## **Unit Test Case Flow Document**

## **Intelligent Book Management System**

This document provides detailed information on the unit test cases implemented for the Intelligent Book Management System. These tests ensure the reliability and correctness of the API endpoints and core functionalities.

**1. Test Suite Overview**

The test suite is built using pytest, a Python testing framework. Tests are located in the tests/ directory and are organized into modules based on the API components (e.g., test\_auth.py, test\_api.py).

**2. Test Environment Setup**

* **Test Database:**
  + A separate, in-memory SQLite database is used for testing to avoid affecting the development or production database.
  + This database is initialized and populated with sample data using pytest fixtures.
* **Flask Test Client:**
  + A Flask test client is used to simulate HTTP requests to the API endpoints.
  + This allows testing of API behavior without running a full web server.
* **Environment Variables:**
  + Environment variables required for testing are set within the test functions or fixtures.

**3. Test Fixtures**

* **test\_client:**
  + Creates a Flask test client instance.
  + Used to make HTTP requests to the API endpoints.
* **init\_database:**
  + Initializes the test database.
  + Creates tables based on the SQLAlchemy models.
  + Populates the database with sample data (e.g., books, users, reviews).
  + Ensures a clean database state before each test.
* **authenticated\_header:**
  + Creates a JWT authorization header for a valid user.
  + Used for testing protected API endpoints.

**4. Test Modules and Test Cases**

**4.1. tests/test\_auth.py (Authentication Tests)**

* **test\_register\_user:**
  + Tests the /auth/register endpoint.
  + Verifies successful user registration.
  + Checks for duplicate username errors.
  + Checks for validation errors on bad input.
* **test\_login\_user:**
  + Tests the /auth/login endpoint.
  + Verifies successful user login and JWT generation.
  + Checks for invalid credentials errors.

**4.2. tests/test\_api.py (API Endpoints Tests)**

* **test\_get\_all\_books:**
  + Tests the /api/books (GET) endpoint.
  + Verifies that all books are returned.
  + Checks the structure and content of the response.
* **test\_get\_book\_by\_id:**
  + Tests the /api/books/{book\_id} (GET) endpoint.
  + Verifies that a specific book is returned.
  + Checks for 404 errors for non-existent books.
* **test\_create\_book:**
  + Tests the /api/books (POST) endpoint.
  + Verifies successful book creation.
  + Verifies validation of input data.
* **test\_update\_book:**
  + Tests the /api/books/{book\_id} (PUT) endpoint.
  + Verifies successful book updates.
  + Checks for 404 errors for non-existent books.
  + Verifies data validation.
* **test\_delete\_book:**
  + Tests the /api/books/{book\_id} (DELETE) endpoint.
  + Verifies successful book deletion.
  + Checks for 404 errors for non-existent books.
* **test\_get\_book\_reviews:**
  + Tests the /api/books/{book\_id}/reviews (GET) endpoint.
  + Verifies that the reviews for a book are returned.
  + Checks for 404 errors for non-existent books.
* **test\_create\_book\_review:**
  + Tests the /api/books/{book\_id}/reviews (POST) endpoint.
  + Verifies that reviews are correctly created.
  + Tests validation of input data.
* **test\_get\_review\_by\_id:**
  + Tests the /api/reviews/{review\_id} (GET) endpoint.
  + Verifies that the reviews are correctly retrieved.
  + Checks for 404 errors for non-existent reviews.
* **test\_update\_review:**
  + Tests the /api/reviews/{review\_id} (PUT) endpoint.
  + Verifies that reviews are correctly updated.
  + Checks for 404 errors for non-existent reviews.
* **test\_delete\_review:**
  + Tests the /api/reviews/{review\_id} (DELETE) endpoint.
  + Verifies that reviews are correctly deleted.
  + Checks for 404 errors for non-existent reviews.
* **test\_generate\_book\_summary:**
  + Tests the /api/books/{book\_id}/summary (POST) endpoint.
  + Verifies that a summary is returned.
  + Checks for 404 errors for non-existent books.
  + Tests for LLM communication errors.
* **test\_get\_book\_recommendations:**
  + Tests the /api/books/{book\_id}/recommendations (GET) endpoint.
  + Verifies that recommendations are returned.
  + Checks for 404 errors for non-existent books.
  + Tests for LLM communication errors.

**5. Test Assertions**

* **Status Code Assertions:**
  + Verifies the HTTP status code of the API responses (e.g., assert response.status\_code == 200).
* **JSON Response Assertions:**
  + Verifies the structure and content of the JSON responses (e.g., assert data['title'] == 'Expected Title').
* **Database Assertions:**
  + Verifies that data is correctly stored or modified in the database.

**6. Running Tests**

| 6.1 Run all tests: Bash Pytest  6.2 Run a specific test module: Bash pytest tests/test\_auth.py  6.3 Run a specific test function: Bash pytest tests/test\_api.py::test\_get\_all\_books  6.4 View test coverage: Bash pytest --cov=app |
| --- |