https://github.com/dike12/cisc327-library-management-a2-1493

# Table of Contents

Table of Contents ..... 1
**R1** ..... 2
**R2** ..... 7
**R3** ..... 13
**R4** ..... 18
**R5** ..... 25
**R6** ..... 32
**R7** ..... 38

# Introduction

You can use the table of contents to navigate the report, its this clunky because of the test cases that have been pasted on here. The pasted tested cases where generated by AI, the difference between the test cases i made from assignment 1 and the ones i added in this assignment is that AI was able to test the edge-cases. I was able toking about testing the input and thinks that came easily to mind but in contrast  the AI generated test cases was very through, testing each functionality the the requirements. The quality of the AI generated test cases were also far superior to mine since it was trained probably all the textbooks that relate to testing software. The AI tool i used was Claude 3.5, through cursor in VScode. The

# R1

**Added these extra test cases to add_book_to_catalog_test():**

```
    def test_add_book_special_characters_in_title(self):
        """
        Positive test: Title containing special characters
        Expected: Success
        """
        success, message = add_book_to_catalog("Book! @#$%^&*()", "Valid Author",
"1234567890123", 1)

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

```python
def test_add_book_unicode_characters(self):
    """
    Positive test: Title and author with Unicode characters
    Expected: Success
    """
    success, message = add_book_to_catalog("título del libro", "José García", "1234567890123", 1)

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


def test_add_book_numbers_in_title(self):
    """
    Positive test: Title containing numbers
    Expected: Success
    """
    success, message = add_book_to_catalog("Book 123", "Valid Author", "1234567890123", 1)

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


def test_add_book_duplicate_title_different_isbn(self):
    """
    Positive test: Same title but different ISBN
    Expected: Success
```

```
        """
        add_book_to_catalog("Duplicate Title", "Author One", "1111111111111", 1)
        success, message = add_book_to_catalog("Duplicate Title", "Author Two", "2222222222222", 1)

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

    def test_add_book_max_copies(self):
        """
        Positive test: Large number of copies
        Expected: Success
        """
        success, message = add_book_to_catalog("Valid Title", "Valid Author", "1234567890123",
999999)

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

    def test_add_book_title_only_numbers(self):
        """
        Positive test: Title consisting only of numbers
        Expected: Success
        """
        success, message = add_book_to_catalog("12345", "Valid Author", "1234567890123", 1)

        assert success == True
```

```
    assert "successfully added" in message.lower()


def test_add_book_null_title(self):
    """
    Negative test: None as title
    Expected: Failure with appropriate error message
    """
    success, message = add_book_to_catalog(None, "Valid Author", "1234567890123", 1)

    assert success == False
    assert "title is required" in message.lower()


def test_add_book_null_author(self):
    """
    Negative test: None as author
    Expected: Failure with appropriate error message
    """
    success, message = add_book_to_catalog("Valid Title", None, "1234567890123", 1)

    assert success == False
    assert "author is required" in message.lower()


def test_add_book_invalid_isbn_letters_and_numbers(self):
    """
    Negative test: ISBN with mix of letters and numbers but correct length
```

```
    Expected: Failure with ISBN validation error
    """
    success, message = add_book_to_catalog("Valid Title", "Valid Author", "12345ABC67890", 1)

    assert success == False
    assert "13 digits" in message


def test_add_book_special_characters_in_isbn(self):
    """
    Negative test: ISBN with special characters
    Expected: Failure with ISBN validation error
    """
    success, message = add_book_to_catalog("Valid Title", "Valid Author", "123456!@#$%90", 1)

    assert success == False
    assert "13 digits" in message
```

# R2

**Added the test cases for R2 functionality since i missed it in Assignment 1 - book_catalog_display_test.py**

```
def setup_database(self):
        """Initialize database before each test"""
        init_database()
        add_sample_data()


    def test_get_all_books_not_empty(self):
        """
        Test that catalog returns books when database is populated
        Expected: List containing at least one book
        """
        books = get_all_books()
        assert len(books) > 0
        assert isinstance(books, list)
```

```python
def test_book_catalog_structure(self):
    """
    Test that each book entry contains all required fields
    Expected: All required fields present with correct types
    """
    books = get_all_books()
    required_fields = ['id', 'title', 'author', 'isbn', 'total_copies', 'available_copies']

    for book in books:
        assert isinstance(book, dict)
        for field in required_fields:
            assert field in book

        # Verify field types
        assert isinstance(book['id'], int)
        assert isinstance(book['title'], str)
        assert isinstance(book['author'], str)
        assert isinstance(book['isbn'], str)
        assert isinstance(book['total_copies'], int)
        assert isinstance(book['available_copies'], int)

def test_available_copies_less_or_equal_total(self):
    """
    Test that available copies is always <= total copies
    Expected: Available copies not exceeding total copies
    """
```

```
    books = get_all_books()
    for book in books:
        assert book['available_copies'] <= book['total_copies']
        assert book['available_copies'] >= 0

def test_catalog_alphabetical_order(self):
    """
    Test that books are returned in alphabetical order by title
    Expected: Books sorted alphabetically by title
    """
    books = get_all_books()
    titles = [book['title'] for book in books]
    sorted_titles = sorted(titles)
    assert titles == sorted_titles

def test_newly_added_book_appears(self):
    """
    Test that newly added books appear in catalog
    Expected: New book present in catalog
    """
    new_book = {
        'title': 'New Test Book',
        'author': 'Test Author',
        'isbn': '1234567890123',
        'total_copies': 1
    }
```

```
    add_book_to_catalog(new_book['title'], new_book['author'], new_book['isbn'],
new_book['total_copies'])


    books = get_all_books()
    found = False
    for book in books:
        if (book['title'] == new_book['title'] and
            book['author'] == new_book['author'] and
            book['isbn'] == new_book['isbn']):
            found = True
            break

    assert found == True


  def test_catalog_with_zero_available_copies(self):
    """
    Test display of books with zero available copies
    Expected: Books with zero copies still displayed
    """
    books = get_all_books()
    unavailable_books = [book for book in books if book['available_copies'] == 0]
    assert len(unavailable_books) > 0


  def test_unique_book_ids(self):
    """
    Test that all book IDs in catalog are unique
    Expected: No duplicate IDs
```

```
    """
    books = get_all_books()
    book_ids = [book['id'] for book in books]
    assert len(book_ids) == len(set(book_ids))

def test_valid_isbn_format(self):
    """
    Test that all ISBNs in catalog are valid 13-digit numbers
    Expected: All ISBNs are 13 digits
    """
    books = get_all_books()
    for book in books:
        assert len(book['isbn']) == 13
        assert book['isbn'].isdigit()

def test_non_empty_required_fields(self):
    """
    Test that no required fields are empty
    Expected: No empty required fields
    """
    books = get_all_books()
    for book in books:
        assert book['title'].strip() != ""
        assert book['author'].strip() != ""
        assert book['isbn'].strip() != ""

def test_positive_copy_numbers(self):
```

```
    """
    Test that copy numbers are non-negative
    Expected: All copy counts >= 0
    """
    books = get_all_books()
    for book in books:
        assert book['total_copies'] > 0
        assert book['available_copies'] >= 0


def test_empty_catalog_after_init(self):
    """
    Test catalog state with fresh database
    Expected: Empty list when no books added
    """
    init_database()  # Reset database without sample data
    books = get_all_books()
    assert len(books) == 0
    assert isinstance(books, list)


def test_multiple_copies_display(self):
    """
    Test display of books with multiple copies
    Expected: Correct total and available copy counts
    """
    add_book_to_catalog("Multiple Copies Book", "Test Author", "9999999999999", 5)
```

```
    books = get_all_books()
    multi_copy_book = None
    for book in books:
        if book['isbn'] == "9999999999999":
            multi_copy_book = book
            break

    assert multi_copy_book is not None
    assert multi_copy_book['total_copies'] == 5
    assert multi_copy_book['available_copies'] == 5
```

# R3

**added these extra test cases to  borrow_book_by_patron_test.py**

```
def test_borrow_multiple_books_within_limit(self):
    """
    Positive test: Patron borrows multiple books within 5-book limit
    Expected: Success for each borrow
    """
    patron_id = "111111"

    # Borrow 3 different books
    success1, _ = borrow_book_by_patron(patron_id, 1)
    success2, _ = borrow_book_by_patron(patron_id, 2)
    success3, _ = borrow_book_by_patron(patron_id, 4)

    assert success1 == True
```

```
    assert success2 == True
    assert success3 == True

def test_borrow_exceeding_limit(self):
    """
    Negative test: Patron attempts to borrow more than 5 books
    Expected: Failure with limit exceeded message
    """
    patron_id = "222222"

    # Borrow 5 books first
    for book_id in range(1, 6):
        borrow_book_by_patron(patron_id, book_id)

    # Try to borrow 6th book
    success, message = borrow_book_by_patron(patron_id, 6)

    assert success == False
    assert "maximum borrowing limit" in message.lower()
    assert "5 books" in message

def test_borrow_same_book_twice(self):
    """
    Negative test: Patron attempts to borrow same book twice
    Expected: Failure with availability error
    """
    patron_id = "333333"
```

```
    # Borrow book first time
    success1, _ = borrow_book_by_patron(patron_id, 1)
    # Try to borrow same book again
    success2, message = borrow_book_by_patron(patron_id, 1)

    assert success1 == True
    assert success2 == False
    assert "not available" in message.lower()

def test_borrow_book_due_date_calculation(self):
    """
    Positive test: Verify due date is exactly 14 days from borrow date
    Expected: Success with correct due date
    """
    success, message = borrow_book_by_patron("444444", 1)

    assert success == True
    expected_date = (datetime.now() + timedelta(days=14)).strftime("%Y-%m-%d")
    assert expected_date in message

def test_borrow_book_last_copy(self):
    """
    Positive test: Borrow last available copy of a book
    Expected: Success and book becomes unavailable
    """
    patron_id = "555555"
```

```
    book_id = 2  # Assuming this book has limited copies

    # Borrow last copy
    success1, _ = borrow_book_by_patron(patron_id, book_id)
    # Try to borrow again
    success2, message = borrow_book_by_patron("666666", book_id)

    assert success1 == True
    assert success2 == False
    assert "not available" in message.lower()

def test_borrow_book_concurrent_requests(self):
    """
    Test concurrent borrowing requests for same book
    Expected: Only one request should succeed
    """
    patron1_id = "777777"
    patron2_id = "888888"
    book_id = 1

    # Simulate concurrent requests
    success1, _ = borrow_book_by_patron(patron1_id, book_id)
    success2, message2 = borrow_book_by_patron(patron2_id, book_id)

    assert success1 == True
    assert (success1 and not success2) or (not success1 and success2)
```

```python
def test_borrow_book_float_book_id(self):
    """
    Negative test: Book ID as float
    Expected: Failure with invalid book ID
    """
    success, message = borrow_book_by_patron("123456", 1.5)

    assert success == False
    assert "book not found" in message.lower()

def test_borrow_book_special_chars_patron_id(self):
    """
    Negative test: Patron ID with special characters
    Expected: Failure with invalid patron ID
    """
    success, message = borrow_book_by_patron("12@456", 1)

    assert success == False
    assert "invalid patron id" in message.lower()

def test_borrow_book_whitespace_patron_id(self):
    """
    Negative test: Patron ID with leading/trailing whitespace
    Expected: Failure with invalid patron ID
    """
    success, message = borrow_book_by_patron(" 123456 ", 1)
```

```
    assert success == False
    assert "invalid patron id" in message.lower()
```

# R4

**Implemented R4**: Book Return Processing requirement, I wrote the test cases before from the last assignment but they obviously failed because there was was no implementation. Also changed the front end of the return book page to remove the not yet implemented sign. **Functionality has been fully implemented**

**Added test extra test cases to R4 test cases:**

```
def setup_method(self):
    """Setup test environment before each test"""
    init_database()
    # Borrow a book for testing returns
    from library_service import borrow_book_by_patron
    borrow_book_by_patron("123456", 1)  # Borrow book ID 1
    borrow_book_by_patron("654321", 2)  # Borrow book ID 2


  def test_successful_book_return(self):
    """
    Positive test: Valid return of borrowed book
    Expected: Success with confirmation message
```

```
    """
    success, message = return_book_by_patron("123456", 1)

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


def test_return_book_not_borrowed(self):
    """
    Negative test: Return book that wasn't borrowed
    Expected: Failure with appropriate message
    """
    success, message = return_book_by_patron("123456", 3)

    assert success == False
    assert "not borrowed" in message.lower()


def test_return_book_by_wrong_patron(self):
    """
    Negative test: Return book borrowed by different patron
    Expected: Failure with appropriate message
    """
    success, message = return_book_by_patron("999999", 1)

    assert success == False
    assert "not borrowed by this patron" in message.lower()
```

```
def test_return_already_returned_book(self):
    """
    Negative test: Return a book that was already returned
    Expected: Failure with appropriate message
    """
    # First return
    return_book_by_patron("123456", 1)
    # Second return attempt
    success, message = return_book_by_patron("123456", 1)

    assert success == False
    assert "not borrowed" in message.lower()


def test_return_nonexistent_book(self):
    """
    Negative test: Return a book that doesn't exist
    Expected: Failure with book not found message
    """
    success, message = return_book_by_patron("123456", 9999)

    assert success == False
    assert "book not found" in message.lower()


def test_return_book_updates_availability(self):
```

```
    """
    Positive test: Verify book availability is updated after return
    Expected: Success and available copies increased
    """
    from library_service import get_book_by_id

    # Get initial availability
    initial_book = get_book_by_id(1)
    initial_copies = initial_book['available_copies']

    # Return book
    success, _ = return_book_by_patron("123456", 1)

    # Check updated availability
    updated_book = get_book_by_id(1)
    assert success == True
    assert updated_book['available_copies'] == initial_copies + 1


def test_return_book_with_late_fee(self):
    """
    Test return of overdue book with late fee
    Expected: Success with late fee message
    """
    from datetime import datetime, timedelta
    # Simulate an overdue book by adjusting the due date in database
    conn = get_db_connection()
```

```
    past_due_date = (datetime.now() - timedelta(days=5)).isoformat()
    conn.execute('''
        UPDATE borrow_records
        SET due_date = ?
        WHERE patron_id = ? AND book_id = ? AND return_date IS NULL
    ''', (past_due_date, "654321", 2))
    conn.commit()
    conn.close()

    success, message = return_book_by_patron("654321", 2)

    assert success == True
    assert "late fee" in message.lower()
    assert "$" in message  # Should mention fee amount

def test_return_book_on_time(self):
    """
    Positive test: Return book before due date
    Expected: Success with no late fee
    """
    success, message = return_book_by_patron("123456", 1)

    assert success == True
    assert "late fee" not in message.lower()
```

```python
def test_return_multiple_books(self):
    """
    Positive test: Return multiple books by same patron
    Expected: Success for each return
    """
    success1, _ = return_book_by_patron("654321", 2)
    success2, _ = return_book_by_patron("123456", 1)

    assert success1 == True
    assert success2 == True


def test_return_book_null_book_id(self):
    """
    Negative test: Return with None as book ID
    Expected: Failure with validation error
    """
    success, message = return_book_by_patron("123456", None)

    assert success == False
    assert "invalid" in message.lower()


def test_return_book_string_book_id(self):
    """
    Negative test: Return with string as book ID
    Expected: Failure with validation error
```

```
    """
    success, message = return_book_by_patron("123456", "1")

    assert success == False
    assert "invalid" in message.lower()


def test_return_book_float_book_id(self):
    """
    Negative test: Return with float as book ID
    Expected: Failure with validation error
    """
    success, message = return_book_by_patron("123456", 1.5)

    assert success == False
    assert "invalid" in message.lower()
```

# R5

Implemented R5: Late Fee Calculation API, but i need to expand test cases to cover functionality  because they currently only different type of inputs. **Functionality has been fully implemented**

**Added this extra test cases:**

```
def setup_method(self):
    """Setup test environment before each test"""
    init_database()
    # Setup borrowed books for testing
    from library_service import borrow_book_by_patron
    from datetime import datetime, timedelta


    # Borrow books and manipulate due dates for testing
    borrow_book_by_patron("111111", 1)  # Will be 5 days overdue
    borrow_book_by_patron("222222", 2)  # Will be 10 days overdue


    # Adjust due dates in database
    conn = get_db_connection()
    five_days_overdue = (datetime.now() - timedelta(days=5)).isoformat()
    ten_days_overdue = (datetime.now() - timedelta(days=10)).isoformat()
```

```
    conn.execute('''
        UPDATE borrow_records
        SET due_date = ?
        WHERE patron_id = ? AND book_id = ?
    ''', (five_days_overdue, "111111", 1))

    conn.execute('''
        UPDATE borrow_records
        SET due_date = ?
        WHERE patron_id = ? AND book_id = ?
    ''', (ten_days_overdue, "222222", 2))

    conn.commit()
    conn.close()


def test_no_late_fee_book_on_time(self):
    """
    Test: Book returned on time
    Expected: No late fee
    """
    from library_service import borrow_book_by_patron
    borrow_book_by_patron("333333", 3)  # Fresh borrow

    result = calculate_late_fee_for_book("333333", 3)

    assert result['status'] == 'success'
```

```
    assert result['fee_amount'] == 0.00
    assert result['days_overdue'] == 0


def test_late_fee_within_first_week(self):
    """
    Test: Book 5 days overdue (within first week)
    Expected: $0.50 per day fee
    """
    result = calculate_late_fee_for_book("111111", 1)

    assert result['status'] == 'success'
    assert result['days_overdue'] == 5
    assert result['fee_amount'] == 2.50  # 5 days * $0.50


def test_late_fee_beyond_first_week(self):
    """
    Test: Book 10 days overdue (beyond first week)
    Expected: First week at $0.50/day + remaining days at $1.00/day
    """
    result = calculate_late_fee_for_book("222222", 2)

    assert result['status'] == 'success'
    assert result['days_overdue'] == 10
    assert result['fee_amount'] == 6.50  # (7 * $0.50) + (3 * $1.00)
```

```python
def test_late_fee_maximum_cap(self):
    """
    Test: Book overdue long enough to exceed maximum fee
    Expected: Fee capped at $15.00
    """
    conn = get_db_connection()
    thirty_days_overdue = (datetime.now() - timedelta(days=30)).isoformat()

    conn.execute('''
        UPDATE borrow_records
        SET due_date = ?
        WHERE patron_id = ? AND book_id = ?
    ''', (thirty_days_overdue, "222222", 2))
    conn.commit()
    conn.close()

    result = calculate_late_fee_for_book("222222", 2)

    assert result['status'] == 'success'
    assert result['days_overdue'] == 30
    assert result['fee_amount'] == 15.00  # Maximum cap


def test_fee_for_non_borrowed_book(self):
    """
    Test: Calculate fee for book not borrowed by patron
    Expected: Error status
```

```
    """
    result = calculate_late_fee_for_book("444444", 1)

    assert result['status'] == 'error'
    assert 'not borrowed' in result['message'].lower()


def test_fee_precision(self):
    """
    Test: Verify fee amount has exactly 2 decimal places
    Expected: Fee amount with correct precision
    """
    result = calculate_late_fee_for_book("111111", 1)

    assert result['status'] == 'success'
    assert isinstance(result['fee_amount'], float)
    assert str(result['fee_amount']).split('.')[-1] == '50'  # Verify 2 decimal places


def test_multiple_overdue_books(self):
    """
    Test: Calculate fees for multiple overdue books
    Expected: Correct individual fees
    """
    result1 = calculate_late_fee_for_book("111111", 1)
    result2 = calculate_late_fee_for_book("222222", 2)
```

```
    assert result1['status'] == 'success' and result2['status'] == 'success'
    assert result1['fee_amount'] != result2['fee_amount']
    assert result1['days_overdue'] < result2['days_overdue']


def test_returned_book_late_fee(self):
    """
    Test: Calculate fee for already returned book
    Expected: Error status
    """
    from library_service import return_book_by_patron
    return_book_by_patron("111111", 1)

    result = calculate_late_fee_for_book("111111", 1)

    assert result['status'] == 'error'
    assert 'not borrowed' in result['message'].lower()


def test_future_due_date(self):
    """
    Test: Calculate fee for book not yet due
    Expected: Zero fee
    """
    from library_service import borrow_book_by_patron
    borrow_book_by_patron("555555", 3)  # Fresh borrow with future due date
```

```
    result = calculate_late_fee_for_book("555555", 3)

    assert result['status'] == 'success'
    assert result['fee_amount'] == 0.00
    assert result['days_overdue'] == 0


def test_fee_calculation_boundary_cases(self):
    """
    Test: Fee calculation at boundary conditions
    Expected: Correct fee amounts
    """
    conn = get_db_connection()

    # Test exactly 7 days overdue
    seven_days_overdue = (datetime.now() - timedelta(days=7)).isoformat()
    conn.execute('''
        UPDATE borrow_records
        SET due_date = ?
        WHERE patron_id = ? AND book_id = ?
    ''', (seven_days_overdue, "111111", 1))
    conn.commit()

    result = calculate_late_fee_for_book("111111", 1)

    assert result['status'] == 'success'
    assert result['days_overdue'] == 7
```

```
    assert result['fee_amount'] == 3.50  # 7 days * $0.50
```

# R6

Implemented R6: Book Search Functionality, removed frontend sign saying function not implemented. **Functionality has been fully implemented**

**Added these extra tests:**

```
def setup_method(self):
    """Setup test data for multiple results"""
    from database import get_db_connection
    conn = get_db_connection()
    # Add books with similar titles/authors for testing multiple results
    conn.execute('''
        INSERT INTO books (title, author, isbn, total_copies, available_copies)
        VALUES
        ("The Book of Python", "John Smith", "1111111111111", 1, 1),
        ("Python Programming", "John Smith", "2222222222222", 1, 1),
        ("Learning Python", "Jane Smith", "3333333333333", 1, 1)
    ''')
    conn.commit()
    conn.close()


  def test_search_multiple_title_matches(self):
```

```
    """
    Test: Search term matching multiple titles
    Expected: Should return all matching books sorted alphabetically
    """
    result = search_books_in_catalog("Python", "title")

    assert len(result) >= 3
    assert all("python" in book['title'].lower() for book in result)
    # Verify alphabetical sorting
    titles = [book['title'] for book in result]
    assert titles == sorted(titles)


def test_search_multiple_author_matches(self):
    """
    Test: Search for author with multiple books
    Expected: Should return all books by author
    """
    result = search_books_in_catalog("Smith", "author")

    assert len(result) >= 3
    assert all("smith" in book['author'].lower() for book in result)


def test_search_with_special_characters(self):
    """
    Test: Search with special characters and spaces
```

```
    Expected: Should handle special characters appropriately
    """
    result = search_books_in_catalog("Book & Python!", "title")

    assert isinstance(result, list)
    assert any("Book of Python" in book['title'] for book in result)


def test_search_with_unicode_characters(self):
    """
    Test: Search with Unicode characters
    Expected: Should handle Unicode characters appropriately
    """
    # Add a book with Unicode characters
    from database import get_db_connection
    conn = get_db_connection()
    conn.execute('''
        INSERT INTO books (title, author, isbn, total_copies, available_copies)
        VALUES ("El Código Python", "José García", "4444444444444", 1, 1)
    ''')
    conn.commit()
    conn.close()

    result = search_books_in_catalog("código", "title")
    assert len(result) >= 1
    assert any("Código" in book['title'] for book in result)
```

```python
class TestSearchBooksEdgeCases:
    """Test search functionality edge cases"""

    def test_search_very_long_search_term(self):
        """
        Test: Search with very long search term
        Expected: Should handle gracefully
        """
        long_term = "a" * 500
        result = search_books_in_catalog(long_term, "title")

        assert isinstance(result, list)

    def test_search_with_sql_injection_attempt(self):
        """
        Test: Search with SQL injection attempt
        Expected: Should handle safely
        """
        malicious_input = "'; DROP TABLE books; --"
        result = search_books_in_catalog(malicious_input, "title")

        assert isinstance(result, list)
        # Verify database is still intact
        all_books = search_books_in_catalog("", "title")
```

```
        assert len(all_books) > 0


    def test_search_mixed_type(self):
        """
        Test: Search with numeric and text mixed
        Expected: Should handle mixed content appropriately
        """
        result = search_books_in_catalog("Python 3", "title")
        assert isinstance(result, list)


    def test_search_results_structure(self):
        """
        Test: Verify search results contain all required fields
        Expected: Each result should have all catalog display fields
        """
        result = search_books_in_catalog("Python", "title")
        required_fields = ['id', 'title', 'author', 'isbn', 'total_copies', 'available_copies']

        for book in result:
            assert all(field in book for field in required_fields)
            assert isinstance(book['id'], int)
            assert isinstance(book['total_copies'], int)
            assert isinstance(book['available_copies'], int)


class TestSearchPerformance:
```

```
"""Test search functionality performance with large dataset"""


def setup_method(self):
    """Setup large dataset for performance testing"""
    from database import get_db_connection
    conn = get_db_connection()
    # Add 100 sample books
    for i in range(100):
        conn.execute('''
            INSERT INTO books (title, author, isbn, total_copies, available_copies)
            VALUES (?, ?, ?, 1, 1)
        ''', (
            f"Test Book {i}",
            f"Author {i}",
            f"{str(i).zfill(13)}"
        ))
    conn.commit()
    conn.close()


def test_search_large_dataset(self):
    """
    Test: Search performance with large dataset
    Expected: Should return results quickly and correctly
    """
    import time
```

```
    start_time = time.time()


    result = search_books_in_catalog("Test", "title")


    end_time = time.time()
    search_time = end_time - start_time


    assert len(result) >= 100  # Should find all test books
    assert search_time < 1.0  # Should complete within 1 second
```

# R7

Implemented R7: Patron Status Report, added the frontend and the implementation of the functionality. **Functionality has been fully implemented**

**Added these test cases**

```
def setup_method(self):
    """Setup test environment before each test"""
    init_database()
    # Setup test data
    self._setup_test_data()


  def _setup_test_data(self):
    """Create test data for patron status testing"""
    # Create a patron with multiple borrowed books
    patron_id = "111111"
    borrow_book_by_patron(patron_id, 1)  # Borrow first book
    borrow_book_by_patron(patron_id, 2)  # Borrow second book


    # Create an overdue book scenario
    conn = get_db_connection()
    past_date = (datetime.now() - timedelta(days=20)).isoformat()
    conn.execute('''
        UPDATE borrow_records
        SET due_date = ?
        WHERE patron_id = ? AND book_id = ?
    ''', (past_date, patron_id, 1))
    conn.commit()
    conn.close()
```

```python
def test_patron_status_valid_patron_comprehensive(self):
    """
    Positive test: Full patron status with borrowed and overdue books
    Expected: Complete status report with all required fields
    """
    result = get_patron_status_report("111111")

    assert result['status'] == 'success'
    assert len(result['currently_borrowed_books']) == 2
    assert result['total_late_fees_owed'] > 0
    assert result['number_of_books_borrowed'] == 2
    assert isinstance(result['borrowing_history'], list)


def test_patron_status_borrowed_books_details(self):
    """
    Test: Verify borrowed books contain all required details
    Expected: Complete book information with due dates
    """
    result = get_patron_status_report("111111")

    for book in result['currently_borrowed_books']:
        assert all(key in book for key in ['book_id', 'title', 'author', 'due_date'])
        assert isinstance(book['due_date'], str)
        assert isinstance(book['book_id'], int)
```

```python
def test_patron_status_late_fees_calculation(self):
    """
    Test: Verify late fees are calculated correctly
    Expected: Accurate late fee calculation
    """
    result = get_patron_status_report("111111")

    assert isinstance(result['total_late_fees_owed'], (int, float))
    assert result['total_late_fees_owed'] <= 15.00  # Maximum per book
    assert result['total_late_fees_owed'] >= 0.00


def test_patron_status_borrowing_history_complete(self):
    """
    Test: Verify borrowing history contains all transactions
    Expected: Complete borrowing history with details
    """
    result = get_patron_status_report("111111")

    assert isinstance(result['borrowing_history'], list)
    for record in result['borrowing_history']:
        assert all(key in record for key in [
            'book_id', 'title', 'author', 'borrow_date'
        ])
        assert isinstance(record['borrow_date'], str)
```

```
def test_patron_status_json_structure(self):
    """
    Test: Verify JSON structure of response
    Expected: Well-formed JSON with all required fields
    """
    result = get_patron_status_report("111111")

    required_fields = [
        'status',
        'patron_id',
        'currently_borrowed_books',
        'total_late_fees_owed',
        'number_of_books_borrowed',
        'borrowing_history'
    ]

    assert all(field in result for field in required_fields)
    assert isinstance(result['currently_borrowed_books'], list)
    assert isinstance(result['borrowing_history'], list)


def test_patron_status_no_borrowed_books(self):
    """
    Test: Patron with no borrowed books
    Expected: Empty lists and zero values
```

```
    """
    result = get_patron_status_report("999999")

    assert result['status'] == 'success'
    assert len(result['currently_borrowed_books']) == 0
    assert result['total_late_fees_owed'] == 0.00
    assert result['number_of_books_borrowed'] == 0
    assert len(result['borrowing_history']) == 0


def test_patron_status_overdue_books_identification(self):
    """
    Test: Verify overdue books are properly identified
    Expected: Overdue status indicated in borrowed books
    """
    result = get_patron_status_report("111111")

    overdue_found = False
    for book in result['currently_borrowed_books']:
        if datetime.fromisoformat(book['due_date']) < datetime.now():
            overdue_found = True
            break

    assert overdue_found == True


def test_patron_status_data_types(self):
```

```
    """
    Test: Verify correct data types in response
    Expected: All fields have correct data types
    """
    result = get_patron_status_report("111111")

    assert isinstance(result['patron_id'], str)
    assert isinstance(result['currently_borrowed_books'], list)
    assert isinstance(result['total_late_fees_owed'], (int, float))
    assert isinstance(result['number_of_books_borrowed'], int)
    assert isinstance(result['borrowing_history'], list)


def test_patron_status_invalid_patron_format(self):
    """
    Negative test: Invalid patron ID format
    Expected: Error status with message
    """
    result = get_patron_status_report("12345")  # Too short

    assert result['status'] == 'error'
    assert 'invalid patron id' in result['message'].lower()


def test_patron_status_special_characters(self):
    """
    Negative test: Patron ID with special characters
```

```
    Expected: Error status with message
    """
    result = get_patron_status_report("12@456")


    assert result['status'] == 'error'
    assert 'invalid patron id' in result['message'].lower()
```