Assignment-2: Report

Part 1: Completing the Implementation:

My first step was to fix a critical bug that was letting patrons borrow one too many books. With that sorted, I moved on to implementing the four main functions that were still needed. I built out the complete book return process, which now handles patron validation, verifies the borrowing record, automatically calculates any late fees, and updates the number of available copies. I also created a more flexible catalog search that allows for partial, case insensitive searches on titles and authors, plus exact matching for ISBNs. Furthermore, I implemented a comprehensive patron status report that gives a full overview of a users account, including all their borrowed books, due dates, total late fees, and complete borrowing history. All the new code follows existing architecture and passed all quality checks, completing the project.

Part 3: AI-Assisted Test Generation:

I use ChatGPT to generate the test cases. The full extent of my conversation is here: https://chatgpt.com/share/68ed9f1c-8388-8010-9886-22080c5ec75b

I did not insert my conversation or the test cases provided by the AI in the PDF file since it would take 12 pages. Nevertheless, the link should forward you to the full conversation. Please let me know if there are any issues with the link.

Part 3: Test-Case Comparison & Analysis:

First, I analyzed my own test cases. My test suite is a strong example of integration testing, as it focuses on making sure that different parts of the business logic work together to complete end to end user scenarios. They are clear, easy to read, and mainly focus on the important essential functionality along with critical business rules like borrowing limits. My tests provide excellent depth and are best seen as integration tests. They do a great job of ensuring that the main features of the library system are working correctly in combination, giving me confidence that the overall system is stable and meets its primary requirements.

Next, I looked at the AI-generated test cases. This suite is a highly detailed example of unit and component testing. It rigorously validates the correctness of each individual function in isolation, with a heavy emphasis on edge cases, input validation, and boundary conditions. The suite is meticulously organized by functional requirements and uses pytest.mark.parametrize to efficiently test dozens of permutations for a single function, making it far more thorough for unit validation. It covers deep edge cases that I might have missed, such as precisely testing late fee calculations at the 7 and 8-day marks, using threading to check for race conditions, and ensuring a core rule like available copies never going negative is always maintained. This suite provides exceptional depth, excelling at validating the internal logic of each business function against all specified constraints and catching bugs at a much more granular level. Nevertheless, the limitations of AI are prevalent here. ChatGPT could not create completely integrable code. Many issues like formatting, running and the division/organization of these test cases is prevalent.

In my final findings, I realized the two test suites are not competing against each other, rather they are highly complementary and represent two essential layers of a robust testing strategy. My test cases are primarily meant to verify that features work together in an integrated way, using scenario-based user workflows to build confidence in the overall systems behavior. The AI-generated test cases, on the other hand, aim to verify that individual functions are correct by using exhaustive, specification-based validation, which provides extreme thoroughness and catches granular bugs. Essentially, my tests answer the question, "Do the features work correctly from end to end?" while the AI's tests answer, "Does each piece of business logic handle every possible input and situation correctly?". I think my test suite is great and perfect for its purpose, but a professional software project would ideally have both a unit test suite like the AIs to guarantee the reliability of the core logic, and integration testing like mine to make sure all the pieces fit together perfectly.