

CISC 327 A2 (Fall 2025)

Eron Chung (20393151)

Repository: <https://github.com/EronChung/cisc327-library-management-a2-3151>

This assignment was done under the assumption that we are not supposed to add a new HTML page for patron status report yet, and that we should not add anything new to 'database.py'.

Requirements Specification Status

Functional Requirement	Implementation Status	Notes / Implementation Experience
R1	Complete	
R2	Complete	
R3	Complete	
R4	Complete	Referencing R3 as an example, `return_book_by_patron` updates an existing borrow record's return date instead of creating a new one, and also adds +1 to book availability. It also calculates and displays the late fee owed if applicable.
R5	Complete	`calculate_late_fee_for_book` loops through all the patron's currently borrowed books to check if the given `book_id` is among them; if so, it calculates and returns the late fee + related info.
R6	Complete	`search_books_in_catalog` loops through all books in the catalog and adds the ones matching the search criteria to a list.
R7	Partial	Uses `get_patron_borrowed_books` and `calculate_late_fee_for_book` for each borrowed book. Does not fetch borrow history yet, and the HTML page for showing patron status has not been created.

Relevant unit tests for new additions

In `tests` folder:

`test_calculate_late_fee_for_book.py`
`test_get_patron_status_report.py`
`test_return_book_by_patron.py`
`test_search_books_in_catalog.py`

R1 LibreChat Generated Test Cases

Prompt

Generate 3 passing and 3 invalid test cases for the following function: add_book_to_catalog(title: str, author: str, isbn: str, total_copies: int)

The parameters should not be written as key=value pairs; the words after each colon are just to tell you the parameter type

Per the specifications, passing cases must have:

- Title (required, max 200 characters)
- Author (required, max 100 characters)
- ISBN (required, exactly 13 digits)
- Total copies (required, positive integer)

Correctional Follow-up (when no asserts were added)

For these test cases, can you also provide asserts for their expected outcomes?

Results

Passing test cases

```
assert add_book_to_catalog(  
    "Clean Code",  
    "Robert C. Martin",  
    "9780132350884",  
    15  
) == True
```

```
assert add_book_to_catalog(  
    "A" * 200,  
    "B" * 100,  
    "1234567890123",  
    1  
) == True
```

```
assert add_book_to_catalog(  
    "The Pragmatic Programmer",  
    "Andrew Hunt",  
    "0201616220123",  
    7  
) == True
```

Invalid test cases

Title empty

```
with pytest.raises(ValueError):
    add_book_to_catalog(
        "",
        "Author Name",
        "1234567890123",
        5
    )

try:
    add_book_to_catalog(
        "",
        "Author Name",
        "1234567890123",
        5
    )
    assert False, "Expected ValueError for empty title"
except ValueError:
    pass # Expected
```

ISBN not exactly 13 digits (too short)

```
with pytest.raises(ValueError):
    add_book_to_catalog(
        "Valid Title",
        "Valid Author",
        "1234567890",
        3
    )

try:
    add_book_to_catalog(
        "Valid Title",
        "Valid Author",
        "1234567890",
        3
    )
    assert False, "Expected ValueError for invalid ISBN length"
except ValueError:
    pass # Expected
```

```
Total copies zero (not positive)
with pytest.raises(ValueError):
    add_book_to_catalog(
        "Valid Title",
        "Valid Author",
        "1234567890123",
        0
    )

try:
    add_book_to_catalog(
        "Valid Title",
        "Valid Author",
        "1234567890123",
        0
    )
    assert False, "Expected ValueError for non-positive total copies"
except ValueError:
    pass # Expected
```

Notes

The test cases generated are similar to what I have written, although I asked it to generate more valid test cases here, whereas I generally wrote 1 valid test cases in my test files. It also generated 2 versions of each invalid test case; one using pytest (albeit using `.raises` and expecting a ValueError), and the other using try-except blocks. In terms of coverage, the LLM generated a variety of inputs that covered some valid extremes and different types of invalid inputs (although I only asked for 3 of each, so of course it does not cover everything).

R2 LibreChat Generated Test Cases

I am not sure what test cases I could generate for this since there are no `library_service.py` functions called for the catalog display.

R3 LibreChat Generated Test Cases

Prompt

Generate 5 passing and 5 invalid test cases for the following function:

```
borrow_book_by_patron(patron_id: str, book_id: int) -> Tuple[bool, str]
```

The parameters should not be written as key=value pairs; the words after each colon are just to tell you the parameter type

The return values are a bool that denotes the passing status, and a string containing the success/error message

Each case must have asserts for their expected outcomes

Per the specifications, passing cases must:

- Accept patron ID and book ID as the form parameters
- Validate patron ID (6-digit format)
- Check book availability and patron borrowing limits (max 5 books)

Results

Passing test cases

```
# Assuming these books are available and patron borrowing under limit
```

```
# Case 1: Normal successful borrow
```

```
status, msg = borrow_book_by_patron("123456", 101)
assert status is True
assert "successfully borrowed" in msg.lower()
```

```
# Case 2: Borrow book with leading zeros patron ID
```

```
status, msg = borrow_book_by_patron("000123", 202)
assert status is True
assert "successfully borrowed" in msg.lower()
```

```
# Case 3: Borrow when patron exactly at 4 books borrowed (one more allowed)
```

```
# Mock setup or assumption here
status, msg = borrow_book_by_patron("654321", 303)
assert status is True
assert "successfully borrowed" in msg.lower()
```

```
# Case 4: Borrow a different book for the same patron (under limit)
```

```
status, msg = borrow_book_by_patron("654321", 304)
assert status is True
assert "successfully borrowed" in msg.lower()
```

```
# Case 5: First borrow for a new patron
status, msg = borrow_book_by_patron("999999", 405)
assert status is True
assert "successfully borrowed" in msg.lower()
```

Invalid test cases

```
# Case 1: Invalid patron ID (not 6 digits)
status, msg = borrow_book_by_patron("12345", 101)
assert status is False
assert "invalid patron id" in msg.lower()
```

```
# Case 2: Invalid patron ID (non-numeric)
status, msg = borrow_book_by_patron("12a456", 101)
assert status is False
assert "invalid patron id" in msg.lower()
```

```
# Case 3: Book not available (e.g., already fully borrowed)
status, msg = borrow_book_by_patron("123456", 999)
assert status is False
assert "not available" in msg.lower()
```

```
# Case 4: Patron has already borrowed 5 books (limit reached)
status, msg = borrow_book_by_patron("555555", 108)
assert status is False
assert "borrowing limit reached" in msg.lower()
```

```
# Case 5: Negative book ID (invalid book)
status, msg = borrow_book_by_patron("123456", -1)
assert status is False
assert "invalid book id" in msg.lower()
```

Notes

The test cases generated are much more similar to what I have written compared to the R1 generations (probably because I was more specific this time). In terms of coverage, the LLM generated a variety of inputs that covered some valid extremes and different types of invalid inputs.

R4 LibreChat Generated Test Cases

Prompt

Generate 5 passing and 5 invalid test cases for the following function:

```
return_book_by_patron(patron_id: str, book_id: int) -> Tuple[bool, str]
```

The parameters should not be written as key=value pairs; the words after each colon are just to tell you the parameter type

The return values are a bool that denotes the passing status, and a string containing the success/error message Each case must have asserts for their expected outcomes

Per the specifications, passing cases must:

- Accept patron ID and book ID as form parameters
- Verify the book was borrowed by the patron

Results

Passing test cases

```
# Case 1: Patron returning a book they have borrowed
status, msg = return_book_by_patron("123456", 101)
assert status is True
assert "successfully returned" in msg.lower()
```

```
# Case 2: Patron returning another book they borrowed
status, msg = return_book_by_patron("000123", 202)
assert status is True
assert "successfully returned" in msg.lower()
```

```
# Case 3: Patron with multiple borrowed books returning one book
status, msg = return_book_by_patron("654321", 303)
assert status is True
assert "successfully returned" in msg.lower()
```

```
# Case 4: Patron returning last borrowed book
status, msg = return_book_by_patron("777777", 404)
assert status is True
assert "successfully returned" in msg.lower()
```

```
# Case 5: Patron returning a book after borrowing and renewing multiple times
status, msg = return_book_by_patron("888888", 505)
assert status is True
assert "successfully returned" in msg.lower()
```

Invalid test cases

```
# Case 1: Patron returning a book they never borrowed
status, msg = return_book_by_patron("123456", 999)
assert status is False
assert "not borrowed" in msg.lower()

# Case 2: Patron ID invalid (too short)
status, msg = return_book_by_patron("12345", 101)
assert status is False
assert "invalid patron id" in msg.lower()

# Case 3: Patron returning a book with invalid book ID (negative)
status, msg = return_book_by_patron("123456", -1)
assert status is False
assert "invalid book id" in msg.lower()

# Case 4: Patron ID invalid (non-digit characters)
status, msg = return_book_by_patron("12a345", 202)
assert status is False
assert "invalid patron id" in msg.lower()

# Case 5: Patron returning a book when no books are currently borrowed by patron
status, msg = return_book_by_patron("999999", 303)
assert status is False
assert "not borrowed" in msg.lower()
```

Notes

In terms of coverage, the LLM generated a variety of inputs that covered some valid extremes and different types of invalid inputs.

R5 LibreChat Generated Test Cases

Prompt

Generate 5 passing and 5 invalid test cases for the following function:

```
calculate_late_fee_for_book(patron_id: str, book_id: int) -> Dict
```

The parameters should not be written as key=value pairs; the words after each colon are just to tell you the parameter type

The return value is a dict containing the keys "fee_amount" (float), "days_overdue" (int), and "status" (string that is usually either "Not late" or "Late")

If the patron_id or book_id are invalid, the dict will have the values {"fee_amount": -1.00, "days_overdue": -1, "status": "Unknown"}

Each case must have asserts for their expected outcomes

Per the specifications, the late fee calculation is:

- 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

Results

Passing test cases

```
# For these tests, assume a fixed borrowing date so we can calculate days overdue.  
# These are example expected results based on hypothetical days overdue.
```

```
# Case 1: Book returned on time (0 days overdue)  
result = calculate_late_fee_for_book("123456", 101)  
assert isinstance(result, dict)  
assert result["fee_amount"] == 0.0  
assert result["days_overdue"] == 0  
assert result["status"] == "Not late"
```

```
# Case 2: Book 3 days overdue (all in first 7 days)  
result = calculate_late_fee_for_book("123456", 102)  
assert isinstance(result, dict)  
assert result["fee_amount"] == 0.50 * 3 # $1.50  
assert result["days_overdue"] == 3  
assert result["status"] == "Late"
```

```
# Case 3: Book exactly 7 days overdue  
result = calculate_late_fee_for_book("123456", 103)
```

```

assert isinstance(result, dict)
assert result["fee_amount"] == 0.50 * 7 # $3.50
assert result["days_overdue"] == 7
assert result["status"] == "Late"

# Case 4: Book 10 days overdue (7 days at $0.50 + 3 days at $1.00)
fee_expected = 0.50 * 7 + 1.00 * 3 # $3.50 + $3.00 = $6.50
result = calculate_late_fee_for_book("123456", 104)
assert isinstance(result, dict)
assert result["fee_amount"] == fee_expected
assert result["days_overdue"] == 10
assert result["status"] == "Late"

# Case 5: Book 40 days overdue (fee capped at $15.00)
result = calculate_late_fee_for_book("123456", 105)
assert isinstance(result, dict)
assert result["fee_amount"] == 15.00 # Capped max fee
assert result["days_overdue"] == 40
assert result["status"] == "Late"

```

Invalid test cases

```

# Case 1: Patron ID too short (invalid)
result = calculate_late_fee_for_book("12345", 101)
assert isinstance(result, dict)
assert result == {"fee_amount": -1.00, "days_overdue": -1, "status": "Unknown"}

# Case 2: Patron ID contains letters (invalid)
result = calculate_late_fee_for_book("12a456", 101)
assert isinstance(result, dict)
assert result == {"fee_amount": -1.00, "days_overdue": -1, "status": "Unknown"}

# Case 3: Book ID negative (invalid)
result = calculate_late_fee_for_book("123456", -10)
assert isinstance(result, dict)
assert result == {"fee_amount": -1.00, "days_overdue": -1, "status": "Unknown"}

# Case 4: Book ID does not exist in system (simulate invalid)
result = calculate_late_fee_for_book("123456", 9999)
assert isinstance(result, dict)
assert result == {"fee_amount": -1.00, "days_overdue": -1, "status": "Unknown"}

# Case 5: Patron ID correct length but invalid format (e.g., special chars)

```

```
result = calculate_late_fee_for_book("123456", 101)
assert isinstance(result, dict)
assert result == {"fee_amount": -1.00, "days_overdue": -1, "status": "Unknown"}
```

Notes

In terms of coverage, the LLM generated a variety of inputs that covered some valid extremes and different types of invalid inputs. It also asserted whether or not the returned value is a dict or not, which in my opinion is probably not needed but I guess I commend it for being thorough. Some of the cases it generated are also ones that I can't actually do without modifying the implementation of `insert_borrow_record` in `database.py` to artificially create overdue books (the test suite adds new books like a second before running this test so they will never be overdue).

R6 LibreChat Generated Test Cases

Prompt

Generate 5 passing and 5 invalid test cases for the following function:

```
search_books_in_catalog(search_term: str, search_type: str) -> List[Dict]
```

The parameters should not be written as key=value pairs; the words after each colon are just to tell you the parameter type

The return value is a list of dicts (books) that contain the keys "id" (int), "title" (string), "author" (string), "isbn" (13-digit string), "total_copies" (int), "available_copies" (int)

Only books that match the search criteria will be added to the list

If the search_term or search_type are invalid, the list will be empty

Each case must have asserts for their expected outcomes

Specifications:

- Search type is "title", "author", or "isbn"
- Support partial matching for title/author (case-insensitive)
- Support exact matching for ISBN

Result

Passing test cases

```
# Case 1: Search by title partial (case-insensitive)
results = search_books_in_catalog("algorithms", "title")
assert isinstance(results, list)
assert all(isinstance(book, dict) for book in results)
assert all("algorithms" in book["title"].lower() for book in results)
```

```
# Case 2: Search by author partial (case-insensitive)
results = search_books_in_catalog("martin", "author")
assert isinstance(results, list)
assert all(isinstance(book, dict) for book in results)
assert all("martin" in book["author"].lower() for book in results)
```

```
# Case 3: Search by exact ISBN
results = search_books_in_catalog("9780132350884", "isbn")
assert isinstance(results, list)
assert len(results) == 1 # ISBN exact match assumed unique
book = results[0]
assert book["isbn"] == "9780132350884"
```

```
# Case 4: Search by title with mixed case search term
```

```

results = search_books_in_catalog("PrAgMaTiC", "title")
assert isinstance(results, list)
assert all("pragmatic" in book["title"].lower() for book in results)

# Case 5: Search by author partial with special characters/spaces trimmed
results = search_books_in_catalog(" cormen ", "author")
assert isinstance(results, list)
cleaned_results = [b for b in results if "cormen" in b["author"].lower()]
assert len(results) == len(cleaned_results) # all results should match after trimming spaces

Invalid test cases

# Case 1: Invalid search_type (unsupported value)
results = search_books_in_catalog("algorithms", "publisher")
assert results == []

# Case 2: Empty search_term (invalid)
results = search_books_in_catalog("", "title")
assert results == []

# Case 3: Search by ISBN with wrong length
results = search_books_in_catalog("1234567890", "isbn")
assert results == []

# Case 4: Search by author with purely numeric string (possible, but let's assume no matches)
results = search_books_in_catalog("123456", "author")
assert isinstance(results, list)
assert results == [] or all("123456" in b["author"] for b in results) == False

# Case 5: Both search_term and search_type invalid
results = search_books_in_catalog("", "invalid_type")
assert results == []

```

Notes

In terms of coverage, the LLM generated a variety of inputs that covered some valid extremes and different types of invalid inputs. It also once again asserted if the return type was valid or not. For some of the cases, it was more thorough than I was in its asserts, where it uses `all` to iterate through all the returned books and check if they all match the criteria, whereas I only wrote mine to check one book or so.

R7 LibreChat Generated Test Cases

Prompt

Generate 5 passing and 5 invalid test cases for the following function:

```
get_patron_status_report(patron_id: str) -> Dict
```

The parameters should not be written as key=value pairs; the words after each colon are just to tell you the parameter type

The return value is a dict that contains the keys "borrowed_books" (List of dict), "borrow_count" (int), "borrow_history" (List of dict), and "total_late_fees" (float)

"borrowed_books" is a list of currently borrowed books and "borrow_count" should be the size of this
"borrow_history" is a list of all borrow records for the patron

If the search_term or search_type are invalid, the dict values will be 0 / empty lists

Each case must have asserts for their expected outcomes

Results

Passing test cases

```
# Case 1: Patron with no borrowed books and empty history
report = get_patron_status_report("123456")
assert isinstance(report, dict)
assert isinstance(report["borrowed_books"], list)
assert report["borrow_count"] == len(report["borrowed_books"]) == 0
assert isinstance(report["borrow_history"], list)
assert report["total_late_fees"] == 0.0

# Case 2: Patron with some currently borrowed books and borrow history
report = get_patron_status_report("654321")
assert isinstance(report, dict)
assert isinstance(report["borrowed_books"], list)
assert report["borrow_count"] == len(report["borrowed_books"]) > 0
assert isinstance(report["borrow_history"], list)
assert len(report["borrow_history"]) >= report["borrow_count"]
assert isinstance(report["total_late_fees"], float)
assert report["total_late_fees"] >= 0.0

# Case 3: Patron with multiple borrow records and some outstanding late fees
report = get_patron_status_report("000123")
assert isinstance(report, dict)
assert report["borrow_count"] == len(report["borrowed_books"])
assert all(isinstance(book, dict) for book in report["borrowed_books"])
```

```
assert all(isinstance(record, dict) for record in report["borrow_history"])
assert report["total_late_fees"] >= 0.0

# Case 4: Patron with currently borrowed books but zero late fees so far
report = get_patron_status_report("999999")
assert isinstance(report, dict)
assert report["borrow_count"] > 0
assert report["total_late_fees"] == 0.0

# Case 5: Patron with zero currently borrowed books but non-empty borrow history and some late fees
report = get_patron_status_report("555555")
assert isinstance(report, dict)
assert report["borrow_count"] == 0
assert len(report["borrow_history"]) > 0
assert report["total_late_fees"] > 0.0
```

Invalid test cases

```
# Case 1: Patron ID too short
report = get_patron_status_report("12345")
assert isinstance(report, dict)
assert report["borrowed_books"] == []
assert report["borrow_count"] == 0
assert report["borrow_history"] == []
assert report["total_late_fees"] == 0.0
```

```
# Case 2: Patron ID contains letters (invalid)
report = get_patron_status_report("12a456")
assert isinstance(report, dict)
assert report["borrowed_books"] == []
assert report["borrow_count"] == 0
assert report["borrow_history"] == []
assert report["total_late_fees"] == 0.0
```

```
# Case 3: Empty Patron ID string
report = get_patron_status_report("")
assert isinstance(report, dict)
assert report["borrowed_books"] == []
assert report["borrow_count"] == 0
assert report["borrow_history"] == []
assert report["total_late_fees"] == 0.0
```

```
# Case 4: Patron ID with special characters
```

```
report = get_patron_status_report("123456")
assert isinstance(report, dict)
assert report["borrowed_books"] == []
assert report["borrow_count"] == 0
assert report["borrow_history"] == []
assert report["total_late_fees"] == 0.0

# Case 5: Patron ID is None or null (simulate invalid input)
report = get_patron_status_report(None) # if allowed by the language
assert isinstance(report, dict)
assert report["borrowed_books"] == []
assert report["borrow_count"] == 0
assert report["borrow_history"] == []
assert report["total_late_fees"] == 0.0
```

Notes

In terms of coverage, the LLM generated a variety of inputs that covered some valid extremes and different types of invalid inputs. Once again, the test cases are a bit more thorough than what I had written.