

GitHub Link:

<https://github.com/SY64I/cisc327-library-management-a2-6063>

Part 1:

Implementing the previously un-implemented functions were straightforward for the most part.

- My implementation for the `return_book_by_patron` function was relatively similar to the implementation for the `borrow_book_by_patron` and made use of several pre-implemented `database.py` functions that weren't previously used.
- My implementation for `calculate_late_fee_for_book` required me to create my own custom `database.py` function called `get_patron_full_borrow_record`, a function which retrieves the title, author, book id, borrow date, due date and return date for all currently and previously borrowed books from a patron. This function made it easy for me to retrieve the borrow, due and return dates of a patron's book to calculate its fee.
- My implementation for `search_books_in_catalog` mainly consists of linear searching for any books that match. either through partial match or by exact match depending on what format is specified.

Implementing `get_patron_status_report` was the only implementation that wasn't as straightforward. I originally was confused about how I would implement 'borrow history' while simply returning a dictionary. However, I realised that the database for `borrow_records` also contained whether a book was returned or not through its `return_date` parameter. So I reused my custom `get_patron_full_borrow_record` `database.py` function that I used previously to check for whether a book was returned or not, and used the existing `get_patron_borrow_count` `database.py` function to check how many of those books were currently borrowed. Throughout I realised that I had to change my pre-existing test cases for this function due to me changing my perception of what was going to be returned from the function (more information about that in Part 2).

Below is an updated version of my table from Assignment 1 to reflect to current implementation status of each function:

Function Name	Implementation Status (Complete/Partial/ Not Yet)	What Is Missing

	Implemented)	
add_book_to_catalog	Partial	<p>Note: All the points below will make the add_book page return with an error, but add_book_to_catalog will return false as it should, so they do not violate R1.</p> <ul style="list-style-type: none"> - The ISBN number field produces an error on the website if you try to submit a book with only 1-12 digits in the field. - The Total Copies field produces an error on the website if you try to submit with a floating point number with a fractional part of 0 (ex. 1.0, 5.0, etc). - The Total Copies field will produce a database error on the website if you try to submit with a very large integer value (since there is no limit to how large the number can be).
borrow_book_by_patron	Complete	
return_book_by_patron	Complete	
calculate_late_fee_for_book	Complete	
search_books_in_catalog	Partial	<ul style="list-style-type: none"> - On the website, if you don't find any search results, the website gives an error saying that 'Search functionality has not been implemented' when it should have different error messages.
get_patron_status_report	Partial	<ul style="list-style-type: none"> - There currently is no place to test this

		function's functionality on the website itself.
--	--	---

Part 2:

The only notable changes I've made to my test scripts since Assignment 1 is that I've changed the naming convention of each script from 'R#_test' to 'test_R#' (for github integration), and that some adjustments have been made to test_R3.py and test_R7.py.

test_R3.py was adjusted since the requirement R3 was slightly adjusted to limit patrons from borrowing multiple copies of the same book.

- `test_borrow_book_invalid_borrow_limit` was reworked to test for a patron's borrow limit using a different method. Previously it had a patron attempt to borrow 6 copies of the same book, now the test function has the patron attempt to borrow 6 different books to test for borrowing limits properly.
- A new test case called `test_borrow_book_invalid_already_borrowed` has been added to test for a patron trying to borrow 2 copies of the same book, which should no longer be possible.

test_R7.py was adjusted since what I expected to return from my `get_patron_status_report` function changed after I actually implemented it.

- The existing tests `test_patron_report_valid_one_book` and `test_patron_report_valid_two_books` were updated to search for books within a patron report through book id rather than through ISBN due to the expected output of `get_patron_status_report` being changed. The implementation of searching through each report was also slightly updated to search for values within each dictionary properly (I couldn't check that my test cases worked properly before since the `get_patron_status_report` function wasn't implemented before).
- Additionally, a new test case called `test_patron_report_valid_returned_late_book` was added to test creating a valid patron report where the patron has returned a book (this mainly checks to ensure the currently borrowed book count is not affected by the number of books within a patron's full report).

Part 3:

Note for this section:

Anything in the Arial font is my own report,

Anything in the Times New Roman font is either what I gave to the AI or what the AI gave back to me.

For this section, I used LibreChat to generate test cases. Below is the initial prompt I used to ask it to generate cases:

Hi, I'm looking to generate some test cases for Library Management Software I'm developing for a course. Below are the functional requirements I'm supposed to follow:

R1: Add Book To Catalog

The system shall provide a web interface to add new books to the catalog via a form with the following fields:

- Title (required, max 200 characters)
- Author (required, max 100 characters)
- ISBN (required, exactly 13 digits)
- Total copies (required, positive integer)
- The system shall display success/error messages and redirect to the catalog view after successful addition.

R2: Book Catalog Display

The system shall display all books in the catalog in a table format showing:

- Book ID, Title, Author, ISBN
- Available copies / Total copies
- Actions (Borrow button for available books)

R3: Book Borrowing Interface

The system shall provide a borrowing interface to borrow books by patron ID:

- Accepts patron ID and book ID as the form parameters
- Validates patron ID (6-digit format)
- Checks book availability and patron borrowing limits (max 5 books)
- Ensures the patron only borrows one copy of a unique book at a time
- Creates borrowing record and updates available copies
- Displays appropriate success/error messages

R4: Book Return Processing

The system shall provide a return interface that includes:

- Accepts patron ID and book ID as form parameters
- Verifies the book was borrowed by the patron
- Updates available copies and records return date
- Calculates and displays any late fees owed

R5: Late Fee Calculation API

The system shall provide an API endpoint GET /api/late_fee/<patron_id>/<book_id> that includes the following.

Calculates late fees for overdue books based on:

- Books due 14 days after borrowing
- \$0.50/day for first 7 days overdue
- \$1.00/day for each additional day after 7 days
- Maximum \$15.00 per book

Returns JSON response with fee amount and days overdue

R6: Book Search Functionality

The system shall provide search functionality with the following parameters:

- q: search term
- type: search type (title, author, isbn)
- Support partial matching for title/author (case-insensitive)
- Support exact matching for ISBN
- Return results in same format as catalog display
-

R7: Patron Status Report

The system shall display patron status for a particular patron that includes the following:

- Currently borrowed books with due dates
- Total late fees owed
- Number of books currently borrowed
- Borrowing history

Can you generate me 4-8 test cases for each requirement specified above, to be tested using Python's pytest framework?

LibreChat first made some assumptions about the software before generating cases:

Notes:

I assume you have some kind of service layer or API client methods to call like add_book(), get_catalog(), borrow_book(), return_book(), get_late_fee(), search_books(), get_patron_status().

You may need to mock data store or API calls for unit tests or run these as integration tests.

Error checks usually verify error messages or HTTP response status/code.

Below are the test cases LibreChat returned to me:

For R1:

```
import pytest
```

```
# Example function under test: add_book(title, author, isbn, total_copies) -> (success:bool, message:str)
```

```

def test_add_book_success():
    success, msg = add_book("Clean Code", "Robert C. Martin", "1234567890123", 5)
    assert success
    assert "success" in msg.lower()

def test_add_book_missing_title():
    success, msg = add_book("", "Robert C. Martin", "1234567890123", 5)
    assert not success
    assert "title" in msg.lower()

def test_add_book_title_length_exceeded():
    long_title = "A" * 201
    success, msg = add_book(long_title, "Author", "1234567890123", 2)
    assert not success
    assert "title" in msg.lower()

def test_add_book_author_length_exceeded():
    long_author = "A" * 101
    success, msg = add_book("Title", long_author, "1234567890123", 2)
    assert not success
    assert "author" in msg.lower()

def test_add_book_invalid_isbn_length():
    success, msg = add_book("Title", "Author", "1234567890", 2) # 10 digits, not 13
    assert not success
    assert "isbn" in msg.lower()

def test_add_book_non_digit_isbn():
    success, msg = add_book("Title", "Author", "12345ABC90123", 2)
    assert not success
    assert "isbn" in msg.lower()

def test_add_book_negative_copies():
    success, msg = add_book("Title", "Author", "1234567890123", -1)
    assert not success
    assert "copies" in msg.lower()

def test_add_book_zero_copies():
    success, msg = add_book("Title", "Author", "1234567890123", 0)
    assert not success
    assert "copies" in msg.lower()

```

For R2:

```

def test_catalog_display_includes_added_book():
    # Assuming adding a book first
    add_book("The Pragmatic Programmer", "Andrew Hunt", "9876543210123", 3)
    books = get_catalog()
    assert any(book['title'] == "The Pragmatic Programmer" for book in books)

def test_catalog_display_fields():
    books = get_catalog()
    for book in books:
        assert 'book_id' in book
        assert 'title' in book
        assert 'author' in book
        assert 'isbn' in book
        assert 'available_copies' in book
        assert 'total_copies' in book
        assert 'actions' in book

def test_catalog_actions_borrow_button_present_if_available():
    # Assuming book with available copies > 0
    add_book("Refactoring", "Martin Fowler", "1234987654321", 2)
    books = get_catalog()
    for book in books:
        if book['available_copies'] > 0:
            assert 'borrow' in book['actions'].lower()

def test_catalog_actions_borrow_button_absent_if_none_available():
    # Add book with no available copies (simulate fully borrowed)
    add_book("Unavailable Book", "Author", "111111111111", 1)
    # Simulate it's borrowed out - you may need an internal method or fixture to set available to 0
    set_book_availability(book_id_for("Unavailable Book"), 0)
    books = get_catalog()
    for book in books:
        if book['title'] == "Unavailable Book":
            assert 'borrow' not in book['actions'].lower()

```

For R3:

```

def test_borrow_book_success():
    add_book("Effective Python", "Brett Slatkin", "222222222222", 3)
    patron_id = "123456"
    book_id = get_book_id_by_isbn("222222222222")
    success, msg = borrow_book(patron_id, book_id)
    assert success

```

```

assert "success" in msg.lower()

def test_borrow_book_invalid_patron_id_format():
    add_book("Book", "Author", "333333333333", 1)
    book_id = get_book_id_by_isbn("333333333333")
    # Patron ID not 6-digit
    success, msg = borrow_book("abc123", book_id)
    assert not success
    assert "patron id" in msg.lower()

def test_borrow_book_no_copies_available():
    add_book("No Copies Book", "Author", "444444444444", 1)
    book_id = get_book_id_by_isbn("444444444444")
    # Borrow it all copies to 0
    patron_id1 = "111111"
    borrow_book(patron_id1, book_id)
    # Now try borrowing again by other patron
    success, msg = borrow_book("222222", book_id)
    assert not success
    assert "available" in msg.lower()

def test_borrow_book_patron_borrow_limit_reached():
    patron_id = "555555"
    # Add 6 books, try to borrow 6th (limit is 5)
    for i in range(6):
        isbn = f'555555555555{i}'
        add_book(f'Book{i}', "Author", isbn, 1)
        book_id = get_book_id_by_isbn(isbn)

        if i < 5:
            borrow_book(patron_id, book_id)
        else:
            success, msg = borrow_book(patron_id, book_id)
            assert not success
            assert "limit" in msg.lower()

def test_borrow_book_duplicate_borrow_of_same_book():
    patron_id = "666666"
    add_book("Duplicate Borrow", "Author", "666666666666", 2)
    book_id = get_book_id_by_isbn("666666666666")
    borrow_book(patron_id, book_id) # first borrow
    success, msg = borrow_book(patron_id, book_id) # second borrow attempt same book
    assert not success
    assert "already borrowed" in msg.lower()

```

```

def test_borrow_book_invalid_book_id():
    success, msg = borrow_book("123456", 999999) # non-existent book ID
    assert not success
    assert "book" in msg.lower()

```

For R4:

```

def test_return_book_success():
    patron_id = "777777"
    add_book("Return Test Book", "Author", "77777777777777", 1)
    book_id = get_book_id_by_isbn("77777777777777")
    borrow_book(patron_id, book_id)

    success, msg = return_book(patron_id, book_id)
    assert success
    assert "success" in msg.lower()

def test_return_book_not_borrowed_by_patron():
    patron_id1 = "888888"
    patron_id2 = "999999"
    add_book("Return Test Book 2", "Author", "888888888888", 1)
    book_id = get_book_id_by_isbn("888888888888")
    borrow_book(patron_id1, book_id)

    success, msg = return_book(patron_id2, book_id) # different patron
    assert not success
    assert "not borrowed" in msg.lower()

def test_return_book_updates_available_copies():
    patron_id = "123456"
    add_book("Copies Update Book", "Author", "1231231231231", 1)
    book_id = get_book_id_by_isbn("1231231231231")
    borrow_book(patron_id, book_id)

    # Check availability before return
    book_info = get_book_info(book_id)
    before = book_info['available_copies']
    assert before == 0

    return_book(patron_id, book_id)

    book_info_after = get_book_info(book_id)
    assert book_info_after['available_copies'] == before + 1

```

```

def test_return_book_calculates_late_fees():
    # You may need to mock borrow date to be > 14 days ago
    patron_id = "234567"
    add_book("Late Fee Book", "Author", "2345672345672", 1)
    book_id = get_book_id_by_isbn("2345672345672")
    borrow_date = fake_past_date(days_ago=25)
    borrow_book(patron_id, book_id, borrow_date=borrow_date)

    success, msg = return_book(patron_id, book_id)
    assert success
    assert "$" in msg or "late fee" in msg.lower()

```

For R5:

```

def test_late_fee_no_overdue():
    patron_id = "345678"
    add_book("On Time Book", "Author", "3456783456783", 1)
    book_id = get_book_id_by_isbn("3456783456783")
    borrow_date = fake_past_date(days_ago=10) # less than due date(14)
    borrow_book(patron_id, book_id, borrow_date=borrow_date)

    resp = get_late_fee(patron_id, book_id)
    assert resp['days_overdue'] == 0
    assert resp['fee'] == 0.0

def test_late_fee_within_first_7_days_overdue():
    patron_id = "456789"
    add_book("7 Day Late", "Author", "4567894567894", 1)
    book_id = get_book_id_by_isbn("4567894567894")
    borrow_date = fake_past_date(days_ago=20) # 6 days overdue (20-14)
    borrow_book(patron_id, book_id, borrow_date=borrow_date)

    resp = get_late_fee(patron_id, book_id)
    assert resp['days_overdue'] == 6
    assert resp['fee'] == pytest.approx(6 * 0.50)

def test_late_fee_more_than_7_days_overdue():
    patron_id = "567890"
    add_book("Over 7 Days Late", "Author", "5678905678905", 1)
    book_id = get_book_id_by_isbn("5678905678905")
    borrow_date = fake_past_date(days_ago=25) # 11 days overdue
    borrow_book(patron_id, book_id, borrow_date=borrow_date)

```

```

resp = get_late_fee(patron_id, book_id)
# 7 days * $0.50 + 4 days * $1.00 = $3.5 + $4 = $7.5
expected_fee = 7 * 0.50 + 4 * 1.00
assert resp['days_overdue'] == 11
assert resp['fee'] == pytest.approx(expected_fee)

def test_late_fee_max_fee_capped():
    patron_id = "678901"
    add_book("Max Fee Book", "Author", "6789016789016", 1)
    book_id = get_book_id_by_isbn("6789016789016")
    borrow_date = fake_past_date(days_ago=50) # 36 days overdue
    borrow_book(patron_id, book_id, borrow_date=borrow_date)

    resp = get_late_fee(patron_id, book_id)
    assert resp['fee'] <= 15.00
    assert resp['fee'] == 15.00

```

For R6:

```

def test_search_title_partial_case_insensitive():
    add_book("The Lord of the Rings", "Tolkien", "11111111111111", 3)
    results = search_books(q="lord", type="title")
    assert any("lord" in book['title'].lower() for book in results)

def test_search_author_partial_case_insensitive():
    add_book("Book", "Jane Austen", "2222222222222", 2)
    results = search_books(q="austen", type="author")
    assert any("austen" in book['author'].lower() for book in results)

def test_search_isbn_exact_match():
    add_book("Title", "Author", "333333333333", 1)
    results = search_books(q="333333333333", type="isbn")
    assert all(book['isbn'] == "333333333333" for book in results)

def test_search_isbn_no_partial_match():
    add_book("Title", "Author", "444444444444", 1)
    results = search_books(q="4444444", type="isbn") # partial, should not return book
    assert all(book['isbn'] != "444444444444" for book in results)

def test_search_empty_query_returns_all():
    add_book("Title1", "Author1", "555555555555", 1)
    add_book("Title2", "Author2", "666666666666", 1)
    results = search_books(q="", type="title")
    all_books = get_catalog()

```

```

assert len(results) == len(all_books)

def test_search_invalid_type():
    with pytest.raises(ValueError):
        search_books(q="anything", type="invalid")

```

For R7:

```

def test_patron_status_currently_borrowed_books():
    patron_id = "777888"
    add_book("Status Book", "Author", "7778887778887", 2)
    book_id = get_book_id_by_isbn("7778887778887")
    borrow_book(patron_id, book_id)

    status = get_patron_status(patron_id)
    assert 'currently_borrowed_books' in status
    assert any(b['book_id'] == book_id for b in status['currently_borrowed_books'])

def test_patron_status_total_late_fees():
    patron_id = "888999"
    add_book("Late Fee Book", "Author", "8889998889998", 1)
    book_id = get_book_id_by_isbn("8889998889998")
    borrow_date = fake_past_date(days_ago=30)
    borrow_book(patron_id, book_id, borrow_date=borrow_date)

    status = get_patron_status(patron_id)
    assert isinstance(status['total_late_fees'], float)
    assert status['total_late_fees'] > 0

def test_patron_status_number_of_books_borrowed():
    patron_id = "999000"
    add_book("Book1", "Author1", "9990001112223", 1)
    add_book("Book2", "Author2", "9990001112224", 1)
    borrow_book(patron_id, get_book_id_by_isbn("9990001112223"))
    borrow_book(patron_id, get_book_id_by_isbn("9990001112224"))

    status = get_patron_status(patron_id)
    assert status['num_books_borrowed'] == 2

def test_patron_status_borrowing_history_includes_past_returns():
    patron_id = "123123"
    add_book("History Book", "Author", "1231231231234", 1)
    book_id = get_book_id_by_isbn("1231231231234")
    borrow_book(patron_id, book_id)

```

```
return_book(patron_id, book_id)

status = get_patron_status(patron_id)
assert 'borrowing_history' in status
assert any(record['book_id'] == book_id for record in status['borrowing_history'])
```

Part 4:

Despite me only providing the AI with information about the requirements and that the test cases generated should be testable with pytest, many of the AI test cases are quite similar to the test cases I created myself. The AI immediately assumed that some of the functions that exist in database.py exist somewhere despite me not providing the AI with that information. This led the AI to generate test cases using very similar functions that I used in my own test cases to help create tests.

The AI's R1 test cases for the most part matched with the test cases I created. The AI additionally tested for cases for adding a book with zero copies and adding a book with an invalid ISBN length, which are both cases I did not test for. I believe at the time I didn't test for either of these cases since the website would already block both of those cases from my initial testing of it, so I figured I didn't need to include them. One thing that I did test for that AI did not however was testing for a missing author, but this isn't as important of a case to miss. One additional thing that the AI did with its test cases that I did not however was that it generally used the same ISBN for every book added into the catalog across its cases. This is something I explicitly avoided when creating my test cases since trying to add a book with the same ISBN as another book would return an error. Therefore trying to run the test cases that the AI provided all at once probably wouldn't work without changing a few of the ISBNs in these cases to unique ISBNs.

The test cases between me and the AI for R2 were vastly different. Since we weren't supplied with a clear function to test in library_service.py for R2, I mainly tested database.py's `get_all_books` function, since that function would return a full list of books with all the information that satisfies R2. I tested this by checking for a valid book list, an empty book list, and other tests that checked to see if the book list would be correct after changing the number of copies a book has or after removing a book from the list. The AI also assumed that it would be testing for some '`get_all_books`' type of function (in its case, it assumed it was called '`get_catalog`'), but it tested for different cases. The AI tested for seeing if an added book is in the list, checking the fields of the list to ensure that the correct information exists, and checking to see if a book in the list is either borrowable or unborrowable. I believe that I could have tested for checking to see if the correct fields were a part of an item in the list, however I believe that checking for whether a book is borrowable or not is unnecessary in this case, since we're already verifying that in testing for R3.

The test cases for R3 that the AI created were all the same cases I tested for as well. We both tested for a successful borrow, borrows with an invalid book ID or patron ID, a borrow of a book with no copies available, a borrow whilst a patron already has the same book borrowed, and a borrow whilst a patron has hit their borrow limit.

The test cases generated by the AI for R4 were not all the same as mine. The AI tested for a valid return, a return whilst a patron hasn't borrowed the book, a test to see if the book's availability is updated properly after a return, and to see if late fees are properly calculated for a book return. I didn't test to see if the book's availability is updated or if late fees are calculated properly. I should have tested for the correct book availability, however I don't believe that testing the late fees is necessary here since that's what I would test in R5, and I've treated the line regarding that in R4 to be something that's checked for by the website rather than library_service.py. I additionally checked for cases where an invalid patron ID or book ID were passed into the function as well, which are both things that the AI did not test for that I think it should have.

The AI generated cases for R5 mostly covered the same things my cases did. The AI tested for returning late fee calculations for books that were not overdue, were within 7 days of being overdue, were more than 7 days overdue but not the max fee, and were the max late fee possible. I checked for if a book was overdue within 7 days and if a book was overdue with a max late fee, but not for when a book was not the max fee but more than 7 days overdue or if the book had no fees, which are both cases I probably should have covered. Like R4 however, the AI did not check for cases of invalid book IDs and patron IDs, which are both cases I covered in my test cases.

For R6, the AI tested for every case that I had. We both tested for searching for books with a partially matching title, a partialing matching author, an exact matching isbn, an invalid partially matching isbn, and for an invalid search option. The AI had an additional test case testing for if an empty title was passed into a partially matching title case, where it expected any title of book to be returned. This is an interesting test case to me since the requirements do not explicitly specify what should happen if an empty search value is passed in for what to search for. When I created my test cases, I assumed that an empty string simply shouldn't work since it wouldn't make sense to search for nothing and get every book title back in return. I also assumed this because the website already blocks users from not inputting anything into its search field if a user attempts to. However since there's nothing explicitly stated in the requirements about this, the AI assumed that an empty string should return everything, so it created a case for it. I don't believe that this case is necessary, but it's interesting to see nonetheless.

Lastly for R7, the AI's test cases and my test cases were completely different. When I created my test cases for R7, I had to make heavy assumptions about what the report would look like and return. Therefore I tested for different cases of patron reports having varying numbers of books to ensure that the information within each report would be correct, with each test case checking all the information in each report to verify them. The AI however created different test cases that checked for only specific information within each report. checking separate cases for

if a report has the correct books in it, if a report's late fees are correct, if a book's borrow count is correct, and if a returned book appears on the report correctly. I did also add in a test case later checking for if a returned book appears on the report since I treated that as a special case, however the other specific information were all things I checked in my test cases collectively to verify the function was working. I additionally also tested for an invalid patron ID being passed into the function, which once again was something that the AI missed.

Looking at all of the cases, the AI did a decent job at covering what test cases needed to be covered by the requirements, even covering some cases I missed during some points. There were a few instances of the AI covering unnecessary cases due to what the requirements specified, but even in those cases the test cases it produced made sense having only given it the requirements. The main thing that the AI missed sometimes however was testing for if the inputs to each function were valid. It did cover invalid function inputs sometimes, but it missed many instances of checking to see if either patron IDs or book IDs being passed in were valid or not, which were cases that I covered in my test cases.

Part 5:

Note: I'm assuming that when integrating the badges and code coverage into our github projects, we only need to be concerned about making the badges actually appear, and worry less about what numbers the badges display (for example, my codecov badge displays I only have 92% coverage in my code right now, which I'm assuming is something I'll address in later assignments)