CISC 327 Assignment 2

GitHub Repository Link: https://github.com/Ansh-Sudani/cisc327-library-management-a2-0507

1. Complete Function Implementation [25%]

I completed all remaining function implementations in the library_service.py module:

- The book borrowing function (R3) was adjusted to correctly enforce the maximum borrowing limit using >= 5.
- The book return function (R4) and late fee calculation function (R5) were newly implemented according to the assignment requirements, including updating book availability, recording return dates, and computing overdue fees with the correct daily rates and maximum cap.
- Basic search functionality (R6) was implemented to support title, author, and ISBN searches, including partial matches for title/author and case-insensitive matching.
- The patron status report function (R7) was implemented to return the patron's borrowed books, due dates, total number of books currently borrowed, borrowing history, and outstanding late fees.

All features were successfully tested locally using pytest and verified through the web application interface, ensuring proper integration and correctness.

2. Comprehensive Test Suite Development [10%]

I extended the test suite to cover the newly implemented functions:

- R4 (Book Return) – tests include successful returns, invalid patron IDs, and unavailable books.
- R5 (Late Fee Calculation) – tests include on-time returns, overdue ≤ 7 days, overdue > 7 days, and maximum fee cap ($15).
- R6 (Book Search) – tests include searches by title, author, and ISBN, partial and case-insensitive matching, and exact ISBN matching.
- R7 (Patron Status Report) – tests include verification of borrowed books, due dates, total borrowed count, outstanding fees, and report structure.

Each test verifies input validation, expected outputs, and edge conditions using mocks where necessary. All tests passed successfully with pytest, confirming the correctness of the implementations and ensuring full coverage of all functional requirements.

3. AI-Assisted Test Generation [20%]

I decided to use LibreChat:

CISC 327 Assignment 2

https://librechat.queensu.ca/c/37206222-d430-42b4-af53-122b4fe6c42e

If the link is inaccessible, here's what happened: I gave Libre my library_service.py code, requirements_specification.md, as well as the 6 test files I wrote. I told it to "Generate TestCases for R1, R3, R4, R5, R6, R7, like python test_ .... .py files for each, thanks". Libre generated the code for each of the test files (test_r1_add_book_to_catalog.py, test_r3_borrow_book_by_patron.py, test_r4_return_book_by_patron.py, test_r5_calculate_late_fee.py, test_r6_search_books_in_catalog.py, test_r7_get_patron_status_report.py). These files can be found inside the project.

4. Test-Case Comparison & Analysis [20%]

R1: Add Book to Catalog — Test Case Comparison & Analysis

Human-Written Test Cases (Manual)

My manually written test suite for add_book_to_catalog() used pytest and focused on the following:

- Verifying that valid book entries were successfully added.
- Ensuring appropriate error handling for missing title, invalid ISBN, and negative copy counts.
- Using mocks for database-related functions (get_book_by_isbn, insert_book) to isolate logic and avoid dependency on external systems.

Strengths:

- Concise and efficient tests covering all primary functional requirements.
- Focused on realistic user-facing inputs.
- Ensured both Boolean success flags and message correctness.

Limitations:

- Did not include boundary checks for overly long titles or authors.
- Did not test multiple invalid data types for copy count.
- Message assertions used partial string matching rather than full message validation.

AI-Generated Test Cases (Libre)

Libre's AI-generated test suite, created using the unittest framework, tested the same function but with broader validation coverage:

- Included edge cases for long titles (over 200 chars) and long author names (over 100 chars).
- Checked non-positive and non-integer copy counts (0, -1, "five", 3.5).

- Simulated duplicate ISBN entries and database errors.
- Verified full message text for exact matching.

Strengths:

- Comprehensive coverage across all validation paths.
- Detailed assertions for both success and failure messages.
- Automatically identified edge cases not covered in manual testing.

Limitations:

- Slightly more verbose, which could make maintenance slower.
- Some redundancy (multiple tests for similar invalid input types).

Findings

- Both test suites validated the main success and failure paths.
- The AI-generated tests offered better input boundary coverage and data validation.
- My tests were more concise and focused on functional correctness.
- Libre's output complemented the manual suite by revealing missing edge cases that strengthened overall coverage.

R3: Book Borrowing Interface — Test Case Comparison & Analysis

Human-Written Test Cases (Manual)

My manually written test suite for borrow_book_by_patron() used pytest and focused on the following:

- Verifying successful borrowing when the patron ID and book availability meet requirements.
- Handling invalid patron IDs, unavailable books, and books not found.
- Ensuring the maximum borrowing limit (5 books) is enforced.
- Using mocks for database interactions (get_book_by_id, get_patron_borrow_count, insert_borrow_record, update_book_availability) to isolate logic.

Strengths:

- Concise tests that clearly target functional paths.

- Covers both success and common failure scenarios.
- Simple to read and maintain, with realistic input scenarios.

Limitations:

- Did not simulate database errors for insert or update operations.
- Did not fully check edge cases for patron borrow counts at exactly the limit.
- Partial assertions for success messages (did not always check for due date formatting).

AI-Generated Test Cases (Libre)

Libre's AI-generated tests, using the unittest framework, covered the same function but with broader validation:

- Included successful borrow scenarios with due date verification.
- Checked invalid patron ID variations, including empty strings, non-digit IDs, and IDs longer/shorter than 6 digits.
- Simulated book unavailability, book not found, and maximum borrow limit reached.
- Added tests for database errors during borrow record insertion and book availability updates.
- Verified full message text for exact correctness, including due date formatting.

Strengths:

- Comprehensive coverage of both normal and edge/error scenarios.
- Validates exact success/error messages, improving consistency.
- Identifies potential database failure points not covered in manual tests.

Limitations:

- More verbose and slightly repetitive.
- Some overlap in scenarios (e.g., multiple tests for patron ID errors).

Findings

- Both test suites validated main functional paths and common failure cases.
- AI-generated tests extended coverage by adding database error scenarios and stricter message validation.
- Manual tests were concise and focused on typical user flows, while AI tests caught edge cases.

CISC 327 Assignment 2

R4: Book Return Processing — Test Case Comparison & Analysis

Human-Written Test Cases (Manual)

My manually written pytest suite for return_book_by_patron() verified:

- Successful book returns when patron ID and book availability are valid.
- Handling of invalid patron IDs and missing borrow records.
- Proper updates to book availability and return date.
- Use of mocks for all DB operations to ensure isolated logic.

Strengths:

- Concise and easy to follow.
- Covers primary success and basic error scenarios.
- Focused on ensuring the main return process works correctly.

Limitations:

- Did not include tests for missing borrow records or unavailable database updates.
- Limited validation for message accuracy.
- Lacked diverse invalid ID edge cases.

AI-Generated Test Cases (Libre)

Libre's unittest-based suite covered the same feature but extended validation significantly:

- Tested successful returns, invalid patron IDs (various forms), and missing borrow records.
- Checked book-not-found conditions and simulated database update failures.
- Verified exact output messages for all scenarios.
- Used additional mocks to simulate database failures cleanly.

Strengths:

- Comprehensive coverage across all expected and edge scenarios.
- Full validation of user-facing messages.
- Includes negative testing for unavailable database updates and missing borrow records.

CISC 327 Assignment 2

Limitations:

- More verbose; repeated setup for mocks.
- Some scenarios overlap slightly (book not found vs no borrow record).

Findings

- Both suites validated main functionality, but the AI-generated version extended error-handling and messaging coverage.
- Manual tests confirmed correct logic for successful operations, while AI tests highlighted edge scenarios and database inconsistencies.

R5: Late Fee Calculation API — Test Case Comparison & Analysis

Human-Written Test Cases (Manual)

My pytest suite for calculate_late_fee_for_book() focused on validating fee calculations for typical return timelines:

- Verified no fee for on-time returns.
- Correctly applied rates for ≤7 days and >7 days overdue.
- Enforced a maximum late fee cap of $15.
- Asserted correct fee amounts, overdue days, and return status.

Strengths:

- Covered all major computational paths (on time, minor delay, major delay, capped fee).
- Simple, clean assertions verifying both numeric and string results.
- Used datetime arithmetic to simulate realistic borrowing timelines.

Limitations:

- Did not test returns before due date (early return).
- Did not include multiple edge day boundaries (e.g., exactly 7 or 8 days late).
- Partial string matching for status messages.

AI-Generated Test Cases (Libre)

Libre's unittest-based test suite expanded upon this by adding more detailed edge testing:

- Included explicit checks for early returns and exact 7-day boundaries.
- Used assertAlmostEqual for floating-point comparisons (improves precision testing).
- Validated exact message text ("On time", "Overdue") for all scenarios.
- Reused consistent setup for cleaner test initialization.

Strengths:

- High precision and structured setup method (setUp) improved maintainability.
- Comprehensive coverage of all rate tiers and the capped-fee logic.
- Explicit day-based tests (5, 7, 10, 50 days late) ensured correctness across fee thresholds.

Limitations:

- Slightly repetitive in structure.
- Minor redundancy between 5-, 7-, and 10-day tests (similar logic).

Findings

- Both test sets confirmed correct fee computation logic.
- Libre's suite offered broader temporal coverage (before, exact, after due date).
- Manual tests were efficient but skipped a few day-boundary checks and message precision.

R6: Book Search Functionality — Test Case Comparison & Analysis

Human-Written Test Cases (Manual)

My pytest-based suite for search_books_in_catalog() focused on verifying partial matching (case-insensitive) and exact ISBN matches using realistic catalog data. Key scenarios covered included:

- Partial, case-insensitive search by title and author.
- Exact ISBN match (no partial matches allowed).
- No-results cases for nonexistent queries.

Strengths:

- Clear functional coverage of all major query types.
- Case-insensitive verification for both title and author.

- Asserted no false positives on partial ISBN input.

Limitations:

- No invalid type parameter testing.
- Limited sample dataset (3 books).
- No checks for multi-result queries (e.g., multiple titles containing “Python”).

AI-Generated Test Cases (Libre)

Libre’s unittest suite expanded coverage and refined edge-case testing:

- Added explicit tests for invalid type parameter (returns empty list).
- Broader dataset enabling multi-result queries (e.g., “Python” in two titles).
- Consistent @patch setup and structured unittest class.
- Used containment checks (in b['title'].lower()) for more robust validation.

Strengths:

- Comprehensive coverage of valid, invalid, and multi-match scenarios.
- Enhanced assertion accuracy via logical conditions.
- Improved maintainability using setUp-like reuse of sample data.

Limitations:

- Did not explicitly check message or response formatting consistency with catalog view.
- Focused primarily on logical correctness, less on display conformity.

Findings

- Both suites validated correct search behavior across title, author, and ISBN inputs.
- Libre’s version enhanced realism and completeness through multi-result and invalid-type coverage.
- Manual tests provided concise, functional validation but with narrower scope.

R7: Patron Status Report — Test Case Comparison & Analysis

Human-Written Test Cases (Manual)

CISC 327 Assignment 2

My pytest-based test suite for get_patron_status_report() validated the completeness and structure of the returned report. Key coverage included:

- Ensuring the report is a dictionary with required fields (patron_id, borrowed_books, total_borrowed, outstanding_fees, borrowing_history).
- Verifying type consistency (e.g., borrowed_books as list, outstanding_fees as number).
- Checking that total_borrowed equals the number of currently borrowed books.

Strengths:

- Clear validation of report structure and field presence.
- Asserted consistency between field values (e.g., total count).
- Simple and readable layout.

Limitations:

- Did not verify contents of individual borrowed book entries (e.g., missing due_date or status).
- Lacked checks for empty or invalid patron IDs.
- No mocks for external data dependencies.

AI-Generated Test Cases (Libre)

Libre's unittest suite expanded on the same base but with richer field-level validation:

- Verified that each borrowed book entry included title, due_date, and status.
- Added explicit type assertions for numerical and list-based fields.
- Checked logical consistency between total count and borrowed list length.
- Used the unittest framework for better modularity and reusability.

Strengths:

- More granular validation of nested structures.
- Stronger field completeness and consistency checks.
- Type-safe assertions improve robustness.

Limitations:

- Still assumes valid patron ID input (no invalid-user testing).
- Did not test edge cases such as zero borrowed books or no fees.

CISC 327 Assignment 2

Findings

- Both test suites correctly validated the general structure of patron reports.
- The AI-generated tests provided more rigorous, per-field validation within nested data structures.
- Manual tests ensured overall integrity but missed fine-grained checks.

5. CI/CD Pipeline Setup [25%]

Line 52: README.md