

Library Management System Project

Iteration 3 – FINAL ITERATION

Submitted by: Arshnoor Batra (434007284) and Varun Sahni (834006628)

Introduction:

This Server-Client project is a comprehensive implementation of a microservices architecture designed to efficiently manage and interact with various entities in a library system. The project is structured as a collection of microservices that handle distinct functionalities, ensuring scalability, flexibility, and ease of maintenance.

Server Components:

1. User Server (Port 5057):

- a. Manages user-related operations such as user authentication.
- b. Utilizes a microservice approach to handle user data retrieval.

2. Book Server (Port 5058):

- a. Manages book-related operations including retrieval, creation, and modification.
- b. Implements RESTful services for seamless integration with clients.

3. OrderBook Server (Port 5059):

- a. Handles order-related processes, such as order creation and receipt generation.
- b. Utilizes microservices for managing orders, students, and receipts.

Client Components:

The project includes two distinct clients, each tailored for specific interfaces:

1. Desktop Application Client:

- a. Developed in Java, the desktop client interacts with the microservices via HTTP requests.
- b. Facilitates functionalities such as updating book details, placing orders, and retrieving student information.
- c. Employs a socket connection to communicate with the main server, showcasing versatility in communication protocols.

2. Web Browser Client:

- a. Comprising HTML files (index.html, loadsave.html, login.html, manage.html, and orderbook.html) and a Node.js server (server.js).

- b.** Offers a user-friendly web interface for accessing library services.
- c.** Uses asynchronous HTTP requests to interact with the microservices, enabling seamless integration with modern web browsers.

DataAdapter for Remote Communication:

The '**RemoteDataAdapter**' class acts as a crucial bridge between the desktop application and the microservices. It provides methods for connecting to the main server, updating book and order details, and retrieving user and student information.

Key Features:

- 1. Microservices Architecture:** The project adopts a microservices architecture, enhancing modularity and enabling independent development and deployment of each service.
- 2. Java Desktop Client:** The desktop client, written in Java, showcases the versatility of the microservices by utilizing both HTTP and socket-based communication.
- 3. Web Browser Client:** The web client provides an intuitive and accessible interface for users to interact with the library system.
- 4. Data Access and Transfer:** The **RemoteDataAdapter** class ensures smooth communication between the desktop client and the microservices, employing HTTP requests for data retrieval and updates.

Conclusion:

This Server-Client project illustrates the power and adaptability of microservices in building a robust and scalable library management system. It combines a desktop application and a web client to cater to diverse user preferences, showcasing the flexibility and efficiency of the microservices architecture in modern software development.

ASSIGNMENT QUESTIONS:

Ques 1) Requirements: Description with UI sketch of all use cases. Use cases should be described with the standard format including a short description and a sequence of user actions and system reactions. Each user action or system reaction might include a UI sketch.

Answer 1) All use cases of the library management are described below:

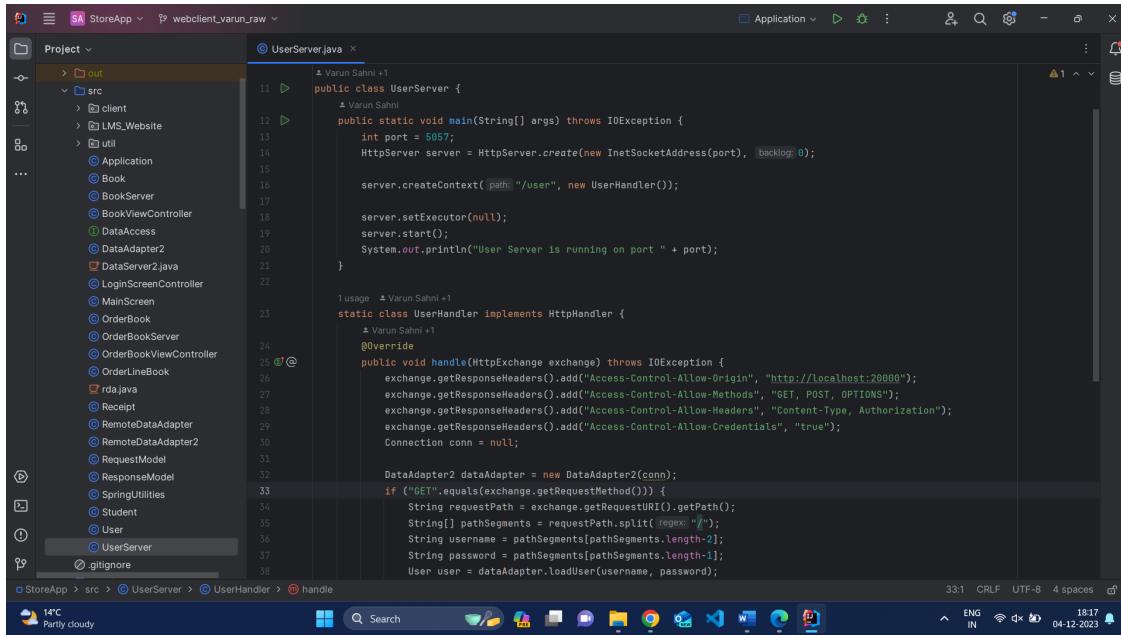
1st use case:

Use Case	Login	
Actor	Administrator	
Description	The administrator shall be able to log in to the library management system application.	
Precondition	The username and password of the administrator asking to be logged in should be present in the Redis database so that it can be matched against it.	
Ordinary Sequence	Step	Action
	1	The User server will be started to process the data.
	2	The application will be started to load the UI.
	3	A login screen window will appear on the screen.
	4	The admin will enter login details in the login window.
	5	The admin will click on the login button.
	6	The login details of the admin will be verified against the information present in the database.
Postcondition	The admin will be able to log into the application.	
Comment	If the login details entered by the admin do not match the database, the “The user does not exist!” window pops up.	

Actor Action	System Response
1 Admin enters username	
2 Admin enters a password	
3 Admin clicks on the login button	4 validates the user details
	5 takes admin to the main screen

UI Sketches for all the Action Steps:

1)



The screenshot shows a Java IDE interface with the project 'StoreApp' open. The 'UserServer.java' file is the active editor. The code implements a UserServer class that starts an HttpServer on port 5057, creates a context at '/user', and sets up a UserHandler. The UserHandler handles GET requests by splitting the path and calling a DataAdapter2 to load a user. The code uses annotations like Varun Sahn +1 and @Override.

```
public class UserServer {
    public static void main(String[] args) throws IOException {
        int port = 5057;
        HttpServer server = HttpServer.create(new InetSocketAddress(port), backlog: 0);

        server.createContext(path: "/user", new UserHandler());

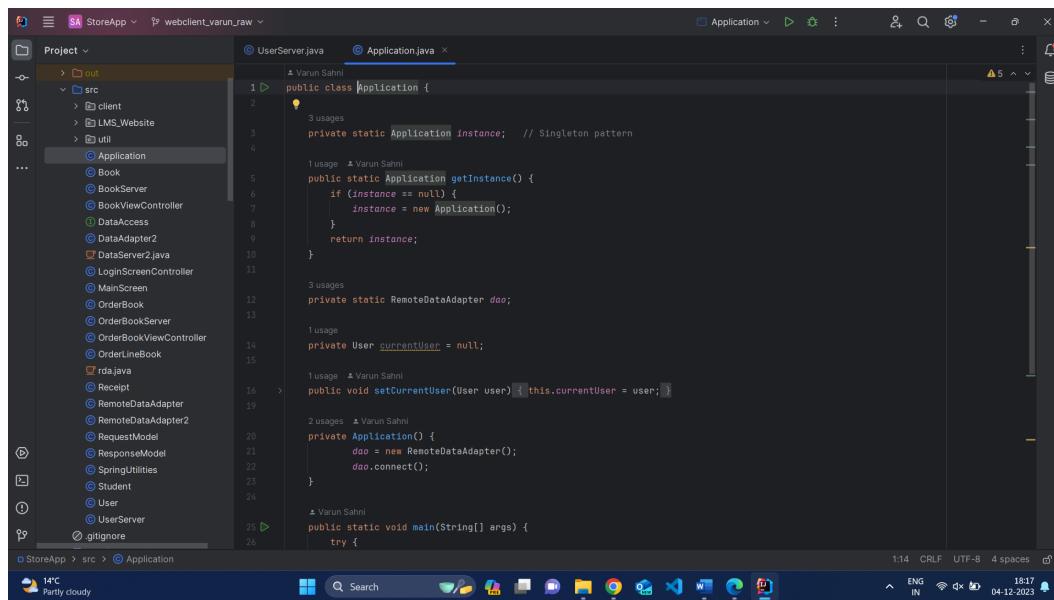
        server.setExecutor(null);
        server.start();
        System.out.println("User Server is running on port " + port);
    }
}

static class UserHandler implements HttpHandler {
    @Override
    public void handle(HttpExchange exchange) throws IOException {
        exchange.getResponseHeaders().add("Access-Control-Allow-Origin", "http://localhost:2000");
        exchange.getResponseHeaders().add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
        exchange.getResponseHeaders().add("Access-Control-Allow-Headers", "Content-Type, Authorization");
        exchange.getResponseHeaders().add("Access-Control-Allow-Credentials", "true");
        Connection conn = null;

        DataAdapter2 dataAdapter = new DataAdapter2(conn);
        if ("GET".equals(exchange.getRequestMethod())) {
            String requestPath = exchange.getRequestURI().getPath();
            String[] pathSegments = requestPath.split(regex: "/");
            String username = pathSegments[pathSegments.length - 2];
            String password = pathSegments[pathSegments.length - 1];
            User user = dataAdapter.loadUser(username, password);
        }
    }
}
```

The user server is loaded to start the application.

2)



The screenshot shows the same Java IDE interface with the 'Application.java' file as the active editor. The code defines a static Application instance using a Singleton pattern. It provides a getInstance() method and a setCurrentUser() method. The Application constructor initializes a RemoteDataAdapter. The main() method is also shown.

```
public class Application {
    private static Application instance; // Singleton pattern

    public static Application getInstance() {
        if (instance == null) {
            instance = new Application();
        }
        return instance;
    }

    private static RemoteDataAdapter dao;

    private User currentUser = null;

    public void setCurrentUser(User user) { this.currentUser = user; }

    private Application() {
        dao = new RemoteDataAdapter();
        dao.connect();
    }
}

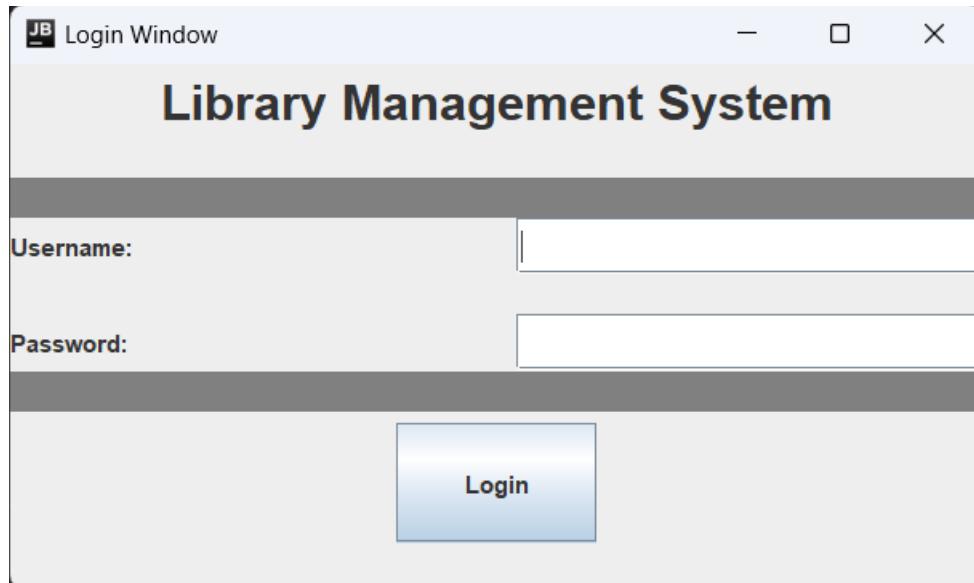
public static void main(String[] args) {
    try {

```

The application class is run to load the User Interface

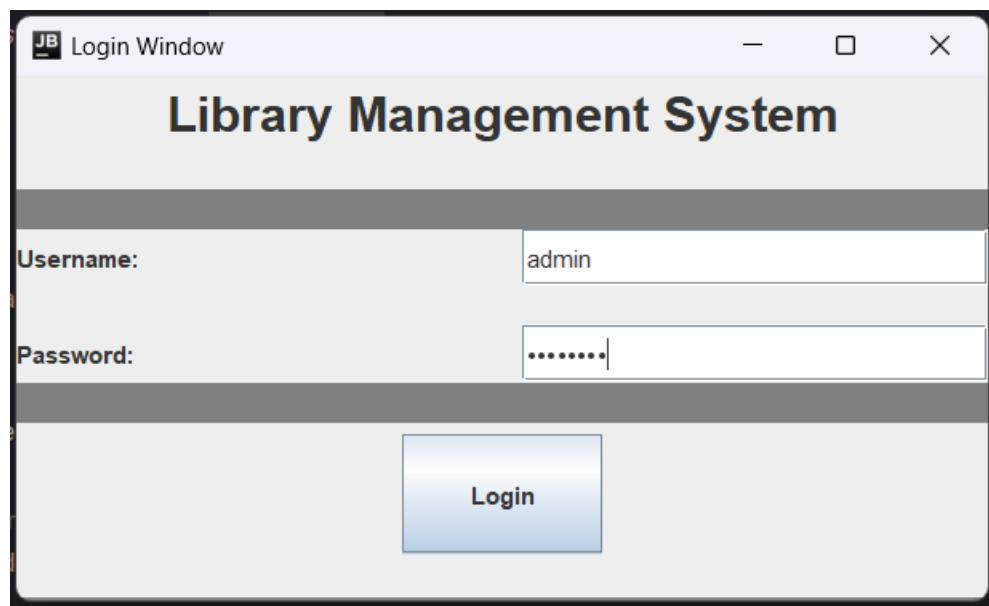
DESKTOP APPLICATION UI

3)



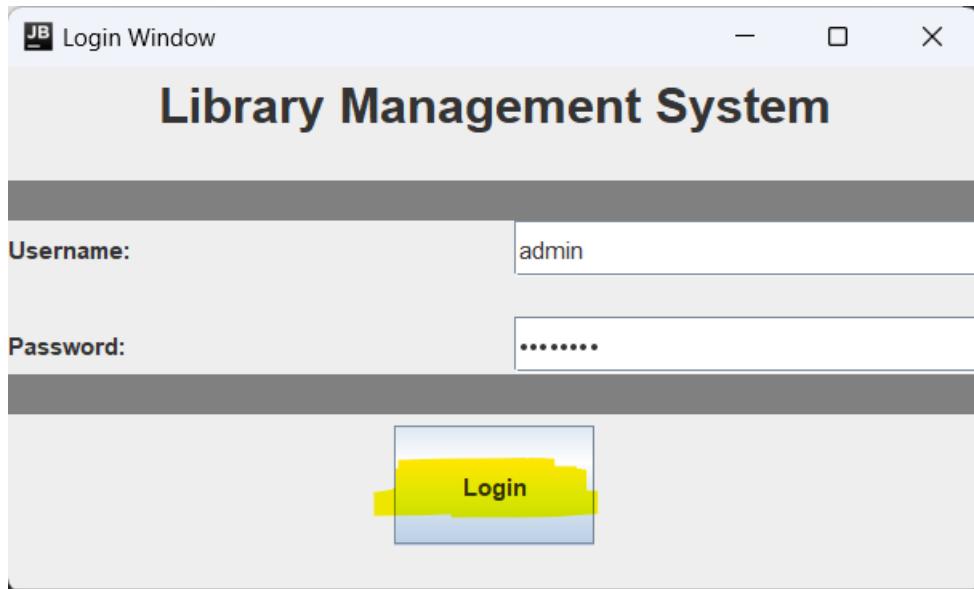
The login window pops up.

4)



The admin will enter login details in the window.

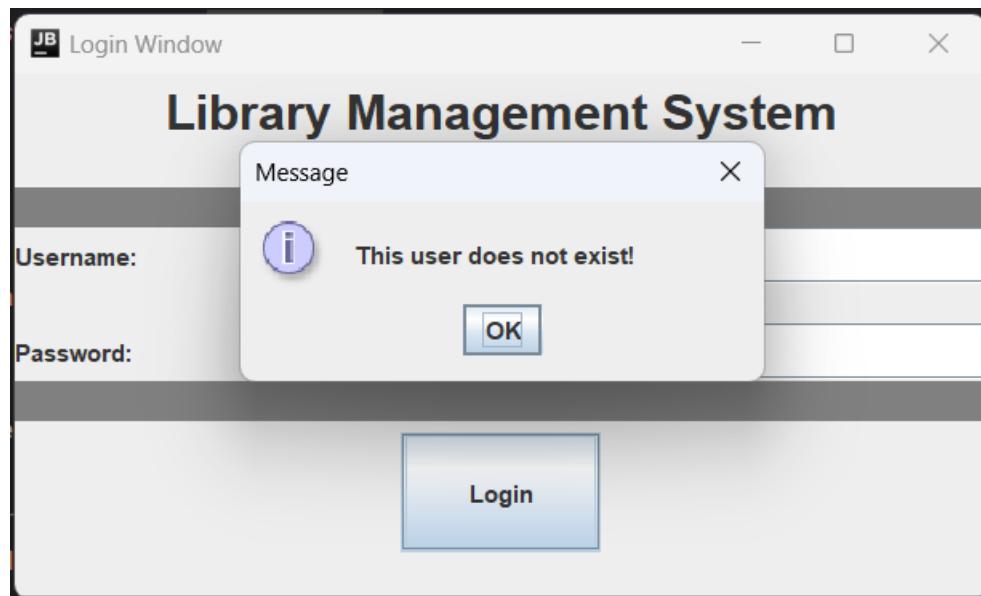
5)



The admin will click on the login button.

6)

Validation:



In case user detail does not match to the DB, "This user does not exist" window appears.

7)

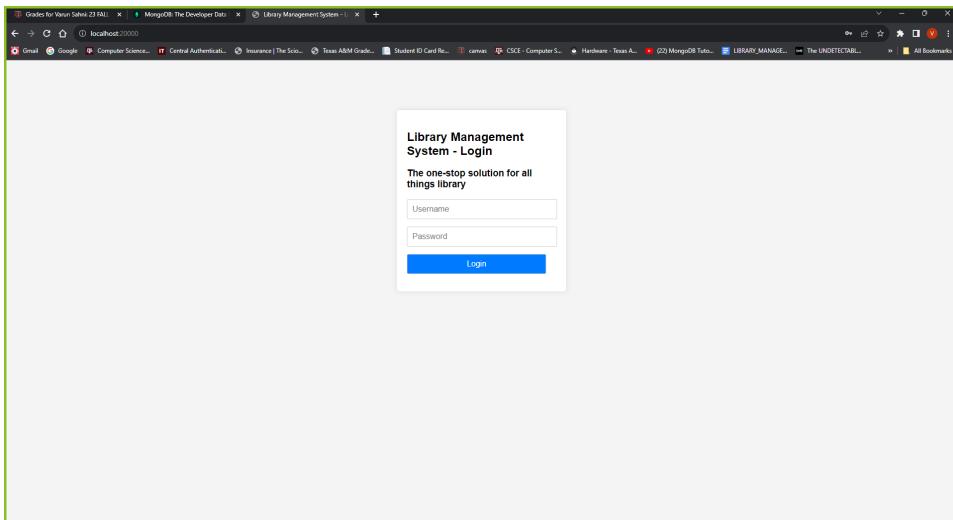
The screenshot shows a Redis database interface with a dark theme. At the top, it displays "HASH" and "User:admin". Below that, it shows the size as "136 B", the length as "4", and the TTL as "No limit". There are also buttons for "now", "G", "C", "R", "+", and "E".

Field	Value	edit	del
UserID	1	edit	del
UserName	admin	edit	del
Password	password	edit	del
DisplayName	Adam Smith	edit	del

Redis Database

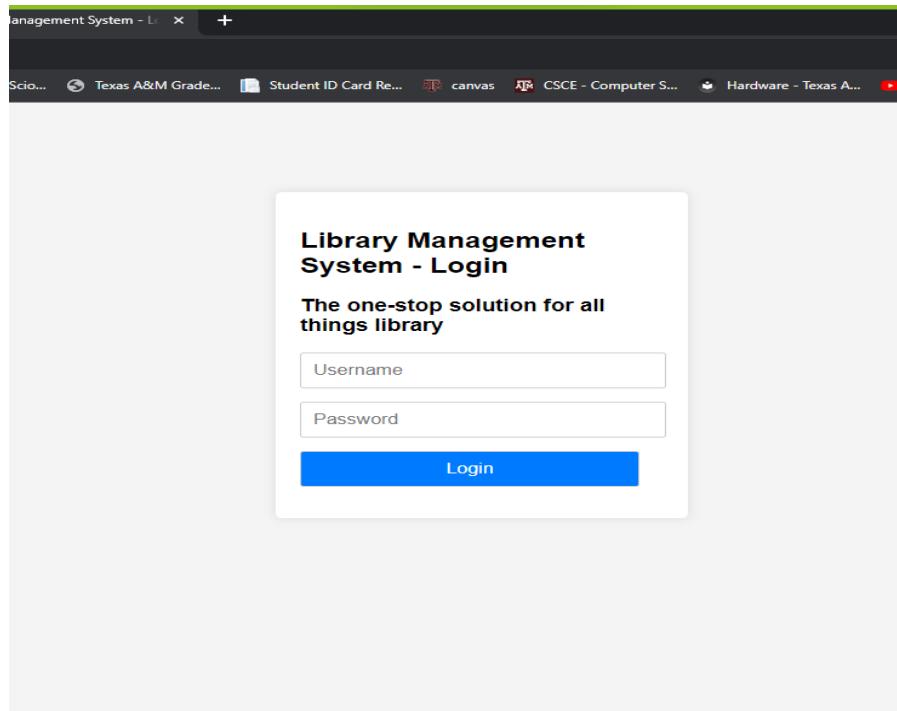
WEB APPLICATION UI

1)



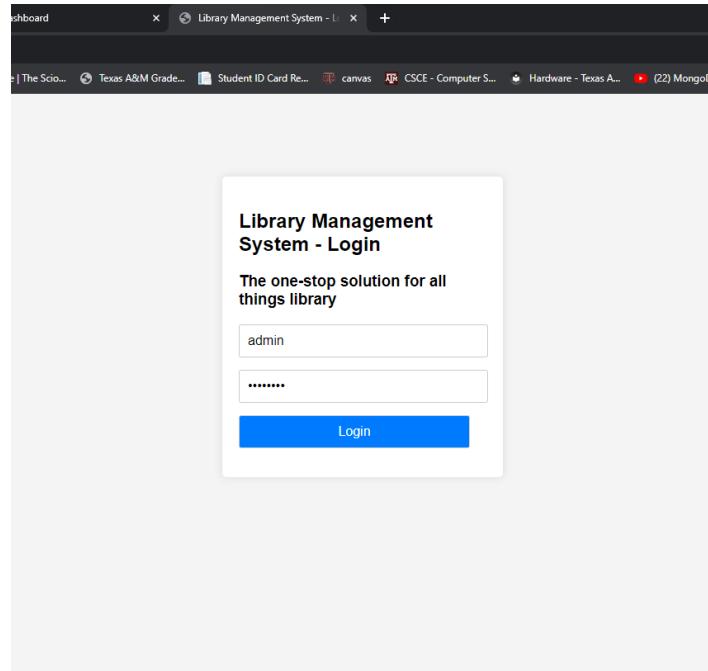
Full Screenshot of web browser

2)



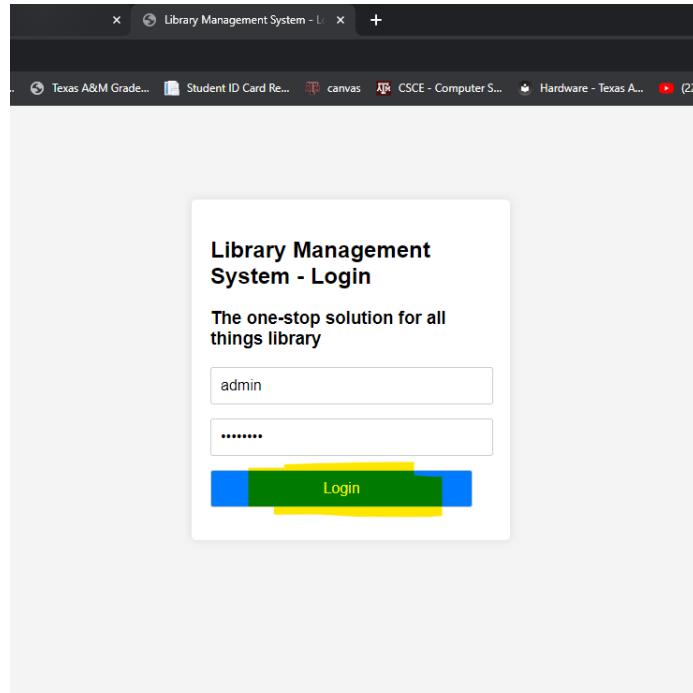
The login page.

3)



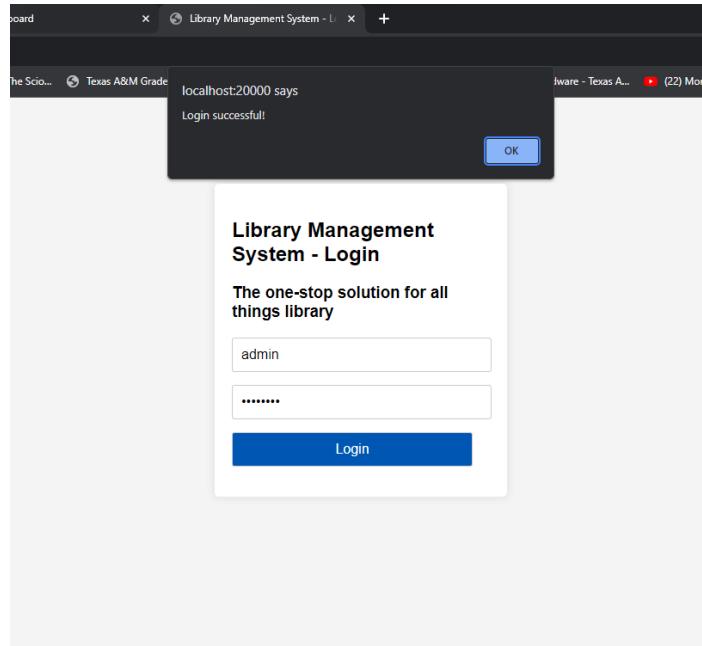
The admin will enter login details.

4)



The admin will click on the login button.

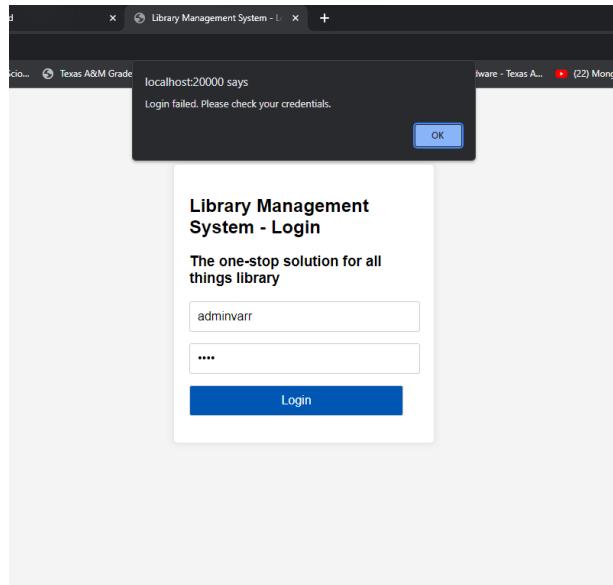
5)



Login Successful alert appears post user validation from UserServer

6)

Validation:



The entered login details are incorrect.

7)

The screenshot shows a Redis database interface with a dark theme. At the top, it displays "HASH User:admin". Below this, it shows the size as "136 B" and the length as "4", with a TTL of "No limit". There are buttons for "now", "G", "v", "r", "+", and "E".

The main area is a table with "Field" and "Value" columns. The fields listed are UserID, UserName, Password, and DisplayName, each with edit and delete icons.

Field	Value	
UserID	1	
UserName	admin	
Password	password	
DisplayName	Adam Smith	

Redis Database

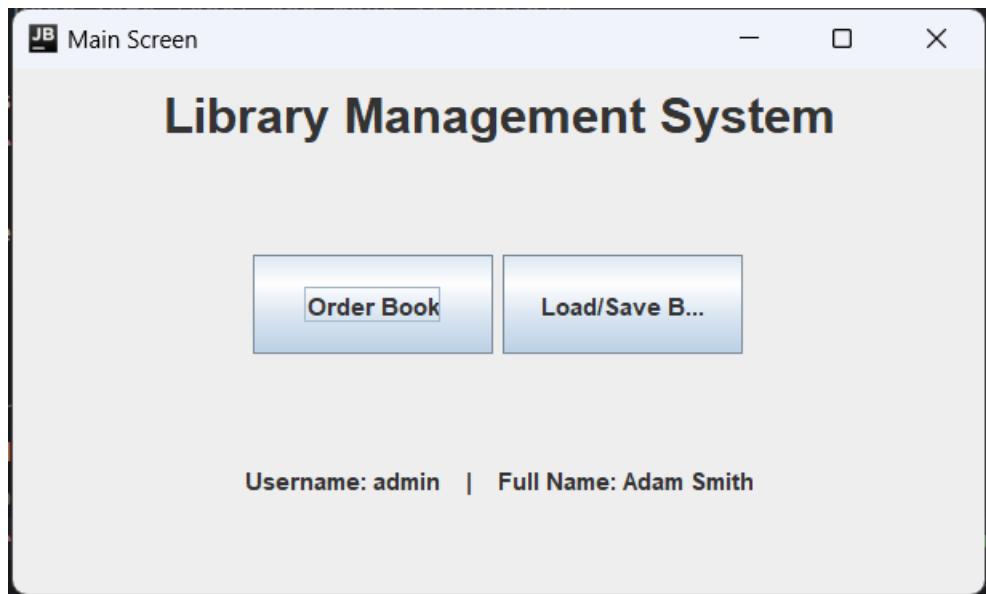
2nd Use Case:

Use Case	Main Screen	
Description	The administrator shall be able to view the main screen of the library management system along with the details of the administrator. The details shown would contain the Username and full name.	
Precondition	The administrator must be logged in to the application.	
Ordinary Sequence	Step	Action
	1	The main screen will have 2 buttons.
	2	The first button is for Order Book.
	3	The second button is for Load/Save Book.
Postcondition	The admin can perform multiple functions in the library management system.	

Actor Action	System Response
1 The admin can click on Load/Save Book and Order Book.	2 The system will open windows according to the user action.

DESKTOP APPLICATION UI

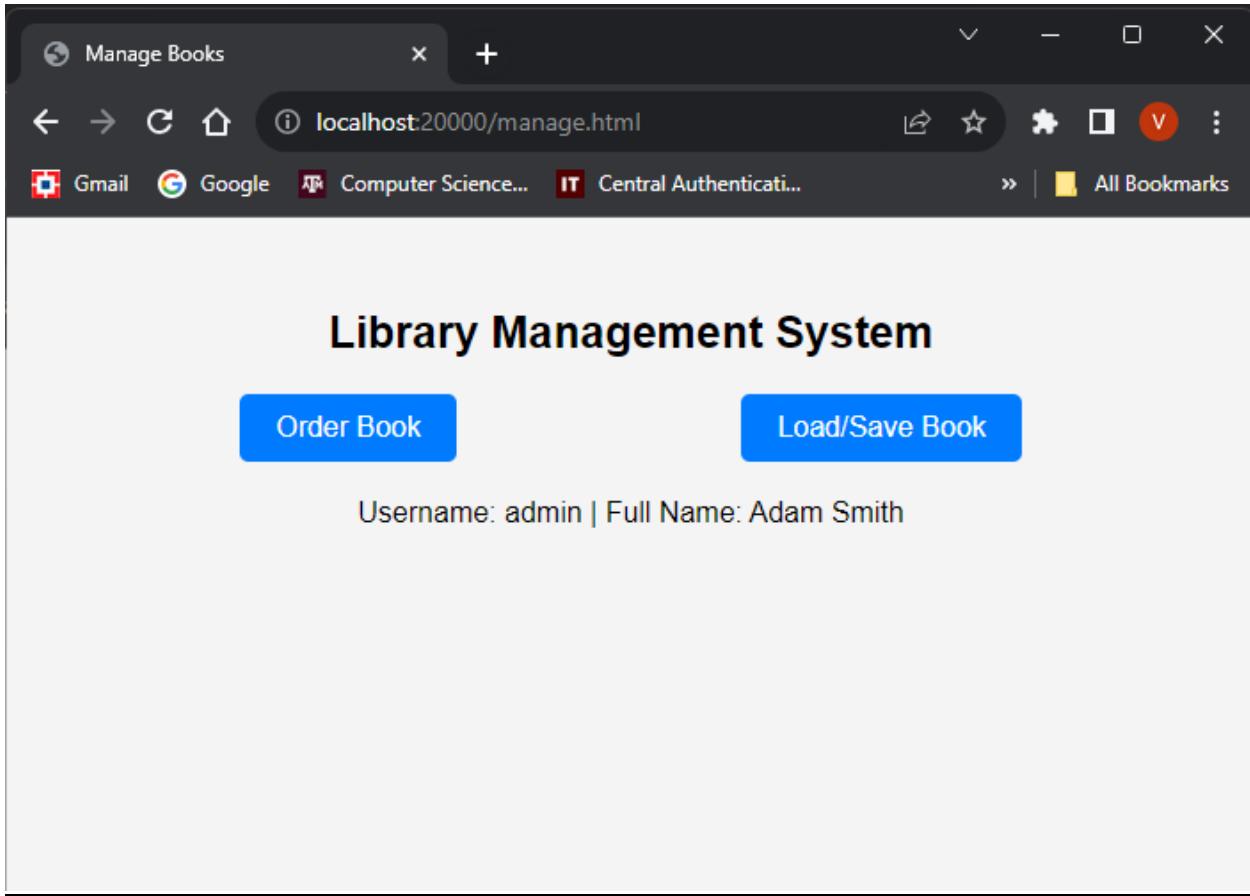
1)



The main screen has 2 buttons: Order Book and Load/Save Book.

WEB APPLICATION UI

1)



The manage books page has 2 buttons: Order Book and Load/Save Book.

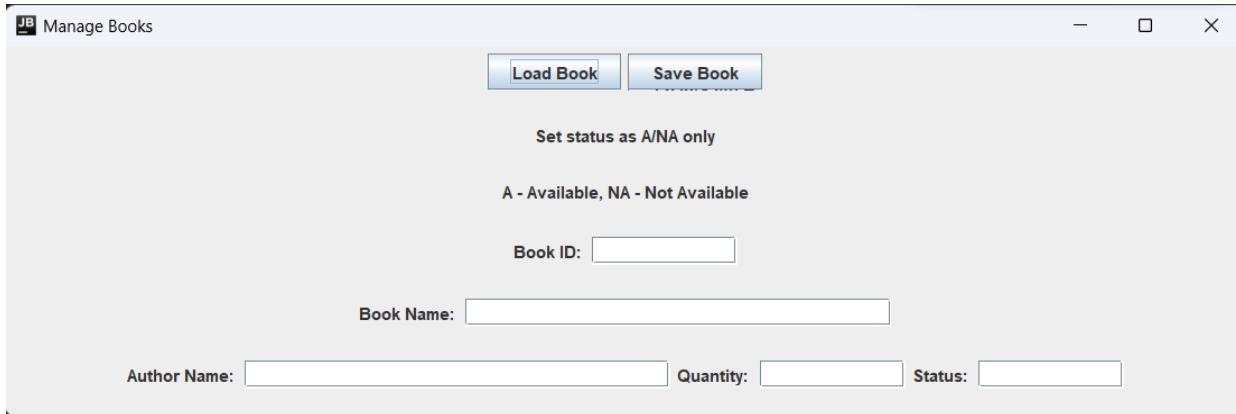
3rd Use Case:

Use Case	Save Book	
Description	The administrator shall be able to save a book to the database of library management system.	
Precondition	The main screen should be visible to the administrator.	
Ordinary Sequence	Step	Action
	1	The administrator will click on the Load/Save book button on main screen.
	2	The administrator would enter all the book details on the Manage Books window.
	3	The details include BookID, Book Name, Author Name, Quantity, and Status.
	4	After entering the details, the administrator can save the book by clicking the Save Book button.
Postcondition	The entered book details will be saved in the Redis Database.	
Comment	If the BookID corresponding to a particular book is present in the database, then that BookID would be overwritten.	
Exceptions	If the BookID is negative or zero, a system pop-up will appear alerting the admin to enter a valid Book ID.	

Actor Action	System Response
1 The admin clicks on the Save Book button from Main Screen window.	2 The “Manage Books” window appears.
3 Admin types in the book details like BookID, Book Name, Author Name, Quantity, Status.	
4 The admin clicks on “Save Book” button.	5 The system responds with “Book Saved Successfully” dialog box.
	6 The book details get saved in the database.

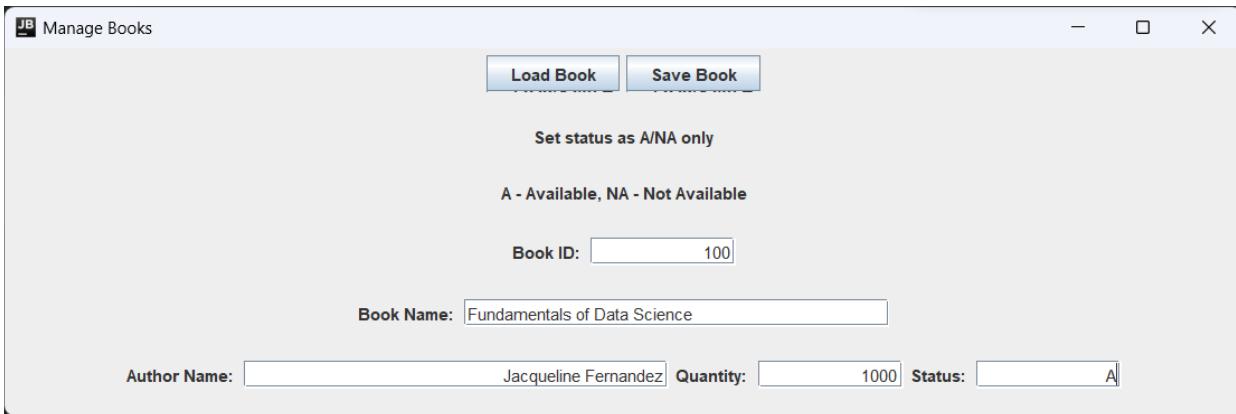
DESKTOP APPLICATION UI

1)



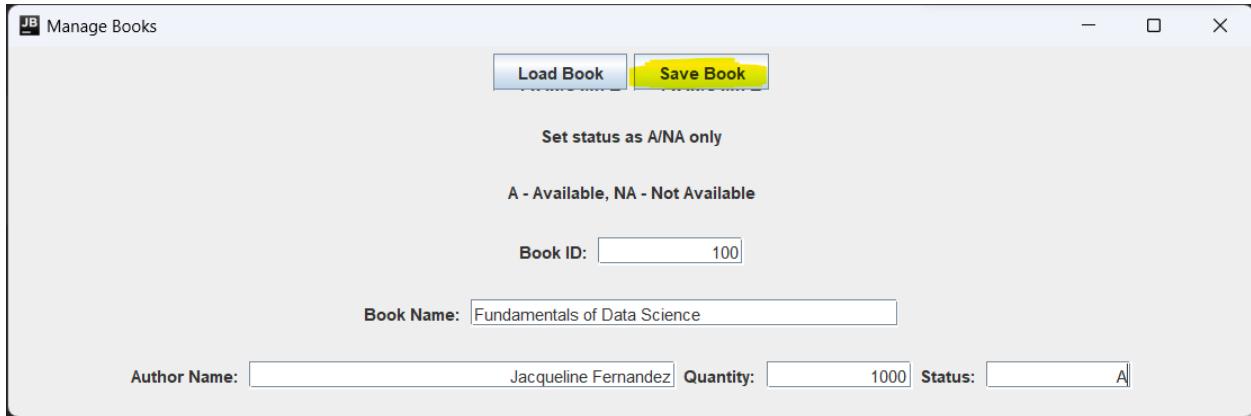
Manage Books window of Desktop Application.

2)



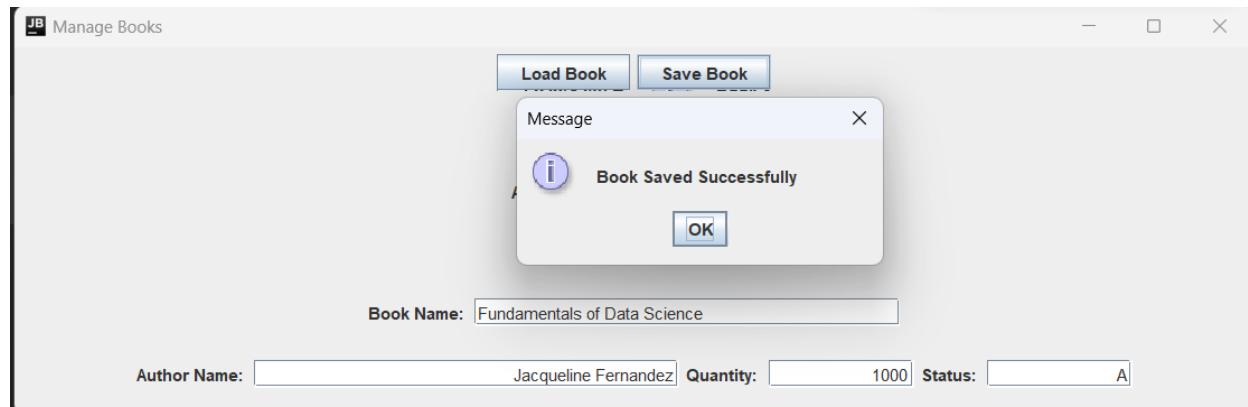
The details have been entered in the Manage Books window.

3)



The save Book button is clicked.

4)



The "Book Saved Successfully" pop up window appears.

5)

The screenshot shows the Redis Data Inspector interface. At the top, it displays "HASH" and the key "Book100". Below this, it shows the size as "168 B", the length as "4", and the TTL as "No limit". There are buttons for "now", "G", "C", "R", "+", and "E". The main area is a table with two columns: "Field" and "Value". The fields listed are BookName, AuthorName, Quantity, and Status. The values are Fundamentals of Data Science, Jacqueline Fernandez, 1000, and A respectively. Each row has edit icons (pencil and trash) next to the value.

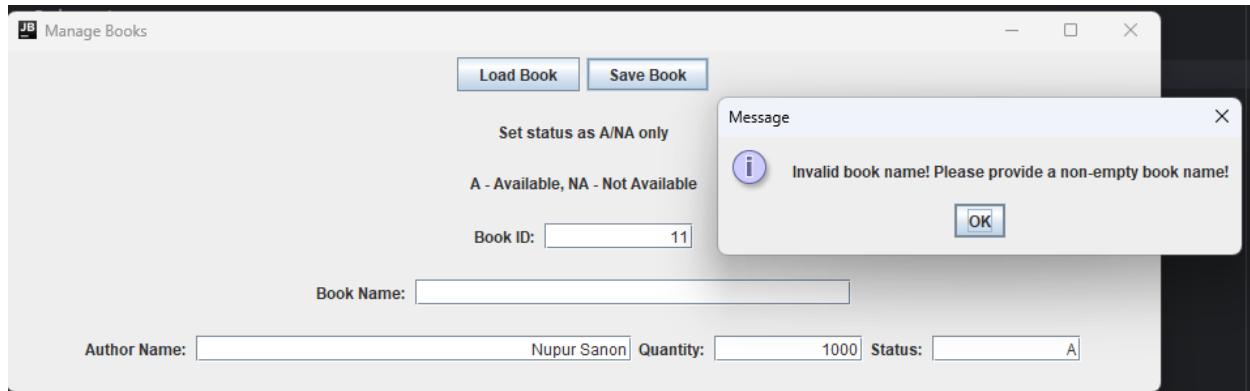
Field	Value
BookName	Fundamentals of Data Science
AuthorName	Jacqueline Fernandez
Quantity	1000
Status	A

The book details are saved in the Redis database table.

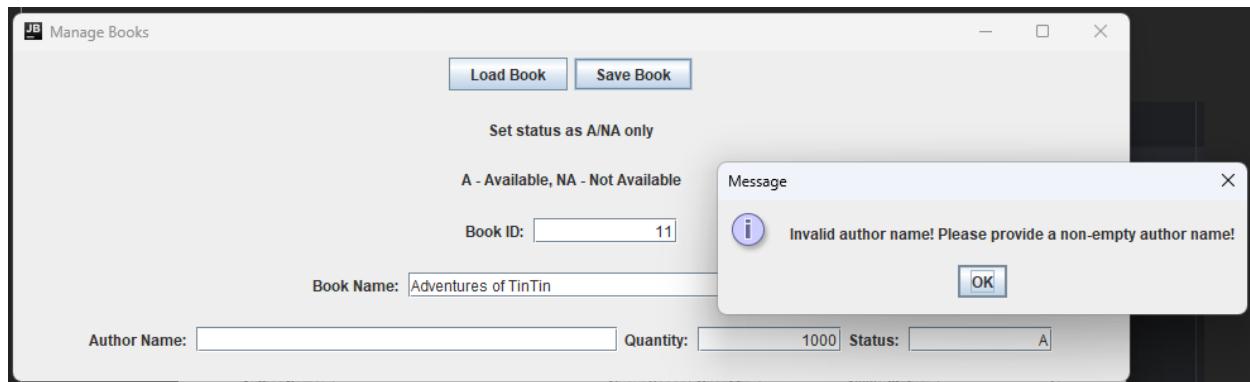
6) Validations:

The screenshot shows a Windows application window titled "Manage Books". It has buttons for "Load Book" and "Save Book". Below these is a note: "Set status as A/NA only" and "A - Available, NA - Not Available". There are input fields for "Book ID" (containing "0"), "Book Name", "Author Name", "Quantity", and "Status". A message dialog box is displayed with the title "Message" and the message "Book ID must be a positive integer!". There is an "OK" button at the bottom of the dialog.

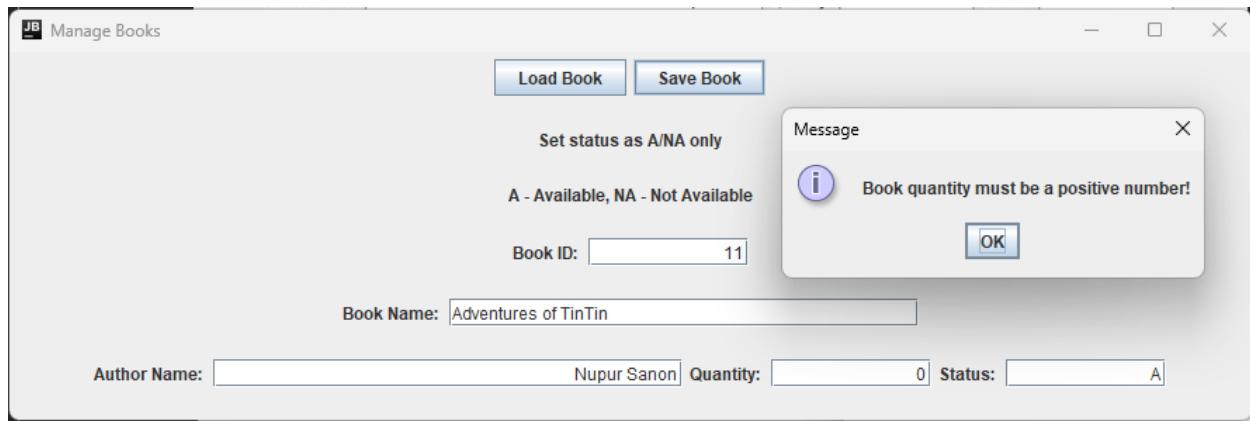
Book ID should be positive.



Book Name cannot be empty



Author Name cannot be empty



Quantity must be positive

WEB APPLICATION UI

1)

The screenshot shows a web browser window titled "Load/Save Books". The address bar displays "localhost:20000/loadsav.html". Below the address bar, there are several bookmark icons for "Gmail", "Google", "Computer Science...", "Central Authenti...", "Insurance | The Scio...", and "Texas A&M Grade...". The main content area is titled "Load/Save Books" and contains three blue buttons: "Load Book", "Save Book", and "Back to Main Page". Below these buttons are five input fields with labels: "Book ID:" (empty), "Book Name:" (empty), "Author Name:" (empty), "Quantity:" (empty), and "Status:" (empty).

Load/Save Books page of Desktop Application.

2)

Load/Save Books

Load Book Save Book Back to Main Page

Book ID:
50

Book Name:
Unfulfilled Desires

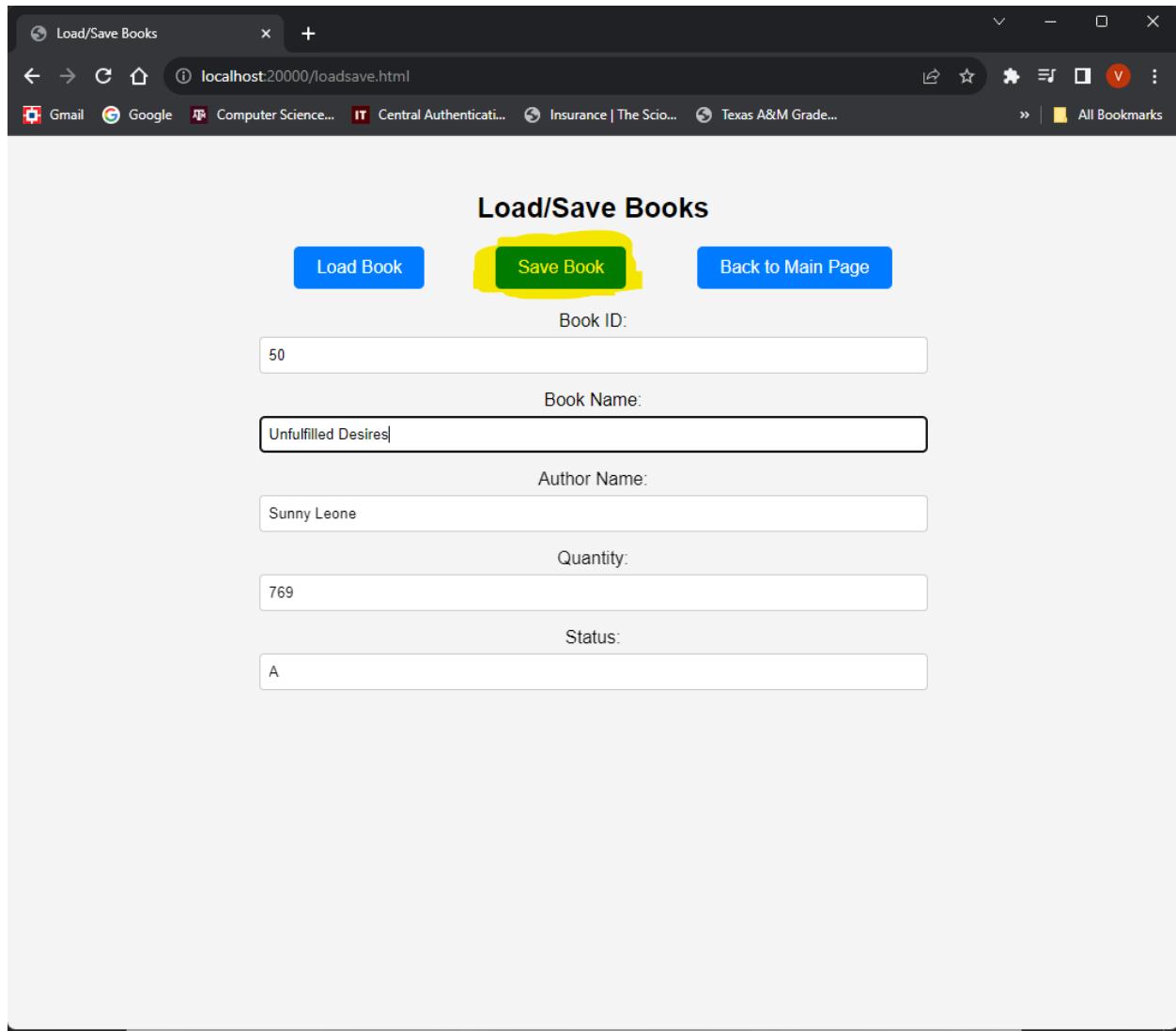
Author Name:
Sunny Leone

Quantity:
769

Status:
A

The details have been entered in the Load/Save Books page.

3)

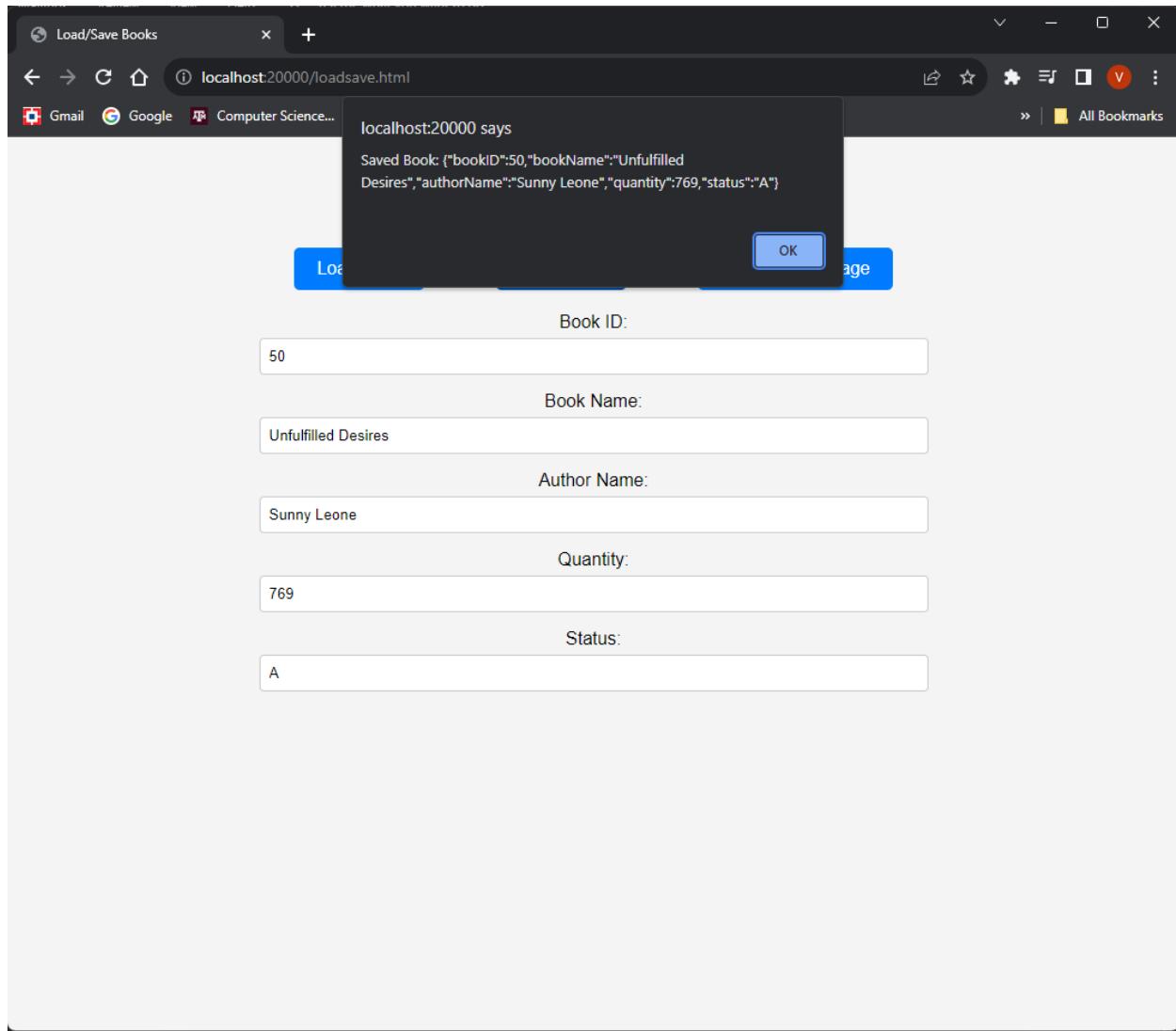


The screenshot shows a web browser window titled "Load/Save Books" with the URL "localhost:20000/loadsav.html". The page contains a form with the following fields:

- Load Book** (blue button)
- Save Book** (green button, highlighted with a yellow box)
- Back to Main Page** (blue button)
- Book ID:**
- Book Name:**
- Author Name:**
- Quantity:**
- Status:**

The save Book button is clicked.

4)



The “Book Saved Successfully” alert appears.

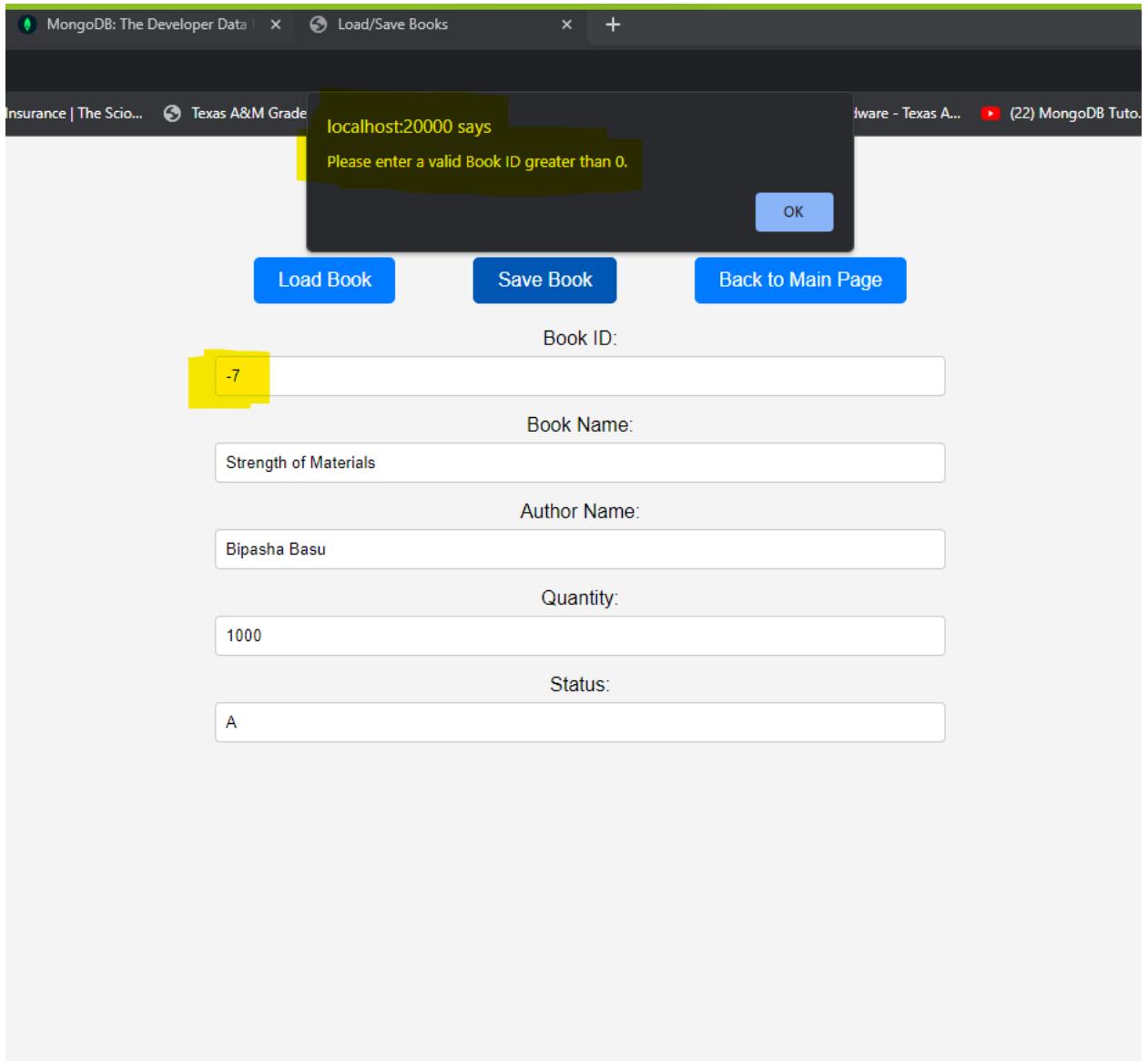
5)

The screenshot shows the Redis UI interface. At the top, it displays "HASH Book50". Below that, status information: "Key Size: 144 B", "Length: 4", "TTL: No limit", "Last refresh: now", and "JSON" dropdown. There are also "Add Fields" and "Delete" buttons. The main area is a table with four rows:

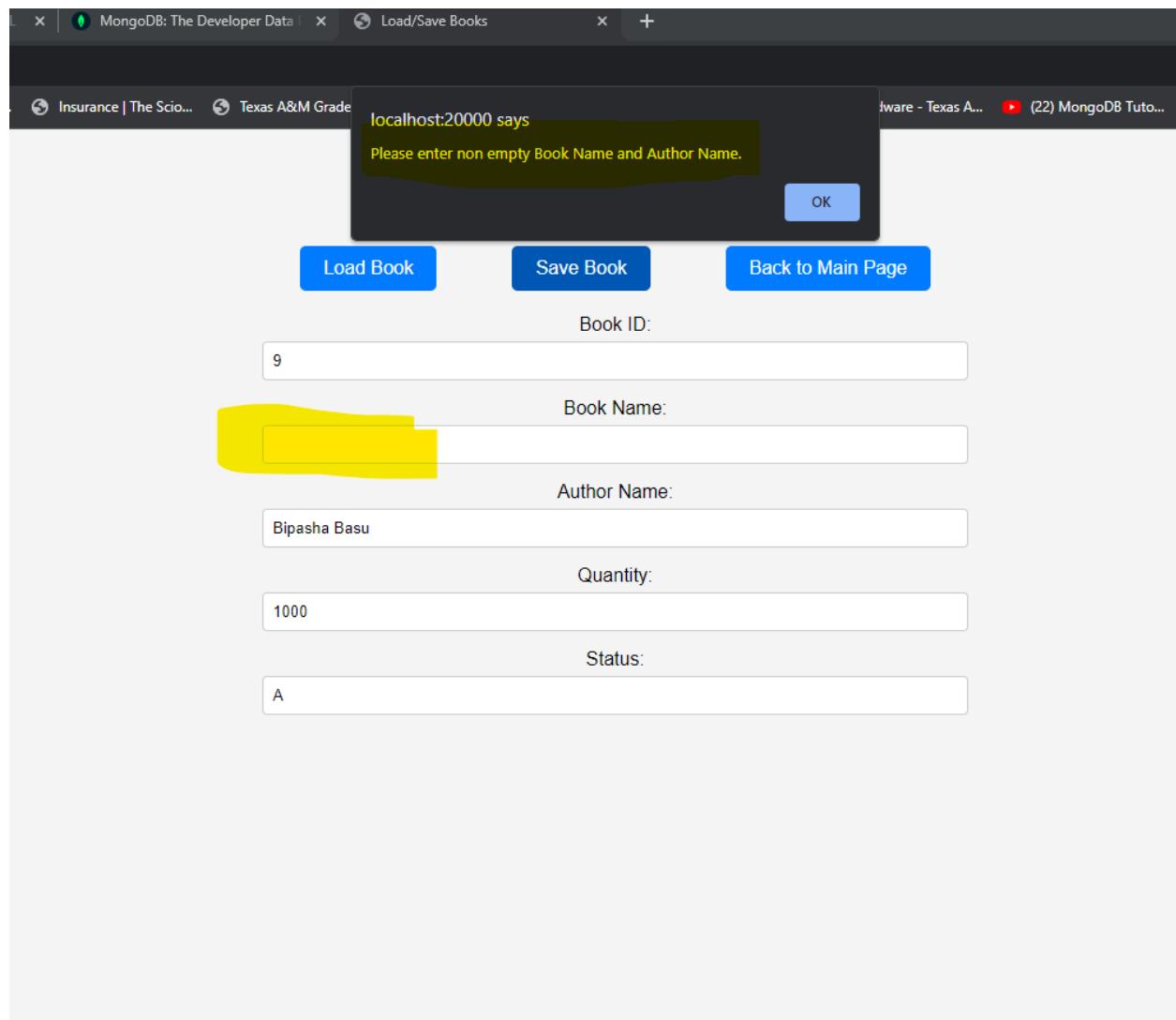
Field	Value	Edit	Delete
BookName	Unfulfilled Desires	edit	trash
AuthorName	Sunny Leone	edit	trash
Quantity	769	edit	trash
Status	A	edit	trash

The book details are saved in the Redis database table.

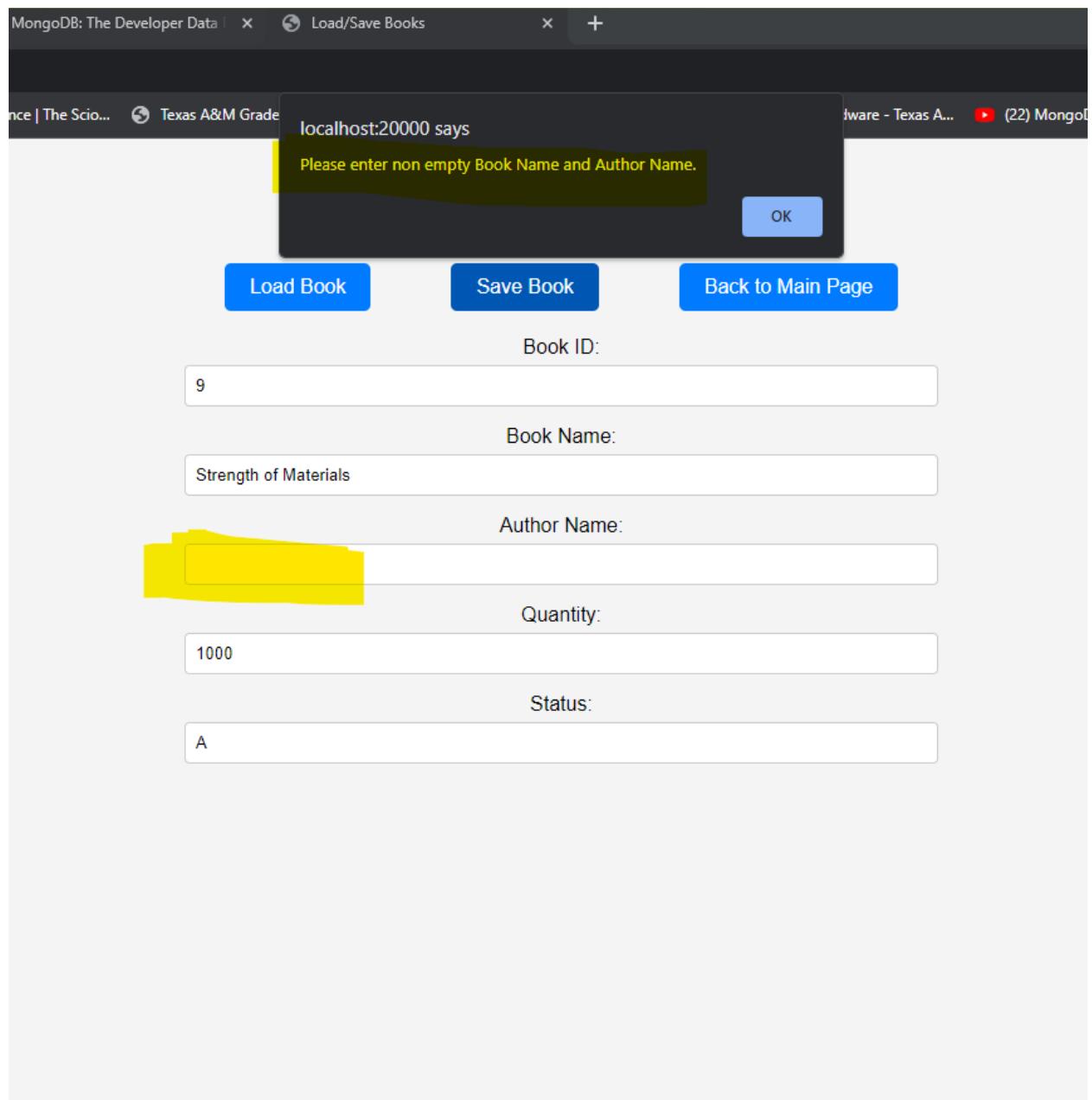
Validations:



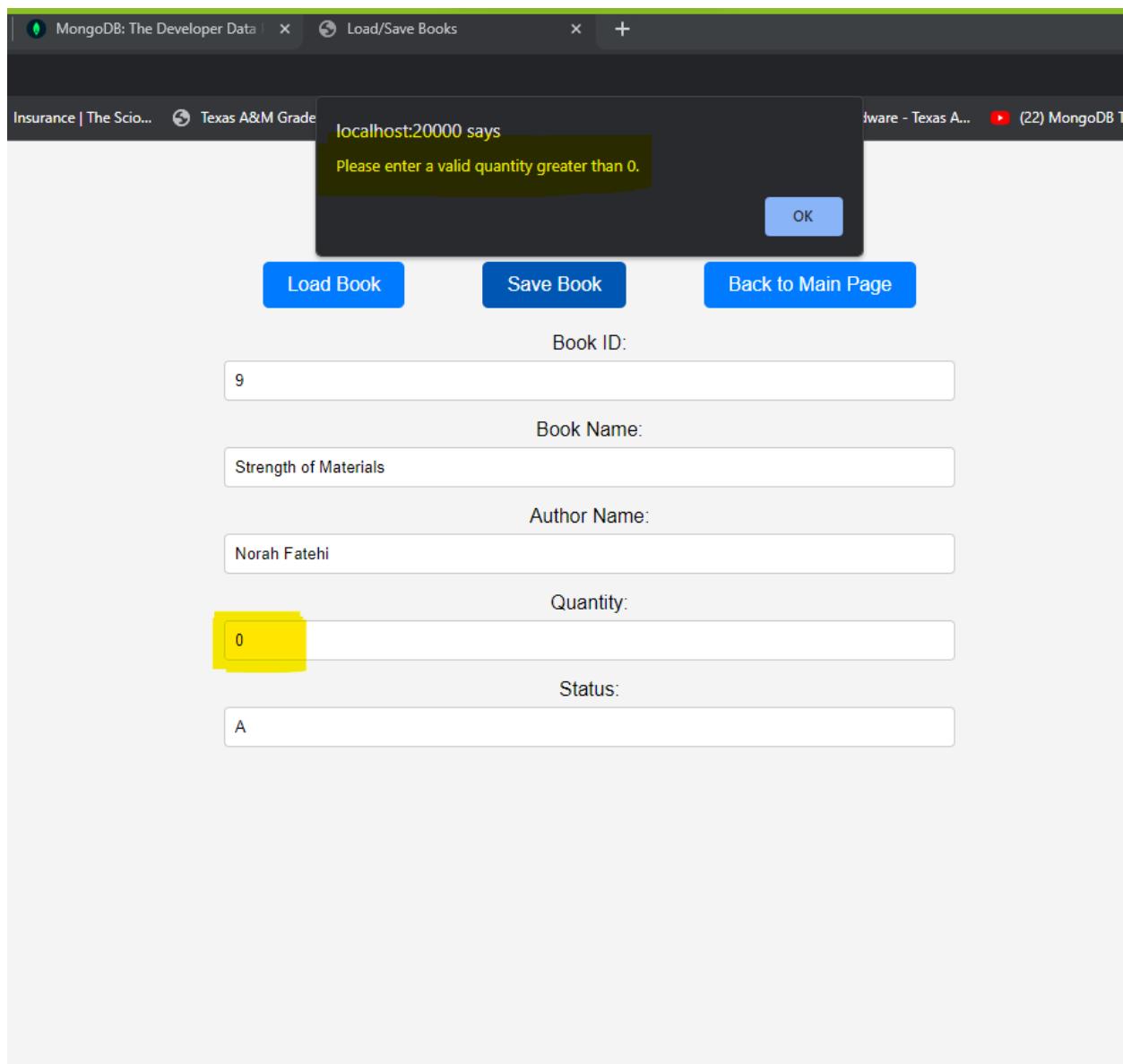
Book ID should be positive alert pops up on the page.



Book Name cannot be empty alert pops up on the page



Author Name cannot be empty alert pops up on the page



Quantity must be positive alert pops up on the page

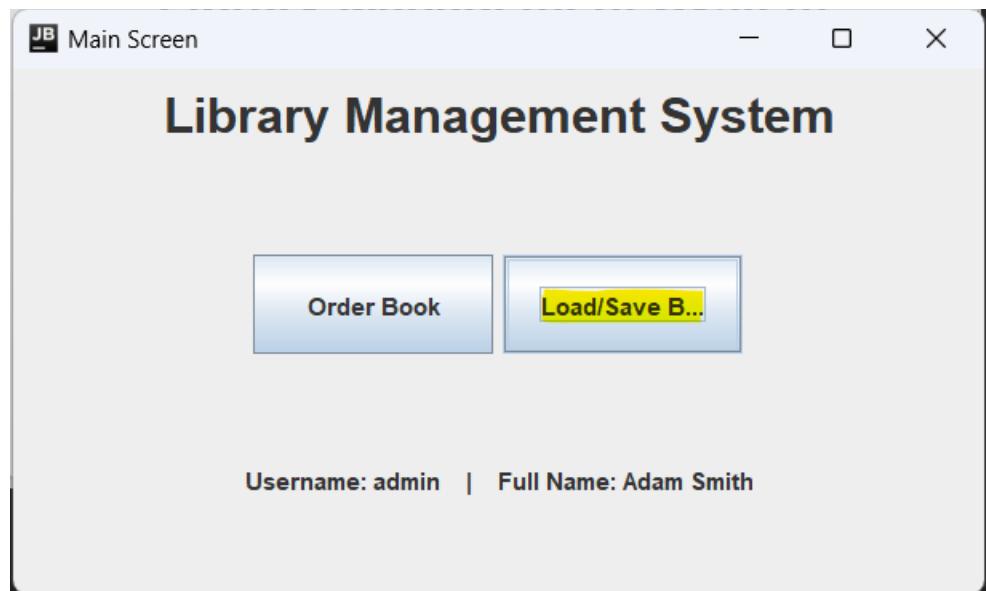
4th Use Case:

Use Case	Load Book	
Description	The administrator shall be able to load a book to the database of the library management system	
Precondition	The main screen should be visible to the administrator. The book should be present in the Redis Database (with the same Book Id).	
Ordinary Sequence	Step	Action
	1	The administrator will click on the Load/Save book button on the main screen.
	2	The administrator would enter the Book ID against the “Book ID” label in the Manage Books window.
	3	After entering the Book ID, the administrator will click on the “Load Book” button.
	4	Library Management System application will automatically search for the book in the database, and auto-populate the Book Name, Author Name, Quantity, and Status if the Book exists in the Database. Else, a window will pop up that “No such book exists with this book id”.
Postcondition	The entered book details will be displayed to the user.	
Exceptions	Step	Action
	1	If the Book ID corresponding to the particular book is not present in the database, then book details will not be auto-populated.

Actor Action	System Response
1 The admin clicks on the Load Book button from Main Screen window.	2 The “Manage Books” window appears.
3 Admin types in the Book ID.	
4 The admin clicks on “Load Book” button.	5 The system responds by displaying the Book Name, Author Name, Quantity, and Status of the respective Book. In case the book is not present in the Redis Database, the system displays a pop-up window that “No book with this book id exists in Database”.
	6 The book details get saved in the database.

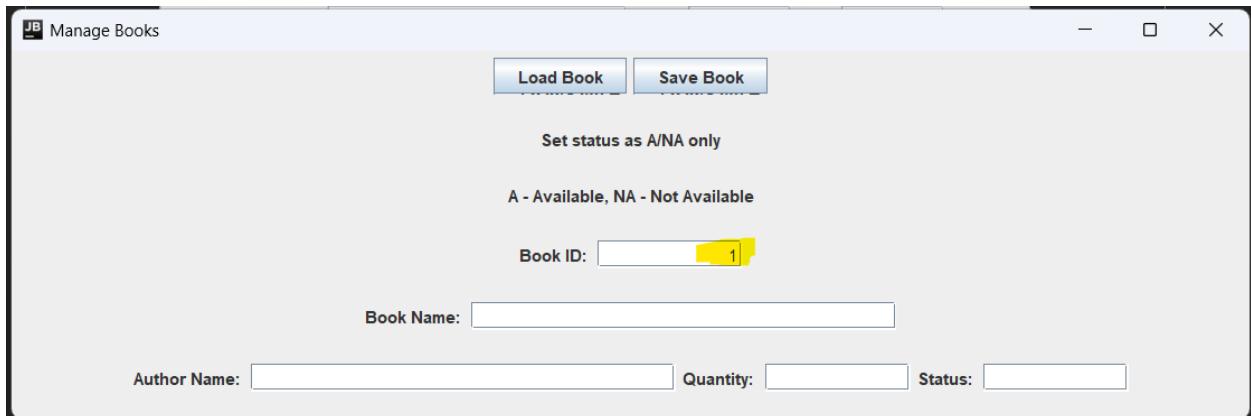
DESKTOP APPLICATION UI

1)



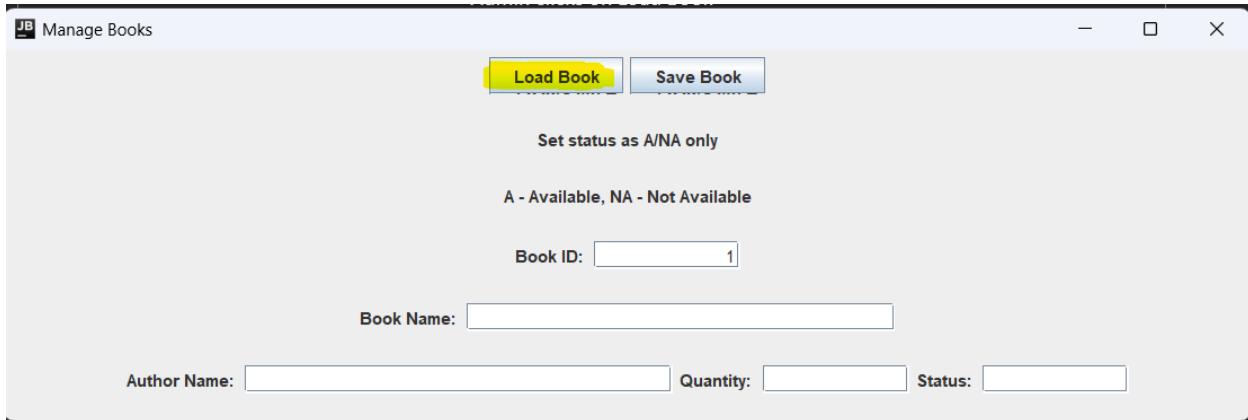
Main Screen window -> Load/Save Book button

2)



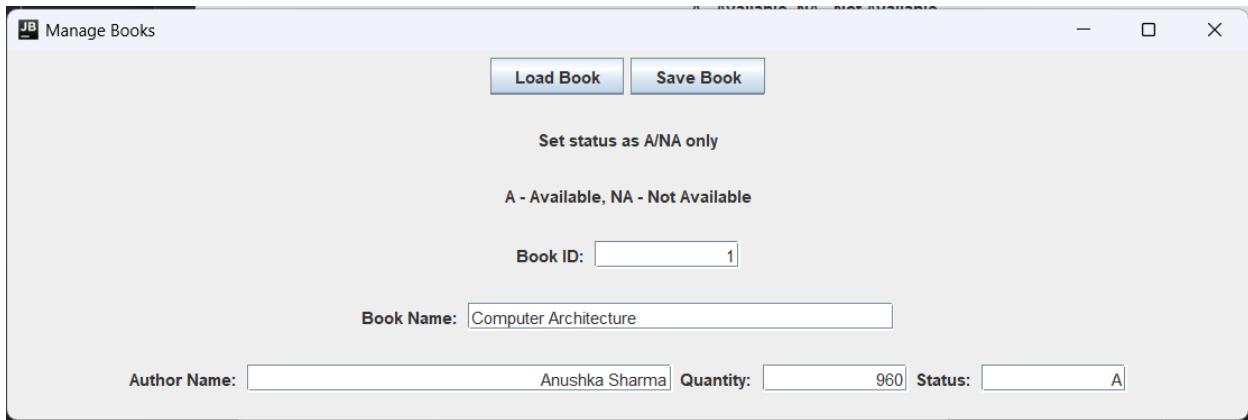
Manage Books screen -> Admin enters Book ID 1

3)



Admin clicks on Load Book button

4)



The system auto-fills the Book Name, Author Name, Quantity, Status from the Database in case the Book Id is available in the database.

HASH Book1

Key Size: 144 B Length: 4 TTL: No limit Last refresh: < 1 min JSON + Add Fields trash

Field	Value	edit	trash
BookName	Computer Architecture	edit	trash
AuthorName	Anushka Sharma	edit	trash
Quantity	960	edit	trash
Status	A	edit	trash

Redis Database

6)

JB Manage Books

Load Book Save Book

Set status as A/NA only

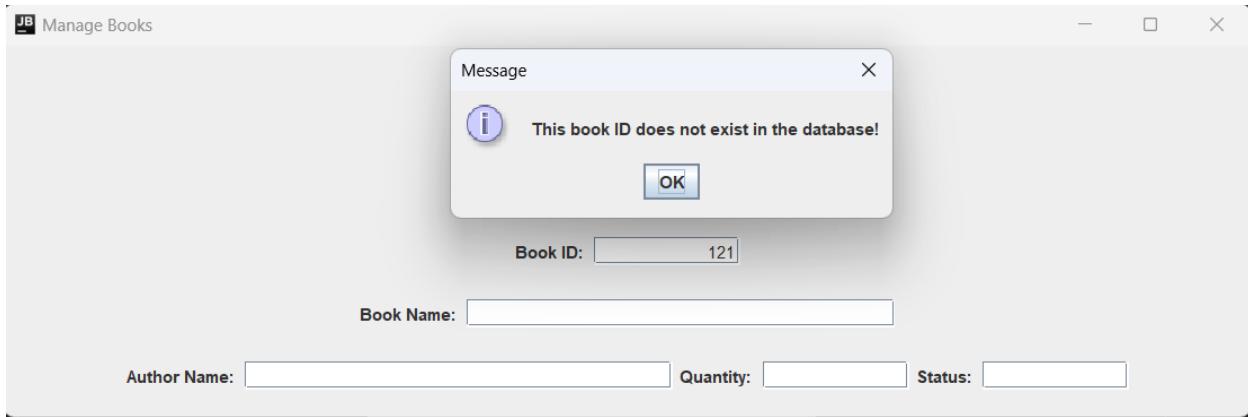
A - Available, NA - Not Available

Book ID:

Book Name:

Author Name: Quantity: Status:

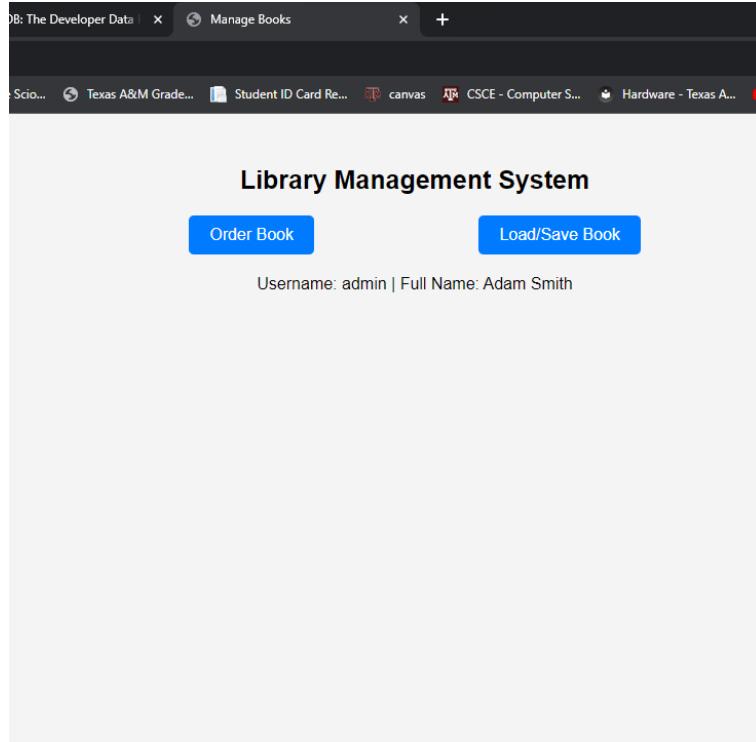
In case user enters invalid Book Id and clicks on Load Book Button.



A pop-up window appears stating that this book Id does not exist in database.

WEB APPLICATION UI

1)



Main Screen window -> Load/Save Book button

2)

The screenshot shows a web browser window titled "Load/Save Books". The address bar contains the URL "The Developer Data" and the page title "Load/Save Books". Below the address bar, there is a horizontal navigation bar with several links: "Texas A&M Grade...", "Student ID Card Re...", "canvas", "CSCE - Computer S...", "Hardware - Texas A...", and "(22) Mon".

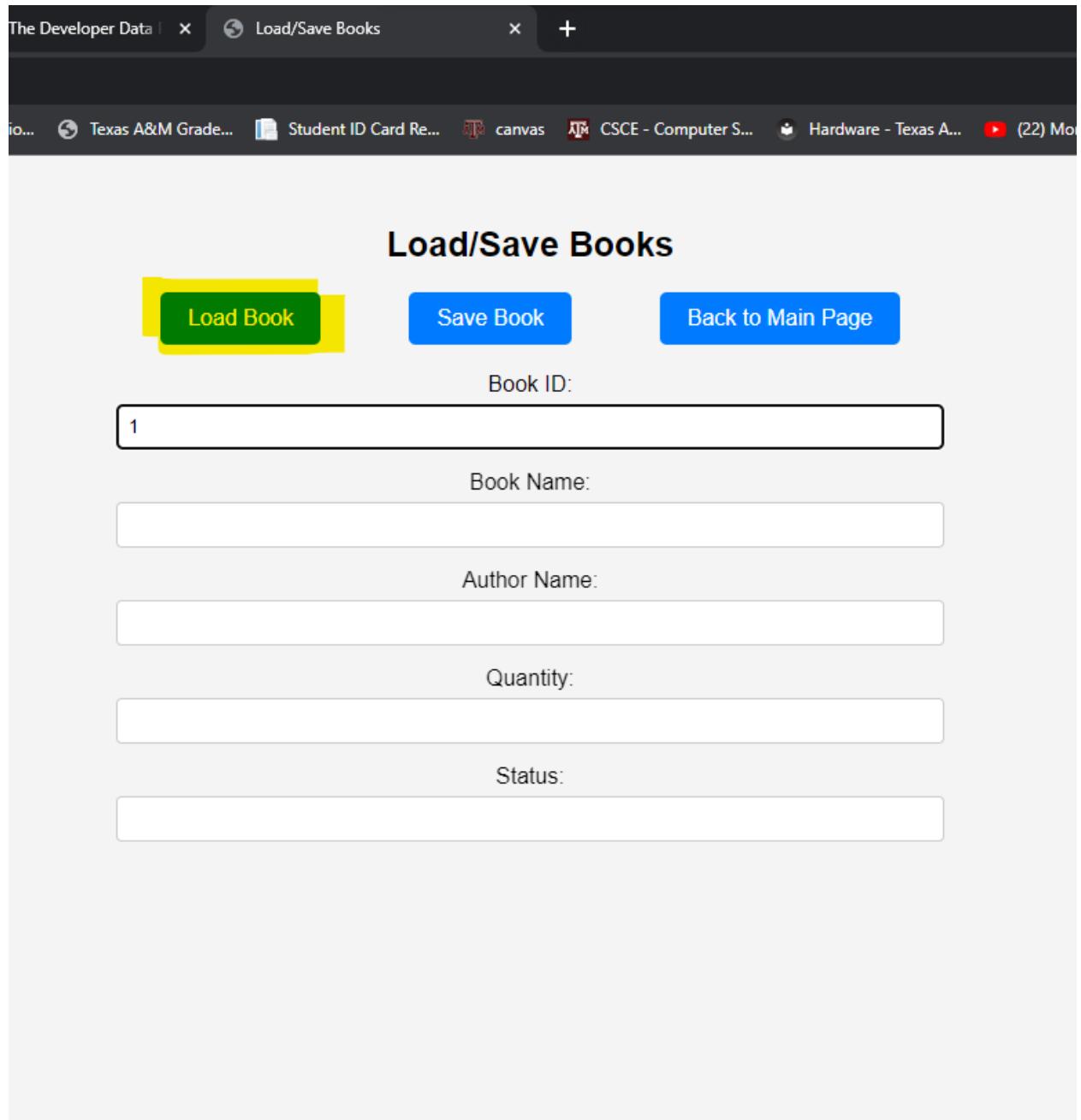
The main content area is titled "Load/Save Books". It features three blue buttons at the top: "Load Book", "Save Book", and "Back to Main Page".

Below the buttons, there are five input fields with labels:

- Book ID: (The value "1" is highlighted with a yellow box.)
- Book Name:
- Author Name:
- Quantity:
- Status:

Manage Books screen -> Admin enters Book ID 1

3)



The Developer Data | [Load/Save Books](#) | +

Texas A&M Grade... Student ID Card Re... canvas CSCE - Computer S... Hardware - Texas A... (22) Mo...

Load/Save Books

[Load Book](#) [Save Book](#) [Back to Main Page](#)

Book ID:

Book Name:

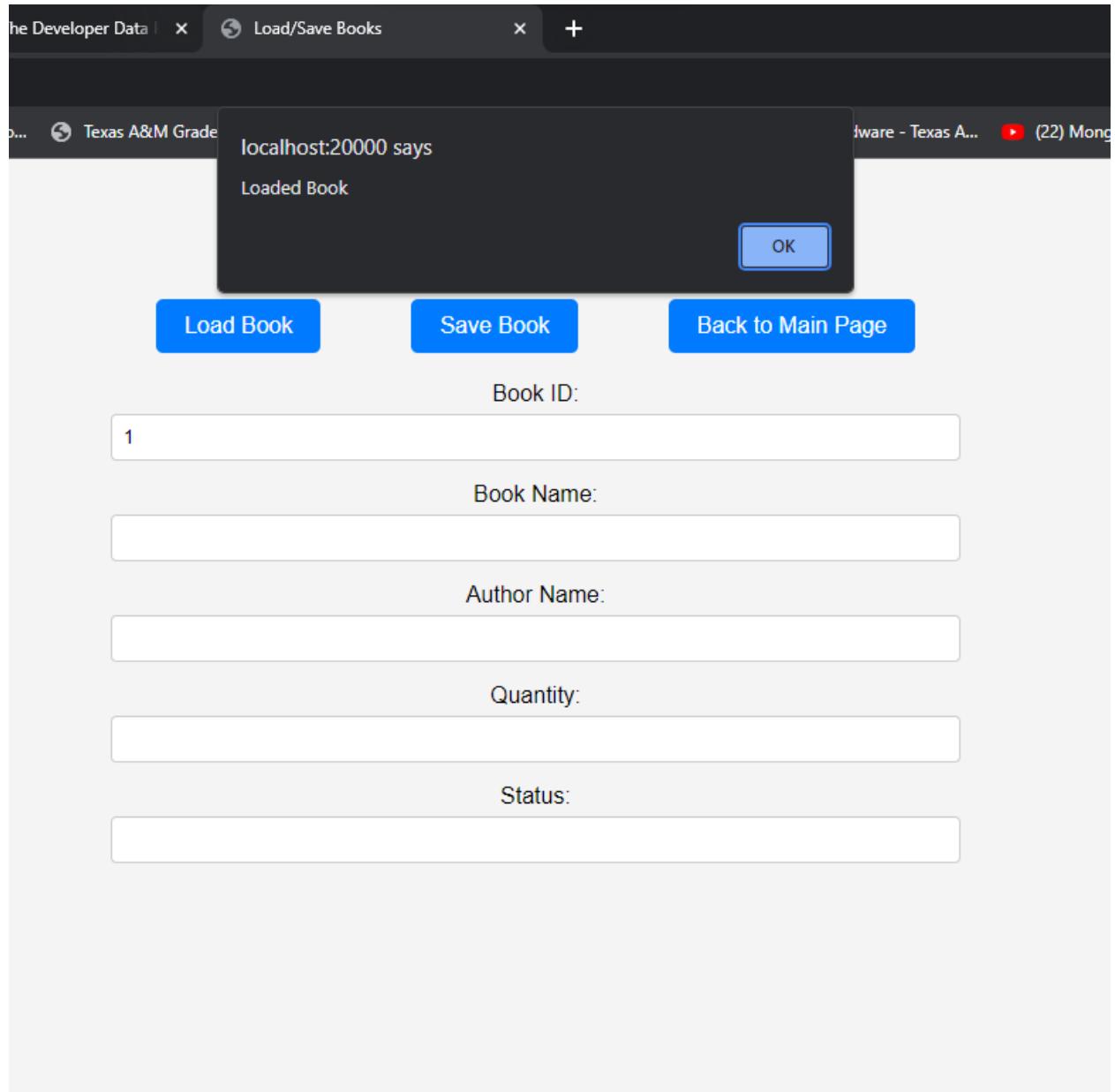
Author Name:

Quantity:

Status:

Admin clicks on Load Book button

4)



Loaded Book alert pops up

5)

The screenshot shows a web application titled "Load/Save Books". At the top, there are three blue buttons: "Load Book", "Save Book", and "Back to Main Page". Below these buttons are five input fields with labels and placeholder text:

- Book ID: 1
- Book Name: Computer Architecture
- Author Name: Anushka Sharma
- Quantity: 960
- Status: A

The system auto-fills the Book Name, Author Name, Quantity, Status from the Database in case the Book Id is available in the database.

HASH Book1			
Key Size: 144 B Length: 4 TTL: No limit		Last refresh: < 1 min	JSON
Field	Value		
BookName	Computer Architecture		
AuthorName	Anushka Sharma		
Quantity	960		
Status	A		

Redis Database

6)

Validations:

Load/Save Books

Book ID:

Book Name:

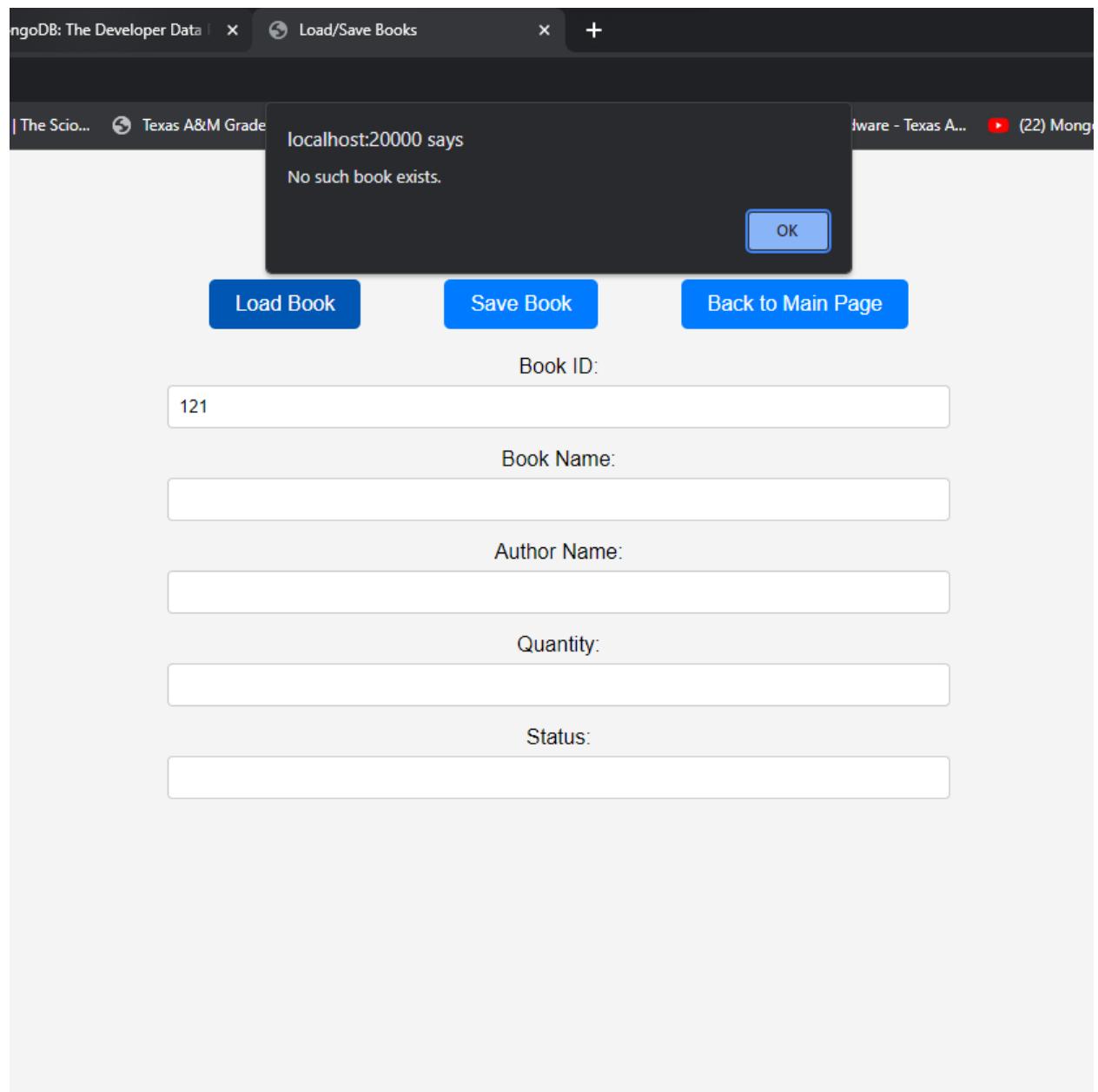
Author Name:

Quantity:

Status:

[Load Book](#) [Save Book](#) [Back to Main Page](#)

In case user enters invalid Book Id and clicks on Load Book Button.



"No such Book exists" alert pops up

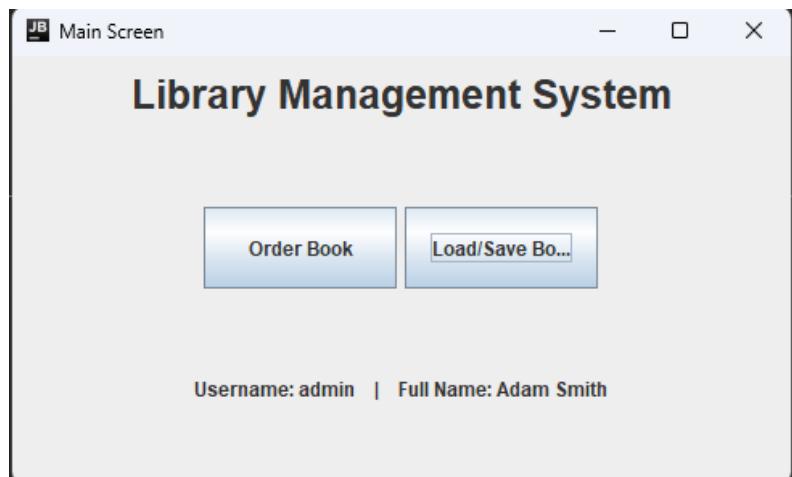
5th Use Case:

Use Case	Book Order View	
Description	The administrator shall be able to access the Book Order View screen in the library management system.	
Precondition	The administrator should have logged in to the application	
Ordinary Sequence	Step	Action
	1	The administrator will click on the Order Book button on the main screen
	2	Book Order View page appears on the screen.
Postcondition	<p>This screen shall also display two buttons - the “Add a new book” button, used to select books to order and the “Save book order” button, used to create an order</p> <p>This screen shall display the Book ID, Book Name, Book Author, Order Date, and Return Date after entering the book details to current order</p>	
Comment	This screen shall be used to display the books that are being added to the current order.	

Actor Action	System Response
1 The administrator will access the Login screen and enter his username and password	2 System will validate the user's details and open the Main Screen window.
3 The administrator will click on the Order Book button on the main screen	4 System will display the Book Order view page.

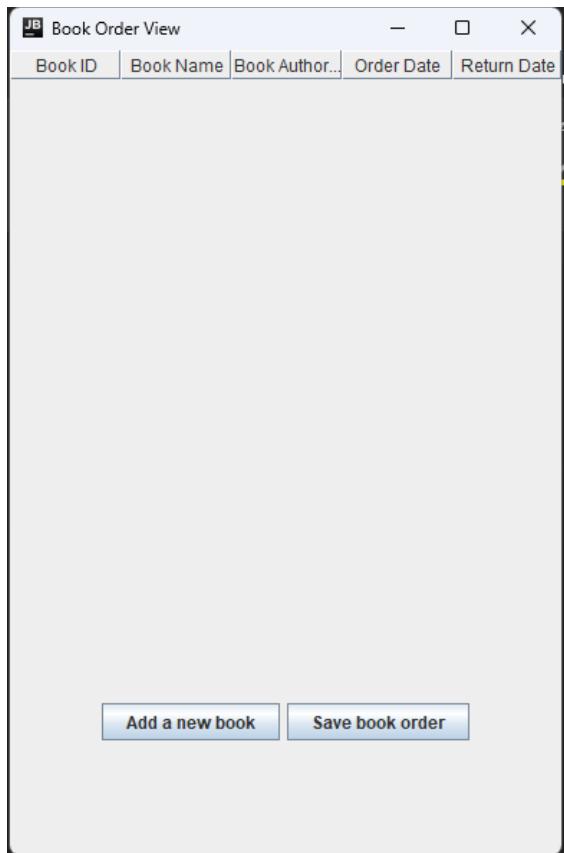
DESKTOP APPLICATION UI

1)



Main Screen -> Admin clicks on Order Book button

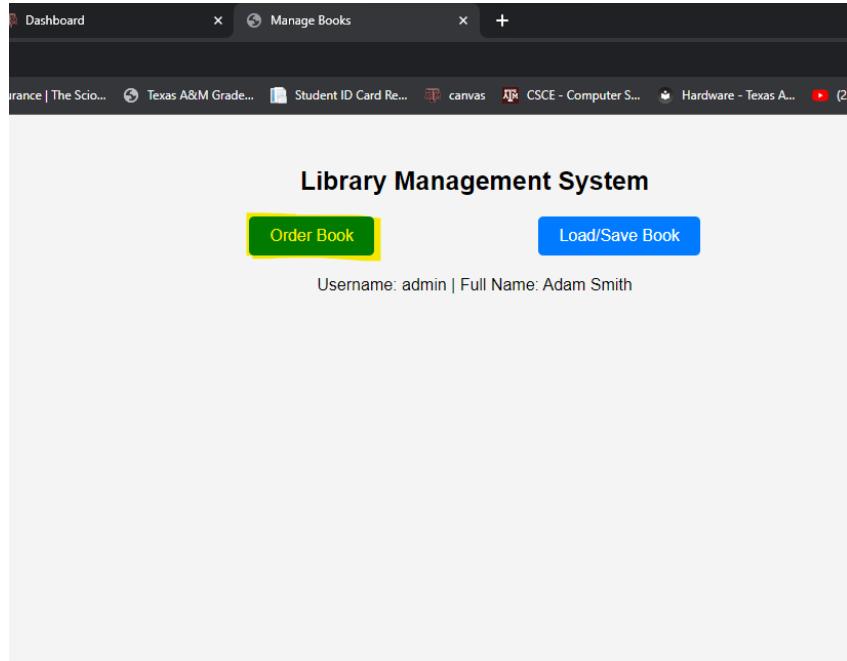
2)



Book Order View screen appears on the screen

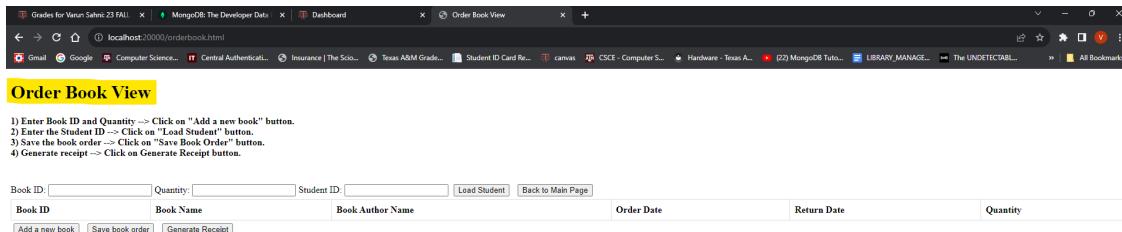
WEB APPLICATION UI

1)



Manage Books Page -> Admin clicks on Order Book button

2)



The book Order View page opens up and appears on the screen

6th Use Case:

Use Case	Add a new book to the Cart (Book Order View table)	
Description	The administrator shall be able to add books to the current order	
Precondition	The Book Order view screen should be visible to the administrator.	
Ordinary Sequence	Step	Action
	1	The administrator will click on the “Add a new book” button on the Book Order View screen.
	2	The book ID input dialog box appears
	3	The administrator will enter the book ID.
	4	The system will check if the respective book exists in the database and display the quantity input dialog box if the book exists. Otherwise, it shall display a pop-up window saying that this book does not exist in the database.
	5	The administrator will enter the quantity
	6	The system will check If the respective book has the respective quantity available in the database, and display the Book ID, Book Name, Author Name, Order Date, and Return Date on the Book Order View screen.
	7	The administrator will click on the Save Book Order button on the Book Order View screen.
Postcondition	The system will show a Student Information dialog box to capture Student details for completing further orders.	
Comment	This shall validate the availability of the respective books from the books database.	

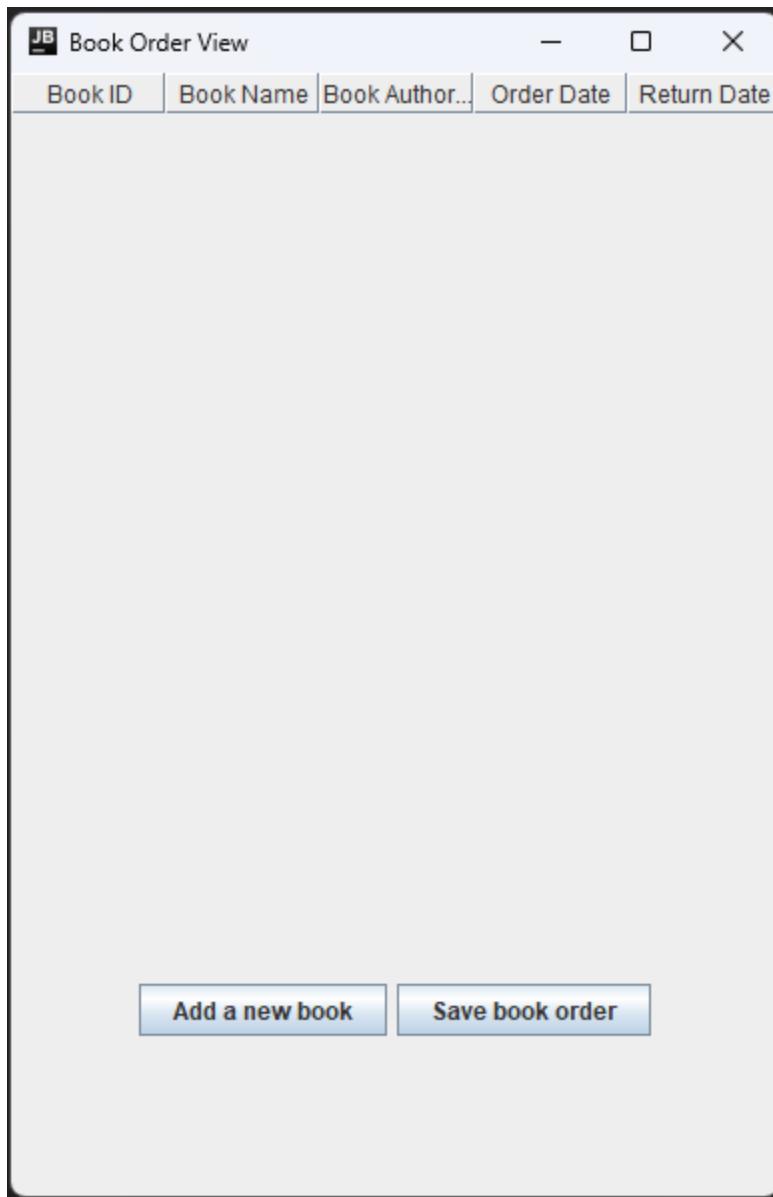
Actor Action	System Response
1 The administrator will click on the Order Book button on the main screen.	2 The “Book Order View” window appears.
3 The administrator will click on “Add a new book”.	4 Book ID input dialog box appears
4 The administrator will enter the book ID.	5 System will check if the respective book exists in the database and display the quantity input dialog box if the book exists. Otherwise, it shall display a pop-up window saying that this book does not exist in the database.
6 The administrator will enter the quantity	6 System will check If the respective book has the respective quantity available in the database, and display the Book Id, Book Name, Author Name, Order Date, and Return Date on the Book Order View screen.

7 The administrator will click on the Save Book Order button on the Book Order View screen.

8 System will show a Student Information dialog box.

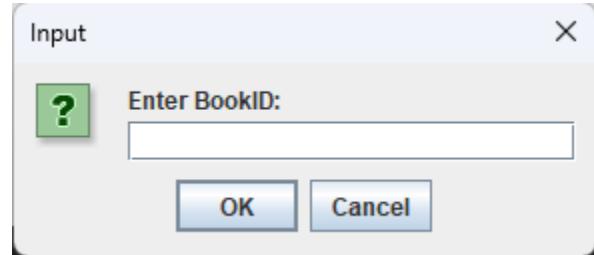
DESKTOP APPLICATION UI

1)



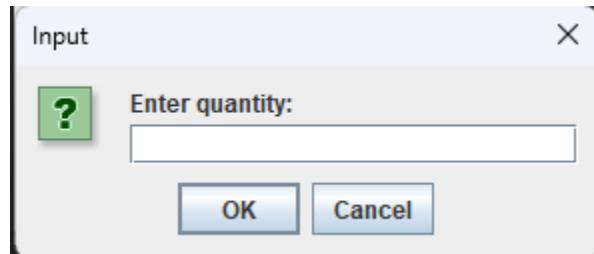
Book Order View screen appears and Admin clicks on “Add a new book” button

2)



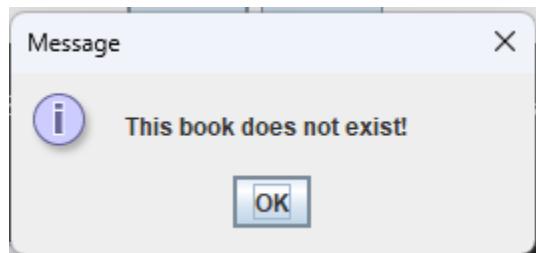
Enter Book Id dialog box appears

3)



Enter Quantity dialog box – Shown only if the book is available in Database -> user clicks on OK

Otherwise – “This book does not exist” dialog box appears



Book does not exist dialog box

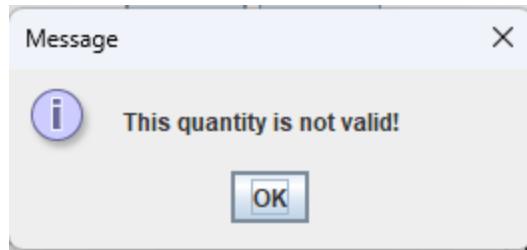
4)

The entered book gets added to Book Order View table and all the fields – Book Name, Book Author, Order Date and Return Date gets auto-filled in case User enters valid quantity.

JB Book Order View				
Book ID	Book Name	Book Author...	Order Date	Return Date
2	Physics 2.0	katrina Kaif	05-12-2023	03-02-2024
Add a new book		Save book order		

The entered book appears on Book Order View window

Otherwise – “Quantity is not valid” dialog box appears



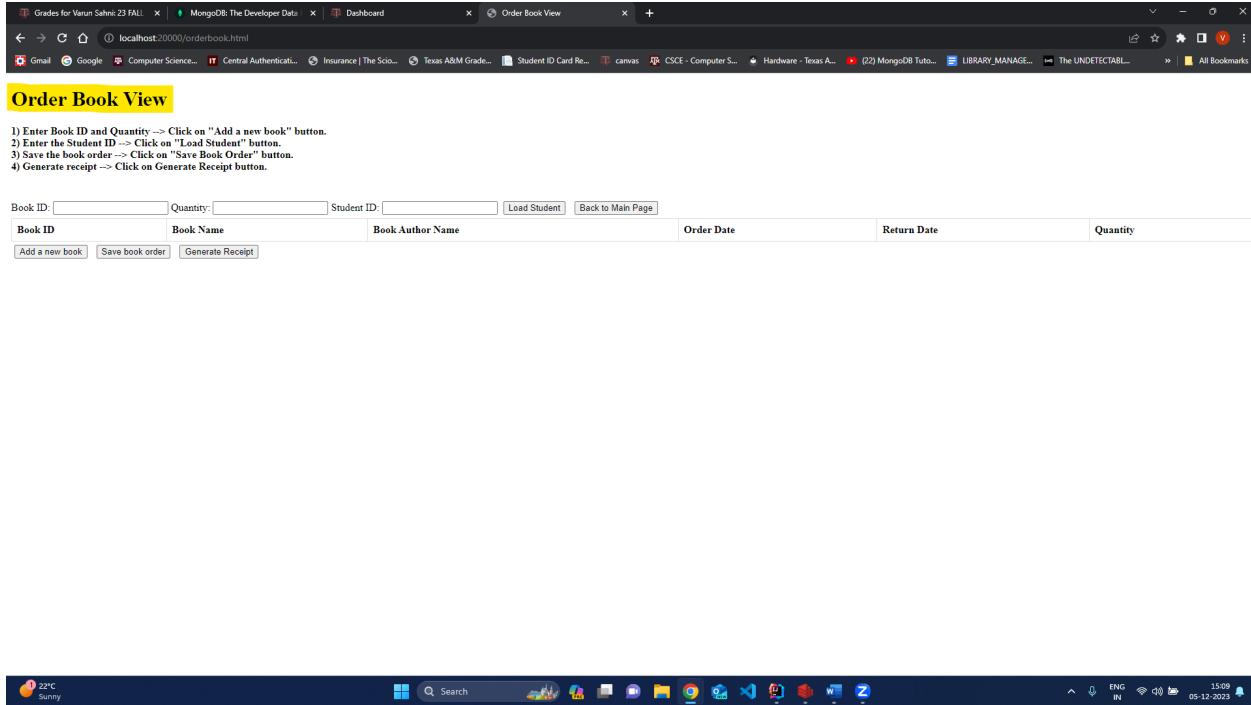
Quantity is not valid dialog box

Field	Value	edit	trash
BookID	2	edit	trash
BookName	Physics 2.0	edit	trash
AuthorName	katrina Kaif	edit	trash
Quantity	1994	edit	trash
Status	A	edit	trash

This book exists in Redis Database

WEB APPLICATION UI

1)



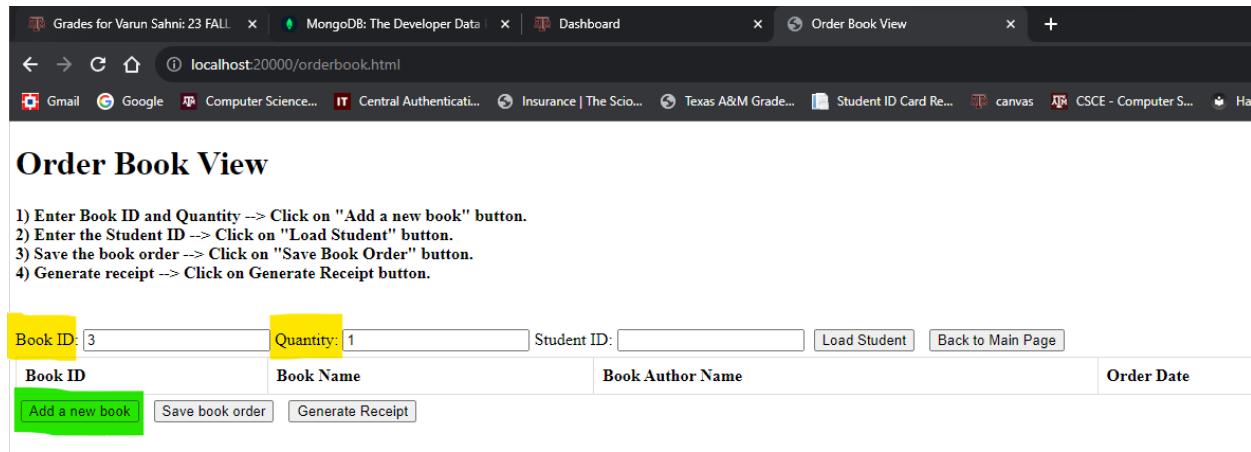
Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID:	Quantity:	Student ID:	Load Student	Back to Main Page	
Book ID	Book Name	Book Author Name	Order Date	Return Date	Quantity
<input type="button" value="Add a new book"/> <input type="button" value="Save book order"/> <input type="button" value="Generate Receipt"/>					

Book Order View screen appears

2)



Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID: 3	Quantity: 1	Student ID:	Load Student	Back to Main Page
Book ID	Book Name	Book Author Name	Order Date	
<input type="button" value="Add a new book"/> <input type="button" value="Save book order"/> <input type="button" value="Generate Receipt"/>				

Admin enters Book ID and Quantity → Then clicks on Add a new Book button

3)

Order Book View

1) Enter Book ID and Quantity -> Click on "Add a new book" button.
2) Enter the Student ID -> Click on "Load Student" button.
3) Save the book order -> Click on "Save Book Order" button.
4) Generate receipt -> Click on Generate Receipt button.

Book ID	Book Name	Book Author Name	Order Date	Return Date	Quantity
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 15:13:27	03/02/2024, 15:13:27	1

Add a new book | Save book order | Generate Receipt

Book ID, Book Name, Book Author Name, Order Date, Return Date, and Quantity gets added to the Cart and auto-appears on the page in tabular format

4)

HASH Book3

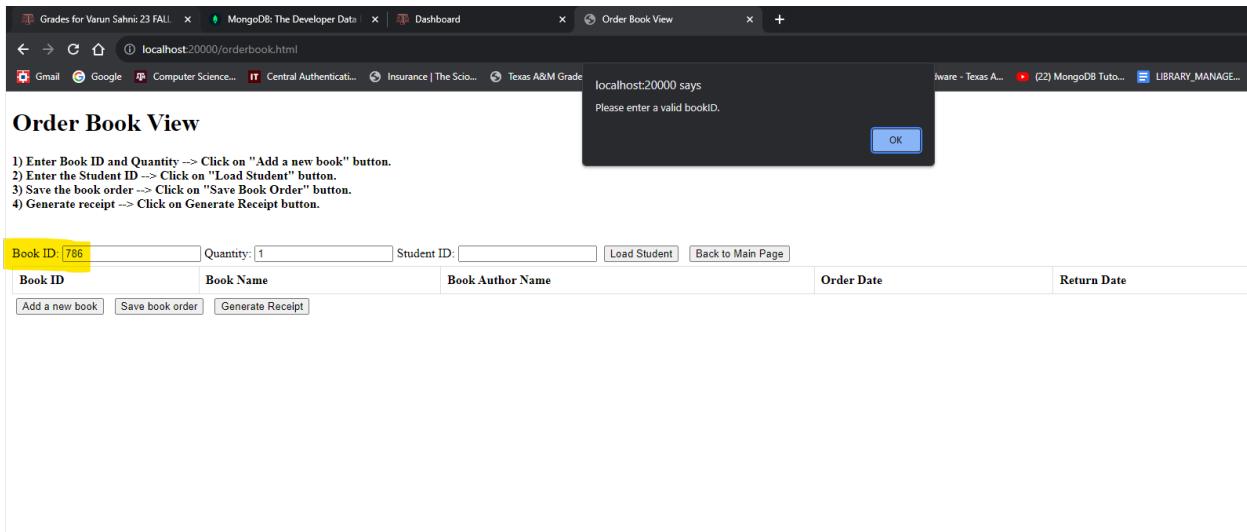
Key Size: 160 B Length: 5 TTL: No limit Last refresh: <1 min JSON Add Fields

Field	Value
BookID	3
BookName	Advance Mathematics with C#
AuthorName	Nushrat Bharucha
Quantity	2997
Status	A

This book exists in Redis Database

5)

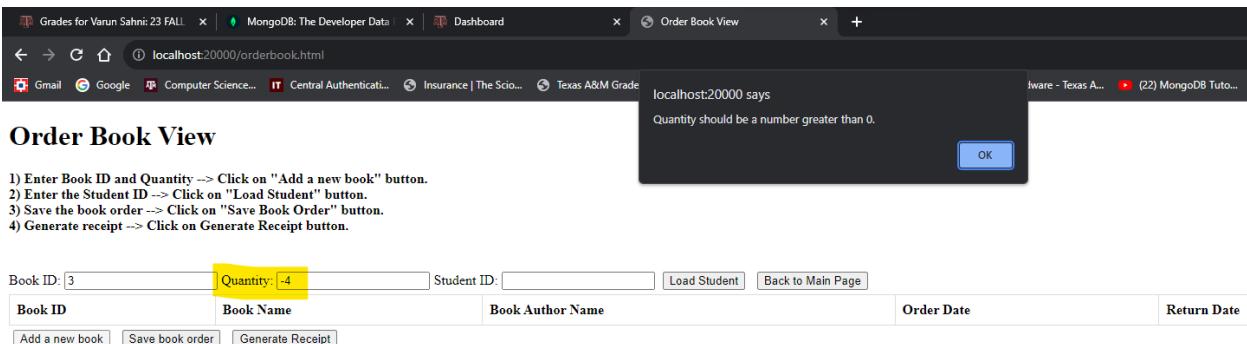
Validations:



Invalid Book ID entered alert pops up in case the Book does not exists in the Database

6)

Validations:



Invalid Quantity entered alert pops up in case Quantity <= 0

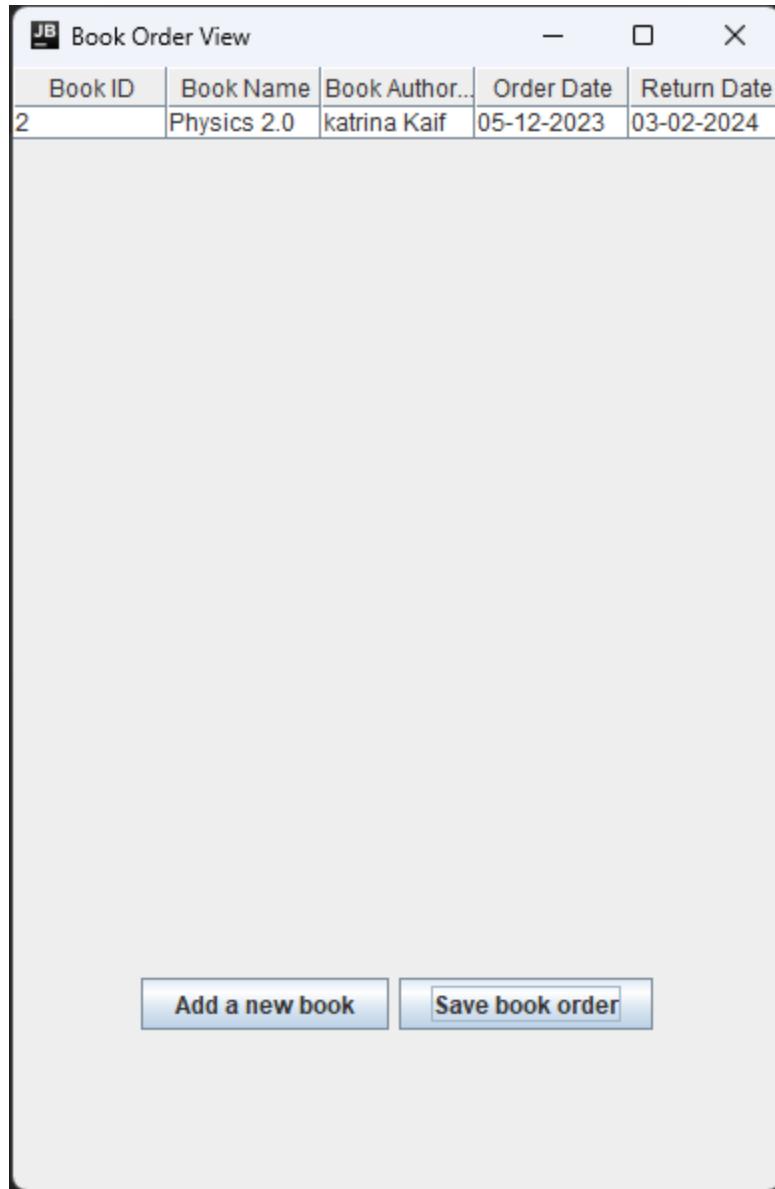
7th Use Case:

Use Case	Load student details for the current order	
Description	The administrator shall be adding Student details to the current order	
Precondition	Book Order View screen should be visible to the user. User should have added books to the Book Order View screen. Student Information dialog box should have been appeared on screen as in previous use case	
Ordinary Sequence	Step	Action
	1	The administrator will enter Student Id, and Click on “Load Student Details” button.
	2	System will validate the student’s details from the database and auto-populate the student’s name, email id and mobile number. In case, student does not exist in the database, a pop-up window will appear saying that this student does not exist in the database.
	3	The administrator will click on OK button
	4	System will generate an order.
Postcondition	System will generate an order.	
Comment	If the student ID corresponding is not present in the database, the application shall generate a pop up stating that this student does not exist in the database.	

Actor Action	System Response
1 The administrator will enter Student Id, and Click on “Load Student Details” button.	2 System will validate the student’s details from the database and auto-populate the student’s name, email id and mobile number. In case, student does not exist in the database, a pop-up window will appear saying that this student does not exist in the database.
3 The administrator will click on OK button	4 System will generate an order.

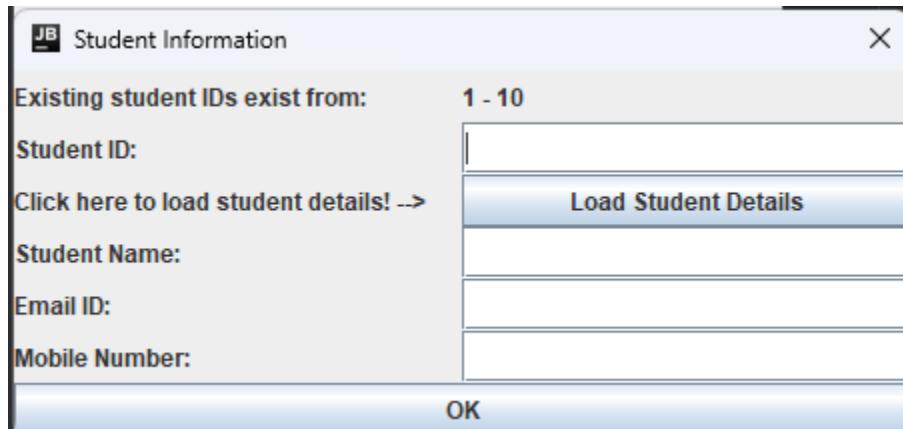
DESKTOP APPLICATION UI

1)



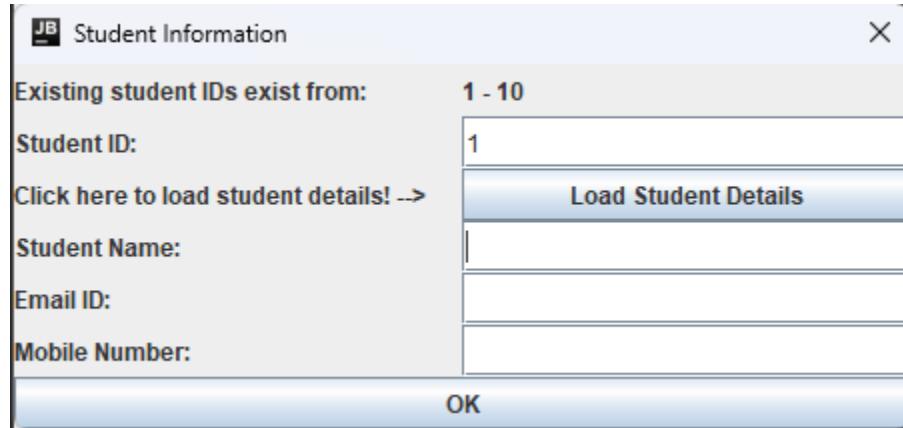
Book Order View page -> Admin clicks on Save book order button post adding the books

2)



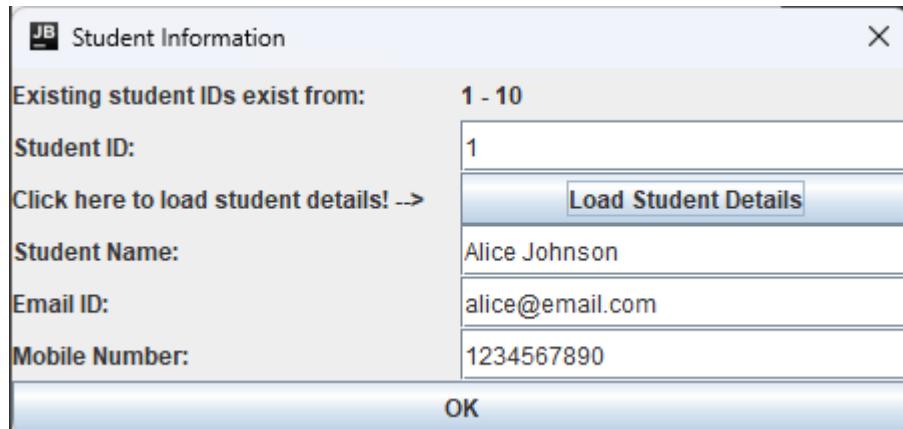
Student Information Dialog box appears

3)



Admin enters student id -> Admin clicks on Load Student Details button

4)



Student details gets updated from Database if student exists.

HASH Student1

Key Size: 152 B Length: 4 TTL: No limit Last refresh: now JSON Add Fields

Field	Value
StudentID	1
StudentName	Alice Johnson
EmailID	alice@email.com
StudentNumber	1234567890

Student with Student ID: 1 in Redis Database

Validations:

If Admin enters an invalid student ID

Student Information

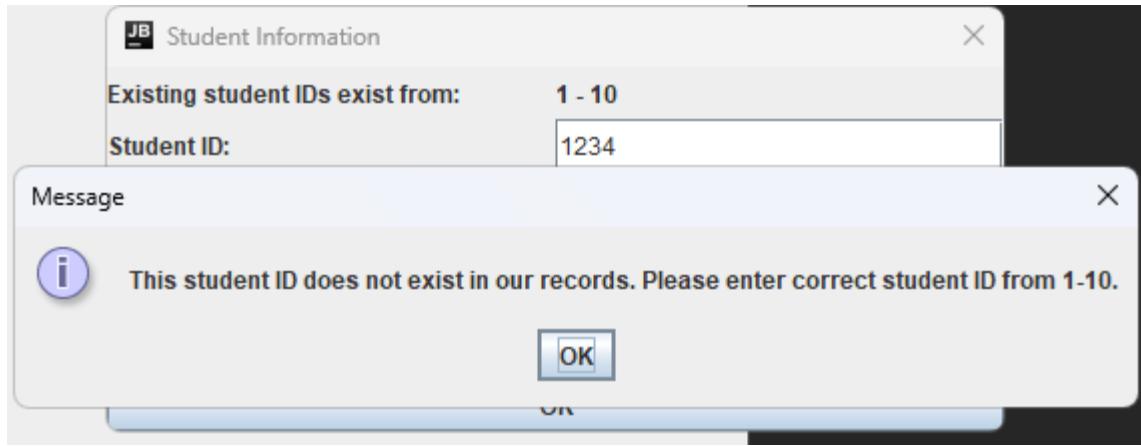
Existing student IDs exist from: 1 - 10

Student ID:	1234
Click here to load student details! -->	
Student Name:	
Email ID:	
Mobile Number:	

Load Student Details

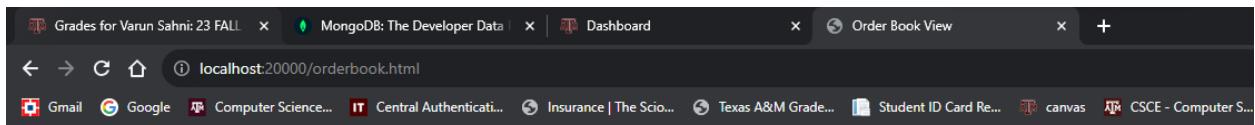
OK

Pop-up message appears stating that this student does not exists



WEB APPLICATION UI

1)



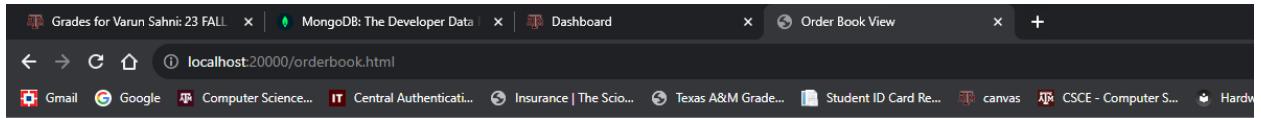
Order Book View

- 1) Enter Book ID and Quantity --> Click on "Add a new book" button.
- 2) Enter the Student ID --> Click on "Load Student" button.
- 3) Save the book order --> Click on "Save Book Order" button.
- 4) Generate receipt --> Click on Generate Receipt button.

Book ID: <input type="text" value="3"/>	Quantity: <input type="text" value="1"/>	Student ID: <input style="background-color: yellow;" type="text" value=""/>	<input type="button" value="Load Student"/>	<input type="button" value="Back to Main Page"/>
Book ID	Book Name	Book Author Name	Order Date	
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 15	

Order Book View page → Admin enters the Student ID

2)



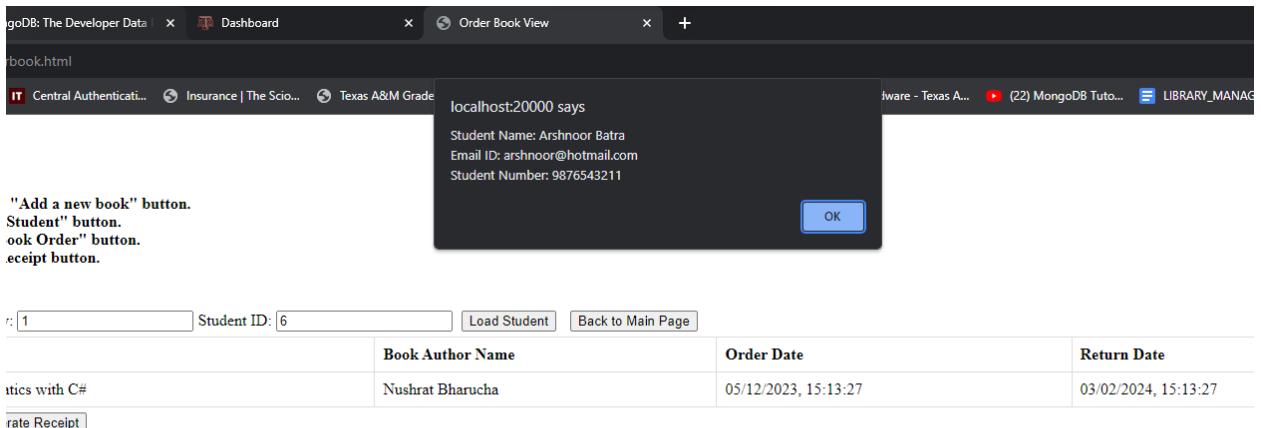
Order Book View

- 1) Enter Book ID and Quantity --> Click on "Add a new book" button.
- 2) Enter the Student ID --> Click on "Load Student" button.
- 3) Save the book order --> Click on "Save Book Order" button.
- 4) Generate receipt --> Click on Generate Receipt button.

Book ID:	3	Quantity:	1	Student ID:	6	<input type="button" value="Load Student"/>	<input type="button" value="Back to Main Page"/>
Book ID	Book Name	Book Author Name			Order Date		
3	Advance Mathematics with C#	Nushrat Bharucha			05/12/2023, 15:13:27		
<input type="button" value="Add a new book"/> <input type="button" value="Save book order"/> <input type="button" value="Generate Receipt"/>							

Admin clicks on “Load Student” button

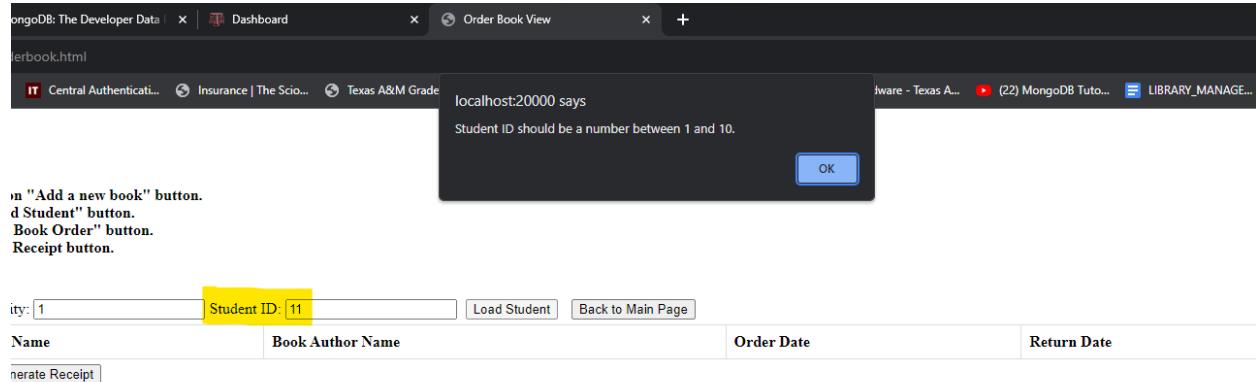
3)



Student Details alert pops up on the page and it shows the Student Name, Email ID, and Student Number

4)

Validations:



In case admin enters Student Id not between 1 to 10 → Invalid Student ID alert pops up on the page

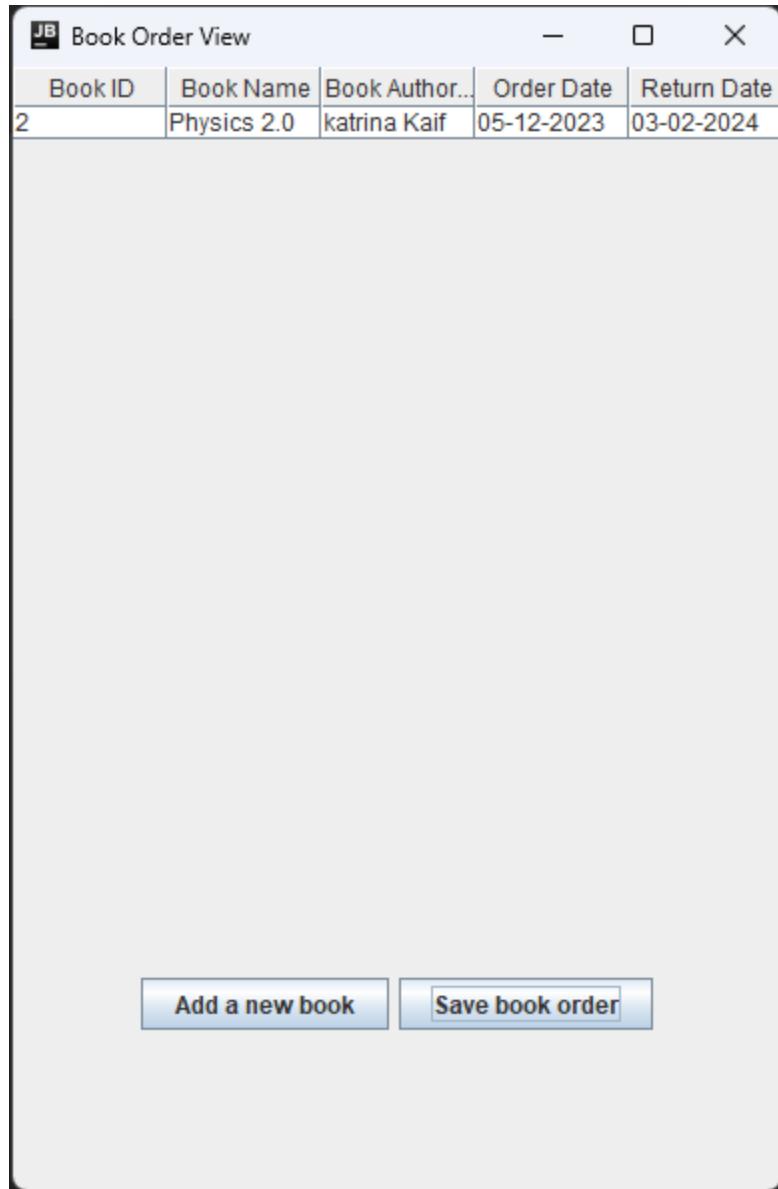
8th Use Case:

Use Case	Create Order	
Description	The administrator shall be able to create an order (OrderBook) for a student using library management system.	
Precondition	The book should be available in the Database of the Library Management System. The student should exist in the Library Database. The administrator should have accessed to Book Order View page The administrator must have added the books to Book Order View page The administrator must have entered and loaded the student's details and have clicked on Load student details button as done in previous use case	
Ordinary Sequence	Step	Action
	1	The administrator will click on Save Book Order button on Book Order View screen. A "Student Information" dialog box will appear.
	2	The administrator will enter Student Id, and Click on "Load Student Details" button.
	3	System will auto-populate the student's name, email id and mobile number. In case, student does not exist in the database, a pop-up window will appear saying that this student does not exist in the database.
	4	After the student details are loaded, system will generate an order, save it to the database, and order confirmation window will appear on screen.
Postcondition	The order will be saved in the database (Table: OrderBook). Book quantity for respective book decreases in database.	
Comment	The student Id field in each order shall contain the student Id of the respective student ordering the book.	
Exceptions	The student and the book should be present in the database.	

Actor Action	System Response
1 The administrator will click on Save Book Order button on Book Order View screen.	2 System will show a Student Information dialog box.
3 The administrator will enter the Student Id for which order needs to be made.	
4 The administrator will click on Load Student details button	5 System will validate the student details from the database and auto-populate the student's name, email id and mobile number.
6 The administrator will click on OK button	6 System will save the order in database, and display order confirmation page

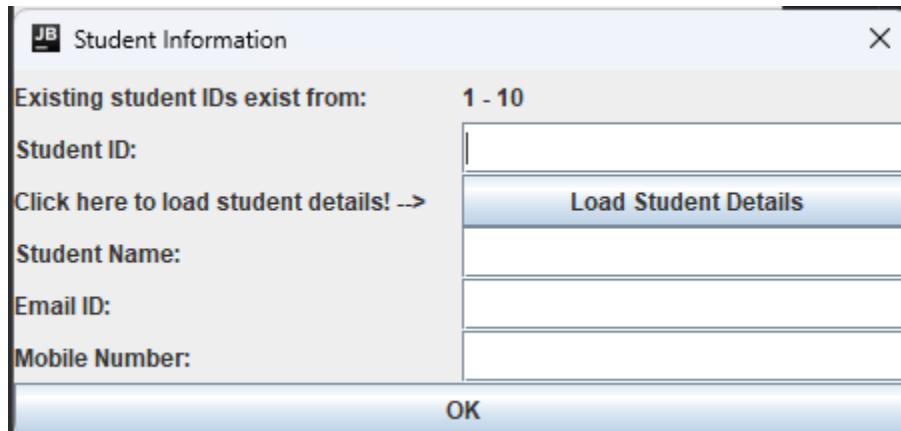
DESKTOP APPLICATION UI

1)



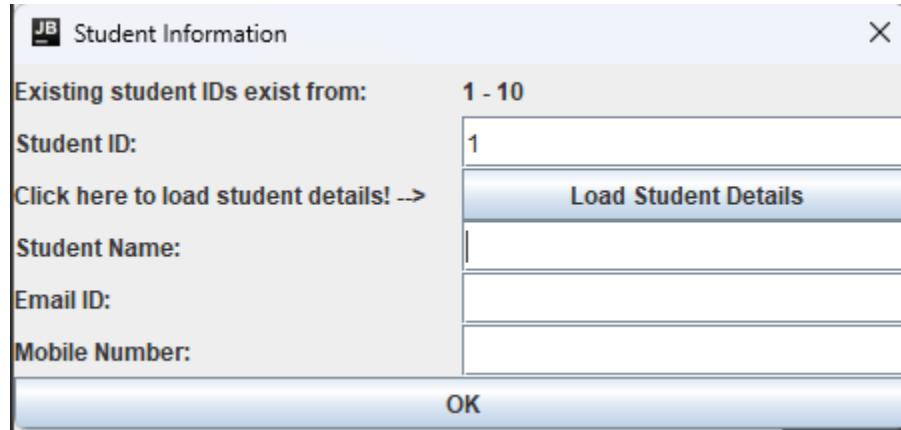
Book Order View page -> Admin clicks on Save book order button post adding the books

2)



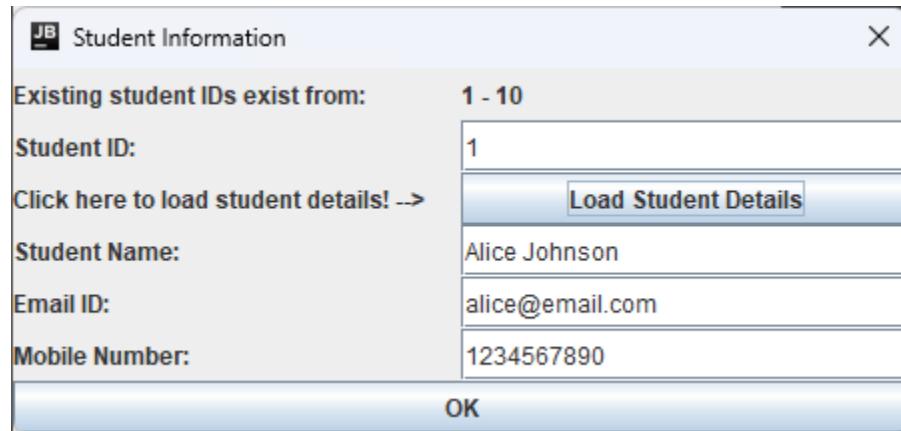
Student Information Dialog box appears

3)



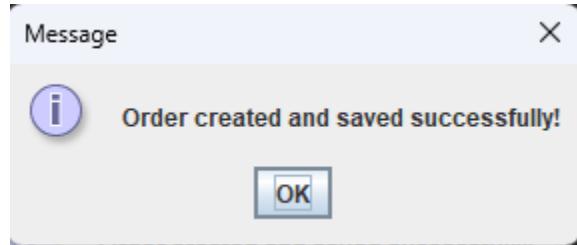
Admin enters student id -> Admin clicks on Load Student Details button

4)



Student details gets updated from Database if student exists.

5)



Order gets generated and saved to the MongoDB database.

LibraryApp.OrderBook

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 6KB TOTAL DOCUMENTS: 55 INDEXES TOTAL SIZE: 36KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#) [INSERT DOCUMENT](#)

[Filter](#) Type a query: { field: 'value' } [Reset](#) [Apply](#) [Options](#)

ReturnDate: "03-02-2024"

`_id: ObjectId('656f55fae471bc50564623e6')
OrderID: 54
OrderDate: "05-12-2023 10:52:24"
StudentID: 0
ReturnDate: "03-02-2024"`

`_id: ObjectId('656f56b5e471bc50564623ec')
OrderID: 55
OrderDate: "05-12-2023 10:55:24"
StudentID: 1
ReturnDate: "03-02-2024"`

MongoDB Database – OrderBook collection

WEB APPLICATION UI

1)

Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID:	Quantity:	Student ID:	Load Student	Back to Main Page	
Book ID	Book Name	Book Author Name	Order Date	Return Date	Quantity
<input type="button" value="Add a new book"/> <input type="button" value="Save book order"/> <input type="button" value="Generate Receipt"/>					

Book Order View screen appears

2)

Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID:	Quantity:	Student ID:	Load Student	Back to Main Page
Book ID	Book Name	Book Author Name	Order Date	
<input type="button" value="Add a new book"/>	<input type="button" value="Save book order"/>	<input type="button" value="Generate Receipt"/>		

Admin enters Book ID and Quantity → Then clicks on Add a new Book button

3)

Order Book View

1) Enter Book ID and Quantity -> Click on "Add a new book" button.
2) Enter the Student ID -> Click on "Load Student" button.
3) Save the book order -> Click on "Save Book Order" button.
4) Generate receipt -> Click on Generate Receipt button.

Book ID	Book Name	Book Author Name	Order Date	Return Date	Quantity
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 15:13:27	03/03/2024, 15:13:27	1

Add a new book | Save book order | Generate Receipt

Book ID, Book Name, Book Author Name, Order Date, Return Date, and Quantity gets added to the Cart and auto-appears on the page in tabular format

4)

HASH Book3

Key Size: 160 B Length: 5 TTL: No limit

Last refresh: <1 min

Add Fields

Field	Value
BookID	3
BookName	Advance Mathematics with C#
AuthorName	Nushrat Bharucha
Quantity	2997
Status	A

This book exists in Redis Database

5)

Grades for Varun Sahni: 23 FALL MongoDB: The Developer Data Dashboard Order Book View

localhost:20000/orderbook.html

Gmail Google Computer Science... Central Authenti... Insurance | The Scio... Texas A&M Grade... Student ID Card Re... canvas CSCE - Computer S...

Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID	Book Name	Book Author Name	Order Date
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 15

Add a new book Save book order Generate Receipt

Admin enters the Student ID

6)

Grades for Varun Sahni: 23 FALL MongoDB: The Developer Data Dashboard Order Book View

localhost:20000/orderbook.html

Gmail Google Computer Science... Central Authenti... Insurance | The Scio... Texas A&M Grade... Student ID Card Re... canvas CSCE - Computer S... Hardw...

Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID	Book Name	Book Author Name	Order Date
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 15:13:27

Add a new book Save book order Generate Receipt

Admin clicks on “Load Student” button

7)

A screenshot of a web browser window titled "Order Book View". The URL in the address bar is "localhost:20000/orderbook.html". A modal dialog box is displayed in the center of the page, containing the text: "localhost:20000 says", followed by "Student Name: Arshnoor Batra", "Email ID: arshnoor@hotmail.com", and "Student Number: 9876543211". At the bottom right of the dialog is a blue "OK" button. Below the dialog, there is a table with the following data:

Book ID	Quantity	Student ID	Load Student	Back to Main Page
3	1	6		
Book ID	Book Name	Book Author Name	Order Date	Return Date
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 15:13:27	03/02/2024, 15:13:27

At the bottom left of the page, there is a button labeled "Generate Receipt".

Student Details alert pops up on the page and it shows the Student Name, Email ID, and Student Number

8)

A screenshot of a web browser window titled "Order Book View". The URL in the address bar is "localhost:20000/orderbook.html". The page displays the same table and "Generate Receipt" button as in the previous screenshot. At the bottom of the page, there is a row of buttons: "Add a new book", "Save book order" (which is highlighted with a yellow box), and "Generate Receipt".

Admin clicks on the “Save book order” button

9)

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Order Book View". Below the title, there is a list of instructions:

- 1) Enter Book ID and Quantity --> Click on "Add a new book" button.
- 2) Enter the Student ID --> Click on "Load Student" button.
- 3) Save the book order --> Click on "Save Book Order" button.
- 4) Generate receipt --> Click on Generate Receipt button.

Below the instructions, there is a form with fields for Book ID (3), Quantity (1), and Student ID (6). Buttons include "Load Student", "Back to Main Page", "Add a new book", "Save book order", and "Generate Receipt".

A modal dialog box is centered on the page, displaying the message "localhost:20000 says Book order saved" with an "OK" button.

Book Order gets generated and “Book order saved” alert pops up on the page → Admin to click on OK button in alert

10)

The screenshot shows the MongoDB Cloud interface. On the left, there is a sidebar with "Data Services", "App Services", and "Charts" options. Under "Data Services", "Create Database" is available. Below that is a search bar for "Search Namespaces".

The main area shows the "LibraryApp.OrderBook" collection. It displays storage details: STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 6.99KB, TOTAL DOCUMENTS: 64, INDEXES TOTAL SIZE: 36KB.

There are several tabs at the top of the collection view: Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. On the right, there is an "INSERT DOCUMENT" button.

The "Find" section has a query builder: "Filter" (dropdown), "Type a query: { field: 'value' }", and buttons for "Reset", "Apply", and "Options".

The results list shows two documents highlighted with yellow boxes:

```
_id: ObjectId("656f6dbcd016084630d1d57c")
OrderID: 63
OrderDate: "05-12-2023 12:35:51"
StudentID: 0
ReturnDate: "03-02-2024"

_id: ObjectId("656fa6573d68dd56b4ac000c")
OrderID: 64
OrderDate: "05/12/2023, 16:38:14"
StudentID: 6
ReturnDate: "03-02-2024"
```

At the bottom, there are navigation buttons for "PREVIOUS" and "NEXT", and a status message "61-64 of 64 results".

Book Order gets saved in MongoDB database OrderBook collection

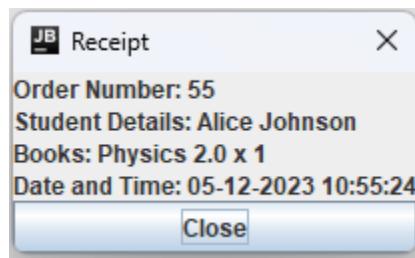
9th Use Case:

Use Case	Generate Receipt	
Description	The administrator shall be able to generate a receipt.	
Precondition	The order should have been generated and order confirmation page should have been appeared on screen	
Ordinary Sequence	Step	Action
	1	The administrator will click on OK button on the order generation confirmation age.
	2	The system will generate a receipt and receipt page will be displayed on screen.
Postcondition	The receipt will be saved in database (Table: Receipt). The receipt will be saved in the database.	
Comment	Receipt Table contains the books information.	
Exceptions	The student and the book should be present in the database.	

Actor Action	System Response
1 The administrator will click on OK button on order confirmation page	2 System will generate a receipt, save it to the database and display receipt dialog box.

DESKTOP APPLICATION UI

1)



Receipt gets generated

2)

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with a tree view of databases and collections. Under the 'LibraryApp' database, the 'Receipts' collection is selected. At the top, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. Below these are buttons for 'INSERT DOCUMENT', 'Reset', 'Apply', and 'Options'. A search bar labeled 'Filter' is present. The main area displays two documents from the 'Receipts' collection:

```
_id: ObjectId('656f58803f5599432ccb367a')
orderID: 54
userID: 2
dateTime: "05-12-2023 01:28:38"
studentDetails: "Bob Smith"
books: "Physics 2.0 x 1"

_id: ObjectId('656f58803f5599432ccb367b')
orderID: 55
userID: 1
dateTime: "05-12-2023 16:55:24"
studentDetails: "Alice Johnson"
books: "Physics 2.0 x 1"
```

At the bottom, it says '41-55 of 55 results'.

Receipt gets saved in the MongoDB Database – Receipts collection

WEB APPLICATION UI

1)

The screenshot shows a web browser window titled 'Order Book View' at the URL 'localhost:20000/orderbook.html'. The page has a header with tabs: 'Grades for Varun Sahni 23 FALL', 'Data | Cloud: MongoDB Cloud', 'Dashboard', 'Order Book View', and a '+' sign. Below the header, there's a navigation bar with links: 'Gmail', 'Google', 'Computer Science...', 'Central Authentication...', 'Insurance | The Scio...', 'Texas A&M Grade...'. A modal dialog box is open in the center of the page, displaying the message 'localhost:20000 says Book order saved' with an 'OK' button. The main content area is titled 'Order Book View' and contains instructions:

- 1) Enter Book ID and Quantity --> Click on "Add a new book" button.
- 2) Enter the Student ID --> Click on "Load Student" button.
- 3) Save the book order --> Click on "Save Book Order" button.
- 4) Generate receipt --> Click on Generate Receipt button.

Below the instructions is a form with fields: 'Book ID: [3]', 'Quantity: [1]', 'Student ID: [6]', 'Load Student', and 'Back to Main Page'. There's also a table:

Book ID	Book Name	Book Author Name	Order Date	Return
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 16:37:57	03/02/2024

At the bottom of the form are buttons: 'Add a new book', 'Save book order', and 'Generate Receipt'.

"Book order saved" alert pops up on the page → Admin to click on OK button in alert

2)

Grades for Varun Sahni: 23 FALL | Data | Cloud: MongoDB Cloud | Dashboard | Order Book View

localhost:20000/orderbook.html

Gmail Google Computer Science... Central Authentication... Insurance | The Scio... Texas A&M Grade... Student ID Card Re... canvas CSCE - Computer S... Hardware -

Order Book View

1) Enter Book ID and Quantity --> Click on "Add a new book" button.
2) Enter the Student ID --> Click on "Load Student" button.
3) Save the book order --> Click on "Save Book Order" button.
4) Generate receipt --> Click on Generate Receipt button.

Book ID: 3 Quantity: 1 Student ID: 6 Load Student Back to Main Page

Book ID	Book Name	Book Author Name	Order Date
3	Advance Mathematics with C#	Nushrat Bharucha	05/12/2023, 16:37:57

Add a new book Save book order Generate Receipt

Admin to click on Generate Receipt button

3)

Dashboard Order Book View

localhost:20000 says

Receipt Number: 685
Student Details: "Arshnoor Batra"
Books: "Advance Mathematics with C# x 1"
Date and Time: 12/5/2023 4:41:51 PM

OK

Student ID: 6 Load Student Back to Main Page

	Book Author Name	Order Date	Return
	Nushrat Bharucha	05/12/2023, 16:37:57	03/02/2

Receipt gets generated and Receipt Alert pops up on the page

4)

The screenshot shows the MongoDB Compass application interface. The top navigation bar has tabs for "Data Services", "App Services", and "Charts". Below the navigation is a search bar labeled "Search Namespaces". On the left, there's a sidebar with a tree view of namespaces: "YourDatabaseName", "library", "sample_airbnb", "sample_analytics", "sample_geospatial", "sample_guides", "sample_mflix", "sample_restaurants", "sample_supplies", "sample_training", and "sample_weatherdata". Under "YourDatabaseName", "Receipts" is selected and highlighted in green. The main panel title is "LibraryApp.Receipts" with sub-information: "STORAGE SIZE: 36KB", "LOGICAL DATA SIZE: 34KB", "TOTAL DOCUMENTS: 64", and "INDEXES TOTAL SIZE: 36KB". Below the title are buttons for "Find", "Indexes", "Schema Anti-Patterns", "Aggregation", and "Search Indexes", along with an "INSERT DOCUMENT" button. A "Filter" input field contains the query: { field: 'value' }. The results list shows two documents, both highlighted with a yellow box. The first document is: _id: ObjectId("656f95b34a65fb13146d39cf"), orderID: 63, userID: 1, dateTime: "12/5/2023 3:27:14 PM", studentDetails: "Alice Johnson", books: "Computer Architecture x 1". The second document is: _id: ObjectId("656fa7313d68dd56b4ac000ef"), orderID: 64, userID: 6, dateTime: "12/5/2023 4:41:51 PM", studentDetails: "Arshnoor Batra", books: "Advanced Mathematics with C# x 1". At the bottom of the results panel are buttons for "PREVIOUS" and "NEXT".

Receipt gets saved in the MongoDB Database – Receipts collection

Ques 2) Database design: Description of data entities and relationships, entity-relationship diagram, sample data.

A2. Below is the description of the data entities and relationships in our Library Management System App.

a. **Entity: User**

Attributes: UserID (Type: Int) (Primary Key)
UserName (Type: String)
Password (Type: String)
FullName (Type: String)

b. **Entity: Book**

Attribute: BookID (Type: Int) (Primary Key)
BookName (Type: String)
AuthorName (Type: String)
Quantity (Type: Int)
Status (Type: String)

c. **Entity: OrderBook**

Attribute: OrderID (Type: Int) (Primary Key)
StudentID (Type: Int)
OrderDate (Type: DATE)
ReturnDate (Type: DATE)

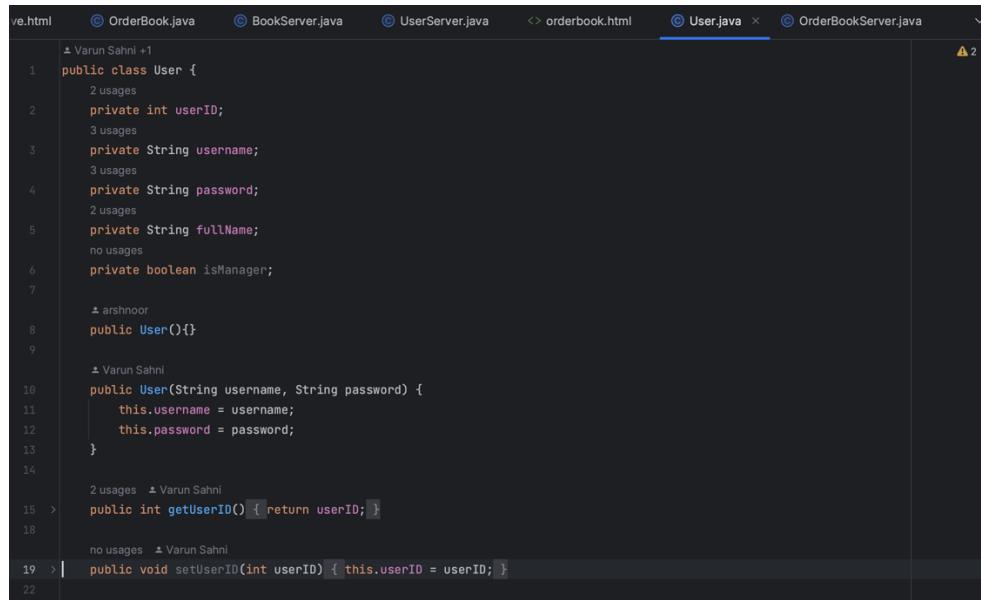
d. **Entity: Student**

Attribute: StudentID (Type: Int) (Primary Key)
StudentName (Type: String)
EmailID (Type: String)
StudentNumber (Type: String)

e. **Entity: Receipt**

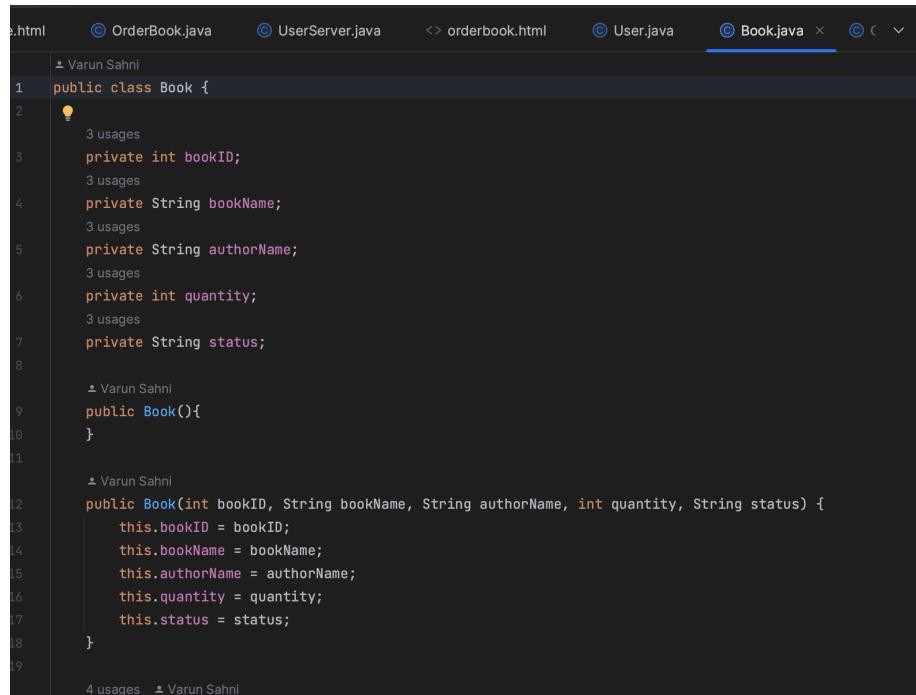
Attribute: ReceiptNumber (Type: Int) (Primary Key)
OrderID (Type: Int)
DateTime (Type: String)
StudentDetails (Type: String)
Books (Type: String)

DATA ENTITIES SCREENSHOTS:



```
ve.html OrderBook.java BookServer.java UserServer.java orderbook.html User.java OrderBookServer.java
1 Varun Sahni +1
2 public class User {
3     2 usages
4     private int userID;
5     3 usages
6     private String username;
7     3 usages
8     private String password;
9     2 usages
10    private String fullName;
11    no usages
12    private boolean isManager;
13
14    ▲ arshnoor
15    public User(){}
16
17    ▲ Varun Sahni
18    public User(String username, String password) {
19        this.username = username;
20        this.password = password;
21    }
22
23    2 usages ▲ Varun Sahni
24    public int getUserId() { return userID; }
25
26    no usages ▲ Varun Sahni
27    public void setUserId(int userID) { this.userID = userID; }
```

User Data Entity



```
ve.html OrderBook.java UserServer.java orderbook.html User.java Book.java
1 Varun Sahni
2 public class Book {
3     3 usages
4     private int bookID;
5     3 usages
6     private String bookName;
7     3 usages
8     private String authorName;
9     3 usages
10    private int quantity;
11    3 usages
12    private String status;
13
14    ▲ Varun Sahni
15    public Book(){
16    }
17
18    ▲ Varun Sahni
19    public Book(int bookID, String bookName, String authorName, int quantity, String status) {
20        this.bookID = bookID;
21        this.bookName = bookName;
22        this.authorName = authorName;
23        this.quantity = quantity;
24        this.status = status;
25    }
26
27    4 usages ▲ Varun Sahni
```

Book Data Entity

```
1.html  OrderBook.java  orderbook.html  User.java  Book.java  Student.java  Order...  
1  public class Student {  
2      private int studentID;  
3      private String studentName;  
4      private String emailID;  
5      private String studentNumber;  
6  
7      // Constructor with parameters  
8      public Student(int studentID, String studentName, String emailID, String studentNumber) {  
9          this.studentID = studentID;  
10         this.studentName = studentName;  
11         this.emailID = emailID;  
12         this.studentNumber = studentNumber;  
13     }  
14  
15     public Student() {  
16     }  
17  
18     // Getter and Setter methods for studentID  
19     public int getStudentID() { return studentID; }  
20 }
```

Student Data Entity

```
1.html  OrderBook.java  orderbook.html  User.java  Book.java  Student.java  Order...  
1  import java.util.ArrayList;  
2  
3  public class OrderBook {  
4      private int orderID;  
5      private int studentID;  
6      private String orderDate;  
7      private String returnDate;  
8      private Student student;  
9  
10     public Student getStudent() { return student; }  
11  
12     public void setStudent(Student student) { this.student = student; }  
13  
14     private List<OrderLineBook> lines;  
15  
16     public OrderBook() { lines = new ArrayList<>(); }  
17
```

OrderBook Data Entity

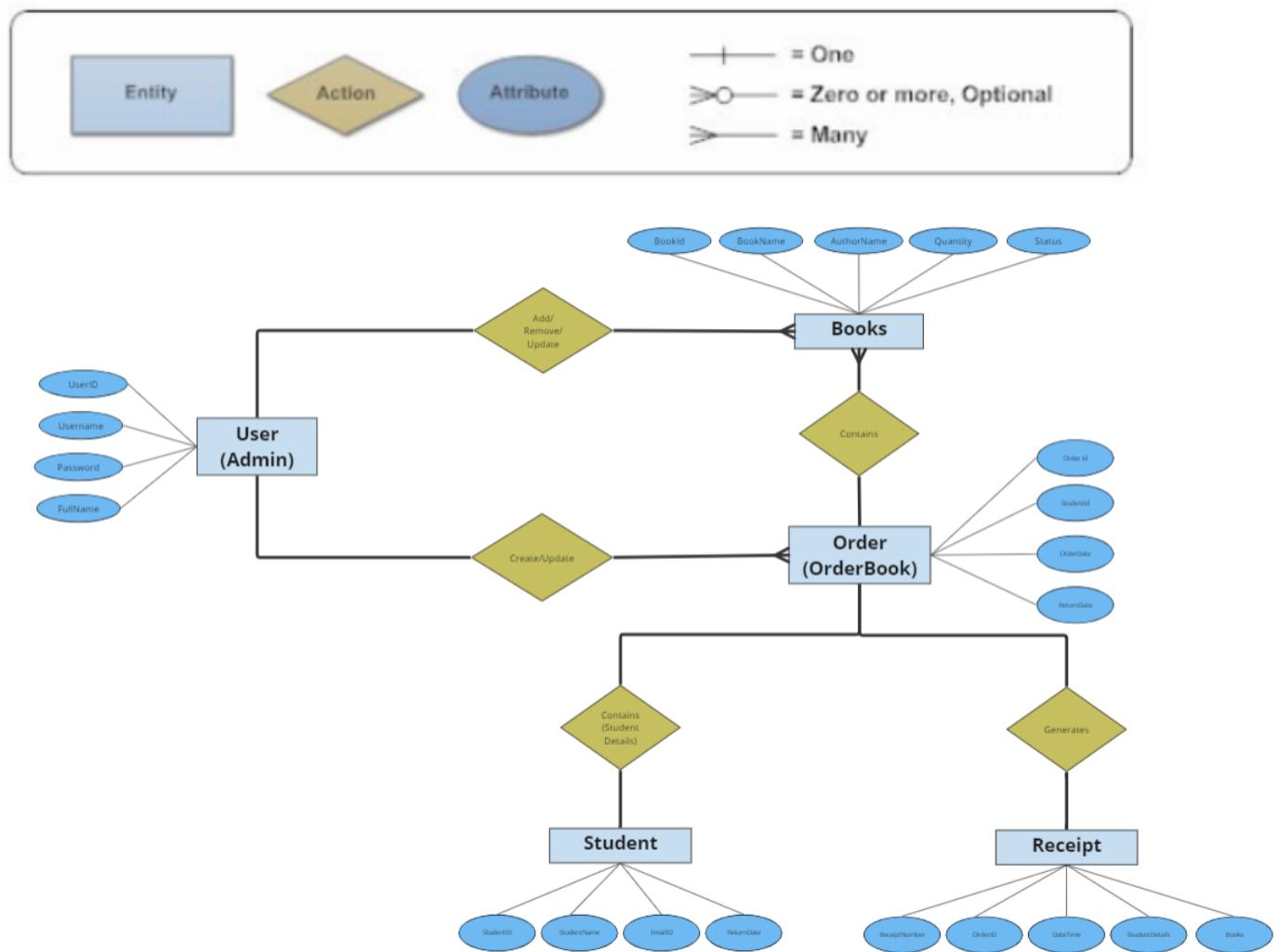
```
book.java      Receipt.java x  orderbook.html      User.java      Book.java      Student.java      OrderBook.java
21 usages  Varun Sahni +1
4   public class Receipt {
5       3 usages
6           private int receiptNumber;
7           4 usages
8           private int orderId;
9           4 usages
10          private String dateTime;
11          4 usages
12          private String student;
13          2 usages
14          private int studentID;
15          3 usages
16          private String books;
17
18          1 usage  arshnoor
19
20      >     public int getStudentID() { return studentID; }
21
22      >     1 usage  Varun Sahni
23      >     public void setStudentID(int studentID) { this.studentID = studentID; }
24
25      >     2 usages  Varun Sahni
26      >     public String getBooks() { return books; }
27
28      >     1 usage  Varun Sahni
29      >     public void setBooks(String books) { this.books = books; }
30
31      >     6 usages  Varun Sahni
```

Receipt Data Entity

Relationships:

- User-Book: One-to-many (An admin or librarian can manage multiple books)
- User-OrderBook: One-to-many (An admin can place multiple orders)
- Student-OrderBook: One-to-many (A student can have multiple orders)
- OrderBook-Book: One-to-many (An order can have multiple books)
- OrderBook-Receipt: One-to-one (Every order will generate a new receipt)

Entity Relationship Diagram:



Sample Data:

Redis Database:

The screenshot shows the Redis Database interface with the following details:

- Left Panel:** Shows a list of keys with their type (HASH), name, TTL, and size. Total count is 24.
- Right Panel:** A detailed view of the **Book2** key, which is a HASH type. It contains the following fields and values:

Field	Value
BookID	2
BookName	Physics 2.0
AuthorName	katrina kaf
Quantity	1992
Status	A

MongoDB Database:

The screenshot shows the MongoDB Atlas interface with the following details:

- Left Sidebar:** Shows the Project O project structure, including Database, SERVICES, SECURITY, and Goto sections.
- Top Bar:** Shows the URL as https://cloud.mongodb.com/v2/654d2bf8a5cb72736506b54#/metrics/replicaSet/654d2ca200fb43437f36144/explorer/LibraryApp/Receipts/_id.
- Main Area:**
 - Data Services:** Shows the **Store** database and collection.
 - LIBRARYAPP RECEIPTS:** Shows the **Receipts** collection with 13 databases and 35 collections. It includes a search bar for "Search Namespaces".
 - Details View:** Shows the **LibraryApp.Receipts** collection with the following details:
 - STORAGE SIZE: 34KB LOGICAL DATA SIZE: 4.94KB TOTAL DOCUMENTS: 34 INDEXES TOTAL SIZE: 86KB
 - VERSION: 6.0.12 REGION: AWS N. Virginia (us-east-1)
 - Find:** A search bar with the query: { field: 'value' }.
 - QUERY RESULTS:** Shows the results for the query: { field: 'value' } with 1-20 of many documents. One document is shown in full:


```
_id: ObjectId("656077fc1834ef432cb8c7b927")
orderID: 1
userID: 8
dateTime: "29-11-2023 11:33:35"
studentDetails: "Eva Garcia"
books: "Physics 2.0 x 1, Computer Arch x 1"

_id: ObjectId("656077fc1834ef432cb8c7b927")
orderID: 1
userID: 8
dateTime: "29-11-2023 11:59:41"
studentDetails: "Alice Johnson"
books: "Physics 2.0 x 1"
```

The screenshot shows the MongoDB Cloud interface for the 'LibraryApp.OrderBook' collection. The left sidebar lists databases and collections under 'LibraryApp'. The 'OrderBook' collection is selected. The main panel displays the collection's metrics (Storage Size: 34KB, Logical Data Size: 7.9KB, Total Documents: 73, Indexes Total Size: 34KB) and provides options for Find, Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. A search bar at the top allows querying documents. Below the search bar, the results are displayed as a list of documents, each with its _id, OrderID, OrderDate, StudentID, and ReturnDate.

```

_id: ObjectId('656775ea21188b55f6748d4b')
OrderID: 1
OrderDate: 2023-11-29T17:33:30.299+00:00
StudentID: 5
ReturnDate: "28-01-2024"

_id: ObjectId('65677c1434ef432cb8cf7b921')
OrderID: 2
OrderDate: 2023-11-29T17:59:40.491+00:00
StudentID: 1
ReturnDate: "28-01-2024"

_id: ObjectId('65677f99904383697544e4f8')
OrderID: 3

```

Specific Entities:

"Users" : Stored as HASH data type in Redis Database

The screenshot shows a Redis Hash Editor interface for a 'User:admin' entry. The key size is 136 B, length is 4, and TTL is No limit. The last refresh was just now. The JSON representation of the hash is shown as:

```

{
  "UserID": 1,
  "UserName": "admin",
  "Password": "password",
  "DisplayName": "Adam Smith"
}

```

The table below shows the field values:

Field	Value
UserID	1
UserName	admin
Password	password
DisplayName	Adam Smith

"Books" : stored as HASH datatype in Redis database. Redis is used for Books as books in the project scope of Librray Management Sytem is not a complicated entity as we only have books with single author and no reviews. Complicated data entities require Redis. But due to our project scope, we have made “Book” entity in Redis.

The screenshot shows the Redis UI interface. At the top, there is a blue button labeled "HASH" and the key name "Book1". Below the key name, it displays "Key Size: 144 B", "Length: 4", and "TTL: No limit". On the right side, there are buttons for "Last refresh: 6 min", "JSON", "Add Fields", and edit icons. The main area is a table with two columns: "Field" and "Value". The fields listed are BookName, AuthorName, Quantity, and Status, each with its corresponding value and edit icons.

Field	Value
BookName	Computer Architecture
AuthorName	Anushka Sharma
Quantity	960
Status	A

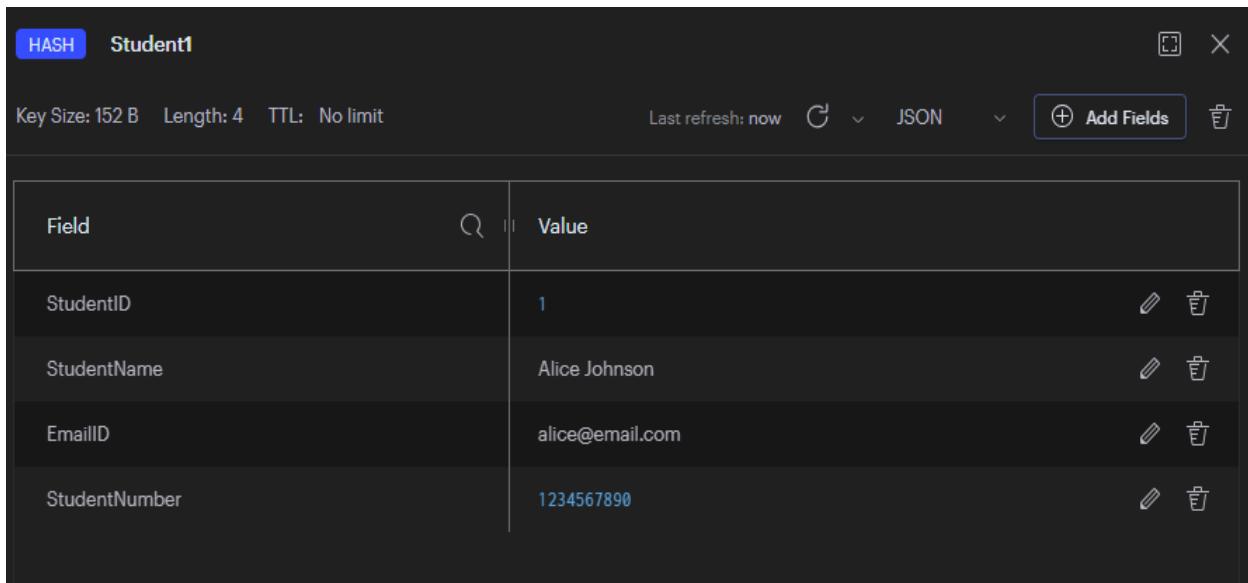
"OrderBook": Stored in MongoDB Atlas

The screenshot shows the MongoDB Atlas UI for the "LibraryApp.OrderBook" collection. At the top, it displays storage details: "STORAGE SIZE: 36KB", "LOGICAL DATA SIZE: 3.69KB", "TOTAL DOCUMENTS: 34", and "INDEXES TOTAL SIZE: 36KB". Below this, there are tabs for "Find", "Indexes", "Schema Anti-Patterns", "Aggregation", and "Search Indexes", along with an "INSERT DOCUMENT" button. A search bar at the top says "Type a query: { field: 'value' }" with "Filter" and "Apply" buttons. The results section shows "QUERY RESULTS: 1-20 OF MANY" and lists a single document with fields: _id, OrderID, OrderDate, StudentID, and ReturnDate.

```

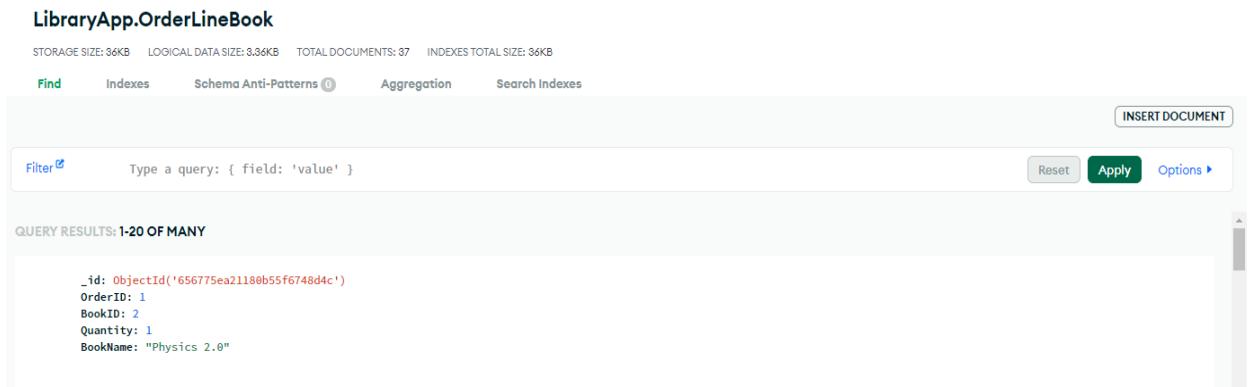
_id: ObjectId('1656775ea21180b55f6748d4b')
OrderID: 1
OrderDate: 2023-11-29T17:33:30.299+00:00
StudentID: 5
ReturnDate: "28-01-2024"
  
```

"Student" : stored as HASH datatype in Redis Database



Field	Value	edit	trash
StudentID	1	edit	trash
StudentName	Alice Johnson	edit	trash
EmailID	alice@email.com	edit	trash
StudentNumber	1234567890	edit	trash

"OrderLineBook" : Stored in MongoDB Atlas



LibraryApp.OrderLineBook	
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 3.34KB TOTAL DOCUMENTS: 37 INDEXES TOTAL SIZE: 36KB	
Find	Indexes
Schema Anti-Patterns	Aggregation
Search Indexes	INSERT DOCUMENT
Filter	Type a query: { field: 'value' }
	Reset Apply Options
QUERY RESULTS: 1-20 OF MANY	
<pre>_id: ObjectId('656775ea21180b55f6748d4c') OrderID: 1 BookID: 2 Quantity: 1 BookName: "Physics 2.0"</pre>	

"Receipt": Stored in MongoDB Atlas

LibraryApp.Receipts

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 4.94KB TOTAL DOCUMENTS: 34 INDEXES TOTAL SIZE: 36KB

[Find](#) [Indexes](#) [Schema Anti-Patterns](#) [Aggregation](#) [Search Indexes](#) [INSERT DOCUMENT](#)

[Filter](#) Type a query: { field: 'value' } [Reset](#) [Apply](#) [Options](#)

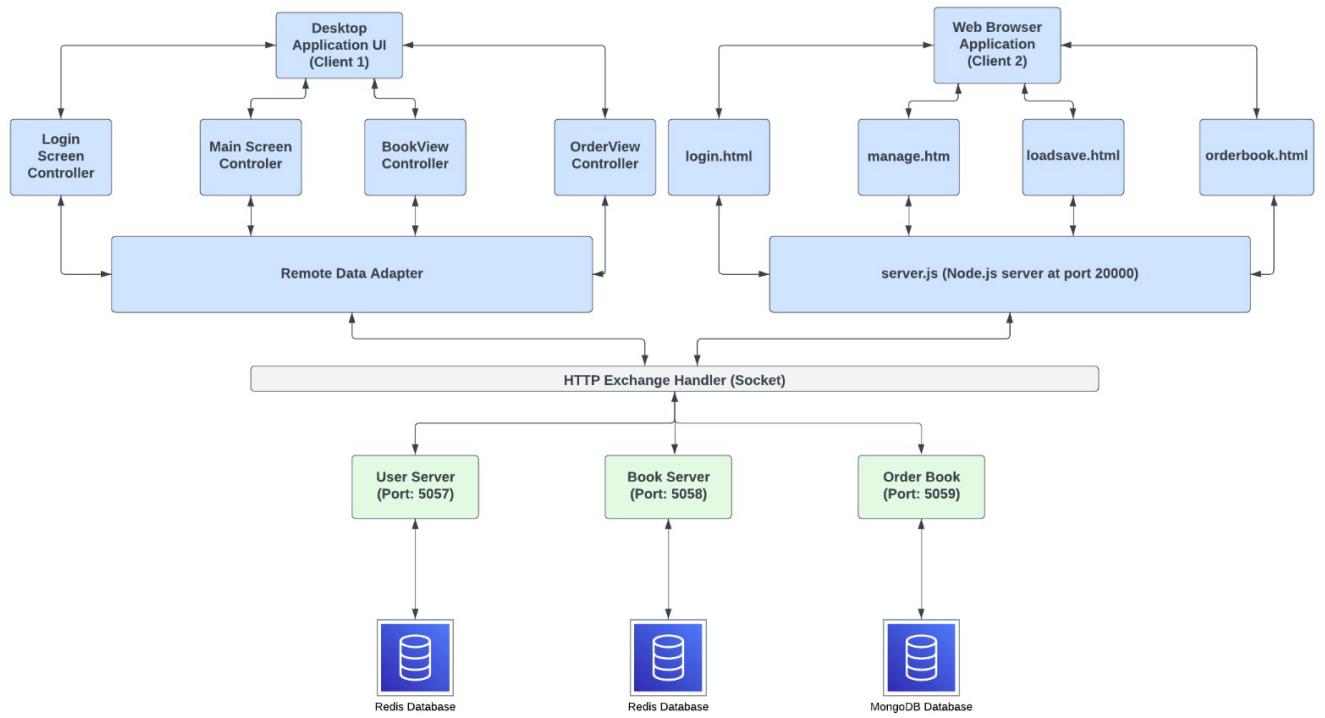
QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('656775f321180b55f6748d55')
orderID: 1
userID: 1
dateTime: "29-11-2023 11:33:35"
studentDetails: "Eva Garcia"
books: "Physics 2.0 x 1, Computer Arch x 1"
```

Ques 3) Architectural design: Description of client and server components; communication protocol (including data format), overall diagram of the system architecture.

Answer 3)

Client Server Architecture of Library Management System



Client Components:

- Application Class:** This serves as the entry point for the client-side application. It orchestrates the interaction between various components. This class will call various other client components to load the user interfaces to interact with the user.
- LoginScreenController Class:** Responsible for displaying the login screen and handling user authentication. It communicates with the server for user validation through Remote Data Adapter class.
- MainScreen Class:** Once a user is authenticated, this class displays the main screen of the library management system, allowing users to choose between book management and order creation. This has load/save book and create order features.
- BookViewController Class:** This class is responsible for displaying book-related information, allowing users to add or load books. It communicates with the server to fetch

book data through Remote Data Adapter class. This sends response as json, receives response in json and converts it into book objects.

5. **OrderViewController Class:** Handles the order creation and management, enabling user to create new orders. It communicates with the server for order-related data (passing the books in the current order from UI to the server and displaying the response received from the server). This sends response as json, receives response in json and converts it into order objects.
6. **RemoteDataAdapter Class:** Handles requests that are received from the controller screens (i.e. UI) and forwards these requests to respective Servers i.e. UserServer, BookServer, OrderBookServer. Reads server response and displays it to the User Interface.
7. **Index.html file:** Facilitates updating product information in a web-based inventory system. Users input product details in a form, triggering a server update via a POST request for seamless inventory management.
8. **Loadsave.html file:** Enables users to efficiently load and save book details in a user-friendly interface. It provides functionalities for retrieving book information based on ID and saving new book details through asynchronous interactions with a server.
9. **Login.html file:** Serves as the login interface for a Library Management System, allowing users to securely access the system by submitting their credentials. Successful logins redirect users to the management dashboard.
10. **Manage.html file:** Functions as the central dashboard for managing library operations after a successful login. Users can seamlessly navigate to order books or load/save book details, with their user information dynamically displayed.
11. **Orderbook.html file:** Facilitates the process of ordering books in a library setting. Users input book and student details, and the interface dynamically updates a table, allowing for the efficient placement and tracking of book orders.

Below are the screenshots for various client components.

Application class

```
② Application.java ×
  ↳ Varun Sahni
1 ⌂ public class Application {
2
3     3 usages
4     private static Application instance;    // Singleton pattern
5
6         1 usage  ↳ Varun Sahni
7         public static Application getInstance() {
8             if (instance == null) {
9                 instance = new Application();
10            }
11            return instance;
12        }
13
14     3 usages
15     private static RemoteDataAdapter dao;
16
17     1 usage
18     private User currentUser = null;
19
20         1 usage  ↳ Varun Sahni
21         public void setCurrentUser(User user) { this.currentUser = user; }
22
23     2 usages  ↳ Varun Sahni
24     private Application() {
25         dao = new RemoteDataAdapter();
26         dao.connect();
27     }
28
29         ↳ Varun Sahni
30         public static void main(String[] args) {
31             try {
32                 Application application = new Application();
33
34                 LoginScreenController login = new LoginScreenController(dao);
35                 login.setVisible(true);
36
37             } catch (Exception e) {
38                 System.out.println("e2: "+e.toString());
39                 e.printStackTrace();
40             }
41         }
42     }
```

Login screen controller

```
© LoginScreenController.java ×

1  > import ...
8
9  public class LoginScreenController extends JFrame implements ActionListener {
10    private JTextField txtUserName = new JTextField( columns: 20 );
11    private JPasswordField txtPassword = new JPasswordField( columns: 20 );
12    private JButton btnLogin = new JButton( text: "Login" );
13
14    RemoteDataAdapter dao;
15
16    public LoginScreenController(RemoteDataAdapter dao) {
17
18        this.dao = dao;
19
20        setTitle("Login Window");
21        this.setSize( width: 500, height: 300 );
22        getContentPane().setBackground(Color.GRAY);
23        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24
25        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
26        int x = (screenSize.width - getWidth()) / 2;
27        int y = (screenSize.height - getHeight()) / 2;
28        this.setLocation(x, y);
29
30        JPanel mainPanel = new JPanel();
31        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
32
33        mainPanel.setBackground(Color.GRAY);
34
35        JPanel titlePanel = new JPanel();
36        titlePanel.setAlignmentX(Component.CENTER_ALIGNMENT);
37        JLabel titleLabel = new JLabel( text: "Library Management System" );
38        titleLabel.setFont(new Font( name: "Arial", Font.BOLD, size: 25 ) );
39        titlePanel.add(titleLabel);
40
```

Main screen controller

```
© MainScreen.java ×
1 > import ...
5
2 usages ▾ Varun Sahni *
6 public class MainScreen extends JFrame {
7
8     3 usages
9         private JButton btnBuy = new JButton( text: "Order Book");
10    3 usages
11        private JButton btnSell = new JButton( text: "Load/Save Book");
12    1 usage
13        RemoteDataAdapter dao;
14
15    1 usage ▾ Varun Sahni
16    public MainScreen(RemoteDataAdapter dao) {
17
18        this.dao = dao;
19        this.setLayout(new BoxLayout(this.getContentPane(), BoxLayout.Y_AXIS));
20        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21        setTitle("Main Screen");
22        this.setSize( width: 500, height: 300);
23
24        // Center the application window on the screen
25        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
26        int x = (screenSize.width - getWidth()) / 2;
27        int y = (screenSize.height - getHeight()) / 2;
28        this.setLocation(x, y);
29
30        btnSell.setPreferredSize(new Dimension( width: 120, height: 50));
31        btnBuy.setPreferredSize(new Dimension( width: 120, height: 50));
32
33        JLabel title = new JLabel( text: "Library Management System");
34        title.setFont(new Font( name: "Sans Serif", Font.BOLD, size: 25));
35        JPanel panelTitle = new JPanel();
36        panelTitle.add(title);
            this.getContentPane().add(panelTitle);
```

Book View controller

```
⑥ BookViewController.java ×
1 > import ...
2 usages  ▲ Varun Sahni +1
3
4 usages
5 public class BookViewController extends JFrame implements ActionListener {
6
7     private JTextField txtBookID = new JTextField( columns: 10);
8
9     private JTextField txtBookName = new JTextField( columns: 30);
10
11    private JTextField txtAuthorName = new JTextField( columns: 30);
12
13    private JTextField txtBookQuantity = new JTextField( columns: 10);
14
15    private JTextField txtStatus = new JTextField( columns: 10);
16
17    private JButton btnLoad = new JButton( text: "Load Book");
18
19    private JButton btnSave = new JButton( text: "Save Book");
20
21
22    RemoteDataAdapter dao;
23
24
25    public BookViewController(RemoteDataAdapter dao) {
26
27        this.dao = dao;
28
29        this.setTitle("Manage Books");
30        this.setLayout(new BoxLayout(this.getContentPane(), BoxLayout.PAGE_AXIS));
31        this.setSize( width: 900, height: 300);
32
33        JPanel panelButton = new JPanel();
34        panelButton.add(btnLoad);
35        panelButton.add(btnSave);
36        this.getContentPane().add(panelButton);
37
38        JLabel statusDescription1 = new JLabel( text: "Set status as A/NA only");
39        statusDescription1.setFont(new Font( name: "Sans Serif", Font.BOLD, size: 12));
40        JLabel statusDescription2 = new JLabel( text: "A - Available, NA - Not Available");
```

OrderBook view controller

```
OrderBookViewController.java > import ...  
2 usages + Varun Sahni +1  
public class OrderBookViewController extends JFrame implements ActionListener {  
    16 usages  
    private OrderBook orderBook = null;  
    3 usages  
    private JButton btnAddBook = new JButton( text: "Add a new book");  
    3 usages  
    private JButton btnOrderBook = new JButton( text: "Save book order");  
    8 usages  
    private DefaultTableModel items = new DefaultTableModel(); // store information for the  
    4 usages  
    private JTable tblItems = new JTable(items);  
    // private DataAdapter dataAdapter;  
    9 usages  
    private RemoteDataAdapter dao;  
    no usages  
    private Book book; // Store the selected book  
    no usages  
    private int quantity; // Store the selected quantity  
    no usages  
    private Receipt receipt;  
    7 usages  
    private Student student;  
    1 usage + Varun Sahni  
    public OrderBookViewController(RemoteDataAdapter dao) {  
        this.dao = dao;  
        this.setTitle("Book Order View");  
        this.setLayout(new BoxLayout(this.getContentPane(), BoxLayout.Y_AXIS));  
        this.setSize( width: 400, height: 600);  
        items.addColumn( columnName: "Book ID");
```

Remote Data Adapter

```
© RemoteDataAdapter.java ×

1  > import ...
17
18  public class RemoteDataAdapter {
19      1 usage
20      private static final String URLUser = "http://localhost:5057";
21      1 usage
22      private static final String URLBook = "http://localhost:5058";
23      1 usage
24      private static final String URLStudent = "http://localhost:5059";
25      1 usage
26      private static final String BOOK = "/book";
27      1 usage
28      private static final String USER = "/user";
29      1 usage
30      private static final String STUDENT = "/student";
31      2 usages
32      public int orderCount;
33      no usages
34      private Gson gson = new Gson();
35      3 usages
36      private Socket s = null;
37      1 usage
38      private DataInputStream dis = null;
39      1 usage
40      private DataOutputStream dos = null;

        1 usage  ▲ Varun Sahni
41  public void connect() {
42
43      try {
44          s = new Socket("localhost", MAIN_SERVER_PORT);
45          dis = new DataInputStream(s.getInputStream());
46          dos = new DataOutputStream(s.getOutputStream());
47      } catch (Exception ex) {
48          System.out.println("connect Ex:" + ex.toString());
49          ex.printStackTrace();
50      }
51  }
```

Index.html

```
<> index.html <-->

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Update Product</title>
5  </head>
6  <body>
7      <h1>Update Product</h1>
8      <form id="productForm">
9          Product ID:<br>
10         <input type="text" id="productId" name="productId"><br>
11         Name:<br>
12         <input type="text" id="productName" name="productName"><br>
13         Price:<br>
14         <input type="text" id="productPrice" name="productPrice"><br>
15         Quantity:<br>
16         <input type="text" id="productQuantity" name="productQuantity"><br><br>
17         Seller Id:<br>
18         <input type="text" id="sellerId" name="sellerId"><br>
19
20         <input type="button" value="Update" onclick="postData()">
21     </form>
22
23     <div id="responseMessage"></div>
24
25     <script>
26         usage  ↗ arshnoor
27         function postData() {
28             const formData = {
29                 productID: document.getElementById('productId').value,
30                 name: document.getElementById('productName').value,
31                 price: document.getElementById('productPrice').value,
32                 quantity: document.getElementById('productQuantity').value,
33                 sellerID: document.getElementById('sellerId').value
34             };
35
36             fetch('http://localhost:8080/product/update', {
37                 method: 'POST',
38                 headers: {
39                     'Content-Type': 'application/json'
40                 }
41             )
42             .then(response => response.json())
43             .then(data => {
44                 const messageElement = document.getElementById('responseMessage');
45                 messageElement.textContent = `Product updated successfully! ${data.message}`;
46             })
47             .catch(error => {
48                 console.error('Error updating product:', error);
49             });
50         }
51     </script>
```

Loadsav.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Load/Save Books</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            text-align: center;
            margin: 50px auto;
            max-width: 600px;
        }
        .title {
            font-size: 25px;
            font-weight: bold;
            margin-bottom: 20px;
        }
        .button-container {
            display: flex;
            justify-content: space-around;
            margin-bottom: 20px;
        }
        .button {
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
            background-color: #007bff;
            color: #fff;
            border: none;
            border-radius: 5px;
        }
        .button:hover {
            background-color: #0056b3;
        }
        .input-container {
            margin-bottom: 15px;
        }
        .label {
```

Login.html

```
<> login.html <

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Login</title>
6      <style>
7          body {
8              font-family: Arial, sans-serif;
9              background-color: #f4f4f4;
10         }
11         .login-box {
12             width: 300px;
13             margin: 100px auto;
14             background-color: #fff;
15             padding: 20px;
16             border-radius: 5px;
17             box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
18         }
19         .login-box input[type="text"],
20         .login-box input[type="password"],
21         .login-box input[type="submit"] {
22             width: calc(100% - 20px);
23             padding: 10px;
24             margin-bottom: 15px;
25             border: 1px solid #ccc;
26             border-radius: 3px;
27             font-size: 16px;
28         }
29         .login-box input[type="submit"] {
30             background-color: #007bff;
31             color: #fff;
32             cursor: pointer;
33             transition: background-color 0.3s;
34         }
35         .login-box input[type="submit"]:hover {
36             background-color: #0056b3;
37         }
38     </style>
39 </head>
```

Manage.html

```
<> manage.html <x>
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Manage Books</title>
6      <style>
7          body {
8              font-family: Arial, sans-serif;
9              background-color: #f4f4f4;
10         }
11         .main-container {
12             text-align: center;
13             margin: 50px auto;
14             max-width: 600px;
15         }
16         .title {
17             font-size: 25px;
18             font-weight: bold;
19             margin-bottom: 20px;
20         }
21         .button-container {
22             display: flex;
23             justify-content: space-around;
24             margin-bottom: 20px;
25         }
26         .button {
27             padding: 10px 20px;
28             font-size: 16px;
29             cursor: pointer;
30             background-color: #007bff;
31             color: #fff;
32             border: none;
33             border-radius: 5px;
34         }
35         .button:hover {
36             background-color: #0056b3;
37         }
38     </style>
39 </head>
```

Orderbook.html

```
<> orderbook.html <x>

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Order Book View</title>
7      <style>
8          /* Add your CSS styles here */
9          table {
10              border-collapse: collapse;
11              width: 100%;
12          }
13
14          th, td {
15              border: 1px solid #ddd;
16              padding: 8px;
17              text-align: left;
18          }
19
20          button {
21              margin: 5px;
22          }
23      </style>
24  </head>
25  <body>
26      <h1>Book Order View</h1>
27
28      <!-- Input fields for Book ID and Quantity -->
29      <label for="bookID">Book ID:</label>
30      <input type="text" id="bookID">
31
32      <label for="quantity">Quantity:</label>
33      <input type="text" id="quantity">
34
35      <label for="studentID">Student ID:</label>
36      <input type="text" id="studentID">
37
38
```

Server Components:

Our project embraces a microservices architecture, featuring distinct servers for specialized operations:

1. Book Microservice:

- a. Employs the BookServer class, initializing an HTTP server on port 5058, with a dedicated '/book' endpoint handling GET and POST requests. The BookHandler class processes these requests, interacting with a database via DataAdapter2 for efficient book data management.
 - i. HTTP Server Initialization: Establishes an HTTP server on port 5058, with a '/book' endpoint handling GET and POST requests for retrieving and saving book information, respectively.
 - ii. BookHandler Class: Implements an HTTPHandler to process requests related to books, handling CORS headers, supporting GET (for retrieving book details) and POST (for saving new books), and interacting with a database through DataAdapter2.
 - iii. Data Processing: Utilizes Jackson's ObjectMapper for JSON serialization/deserialization, processes incoming book data, and communicates with a database to load or save book information based on the HTTP method.

2. User Microservices:

- a. Utilizes the UserServer class, initializing an HTTP server on port 5057, equipped with a '/user' endpoint handling GET requests for retrieving user details. The UserHandler class manages these requests, communicating with the database through DataAdapter2 to load user information.
 - i. HTTP Server Initialization: Initializes an HTTP server on port 5057, featuring a '/user' endpoint for handling GET requests to retrieve user details based on provided credentials.
 - ii. UserHandler Class: Implements an HTTPHandler for handling user-related requests, supporting GET requests to load user details from a database using DataAdapter2, and handling CORS headers.
 - iii. Data Processing: Utilizes Jackson's ObjectMapper for JSON processing, reads username and password from the request path, and communicates with a database to load user information.

3. Order Microservices:

- a. Incorporates the OrderBookServer class, establishing an HTTP server on port 5059, with '/order', '/student', and '/receipt' endpoints for handling POST requests related to order book, receipt, and student information. The OrderHandler,

StudentHandler, and ReceiptHandler classes process these requests, interacting with DataAdapter2 for efficient data storage and retrieval.

- i. HTTP Server Initialization: Establishes an HTTP server on port 5059, creating '/order', '/student', and '/receipt' endpoints for handling POST requests to save order book, receipt, and student information.
- ii. OrderHandler Class: Manages POST requests to '/order', processing incoming JSON data using Jackson's ObjectMapper, saving order book details to a database through DataAdapter2, and responding with JSON-formatted data.
- iii. StudentHandler and ReceiptHandler Classes: Handle GET and POST requests for student and receipt information, respectively, interacting with a database through DataAdapter2, and employing Jackson's ObjectMapper for JSON processing.

Common Elements of Server Classes:

Database Interaction: Utilizes DataAdapter2 to interact with Redis database and MongoDB database for reading and writing information related to books, users, orders, students, and receipts.

CORS Headers: Ensures Cross-Origin Resource Sharing (CORS) compatibility by setting appropriate headers to handle requests from different origins.

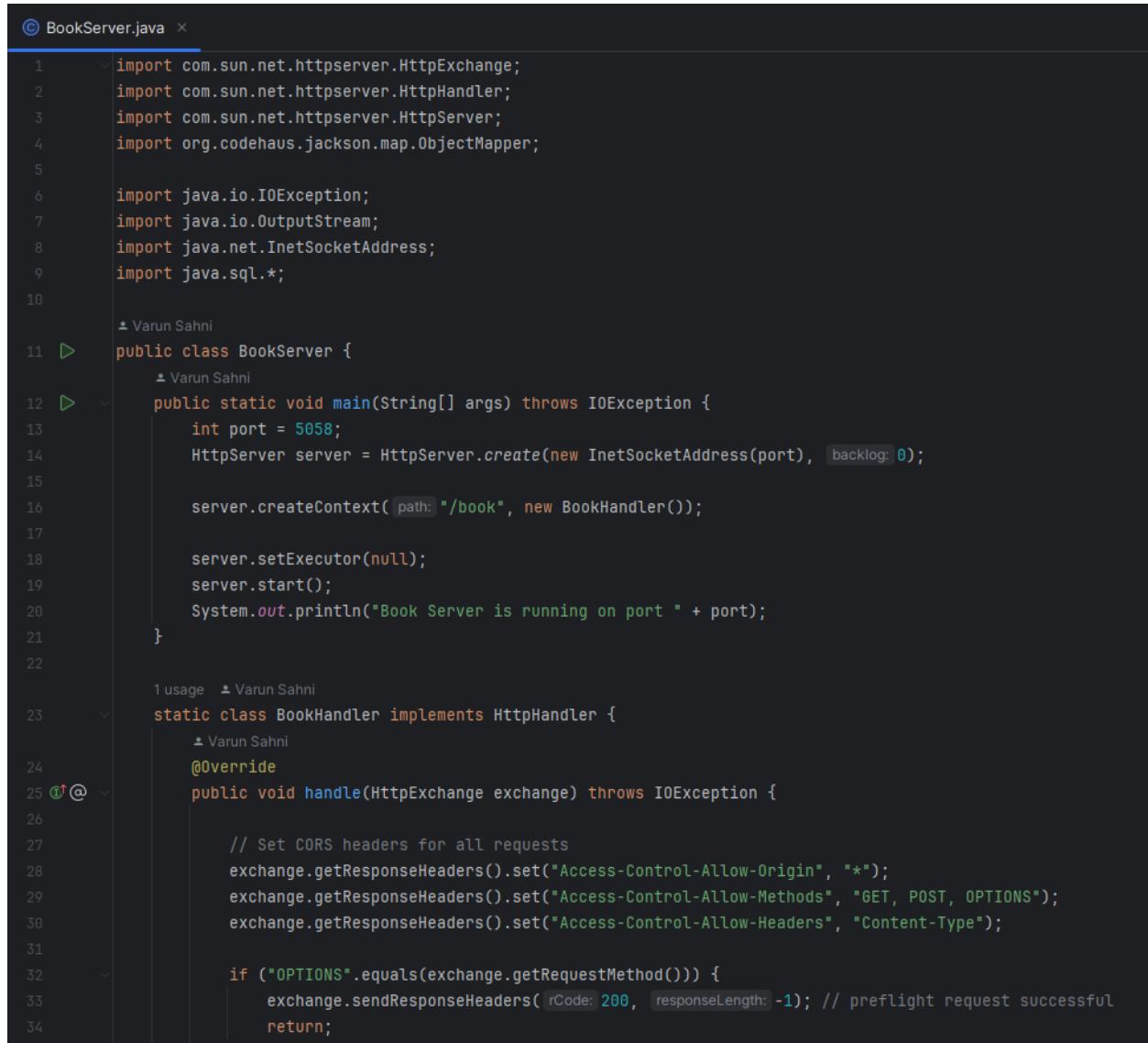
Server Output: Provides console output indicating the successful initiation of each server on specific ports for monitoring purposes.

4. DataAdapter2 Class:

- a. **Multi-Database Connectivity:** Connects seamlessly to both Redis and MongoDB, enabling comprehensive data retrieval and storage capabilities.
- b. **Operations Handling:** Manages loading and saving operations efficiently for books, orders, receipts, users, and students. Includes potential sections for Redis database and MongoDB database interactions, enhancing adaptability.

Below are the screenshots of the Server Components:

BookServer Class



```
© BookServer.java ×

1 import com.sun.net.httpserver.HttpExchange;
2 import com.sun.net.httpserver.HttpHandler;
3 import com.sun.net.httpserver.HttpServer;
4 import org.codehaus.jackson.map.ObjectMapper;
5
6 import java.io.IOException;
7 import java.io.OutputStream;
8 import java.net.InetSocketAddress;
9 import java.sql.*;
10
11 // Varun Sahni
12 public class BookServer {
13     // Varun Sahni
14     public static void main(String[] args) throws IOException {
15         int port = 5058;
16         HttpServer server = HttpServer.create(new InetSocketAddress(port), backlog: 0);
17
18         server.createContext(path: "/book", new BookHandler());
19
20         server.setExecutor(null);
21         server.start();
22         System.out.println("Book Server is running on port " + port);
23     }
24
25 // usage // Varun Sahni
26 static class BookHandler implements HttpHandler {
27     // Varun Sahni
28     @Override
29     public void handle(HttpExchange exchange) throws IOException {
30
31         // Set CORS headers for all requests
32         exchange.getResponseHeaders().set("Access-Control-Allow-Origin", "*");
33         exchange.getResponseHeaders().set("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
34         exchange.getResponseHeaders().set("Access-Control-Allow-Headers", "Content-Type");
35
36         if ("OPTIONS".equals(exchange.getRequestMethod())) {
37             exchange.sendResponseHeaders(rCode: 200, responseLength: -1); // preflight request successful
38             return;
39         }
40     }
41 }
```

UserServer Class

```
② UserServer.java ×

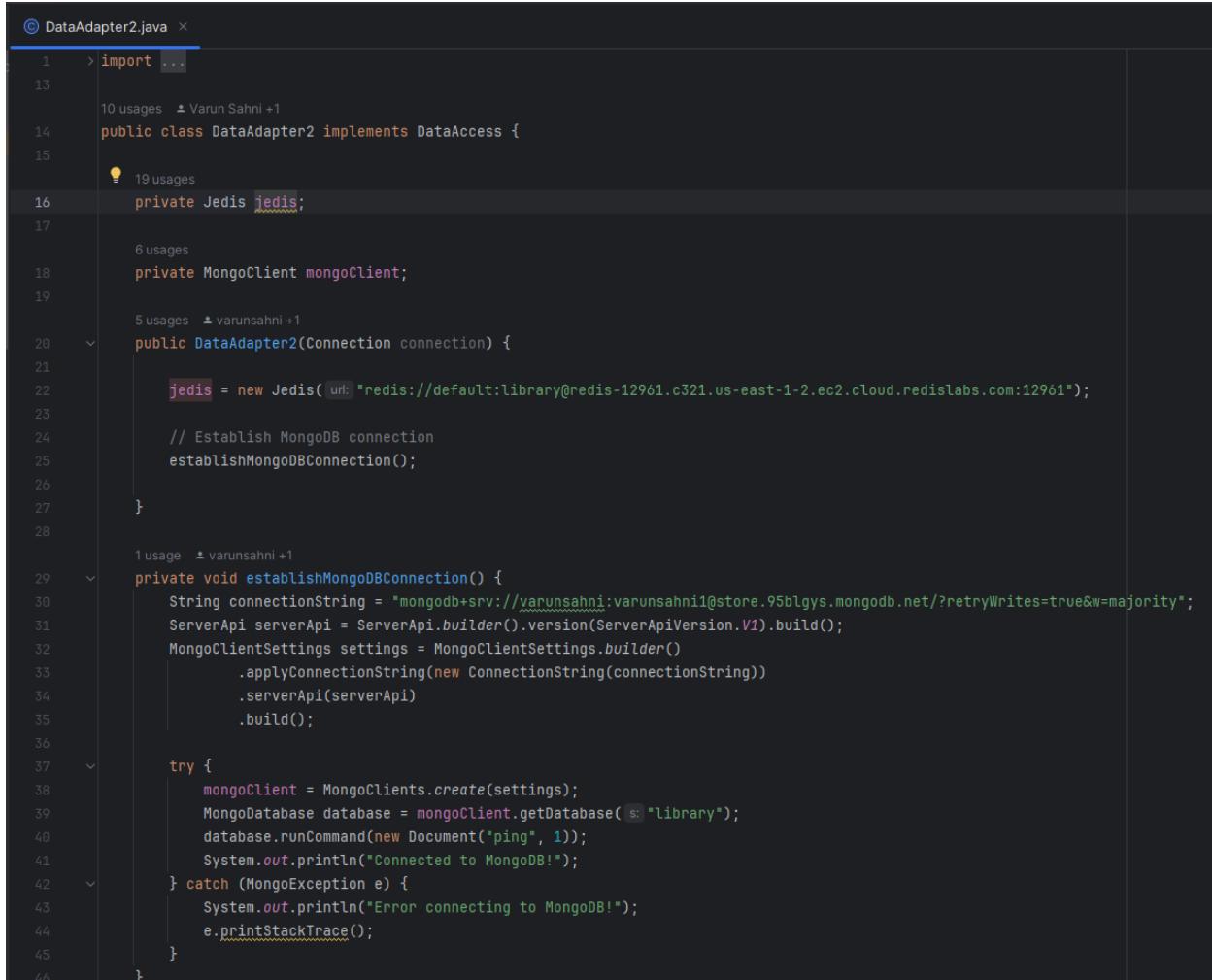
1  > import ...
9
10
11  ▷ public class UserServer {
12    ▷ Varun Sahni
12  ▷   public static void main(String[] args) throws IOException {
13    int port = 5057;
14    HttpServer server = HttpServer.create(new InetSocketAddress(port), backlog: 0);
15
16    server.createContext( path: "/user", new UserHandler());
17
18    server.setExecutor(null);
19    server.start();
20    System.out.println("User Server is running on port " + port);
21  }
22
23  1 usage  ▷ Varun Sahni
23  ▷ static class UserHandler implements HttpHandler {
24    ▷ Varun Sahni
24  ▷   @Override
25  ▷     public void handle(HttpExchange exchange) throws IOException {
26    exchange.getResponseHeaders().add("Access-Control-Allow-Origin", "http://localhost:20000");
27    exchange.getResponseHeaders().add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");
28    exchange.getResponseHeaders().add("Access-Control-Allow-Headers", "Content-Type, Authorization");
29    exchange.getResponseHeaders().add("Access-Control-Allow-Credentials", "true");
30    Connection conn = null;
31
32    DataAdapter2 dataAdapter = new DataAdapter2(conn);
33    if (*GET*.equals(exchange.getRequestMethod())) {
34      String requestPath = exchange.getRequestURI().getPath();
35      String[] pathSegments = requestPath.split( regex: "[/]" );
36      String username = pathSegments[pathSegments.length-2];
37      String password = pathSegments[pathSegments.length-1];
38      User user = dataAdapter.loadUser(username, password);
39      String jsonResponse = objectToJson(user);
40
41      exchange.getResponseHeaders().set("Content-Type", "application/json");
42      exchange.sendResponseHeaders( rCode: 200, jsonResponse.getBytes().length );
43      OutputStream os = exchange.getResponseBody();

```

OrderBook Server

```
① OrderBookServer.java ×
1  > import ...
10
11  ▷ public class OrderBookServer {
12    ▷ Varun Sahni
13    ▷ Varun Sahni
14      public static void main(String[] args) throws IOException {
15        int port = 5059;
16        HttpServer server = HttpServer.create(new InetSocketAddress(port), backlog: 0);
17
18        server.createContext(path: "/order", new OrderHandler());
19        server.createContext(path: "/student", new StudentHandler());
20        server.createContext(path: "/receipt", new ReceiptHandler());
21        server.setExecutor(null);
22        server.start();
23
24        System.out.println("OrderBook Server is running on port " + port);
25
26  @Override
27  static class OrderHandler implements HttpHandler {
28    ▷ Varun Sahni
29    @Override
30    public void handle(HttpExchange exchange) throws IOException {
31      Connection conn = null;
32
33      DataAdapter2 dataAdapter = new DataAdapter2(conn);
34      if ("POST".equals(exchange.getRequestMethod())) {
35        ObjectMapper objectMapper = new ObjectMapper();
36        OrderBook receivedOrder = objectMapper.readValue(exchange.getRequestBody(), OrderBook.class);
37        dataAdapter.saveOrderBook(receivedOrder);
38
39        String jsonResponse = objectToJson(receivedOrder);
40        exchange.getResponseHeaders().set("Content-Type", "application/json");
41        exchange.sendResponseHeaders(rCode: 200, jsonResponse.getBytes().length);
42        OutputStream os = exchange.getResponseBody();
43        os.write(jsonResponse.getBytes());
44        os.close();
45      } else {
46        exchange.sendResponseHeaders(rCode: 405, responseLength: 0); // Method not allowed for non-POST requests
47      }
48    }
49  }
```

DataAdapter2 Class



```
⑥ DataAdapter2.java ×
1  > import ...
13
10 usages ▲ Varun Sahni +1
14 public class DataAdapter2 implements DataAccess {
15
16     ⚡ 19 usages
17     private Jedis jedis;
18
19         6 usages
20     private MongoClient mongoClient;
21
22         5 usages ▲ varunsahni +1
23     public DataAdapter2(Connection connection) {
24
25         jedis = new Jedis("redis://default:library@redis-12961.c321.us-east-1-2.ec2.cloud.redislabs.com:12961");
26
27         // Establish MongoDB connection
28         establishMongoDBConnection();
29
30     }
31
32     1 usage ▲ varunsahni +1
33     private void establishMongoDBConnection() {
34         String connectionString = "mongodb+srv://varunsahni:varunsahni@store.95blgys.mongodb.net/?retryWrites=true&w=majority";
35         ServerApi serverApi = ServerApi.builder().version(ServerApiVersion.V1).build();
36         MongoClientSettings settings = MongoClientSettings.builder()
37             .applyConnectionString(new ConnectionString(connectionString))
38             .serverApi(serverApi)
39             .build();
40
41         try {
42             mongoClient = MongoClients.create(settings);
43             MongoDB database = mongoClient.getDatabase("library");
44             database.runCommand(new Document("ping", 1));
45             System.out.println("Connected to MongoDB!");
46         } catch (MongoException e) {
47             System.out.println("Error connecting to MongoDB!");
48             e.printStackTrace();
49         }
50     }
51 }
```

The client-server library management system uses the following communication protocol:

Transport Protocol: HTTP (Hyper Text Transfer Protocol) is used for reliable data transfer between the client and server.

Communication Protocol:

The communication between the client and server classes is based on the HTTP protocol. Both the client and server classes utilize the RESTful architectural style, where HTTP methods (GET, POST, OPTIONS) are employed for communication.

HTTP Methods:

GET: Used for retrieving data (e.g., fetching books, users, or students).

POST: Employed for creating or updating data (e.g., saving a book, order, or receipt).

OPTIONS: Utilized for preflight requests to check server capabilities.

Data Format:

Request Format: Clients send JSON-formatted data in the request body (e.g., creating or updating entities).

Response Format: Servers respond with JSON-formatted data containing the requested information or status of the operation.

System Architecture Overview:

The system architecture comprises multiple components, including client classes (BookClient, UserClient, OrderBookClient) and server classes (BookServer, UserServer, OrderBookServer). The overall architecture follows a client-server model with a RESTful design.

Client Classes:

BookClient: Interacts with the server to perform operations related to books.

UserClient: Manages interactions with the server for user-related tasks.

OrderBookClient: Handles interactions with the server regarding orders.

Server Classes:

BookServer: Listens for and handles HTTP requests related to books.

UserServer: Manages HTTP requests for user-related operations.

OrderBookServer: Handles HTTP requests for order-related functionalities.

Communication Flow:

Clients send HTTP requests with JSON-formatted data to the respective server endpoints (/book, /user, /order).

Servers process the requests, perform necessary actions (e.g., querying a database, saving data), and respond with JSON-formatted results.

The system utilizes the HTTP protocol and JSON format for data exchange, ensuring a standardized and interoperable communication process.

Data Storage:

DataAdapter2: Manages data storage and retrieval using Redis for quick key-value operations and MongoDB for more complex data structures.

This architecture promotes scalability, maintainability, and separation of concerns, allowing each component to handle specific functionalities. The adoption of RESTful principles simplifies integration and supports various client implementations.

Sample Server Response: An example of a server response in JSON format is as follows:

JSON object of ResponseModel:

```
{"code":0,"body":{"\"userID\":1,\"username\":\"admin\",\"password\":\"password\",\"fullNam  
e\":\"Adam Smith\",\"isManager\":false}"}
```

Message from client:

```
{"code":33,"body":{"\"bookID\":1234,\"bookName\":\"yahoo\",\"authorName\":\"yy  
zee\",\"quantity\":20.0,\"status\":\"A\"}}
```

JSON object of ResponseModel: {"code":0,"body":"New book saved successfully."}

Message from client: {"code":3,"body":"1234"}

JSON object of ResponseModel:

```
{"code":0,"body":{"\"bookID\":1234,\"bookName\":\"yahoo\",\"authorName\":\"yy  
zee\",\"quantity\":20.0,\"status\":\"A\"}}
```

Message from client: {"code":3,"body":"1"}

JSON object of ResponseModel: {"code":0,"body":{"\"bookID\":1,\"bookName\":\"Engineering
Mechanics\",\"authorName\":\"Varun S.\",\"quantity\":927.0,\"status\":\"A\"}}

Message from client: {"code":555,"body":"1"}

JSON object of ResponseModel: {"code":0,"body":{"\"studentID\":1,\"studentName\":\"Alice
Johnson\",\"emailID\":\"alice@email.com\",\"studentNumber\":\"1234567890\"}}

Message from client:

```
{"code":22,"body":{"\"orderID\":0,\"studentID\":1,\"student\":{\"\"studentID\":1,\"studentName  
\":\"Alice  
Johnson\",\"emailID\":\"alice@email.com\",\"studentNumber\":\"1234567890\"},\"lines\":[{\\"  
orderID\":0,\"bookID\":1,\"quantity\":1.0,\"bookName\":\"Engineering Mechanics\"}]}}
```

JSON object of ResponseModel: {"code":0,"body":"Order and order lines saved successfully."}

Message from client: {"code":3,"body":"1"}

JSON object of ResponseModel: {"code":0,"body":{"\"bookID\":1,\"bookName\":\"Engineering
Mechanics\",\"authorName\":\"Varun S.\",\"quantity\":927.0,\"status\":\"A\"}}

Message from client: {"code":33,"body":{"\"bookID\":1,\"bookName\":\"Engineering
Mechanics\",\"authorName\":\"Varun S.\",\"quantity\":926.0,\"status\":\"A\"}}

JSON object of ResponseModel: {"code":0,"body":"Book updated successfully."}

Message from client: {"code":3,"body":"1"}

JSON object of ResponseModel: {"code":0,"body":"{\"bookID\":1,\"bookName\":\"Engineering Mechanics\",\"authorName\":\"Varun S.\",\"quantity\":926.0,\"status\":\"A\"}"}

Message from client: {"code":7,"body":"{\"receiptNumber\":0,\"orderId\":0,\"dateTime\":\"29-10-2023 23:08:47\",\"student\":\"Student ID: 1 Name: Alice Johnson Email ID: alice@email.com Number: 1234567890\",\"books\":\"Engineering Mechanics x 1.0\"}"}

JSON object of ResponseModel: {"code":0,"body":"Receipt information saved successfully."}
Message from client: {"code":8}

JSON object of ResponseModel: {"code":0,"body":"100"}

This response includes a response code (0 for success) and a JSON body containing specific data related to the query.

This architectural design allows for a clear separation of concerns, making your library management system modular and scalable. It uses a robust communication protocol and data format to ensure reliable data exchange between the client and server components.

Ques 6) Manual for installation and usage with information on necessary libraries, frameworks, and running guidelines.

Answer 6)

INSTALLATION MANUAL:

System Requirements:

1. IntelliJ Idea – Java and Kotlin IDE
2. Java Development Kit (JDK) 8 or later
3. Node.js
4. Redis Server
5. MongoDB Server

JARs Required: Install all the JARs present in the Jars folder in a zip file. These are required to run the program.



JARs' Function:

com.fasterxml.jackson.databind: Jackson Databind, a general-purpose data-binding package for JSON and other data formats.

gson-2.9.0: Gson, a library for JSON serialization and deserialization in Java.

jackson-all-1.9.0: Jackson All-in-One, a bundle including various Jackson libraries for JSON processing.

jedis-5.0.2: Jedis, a Java client library for interacting with Redis, a key-value store.

mongodb-jdbc-2.0.3-all: MongoDB JDBC Driver, a JDBC-compliant driver for connecting to MongoDB.

These libraries provide essential functionality for data storage, serialization, and communication in the context of MongoDB, Redis, and general-purpose Java development. They are included as dependencies to enable seamless integration and smooth functioning of the code.

Installation Steps:

- Installation of any JAVA IDE is also very important. We have used IntelliJ IDE. Any other IDE can also be used for e.g. Eclipse.
- Java Development Kit (JDK): Ensure that you have Java installed. You can download it from Oracle JDK or use OpenJDK.
- Redis Server: Start the Redis server by running the classes UserServer and BookServer. The access of this is public so it is available.
- MongoDB Server: Start MongoDB server by running the classes OrderBookServer. The access of this is public so it is available.
- Node.js server: Start server.js by running server.js file in Terminal.
- Change the active directory to the path wherein node server file is present.
 - Write below command in Terminal:
 - cd <path_to_node_server_directory>
 - node server.js

Alternatively, you may write the below lines in Terminal to start server.js:

```
cd src  
cd LMS_Website  
node server.js
```

Terminal Screenshot:



```
Terminal Local × + ▾  
PS C:\Users\Varun Sahni\IdeaProjects\StoreApp> cd src  
PS C:\Users\Varun Sahni\IdeaProjects\StoreApp\src> cd LMS_Website  
PS C:\Users\Varun Sahni\IdeaProjects\StoreApp\src\LMS_Website> node server.js  
Server is running on port 20000
```

Running the Clients:

- 1) Run User server at port 5057.
- 2) Run Book server at port 5058.
- 3) Run OrderBook server at port 5059.
 - a. Desktop Client (Client 1): Run the “Application” class.
 - b. Web Browser Client (Client 2) : Type this URL in web browser:
<http://localhost:20000/>. This will lead to library management system website.