Design Rationale — Mini Library Management System

Design Rationale — Mini Library Management System

Course: Object-Oriented Programming 1 Assignment: Mini Library Management System Student: Buhari Bangura Semester: September 2025 – February 2026

Overview The Mini Library Management System demonstrates core programming concepts and data structure selection in Python. The system supports adding, searching, updating, deleting, borrowing, and returning books while enforcing logical constraints such as a maximum of three books borrowed per member.

Data Structure Choices • Books — Dictionary keyed by ISBN: ISBNs are unique identifiers; using a dictionary provides O(1) lookup, simple updates, and straightforward containment checks. Each ISBN maps to a nested dictionary with attributes (title, author, genre, total_copies, available_copies). • Members — List of dictionaries: Each member record contains member_id, name, email, and borrowed_books. A list provides straightforward append/search semantics for this assignment; for scalability a dict keyed by member_id is a recommended future improvement. • Genres — Tuple: Genres are a fixed set of allowed values. A tuple's immutability prevents accidental modification and simplifies validation.

Functional Design & Responsibilities The system follows a modular design: functions encapsulate single responsibilities (add_book, search_books, update_book, delete_book, add_member, borrow_book, return_book, etc.). This keeps logic testable and easy to reason about. Borrowing updates available_copies and member borrowed_books in a single transaction-like operation to avoid inconsistency.

Design Constraints & Validation • ISBNs and member IDs are unique — duplication checks prevent data conflicts. • Genre validation uses the immutable genre tuple. • A member cannot borrow more than three books. • Books cannot be deleted if any copies are currently borrowed. • Reducing total_copies below the number already borrowed is disallowed.

Testing Strategy Simple assert-based tests validate key behaviors: adding and updating items, borrowing limits, borrowing when copies are exhausted, and deletion restrictions. These tests are lightweight and satisfy the assignment testing requirement without external frameworks.

Extendibility & Alternatives The procedural design is intentionally simple for clarity, but it can be refactored to OOP with Book, Member, and Library classes for larger systems. Future improvements include persisting data (SQLite/JSON), replacing member list with a dict for O(1) lookups, adding due dates/fines, and a user interface.

Conclusion This design balances simplicity, correctness, and extensibility—showcasing practical use of Python data structures while meeting all assignment requirements.