SOEN 6441 - Advanced Programming Practices

PROJECT

Submitted to: Dr. Constantinos Constantinides

Submitted by:

Arnav Ishaan (40216397)

Reva Balasundaram (40227644)

Section: W-WI

Project Name: User Library

Project Link: [userLibrary Project - Github: knightwayne]

Index:

Report

1. **Abstract**:

The objective of the project is to design & implement an application which fetches data from an API, and stores it in a local database, where we can perform basic CRUD(Create, Read, Update, Delete) operations to validate its functioning.
For this, we are using 3 APIs supported by NewYorkTimes, to fetch data of Articles, Movie Reviews & Book Reviews, given a *Query* string is captured by the user input, and storing it in the local database developed using SQLite.
The user gives an integral rating to the fetched Articles, Movie Reviews & Book Reviews, and can perform basic CRUD operations on the tables of Articles, Movie Reviews & Table Reviews.

This application follows the conventional Java Coding Standards, and employs Object Relational Structural Patterns, and Data Source Architectural Pattern to design the application.We use refactoring standards to optimize the code, and JUnit framework for unit testing of the application.

a. Languages: Java, Sqlite as Query Language
b. Environment: Eclipse for Development,
c. External JARs:
    i. For JSON processing: json-simple-1.1.1.jar, json-20220924.jar,
    ii. For IO & parsing: commons-io-2.4.jar,
    iii. For Sqlite Database: sqlite-jdbc-3.39.3.0.jar,
    iv. For testing: junit-4.10.jar, junit-jupiter-api-5.9.1.jar
d. Frameworks & Libraries: JUnit for testing, JDBC for database connection

A detailed description of the program execution is discussed below.

2. Program Execution Cycle:

Initialization of Program

This application allows the user to fetch data from the NewYorkTimes APIs, and perform basic query processing operations, Create, Read, Update & Delete on the tables created in the application.

The program's execution calls the main( ) method in the MainApp.java file. Firstly, it initializes the database, through initDatabase( ) method defined in initDb.java file.
Four tables, namely USERINFO, ARTICLE, MOVIEREVIEW & BOOKREVIEW schemas are created, and pre-populated with few values. This is achieved using the Sqlite Database engine, and connected concurrently with the Java program using Java Database Connectivity(JDBC) API.
The tables created in the application are analogous to the objects instantiated in the program of Class type, User, Article, MovieReview & BookReview respectively. The attributes of each class are the Column Names of the tables, and each newly created object acts as a row of their table. To maintain the link between the in memory object and the row in the table, we create a new attribute in each class, an Id, which is equal to the primary key of the row of the table.

Switch Case & CRUD Operations

Next, using a switch case conditional loop, we process the user input, an integer which reflects their choice to perform one of the following operations -
Display All Tables, Create New Data (create new user or use APIs to fetch desired Media type), Read Data from Tables, Update Data from Tables, Delete Data from Tables & Exit from the program. For performing the CRUD operations on each table, a respective Table Data Gateway object is created, employing the use of Data Source Architectural Pattern to create abstraction between in memory objects and the database, and the appropriate method is called. These are discussed below.

1. For switch case 0, Display All Tables, we display all the records present in our tables. This function helps the end user to see the real time status of the data stored in the database.
2. For Switch Case 1: the Create New Entry for Table, createDataFunc( ) function creates a new entry in one of the four tables mentioned above.

Table Data Gateway Class & Objects

a. For creating a new record in the table and a new in memory object, a new object of the Table Data Gateway class is created for the respective table, which acts as an intermediate interface between objects and the data in databases.
b. For creating a new System user in the application(in the USERINFO table), an object of class UserTdg is created. Input fields of Name, Email, Password & Address are required to create a new object and a

new record in the USERINFO table. A new user object is created and using the public setter() methods, its attributes are set as the given user input value.

c. For creating a new data record in the Article, Movie Review or Book Review table, the user inputs a query string to fetch a list of articles, movie reviews or book reviews from the NewYorkTimes websites, and store it in the local database. Additionally, the user inputs and sets the rating for the particular Article, Movie Review or Book Review which they have queried for.

<u>NewYorkTimes APIs & fetching JSON Data</u>

d. For the articles, if the query string matches with the headline of the article, the article is fetched. For the Movie Review, the query string should match with the title of the movie & for Book Review, the query string should match *exactly* with the title of the book.
e. For each of the following tables, a new object of ArticleTdg, MovieTdg or BookTdg class respectively is created, which calls the createData( ) method to create the table records & objects of the class.
f. Using the fetchArticleAPI, fetchMovieAPI or fetchBookAPI for the respective tables, a connection to the API is set up, and data is fetched back and processed. If the application is unable to make a connection to the API, it sends back an Error Signal Code.
g. The processed data is filled in the corresponding rows of table, and a new object of the type is instantiated and using the public setter() methods, its attributes are set as the given user input value.

3. For Switch Case 2: Read from Table, readDataFunc( ) displays data records from one of the four tables. The user can either input the column name/attribute, whose value the application would read, or enter 'All' to print the values of all attributes of the table.

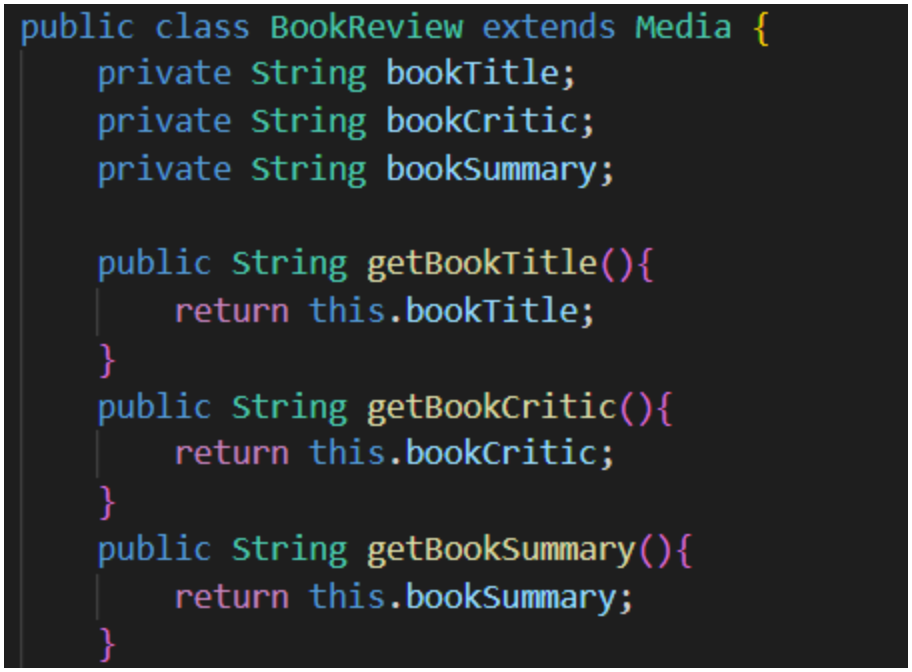
<u>Where Clause for Update & Delete Operations</u>

4. For Switch Case 3: Update from Table, updateDataFunc( ) is called. This method updates the rating for the Article, Movie Review & Book Review from its respective table which it is queried for. It takes the input as updated Rating value, and 2 queries, which acts as the conditionals for the where clause. To update the rating for some records in the table, the application needs to narrow down the records using the Where clause, which is used here as 'WHERE ColName = DataValue'. This means the rating is updated only for those records, whose ColName attribute value matches with the

given DataValue. This application also inputs the attribute and the data value, along with the rating to execute the Update statement.
    a. For the User Table, the application takes 4 inputs, respectively being the attribute name to be updated, the new value for the attribute, and the conditional attribute name and value mentioned above.
5. For switch Case 4, Delete from table, deleteDataFunc( ) is called, which again takes 2 input values from the user. The conditional attribute name and the value which the attribute should hold to satisfy the Where clause, and narrow down to the records to be deleted.
    a. For the User Info Table, if we delete a particular user with userId, then all articles, movie review and book review for the particular user is deleted from their respective table. This is done using one of the Object Relational Structural Pattern, Foreign Key Mapping between different tables, which associates dependent attributes among different classes, here showing one to one relation between User and Article, User and MovieReview & User and BookReview tables.
6. For Switch Case 5: Exit program, we break from the switch loop, and delete the local database created by the program. This terminates the program.

3. Coding Standards

1. Variable Scopes:
   We use Public & Private access modifiers to encapsulate attributes and methods of the classes in a different way. The class has private attributes which can only be accessed by the public getter() and setter() methods of the respective class.

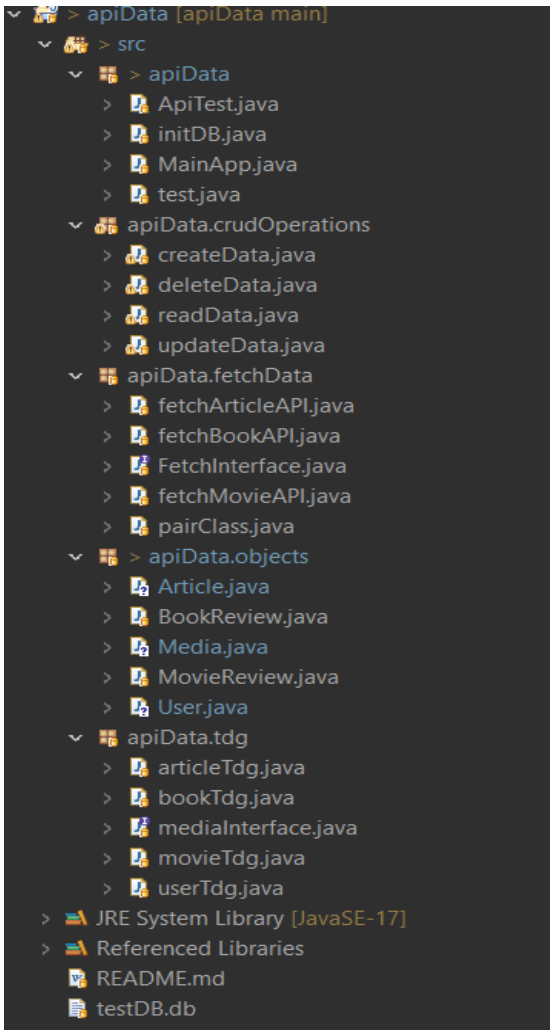


```
public class BookReview extends Media {
    private String bookTitle;
    private String bookCritic;
    private String bookSummary;

    public String getBookTitle(){
        return this.bookTitle;
    }
    public String getBookCritic(){
        return this.bookCritic;
    }
    public String getBookSummary(){
        return this.bookSummary;
    }
```

2. Encapsulation:
   The code also uses encapsulation, in combining the attributes & methods of the class & hiding it from outside the scope. This is done using access modifiers, as discussed in the first point.
   The above picture showcases the example of Encapsulation.

3. Readability, Modularity & Indentation:
   The code is readable, simple, and concise. Appropriate indentations have been used to make the code more understandable for developers & end users.
   It is divided into folder structure with appropriate names which indicate the function of the files in the folder, hence making the code more modular.



This shows the code has been modularized into different folder structures, with each folder named after the code functionality.

```java
public class MainApp{
    Run | Debug
    public static void main(String[] args) throws IOException, SQLException, SQLWarning
    {
        try
        {
            initDB.initDatabase();

            int input = 0;
            Scanner reader = new Scanner(System.in);
            do{
                System.out.println("Enter your Choice Input as a Number");
                System.out.println("0.Display All Tables\t1.Create new entry in table afte
                input = reader.nextInt();
                System.out.println(input);

                switch (input) {
                case 0:
                {
                    System.out.println("Display All Tables");
                    //displayTables();
                }
                break;
                case 1:
                {
                    System.out.println("Create a New Entry in Table");
                    createData.createDataFunc();
                }
                break;
                case 2:
                {
```

The code here is clean, concise and well indented, and this is followed throughout the code-base.

4. <u>Naming Conventions:</u>
   Standard Enterprise accepted naming conventions have been used to name the identifiers in the code
   a. Variable: Noun starting with lowercase letter, if it contains multiple words then every inner word should start with uppercase(follow Camelcase pattern).

b. Class: Noun starting with uppercase letter, or if the class name is a combination of multiple nouns, the class name should be written in camelcase pattern.
c. Methods: Method name should either be verb or verb noun combination starting with lower letter.

```
package apiData.objects;

public class MovieReview extends Media {
    private String movieTitle;
    private String movieCritic;
    private String movieSummary;

    public String getMovieTitle(){
        return this.movieTitle;
    }
    public String getMovieCritic(){
        return this.movieCritic;
    }
    public String getMovieSummary(){
        return this.movieSummary;
    }

    public void setMovieTitle(String movieTitle){
        this.movieTitle=movieTitle;
    }
    public void setMovieCritic(String movieCritic){
        this.movieCritic=movieCritic;
    }
    public void setMovieSummary(String movieSummary){
        this.movieSummary=movieSummary;
    }
}
```

The naming for class MovieReview, variable movieTitle and method getMovieTitle( ) follows the above described naming conventions.

5. Documentation & Comments:

Comments have been added describing the action performed by the method along with the parameters used in the method and the return type if any are present.

```
/**
 * Uses the attribute name selected as parameter and reads the data from the database
 * Starts an SQL connection
 * Executes the read query from the article table
 * @param query the attribute name to be read from the table
 */

public void read(String query)
{
    try
    {
        Connection c = null;
        Statement stmt = null;
        String sql="";
```

6. Abstraction

The code uses the principle of abstraction by hiding the implementation details in methods in such a way that the caller of the method is only familiar with the input parameters and the return type of the method.

```
public void create(int userId, int rating, String query)
{
    try
    {
        Connection c = null;
        Statement stmt = null;

        //Class.forName("org.sqlite.JDBC");
        c = DriverManager.getConnection("jdbc:sqlite:testDB.db");
        c.setAutoCommit(false);
        System.out.println("Opened database successfully");
        stmt = c.createStatement();

        pairClass recv = new pairClass();
        recv=fetchArticleAPI.getResponse(query);
```

The create method which calls the fetchArticleAPI.getResponse( ) method knows nothing about the implementation detail of this method, other than the input parameters is a String, and the return type is an object of class pairClass.

4. Design Pattern:

a. Object Relational Structural Pattern:

The application uses object oriented logic for the business end & relational database logic for storing the data captured during the execution of a program in the database.

To create a synchronized state between the object model and relational databases, we employ Object Relational Structural Pattern. This application uses the following-

1. Simple Mapping: Individual Classes of User, Article, Movie & Book are mapped to the UserInfo, Article, MovieReview & BookReview table. An instance of the created object is analogous to the rows of the respective table, where each column of the table is one of the attributes of the respective class.
2. Identity Field Mapping: Relational Databases use primary key, a uniquely generated Id to differentiate between the rows of table. To tie the in memory object to the particular row, we create an Id attribute in the class declaration, which helps in linking the object & data row.

```
public class Media {
    private int id;
```

```
public class User {

    private int id;
```

3. Foreign Key Mapping: The code uses a foreign key mapping to show the one to many association between the User class and the Article Class, Movie Class & Book Class.

```
public class Media {
    private int id;
    private String mediaURL;
    private String mediaDate;
    private String mediaRating;
    private int userId;
```

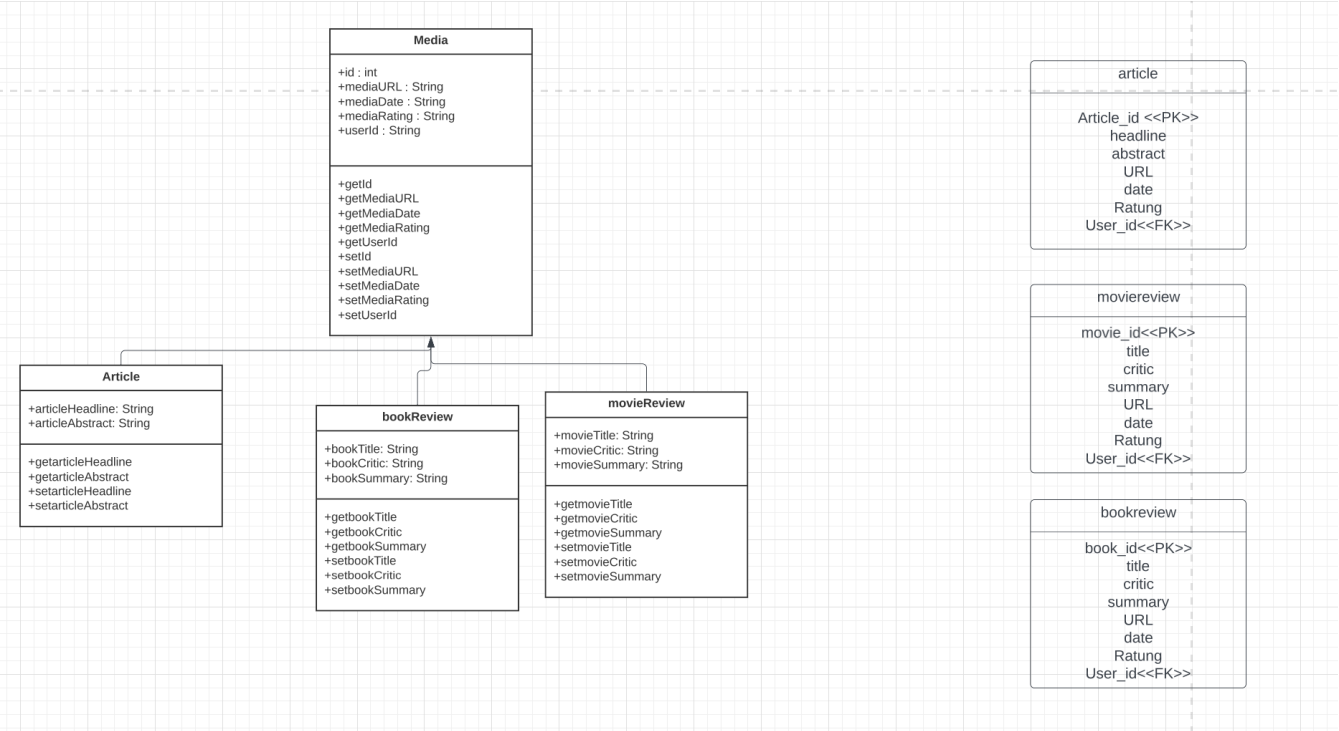
```
articleTdg tdg1=new articleTdg();
tdg1.delete(lhs: "USER_ID",rhs);
movieTdg tdg2=new movieTdg();
tdg2.delete("USER_ID",rhs);
bookTdg tdg3=new bookTdg();
tdg3.delete("lhs: USER_ID",rhs);
```

The userId in the media class is a foreign key linking it with the primary key of userId of user. Deleting the user from the UserInfo table deletes all the linked Articles, Movie Review & Book Review data from their respective tables.

4. Horizontal Mapping: The code uses Horizontal Mapping as it has tables only for concrete(non abstract) classes. The Media class is a super class which is extended by Article, Movie & Book Class. The primary purpose of using a superclass is to extract common attributes and methods, which are used by other children classes. Hence, the application does not require any explicit implementation of the article object. Therefore creating a table would only be for storing the

common attributes that would lead to multiple access of the Superclass, and multiple uses of join statements, in turn leading to unnecessary waste of time & resources, hence the application uses a Table per concrete class inheritance mapping pattern.

The 3 concrete classes here are Article, MovieReview & BookReview, and the application uses 3 corresponding tables to the class, namely Article, MovieReview & BookReview. No table for article class is created.



b. Data Source Architectural Pattern:

To add a layer of abstraction between our business logic and database, the application utilizes a data source architectural pattern, namely Table Data Gateway.

Table Data Gateway Pattern: The application utilized a special set of classes, implemented in the Table Data-Gateway folder(tdg), whose objective is to create an access gateway for the tables in the database. An object of the classes implemented in the tdg folder, like userTdg, articleTdg, movieTdg, bookTdg acts as a layer of information transfer between objects of classes user, article, movie, book & their respective tables.

This ensures that the in memory objects, and by extension the business end of logic, where the objects are instantiated are independent of the database schema & implementation, hence providing an added layer of abstraction.

```
userTdg TDG = new userTdg();
TDG.create(NAME, EMAIL, PASSWORD, ADDRESS);
```

Here, the application instantiates a new object of class userTdg, and it invokes the create method to create a new data-row in UserInfo Table.

```
User user = new User();
user.setName(NAME); user.setEmail(EMAIL);
user.setPassword(PASSWORD); user.setAddress(ADDRESS);
user.setId(TDG.returnUserId(query: "ID"));
```

Then, we create a new object of class User, and use the setter methods to correctly initialize the attributes of user object with the help of the TDG object, like in this case, using the returnUserId( ) method to set the auto-generated primary key.

Below code shows the userTdg class's create( ) method which creates a new entry in the UserInfo Table.

```
public void create(String NAME,String EMAIL,String PASSWORD,String ADDRESS)
{
    try
    {
        Connection c = null;
        Statement stmt = null;
        String sql="";
        //Class.forName("org.sqlite.JDBC");
        c = DriverManager.getConnection("jdbc:sqlite:testDB.db");
        c.setAutoCommit(false);
        System.out.println("Opened database successfully");
        stmt = c.createStatement();

        sql = "INSERT INTO USERINFO (NAME,EMAIL,PASSWORD,ADDRESS) "+
            "VALUES ('" +
            NAME + "', '" +
            EMAIL + "', '" +
            PASSWORD + "', '" +
            ADDRESS +
            "');";
```

5. Refactoring:

a. Use of Interface: Interfaces like mediaInterface & FetchInterface are implemented by classes of Table Data-Gateway folder(tdg) & fetchData folder respectively, which helps in achieving code abstraction.

```
public interface mediaInterface {

    public void create(int userId, int rating, String query);

    public void read(String query);

    public void update(String queryValue, String lhs, String rhs);

    public void delete(String lhs, String rhs);

}
```

```
public interface FetchInterface {

    public static pairClass getResponse(String query){
        pairClass res = new pairClass();
        return res;
    }
}
```

b. Use of Inheritance: A superclass Media is created, which contains the common attributes and methods of Article, Movie & Book class.

```
public class MovieReview extends Media {
```

```
public class BookReview extends Media {
```

```
public class Article extends Media {
```

c. Replace Array with Object: The code returns multiple objects in form of a class rather than using an array or 2 different methods, hence refactoring the code.

```
package apiData.fetchData;

import java.util.Scanner;

public class fetchArticleAPI implements FetchInterface{

    public static pairClass getResponse(String query)
    {
        pairClass res = new pairClass();
```

Here, the return type of getResponse() method is a pairClass object, which has 2 attributes, an int responseCode, and JSONArray object mediaArray.

d. Decompose Conditional Statements: The code employs the use of switch cases at appropriate places and encapsulates parts of the conditional logic into separate methods.

```
int table=reader.nextInt();
if(table==1)
{
    System.out.println("Enter User Id");
    int userId=reader.nextInt();
    System.out.println("Enter Article Query String");
    reader.nextLine();
    String query=reader.nextLine();
    System.out.println("Enter Rating for Article");
    int rating=reader.nextInt();
    articleTdg TDG = new articleTdg();
    TDG.create(userId, rating, query);
}
else if(table==2)
{
    System.out.println("Enter User Id");
    int userId=reader.nextInt();
    System.out.println("Enter Movie Query String");
    reader.nextLine();
    String query=reader.nextLine();
    System.out.println("Enter Rating for Movie");
    int rating=reader.nextInt();
    movieTdg TDG= new movieTdg();
    TDG.create(userId, rating, query);
}
else if(table==3)
```

6. Testing

In order to write and run tests, JUnit is being used. JUnit is a framework that gives test runners for running test cases and has annotations for test cases. JUnit makes it easier to find errors earlier on in the development phase and then rectify them. The code uses some data to check for the creation,updation, insertion and deletion of the data into the tables that have been created. The code also tests if the API is being called properly, if the return type of the API is indeed a JSON and checks if the data being returned in the JSON is the data that is being requested.

```
//To test the database inserts
@Test
public void createDisplayUpdateUser()
```

```
// to check API calls
@Test
public void articleAPIcallcheck()
  throws IOException, ParseException
```

7. Software Architecture Document:

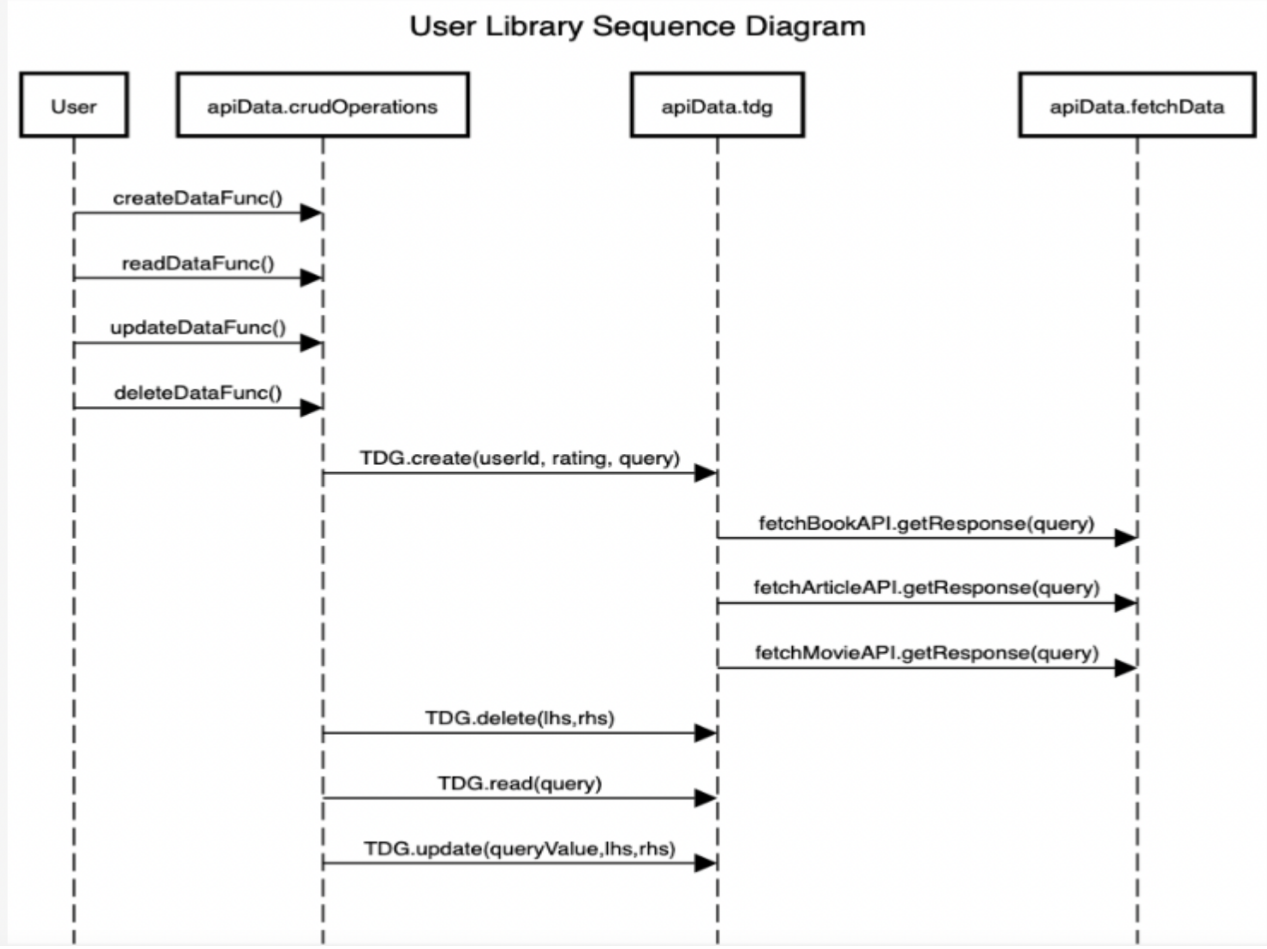
a. Class Diagram:

This is an auto-generated class diagram for the application, which includes the Classes & Interfaces in the application.

b. Sequence Diagram:

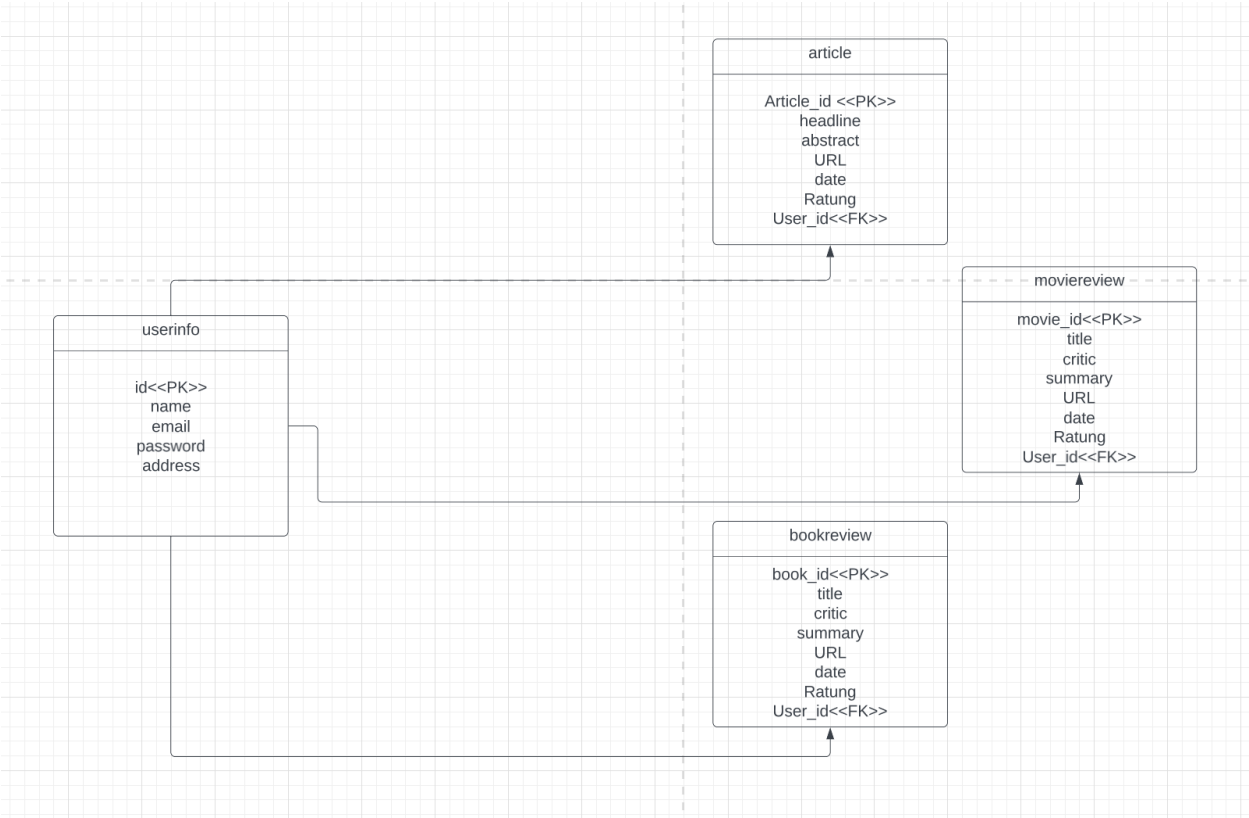
The following sequence diagram shows the flow of execution in the application.

The TDG object represented in the diagram below is an object of any of the four Tdg classes, userTdg, articleTdg, movieTdg or bookTdg.



c. <u>Database Schema</u>

The database schema for the application is shown below. The arrow mapping represents the foreign key relation between the tables.



8. Conclusion:

The above report corroborates that the application is designed with keeping in mind the Standard Coding practices, and utilizes design patterns, code refactoring & testing to develop the application.