Textbook Manager Project

Students
James Bell
Negin Jahanbakhsh
Jose Fidel Paredes Sanchez
Hanadi Jusufovic

Professor Fairouz Medjahed
Saint Louis University Madrid Campus
CSCI 4710: Databases
Fall 2021

Project Description

The project to be developed is for a theoretical university who wants a database to manage textbooks for courses as well as the student evaluations for these books. Teachers select textbooks for their classes which means the same course may have different textbooks. In addition, various courses may have the same textbooks. Students are also given the option to evaluate these books. The requirements and design for the database are listed in detail below.

Requirements

Functional Requirements

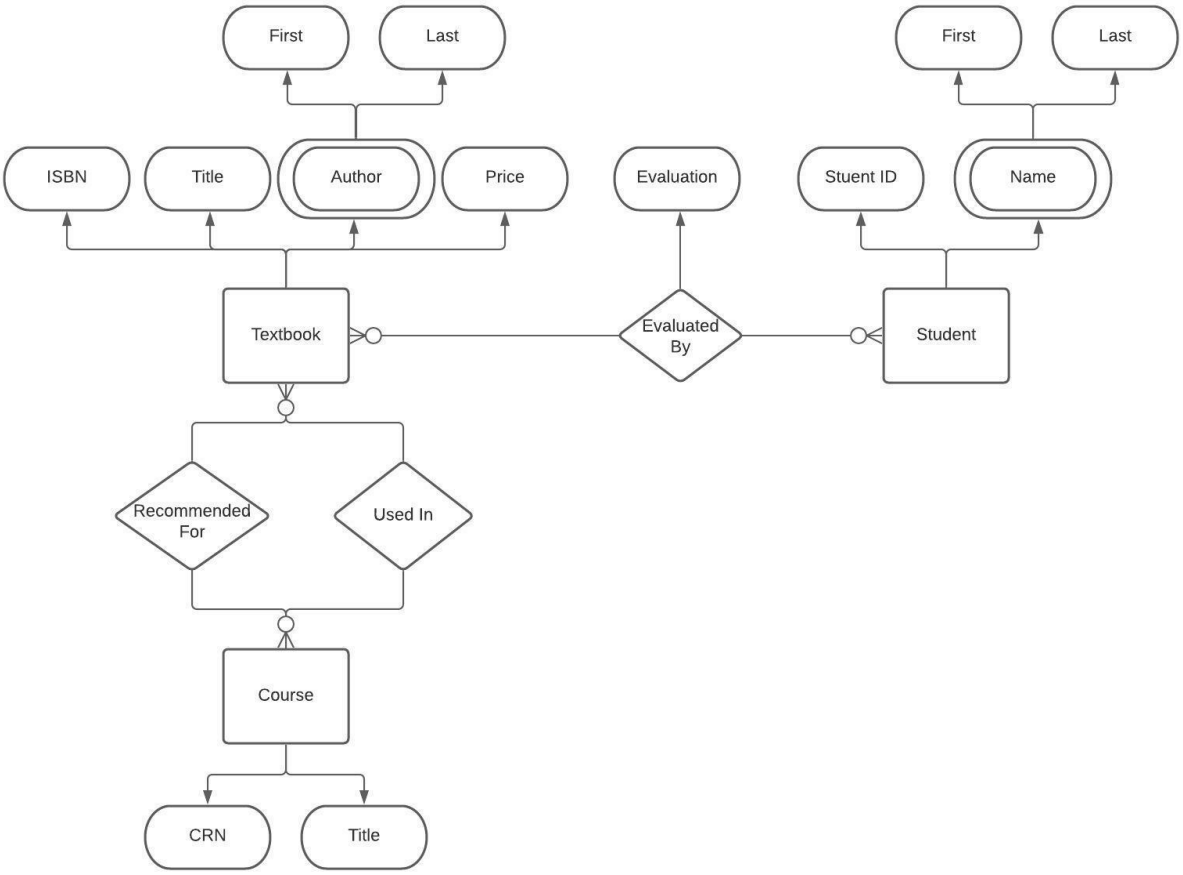
- Must be able to view reports such as recommended books for a course or required books for a course
- Must be able to add new data such as course and book
- Must be able to link books to courses
- Provide login page for managing database (adding & removing data)

Non Functional Requirements

- Require login security for certain tasks
- Changes updated on user input

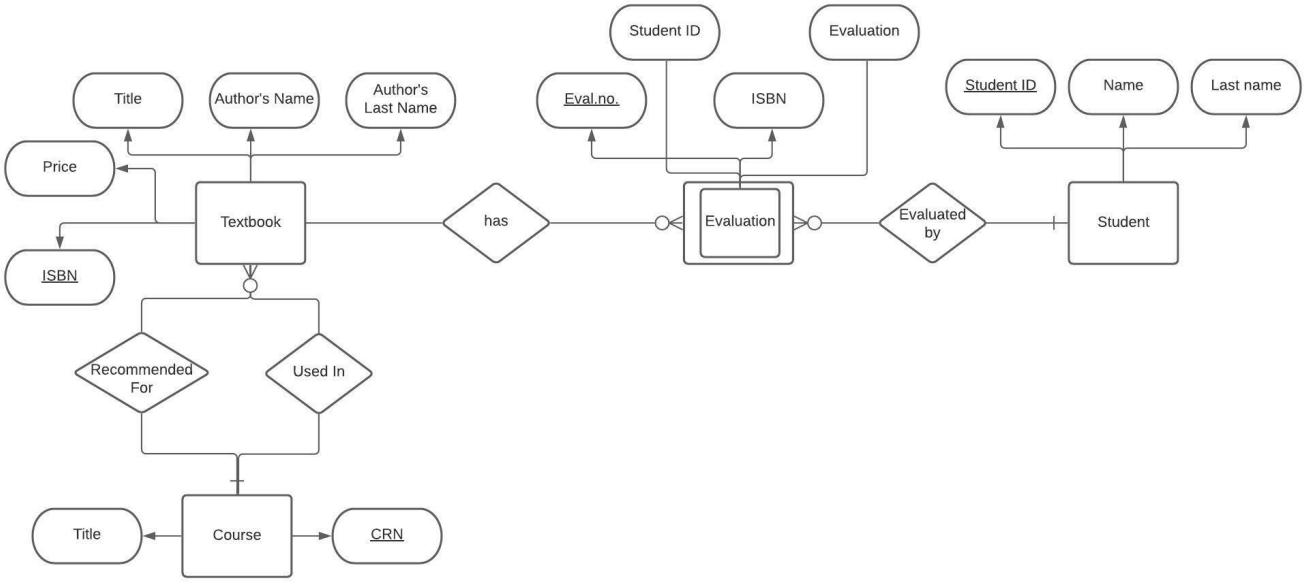
System Models

Conceptual Model



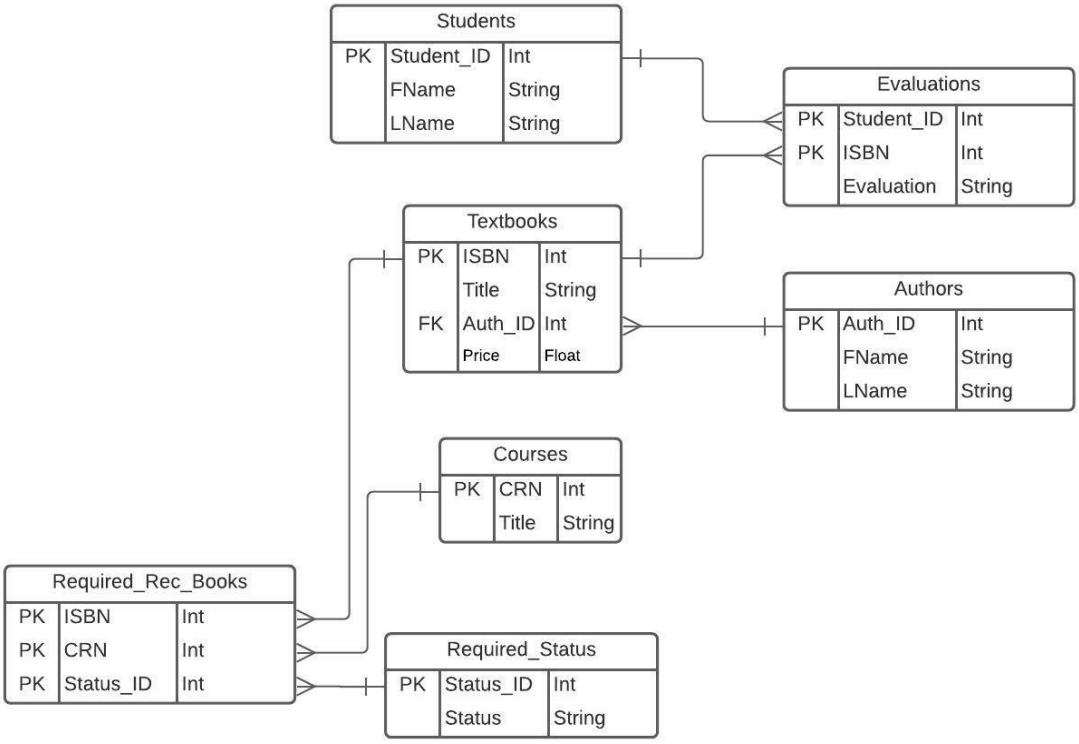
One important thing to note here is the difference in ‘recommended’ and ‘required’ in the relationship between textbook and course. Not all textbooks used in a course are required, some are simply recommended for the benefit of students.

Logical Model



The logical model is similar to the conceptual model except it includes more information regarding the evaluation entity as well as declaring the primary keys in the entities.

Physical Model



After normalizing the diagram, we landed on this ER diagram for the database. As you can see the status including ‘required’ or ‘recommended’ has its own table and ID. The authors have their own table as well, and the table that is the central entity is the ‘Required_Rec_Books’ which holds the data for what books are required and recommended for courses. The Evaluations table also holds the student evaluations for the books. As for data types, all keys that have ‘String’ data type will be varchar. We also realized there was no need for an evaluation_id because the student_id along with the isbn was enough for a primary key given that a student can only leave one evaluation per book. The Auth_ID primary key will be auto incremented as shown in the DDL code below:

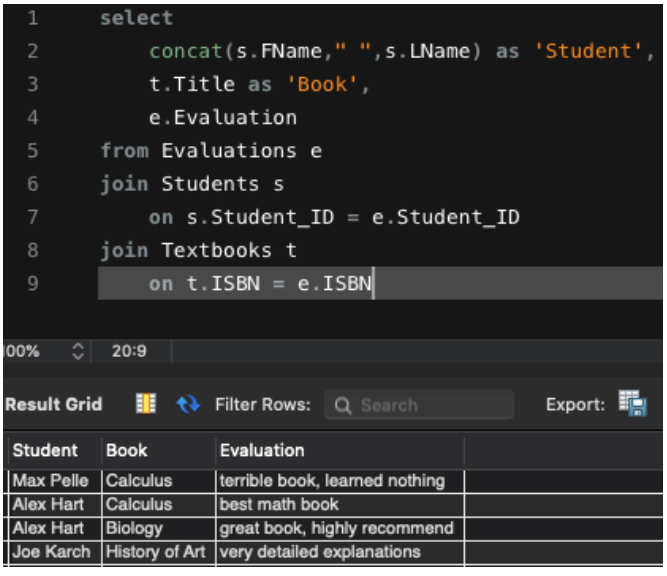
DDL Code for Textbook Manager Database:

```
CREATE DATABASE JB_NJ_HJ_JFP_TextbookManager;

CREATE TABLE `Authors` (
  `Auth_ID` Int NOT NULL AUTO_INCREMENT,
  `FName` varchar(50),
  `LName` varchar(50),
  PRIMARY KEY (`Auth_ID`)
);

CREATE TABLE `Textbooks` (
  `ISBN` Int NOT NULL,
  `Title` varchar(50),
  `Auth_ID` Int,
  `Price` Float,
  PRIMARY KEY (`ISBN`),
  FOREIGN KEY (`Auth_ID`) REFERENCES `Authors`(`Auth_ID`)
);

CREATE TABLE `Students` (
  `Student_ID` Int,
  `FName` varchar(50),
  `LName` varchar(50),
  PRIMARY KEY (`Student_ID`)
);

CREATE TABLE `Evaluations` (
  `Student_ID` Int,
  `ISBN` Int,
  `Evaluation` varchar(500),
  PRIMARY KEY (`Student_ID`, `ISBN`),
  FOREIGN KEY (`ISBN`) REFERENCES `Textbooks`(`ISBN`),
  FOREIGN KEY (`Student_ID`) REFERENCES `Students`(`Student_ID`)
);

CREATE TABLE `Required_Status` (
  `Status_ID` Int not null auto_increment,
  `Status` varchar(20),
  PRIMARY KEY (`Status_ID`)
);

CREATE TABLE `Courses` (
```

```
  `CRN` Int,
  `Title` varchar(50),
  PRIMARY KEY (`CRN`)
);

CREATE TABLE `Required_Rec_Books` (
  `ISBN` Int,
  `CRN` Int,
  `Status_ID` Int,
  PRIMARY KEY (`ISBN`, `CRN`, `Status_ID`),
  FOREIGN KEY (`Status_ID`) REFERENCES `Required_Status`(`Status_ID`),
  FOREIGN KEY (`ISBN`) REFERENCES `Textbooks`(`ISBN`),
  FOREIGN KEY (`CRN`) REFERENCES `Courses`(`CRN`)
);
```

Database Testing

The tables were populated with sample data and the relations work flawlessly, here are some example queries we ran that tests many of the tables in the database:

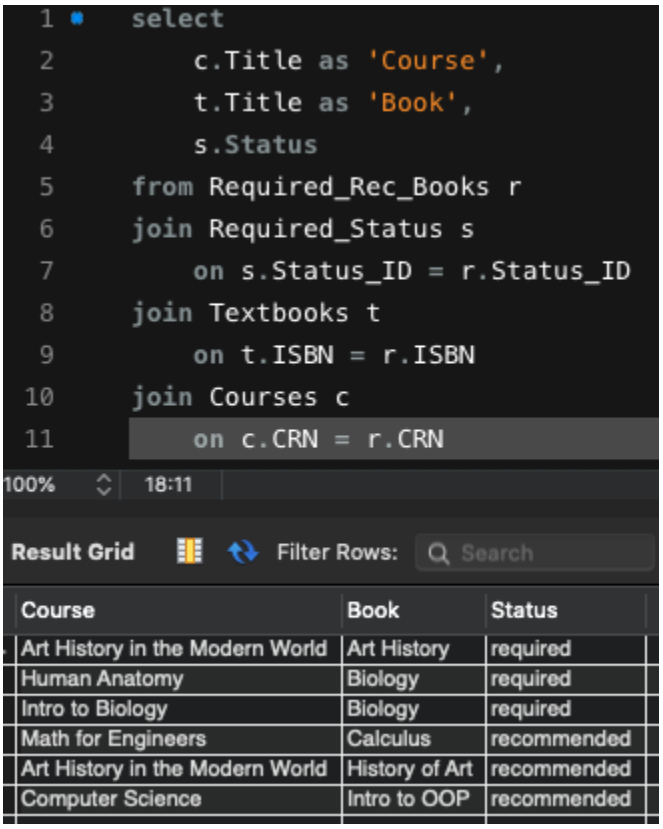


```
select
    concat(s.FName," ",s.LName) as 'Student',
    t.Title as 'Book',
    e.Evaluation
from Evaluations e
join Students s
    on s.Student_ID = e.Student_ID
join Textbooks t
    on t.ISBN = e.ISBN
```

100% 20:9

Result Grid Filter Rows: Search Export:

Student	Book	Evaluation
Max Pelle	Calculus	terrible book, learned nothing
Alex Hart	Calculus	best math book
Alex Hart	Biology	great book, highly recommend
Joe Karch	History of Art	very detailed explanations



```
select
    c.Title as 'Course',
    t.Title as 'Book',
    s.Status
from Required_Rec_Books r
join Required_Status s
    on s.Status_ID = r.Status_ID
join Textbooks t
    on t.ISBN = r.ISBN
join Courses c
    on c.CRN = r.CRN
```

100% 18:11

Result Grid Filter Rows: Search

Course	Book	Status
Art History in the Modern World	Art History	required
Human Anatomy	Biology	required
Intro to Biology	Biology	required
Math for Engineers	Calculus	recommended
Art History in the Modern World	History of Art	recommended
Computer Science	Intro to OOP	recommended

Part 2

First class that we created was EstablishConnection. This is the most important class since it allows us to create a database, populate it with data, execute queries, etc.

```java
import java.sql.*;

public class EstablishConnection {

    private Connection connection;
    private Statement statement;
    static final String url = "JB_NJ_HJ_JFP_TextbookManager";
    static final String user = "root";
    static final String pass = "Brettlee11";

    public EstablishConnection() {

        try {

        this.connection =  DriverManager.getConnection("jdbc:mysql://localhost:3306/" + url, user,
        this.statement = connection.createStatement();
        } catch (Exception e) {

            System.out.println(e);
            System.exit(1);
        }
    }

    public Statement getStatement() {

        return this.statement;
    }

    public Connection getConnection() {

        return this.connection;
    }
}
```

In order to initialize the database we created a class for each table and created InitializeDatabase class.

For example, the class for Authors and code to Initialize and Populate Authors table is shown below.

Authors class

```java
public class Authors {

    private String FName;
    private String LName;

    // Likely constructor to use
    public Authors(String FName, String LName) {

        this.FName = FName;
        this.LName = LName;
    }

    public String getFName() {

        return this.FName;
    }

    public void setFName(String FName) {

        this.FName = FName;
    }

    public String getLName() {

        return this.LName;
    }

    public void setLName(String LName) {

        this.LName = LName;
    }
```

```java
    //Following methods are used for initializing database

    public Authors() {

        this("", "");
    }

    public void set(String data, int i) {

        switch(i) {
            case 0:
                setFName(data);
                break;
            case 1:
                setLName(data);
                break;
        }
    }
    public String get(int i) {

        String value = "";
        switch(i) {
            case 0:
                value = getFName();
                break;
            case 1:
                value = getLName();
                break;
        }

        return value;
    }
}
```

InitializeDatabase class

```java
import java.sql.*;
import java.util.Scanner;
import java.io.File;

public class InitializeDatabase {

    private EstablishConnection conn;

    public InitializeDatabase() {

        conn = new EstablishConnection();
    }

    public void createTables() {

        createAuthors();
        createTextbooks();
        createStudents();
        createEvaluations();
        createRequiredStatus();
        createCourses();
        createRequiredRecBooks();
}
    public void populateTables() {

        populateAuthors();
        populateTextbooks();
        populateStudents();
        populateEvaluations();
        populateRequiredStatus();
        populateCourses();
        populateRequiredRecBooks();
    }
```

Create and Populate Authors

```
public void createAuthors() {

    DatabaseMetaData dbm;
    ResultSet tables;
    try {
        dbm = conn.getConnection().getMetaData();
        tables = dbm.getTables(null, null, "Authors", null);

        if(tables.next()) {

    }
        else {
            String createString =
            "CREATE TABLE `Authors` " + "(`Auth_ID` Int NOT NULL AUTO_INCREMENT, " +
            "`FName` varchar(50), " + "`LName` varchar(50), " +
             "PRIMARY KEY(`Auth_ID`))";

            try {
                conn.getStatement().executeUpdate(createString);
             } catch(Exception e) {
                System.out.println(e);
                System.exit(1);
            }
    }
} catch(Exception e) {
    System.out.println(e);
    System.exit(1);
}
```

```
public void populateAuthors() {

    File file;
    Scanner input;

    try {
        file = new File("Authors.txt");
        input = new Scanner(file);

        Authors authors = new Authors();

        while(input.hasNextLine()) {

            String line = input.nextLine();
            String[] splitLine = line.split(";");

            for(int i=0; i<splitLine.length; i++) {
                authors.set(splitLine[i], i);
            }

            String query = "INSERT INTO Authors (FName, LName) " +
            "values (?, ?)";
            PreparedStatement preparedStmt = conn.getConnection().prepareStatement(query);

            for(int i=0; i<splitLine.length; i++) {
                preparedStmt.setString(i+1, authors.get(i));
            }
            preparedStmt.execute();
        }
        input.close();
    } catch(Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}
```

All other classes such as Courses, Students, and methods createCourses(), createStudents(), etc., are identical to Authors shown above, except they contain variables that correspond to the physical model in part 1.

Then we create Database class which contains all the methods with queries
Here are all the methods in Database class:

```
import java.sql.*;

public class Database {

    private EstablishConnection connection;
    private ResultSet rs;

    public Database() {

        connection = new EstablishConnection();
    }

    public ResultSet getTables() {…

    public ResultSet getAuthors() {…

    public ResultSet getCourses() {…

    public ResultSet getStudents() {…

    public ResultSet getTextbooks() {…

    public ResultSet getEvaluations() {…

    public ResultSet getRequiredRecBooks() {…

    public ResultSet getRequiredStatus() {…

    public ResultSet getCourseBookStatus() {…

    public ResultSet getISBN(String title) {…

    public ResultSet getTextbookInfo(int ISBN) {…

    public ResultSet getDetailsAboutBook(int ISBN) {…

    public ResultSet getCourseBook(int CRN) {…

    public void insertAuthor(String fName, String lName) {…

    public void insertTextbook(int ISBN, String Title, int Auth_ID, Float Price) {…

    public void insertEvaluation(int Student_ID, int ISBN, String Evaluation) {…

    public void insertStudent(int Student_ID, String fName, String lName) {…
```

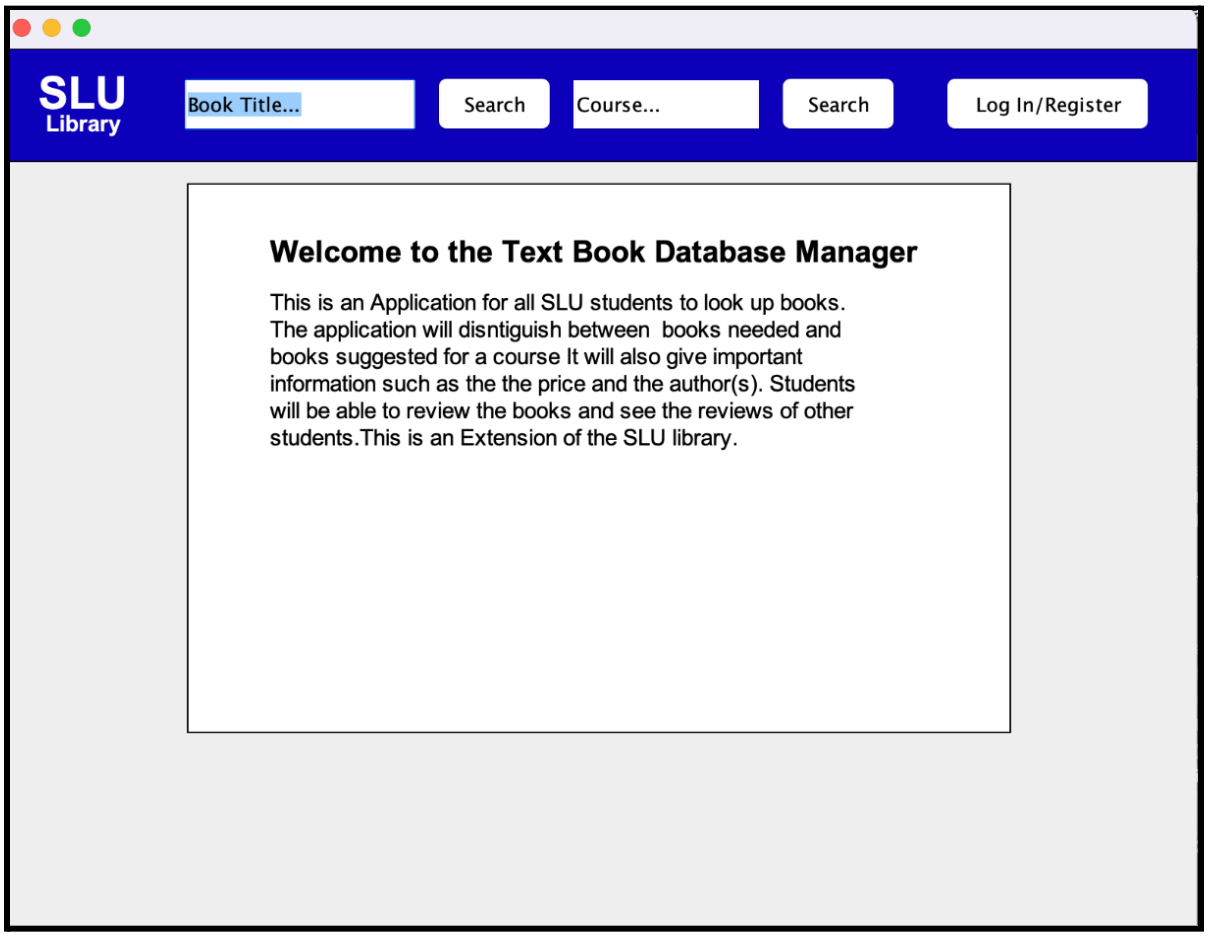
```
    public void insertCourse(int CRN, String title) {…
    public void insertRequiredRecBook(int ISBN, int CRN, int Status_ID) {…
    public void insertRequiredRecBook(int ISBN, int CRN, String Status) {…
    public void deleteAuthor(String fName, String lName) {…
    public void deleteAuthor(int Auth_ID) {…
    public void deleteTextbook(int ISBN) {…
    public void deleteEvaluation(int Student_ID, int ISBN) {…
    public void deleteStudent(int Student_ID) {…
    public void deleteCourse(int CRN) {…
    public void updateTextbook(int ISBN, Float price) {…
    public void updateRequiredRecBooks(int ISBN, int CRN, int Status_ID) {…
    public void updateRequiredRecBooks(int ISBN, int CRN, String Status_ID) {…
}
```

And here are two examples of those queries

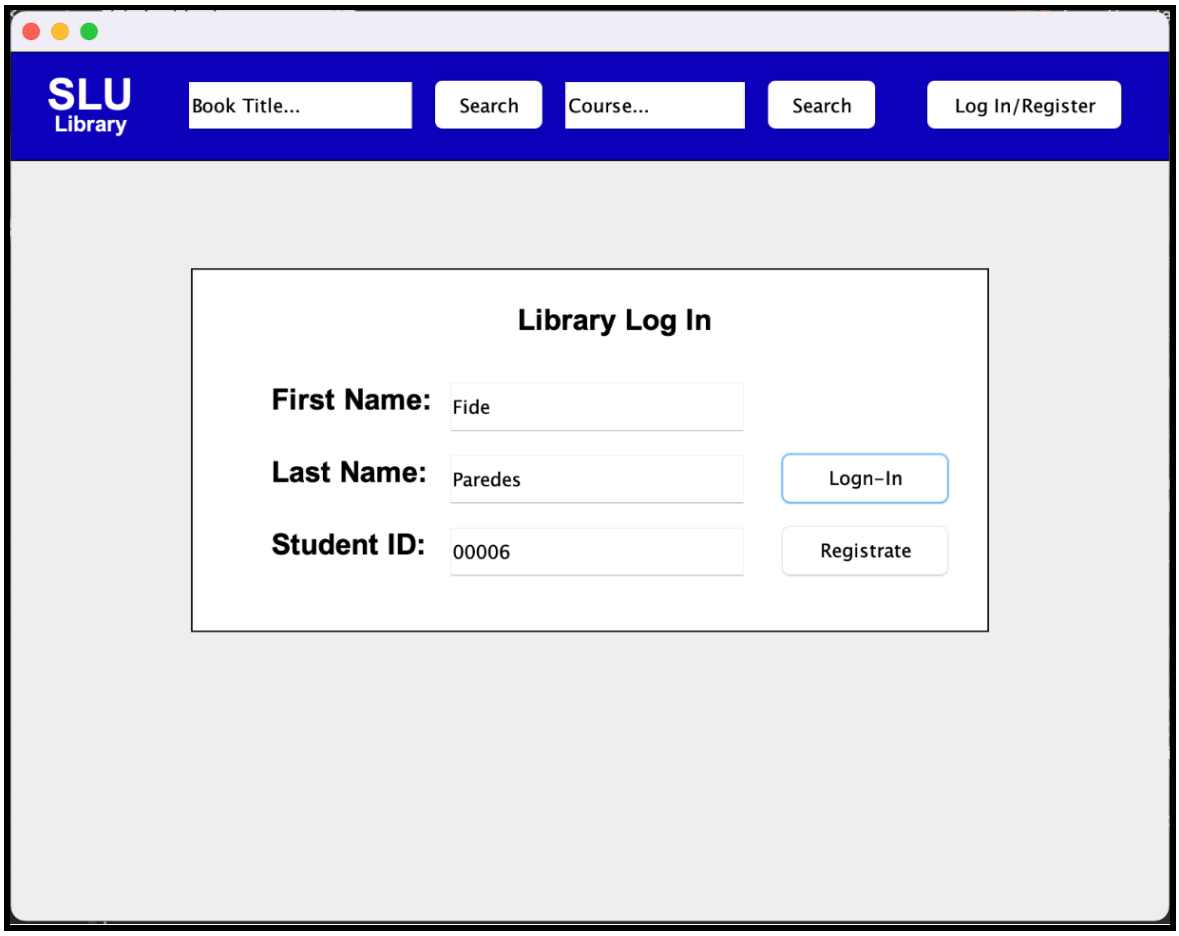
```
public void updateTextbook(int ISBN, Float price) {

    try {

        String query = "UPDATE Textbooks SET Price = ? WHERE ISBN = ?";
        PreparedStatement pstm = connection.getConnection().prepareStatement(query);

        pstm.setFloat(1, price);
        pstm.setInt(2, ISBN);

        pstm.executeUpdate();

    } catch (Exception e) {

        System.out.println(e);
        System.exit(1);
    }
}
```

```java
public ResultSet getDetailsAboutBook(int ISBN) {

    try {

        String query = "SELECT t.Title as 'Book', t.ISBN, t.Price, " +
                "c.Title as 'Course', s.Status, concat(stu.FName, ' ', stu.LName) as 'Student', " +
                "e.Evaluation " + " FROM Textbooks t " + "JOIN Required_Rec_Books rr " +
                "ON rr.ISBN = t.ISBN " + "JOIN Courses c ON c.CRN = rr.CRN " +
                "JOIN Required_Status s ON s.Status_ID = rr.Status_ID " +
                "JOIN Evaluations e ON e.ISBN = t.ISBN " + "JOIN Students stu " +
                "ON stu.Student_ID = e.Student_ID " + "JOIN Authors a " +
                "ON a.Auth_ID = t.Auth_ID " + "WHERE t.ISBN LIKE " + String.valueOf(ISBN);

        rs = connection.getStatement().executeQuery(query);
    } catch (Exception e) {

        System.out.println(e);
        System.exit(1);
    }
    return rs;
}
```

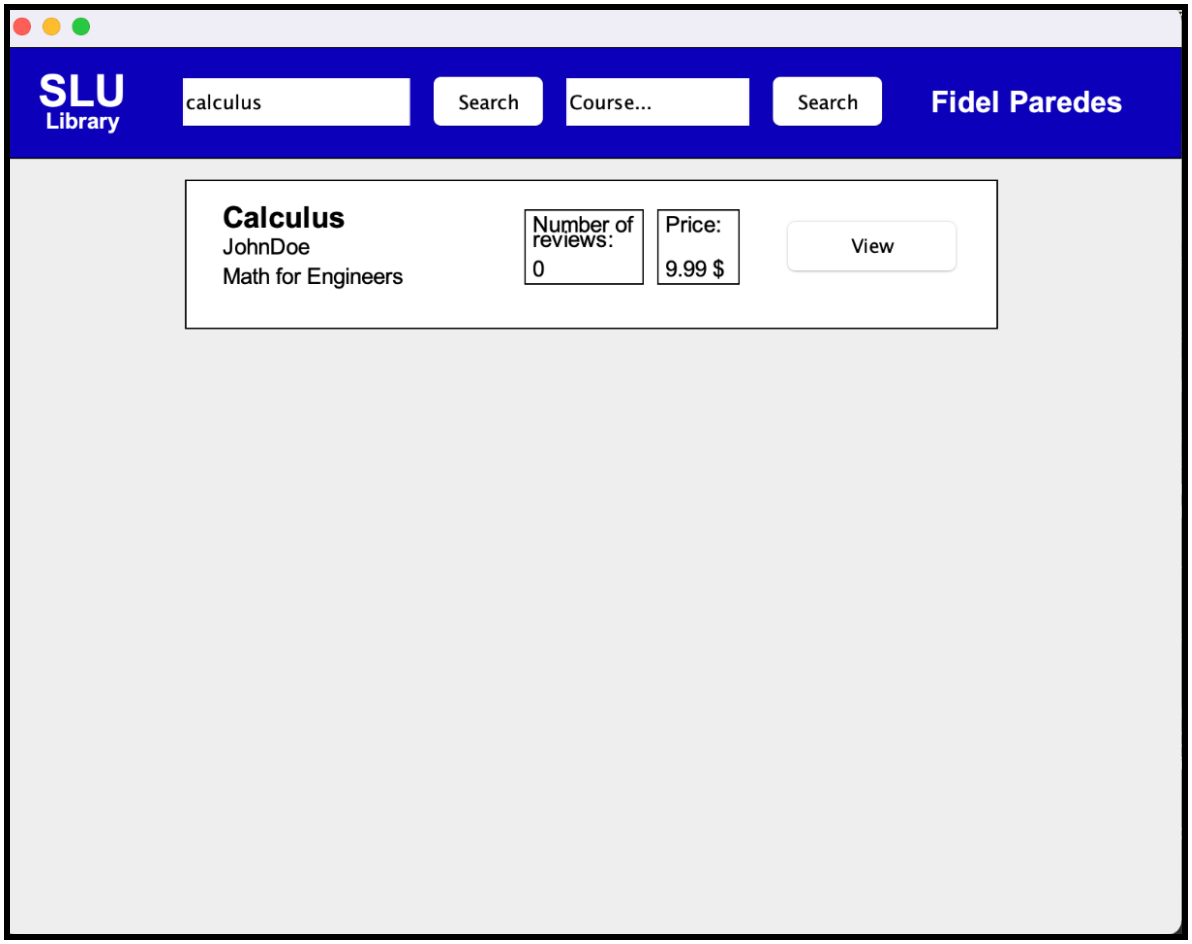
Display of Code:



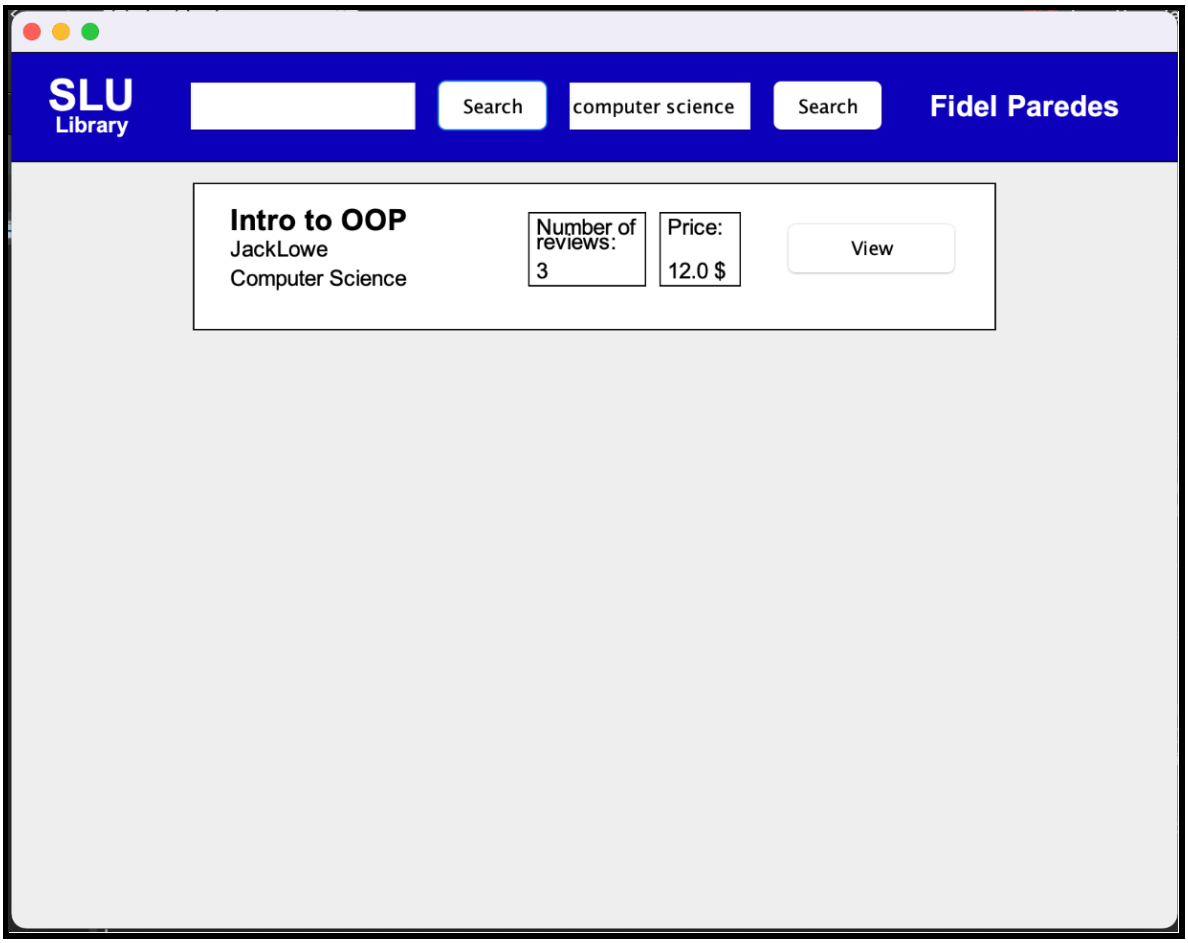
For the display of the application we used Java Swing and Graphics2D.
The first page that our code will show when it runs will be the welcome page. In this page a brief description of what our app is about will be displayed.



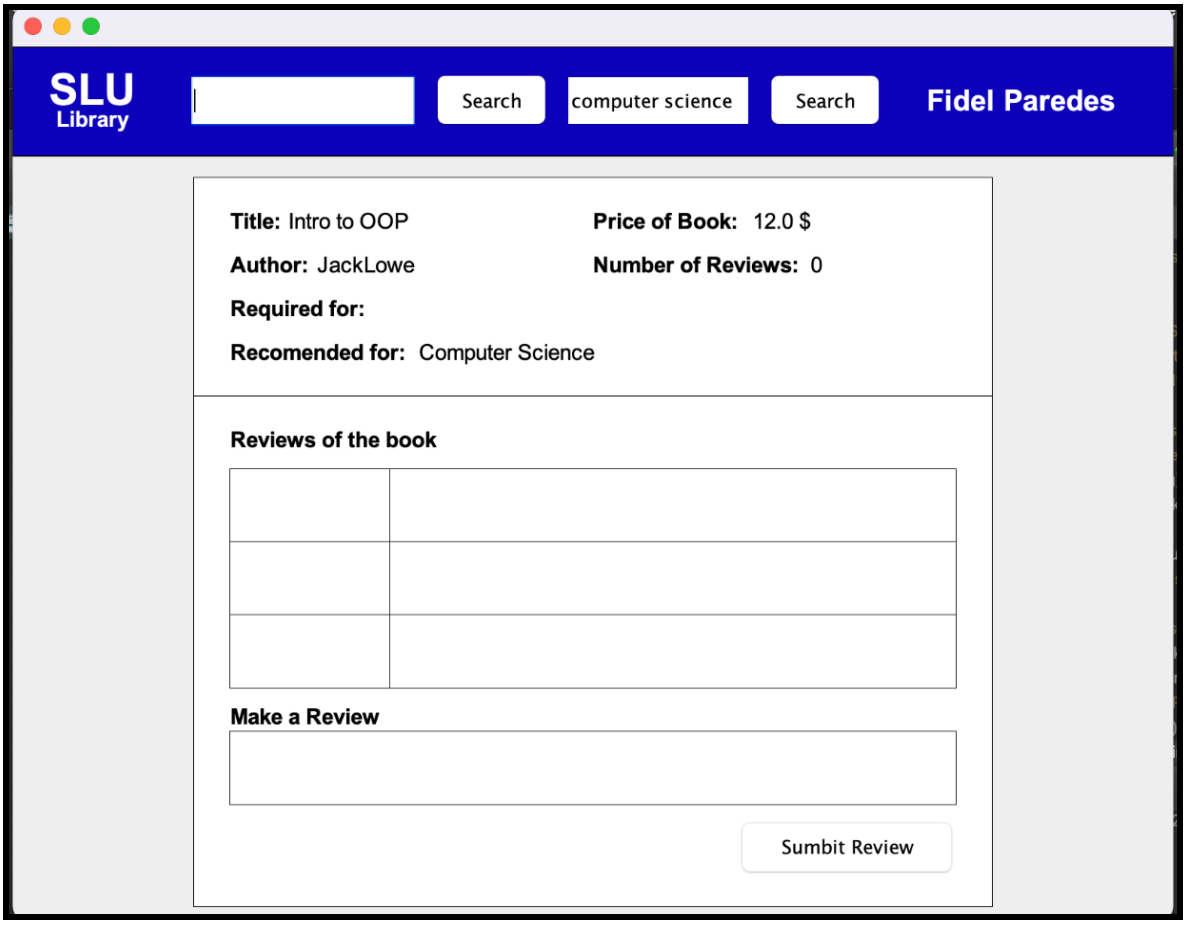
If the user clicks the Button “LogIn/Register” a Page will open where the user will be able to introduce their login/registration information. If the Register button is clicked then the information provided will be added to the database (the user will still need to login, registration does not automatically log in a user). If the user clicks login, our application will check that the student Id exists within the database and will login. After log-in and making a search, the name of the user will be displayed in one corner.



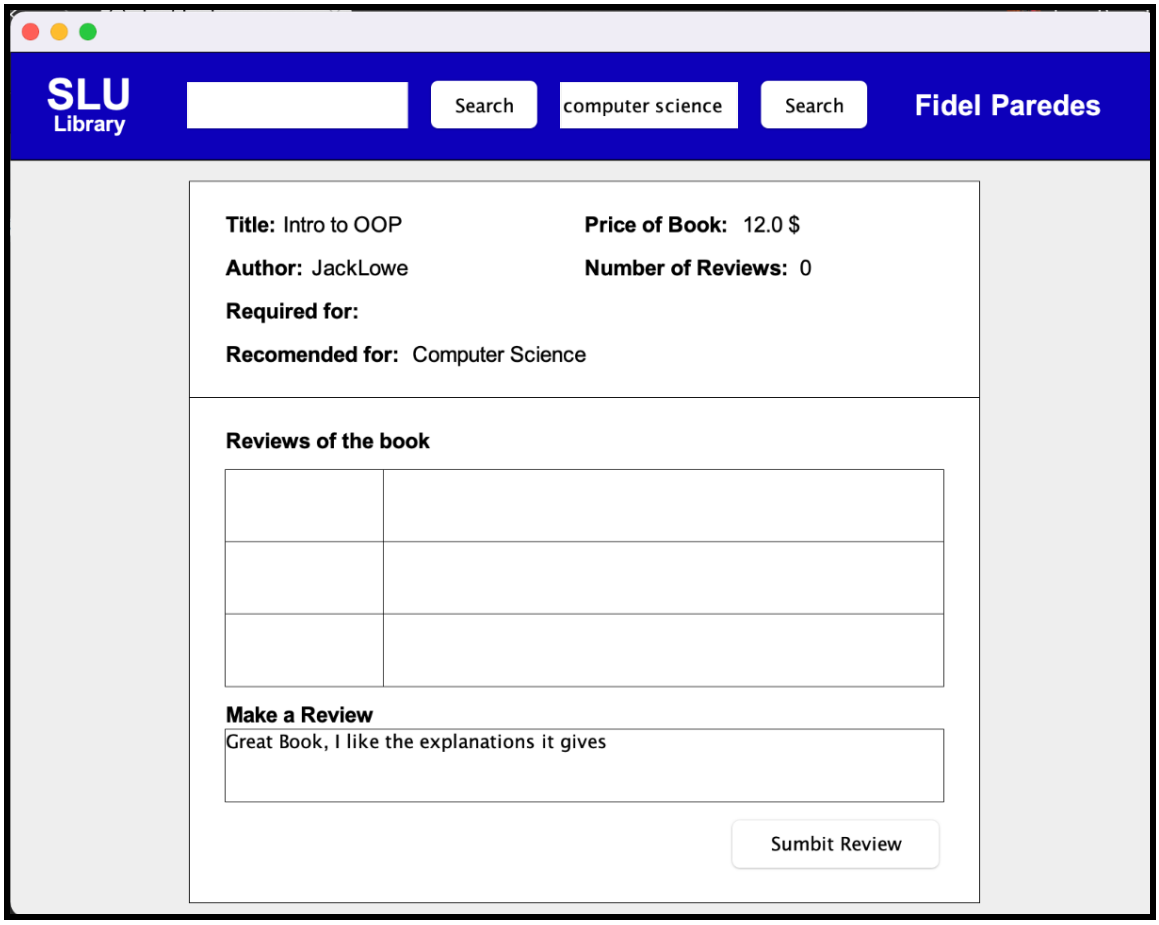
Our application has two ways of searching, searching by book title and searching by course. When the search is through book title, the application will check that the name given corresponds to the title of a book or books in our database. If found, for each book, our application will get all the book info (ISBN, Title, Author, and Price) and store it in a variable book, for a more accessible use. With the ISBN of the book, our application will access the tables of Required_Rec_Books in the database and get all the courses related to the book and store them in variables.



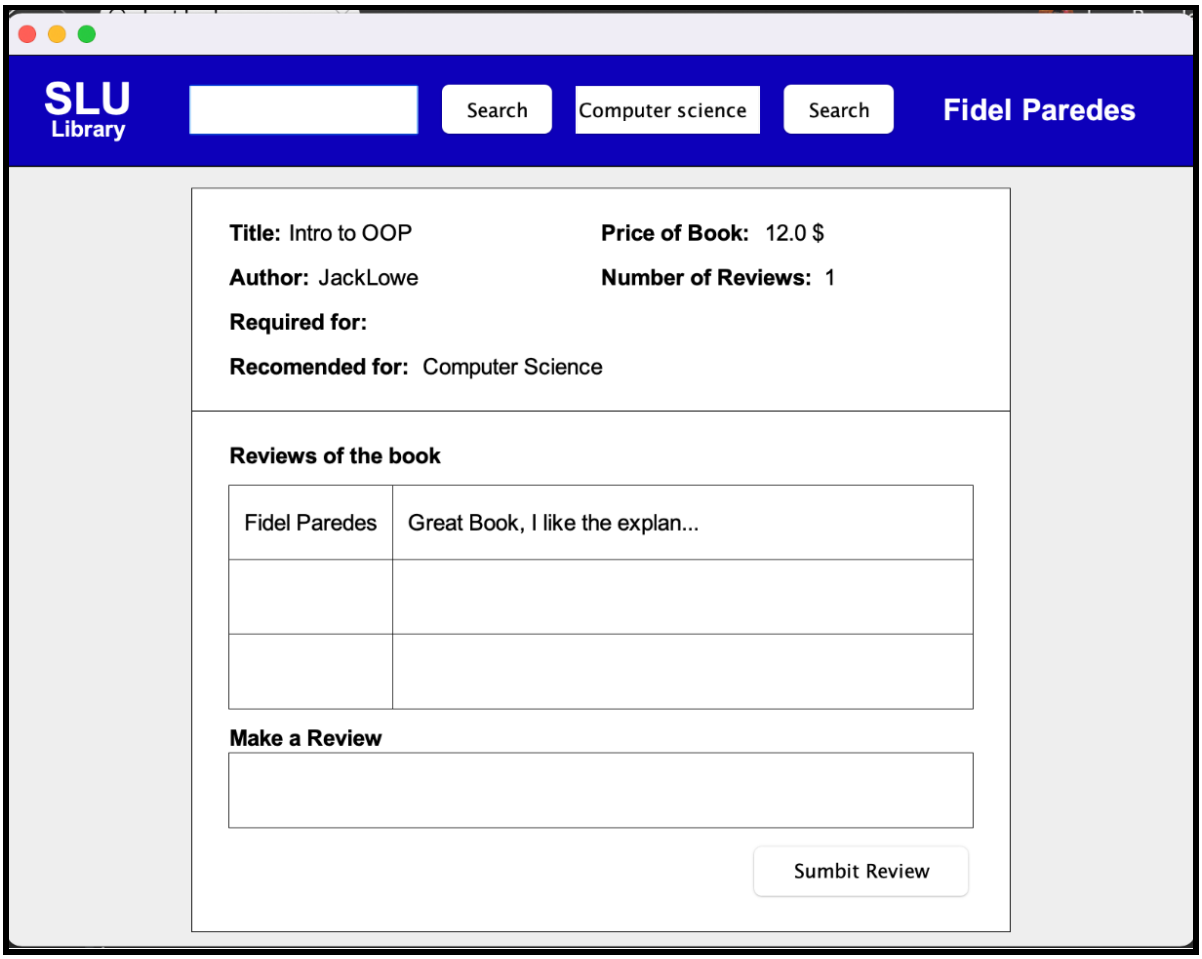
If the user clicks the button ‘View’, a page will open displaying all the book information, the evaluations of the book and comment section where the user will be able to make an evaluation. To display the evaluations, our application will access the database and search in the ‘Evaluations’ table all the evaluations that correspond to the ‘ISBN’ of the book. It will get all the information about the evaluation (ISBN, Student_ID, and Evaluation)and store it in a variable of the class Evaluation. The application will access the database again to get all the Student information with the Student_ID of the evaluation. Finally our application will display all the evaluations.



When the search is by course, our application will access the database and get all the info about the course searched (CRN and Title) and store it in a variable of the class Course. With the CRN it will access the database and look in the table Required_Rec_Books for all the books that relate to this class, and store them in the respective class variable. To distinguish between a course being ‘Required for’ and a course being ‘Recommended for’ a book, our application will look in the Required_Rec_Books table in the database to determine.



If a user wants to add an evaluation, the user will need to login to its user account first, write an evaluation and hit the ‘Submit Review’ button.



The evaluation will be displayed the next time the book is searched.

Outline of code of interface:

```
    mainPage(){
    //For the buttons actions
    public void actionPerformed(ActionEvent e) {
    //For the look of the different pages
    public void header() {
    public void searchedBooks() throws SQLException {
    public void welcome() {
    public void searchPage(int pos) {
    public void bookPage() throws SQLException {
    public void loginPage() {

    //To set the different Classes
    public void setClassStudent(ResultSet rs,student st) throws SQLExcepti
    public void setClassEvaluation(ResultSet rs,evaluation e0) throws SQLE
    public void setClassEvaluations(ResultSet rs,evaluation e1,evaluation
    public void setClassBook(ResultSet rs,book b0) throws SQLException {
    public void setClassCourse(ResultSet rs,course b0) throws SQLException
    public void setClassAuthor(ResultSet rs,author a0,book b0) throws SQLE
    public void setRequiredRecBook(ResultSet rs,requiredRecBook a0) throws
    //To clear the info in the Classes
    public void clearCourse(course b1,course b2) {
    public void clearBookReview(evaluation e0,evaluation e1,evaluation e2)
    public void clearStudents(student e0,student e1,student e2) {
```