

CMPE 327 Assignment 2

Lucas Srigley

Student ID: 20289448

GitHub link: <https://github.com/lucas-srigley/cisc327-library-management-a2-9448>

Table of Implementation Status

Requirement	Function Name	Implementation Status	Missing
R1	add_book_to_catalog()	Complete	Nothing
R2	get_all_books (database function)	Complete	Nothing
R3	borrow_book_by_patron()	Complete	Nothing
R4	return_book_by_patron()	Complete	Nothing
R5	calculate_late_fee_for_book()	Complete	Nothing
R6	search_books_in_catalog()	Complete	Nothing
R7	get_patron_status_report()	Complete	Nothing

Complete Function Implementation

All functions have been implemented. In R4, the function allows the user to return a book and includes validation checks for invalid patron or book IDs. In R5, the function calculates the late fee for a patron and validates the provided patron and book IDs. In R6, the function allows searching for books, validates empty search terms and search types, and ensures that the search type is either title, author, or ISBN. In R7, the function retrieves the patron status report and displays it in its own “Patron Status” tab, with validation for invalid patron IDs.

Comprehensive Test Suite Development

For most test cases (R1-R6), I had already written complete test cases. There were some cases (R4-R6) that would say “assert success is False” even though there was a valid input because they weren’t implemented in Assignment 1. Now, the valid input tests are successful because the functions are implemented so I changed it to “assert success is True”. For R7, I had most of the test cases, but I added some additional functionality with the implemented function so there were some additional variable asserts.

AI-Assisted Test Generation

LLM Model: Microsoft 365 Copilot

Prompt

Based on the attached requirements_specification.md, library_service.py, and database.py files, create individual test suites for R1-R7 in pytest. This is the format the tests should follow:

```
import pytest
from library_service import (
    add_book_to_catalog
)
```

```
def test_add_book_valid_input():
    """Test adding a book with valid input."""
    success, message = add_book_to_catalog("Test Book", "Test Author", "1234567890123", 5)

    assert success == True
    assert "successfully added" in message.lower()
```

Follow-up Prompts

- There should be at least 4-5 test cases per requirement, including positive and negative test cases. (It only generated 1 or 2 cases for each)

Outcome

```
import pytest
from datetime import datetime, timedelta
from library_service import (
    add_book_to_catalog,
    borrow_book_by_patron,
    return_book_by_patron,
    calculate_late_fee_for_book,
    search_books_in_catalog,
    get_patron_status_report
)
from database import init_database, reset_db, insert_book, insert_borrow_record

# Setup
init_database()
reset_db()

# R1: Add Book To Catalog
def test_add_book_valid():
    success, message = add_book_to_catalog("Valid Book", "Author", "1234567890123", 3)
    assert success
    assert "successfully added" in message.lower()

def test_add_book_missing_title():
    success, message = add_book_to_catalog("", "Author", "1234567890124", 2)
    assert not success
    assert "title is required" in message.lower()

def test_add_book_long_title():
    long_title = "A" * 201
    success, message = add_book_to_catalog(long_title, "Author", "1234567890125", 2)
    assert not success
    assert "title must be less than 200 characters" in message.lower()

def test_add_book_invalid_isbn_length():
    success, message = add_book_to_catalog("Book", "Author", "12345", 2)
```

```

    assert not success
    assert "isbn must be exactly 13 digits" in message.lower()

def test_add_book_duplicate_isbn():
    add_book_to_catalog("Book1", "Author1", "1234567890126", 2)
    success, message = add_book_to_catalog("Book2", "Author2", "1234567890126", 2)
    assert not success
    assert "already exists" in message.lower()

# R3: Book Borrowing Interface
def test_borrow_book_valid():
    add_book_to_catalog("Borrowable", "Author", "1234567890127", 1)
    book = search_books_in_catalog("borrowable", "title")[0]
    success, message = borrow_book_by_patron("111111", book["id"])
    assert success
    assert "successfully borrowed" in message.lower()

def test_borrow_book_invalid_patron_id():
    book = search_books_in_catalog("borrowable", "title")[0]
    success, message = borrow_book_by_patron("abc", book["id"])
    assert not success
    assert "invalid patron id" in message.lower()

def test_borrow_book_unavailable():
    add_book_to_catalog("Unavailable", "Author", "1234567890128", 1)
    book = search_books_in_catalog("unavailable", "title")[0]
    borrow_book_by_patron("111112", book["id"])
    success, message = borrow_book_by_patron("111113", book["id"])
    assert not success
    assert "not available" in message.lower()

def test_borrow_book_limit_exceeded():
    for i in range(6):
        add_book_to_catalog(f"Book{i}", "Author", f"12345678901{i+30}", 1)
        book = search_books_in_catalog(f"book{i}", "title")[0]
        borrow_book_by_patron("222222", book["id"])
    book = search_books_in_catalog("book5", "title")[0]
    success, message = borrow_book_by_patron("222222", book["id"])
    assert not success
    assert "maximum borrowing limit" in message.lower()

def test_borrow_book_invalid_book_id():
    success, message = borrow_book_by_patron("111111", 9999)
    assert not success
    assert "book not found" in message.lower()

# R4: Book Return Processing
def test_return_book_valid():

```

```

add_book_to_catalog("Returnable", "Author", "1234567890130", 1)
book = search_books_in_catalog("returnable", "title")[0]
borrow_book_by_patron("333333", book["id"])
success, message = return_book_by_patron("333333", book["id"])
assert success
assert "successfully returned" in message.lower()

def test_return_book_not_borrowed():
    add_book_to_catalog("NotBorrowed", "Author", "1234567890131", 1)
    book = search_books_in_catalog("notborrowed", "title")[0]
    success, message = return_book_by_patron("333333", book["id"])
    assert not success
    assert "not been borrowed" in message.lower()

def test_return_book_invalid_patron_id():
    book = search_books_in_catalog("returnable", "title")[0]
    success, message = return_book_by_patron("abc", book["id"])
    assert not success
    assert "invalid patron id" in message.lower()

def test_return_book_invalid_book_id():
    success, message = return_book_by_patron("333333", 9999)
    assert not success
    assert "book not found" in message.lower()

# R5: Late Fee Calculation
def test_late_fee_no_overdue():
    add_book_to_catalog("FeeBook", "Author", "1234567890132", 1)
    book = search_books_in_catalog("feebook", "title")[0]
    insert_borrow_record("444444", book["id"], datetime.now(), datetime.now() + timedelta(days=14))
    fee_info = calculate_late_fee_for_book("444444", book["id"])
    assert fee_info["fee_amount"] == 0.00
    assert fee_info["days_overdue"] == 0

def test_late_fee_5_days():
    add_book_to_catalog("Late5", "Author", "1234567890133", 1)
    book = search_books_in_catalog("late5", "title")[0]
    insert_borrow_record("555555", book["id"], datetime.now() - timedelta(days=19), datetime.now() - timedelta(days=5))
    fee_info = calculate_late_fee_for_book("555555", book["id"])
    assert fee_info["fee_amount"] == 2.5
    assert fee_info["days_overdue"] == 5

def test_late_fee_10_days():
    add_book_to_catalog("Late10", "Author", "1234567890134", 1)
    book = search_books_in_catalog("late10", "title")[0]
    insert_borrow_record("666666", book["id"], datetime.now() - timedelta(days=24), datetime.now() - timedelta(days=10))
    fee_info = calculate_late_fee_for_book("666666", book["id"])
    assert fee_info["fee_amount"] == 6.5

```

```

    assert fee_info["days_overdue"] == 10

def test_late_fee_max_cap():
    add_book_to_catalog("LateMax", "Author", "1234567890135", 1)
    book = search_books_in_catalog("latemax", "title")[0]
    insert_borrow_record("777777", book["id"], datetime.now() - timedelta(days=40), datetime.now() - timedelta(days=26))
    fee_info = calculate_late_fee_for_book("777777", book["id"])
    assert fee_info["fee_amount"] == 15.00
    assert fee_info["days_overdue"] == 26

def test_late_fee_invalid_book():
    fee_info = calculate_late_fee_for_book("888888", 9999)
    assert fee_info["fee_amount"] == 0.00
    assert "book not found" in fee_info["status"].lower()

# R6: Book Search Functionality
def test_search_by_title():
    results = search_books_in_catalog("gatsby", "title")
    assert any("gatsby" in book["title"].lower() for book in results)

def test_search_by_author():
    results = search_books_in_catalog("fitzgerald", "author")
    assert any("fitzgerald" in book["author"].lower() for book in results)

def test_search_by_isbn():
    results = search_books_in_catalog("9780743273565", "isbn")
    assert any(book["isbn"] == "9780743273565" for book in results)

def test_search_invalid_type():
    results = search_books_in_catalog("gatsby", "genre")
    assert results == []

def test_search_no_match():
    results = search_books_in_catalog("nonexistentbook", "title")
    assert results == []

# R7: Patron Status Report
def test_patron_status_valid():
    report = get_patron_status_report("123456")
    assert report["success"]
    assert "currently_borrowed" in report

def test_patron_status_invalid_id():
    report = get_patron_status_report("abc")
    assert not report["success"]
    assert "invalid patron id" in report["message"].lower()

def test_patron_status_no_books():

```

```

report = get_patron_status_report("999999")
assert report["success"]
assert report["num_books_borrowed"] == 0

def test_patron_status_with_late_fees():
    add_book_to_catalog("LateStatus", "Author", "1234567890136", 1)
    book = search_books_in_catalog("latestatus", "title")[0]
    insert_borrow_record("888888", book["id"], datetime.now() - timedelta(days=20), datetime.now() - timedelta(days=6))
    report = get_patron_status_report("888888")
    assert report["success"]
    assert report["total_late_fees"] > 0

def test_patron_status_history():
    report = get_patron_status_report("123456")
    assert "borrowing_history" in report
    assert isinstance(report["borrowing_history"], list)

```

Test-Case Comparison & Analysis

Comparing the Copilot-generated tests to my own, I found that my tests are more thorough and better organized. My test files focus on specific edge cases like empty inputs, overly long strings, and invalid data types. I verified the exact error messages that should appear. For instance, I check for precise messages like "Title is required." rather than just looking for keywords somewhere in the response. This level of detail helps my tests catch more bugs and ensures the system works exactly as intended. My req1_test.py includes 13 separate validation tests, while the AI version only provided around 5 tests for the same requirement with looser assertions using "in message.lower()" (although I did only request 4-5 test cases). I also organized my tests by requirement into separate files with clear naming conventions, making it much easier to understand what each test covers. The AI combined everything into a single file, which is less organized. That said, the AI-generated tests did show some efficiency, like using loops to test exceeding borrowing limits instead of repeating the same code multiple times like I did. Overall, while AI tools can be helpful for generating a starting point, my manual tests cover all the edge cases more comprehensively. I trust my own testing approach more since AI-generated code needs to be carefully reviewed and validated.