Level 2 - Kyle de Vos (KD23040008523)
L2T13 – Capstone Project I
Date: 23 July 2023

Software Requirements Document

Bookstore Database Management System for Book Stock Control

Table of Contents

1. Preface

1.1 Version History

This is the first designed *Book Management System* with no prior system in place. Version History is none.

1.2 Intended Audience

This document and its accompanying project form a Capstone Project simulating a *Book Store Management System* for the HyperionDev Software Engineering Course. It is not intended for real-world use. The audience of this project are the HyperionDev markers and reviewers. Docstrings for Modules, Classes and Methods are worded to describe thought processes or design styles in addition to function.

1.3 Versions Overview

First Version. Includes full CRUD operations and functionality to interact with database, retrieve and validate user input and return data to UI. Functions allowing user to use the 'Main Menu' are included with setup of the console UI. This version includes the modules and classes required for the above.

2. Project Introduction

2.1 Purpose

The simulated purpose of this project is a Book Stock Management System for a bookstore with an included database. This project aims to allow access to this database and perform CRUD operations on its entities, utilising a terminal as its UI. Finance Tracking Functionality has not been incorporated. The project has elements of dependency de-coupling and the SOLID Principles to maximise re-use and scalability, but is designed to meet requirements of task for a table handling only 'book' elements.

2.2 Function and Business Integration

This project is simulated (and assumed) to assist a business with no prior software-based system. It is assumed that the business has traditional 'paper' systems in place. Requirements Elicitation did not show evidence of a current system.

This project aims to replace a potential 'paper' stock management system allowing for simple CRUD operations only. The simulated business would be

intended to perform financial operations utilising data from this project (more specifically stored within its database).

2.3 Project Scope

This project will look to manage Book Stock for a simulated Book Store Business. A database is included and will allow a user to perform addition, manipulation and deletion of Books stored in the database. Additional classes will be added to ensure this project is scalable and follows the SOLID Principles [1].

2.4 References

[0] – Software Requirements Document compared to style from KrazyTech.
Ravi Bandakkananavr, Software Requirements Specification document with example. Last Accessed on 4 July 2023.
Available from: https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database

[1] – Stephen Watts. The Importance of SOLID Design Principles.
Last Accessed on 4 July 2023
Available from: https://www.bmc.com/blogs/solid-design-principles/#:~:text=SOLID%20is%20an%20acronym%20that,some%20important%20benefits%20for%20developers.

3. Glossary

- SOLID Principle - A set of 5 design principles used by developers to ensure a program is designed in an agile manner – allowing for easy modifications and scalability. [1]

- UI – User-Interface – Manner in which the application is presented to the user.

- CRUD Operations – Basic actions performed against a Database as: **C**reate, **R**ead, **U**pdate and **D**elete.

4. User Requirements Definition

4.1 Functional Requirements

- Main Menu incorporated into terminal interface to allow user to select from options
- Add new books to the database
- Update (change) details for an existing book stored in database

- Delete a book from the database
- Search for a specified book within the database – assumed to allow search for book by title, author and or/quantity
- Show all Books in database (added as extra)
- Menu Option to exit application

4.2 Non-Functional Requirements

- **Usability** – Main Menu is concise with clear menu options. Information returned from the system should be readable.
- **Reliability** – System should validate inputs to ensure data written to database or accepted from user is correct (and in correct format). Program must be tested thoroughly ensuring invalid input does not crash program.
- **Performance** – Program should aim to minimise delays in data retrieval and processing of large amounts of information.
- **Security** – Current Project does not include requirements to authenticate user but should be considered and recommended for future versions. Data must be checked before being entered into the database.
- **Implementation Constraints**: This project is a simulation business application that is created only using Python and SQLite.

4.3 Product Standards

This Project is designed to work with 'books' recording the attributes id, title, author and quantity. Data for each of these values is required for the database and books in stock must conform to these attributes at minimum. Additional attributes (such as an ISBN number or year of publication and version) are not included and would need to be added in later versions of this system – noting that data corrections for books present in database would be needed.

5. System Requirements and Specification

5.1 Functional Requirements

- Display all input requests and data output to terminal console window.
- Display Main Menu to user requiring only Integer input retrieval matching option list. Loop Main Menu until application is closed.
- Further user requests for application usage use only Integer Inputs matching sub-menu options.
- User Inputs for menus and sub-menus must have validation for empty inputs, characters, decimal numbers and range checks.
- Strong consideration of class and module separations making each as generic as possible

    - Main Menu controlled by an Application Controller
    - View Module and/or class to easily change user view output type from console to HTML for application evolution
    - Application Entity of 'Book' to be created in an Entity Class with sub-classes handling separate CRUD Operations
    - Class to control Field Names, types, restrictions, etc must have centralised control of these attributes and should be accessible from only one place within application.
    - Module to abstract all database query creations and connection management away from rest of application
    - Book id's act as primary key and must be set and incremented by program not allowing user to make any changes to book id
    - Classes to match 'Book' Entity to appropriate database controllers called by Application Controller
- Error Handling for Database connections and incorrect queries
- Error Handling for invalid user-input

5.2 Non-Functional Requirements

- **Usability:**
    - User Interface in Console Window must use short, concise statements with options appearing on a new line. Data from Database presented in a tabular, consistent form with field names as column headings.
    - Errors from user-input must explain cause of error in a non-complex manner and allow user opportunity to re-enter data
    - Program errors should be marked as such and printed as "Error Log" with a short description
    - All technical aspects (such as database control or query generation) must be correctly abstracted away from user
    - All Modules, Classes and Functions must have completed Docstrings (as is expected) for developer reference

- **Reliability:** Program must be designed with error handling for input and database errors and recover from these successfully. Field and Data Types specified by 'Field Control' must be adhered to, to ensure behaviour of application is predictable. Program must run continuously allowing for repetitive CRUD Operations

- **Performance:** Program is expected to run smoothly with minimal delay between Menu prints, input requests, Book Entity creations and persistence matching, database query executions and return of data back to View Class for render to user.

- **Security:** Field Names and Types must be controlled by one class in Entity Module and should not be changed or modified in any other part of application. User access should not allow modifications of Entity Fields. Primary Key values must be set and monitored by system - no functionality for user to change a primary key value.

6. System Architecture

6.1 Architecture Patterns Overview

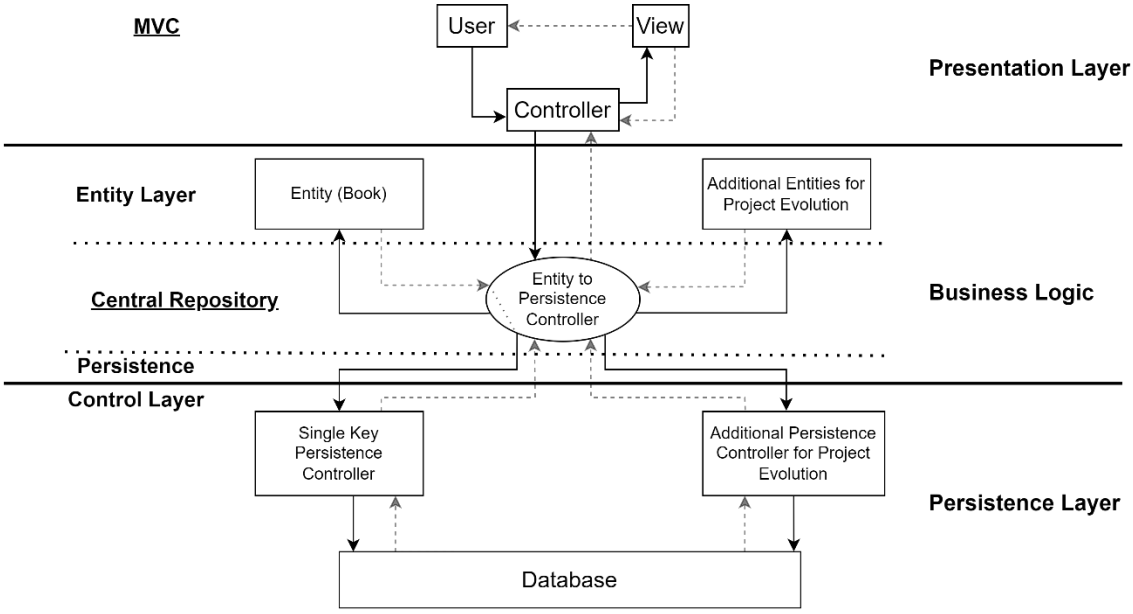

Fig 1. – System Architecture Overview

The Book Management System will use a composite Architecture Design consisting of MVC, Layered and Central Repository (with inner layered) Architectures (see Fig 1. Above) described as:

1) **Layered Architecture** (headings on right-hand side of Fig 1.)
   The system is broken down into three main layers as the *Presentation Layer, Business Logic Layer* and *Persistence Layer.*
   - The **Presentation Layer** has 'Main Menu' responsibility and control, user-interaction, and display of data to user.
   - The **Business Logic layer** holds Entities (and business rules for correct definition) and a central application controller responsible for matching an Entity to a Persistence Control class. This layer may also request and validate user-input as well as render views (display data) to the user where needed.

- The **Persistence Layer** holds all modules and classes responsible for database creation, connection, query execution and data retrieval from database.

2) **MVC Architecture**

System uses a variant of the MVC architecture with the 'Business Logic' and 'Persistence' Layers of the main Layered Architecture acting as the 'Model'. The Presentation Layer houses this MVC architecture with responsibilities as:

- **Controller** – Retrieve and validate user-input, determine or store database and table(s) names for application and pass correct instruction to Central Application Controller based on user-inputs.
- **View –** Receive information from Controller and display in desired form to user. Current Application implementation will print data to user in terminal console of application window. View may also format requests for user input and pass data back to Controller.

3) **Central Repository Architecture**

The Business Logic Layer will utilise a Central Application Controller matching an initialised Entity to a Persistence Controller using instructions received from Controller in Presentation Layer based on desired user action.

Entity – An Entity is deemed to be a class (or subset of classes in a module) responsible for retrieving user input to change characteristics of an object (a book for example). The attributes are stored in a separate class and accessed by these Entity classes.

Persistence Controller – A Persistence Controller is a set of classes used to execute queries against a database using data retrieved from an Entity.

The Central Application Controller matches an Entity to a correct Persistence Controller based on the name of a table and database. It will initialise the appropriate Entity class and pass the information to a Persistence Controller for database execution. Although the Central Application Controller acts as a central repository – passing information from Entities to Persistence Controllers, there are ideal layers created as Entities above and Persistence Controllers below.

Note: Fig 1. Also includes extra Entity and Persistence Controller demonstrating where additional functionality for this application should be added for application integration.

7. System Architecture

7.1 Architecture Class Model

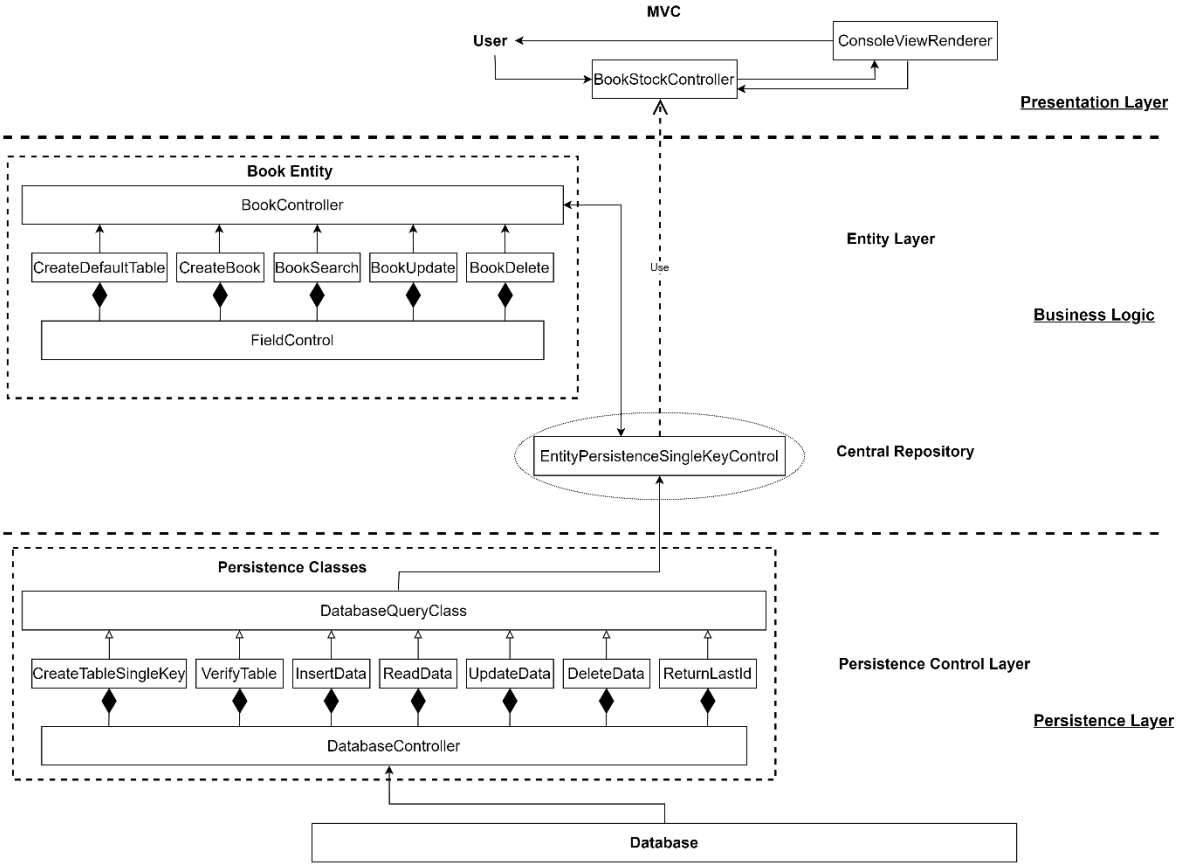

Fig 2. Main Class UML diagram of Book Management System Architecture Relations

8. System Evolution

8.1 Anticipated Feature Modifications

- It is anticipated that more attributes of each 'book' stored in the database would be added inclusive of ISBN number, Genre, Age Restriction, Price, Date of Publication, etc. 'FieldControl' class for Book Entity (see Fig 2.) must allow for easy addition of new fields, but manual correction of present data in database would be needed.
- Current view of application data is displayed in console window, however application would likely need this to be more user friendly and would be changed to HTML or a stand-alone application display window.
- Addition of new fields to book entity would require Normalization checks of database table likely requiring additional tables with new primary and foreign keys needed. Persistence Controller would need to be modified (if a non-compound primary key is maintained)

8.2 Anticipated Feature Additions

- Entities – It is anticipated that this system will need to include Financial, Employee and Customer Entities to allow it to integrate fully with the Book Store
- Persistence Controller – As the complexity of database tables increases and more entities are added, new persistence controllers will be needed. The current architecture would allow for integration of these systems through the use of its 'EntityPersistenceController'
- Network Expansions – The current system uses a local database on a single computer but expansion of the Book store may require the system to be integrated on multiple computers which could extend to online capabilities

8.3 Security Additions

- The current system does not have any security features apart from the use of careful query construction to prevent SQL Injection
- User Creations and Login – Restricting access of the system to registered users with required passwords would be a strong recommendation and allows for quick integration with the current system requiring very few changes.