# Library Management System

# Project Report

## 1. Overview

The Library Management System is a simple Java-based application designed to manage a library's operations. The system allows users to perform basic operations such as adding books, borrowing books, returning books, and viewing the list of available books. The project is implemented following clean coding practices and Test-Driven Development (TDD).

## 2. Objectives

The main objectives of this project are:

- To create a functional Library Management System that can handle basic library operations.
- To apply the principles of Test-Driven Development (TDD) in designing and implementing the system.
- To ensure code quality and maintainability through the use of Java, Maven, and JUnit.
- To document the process and provide clear instructions for building and running the project.

## 3. Project Structure

The project follows a standard Maven project structure:

- src/main/java/com/example/library/: Contains the core classes like Library.java, Book.java.
- src/test/java/com/example/library/: Contains unit tests.

**Key Components:**

- **Library.java**: Manages the collection of books and handles borrowing, returning, and listing available books.
- **Book.java**: Represents a book with attributes like ISBN, title, author, and publication year.
- **LibraryManagementSystem.java**: The main class to run the application.
- **LibraryTest.java & BookTest.java**: Unit tests for the Library and Book classes to ensure functionality and correctness.

## 4. Development Process

### 4.1. Test-Driven Development (TDD)

The project was developed using Test-Driven Development (TDD). The process involved:

1. **Writing Tests**: Before implementing any functionality, corresponding unit tests were written in JUnit to define the expected behavior.
2. **Running Tests**: Initially, the tests would fail since the functionality was not yet implemented.

3. **Implementing Code**: The code was then written to pass the tests.
4. **Refactoring**: The code was refactored for optimization, ensuring that all tests continued to pass.

### 4.2. Tools Used

- **Java**: For writing the main application code.
- **Maven**: For project management, building the application, and managing dependencies.
- **JUnit**: For unit testing, ensuring code correctness and reliability.

## 5. Testing

### 5.1. Test Coverage

The application was thoroughly tested using unit tests. Key functionalities such as adding a book, borrowing a book, returning a book, and listing available books were tested to ensure they work as expected.

### 5.2. Test Results

All tests were executed successfully, confirming that the core functionalities of the system are working correctly.

**Example Test Cases:**

- **Add Book Test**: Ensures that a new book can be added to the library.
- **Borrow Book Test**: Verifies that a book can be borrowed if available, and checks that an error is raised if the book is unavailable.
- **Return Book Test**: Confirms that a borrowed book can be returned and becomes available for borrowing again.
- **View Available Books Test**: Checks that only available books are listed.

## 6. Conclusion

The Library Management System project successfully demonstrates a simple library management application built using Java, Maven, and JUnit. The project adheres to clean coding practices and leverages Test-Driven Development (TDD) to ensure code quality and functionality. The project is easy to set up, build, and run, with clear documentation provided in the README.md file.

This project serves as a solid foundation for further development, where additional features such as user management, advanced search capabilities, and a graphical user interface (GUI) can be implemented.