Student Instructions - Assignment 2

Project: Library Management System - Complete Implementation & CI/CD

Project Overview

Building upon Assignment 1, you will now complete the Library Management System implementation, create comprehensive test suites, leverage AI tools for test generation, and set up professional CI/CD workflows using GitHub Actions.

Learning Objectives

By the end of this assignment, you will be able to:

- Implement complete business logic functions following specifications
- Create comprehensive test suites for both existing and new functionality
- Use Large Language Models (LLMs) to generate effective test cases
- Set up Continuous Integration/Continuous Deployment (CI/CD) pipelines
- Implement professional project documentation with status badges

Tasks to Complete

You are required to complete the following 4 main tasks:

1. **Complete Function Implementation** - Implement all remaining TODO functions
2. **Comprehensive Test Suite Development** - Create tests for all functionality (old + new)
3. **AI-Assisted Test Generation** - Use LLMs to generate additional test cases
4. **CI/CD Pipeline Setup** - Deploy to GitHub with automated testing and status badges

Task 1: Complete Function Implementation (40%)

1.1 Implement Missing Functions

Complete the implementation of all TODO functions in `library_service.py`:

1.1.1 `return_book_by_patron(patron_id: str, book_id: int)`

- **Requirements**: Implement R4 (Book Return Processing)
- **Input Validation**: 6-digit patron ID, positive integer book_id
- **Business Logic**:
  - Verify book is actually borrowed by the patron
  - Update return_date in borrow_records table
  - Increment available_copies for the book
  - Handle edge cases (book not borrowed, invalid IDs)
- **Return**: `Tuple[bool, str]` - (success, message)

1.1.2 `calculate_late_fee_for_book(book_id: int, patron_id: str)`

- **Requirements**: Implement R5 (Late Fee Calculation API)
- **Input Validation**: Positive integer book_id, 6-digit patron_id
- **Business Logic**:
  - Calculate days overdue (if any)
  - Apply late fee rate: $1.00 per day overdue
  - Return 0.00 if book returned on time or early
  - Handle edge cases (book not found, not borrowed by patron)
- **Return**: `Tuple[bool, str, float]` - (success, message, fee_amount)

1.1.3 `search_books_in_catalog(query: str, search_type: str)`

- **Requirements**: Implement R6 (Book Search Functionality)
- **Input Validation**: Non-empty query string, valid search_type ('title', 'author', 'isbn')
- **Business Logic**:
  - Perform case-insensitive partial matching for title/author
  - Exact matching for ISBN
  - Return list of matching books with availability info
- **Return**: `Tuple[bool, str, List[Dict]]` - (success, message, book_list)

1.1.4 `get_patron_status_report(patron_id: str)`

- **Requirements**: Implement R7 (Patron Status Report)
- **Input Validation**: 6-digit patron ID format
- **Business Logic**:
  - List all currently borrowed books
  - Calculate total late fees owed
  - Show due dates and overdue status
- **Return**: `Tuple[bool, str, Dict]` - (success, message, status_report)

1.2 Implementation Guidelines

- Follow existing code patterns and style
- Maintain consistent error handling
- Use proper type hints
- Include comprehensive docstrings
- Ensure all functions pass basic functionality tests

1.3 Testing Your Implementation

```
# Test your implementations
python -m pytest tests/ -v -k "not test_unimplemented"
```

Task 2: Comprehensive Test Suite Development (30%)

2.1 Update Existing Tests

- **Fix Assignment 1 bugs**: Update tests to handle the corrected ISBN validation and borrowing limit bugs
- **Extend coverage**: Add more edge cases to existing test files
- **Integration testing**: Create tests that verify multiple functions work together

2.2 Create Tests for New Functions

For each newly implemented function, create comprehensive test files:

2.2.1 `tests/test_return_book_by_patron.py`

- **Positive cases**: Valid return scenarios, updating database state
- **Negative cases**: Invalid patron ID, book not borrowed, already returned
- **Edge cases**: Concurrent returns, database errors
- **Minimum**: 8-10 test cases

2.2.2 `tests/test_calculate_late_fee_for_book.py`

- **Positive cases**: On-time returns, various overdue scenarios
- **Negative cases**: Invalid inputs, book not found, calculation errors
- **Edge cases**: Same-day returns, maximum overdue periods
- **Minimum**: 8-10 test cases

2.2.3 `tests/test_search_books_in_catalog.py`

- **Positive cases**: Title search, author search, ISBN search
- **Negative cases**: No matches, invalid search type, empty query
- **Edge cases**: Special characters, case sensitivity, partial matches
- **Minimum**: 10-12 test cases

2.2.4 `tests/test_get_patron_status_report.py`

- **Positive cases**: Active borrower, no books borrowed, mixed status
- **Negative cases**: Invalid patron ID, non-existent patron
- **Edge cases**: Recently returned books, multiple overdue books
- **Minimum**: 8-10 test cases

2.3 Integration Test Suite

Create `tests/test_integration.py` with end-to-end scenarios:

- Complete borrow-to-return workflow
- Search and borrow workflow
- Late fee calculation with actual overdue books
- Patron status with real borrowing history

Task 3: AI-Assisted Test Generation (15%)

3.1 LLM Test Case Generation

Use Large Language Models (ChatGPT, Claude, Copilot, etc.) to generate additional test cases:

3.1.1 Generate Edge Cases

- **Prompt engineering**: Create effective prompts to generate comprehensive edge cases
- **Documentation**: Save your prompts and LLM responses in `docs/llm_test_generation.md`
- **Implementation**: Add the best generated test cases to your test suite

3.1.2 Example Prompt Template

```
I have a Python function that [describe function]. The function should
handle [describe requirements].
Generate 5 comprehensive test cases including edge cases that might break
this function.
Format as pytest test functions with clear test names and assertions.
```

3.2 LLM-Generated Test Requirements

- Generate **at least 10 additional test cases** using LLM assistance
- **Document the process**: Include prompts used and LLM responses
- **Validate and adapt**: Ensure generated tests are accurate and valuable
- **Integration**: Incorporate the best tests into your main test suite

3.3 LLM Usage Documentation

Create `docs/llm_test_generation.md` containing:

- LLM platform used (ChatGPT, Claude, etc.)
- Exact prompts used for test generation
- Raw LLM responses
- Analysis of which generated tests were most valuable
- Lessons learned about effective prompt engineering

Task 4: CI/CD Pipeline Setup (15%)

4.1 GitHub Repository Setup

4.1.1 Repository Creation

- Create a **public** GitHub repository named `cisc327-library-management-a2-[your-student-id]`
- Initialize with proper `.gitignore` for Python projects
- Include comprehensive README.md with project description

4.1.2 Repository Structure

```

```

```
your-repo/
├── .github/
│   └── workflows/
│       └── tests.yml
├── library_service.py
├── database.py
├── app.py
├── tests/
│   ├── test_*.py files
├── docs/
│   └── llm_test_generation.md
├── requirements.txt
├── README.md
└── .gitignore
```

4.2 GitHub Actions Configuration

4.2.1 Create Workflow File

Create `.github/workflows/tests.yml`:

```yaml
name: Library Management Tests

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.8, 3.9, '3.10']

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with:
        python-version: ${{ matrix.python-version }}

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install pytest pytest-cov
```

```
      - name: Run tests with coverage
        run: |
          pytest tests/ -v --cov=library_service --cov-report=xml

      - name: Upload coverage to Codecov
        uses: codecov/codecov-action@v3
        with:
          file: ./coverage.xml
```

4.2.2 Test Multiple Python Versions

- Configure matrix testing for Python 3.8, 3.9, and 3.10
- Ensure all tests pass on all versions
- Document any version-specific issues

4.3 Status Badges Implementation

4.3.1 Add Test Status Badge

Update your README.md with status badges:

```
# Library Management System - Assignment 2

[![Tests](https://github.com/[username]/[repo-
name]/workflows/Library%20Management%20Tests/badge.svg)]
(https://github.com/[username]/[repo-name]/actions)
[![codecov](https://codecov.io/gh/[username]/[repo-
name]/branch/main/graph/badge.svg)](https://codecov.io/gh/[username]/[repo-
name])

[Rest of your README content]
```

4.3.2 Professional README Requirements

Your README.md must include:

- Project title and description
- Test status badges
- Installation instructions
- Usage examples
- Test coverage information
- Contributing guidelines
- Your name and student ID

4.4 Continuous Integration Requirements

- **All tests must pass** in GitHub Actions
- **Test coverage** should be at least 90%

- **Multiple Python versions** support
- **Automated testing** on every push and pull request
- **Badge status** correctly reflects current test status

Deliverables & Submission

Submission Requirements

Create a comprehensive report `A2_LastName_last4digitID.md` containing:

1. **Implementation Summary**
    - List of completed functions
    - Key implementation challenges and solutions
    - Testing strategy for new functions
2. **Test Suite Analysis**
    - Total number of test cases created
    - Coverage analysis and improvements from Assignment 1
    - Integration testing approach
3. **LLM-Assisted Development Report**
    - LLM platforms used and effectiveness
    - Best prompts discovered for test generation
    - Analysis of AI-generated vs manually-written tests
4. **CI/CD Setup Documentation**
    - GitHub repository URL
    - Workflow configuration choices
    - Badge implementation and current status
    - Any challenges encountered with GitHub Actions

GitHub Repository Submission

Your final GitHub repository must contain:

- ✅ All implemented functions working correctly
- ✅ Comprehensive test suite (minimum 50 total test cases)
- ✅ Working GitHub Actions workflow
- ✅ Green test status badge in README
- ✅ Professional documentation
- ✅ LLM test generation documentation

Grading Criteria

- **Function Implementation (40%)**: Correctness, code quality, following specifications
- **Test Suite Quality (30%)**: Coverage, edge cases, integration tests
- **AI-Assisted Development (15%)**: Effective LLM usage, documentation, integration

- **CI/CD Pipeline (15%)**: Working GitHub Actions, status badges, professional setup

Due Date

[Insert due date - typically 2-3 weeks from Assignment 1]

Academic Integrity

- You may use LLMs for test generation as specified in Task 3
- All LLM usage must be documented as required
- Function implementations must be your own original work
- Collaboration on implementation is not permitted
- You may discuss testing strategies with classmates

Getting Help

- Office hours: [Insert times]
- Discussion forum: [Insert link]
- Email: [Insert email]

Good luck with Assignment 2! This assignment will give you hands-on experience with modern software development practices including AI-assisted development and professional CI/CD workflows.