron_status_report	12	0	0	100%
services/library_service.py	*(no function)*	12	0	0	100%
services/payment_service.py	PaymentGateway.process_payment	9	9	0	0%
services/payment_service.py	*(no function)*	8	0	0	100%
services/payment_service.py	PaymentGateway.refund_payment	7	7	0	0%
services/payment_service.py	PaymentGateway.verify_payment_status	4	4	0	0%
services/payment_service.py	PaymentGateway.__init__	2	2	0	0%
Total		**204**	**24**	**0**	**88%**

coverage.py v7.11.0, created at 2025-11-10 22:45 -0500



Coverage report: 88%

Files | Functions | Classes

coverage.py v7.11.0, created at 2025-11-10 22:45 -0500

File	class	statements ▼	missing	excluded	coverage
services/library_service.py	*(no class)*	174	2	0	99%
services/payment_service.py	PaymentGateway	22	22	0	0%
services/payment_service.py	*(no class)*	8	0	0	100%
Total		**204**	**24**	**0**	**88%**

coverage.py v7.11.0, created at 2025-11-10 22:45 -0500

Initially, every instance of a `library_service` function did not have any coverage on paths of code that would stem from database errors. This was because I wasn't aware of stubbing and mocking previously, so I had no way of testing for if a database function were to fail. Additionally, there were a few code paths in the `calculate_late_fee_for_book` that I had overlooked in previous testing. I implemented tests that would both go through the database error paths with stubbing and created test cases for the paths in `calculate_late_fee_for_book` I had overlooked. Additionally, I rewrote almost every test case I had to implement stubbing when possible to ensure that the tests never use the database API in any way, since previously they had used the API directly for testing. My statement (and branch) coverage percentages ended

up being 99% for `library_service`, and 27% for `payment_service` since I couldn't create test cases for `payment_service` even with stubbing or mocking. The only 2 statements/branches I could not cover in library_service were 2 similar if statements checking if the specified PaymentGateway was None in both pay_late_fees and refund_late_fee_payment. This was because the if statements would generate their own PaymentGateway classes if they were true, which I could not let happen since that would connect to a real payment service.

Section 6

When first learning how to use mocking and stubbing, I first ran into issues with stubbing where I couldn't stub specific functions from `library_service` that were imported from `database`. Initially, I had assumed that in order to stub a function using `pytest-mock`'s `mocker.patch()` function, I had to reference the library that the function came from. This led me to trying to use `mocker.patch(services.database.get_book_by_id, …)` as an example. However, I figured out that mocker.patch() actually checks where the function was called from, not from what script it was declared from. This meant I never had to use a database to stub the functions, rather I used library_service for all of them. I ended up using the format of `mocker.patch(services.library_service.get_book_by_id, …)` as an example.

I also ran into the issue of trying to figure out the difference between stubbing and mocking at first. They initially looked to be very similar concepts, and this ended up being because Mocking directly involves stubbing. I ended up with the consensus that mocking was used for classes and that stubbing was used for function return values, and that led me to understanding the differences between the two much better. I also learned to see how useful stubbing in particular was for most of the library_service functions. Before this assignment, I had to write custom functions in `database` and use existing `library_service` functions to test most of the functions I needed to test in `library_service`. But with stubbing, I could test every function without requiring any other additional `database` or `library_service` functions to be run.

Section 7

```
sample_test.py ..
tests/test_R1.py ..........
tests/test_R3.py ........
tests/test_R4.py ......
tests/test_R5.py ......
tests/test_R6.py .....
tests/test_R7.py .....
tests/test_paylate.py .......
tests/test_refundlate.py .......
```

```
Name                            Stmts   Miss  Cover
-------------------------------------------------
services/library_service.py       174      2    99%
services/payment_service.py        30     22    27%
-------------------------------------------------
TOTAL                             204     24    88%
Coverage HTML written to dir tests/coverage_reports
```