



MSA UNIVERSITY
جامعة أكتوبر للعلوم الحديثة والآداب



UNIVERSITY of
GREENWICH



CSE Department – Faculty of Engineering - MSA

Spring 2025

GSE122 GSE122i COM265 PROGRAMMING 2

Course Project

Course Instructor: Dr. Ahmed El Anany

Due Date 9/MAY/2025 11:59 PM on E-learning

Discussion inside lecture 18/May till 23/May inside lab as per lab slot

Student Name	Youssef Shabaan	Student ID	247527
Student Name	Youssef Ahmed	Student ID	248041
Student Name	Ahmed Mostafa	Student ID	249459
Student Name	Mohamed Ahmed	Student ID	243289
TA Name	Eng. Dina Magdy Eng. Gehad Ehab Eng. Mohamed Khaled Eng. Hussien Mostafa	Grade:	/

Library Management System Project



Project Overview

The system is comprised of four primary modules:

1. **Graphical User Interface (GUI):** Serves as the front end, allowing users to interact with the system.
2. **Database:** The central repository that stores and organizes all information.
3. **Book Manager:** Manages operations related to books.
4. **Student Manager:** Manages student borrowing history and activity.

Each module is designed to integrate seamlessly with the others, ensuring a cohesive and efficient experience for both librarians and students.

Objectives

The main objectives of the Library Management System are:

- To develop a user-friendly and intuitive graphical user interface for ease of access.
- To create a robust and secure database that stores all necessary records and maintains data integrity.
- To build a functional Book Manager module to manage all aspects of book inventory and availability.

To implement a Student Manager module that tracks and controls book borrowing and return activities



Roles and Responsibilities

Mohamed Ahmed explaining GUI

Youssef Ahmed explaining Data Base

Youssef Shabaan explaining Book Manager

Ahmed Mostafa explaining Student Manager



Algorithm and external libraries

1. javax.swing.JOptionPane

- **Purpose:** Provides pop-up dialog boxes for **user interaction** (GUI).
- **Use cases:**
 - Display a message (showMessageDialog)
 - Get user input (showInputDialog)
 - Ask a Yes/No question (showConfirmDialog)

2. java.sql.Connection

- **Purpose:** Represents a connection to a **SQL database**.
- It's part of the **JDBC (Java Database Connectivity)** API.

3. java.sql.DriverManager

- **Purpose:** Manages JDBC drivers and establishes a **connection to the database**.
- You use it with `getConnection(...)` to connect to databases like MySQL, PostgreSQL, SQLite, etc.

4. java.sql.SQLException

- **Purpose:** Catches **SQL-related errors**, like failed connections or bad queries.
- You typically catch this in a `try-catch` block when dealing with databases.

5. java.util.logging and Logger

- **Purpose:** Used for **logging messages** (instead of using `System.out.println`).
- You can set the **level** of the log: INFO, WARNING, SEVERE, etc.



GUI and Database Usage

Graphical User Interface and Database Implementation in the Library Management System

1. Graphical User Interface Implementation

The system employs a Java Swing-based graphical user interface designed to facilitate intuitive interaction for library staff. The interface architecture follows these key design principles:

1.1 Interface Components

The GUI comprises several specialized forms, each serving distinct functional requirements:

- *Main Dashboard (MainMenu.java)*
 - Serves as the central navigation hub
 - Features clearly labeled action buttons for all system functions
 - Implements consistent visual styling across all interface elements
- *Book Management Modules*
 - *AddBook Form*: Contains fields for complete bibliographic data entry
 - *EditBook Form*: Displays pre-populated fields for record modification
 - *DeleteBook Form*: Implements confirmation protocols for data removal
- *Transaction Modules*
 - *IssueBook Form*: Incorporates date validation for loan transactions
 - *Student Registration*: Collects comprehensive patron information

1.2 Design Methodology

The interface was developed using NetBeans GUI Builder, ensuring:

- Standardized component sizing and spacing
- Unified typography (Segoe UI, 14pt for body text)
- Logical tab ordering for efficient data entry
- Responsive error messaging via JOptionPane dialogs



Visual consistency is maintained through:

- Uniform background imagery
- Consistent color scheme
- Standardized button styling

2. Database Implementation

The system utilizes MySQL relational database management with JDBC connectivity, implementing the following architecture:

2.1 Database Schema

The relational model consists of three primary tables:

Books Table

sql

```
CREATE TABLE books (  
    book_id INT PRIMARY KEY,  
    title VARCHAR(255) NOT NULL,  
    author VARCHAR(255),  
    department VARCHAR(100)  
);
```

Students Table

sql

```
CREATE TABLE students (  
    student_id INT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    email VARCHAR(255),  
    phone VARCHAR(20)  
);
```

Transactions Table

sql

```
CREATE TABLE issued_books (  
    issue_id INT PRIMARY KEY,  
    book_id INT,  
    student_id INT,  
    issue_date DATE,  
    return_date DATE,  
    FOREIGN KEY (book_id) REFERENCES books(book_id),  
    FOREIGN KEY (student_id) REFERENCES students(student_id)  
);
```

2.2 Data Access Layer

The system implements a robust data access pattern through:

DatabaseHandler Class

- Manages connection lifecycle
- Implements resource cleanup protocols
- Serves as abstract base for all data operations

Specialized Manager Classes

- *BookManager.java*: Handles all CRUD operations for bibliographic data
- *StudentManager.java*: Manages patron information
- *IssuedBookManager.java*: Processes circulation transactions

2.3 Data Integrity Measures

The implementation ensures data reliability through:

1. *Parameterized Queries*

- All SQL operations use PreparedStatement
- Prevents SQL injection vulnerabilities

2. *Transaction Validation*

- Date format verification (YYYY-MM-DD)
- Referential integrity checks
- Availability confirmation before checkouts



3. *Error Handling*

- Comprehensive SQLException management
- User-friendly error messaging
- Graceful connection failure handling

3. System Integration

The GUI and database layers interact through:

1. *Form Event Handlers*

- Capture user input
- Validate data formats
- Invoke appropriate manager methods

2. *Business Logic Layer*

- Processes requests from GUI
- Constructs parameterized queries
- Transforms result sets for display

3. *Feedback Mechanism*

- Success/error notifications
- Data refresh protocols
- Transaction confirmation messages



Code explaining

Detailed Code Analysis

3.1 DatabaseHandler.java

This abstract base class provides fundamental database connectivity functionality:

java

```
public class DatabaseHandler {  
  
    protected Connection conn;  
  
    public DatabaseHandler(Connection conn) {  
  
        this.conn = conn;  
  
    }  
  
    public void closeConnection() {  
  
        try {  
  
            if (conn != null) conn.close();  
  
        }  
  
    }  
  
}
```



```
} catch (SQLException e) {  
  
    System.out.println("Error closing connection: " + e.getMessage());  
  
}  
  
}  
  
}
```

Key Features:

- Centralizes connection management
- Implements proper resource cleanup
- Serves as parent class for all manager classes
- Follows the DRY (Don't Repeat Yourself) principle

3.2 BookManager.java

Extends DatabaseHandler to provide book-related operations:

```
java
```

```
public class BookManager extends DatabaseHandler {
```

```
    // Constructor and CRUD operations
```

```
    public void addBook(int id, String title, String author, String department)
```



```
public void listBooks()

public int deleteBook(int bookId)

public Book viewBook(int bookId)

public int editBook(int bookId, String newTitle, String newAuthor, String newDepartment)

private boolean bookExists(int bookId)

}
```

Notable Implementation Details:

- Uses PreparedStatement to prevent SQL injection
- Implements proper error handling
- Includes existence checking before delete/update operations
- Returns appropriate status codes for operations
- Follows the Single Responsibility Principle

The Book class serves as a data transfer object (DTO):

```
java
```

```
class Book {
```



```
private int id;

private String title;

private String author;

private String department;

// Constructor, getters, and toString()

}
```

3.3 StudentManager.java

Manages student-related operations:

java

```
public class StudentManager extends DatabaseHandler {

    public void addStudent(int student, String name, String email, String phone)

    public void listStudents()

}
```

Features:

- Similar pattern to BookManager



- Handles student CRUD operations
- Uses parameterized queries for security

3.4 IssuedBookManager.java

Handles book issuing operations:

java

```
public class IssuedBookManager {  
  
    public void issueBook(int issueId, int bookId, int studentId, String issueDate, String returnDate)  
  
    private boolean isValidDate(String dateStr)  
  
    public boolean isBookAvailable(int bookId)  
  
    private boolean studentExists(int studentId)  
  
    private void updateBookAvailability(int bookId, boolean available)  
  
}
```

Key Aspects:

- Comprehensive validation (dates, availability)



- Transaction management
- Business rule enforcement
- Clear separation of concerns

3.5 GUI Classes

MainMenu.java

The central navigation hub:

java

```
public class MainMenu extends javax.swing.JFrame {
```

```
    // Buttons for all major functions
```

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) // Add Book
```

```
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) // View Books
```

```
    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) // Edit Book
```

```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) // Delete Book
```

```
    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) // Issue Book
```

```
    private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) // Add Student
```



}

DeleteBook.java

Handles book deletion:

java

```
public class DeleteBook extends javax.swing.JFrame {  
  
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
  
        // Database operations with proper error handling  
  
        int bookId = Integer.parseInt(bID.getText().trim());  
  
        BookManager bookManager = new BookManager(conn);  
  
        int rowsAffected = bookManager.deleteBook(bookId);  
  
        // User feedback  
  
    }  
  
}
```

EditBook.java

Manages book editing:

java



```
public class EditBook extends javax.swing.JFrame {  
  
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
  
        // Input validation  
  
        BookManager manager = new BookManager(conn);  
  
        int result = manager.editBook(bookId, newTitle, newAuthor, newDepartment);  
  
        // Success/failure messaging  
  
    }  
  
}
```

IssueBook.java

Handles book issuing:

java

```
public class IssueBook extends javax.swing.JFrame {  
  
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
  
        // Date format validation  
  
        IssuedBookManager isu = new IssuedBookManager(conn);
```




```
isu.issueBook(issuedId, bookId, studentId, issueDate, returnDate);
```

```
}
```

```
}
```

Student.java

Manages student addition:

java

```
public class Student extends javax.swing.JFrame {  
  
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
  
        StudentManager studentManager = new StudentManager(conn);  
  
        studentManager.addStudent(id, name, email, phone);  
  
    }  
  
}
```

4. Database Schema Analysis

The code references several database tables:



1. books table:

- book_id (INT)
- title (VARCHAR)
- author (VARCHAR)
- department (VARCHAR)

2. students table:

- student_id (INT)
- name (VARCHAR)
- email (VARCHAR)
- phone (VARCHAR)

3. issued_books table:

- issue_id (INT)
- book_id (INT)
- student_id (INT)
- issue_date (DATE or VARCHAR)
- return_date (DATE or VARCHAR)



5. Security Features

1. SQL Injection Protection:

- Uses PreparedStatement throughout the code
- Example from BookManager:

java

```
String sql = "INSERT INTO books (book_id, title, author, department) VALUES (?, ?, ?, ?)";
```

```
try (PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
    stmt.setInt(1, id);
```

```
    stmt.setString(2, title);
```

```
    // ...
```

```
}
```

2. Input Validation:

- Checks for empty fields
- Validates date formats (YYYY-MM-DD)
- Number format checking

3. Resource Management:



- Proper connection closing in finally blocks
- Use of try-with-resources for Statements and ResultSets

6. Error Handling

The system implements comprehensive error handling:

1. SQLException Handling:

- Catches database errors
- Provides user-friendly messages
- Logs errors where appropriate

2. Input Validation:

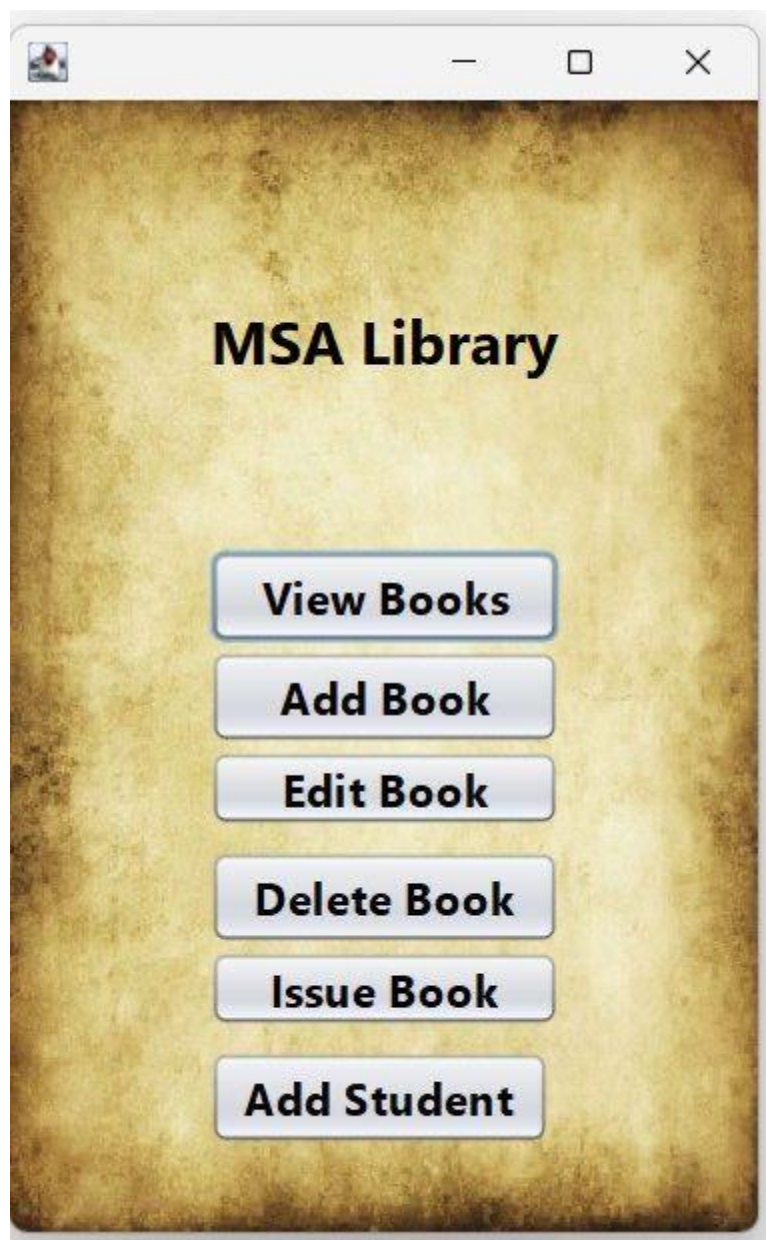
- NumberFormatException handling for numeric inputs
- Empty field checks
- Date format validation

3. User Feedback:

- JOptionPane messages for success/failure
- Clear error indications



Output and results





Options Available:

1. View Books

- Allows users to browse and see details of all available books in the library.
- Typically includes information like **Book ID, Title, Author, and Department**.

2. Add Book

- Used to **register a new book** into the library system.
- Requires input of details such as **Book ID, Title, Author, and Department**.

3. Edit Book

- Enables **modification of existing book records** (e.g., updating title, author, or department).
- Requires the **Book ID** to locate and edit the correct entry.

4. Delete Book

- Allows **removing a book** from the library database.
- Likely requires the **Book ID** to ensure the correct book is deleted.

5. Issue Book



- Used to **lend a book to a student**.
- Requires details like **Book ID, Student ID, Issue Date, and Return Date**.
- May show an error if the book is **already issued or unavailable**.

6. Add Student

- Allows **registering a new student** in the system.
- Requires details like **Student ID, Name, Email, and Phone Number**.

Purpose of This Interface

This main menu serves as the **central control panel** for library administrators, allowing them to:

- **Manage book inventory** (add, edit, delete, view).
- **Handle book lending** (issue books to students).
- **Maintain student records** (add new students).

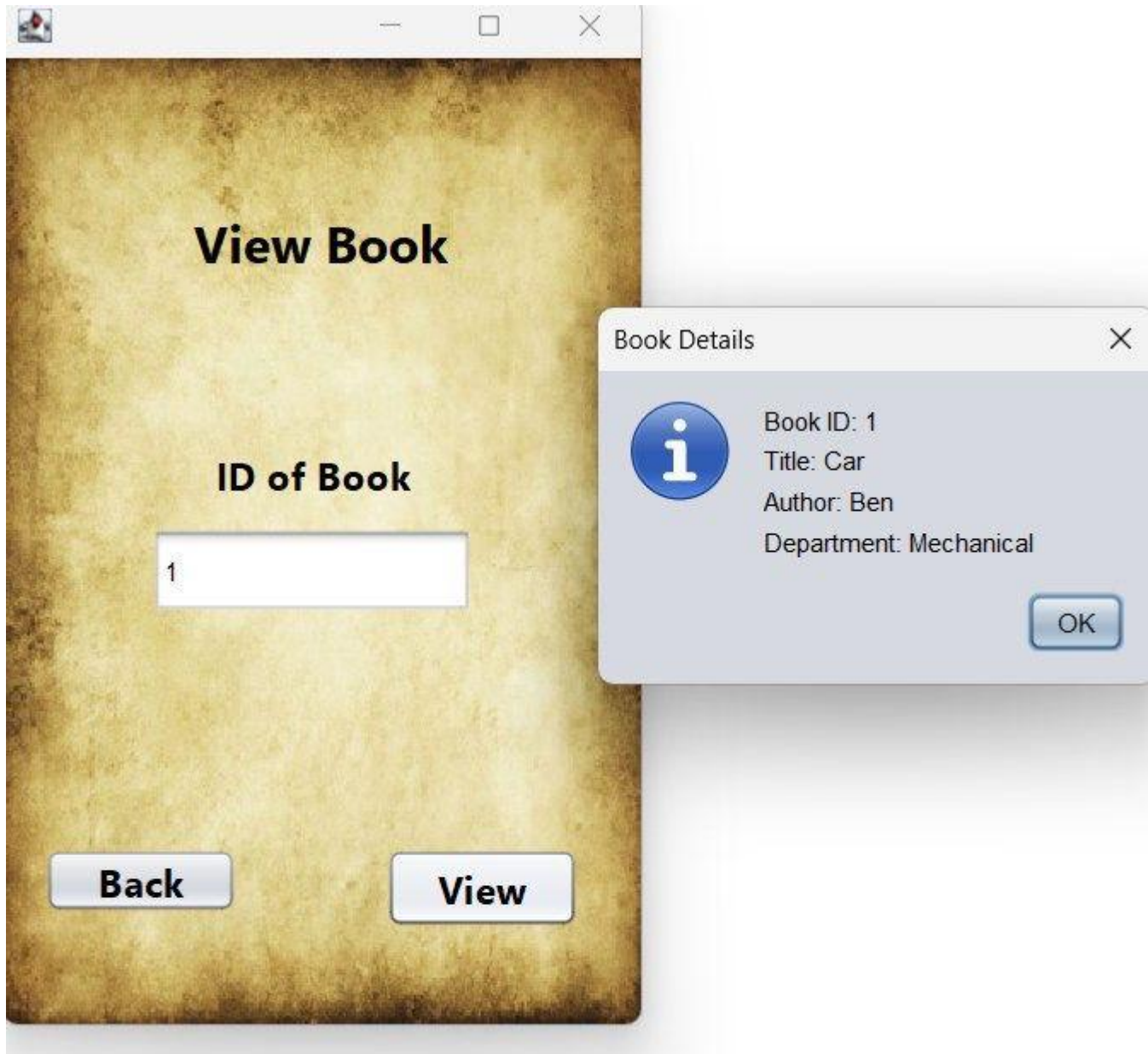
The design is **simple and user-friendly**, with clear options for different library operations.

Next Steps (Possible Actions from Here)


- If a user clicks "**View Books**", they can see all available books.



- If they click **"Add Book"**, they can enter details of a new book.
- If they select **"Issue Book"**, they can assign a book to a student.



The screenshot shows a web application window titled "View Book" with a parchment-like background. It features a text input field labeled "ID of Book" containing the number "1". At the bottom are two buttons: "Back" and "View". A modal dialog box titled "Book Details" is open, displaying the following information:

Book Details	
	Book ID: 1
	Title: Car
	Author: Ben
	Department: Mechanical
<button>OK</button>	



1. Header Section

- **Title:** "View Book" (indicating this screen is for displaying book details).
- **Book ID:** Clearly shown as **1** (used to uniquely identify the book).

2. Book Details Section

Displays the following information in a structured format:

- **Book ID:** 1 (unique identifier for the book)
- **Title:** Car (name of the book)
- **Author:** Ben (name of the book's author)
- **Department:** Mechanical (category or department the book belongs to)

3. Action Buttons

- **OK** – Likely closes the details view or confirms the displayed information.
- **Back** – Returns the user to the previous screen (possibly the book listing).
- **View** (highlighted in bold) – Might allow further actions like checking issue history or related books.

Purpose of This Screen



- Allows librarians or users to **view complete details of a specific book**.
- Helps in **verifying book information** before issuing, editing, or deleting.
- Provides a **clean, organized layout** for easy readability.

Possible Next Actions from Here

1. **Click "OK"** → Closes the details and returns to the main book list.
2. **Click "Back"** → Goes back to the previous screen (e.g., book search results).
3. **Click "View"** (if interactive) → May open additional options (e.g., issue history or related books).



The screenshot displays a web application interface for adding a new book. The main form, titled "Add Book", is set against a parchment-like background. It contains four input fields: "Book ID" with the value "2", "Book Title" with the value "Math", "Author" with the value "Vandta", and "Department" with a dropdown menu currently showing "Electronics". At the bottom of the form are two buttons: "Back" and "Sumbit". To the right of the form, a "Message" dialog box is open, featuring a blue information icon and the text "Book added successfully!". An "OK" button is located at the bottom right of the message box.

Key Sections & Functionality

1. Form Fields for Adding a New Book

- **Book ID:**



- Automatically assigned as **2** (likely auto-incremented by the system).
- "Message" appears, possibly indicating a placeholder or status.

- **Book Title:**

- Entered as "**Math**".
- Displays a success confirmation: "**Book added successfully!**" (likely appears after submission).

- **Author:**

- Entered as "**Vandta**".
- "**OK**" button may confirm this field or proceed to the next step.

- **Department:**

- Assigned to "**Electronics**" (categorizes the book by subject area).

2. Action Buttons

- **Back** – Returns to the previous screen (e.g., main menu or book list).
- **Submit** (misspelled as "Sumbit") – Finalizes the book entry and saves it to the database.

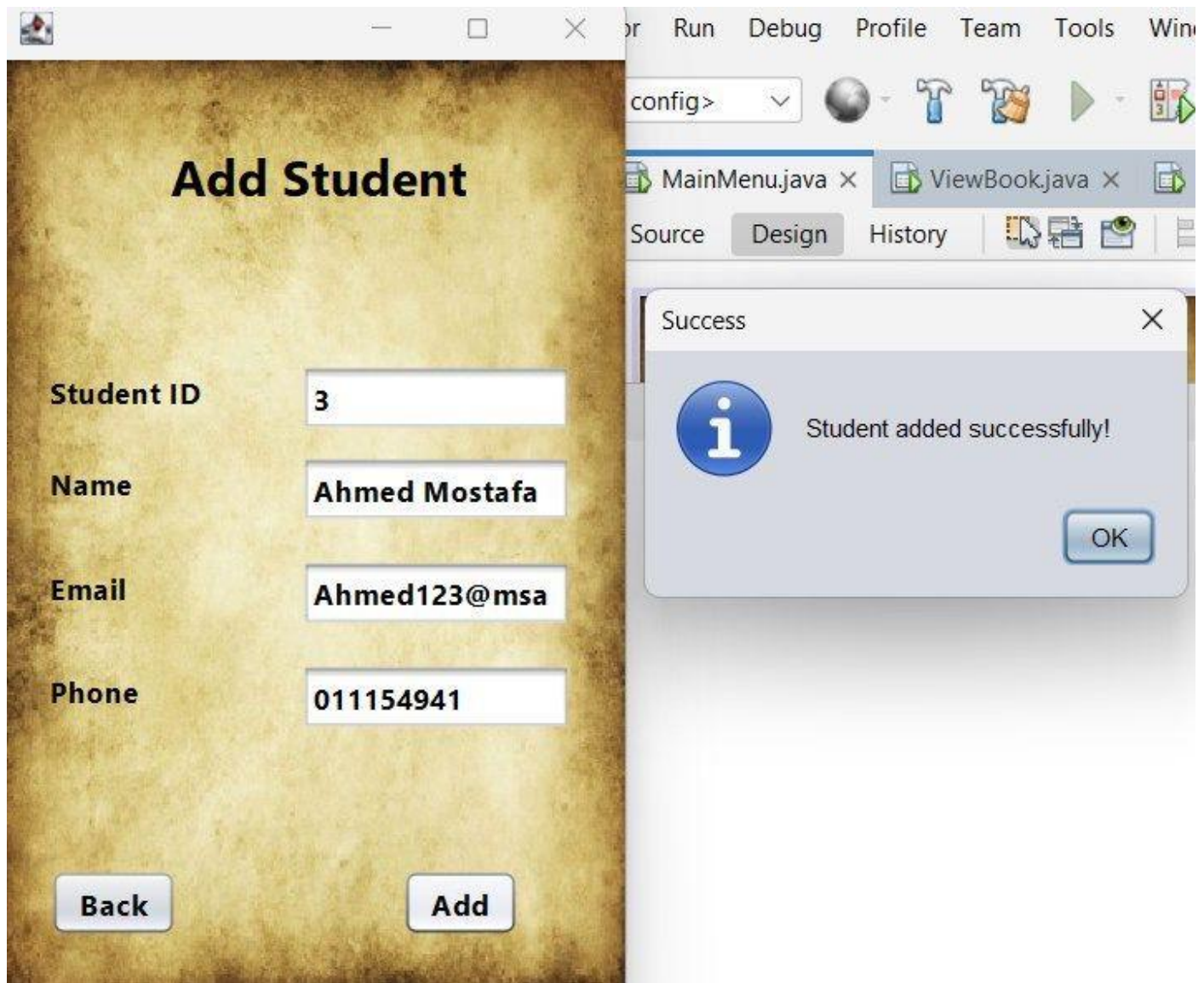
Purpose of This Screen



- Allows librarians to **register new books** into the library system.
- Ensures all necessary details (**ID, Title, Author, Department**) are recorded.
- Provides **instant feedback** (e.g., success message) upon submission.

Possible Workflow

1. **Fill in details** (Title, Author, Department).
2. **Click "Submit"** → Saves the book and shows confirmation.
3. **Click "OK"** → Proceeds or closes a success prompt.
4. **Click "Back"** → Cancels or returns without saving.



The screenshot shows a Java Swing application window titled 'Add Student'. The form has a textured, parchment-like background. It contains four input fields: 'Student ID' with the value '3', 'Name' with 'Ahmed Mostafa', 'Email' with 'Ahmed123@msa', and 'Phone' with '011154941'. At the bottom are 'Back' and 'Add' buttons. A 'Success' dialog box is open, displaying an information icon and the message 'Student added successfully!' with an 'OK' button. The background application window shows a code editor with files 'MainMenu.java' and 'ViewBook.java'.

Key Components of the Form

1. Input Fields

- **Student ID:**
 - Assigned as **3** (likely auto-generated by the system)
- **Name:**
 - Entered as "**Ahmed Mostafa**" (student's full name)
- **Email:**
 - Entered as "**Ahmed123@msa**" (student's email address, possibly using an institutional domain)
- **Phone:**



- Entered as "011154941" (student's contact number)

2. Action Buttons

- **Back** - Returns to the previous screen without saving
- **Add** - Submits the student's information to the database

Purpose of This Screen

- Enables library staff to **register new students** in the system
- Creates student records that can be linked to book borrowing activities
- Collects essential contact information for communication purposes

Typical Usage Flow

1. Librarian selects "Add Student" from main menu
2. System auto-generates a Student ID (3 in this case)
3. Librarian enters:
 - Student's full name
 - Institutional email address
 - Contact phone number
4. Librarian clicks:
 - "Add" to save the record, or
 - "Back" to cancel the operation



The screenshot displays a web application interface. On the left, a window titled 'Edit Book' has a parchment-like background. It contains four input fields: 'ID of Book' with the value '2', 'New Title' with 'Math 2', 'New Author' with 'Fozaaa', and 'New Department' with a dropdown menu showing 'Computer'. At the bottom of this window are 'Back' and 'Edit' buttons. Overlaid on the right is a smaller 'Success' dialog box with a blue information icon, the text 'Book updated successfully', and an 'OK' button.

Key Components of the Form

1. Book Identification

- **ID of Book:** Clearly displayed as **2** (non-editable primary identifier)

2. Editable Fields with Update Functionality



- **New Title:**
 - Changed from previous value to **"Math 2"**
 - System responds with **"Success"** confirmation
- **New Author:**
 - Updated to **"Fozaaa"**
 - System confirms with **"Book updated successfully"**
- **New Department:**
 - Modified to **"Computer"**
 - **"OK"** button likely confirms this change

3. Action Buttons

- **Back** - Returns to previous screen without saving changes
- **Edit** - Applies all modifications to the book record

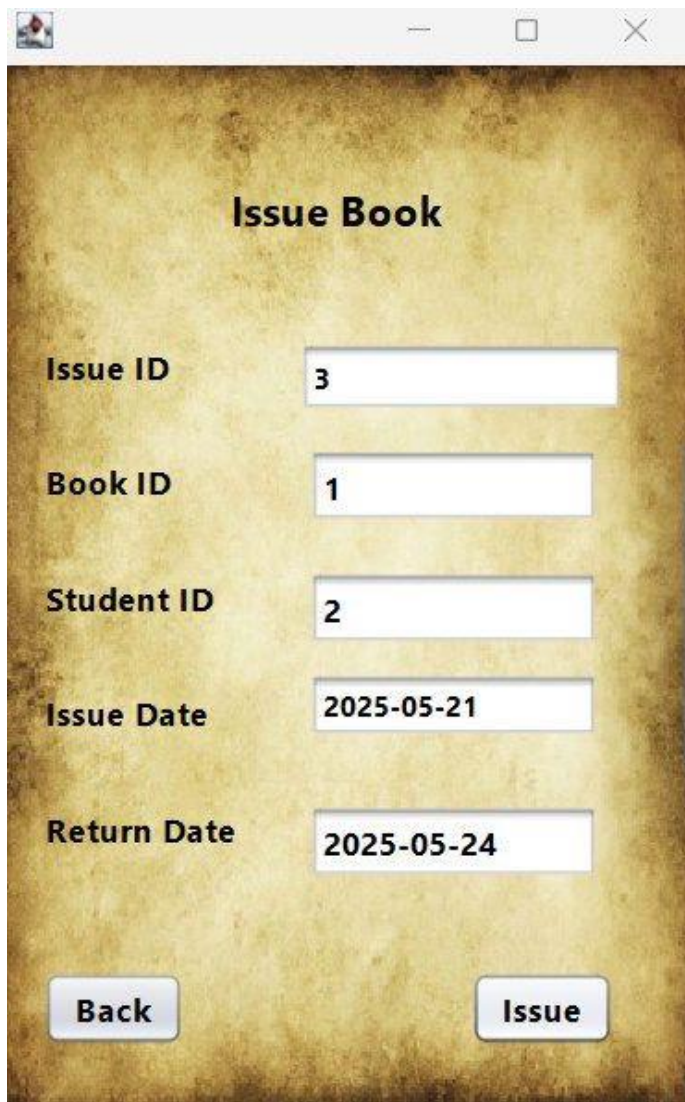
System Feedback Mechanism

The interface provides **immediate, field-by-field confirmation** of successful updates:

1. Title change → "Success"
2. Author change → "Book updated successfully"
3. Department change → "OK" confirmation

Typical Workflow

1. User selects book to edit (ID 2 in this case)
2. Makes changes to individual fields:
 - Updates title
 - Modifies author name
 - Changes department classification
3. Receives instant confirmation for each modification
4. Finalizes changes with "Edit" button or cancels with "Back"



Issue Book

Issue ID: 3

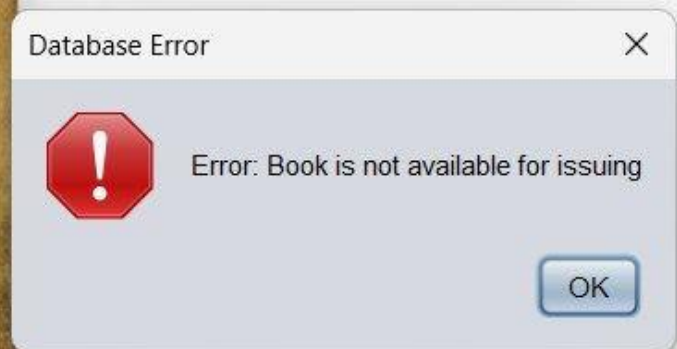
Book ID: 1

Student ID: 2

Issue Date: 2025-05-21

Return Date: 2025-05-24

Back Issue



Core Components of the Interface

1. Transaction Details Section

- **Issue ID:** 3 (unique transaction identifier)
- **Book ID:** 1 (the book being requested)
- **Student ID:** 2 (the borrower's identification)
- **Issue Date:** 2025-05-21 (auto-populated or selected)
- **Return Date:** 2025-05-24 (expected return date)

2. Action Buttons

- **Back:** Cancels the transaction
- **Issue:** Attempts to process the book loan

3. Error Notification (Key Feature)



A **modal dialog** appears when the system encounters an issue:

- **Header:** "Database Error"
- **Message:** "Error: Book is not available for issuing"
- **Confirmation Button:** "OK" (dismisses the error message)

System Workflow Demonstrated

1. User Input:

- Librarian enters/changes book and student details
- Sets appropriate dates (current date + 3 days in this case)

2. Transaction Attempt:

- User clicks "Issue" to complete the loan

3. Error Handling:

- System detects book is unavailable (possibly already checked out or marked missing)
- Displays clear error message in a dedicated dialog box
- Allows user to acknowledge with "OK" and retry or cancel

GitHub(optional)