AN INTERNSHIP REPORT ON

JAVA&FSD PROGRAMMING

A Report Submitted to

Blackbuck Engineers Pvt. Ltd

Submitted by

STUDENT NAME (BBID)

- POKURU GREESHMANTH
ROLL NO.:20731A3542
BBID:BBPBV045

- PASUVULA AKASH
ROLL NO.:20731A3538
BBID:BBPBV044

- KORISAPTI VISHNU VARDHAN REDDY
ROLL NO.:20731A3520
BBID:BBPBV040
- GADDAM RAMANA REDDY
ROLL NO.:20731A3515
BBID:BBPBV038

Blackbuck Engineers Pvt. Ltd

Road no 36, Jubilee Hills,Hyderabad

CERTIFICATE

TABLE OF CONTENTS

‘

ACKNOWLEDGEMENT

We would like to acknowledge all those without whom this project would not have been successful. Firstly, we would like to thank our Internet and Java & FSD Programmer
Mr. Somanapalli RAMAKRISHNA who guided us throughout the project and gave his immense support. he made us understand how to complete this project and without his presence, this project wouldnot have been complete. We also got to know a lot about parallelization and its benefits. This project has been a source to learn and bring our theoretical knowledge to the real-life world. Once again, thanks to everyone for making this project successful.

Place :Kavali

Date :28/09/2023

ABSTRACT

Library management system is a project which aims in developing a computerized system to maintain all the daily work of library .This project has many features which are generally not available in normal library management systems like facility of user login and a facility of teachers login .It also has a facility of admin login through which the admin can monitor the whole system .It also has facility of an online notice board where teachers can student can put up information about workshops or seminars being held in our colleges or nearby colleges and librarian after proper verification from the concerned institution organizing the seminar can add it to the notice board . It has also a facility where student after logging in their accounts can see list of books issued and its issue date and return date and also the students can request the librarian to add new books by filling the book request form. The librarian after logging into his account ie admin account can generate various reports such as student report , issue report, teacher report and book report

1. INTRODUCTION

1.1. SYSTEM OVERVIEW

Library Management System is an application which refers to library systems which are generally small or medium in size. It is used by librarian to manage the library using a computerized system where he/she can record various transactions like issue of books, return of books, addition of new books, addition of new students etc. Books and student maintenance modules are also included in this system which would keep track of the students using the library and also a detailed description about the books a library contains. With this computerized system there will be no loss of book record or member record which generally happens when a non computerized system is used.

In addition, report module is also included in Library Management System. If user's position is admin, the user is able to generate different kinds of reports like lists of students registered, list of books, issue and return reports .All these modules are able to help librarian to manage the library with more convenience and in a more efficient way as compared to library systems which are not computerized

1.2. OBJECTIVE

This project consists of a Efficient data management which streamline the collection, organization, and management of Books data, ensuring a centralized and efficient system for handling the extensive information. To develop a system that can replace the manual library managing system. Develop a database which stores books details. Create an easy to understand user friendly environment.

1.3. PROBLEM IDENTIFICATION

These are some of the problems faced by a person who wants to start a start-up!

1. Efficient Data Management
2. Accurate Books Record Keeping
3. Data Security and Privacy Compliance
4. System Downtime and Performance Issues
5. Lack of Mentorship

1.4. APPLICATIONS

This web application can be used by any College and School to maintain the Library information.

1.Add or Delete Books
2.update the Books
3.find books by id
4.delete book by id
5.Borrow Book
6.Return Book

1.5. LITERATURE SURVEY

1. **Introduction to Library Catlog:**

   Begin with a general introduction to Library Catlog management, discussing its importance in educational institutions, challenges faced, and the need for efficient systems like SRS.

2. **Overview of Library Systems:**

   Review literature discussing various information systems utilized in educational/Library settings, focusing on their functions, features, and benefits for academic administration.

3. **Data Security and Privacy:**

   Examine research on the importance of data security and privacy within SRS, discussing methods to ensure compliance with relevant data protection regulations and best practices for safeguarding Books data.

4. **Integration and Interoperability in SRS:**

   Review research focusing on integrating SRS with existing educational/Library systems, emphasizing interoperability, data sharing, and seamless communication between different platforms.

1.6. LIMITATIONS

1. Cost and Resource Intensiveness.
2. User Training and Adoption Challenges.
3. Customization and Adaptability Challenges
4. Dependency on Technical Infrastructure.

2. SYSTEM ANALYSIS

2.1. EXISTING SYSTEM

1. Manages all library information across ledgers and files.
2. All data related to books,librarians is done manually conventional manner.
3. Not effective and lead to mismangament of data.

Library data	**University /School/Library Database**
Database for Books record's	**MYSQL, Oracle**

2.1.1. PROPOSED SYSTEM

1.computerized
2.Database Maintenance
3.Reports
4.Transaction Entry

3. REQUIREMENT SPECIFICATION

3.1. HARDWARE REQUIREMENTS

1. Laptop 4 GHz minimum, multi-core processor
2. Memory (RAM) 4GB, preferably higher, and commensurate with concurrent usage
3. Hard disk space 1TB or SDD having (256GB and above)

3.2. SOFTWARE REQUIREMENTS

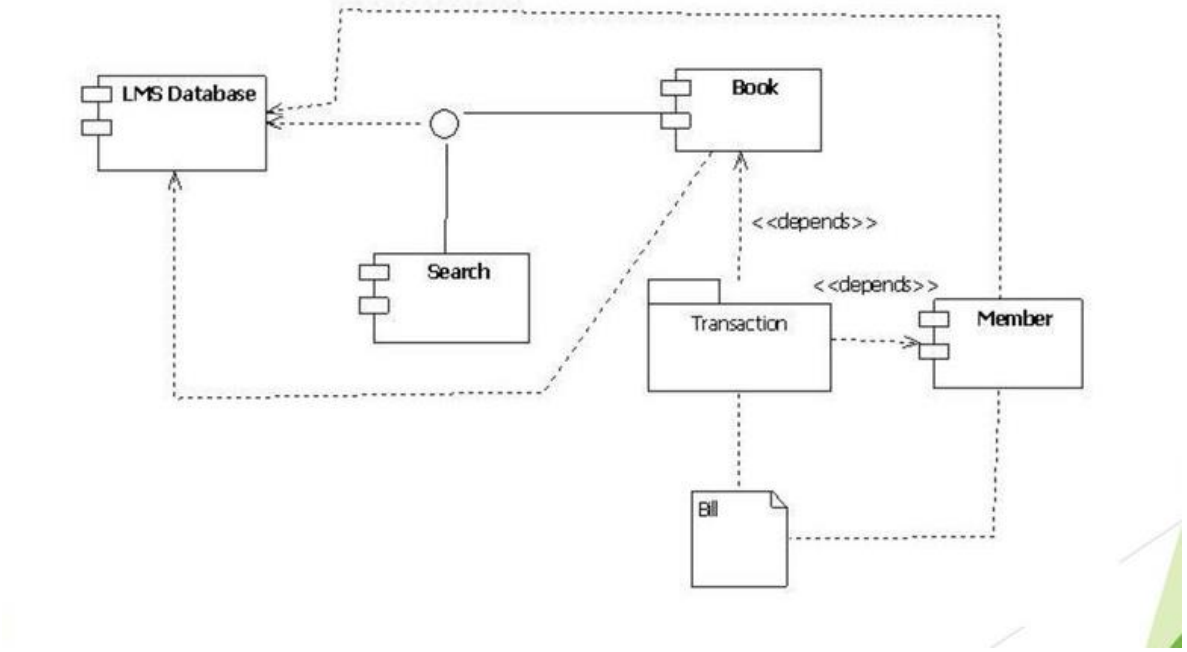
1. Windows 2016
2. Eclipse
3. MYSQL
4. Apache tomcat web server
5. Postman Api

3.3. COMPONENTS USED

1. Postman
2. Google Chrome
3. Eclipse
4. MYSQL

4. ARCHITECTURE DESIGN SPECIFICATION

4.1. SYSTEM ARCHITECTURE



Student side

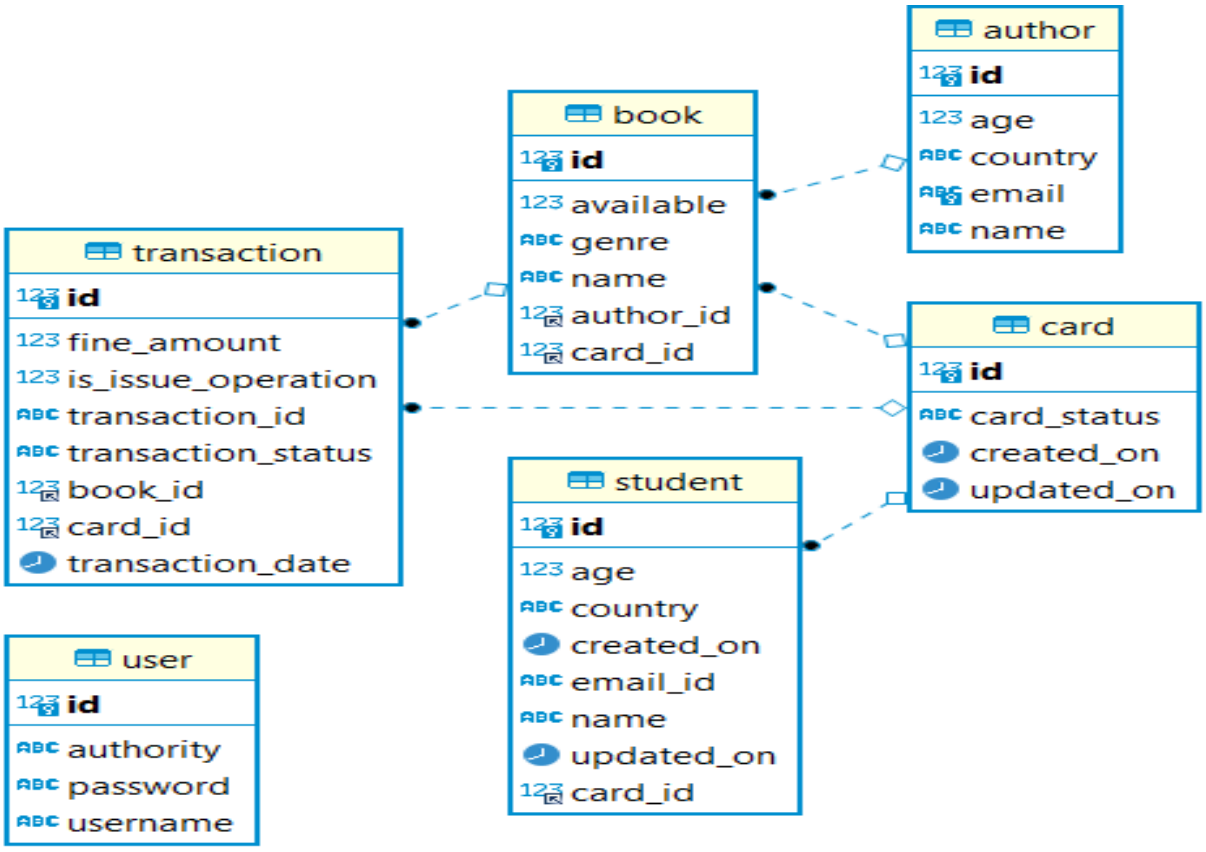
I. Users interact with the system through the user interface (UI), accessing various functionalities and providing input or queries related to books records.

II. The UI triggers requests that are processed by the business logic layer, which validates, processes, and applies business rules to the requests.

III. The business logic layer interacts with the data access layer to retrieve or update books data from the database.

IV. Integration components verify requests, perform security checks, and integrate with other systems or external APIs as needed.

4.2. DETAILED DESIGN



Student Side

I. The student can access the books data through their library website.

II. They have the access to check the updated books and journals and research papers.

III. If there is any issue regarding the available of books and any items in the library they can approach the concern management to rectify the issue.

4.3. COMPONENTS USED

I. Eclipse

II. MySQL/Oracle

III. Postman

IV. HTML

V. Apache (TOMCAT) Web server

VI. Spring initializer

4.4. DATABASE DESIGN

I. For our project we have to create one table that consists of all the information about the student.,

II. Create student table that consists of student id followed by student name, student phone number, student email-id ,student marks in all subjects and overall grade.

III. Open postman and there we have to give commands like GET,POST,PUT,UPDATE,DELLETE,CREATE

IV. By using the following commands we have to modify the data accordingly.

The following are images of our project.

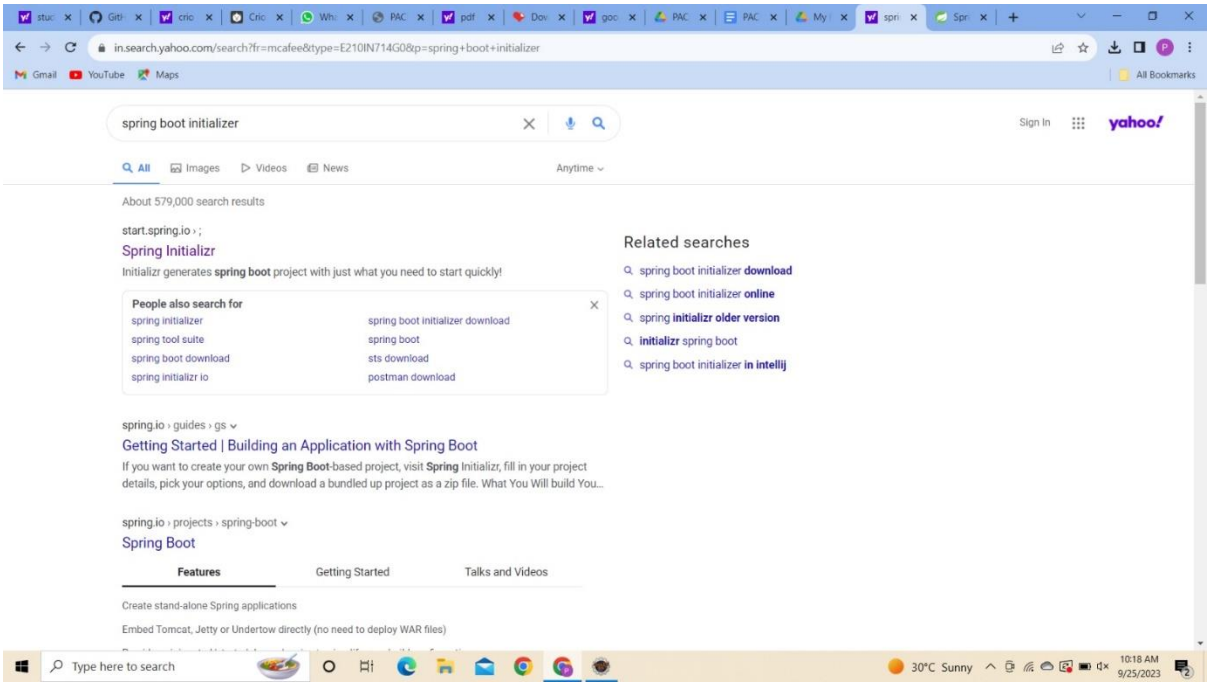


Fig:1.0 open Spring boot initializer

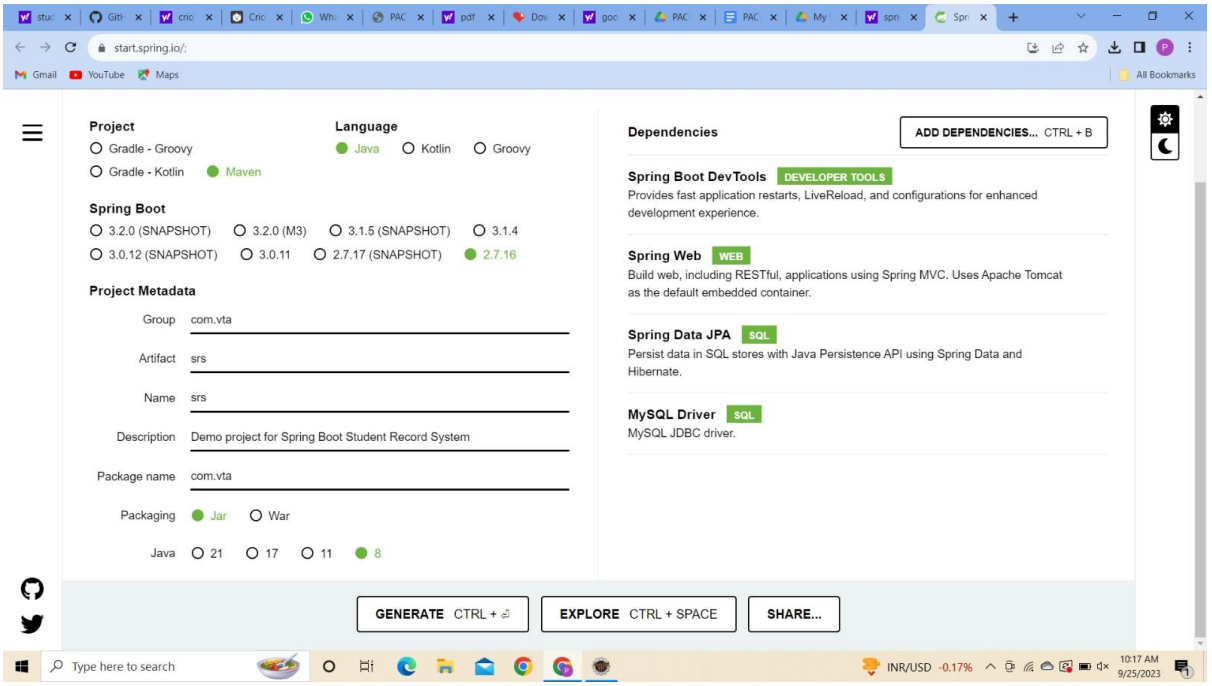


Fig:2.0 select the above options and generate JAR file

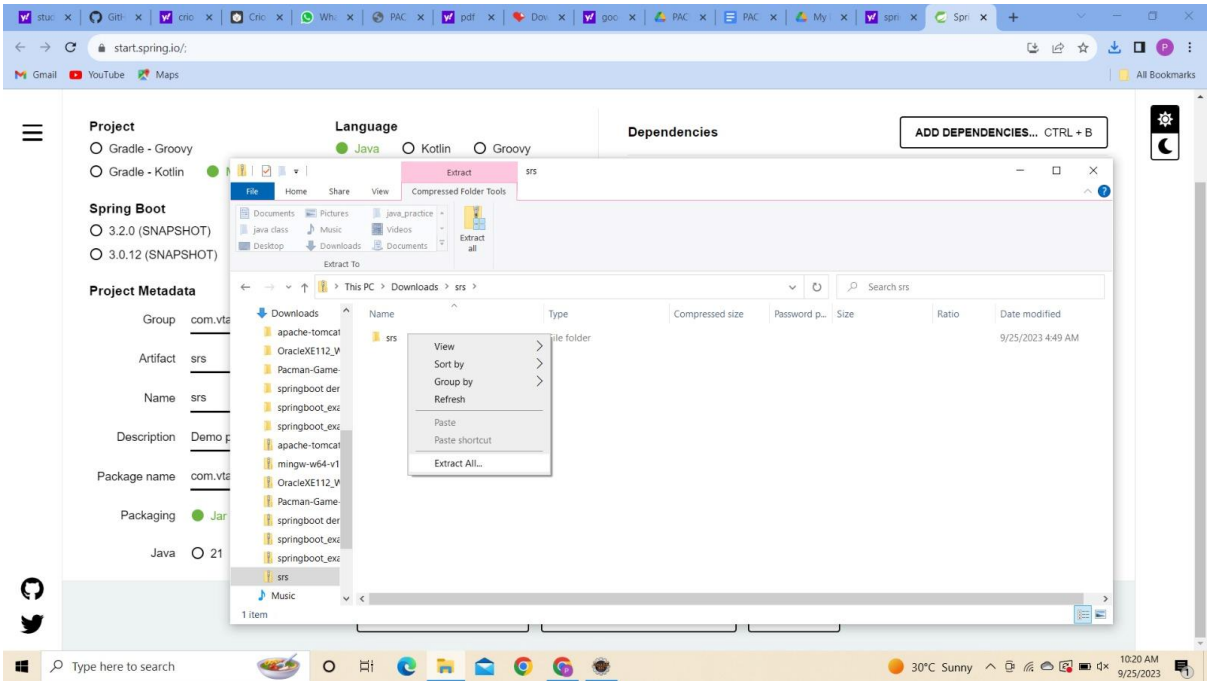


Fig:3.0 Extract the JAR files into a destination file

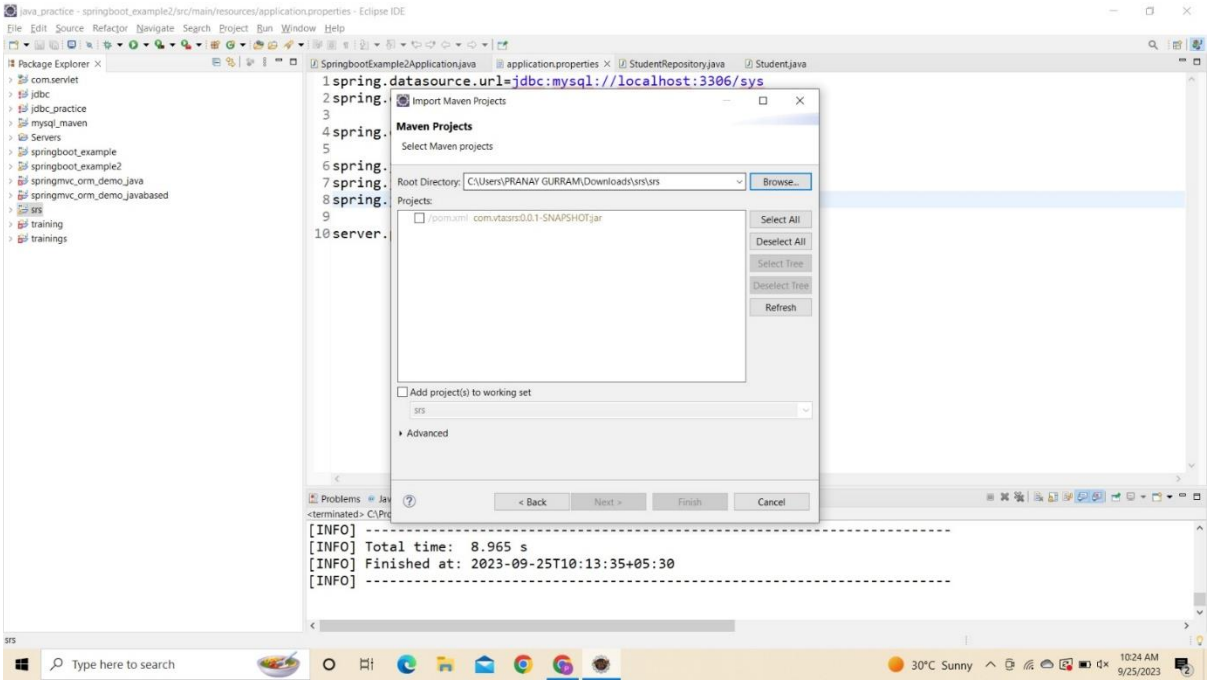


Fig:4.0 Create a new Existing maven project and select the root directory

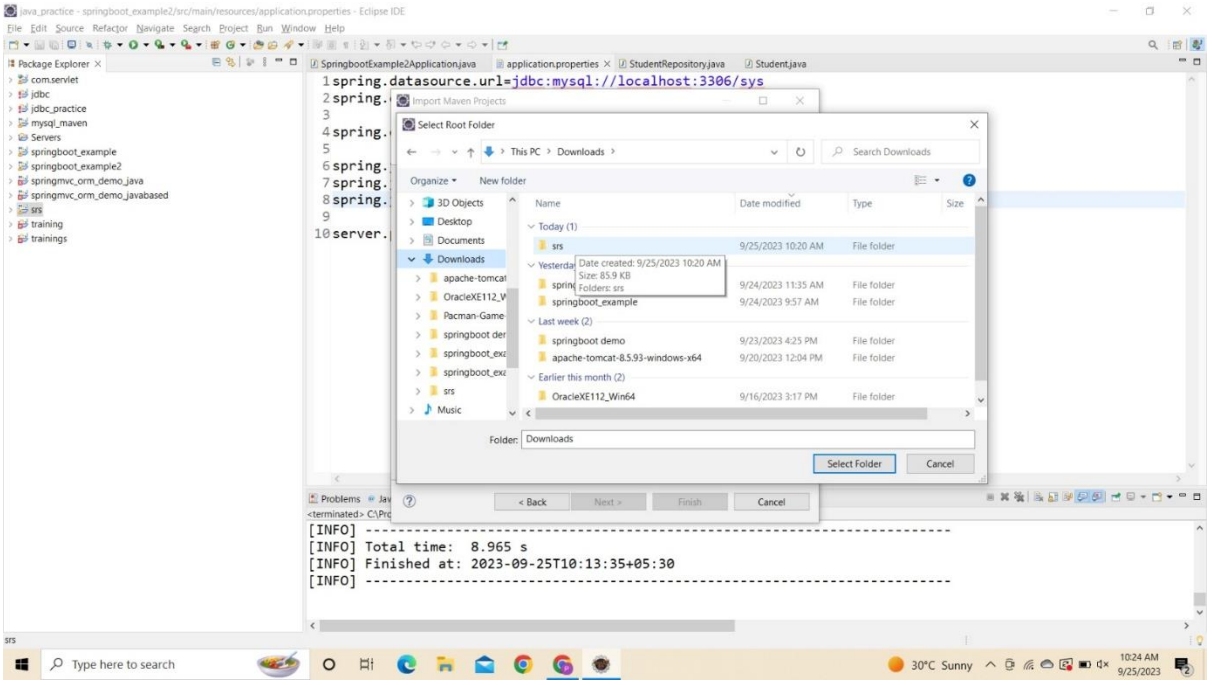


Fig:5.0 select the extracted file here

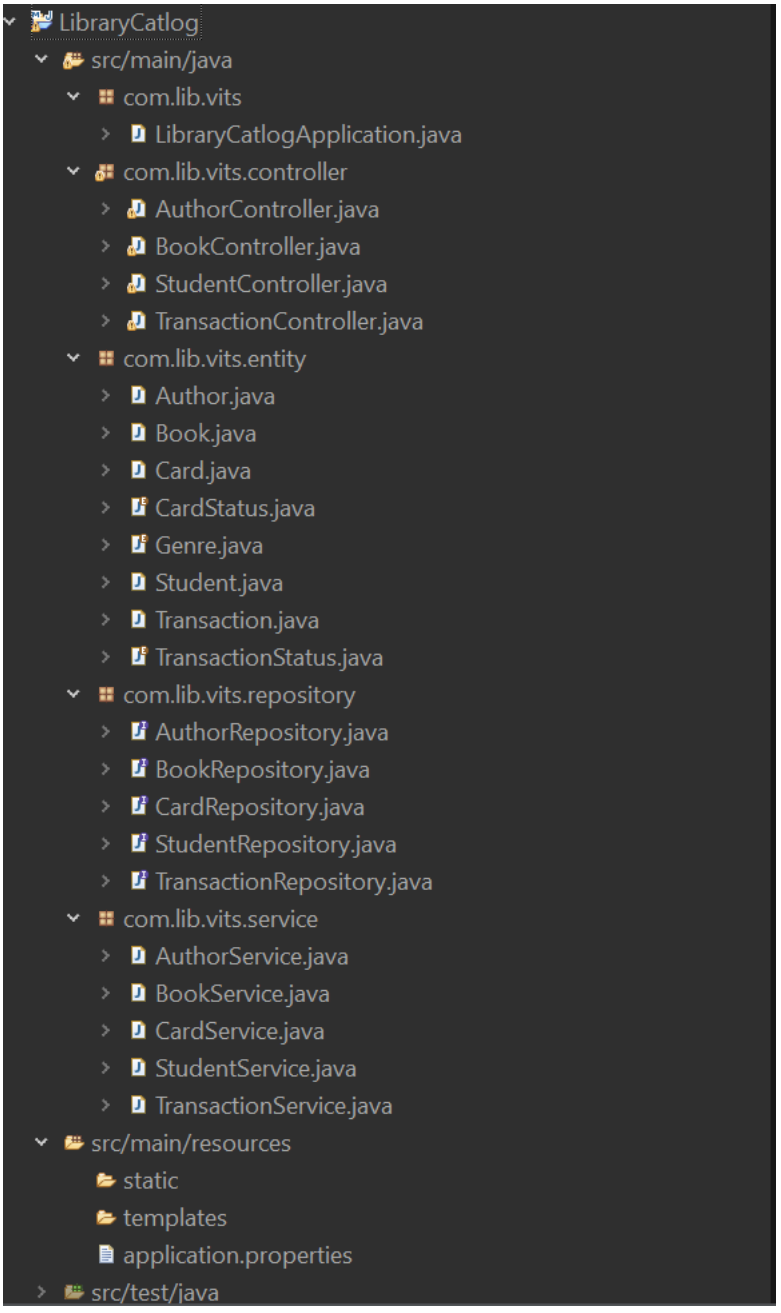


Fig:6.0 A project is created like this

- Copy the code from the below source code and run the code as java application.

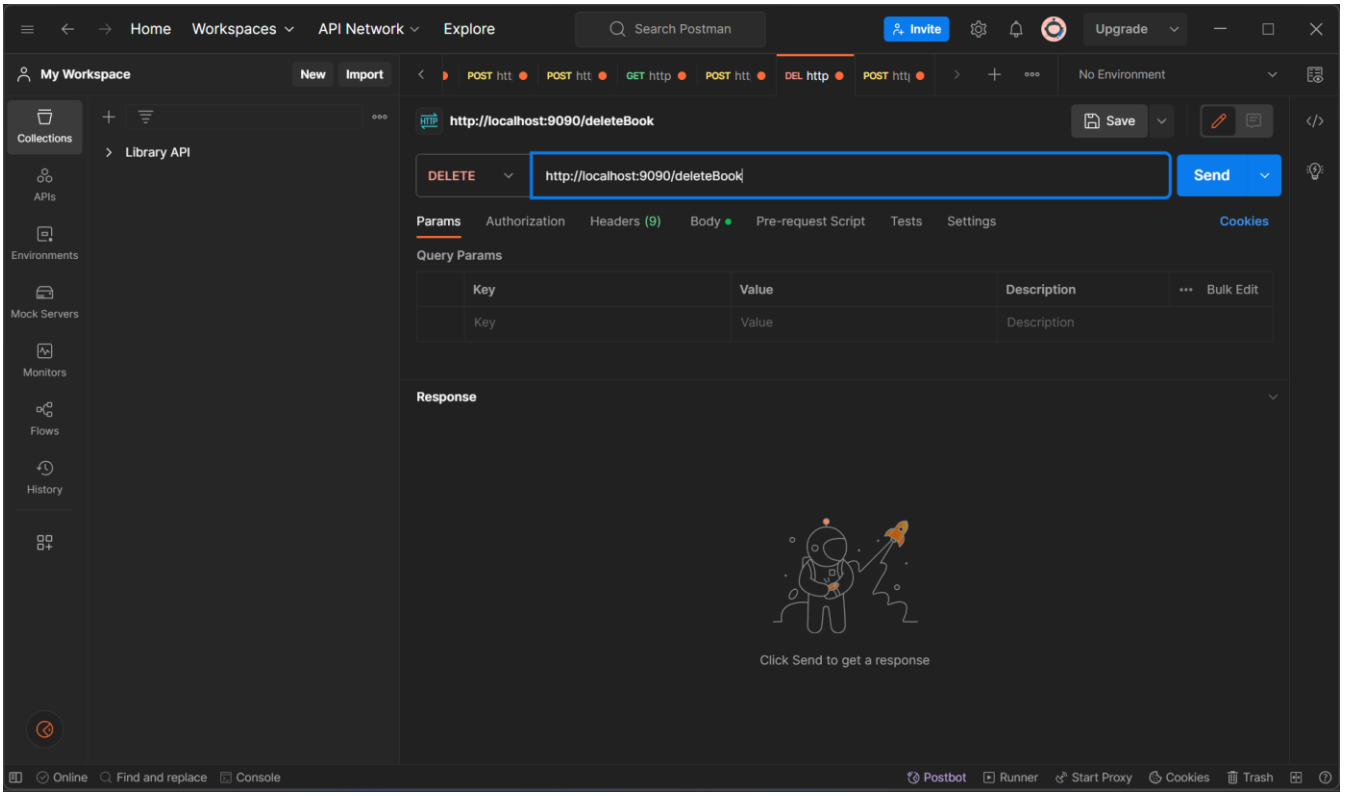


Fig:7.0 This is the postman application view of our project

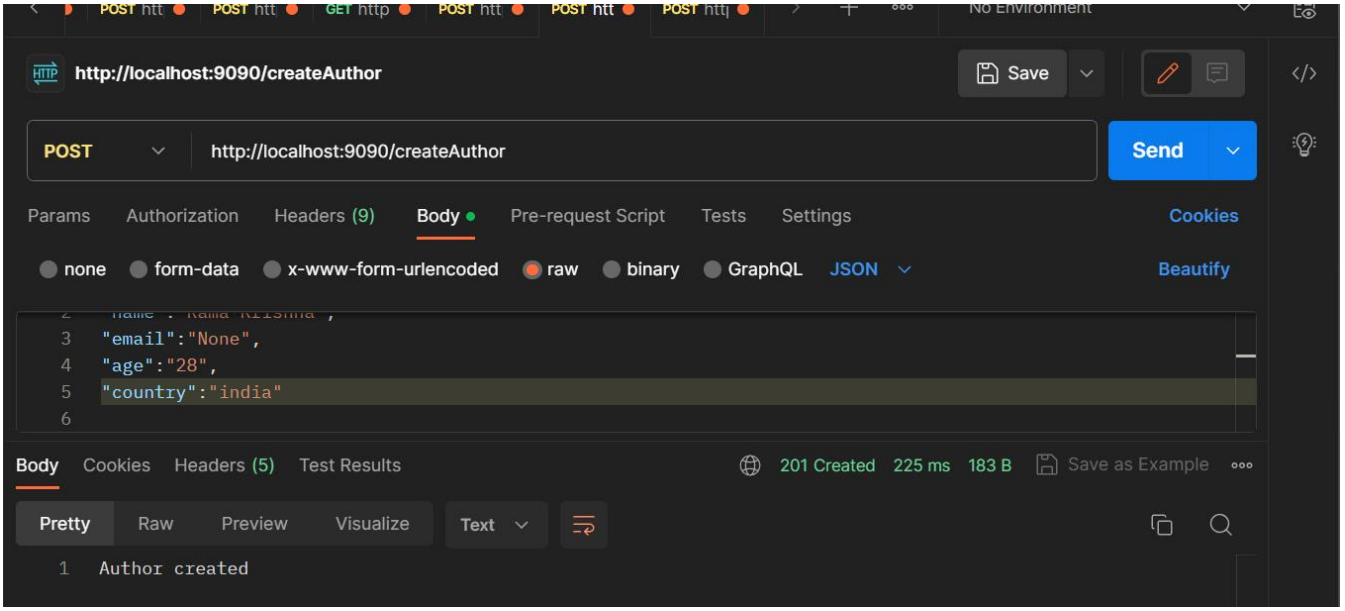


Fig: insert author to the author table.

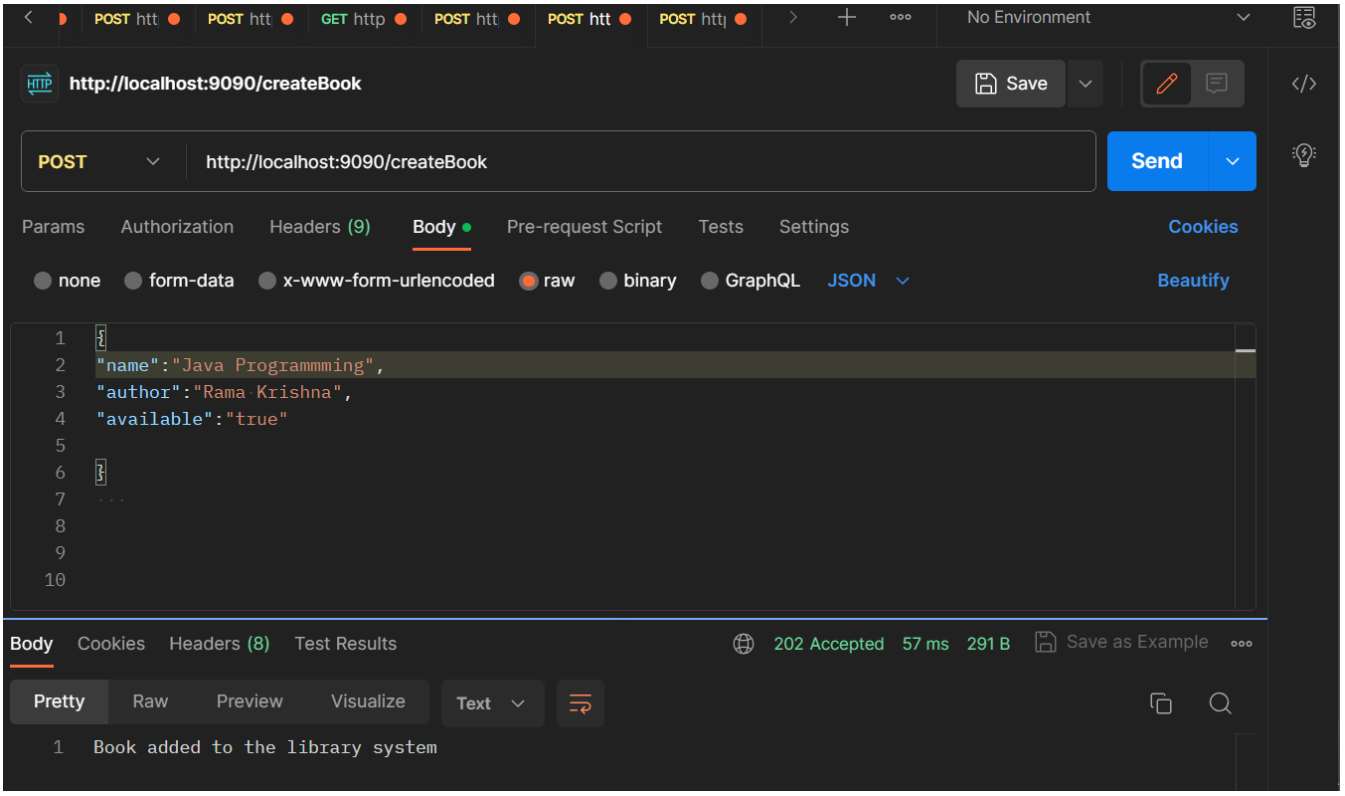


Fig:8.0 Create table and inserting book data to the book table.

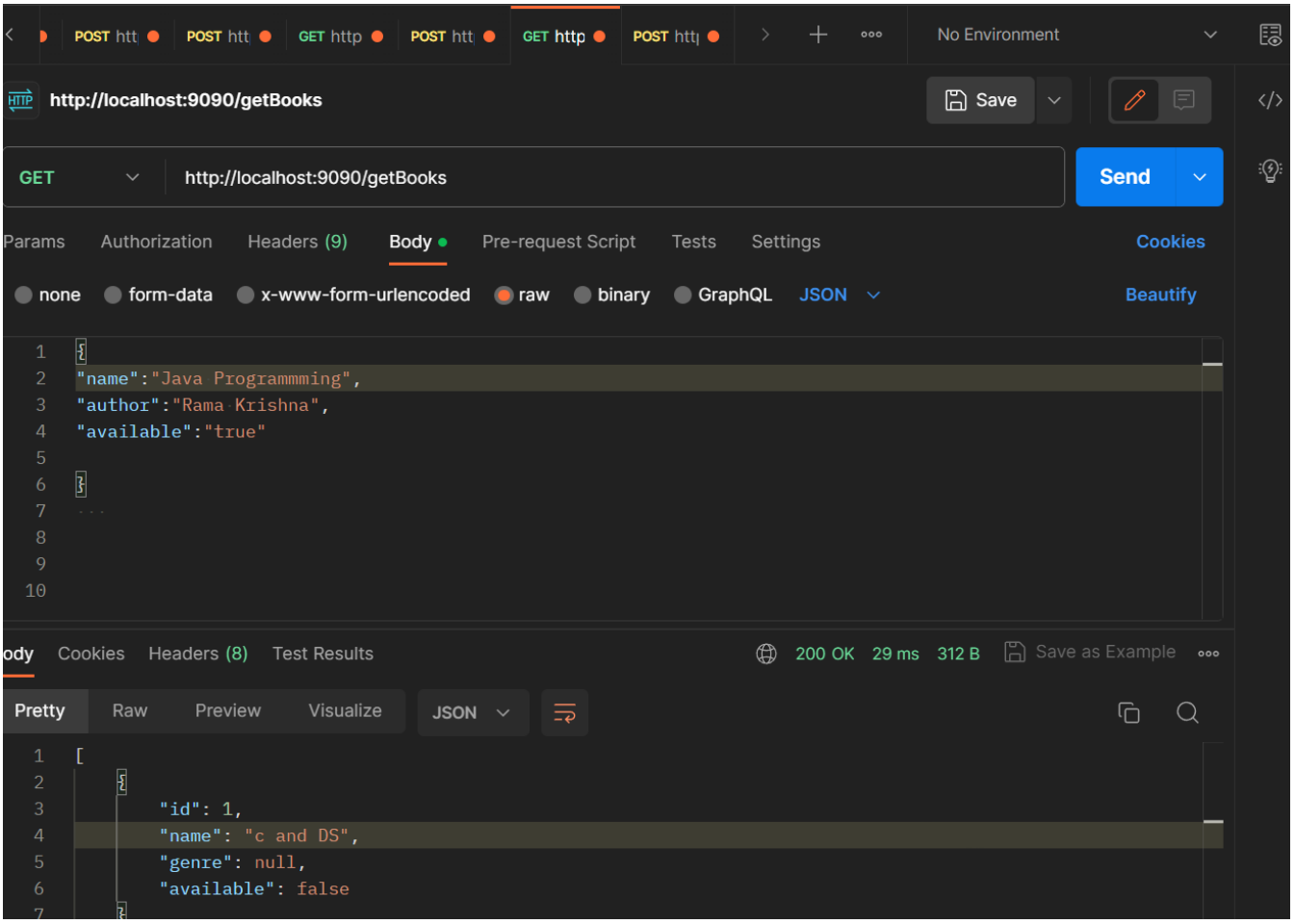


Fig:9.0 to display all the books in the table.

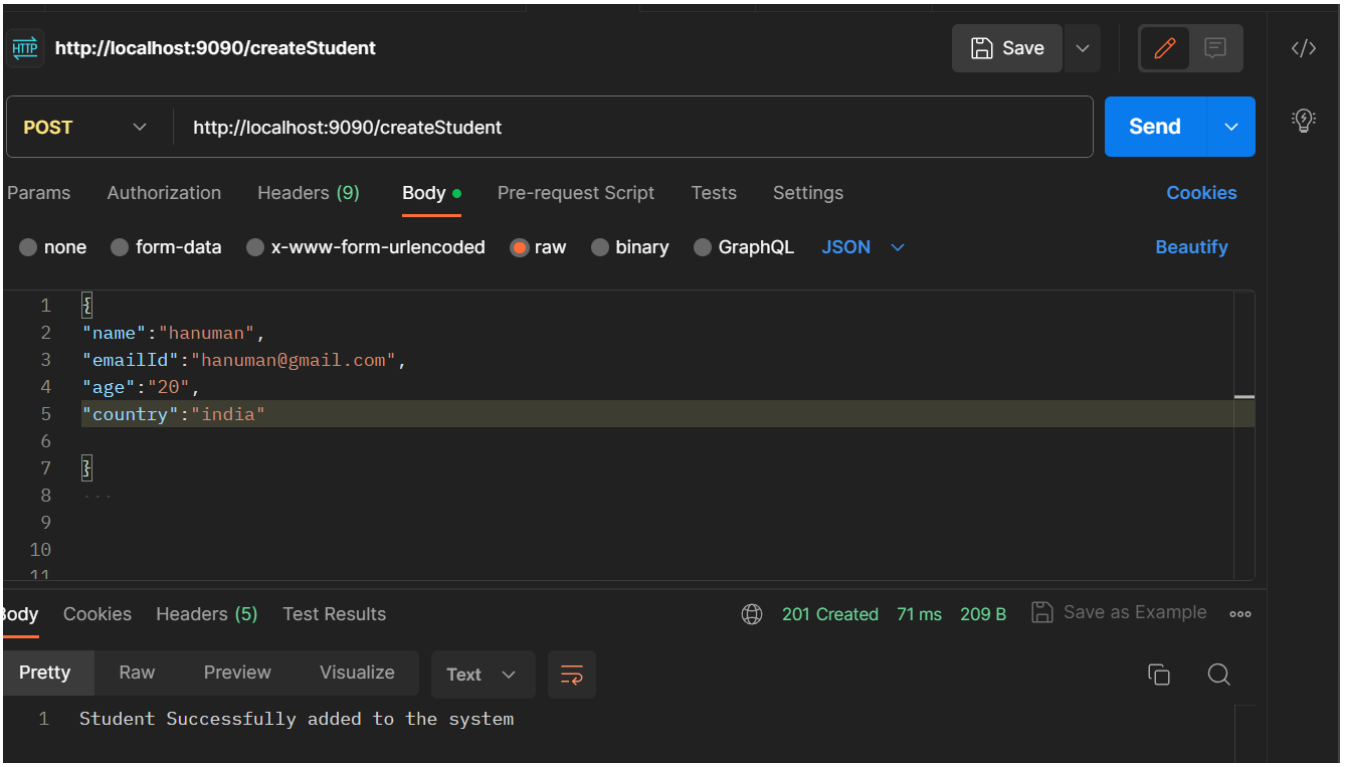


Fig:10.0 Adding student data to the student database

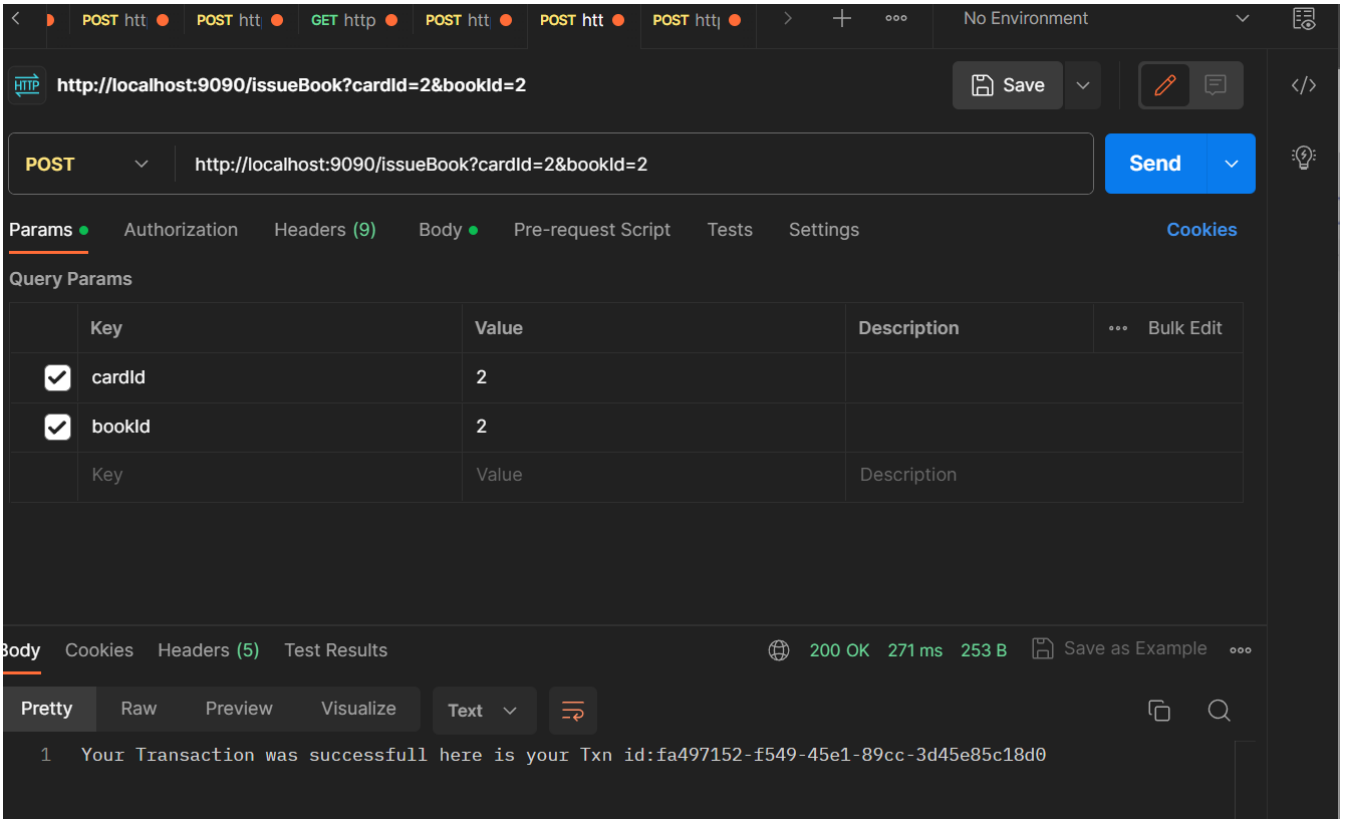
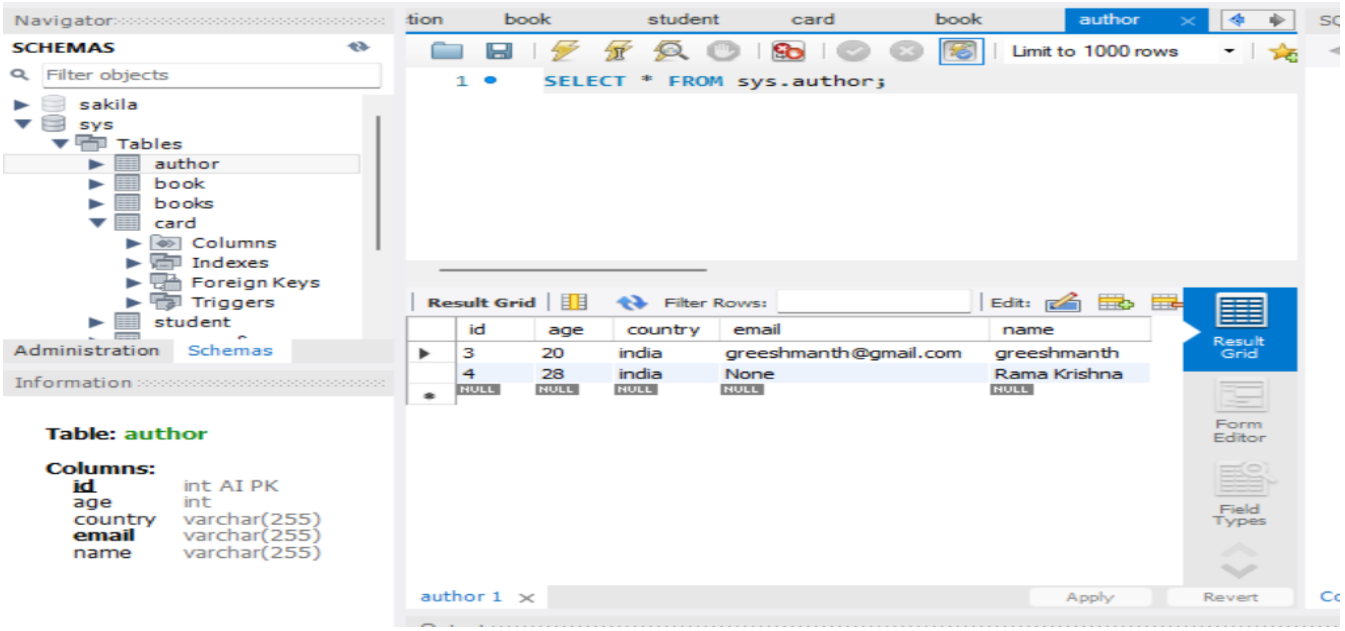


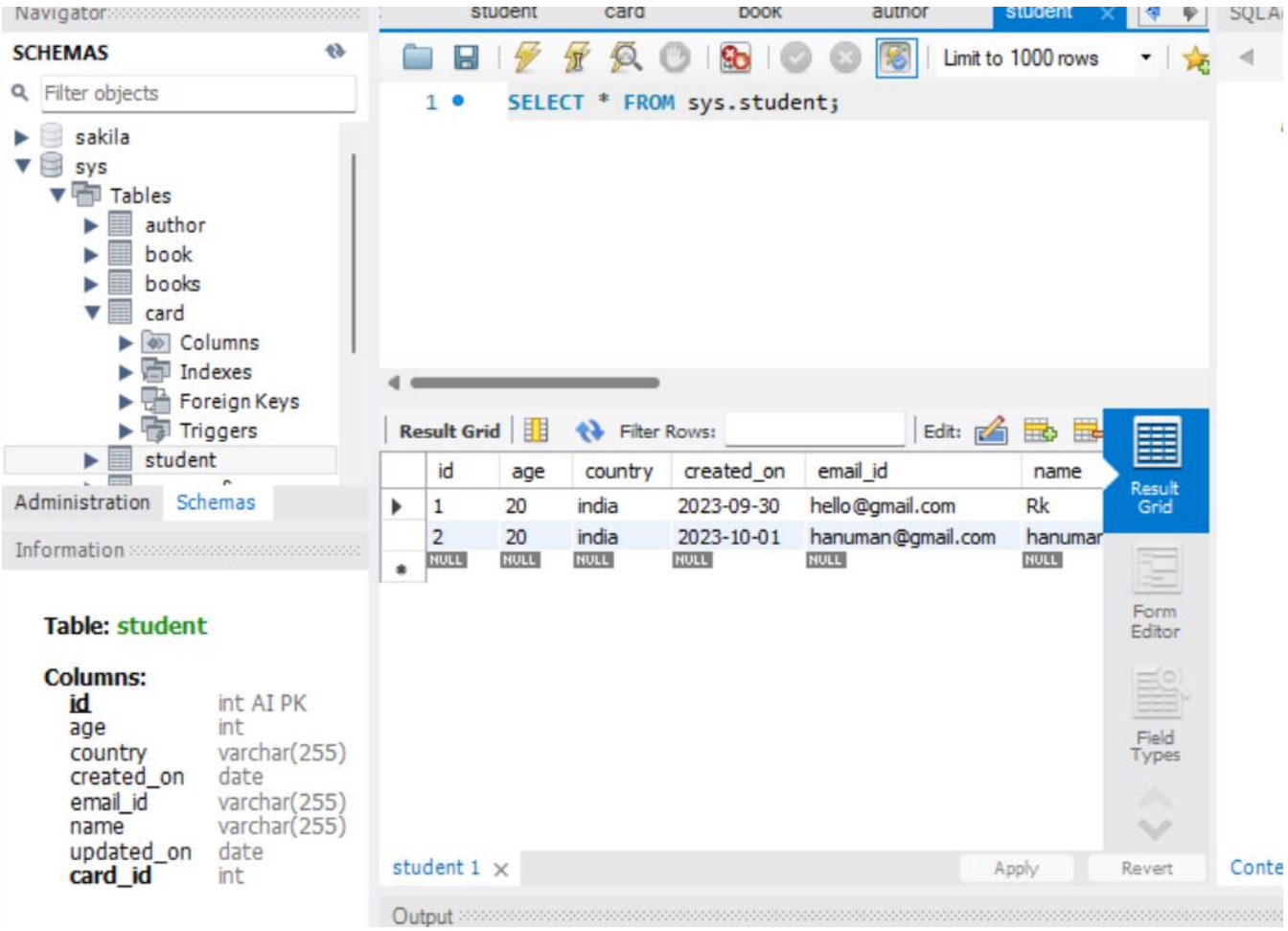
Fig: Transaction of books by book id and card id.

Mysql database :

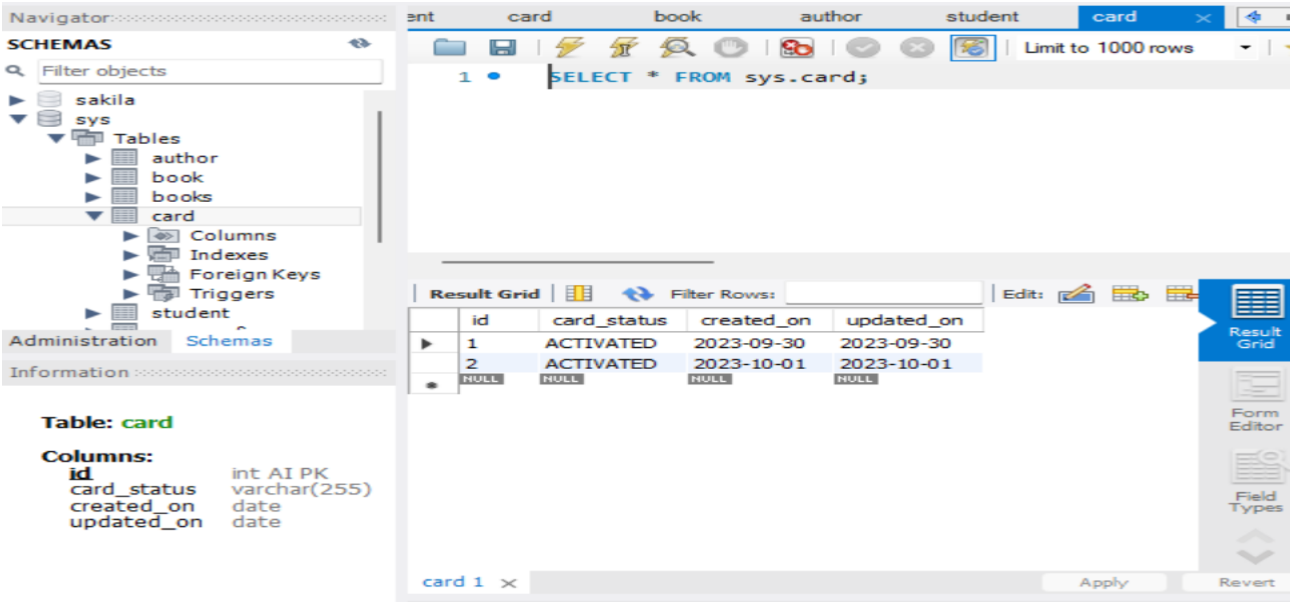
Author table



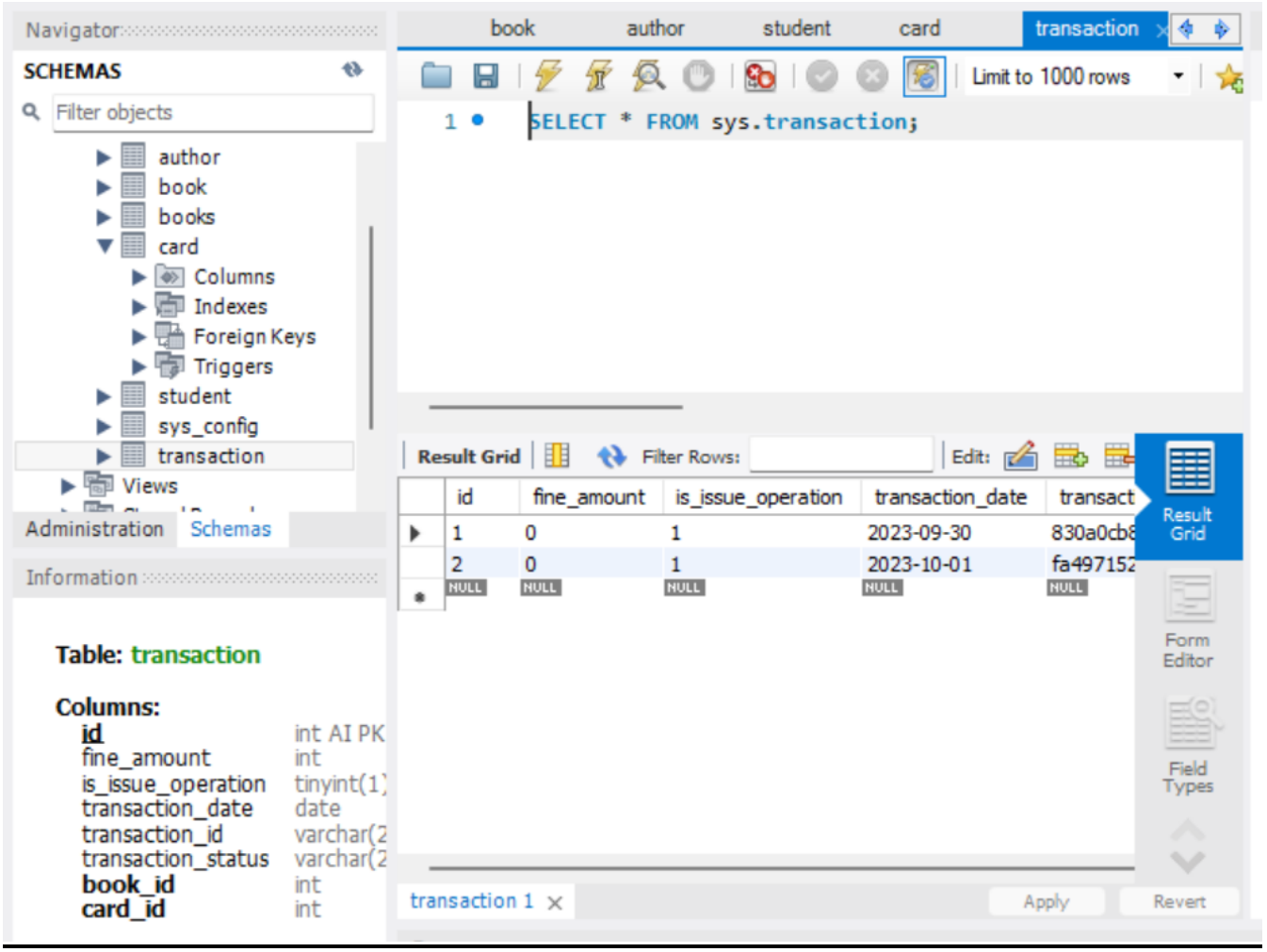
StudentTable



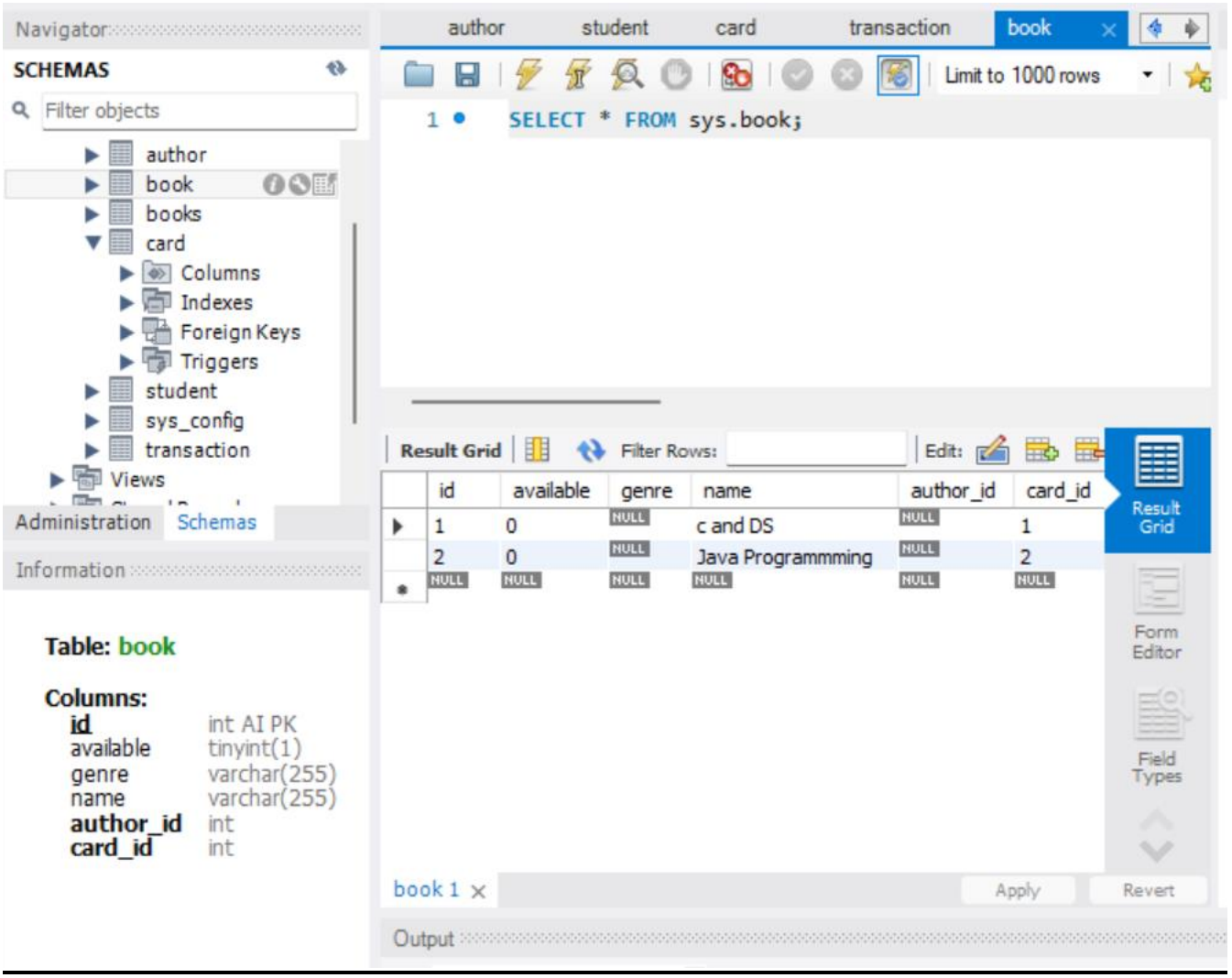
Card Table:



Transaction Table:



BookTable:


Navigator
SCHEMAS
Filter objects
author
book
books
card
Columns
Indexes
Foreign Keys
Triggers
student
sys_config
transaction
Views
Administration
Schemas
Information
Table: book
Columns:
id int AI PK
available tinyint(1)
genre varchar(255)
name varchar(255)
author_id int
card_id int
author
student
card
transaction
book
Limit to 1000 rows
1 SELECT * FROM sys.book;
Result Grid
Filter Rows:
Edit:
id available genre name author_id card_id
1 0 NULL c and DS NULL 1
2 0 NULL Java Programmming NULL 2
NULL NULL NULL NULL NULL NULL
Result Grid
Form Editor
Field Types
book 1
Apply
Revert
Output

4.2 CONCLUSION AND FUTURE ENHANCEMENTS

- The respondents of the study encountered a high degree of difficulty on the existing system, which is relevant to the researchers of the study. This indicates that the respondents experienced problems in securing students records, searching and retrieving student grades, the use of manual procedures by the Registrar's Office in keeping the students record are not secured from alteration or loss, and the students encountered problems in requesting grades at the registrar's office and as well as, the faculty in submitting error-free grade sheets.
- Important features should be included in the development of the system such as login.logout, grade sheets, reports, database maintenance, and help assistant.
- Majority of the respondents prefer to change the current registrar system for accurate, fast, and accessible for the students, faculty, department chairman, and the registrar itself.
- The use of Automated Student Record System of the Northern Negros State College of Science and Technology was readily accepted by the respondents
- The objective of the study had been achieved, that is to develop an automated student record system for college level that is fast, accurate, and accessible..

5.1 APPENDIX 1 - SAMPLE SOURCE CODE

Spring boot Run Code:

```
package com.lib.vits;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.lib.vits.repository.AuthorRepository;
import com.lib.vits.repository.BookRepository;
import com.lib.vits.repository.CardRepository;
import com.lib.vits.repository.StudentRepository;

@SpringBootApplication
public class LibraryCatlogApplication implements CommandLineRunner {

	@Autowired
	StudentRepository studentRepository;

	@Autowired
	CardRepository cardRepository;
```

```
	@Autowired
	AuthorRepository authorRepository;

	@Autowired
	BookRepository bookRepository;

	public static void main(String[] args) {
		SpringApplication.run(LibraryCatlogApplication.class, args);
		System.out.println("completed");
	}

	@Override
	public void run(String... args) throws Exception {
		// TODO Auto-generated method stub

	}

}
```

Controller Class:

AuthorController.java

```
	package com.lib.vits.controller;

	import org.springframework.beans.factory.annotation.Autowired;
	import org.springframework.http.HttpStatus;
	import org.springframework.http.ResponseEntity;
	import org.springframework.web.bind.annotation.DeleteMapping;
	import org.springframework.web.bind.annotation.PostMapping;
	import org.springframework.web.bind.annotation.PutMapping;
	import org.springframework.web.bind.annotation.RequestBody;
	import org.springframework.web.bind.annotation.RequestParam;
	import org.springframework.web.bind.annotation.RestController;

	import com.lib.vits.entity.Author;
	import com.lib.vits.service.AuthorService;

	@RestController
	public class AuthorController {

		@Autowired
		AuthorService authorService;

		@PostMapping("/createAuthor")
		public ResponseEntity createAuthor(@RequestBody Author author) {
		 authorService.createAuthor(author);
		 return new ResponseEntity("Author created", HttpStatus.CREATED);
		}

		@PutMapping("/updateAuthor")
		public ResponseEntity updateAuthor(@RequestBody Author author) {
```

```
        authorService.updateAuthor(author);
        return new ResponseEntity("Auhtor upadted!!", HttpStatus.ACCEPTED);

    }

    @DeleteMapping("/deleteAuthor")
    public ResponseEntity deleteAuthor(@RequestParam("id") int id) {
        authorService.deleteAuthor(id);
        return new ResponseEntity("Author deleted!!", HttpStatus.ACCEPTED);

    }

}
```

BookController.java:

```
package com.lib.vits.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.lib.vits.entity.Book;
import com.lib.vits.service.BookService;

@CrossOrigin(origins = { "http://localhost:3000" })
@RestController
public class BookController {

    @Autowired
    BookService bookService;

    @PostMapping("/createBook")
    public ResponseEntity createBook(@RequestBody Book book) {

        bookService.createBook(book);
        return new ResponseEntity("Book added to the library system",
        HttpStatus.ACCEPTED);

    }

    @GetMapping("/getBooks")
```

```
        public ResponseEntity getBooks(@RequestParam(value = "genre", required = false)
        String genre,
                @RequestParam(value = "available", required = false, defaultValue = "false")
        boolean available,
                @RequestParam(value = "author", required = false) String author) {

         List<Book> bookList = bookService.getBooks(genre, available, author);
         return new ResponseEntity(bookList, HttpStatus.OK);

        }

        @DeleteMapping("/deleteBook")
        public ResponseEntity deleteBook(@RequestParam("id") int id) {
         bookService.deleteBook(id);
         return new ResponseEntity("book deleted!!", HttpStatus.ACCEPTED);

        }

        @PutMapping("/updateBook")
        public ResponseEntity updateBook(@RequestBody Book book) {

         bookService.createBook(book);
         return new ResponseEntity("Book updated to the library system",
        HttpStatus.ACCEPTED);

        }

}
```

StudentController.java:

```
    package com.lib.vits.controller;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.http.HttpStatus;
    import org.springframework.http.ResponseEntity;
    import org.springframework.web.bind.annotation.DeleteMapping;
    import org.springframework.web.bind.annotation.PostMapping;
    import org.springframework.web.bind.annotation.PutMapping;
    import org.springframework.web.bind.annotation.RequestBody;
    import org.springframework.web.bind.annotation.RequestParam;
    import org.springframework.web.bind.annotation.RestController;

    import com.lib.vits.entity.Student;
    import com.lib.vits.service.StudentService;

    @RestController
    public class StudentController {
        @Autowired
        StudentService studentService;

        @PostMapping("/createStudent")
        public ResponseEntity createStudent(@RequestBody Student student) {
```

```
        studentService.createStudent(student);
        return new ResponseEntity("Student Successfully added to the system",
        HttpStatus.CREATED);

        }

        @PutMapping("/updateStudent")
        public ResponseEntity updateStudent(@RequestBody Student student) {
        int lines = studentService.updateStudent(student);
        return new ResponseEntity("Student updated", HttpStatus.OK);
        }

        @DeleteMapping("/deleteStudent")
        public ResponseEntity deleteStudent(@RequestParam("id") int id) {
        studentService.deleteStudent(id);
        return new ResponseEntity("student successfully deleted!!", HttpStatus.OK);
        }
}
```

TransactionController.java:

```
package com.lib.vits.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.lib.vits.service.TransactionService;

@RestController
public class TransactionController {

    @Autowired
    TransactionService transactionService;

    // what i need ideally is card_id and book_id

    @PostMapping("/issueBook")
    public ResponseEntity issueBook(@RequestParam(value = "cardId") int cardId,
    @RequestParam("bookId") int bookId)
            throws Exception {
     String transaction_id = transactionService.issueBooks(cardId, bookId);
     return new ResponseEntity("Your Transaction was successfull here is your Txn id:" +
    transaction_id,
                    HttpStatus.OK);

    }

    @PostMapping("/returnBook")
    public ResponseEntity returnBook(@RequestParam("cardId") int cardId,
```

```
                @RequestParam("bookId") int bookId)
                        throws Exception {
                String transaction_id = transactionService.returnBooks(cardId, bookId);
                return new ResponseEntity("Your Transaction was Successful here is your Txn id:" +
                transaction_id,
                                HttpStatus.OK);

                }
}
```

Entity Class:

AuthorJAVA:

```
package com.lib.vits.entity;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Author {
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Id
        private int id;

        private String name;

        @Column(unique = true)
        private String email;

        private int age;
        private String country;

        @OneToMany(mappedBy = "author", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
        private List<Book> books_written;

        public Author() {
        }

        public Author(String name, String email, int age, String country) {
                this.name = name;
                this.email = email;
                this.age = age;
                this.country = country;
```

```
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getCountry() {
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}

	public List<Book> getBooks_written() {
		return books_written;
	}

	public void setBooks_written(List<Book> books_written) {
		this.books_written = books_written;
	}
}
```

Book.java:

```
package com.lib.vits.entity;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity
public class Book {
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Id
	private int id;

	private String name;

	@Enumerated(EnumType.STRING)
	private Genre genre;

	@ManyToOne
	@JoinColumn
	@JsonIgnore
	Author author;

	@ManyToOne
	@JoinColumn
	@JsonIgnore
	Card card;

	@Column(columnDefinition = "TINYINT(1)")
	private boolean available;

	@OneToMany(mappedBy = "book", cascade = CascadeType.ALL, fetch =
	FetchType.LAZY)
	@JsonIgnore
	private List<Transaction> transactions;
```

```
public Book() {

}

public Book(String name, Genre genre, Author author) {
 this.name = name;
 this.genre = genre;
 this.author = author;
 this.available = true;
}

public int getId() {
 return id;
}

public void setId(int id) {
 this.id = id;
}

public String getName() {
 return name;
}

public void setName(String name) {
 this.name = name;
}

public Genre getGenre() {
 return genre;
}

public void setGenre(Genre genre) {
 this.genre = genre;
}

public Author getAuthor() {
 return author;
}

public void setAuthor(Author author) {
 this.author = author;
}

public Card getCard() {
 return card;
}

public void setCard(Card card) {
 this.card = card;
}

public boolean getAvailable() {
```

```
        return available;
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }

    public boolean isAvailable() {
        return available;
    }

    public List<Transaction> getTransactions() {
        return transactions;
    }

    public void setTransactions(List<Transaction> transactions) {
        this.transactions = transactions;
    }
}
```

Card.java:

```
package com.lib.vits.entity;

import java.sql.Date;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

@Entity
public class Card {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
```

```
@OneToOne(mappedBy = "card", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
private Student student;

@CreationTimestamp
private Date createdOn;

@UpdateTimestamp
private Date updatedOn;

@Enumerated(value = EnumType.STRING)
private CardStatus cardStatus;

@OneToMany(mappedBy = "card", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
private List<Transaction> transactions;

@OneToMany(mappedBy = "card", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
private List<Book> books;

public Card() {
 this.cardStatus = CardStatus.ACTIVATED;
}

public CardStatus getCardStatus() {
 return cardStatus;
}

public void setCardStatus(CardStatus cardStatus) {
 this.cardStatus = cardStatus;
}

public int getId() {
 return id;
}

public void setId(int id) {
 this.id = id;
}

public Student getStudent() {
 return student;
```

```
    }

    public void setStudent(Student student) {
     this.student = student;
    }

    public Date getCreatedOn() {
     return createdOn;
    }

    public void setCreatedOn(Date createdOn) {
     this.createdOn = createdOn;
    }

    public Date getUpdatedOn() {
     return updatedOn;
    }

    public void setUpdatedOn(Date updatedOn) {
     this.updatedOn = updatedOn;
    }

    public List<Transaction> getTransactions() {
     return transactions;
    }

    public void setTransactions(List<Transaction> transactions) {
     this.transactions = transactions;
    }

    public List<Book> getBooks() {
     return books;
    }

    public void setBooks(List<Book> books) {
     this.books = books;
    }
}
```

CardStatus.java:

```
package com.lib.vits.entity;

public enum CardStatus {
    ACTIVATED,
    DEACTIVATED
```

```
}
```

Genre.jav:

```
package com.lib.vits.entity;

public enum Genre {

    FICTIONAL,
    NON_FICTIONAL,
    GEOGRAPHY,
    HISTORY,
    POLITICAL_SCIENCE,
    BOTANY,
    CHEMISTRY,
    MATHEMATICS,
    PHYSICS
}
```

Student.java:

```
package com.lib.vits.entity;

import java.sql.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String emailId;

    private String name;

    // Future scope adult books filter
```

```
private int age;
private String country;

@OneToOne
@JoinColumn
private Card card;

@CreationTimestamp
private Date createdOn;

@UpdateTimestamp
private Date updatedOn;

public Student(String emailId, String name, int age, String country) {
 this.emailId = emailId;
 this.name = name;
 this.age = age;
 this.country = country;
}

public Student() {

}

public int getId() {
 return id;
}

public void setId(int id) {
 this.id = id;
}

public String getEmailId() {
 return emailId;
}

public void setEmailId(String emailId) {
 this.emailId = emailId;
}

public String getName() {
 return name;
}
```

```
    public void setName(String name) {
     this.name = name;
    }

    public int getAge() {
     return age;
    }

    public void setAge(int age) {
     this.age = age;
    }

    public String getCountry() {
     return country;
    }

    public void setCountry(String country) {
     this.country = country;
    }

    public Card getCard() {
     return card;
    }

    public void setCard(Card card) {
     this.card = card;
    }

    public Date getCreatedOn() {
     return createdOn;
    }

    public void setCreatedOn(Date createdOn) {
     this.createdOn = createdOn;
    }

    public Date getUpdatedOn() {
     return updatedOn;
    }

    public void setUpdatedOn(Date updatedOn) {
     this.updatedOn = updatedOn;
    }
}
```

Transaction.java:

```
package com.lib.vits.entity;

import java.sql.Date;
import java.util.UUID;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import org.hibernate.annotations.CreationTimestamp;

@Entity
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String transactionId = UUID.randomUUID().toString();

    @ManyToOne
    @JoinColumn
    private Card card;

    private int fineAmount;

    @ManyToOne
    @JoinColumn
    private Book book;

    @Column(columnDefinition = "TINYINT(1)")
    private Boolean isIssueOperation;

    @Enumerated(EnumType.STRING)
    private TransactionStatus transactionStatus;

    @CreationTimestamp
    private Date transactionDate;

    public Date getTransactionDate() {
     return transactionDate;
    }
```

```
public void setTransactionDate(Date transactionDate) {
 this.transactionDate = transactionDate;
}

public int getId() {
 return id;
}

public void setId(int id) {
 this.id = id;
}

public String getTransactionId() {
 return transactionId;
}

public void setTransactionId(String transactionId) {
 this.transactionId = transactionId;
}

public Card getCard() {
 return card;
}

public void setCard(Card card) {
 this.card = card;
}

public int getFineAmount() {
 return fineAmount;
}

public void setFineAmount(int fineAmount) {
 this.fineAmount = fineAmount;
}

public Book getBook() {
 return book;
}

public void setBook(Book book) {
 this.book = book;
}

public Boolean getIssueOperation() {
 return isIssueOperation;
}

public void setIssueOperation(Boolean issueOperation) {
 isIssueOperation = issueOperation;
}
```

```
        public TransactionStatus getTransactionStatus() {
         return transactionStatus;
        }

        public void setTransactionStatus(TransactionStatus transactionStatus) {
         this.transactionStatus = transactionStatus;
        }
}
```

TransactionStatus.java:

```
package com.lib.vits.entity;

public enum TransactionStatus {

    SUCCESSFUL,
    PENDING,
    FAILED
}
```

Com.lib.vits.Repository:

AuthorRepository.java:

```
        package com.lib.vits.repository;

        import com.lib.vits.entity.Author;
        import org.springframework.data.jpa.repository.JpaRepository;
        import org.springframework.data.jpa.repository.Modifying;
        import org.springframework.data.jpa.repository.Query;
        import org.springframework.data.repository.query.Param;

        import javax.transaction.Transactional;

        @Transactional
        public interface AuthorRepository extends JpaRepository<Author, Integer> {
            @Modifying
            @Query("update Author a set a.name=:#{#new_author.name}," +
            "a.email=:#{#new_author.email},"
                    + "a.age=:#{#new_author.age}," + "a.country=:#{#new_author.country}
            where a.id=:#{#new_author.id}")
            int updateAuthorDetails(@Param("new_author") Author new_author);

            @Modifying
            @Query("delete Author a where a.id=:given_id")
            int deleteCustom(@Param("given_id") int id);
}
```

BookRepositry.java:

```
package com.lib.vits.repository;

import com.lib.vits.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.web.bind.annotation.RequestParam;

import javax.transaction.Transactional;
import java.util.List;

@Transactional
public interface BookRepository extends JpaRepository<Book, Integer> {
    @Modifying
    @Query("update Book b set
    b.card=:#{#book?.card},b.available=:#{#book?.available} where
    b.id=:#{#book?.id}")
    int updateBook(@RequestParam("book") Book book);

    @Query("select b from Book b where b.genre=:genre and
    b.available=:isAvailable and b.author in (select a from Author a where
    a.name=:author)")
    List<Book> findBooksByGenre_Author(@Param("genre") String genre,
    @Param("author") String author,
            @Param("isAvailable") boolean isAvailable);

    @Query("select b from Book b where b.genre=:genre and
    b.available=:isAvailable")
    List<Book> findBooksByGenre(@Param("genre") String genre,
    @Param("isAvailable") boolean isAvailable);

    @Query("select b from Book b where b.available=:isAvailable and b.author
    in(select a from Author a where a.name=:author)")
    List<Book> findBooksByAuthor(@Param("author") String author,
    @Param("isAvailable") boolean isAvailable);

    @Query("select b from Book b where b.available=:isAvailable")
    List<Book> findBooksByAvailability(@Param("isAvailable") boolean
    isAvailable);
```

```
    @Modifying
    @Query("delete Author a where a.id=:given_id")
    int delete(@Param("given_id") int id);

}
```

CardRepository:

```
package com.lib.vits.repository;

import com.lib.vits.entity.Card;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import javax.transaction.Transactional;

@Transactional
public interface CardRepository extends JpaRepository<Card,Integer> {

   @Modifying
   @Query(value = "update card c set c.card_status=:new_card_status where c.id
     in(select card_id from student s where s.id=:student_id)",nativeQuery = true)
   void deactivateCard(@Param("student_id") int
     student_id,@Param("new_card_status") String new_card_status);

}
```

SudentRepository.java:

```
package com.lib.vits.repository;

import java.util.List;

import javax.transaction.Transactional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.lib.vits.entity.Student;
```

```
@Transactional
public interface StudentRepository extends JpaRepository<Student, Integer> {

    @Modifying
    @Query("update Student s set s.emailId=:newEmail where
    s.emailId=:oldEmail")
    int updateStudentEmail(@Param("oldEmail") String oldEmail,
    @Param("newEmail") String newEmail);

    @Modifying
    @Query("delete from Student s where s.id=:id ")
    int deleteCustom(@Param("id") int id);

    @Modifying
    @Query("update Student s set s.emailId= :#{#student.emailId}," +
    "s.name=:#{#student.name},"
            + "s.age=:#{#student.age}," + "s.country=:#{#student.country}
    where s.id=:#{#student.id}")
    int updateStudentDetails(@Param("student") Student student);

    @Query("select b from Student b where b.emailId=: mail")
    List<Student> find_by_mail(String mail);

    @Query(value = "select * from student s where s.email_id=:mail",
    nativeQuery = true)
    List<Student> findbymail(String mail);

}
```

TransactionRepository.java:

```
package com.lib.vits.repository;

import java.util.List;

import javax.transaction.Transactional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.lib.vits.entity.Transaction;
import com.lib.vits.entity.TransactionStatus;
```

```
@Transactional
public interface TransactionRepository extends
     JpaRepository<Transaction,Integer> {


   @Query("select t from Transaction t where t.card.id=:card_id and
     t.book.id=:book_id and t.transactionStatus=:status and
     t.isIssueOperation=:isIssue")
   public List<Transaction> findByCard_Book(@Param("card_id") int card_id,
                                @Param("book_id") int book_id,
                                @Param("status") TransactionStatus status,
                                @Param("isIssue") boolean isIssue);



}
```

Com.lib.vits.service:

AuthorSerivce.java:

```
package com.lib.vits.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.lib.vits.entity.Author;
import com.lib.vits.repository.AuthorRepository;

@Service
public class AuthorService {
   @Autowired
   AuthorRepository authorRepository;


   public void createAuthor(Author author){
      authorRepository.save(author);

   }
   public void updateAuthor(Author author){
      authorRepository.updateAuthorDetails(author);
   }
   public void deleteAuthor(int id ){
```

```
            authorRepository.deleteCustom(id);
        }
}
```

BookService.java:

```
package com.lib.vits.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.lib.vits.entity.Book;
import com.lib.vits.repository.BookRepository;

@Service
public class BookService {

    @Autowired
    BookRepository bookRepository;

    public void createBook(Book book) {
     bookRepository.save(book);

    }

    public List<Book> getBooks(String genre, boolean isAvailable, String
    author) {

     if (genre != null && author != null) {
            return bookRepository.findBooksByGenre_Author(genre, author,
    isAvailable);
     } else if (genre != null) {
            return bookRepository.findBooksByGenre(genre, isAvailable);
     } else if (author != null) {
            return bookRepository.findBooksByAuthor(author, isAvailable);
     }
     return bookRepository.findBooksByAvailability(isAvailable);

    }

    public void deleteBook(int id) {
     bookRepository.delete(id);
```

```
    }

    public void updateBook(Book book) {
     bookRepository.save(book);

    }

}
```

CardService.java:

```
package com.lib.vits.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.lib.vits.entity.Card;
import com.lib.vits.entity.CardStatus;
import com.lib.vits.entity.Student;
import com.lib.vits.repository.CardRepository;

@Service
public class CardService {


   @Autowired
   CardRepository cardRepository;


   public Card createCard(Student student){
      Card card =new Card();
      student.setCard(card);
      card.setStudent(student);
      cardRepository.save(card);
      return card;
   }
   public void deactivate(int student_id){
      cardRepository.deactivateCard(student_id,
     CardStatus.DEACTIVATED.toString());

   }
}
```

StudentService.java:

```
package com.lib.vits.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.lib.vits.entity.Card;
import com.lib.vits.entity.Student;
import com.lib.vits.repository.CardRepository;
import com.lib.vits.repository.StudentRepository;

@Service
public class StudentService {

    Logger logger = LoggerFactory.getLogger(StudentService.class);

    @Autowired
    StudentRepository studentRepository;

    @Autowired
    CardRepository cardRepository;

    @Autowired
    CardService cardService;

    public void createStudent(Student student) {

     Card card = cardService.createCard(student);
     logger.info("The card for the student{} is created with the card details{}",
    student, card);

    }

    public int updateStudent(Student student) {
     return studentRepository.updateStudentDetails(student);

    }

    public void deleteStudent(int id) {
```

```
            cardService.deactivate(id);
            studentRepository.deleteCustom(id);

           }
}
```

TransactionService.java:

```
package com.lib.vits.service;

import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import com.lib.vits.entity.Book;
import com.lib.vits.entity.Card;
import com.lib.vits.entity.CardStatus;
import com.lib.vits.entity.Transaction;
import com.lib.vits.entity.TransactionStatus;
import com.lib.vits.repository.BookRepository;
import com.lib.vits.repository.CardRepository;
import com.lib.vits.repository.TransactionRepository;

@Service
public class TransactionService {

    @Autowired
    TransactionRepository transactionRepository;
    @Autowired
    BookRepository bookRepository;
    @Autowired
    CardRepository cardRepository;
    @Value("${books.max_allowed}")
    int max_allowed_books;
    @Value("${books.max_allowed_days}")
    int max_days_allowed;
    @Value("${books.fine.per_day}")
    int fine_per_day;

    public String issueBooks(int cardId, int bookId) throws Exception {
```

```
Book book = bookRepository.findById(bookId).get();
System.out.println(book);

if (book == null || book.isAvailable() != true) {
        throw new Exception("Book is either unavailable or not present!!");
}
Card card = cardRepository.findById(cardId).get();
System.out.println(card);
if (card == null || card.getCardStatus() == CardStatus.DEACTIVATED) {
        throw new Exception("Card is invalid!!");
}
if (card.getBooks().size() > max_allowed_books) {
        throw new Exception("Book limit reached for this card!!");
}
book.setAvailable(false);
book.setCard(card);
List<Book> books = card.getBooks();
books.add(book);
card.setBooks(books);
bookRepository.updateBook(book);
Transaction transaction = new Transaction();
transaction.setCard(card);
transaction.setBook(book);
transaction.setIssueOperation(true);
transaction.setTransactionStatus(TransactionStatus.SUCCESSFUL);
transactionRepository.save(transaction);
return transaction.getTransactionId();
}

public String returnBooks(int cardId, int bookId) throws Exception {
List<Transaction> transactions =
transactionRepository.findByCard_Book(cardId, bookId,
                TransactionStatus.SUCCESSFUL, true);
Transaction last_issue_transaction = transactions.get(transactions.size() - 1);
// Last transaction that has been done ^^^^
Date issueDate = last_issue_transaction.getTransactionDate();
Long issueTime = Math.abs(issueDate.getTime() -
System.currentTimeMillis());
long number_of_days_passed = TimeUnit.DAYS.convert(issueTime,
TimeUnit.MILLISECONDS);
int fine = 0;
if (number_of_days_passed > max_days_allowed) {
        fine = (int) Math.abs(number_of_days_passed - max_days_allowed)
* fine_per_day;
```

```
            }
            Card card = last_issue_transaction.getCard();
            Book book = last_issue_transaction.getBook();
            book.setCard(null);
            book.setAvailable(true);
            bookRepository.updateBook(book);
            Transaction new_transaction = new Transaction();
            new_transaction.setBook(book);
            new_transaction.setCard(card);
            new_transaction.setFineAmount(fine);
            new_transaction.setIssueOperation(false);
            new_transaction.setTransactionStatus(TransactionStatus.SUCCESSFUL);
            transactionRepository.save(new_transaction);
            return new_transaction.getTransactionId();

           }

}
```

Application Properties Class:

```
server.port=9090
spring.datasource.url=jdbc:mysql://localhost:3306/sys
spring.datasource.username=root
#If MySQL installation is password proctored,then use below property to set password
spring.datasource.password=root
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
#JPA settings
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.hibernate.ddl-auto=update

books.max_allowed=30
books.max_allowed_days=15
books.fine.per_day=5
```

5.2 References

1. Sharma,C.K;Singh,Kiran(2005),Library Management,ISBN978-81-269-0452-5
2. McClure,C.R.(1980).”LIBRARY MANAGERS: CAN THEY MANAGE? Will they Lead?”.Library Journal:2391.
3. http://www.java2s.com/ 1. http://www.javaworld.com/javaworld/jw01...eview.html
4. Database Programming with JDBC and Java by O'Reilly
5. Head First Java 2nd Edition
6. http://java.sun.com/javse/technologies/desktop/
7. Github for reference.

5.3 LIST OF WEBSITES

I. **https://start.spring.io/**

II. **https://www.postman.com/downloads/**

III. **https://www.mysql.com/downloads/**

IV. **https://www.eclipse.org/downloads/**