Database Systems

Bookstore Project

Order Processing System

Team

Fares Mehanna 49
Salma Hesham 33
Dahlia Chehata 28

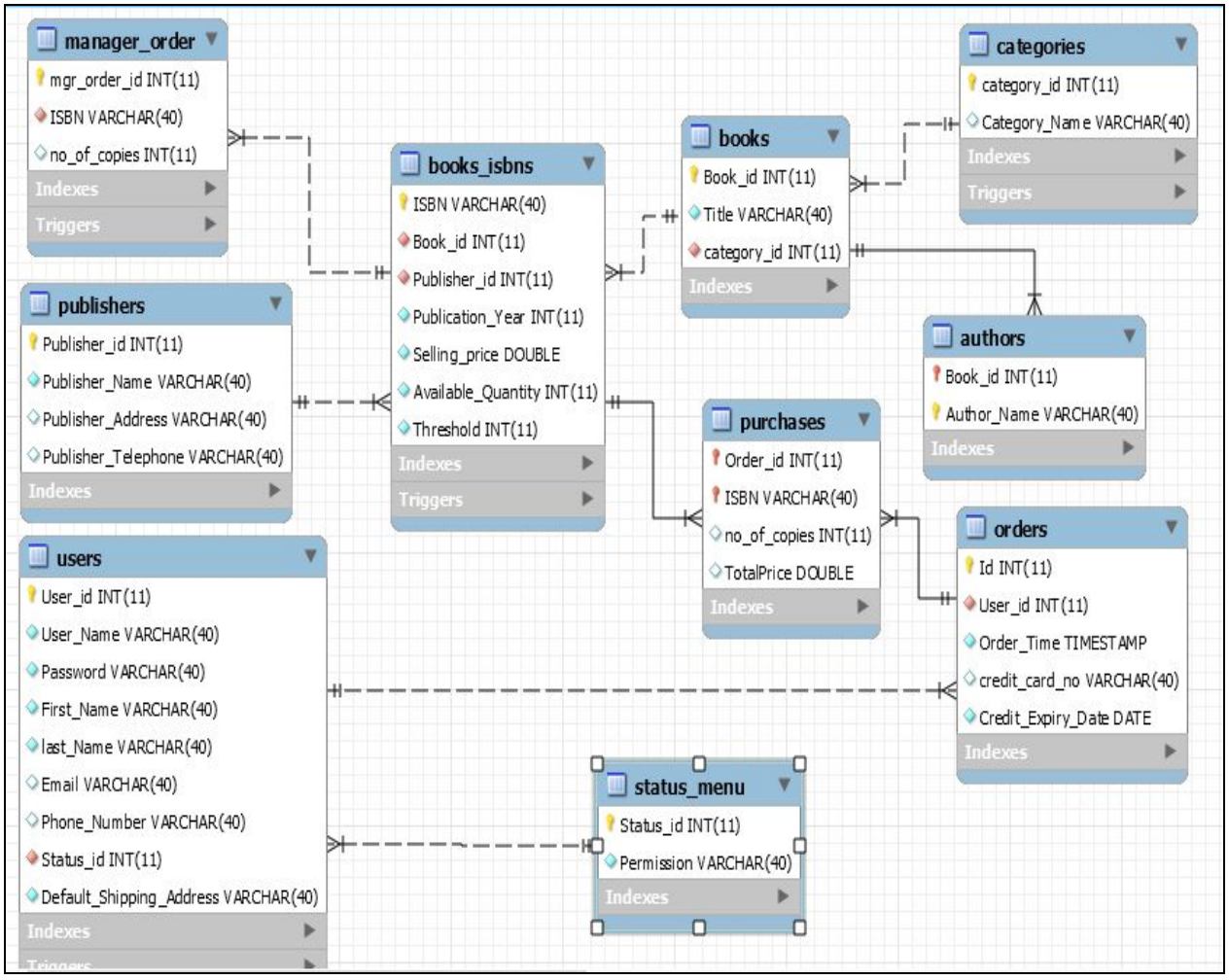
TABLE OF CONTENTS

Overview

The project simulates the database system for a simplified online bookstore that supports the following operations: books addition, modifying existing books,placing orders on books, confirm books, searching by different criteria, support user registration , edition of personal info, customers promotion to managers.

we follow the MVC design pattern where the gui is the view, the database and its supported functions is the model and the controller package maps between the two of them

ER Model



Detailed Description:

I.Mysql Part

1. Triggers

a. check negativity of book count

```
DROP trigger if exists negative_constraint;
DELIMITER $$
CREATE TRIGGER negative_constraint BEFORE update ON Books_ISBNs
FOR EACH ROW BEGIN
   IF (NEW.Available_Quantity < 0) THEN
         CALL raise_error;
   END IF;
END$$
DELIMITER ;
```

b. check if quantity is less than a threshold after update

```
DROP trigger if exists threshold_after_update;
DELIMITER $$
CREATE TRIGGER threshold_after_update AFTER update ON Books_ISBNs
FOR EACH ROW BEGIN
 declare ordered_quantity int;
 set ordered_quantity=0;
 select no_of_copies into ordered_quantity
 from manager_order natural join books_ISBNs
 where ISBN = NEW.ISBN;
   IF (NEW.Available_Quantity < OLD.Threshold+ordered_quantity) THEN
      if ordered_quantity=0 then
         insert into book_store.manager_order (ISBN,no_of_copies)
values(new.ISBN,(OLD.Available_Quantity-NEW.Available_Quantity)*1.5);
      else
       update book_store.manager_order
       set no_of_copies =
no_of_copies+(OLD.Available_Quantity-NEW.Available_Quantity)*1.5
       where manager_order.ISBN = NEW.ISBN;
       END IF;
   END IF;
END$$
DELIMITER ;
```

c. check if quantity is less than a threshold after insertion

```
DROP trigger if exists threshold_after_insert;
DELIMITER $$
CREATE TRIGGER threshold_after_insert AFTER insert ON Books_ISBNs
FOR EACH ROW BEGIN
   IF (NEW.Available_Quantity < NEW.Threshold) THEN
```

```
        insert into book_store.manager_order (ISBN,no_of_copies)
values(NEW.ISBN,(NEW.Threshold - NEW.Available_Quantity)*1.5);
   END IF;
END$$
DELIMITER ;
```

d. check if the order is confirmed after deletion

```
DROP trigger if exists delete_manager_order ;
DELIMITER $$
CREATE TRIGGER delete_manager_order AFTER DELETE ON manager_order
FOR EACH ROW BEGIN
   UPDATE Books_ISBNs SET Books_ISBNs.Available_Quantity =
(Books_ISBNs.Available_Quantity + OLD.no_of_copies) WHERE
   Books_ISBNs.ISBN = OLD.ISBN;
END$$
DELIMITER ;
```

e. check if email is in a valid format before update

```
DROP trigger if exists valid_email_before_update ;
DELIMITER $$
CREATE TRIGGER valid_email_before_update BEFORE UPDATE ON Users
FOR EACH ROW BEGIN
   IF (NEW.Email NOT LIKE '%_@_%._%' ) THEN
            CALL raise_error;
   END IF;
END$$
DELIMITER ;
```

f. check if email is in a valid format before insertion

```
DROP trigger if exists valid_email_before_insert ;
DELIMITER $$
CREATE TRIGGER valid_email_before_insert BEFORE UPDATE ON Users
FOR EACH ROW BEGIN
   IF (NEW.Email NOT LIKE '%_@_%._%' ) THEN
            CALL raise_error;
   END IF;
END$$
DELIMITER;
```

g. check if the category type is valid before insertion

```
DROP trigger if exists valid_category_before_insert;
DELIMITER $$
CREATE TRIGGER  valid_category_before_insert BEFORE insert ON
categories
FOR EACH ROW BEGIN
   IF (NEW.Category_Name !='Science' and NEW.Category_Name !='Art' and
NEW.Category_Name !='Religion' and
   NEW.Category_Name != 'History' and NEW.Category_Name !=
'Geography') THEN
            CALL raise_error;
   END IF;
END$$
DELIMITER ;
```

h. check if the category type is valid before update

```
DROP trigger if exists valid_category_before_update;
DELIMITER $$
CREATE TRIGGER valid_category_before_update BEFORE UPDATE ON
categories
FOR EACH ROW BEGIN
   IF (NEW.Category_Name!='Science' and NEW.Category_Name !='Art' and
NEW.Category_Name !='Religion' and
   NEW.Category_Name != 'History' and NEW.Category_Name !=
'Geography') THEN
            CALL raise_error;
   END IF;
END$$
DELIMITER ;
```

2. Indices

a. FULL TEXT INDEX

i. The user can search for a book by ISBN, and title. The user can search for books of a specific Category, author or publisher. So it is used to index ISBN, title, category,author name, and publisher name

1. books.Title
2. Books_ISBNs.ISBN
3. Authors.Author_Name
4. Publishers.Publisher_Name
5. Categories.Category_Name
6. Books_ISBNs.Threshold
7. Books_ISBNs.Available_Quantity

b. Index

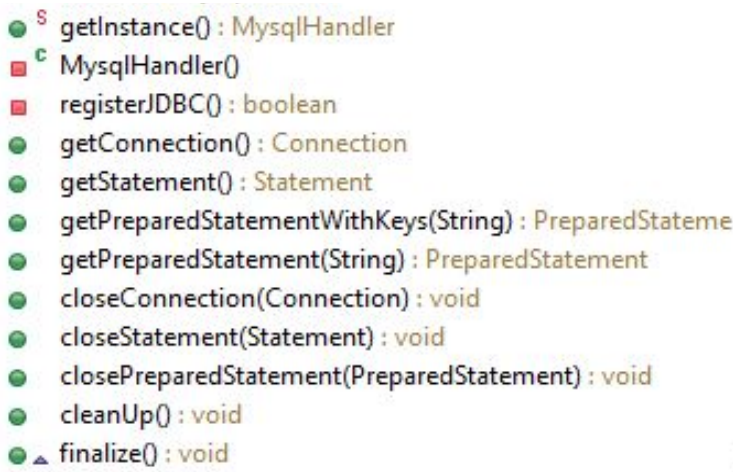
- **Authors**
  - Authors.Author_Name
  - Authors.Book_id
- **Books**
  - Books_ISBNs.Book_id
  - books.Book_id
  - Books_ISBNs.ISBN
  - books.category_id
  - Books_ISBNs.Publication_Year
  - Books_ISBNs.Selling_price
  - Books_ISBNs.Publisher_id
  - Books_ISBNs.Threshold
  - Books_ISBNs.Available_Quantity
  - books.Title
- **Categories**
  - Categories.category_id
  - Categories.Category_Name
- **manager_order**
  - manager_order.mgr_order_id
- **orders**
  - orders.Id
  - orders.User_id
  - orders.credit_card_no
  - orders.Order_Time
- **purchases**
  - purchases.Order_id
  - purchases.ISBN
- **Publishers**
  - Publishers.Publisher_id
  - Publishers.Publisher_Name
- **Users**
  - Users.User_id
  - Users.User_Name
  - Users.Email
- **Status_Menu**
  - Status_Menu.Status_id

II. Java Part

A. backend

1. Mysql package

- **MysqlHandler class**
  - implements a singleton design pattern .
  - This class is responsible for creating the database connection .
  - Any process can take a connection from the one created or statements from the current connection.
  - It contains an instance from the error handler, a hashset of Statements used for static SQL statements at runtime , a hashset of PreparedStatements used for parameterized statements to take input parameters at runtime and a hashset of connections.
  - Each operation could have initiated a new connection , but in this model we used a single connection to prevent memory overhead.
  - These hashsets are containers for used statements or connections to avoid resource leaks.
  - Main functions :

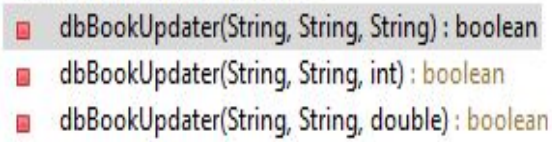


2. ModelsImplementations package

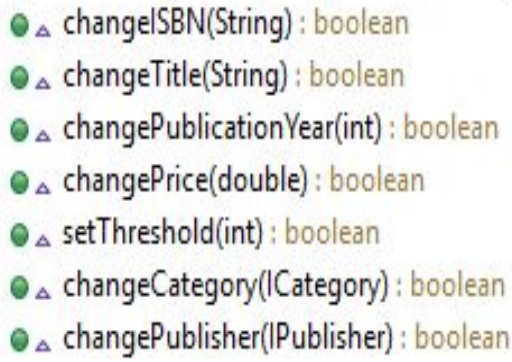
- **Author**
  1. The class contains error handler instance and PreparedStatement
  2. it allows the following operations : get author name, get the list of books written by a given author (perform select on Author_Name), remove a selected book by Id and Author_Name and books insertion in the authors relation
  3. finalize operation closes the prepared statement

- **AuthorManager**
  1. The class contains error handler instance and PreparedStatement
  2. it allows the following operations: get or add an author, get all authors and finalize
- **Book**
  1. The class contains error handler instance and PreparedStatement
  2. **private ResultSet dbBookGetter()** : returns the ResultSet of the select from statement
  3. these functions returns the different attributes in Book relation

  4. these functions are used for the update query statement



  5. these functions implement the latter ones and checks for the existence of the data to be updated before any modification

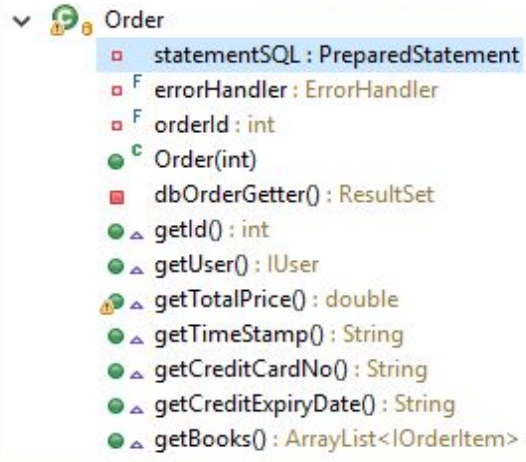


6. **addAuthor** function creates new Author for the book, **removeAuthor** function removes an author from the current book and **finalize** function closes the prepares statement.

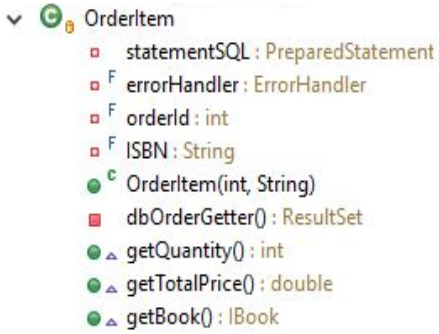
- **BookManager**
  1. The class contains mainly error handler instance and PreparedStatement
  2. **public IBooksGetter getBooks() :** returns the book list after resolving any conjunctive condition operation
  3. **public IBook addBook(String title, ICategory category, String ISBN, IPublisher publisher, int publicationYear, double price, int availableQuantity, int threshold, ArrayList<IAuthor> authors)** performs the insert sql statement
  4. **public ArrayList<IBook> getTop(int numOfBooks, String startingDate, String endingDate)** not supported
- **BooksGetter**
  1. The class contains error handler instance and PreparedStatement
  2. performs the conjunctive condition operation
  3. allows different search techniques for books by author name, publisher name, ISBN, title and category using FULL TEXT INDEX
- **CartManager**
  1. **AddBook()** to cart : checks if the available quantity of the required book allows it to be ordered and placed to cart, checks if the user has already items in its cart and add the new order or create a new cart to the user

  if the book already is in the list, its quantity is increased
  else a new order is to be put in the cart
  2. **flushCart ():**remove any items in the required user's cart
  3. **getUserCart ()** : returns a list of items in the user's cart
- **Category**
  1. The class contains error handler instance and PreparedStatement
  2. **dbCategoryGetter()**: is used to get the category title in the category relation by ID . It is used to speed up the search process
  3. **getID()**: get the category by ID
  4. **getName()**: get the category by name
  5. **changeName()**: is used with the sql update statement to change a book's category.

- ## CategoryManager
    1. The class contains error handler instance and PreparedStatement
    2. **getCategoryByName()**: returns the required category
    3. **getCategoryById()** : returns the required category by id
    4. **getAllCategories()** : returns the category relation with IDs and related category name
    5. **addCategory ()**
    6. **removeCategory()**
- ## Order
    1. The class contains error handler instance and PreparedStatement
    2. get the user's order by its id
    3. the main functions used



- ## OrderItem
    1. The class contains error handler instance and PreparedStatement
    2. used to handle each different book in the same order
    3. its main functions:

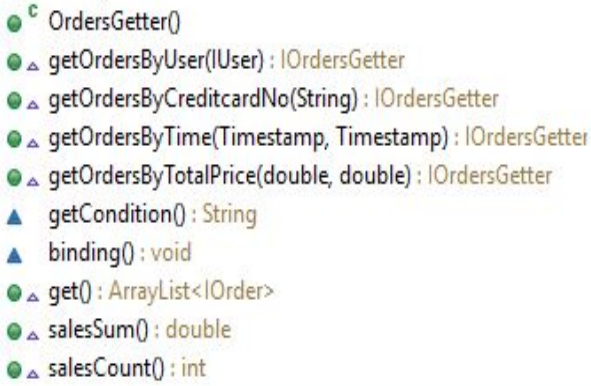


OrderManager

1. contains an instance of the error handler
2. **getOrders()**: returns the user's orders
3. **getOrderbyId()**: returns a single order
4. **addOrder()**: used with the sql insert statement in the orders relation

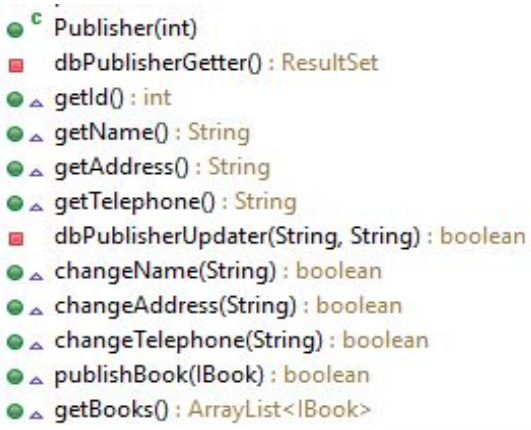
OrderGetter

1. handles different search operations for orders
2. its main functions:



Publisher

1. The class contains error handler instance and PreparedStatement
2. get functions used to get the publisher by different attributes
3. change functions used with the sql update statements for the different publisher's attributes
4. its main functions

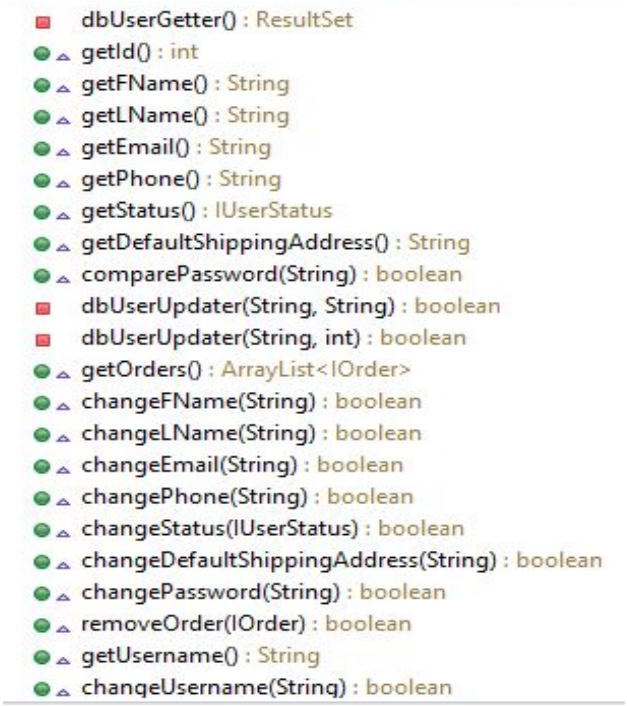


PublisherManager

1. The class contains error handler instance and PreparedStatement

2. **getAllPublishers()** : returns a ResultSet of the publisher's relation
3. **addPublisher()**: used with the sql insert statement

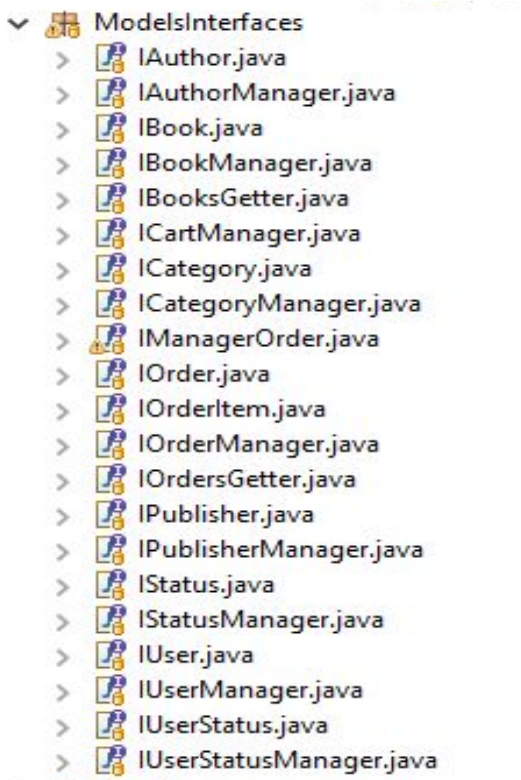
- **User**
  1. The class contains error handler instance and PreparedStatement
  2. get functions used to get the user by different attributes
  3. change functions used with the sql update statements for the different user's attributes
  4. its main functions:



- **UserManager**
  1. **getAllUsers()** : return the result set of the user relation
  2. **addUser()**: for the insert statement
  3. getters to return user using different attributes (id, email, username, email and password)
- **UserStatus**
  1. get the user status : customer or manager
  2. **changeName()**: used to update user's status (customer/manager)
  3. **getId()**
  4. **getName()**

- **UserStatusManager**
  1. **getAllUserStatus():** returns the result set of all the users status
  2. **addStatus()**: for the insert statement
- **singleOrder**
  1. contains manager order id, quantity and book
- **ManagerOrder**
  1. **addOrder():** insert into manager_order relation the book isbn and the number of copies
  2. **confirmOrder():** delete the manager order and adds the number of copies to the available quantity of books
  3. **getManagerOrder():** returns the result set of manager_order relation

- **ModelsInterfaces package**
  - the interfaces for the classes explained in ModelsImplementations package



- **HelperClasses package**
  - **ErrorHandler**
    1. there 2 tupes of errors : the first is reported and the execution is resumed and the second causes the program to terminate
    2. public void report(String errorOrigin, String errorMessage)
    3. public void terminate()
  - **NotFound**
    1. extends throwable class
  - **SHA256**
    1. used to hash the users' passwords

B. front-end

- **gui package**
  - responsible for the view part in MVC design pattern
  - contains the book handling package and the user handling package
  - the book handling is responsible for all books operations on the gui
  - the user handling package contains the login page, sign in page and all the forms that supports the user's operations
  - the main function starts in the GUI_controller class
- **controller**
  - responsible for the controller part in MVC design pattern
  - maps between the model and the gui
  - every main operation has its own controller as book modification, book addition etc…..

concurrency/transaction control measures

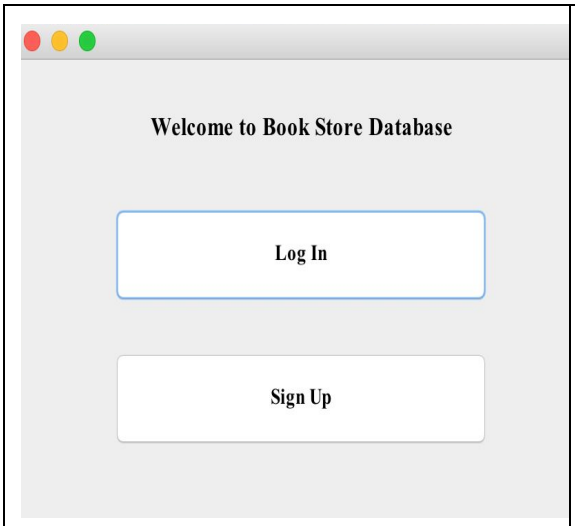
a. **transactions** are used with the sql insert statements in multiple related relations or with the operations affecting the database logical consistency. So Each operation is performed entirely without interruption or else the transaction rollbacks
b. **concurrency** is maintained between transactions to avoid race conditions between multiple processes
c. A **race condition** can occur without the presence of a transaction as when a process tries to read data already deleted or altered by another process, **NotFound** exception is used in the mapping layer to prevent such cases
d. No data is cached in the object so if process A changes book title for example the process B read the book title from different object, it will read the correct value
e. we depend on MYSQL to handle the data consistency between processes and if we are to build a caching system,we must ensure a synchronized update or invalidate between all processes
f. **commit**:
    - In BookManager class . addBook () function requires transaction to make sure everything is working correctly.
    - A connection is used to start the transaction , add the required books with their fields information and then the transaction commits

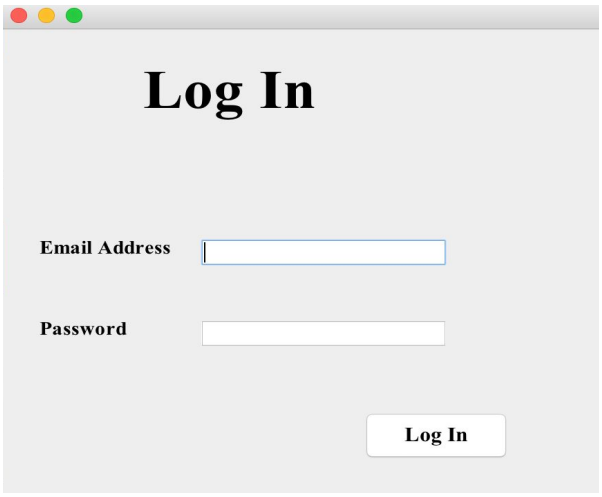
```
//try to add the book
try {
    conn.commit();
} catch (SQLException ex) {
    errorHandler.report("Book Manager Class", ex.getMessage());
    //close the opened connection
```

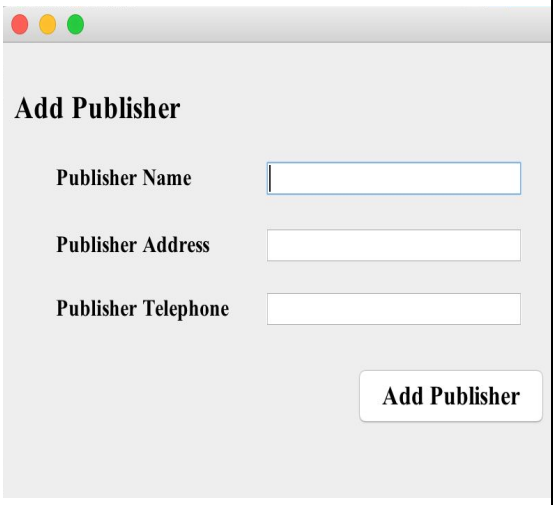
g. **rollback**
    - In each case we use transactions as in Book insertion, rollback will be used in case of any failure during the transaction.

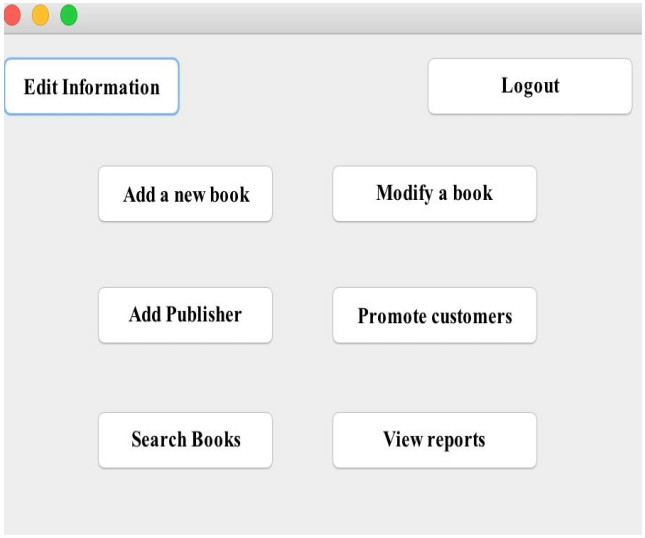
```
//add the authors.
try {
    String sqlQuery = "INSERT INTO `Authors` " +
    "(`Book_id`,`Author_Name`) VALUES (@key,?);";

    statement = conn.prepareStatement(sqlQuery);

    for(int i=0;i<authors.size();i++){
        statement.setString(1, authors.get(i).getName());
        statement.executeUpdate();
    }

} catch (SQLException ex) {
    errorHandler.report("Book Manager Class", ex.getMessage());
    try{conn.rollback();}catch(SQLException e){}
    //close the opened connection
    MysqlHandler.getInstance().closeConnection(conn);
    return null;
}
```

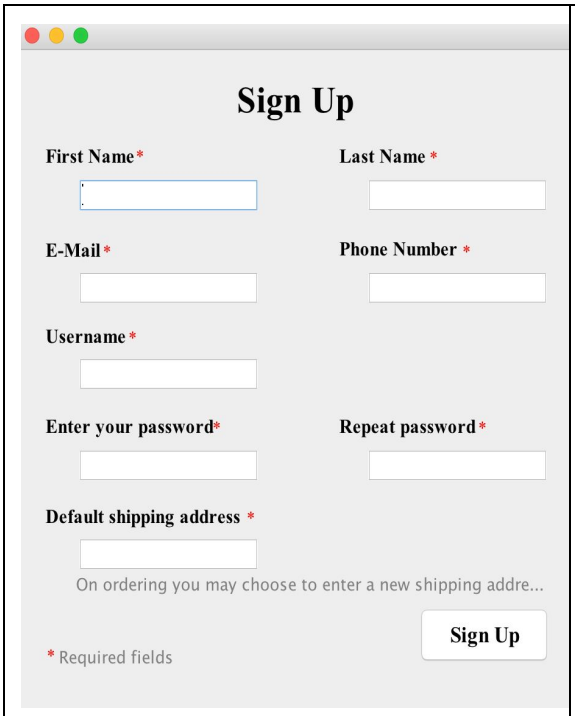
Sample screenshots:

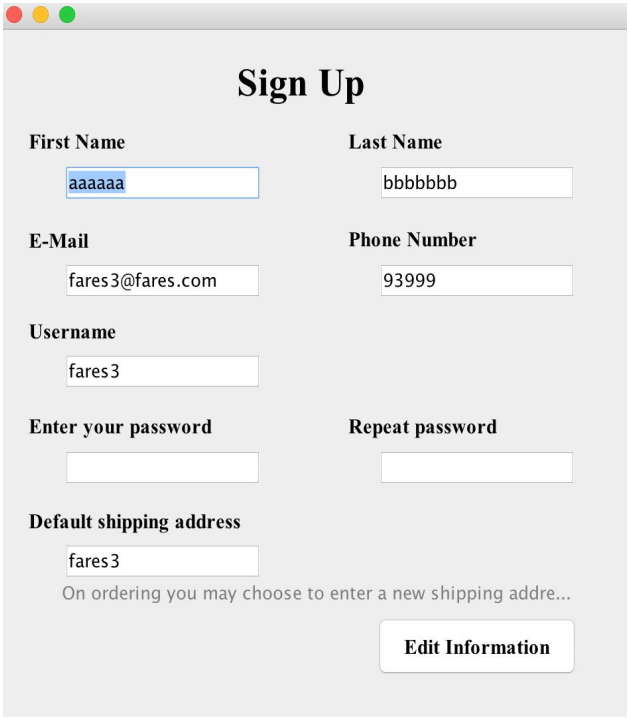


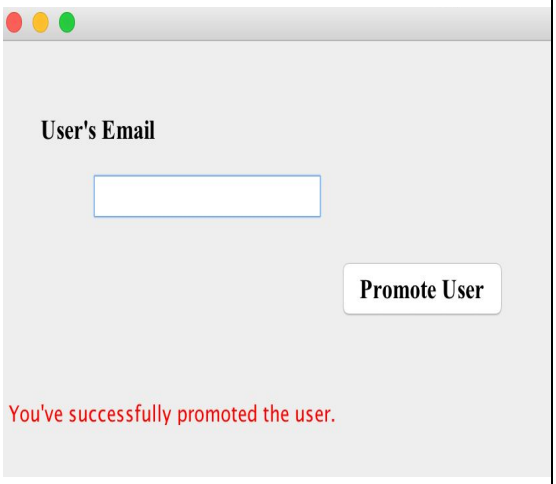


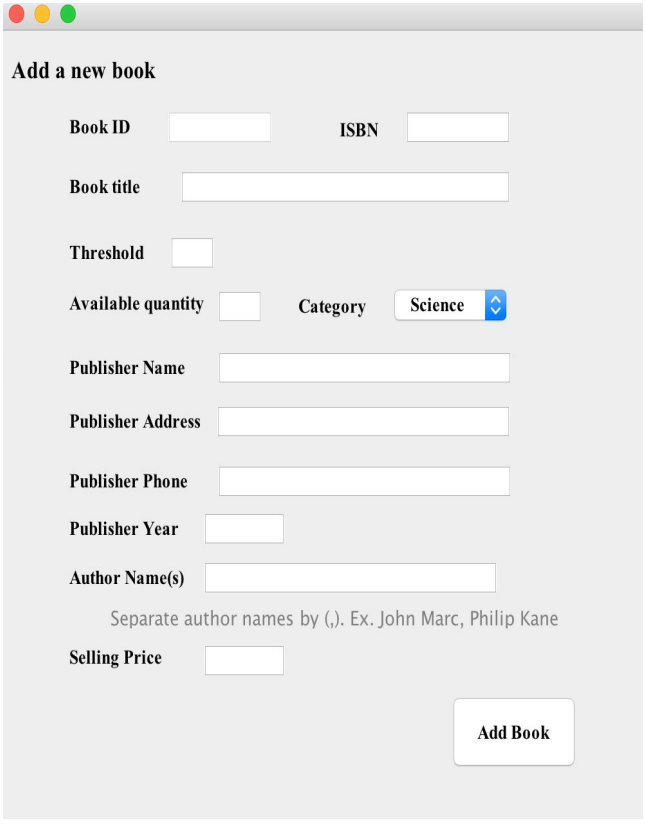





Sign Up
First Name*
Last Name *
E-Mail*
Phone Number *
Username *
Enter your password*
Repeat password*
Default shipping address *
On ordering you may choose to enter a new shipping addre...
Sign Up
* Required fields


Sign Up
First Name
aaaaaa
Last Name
bbbbbbb
E-Mail
fares3@fares.com
Phone Number
93999
Username
fares3
Enter your password
Repeat password
Default shipping address
fares3
On ordering you may choose to enter a new shipping addre...
Edit Information


User's Email
Promote User
You've successfully promoted the user.


Add a new book
Book ID
ISBN
Book title
Threshold
Available quantity
Category
Science
Publisher Name
Publisher Address
Publisher Phone
Publisher Year
Author Name(s)
Separate author names by (,). Ex. John Marc, Philip Kane
Selling Price
Add Book