**A Proposed Library Management System at Navotas City Library**

A Technical Documentation Presented to the

Faculty of Datamex College of Saint Adeline, Inc.

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in Information Technology

Prepared by:

Caldit, Peter Joel M.

September 2025

**TECHNICAL DOCUMENTATION**

**INTRODUCTION**

The purpose of this document is to provide a clear and comprehensive technical guide for the Library Management System (LMS) developed for the Navotas City Library. It aims to assist developers, system administrators, and library staff in understanding how the system is built, installed, configured, and maintained. This documentation also serves as a reference for troubleshooting issues, managing updates, and ensuring the system operates efficiently and reliably over time.

The Library Management System (LMS) is a computer-based application designed to automate and streamline the daily operations of a library. It allows users to register books, manage borrower information, record book borrowings and returns, and monitor inventory and overdue books. The system minimizes manual record-keeping, reduces human error, and ensures all library data is securely stored in a centralized database. It is developed using VB.NET for the user interface and Microsoft SQL Server for the database, providing a stable, user-friendly, and efficient solution for both librarians and administrators.

This documentation covers all key components and functionalities of the Library Management System, including installation procedures, configuration settings, database structure, and user operations. It also provides instructions for managing book records, borrower profiles, transactions (borrowing and returning), and report generation. Furthermore, it includes information about system testing, user access levels, and maintenance procedures. While third-party integrations and network configurations are briefly mentioned, the focus remains on the system’s internal processes and its daily use in library operations.

**SYSTEM OVERVIEW**

**System Architecture**

The Library Management System (LMS) follows a client-server architecture designed to ensure efficient and secure management of library operations. The VB.NET desktop application functions as the client interface, allowing librarians and staff to perform daily tasks such as adding books, registering borrowers, managing borrow and return transactions, and generating reports. The SQL Server 2019 database acts as the backend, securely storing all book, borrower, and transaction data. This setup provides fast, accurate, and reliable communication between the application and the database, minimizing errors and improving workflow efficiency.

**Main Components**

The Library Management System is composed of several interconnected components that work together to perform all library-related operations:

**Frontend:**

* A VB.NET Windows Forms application that provides an intuitive and user-friendly interface for managing books, borrowers, and transactions such as book borrowing, returning, and searching records.

**Backend:**

* A SQL Server 2019 database named LibraryDatabase that stores essential information, including book inventories, borrower details, borrowing records, return logs, and user account credentials.

**Storage:**

* Local directories are used to store book-related files or digital references such as cover images and borrower photos.

**Utilities Module:**

* A backend module containing reusable functions for managing database connections, executing SQL queries, validating data, and ensuring smooth and stable system performance.

**Deployment Architecture**

The Library Management System can be deployed using a Single-Computer Setup, where both the application and database are installed on the same computer. This configuration is ideal for small to medium-sized libraries or institutions that do not require multi-user access.

In this setup, all records are stored centrally in the SQL Server database, ensuring that all data remains consistent and up to date. Users can easily install and run the system through a standalone installer (either .exe or .msi), providing quick and reliable access to the full set of system features.

**INSTALLATION GUIDE**

**System Requirements**

**Hardware:**

* Processor: Intel Core i3 or higher
* Memory: Minimum 4 GB RAM
* Storage: At least 500 MB free disk space

**Software:**

* Operating System: Windows 10 or later
* Framework: Microsoft .NET Framework 4.5 or higher
* Database: Microsoft SQL Server 2019 or later
* Tools: SQL Server Management Studio (SSMS)
* Runtime: Microsoft LocalDB

**Installation Steps**

**Step 1 – Install SQL Server and SSMS**

* Run the installer:
* SQL2019-SSEI-Expr.exe /quiet /ACTION=Install /FEATURES=LocalDB
* Wait for the installation to finish (a command window may appear briefly).
* Open SQL Server Management Studio (SSMS) to verify that LocalDB is installed successfully.

**Step 2 – Attach the Database**

* Copy the database files (LibraryDatabase.mdf and LibraryDatabase\_log.ldf) to a folder, for example:
* C:\LibrarySystem\Database\
* Open SSMS → right-click Databases → select Attach…
* Click Add, browse for the .mdf file, then click OK to complete the database attachment.

**Step 3 – Open the Application**

* Locate and run the LMS\_Admin.exe file (or LibrarySystem.exe, depending on your version).
* Wait for the application to launch and connect to the database.

**Step 4 – Test the Installation**

* Log in using the default admin
* Register a sample book and a borrower.
* Test a sample borrow and return transaction to ensure the system is saving records properly.
* Verify that any related files or images are stored correctly in:
* C:\LibrarySystem\Files\

**CONFIGURATION GUIDE**

**Configuration Settings**

**Database Connection:**

Ensure that the connection string in the system configuration file points to the correct SQL Server instance or LocalDB.

**Example connection string:**

Data Source=(localdb)\MSSQLLocalDB;AttachDbFilename=C:\LibrarySystem\Database\LibraryDatabase.mdf;Integrated Security=True;

Verify that the database name and file path match your installation directory.

**File Paths:**

Book cover images, borrower photos, and digital copies of records are stored in:

C:\LibrarySystem\Files\

Ensure that this folder exists and has read/write permissions for all users.

**Backup Path:**

Create a dedicated folder for backups:

C:\LibrarySystem\Backups\

The system will store all generated .bak files in this directory.

**Best Practices**

* Perform regular backups before applying updates or system changes.
* Store backup copies on an external drive or cloud storage (e.g., OneDrive, Google Drive).
* Restrict access to configuration files and database folders to authorized administrators only.
* Verify paths during installation to prevent missing file errors

**DATABASE DOCUMENTATION**

**Database Structure**

**Database Name:** LibraryDatabase

**Main Tables**

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| **Books** | Stores information about all books in the library, including title, author, category, and availability status. |
| **Borrowers** | Stores borrower or member details such as name, ID number, and contact information. |
| **BorrowRecords** | Logs each borrowing transaction, including the borrower’s ID, book ID, borrow date, and return due date. |
| **ReturnRecords** | Tracks all returned books and calculates any overdue fines. |

**Table 1: Main Tables**

**Relationships (ERD Summary**)

Each Borrower can have multiple BorrowRecords.

Each Book can appear in multiple BorrowRecords but can only be borrowed once at a time.

Each BorrowRecord is linked to a ReturnRecord once the book is returned.

Each User performs system operations such as managing books, borrowers, and transactions.

**Backup and Restore**

**Backup Procedure:**

Open SQL Server Management Studio (SSMS).

Right-click the database → Tasks → Back Up.

Choose a destination folder (e.g., C:\LibrarySystem\Backups\LibraryDatabase.bak).

Click OK to start the backup.

**Restore / Attach Procedure:**

In SSMS, right-click Databases → Attach…

Browse to the .mdf file (e.g., C:\LibrarySystem\Database\LibraryDatabase.mdf).

Click OK to complete the attachment.

Reconnect the system and verify that all tables and records are accessible

**USER MANUAL**

**Instructions for Using the Software**

The Library Management System (LMS) allows librarians and staff to manage books, borrower records, borrowing transactions, and returns through an easy-to-use interface. After installation, double-click the application icon on your desktop or search Library Management System from the Start Menu to launch the program.

**User Interface Overview**

**Dashboard:** Displays key statistics such as the total number of books, total borrowers, books currently borrowed, and overdue returns.

**Books Module:** Used to register new books, update book details, categorize them, and track their availability status.

**Borrowers Module:** Manages member information including registration, profile updates, and borrowing history.

**Borrowing Module:** Handles book lending transactions — selecting a borrower, choosing books, and recording borrow dates and due dates.

**Returns Module:** Used to record returned books, update availability status, and calculate overdue fines if applicable.

**Common Tasks and Workflows**

**Register a New Book**

* Open the Books Module.
* Click Add New Book.
* Enter details such as Title, Author, Category, ISBN, and Quantity.
* Click Save to store the record in the database.

**Register a New Borrower**

* Go to the Borrowers Module.
* Click Add Borrower.
* Fill in the borrower’s information (name, ID number, contact details).
* Click Save to confirm registration.

**Borrow a Book**

* Open the Borrowing Module.
* Select the registered Borrower.
* Choose the Book(s) to be borrowed.
* Set the Borrow Date and Due Date.
* Click Confirm Borrow to record the transaction.

**Return a Book**

* Go to the Returns Module.
* Search for the borrower or book title
* Select the book being returned.
* Enter the Return Date and verify if any fines apply.
* Click Save Return to update the system and mark the book as available.

**TROUBLESHOOTING GUIDE**

**Common Issues and Solutions**

|  |  |  |
| --- | --- | --- |
| **Issue** | **Possible Cause** | **Solution** |
| **Cannot connect to database** | SQL Server not running or incorrect connection string | Open **SQL Server Configuration Manager**, ensure SQL Server service is running, and verify your database connection string. |
| **Books or borrower records not displaying** | Missing or corrupted database entries | Open SQL Server Management Studio (SSMS) and verify that the **Books** and **Borrowers** tables contain valid records. Restore from backup if necessary. |
| **Book cover images not showing** | File path missing or folder deleted | Recreate the folder \*C:\LibrarySystem\Books\* and re-upload book cover images. |
| **Application crashes on startup** | Missing or outdated .NET Framework | Install or update **.NET Framework 4.8** on the system. |
| **Slow performance** | Large database or low computer memory | Perform database cleanup and optimization, and ensure the computer has at least **4 GB RAM** available. |
| **Backup or restore not working** | Incorrect folder path or missing backup file | Verify the folder \*C:\LibrarySystem\Backups\* exists and that the latest backup file is available. |

**Table 2: Common Issues**

**Technical Support**

If problems persist, please contact your system administrator or IT support team.

Email: peterjoelcaldit@gmail.com

**CODE DOCUMENTATION**

**Overview**

The Library Management System (LMS) is developed using VB.NET (Visual Basic .NET) in Visual Studio 2010 or later, with Microsoft SQL Server as the database backend.

The system follows a modular structure, with each form handling specific library operations such as adding, viewing, borrowing, and returning books.

All database operations are managed through System.Data.SqlClient, ensuring secure and reliable data communication.

The user interface is built with Windows Forms and Guna UI components to provide a simple and modern design.

The LMS automates library tasks such as book registration, borrower management, and transaction tracking — improving accuracy and efficiency.

**Code Structure**

|  |  |
| --- | --- |
| **Component / File** | **Description** |
| **Form1.vb** | The main entry form of the system. It acts as the opening interface that allows access to the library modules and system features. |
| **Form2.vb** | Serves as the **Dashboard**, displaying key statistics such as total books, available books, borrowed books, and registered borrowers. |
| **Form3.vb** | Used to **add new books** into the system. Includes fields for title, author, publication date, category, and number of copies. |
| **Form4.vb** | Displays all book records in a **searchable and filterable list**. Allows users to view, update, or delete existing book entries. |
| **Form5.vb** | Handles **borrowing transactions**. It records borrower details, due dates, and updates book availability in the database. |
| **Form6.vb** | Manages **return transactions**. Updates book status when returned and calculates overdue penalties when applicable. |
| **DatabaseModule.vb** | Contains all database functions, including SQL connections, data retrieval, and query execution. |
| **Utilities.vb** | Provides reusable helper functions for data validation, formatting, and message handling. |

Each form is event-driven, reacting to user interactions such as button clicks and selections. SQL operations are parameterized to prevent SQL injection and maintain data integrity. Controls follow a consistent naming convention (e.g., txtBookTitle, btnSave, dgvRecords). The system automatically updates book stock and borrower history during borrow and return operations. The design prioritizes simplicity, ensuring it can be easily maintained and updated by future developers.

**Database Connection Module**

All SQL connections are managed in a dedicated module for efficiency and security.

Imports System.Data.SqlClient

Module DatabaseModule

Public connectionString As String = “Data Source=.\SQLEXPRESS;Initial Catalog=LibrarySystemDB;Integrated Security=True”

Public Function GetConnection() As SqlConnection

Return New SqlConnection(connectionString)

End Function

End Module

This centralized structure ensures that any change in the database configuration only needs to be made once.

Functions

**Add Books**

Using myconnection As New SqlConnection(“SERVER= LAPTOP-2EP3J41N\SQLEXPRESS; Database=book; Trusted\_Connection=True;”)

If String.IsNullOrWhiteSpace(addBn.Text) OrElse

String.IsNullOrWhiteSpace(addBan.Text) OrElse

String.IsNullOrWhiteSpace(addBp.Text) OrElse

String.IsNullOrWhiteSpace(addBpd.Text) OrElse

String.IsNullOrWhiteSpace(addBprice.Text) OrElse

String.IsNullOrWhiteSpace(addBq.Text) Then

MsgBox(“Please fill important details”, MsgBoxStyle.Exclamation, “Details”)

Exit Sub

End If

If MsgBox(“Do you want to add this record?”, MsgBoxStyle.YesNo + MsgBoxStyle.Exclamation, “Add Confirmation”) = MsgBoxResult.Yes Then

Myconnection.Open()

Try

Dim query As String

Query = “insert into books([BookName],[BookAuthorName],[BookGenre],[BookArrivalDate], [BookLocation],[BookQuantity]) values(@bn,@ban,@bp,@bpd, @bprice, @bq)”

Dim cmd As SqlCommand = New SqlCommand(query, myconnection)

Cmd.Parameters.AddWithValue(“@bn”, addBn.Text)

Cmd.Parameters.AddWithValue(“@ban”, addBan.Text)

Cmd.Parameters.AddWithValue(“@bp”, addBp.Text)

Cmd.Parameters.AddWithValue(“@bpd”, addBpd.Text)

Cmd.Parameters.AddWithValue(“@bprice”, addBprice.Text)

Cmd.Parameters.AddWithValue(“@bq”, addBq.Text)

Cmd.ExecuteNonQuery()

MsgBox(“Added Successfully!”, MsgBoxStyle.Information, “Success”)

Clear()

Viewdata()

Myconnection.Close()

Catch ex As Exception

MsgBox(ex.Message)

End Try

End If

End Using

**Update Books**

Using myconnection As New SqlConnection(“SERVER= LAPTOP-2EP3J41N\SQLEXPRESS; Database=book; Trusted\_Connection=True;”)

If String.IsNullOrWhiteSpace(viewbn.Text) OrElse

String.IsNullOrWhiteSpace(viewban.Text) OrElse

String.IsNullOrWhiteSpace(viewbp.Text) OrElse

String.IsNullOrWhiteSpace(viewbpd.Text) OrElse

String.IsNullOrWhiteSpace(viewbprice.Text) OrElse

String.IsNullOrWhiteSpace(viewbq.Text) Then

MsgBox(“Please fill important details”, MsgBoxStyle.Exclamation, “Details”)

Exit Sub

End If

If MsgBox(“Do you want to change this record?”, MsgBoxStyle.YesNo + MsgBoxStyle.Exclamation, “change Confirmation”) = MsgBoxResult.Yes Then

Myconnection.Open()

Try

Dim query As String

Query = “update books SET BookName=@bn,BookAuthorName=@ban,BookGenre=@bp,BookArrivalDate=@bpd, BookLocation=@bprice,BookQuantity= @bq WHERE id=@id”

Dim cmd As SqlCommand = New SqlCommand(query, myconnection)

Cmd.Parameters.AddWithValue(“@id”, txtid.Text)

Cmd.Parameters.AddWithValue(“@bn”, viewbn.Text)

Cmd.Parameters.AddWithValue(“@ban”, viewban.Text)

Cmd.Parameters.AddWithValue(“@bp”, viewbp.Text)

Cmd.Parameters.AddWithValue(“@bpd”, viewbpd.Text)

Cmd.Parameters.AddWithValue(“@bprice”, viewbprice.Text)

Cmd.Parameters.AddWithValue(“@bq”, viewbq.Text)

Cmd.ExecuteNonQuery()

MsgBox(“ Updated Successfully!”, MsgBoxStyle.Information, “Success”)

Clear()

Viewdata()

Myconnection.Close()

Catch ex As Exception

MsgBox(ex.Message)

End Try

End If

End Using

**Delete Books**

Dim result As DialogResult

result = MessageBox.Show("Are you sure you want to delete this?", "Confirm Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

If result = DialogResult.Yes Then

Try

Using myconnection As New SqlConnection("SERVER= LAPTOP-2EP3J41N\SQLEXPRESS;Database=book;Trusted\_Connection=True;")

myconnection.Open()

Dim query As String = "DELETE FROM books WHERE id=@id"

Using cmd As New SqlCommand(query, myconnection)

cmd.Parameters.AddWithValue("@id", txtid.Text)

cmd.ExecuteNonQuery()

End Using

End Using

MsgBox("Deleted Successfully", MsgBoxStyle.Information, "Success")

clear()

viewdata()

Catch ex As Exception

MsgBox(ex.Message)

End Try

End If

End Sub

Private Sub Guna2GradientTileButton3\_Click\_1(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Guna2GradientTileButton3.Click

viewbn.Clear()

viewban.Clear()

viewbq.Clear()

myconnection.Close()

End Sub

Private Sub Guna2GradientTileButton8\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Guna2GradientTileButton8.Click

Me.Hide()

Form2.Show()

End Sub

**Borrow Books**

Using myconnection As New SqlConnection("SERVER= LAPTOP-2EP3J41N\SQLEXPRESS; Database=bookborrow; Trusted\_Connection=True;")

If String.IsNullOrWhiteSpace(issueBn.Text) OrElse

String.IsNullOrWhiteSpace(issueCn.Text) OrElse

String.IsNullOrWhiteSpace(issueP.Text) OrElse

String.IsNullOrWhiteSpace(issueBookn.Text) OrElse

String.IsNullOrWhiteSpace(issueBookan.Text) OrElse

String.IsNullOrWhiteSpace(issueBookid.Text) Then

MsgBox("Please fill important details", MsgBoxStyle.Exclamation, "Details")

Exit Sub

End If

If MsgBox("Do you want to add this record?", MsgBoxStyle.YesNo + MsgBoxStyle.Exclamation, "Add Confirmation") = MsgBoxResult.Yes Then

myconnection.Open()

Try

Dim query As String

query = "insert into borrows ([BorrowerName],[ContactNo],[Place],[BookName], [BookBorrowDate],[BookDueDate]) values(@bon,@cn,@p,@bookn, @bookan, @bid)"

Dim cmd As SqlCommand = New SqlCommand(query, myconnection)

cmd.Parameters.AddWithValue("@bon", issueBn.Text)

cmd.Parameters.AddWithValue("@cn", issueCn.Text)

cmd.Parameters.AddWithValue("@p", issueP.Text)

cmd.Parameters.AddWithValue("@bookn", issueBookn.Text)

cmd.Parameters.AddWithValue("@bookan", issueBookan.Text)

cmd.Parameters.AddWithValue("@bid", issueBookid.Text)

cmd.ExecuteNonQuery()

MsgBox("Added Successfully!", MsgBoxStyle.Information, "Success")

clear()

viewdata()

viewdata1()

myconnection.Close()

Catch ex As Exception

MsgBox(ex.Message)

End Try

End If

End Using

**Return Books**

Using myconnection As New SqlConnection("SERVER= LAPTOP-2EP3J41N\SQLEXPRESS; Database=bookborrow; Trusted\_Connection=True;")

If String.IsNullOrWhiteSpace(Guna2DateTimePicker1.Text) Then

MsgBox("Please fill important details", MsgBoxStyle.Exclamation, "Details")

Exit Sub

End If

If MsgBox("Do you want to change this record?", MsgBoxStyle.YesNo + MsgBoxStyle.Exclamation, "change Confirmation") = MsgBoxResult.Yes Then

myconnection.Open()

Try

Dim query As String

query = "update borrows set BookReturnedDate=@brd WHERE id = @id"

Dim cmd As SqlCommand = New SqlCommand(query, myconnection)

cmd.Parameters.AddWithValue("@id", txtid.Text)

cmd.Parameters.AddWithValue("@brd", Guna2DateTimePicker1.Text)

cmd.ExecuteNonQuery()

MsgBox(" Updated Successfully!", MsgBoxStyle.Information, "Success")

clear()

viewdata1()

myconnection.Close()

Catch ex As Exception

MsgBox(ex.Message)

End Try

End If

End Using

**Error Handling**

Every module includes proper error handling using Try...Catch blocks to ensure system stability and user-friendly messages.

Example:

Try

'Database operation

Catch ex As SqlException

MessageBox.Show("Database error: " & ex.Message)

Catch ex As Exception

MessageBox.Show("Unexpected error: " & ex.Message)

End Try

**Security Implementation**

SQL Injection is prevented using parameterized queries.

Database connection strings are stored in a secured module.

**Coding Standards and Convention**

* Reusability: Shared database and utility modules prevent repetitive code.
* Modularity: Each form handles a single responsibility.
* Naming Consistency: Clear and descriptive variable names.
* Validation: All user inputs are validated before being saved to the database.
* Comments: Inline comments describe the logic of complex operations.
* UI Consistency: Uniform color scheme and button design using Guna UI controls.

**TESTING DOCUMENTATION**

**Testing Plan**

The testing phase was designed to ensure that every feature of the Library Management System (LMS) operates correctly, securely, and according to the system requirements. The goal was to verify functionality, usability, and database reliability before deployment.

**Primary Focus Areas:**

**Functional Testing** – Verified that each core module (Add Book, View Book, Borrow, Return, and Reports) performs its intended task properly.

Example: Borrowing a book automatically updates its availability, while returning a book restores the available stock count.

**Database Integrity Testing** – Ensured that all data relationships and constraints were properly enforced within the SQL Server database.

Example: A book that is currently borrowed cannot be deleted from the system until it is returned.

**User Acceptance Testing (UAT**) – Confirmed that librarians and staff can easily navigate the system, add new books, and process borrow and return transactions without errors.

**UI and Usability Testing** – Checked form layouts, button placements, and text alignment across all modules (Form1–Form6) for consistent and user-friendly operation.

**Performance Testing** – Evaluated how quickly the dashboard loads book data, borrower lists, and report summaries under normal operating conditions.

**Test Data and Methodology**

Testing used realistic sample records, including book titles, authors, and borrower information, to simulate daily library operations. Each form (Form1–Form6) was tested using a variety of valid and invalid inputs to identify logic errors, connection issues, or data inconsistencies.

**The testing process included:**

Manual testing of all buttons, text fields, and database operations.

Verification of message prompts and error handling.

Cross-checking data accuracy between forms and database tables.

**Test Case Summary**

All modules were thoroughly tested to ensure:

* Input validation: Prevents null, incorrect, or duplicate entries for books and borrowers.
* Data accuracy: Ensures proper saving, updating, and deleting of records.
* Database consistency: Changes in one module (e.g., borrowing) correctly reflect in others (e.g., available books).
* Logical operations: Borrowing and returning functions update stock counts and transaction history accurately.
* Error handling: Displays proper alerts for invalid inputs or missing fields.
* System stability: Performs reliably under normal usage without crashes or data loss

**Test results**

|  |  |  |
| --- | --- | --- |
| **Category** | **Description** | **Result** |
| **Functional Tests** | Verified correct behavior of all core modules (Add Book, View Book, Borrow, Return, and Reports). | Passed |
| **Database Integrity** | Confirmed referential integrity between Books, Borrowers, and Transactions tables. | Passed |
| **User Acceptance** | Librarians and staff confirmed that all features are easy to use and accurate during daily operations. | Passed |
| **Performance** | The system showed acceptable response times when loading records, borrowing, and returning books. | Passed |
| **UI/UX** | Minor layout issues (button spacing and label alignment) were detected and corrected. | Fixed |

**Table 4: Test Results**

All major functionalities performed as expected. Minor visual adjustments were implemented before deployment. After successful User Acceptance Testing (UAT), the Library Management System achieved full operational readiness and was approved for official use.

**MAINTENANCE GUIDE**

1. **Maintenance Procedures**

To ensure long-term stability, data reliability, and smooth operation of the Library Management System, administrators and developers should perform the following maintenance tasks regularly:

**Database Maintenance**

* Perform regular backups of the LibraryDatabase using SQL Server Management Studio (SSMS) or the built-in backup feature in the system.
* Store backups in C:\LibrarySystem\Backups\ and periodically copy them to an external drive or cloud storage for extra protection.
* Conduct monthly database integrity checks using the DBCC CHECKDB command to detect and repair any possible database corruption.

**File Maintenance**

* Periodically review the folder C:\LibrarySystem\Books\ to remove unused or outdated book images and documents.
* Monitor disk usage to ensure there is enough storage for future backups and newly added files.

**Configuration Maintenance**

* Update connection strings and file path settings if there are any changes to the server name, database instance, or storage location.
* Verify that file paths for images, backups, and reports are accessible and properly configured.

1. **Version Control and Updates**

All development updates, bug fixes, and feature improvements must be properly documented and managed using a version control system such as GitHub.

**Best Practices:**

* Maintain a dedicated repository named (e.g., LibraryManagementSystem-VBNet).
* Tag every version release using semantic versioning (e.g., v1.0.0, v1.1.0).
* Record all changes, including code modifications and database updates, in a CHANGELOG.md file.
* Commit updates regularly with clear and meaningful commit messages describing each change

**Update policy**

* To maintain the reliability and efficiency of the Library Management System, every update must follow structured procedures to ensure stability and prevent data loss.
* Guidelines:
* Conduct regression testing after every major update to confirm that existing features (e.g., borrowing, returning, and book management) still function correctly.
* Always maintain a backup of both the LibraryDatabase and the executable (.exe) before deploying any new version.
* Record all update details — including date, version number, author, and a summary of changes — in the system’s Maintenance Log for future reference.

1. **Bug Fixes and Enhancements**

**Bug Management Process**

To efficiently manage software issues and ensure timely resolutions

Log all identified bugs in a Bug Report Sheet or digital issue tracker.

Include the following details:

Bug ID

* Description and affected module (e.g., Form3 – Add Form, Form5 – Borrow Form)
* Severity (Low, Medium, High)
* Date Reported and Assigned Developer
* Once fixed, mark the bug as “Resolved” and document the resolution steps taken.
* Conduct a retest during the next testing cycle to ensure the issue no longer appears.

Enhancement Process

All new feature requests or improvements should be properly logged and reviewed for feasibility.

Approved enhancements are then scheduled for the next release cycle and added to the changelog.

Developers must document any new modules, forms, or database changes introduced during development to maintain system transparency.

1. **Long-Term Maintenance Recommendations**

To ensure the system remains stable, secure, and adaptable for future use:

* Implement scheduled automatic database backups using SQL Agent or Windows Task Scheduler.
* Periodically gather user feedback from librarians and staff to identify areas for usability improvement.
* Keep Visual Studio and SQL Server updated to the latest stable versions to maintain compatibility and security.
* Plan for future web-based or cloud migration to support remote access, multi-branch operations, and improved scalability