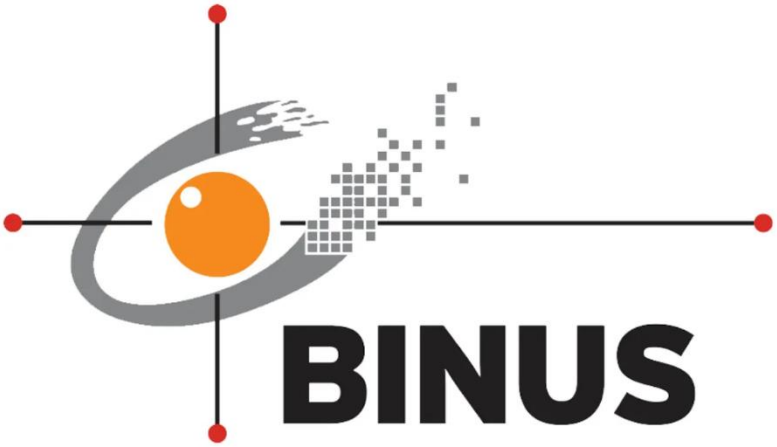
Object Oriented Programming Java Final Project



BINUS UNIVERSITY INTERNATIONAL

By:

Michael Christopher

2440047362

Class L2AC

Lecturer:

Jude Joseph Lamug Martinez

D4017

Project Specification

1. What's the project?

In this project, I made a simple GUI using the Java programming language and JavaFX. It is a watch/read list GUI application that is connected to databases. The application also has a login menu to ask users to input their username and password.

2. Purpose of the project

In this project, I create a GUI in Java for the final project and this GUI allows users to record their watch/read list. For example, users can add movies that they haven't watched and want to watch later, they can check what books they are reading, they can keep track on tv series that they have watched, they can update the records, and delete the record if it is no longer necessary.

3. What's the vision and mission for this project?

To make a functional program that a lot of people will find it useful and they like using it. This program needs to be able to keep track of the users record by saving data to database.

Solution Design

1. Introduction

In this second semester majoring Computer Science in BINUS University International, we were taught about Object Oriented Programming with Java as its programming language. The lecturer taught us from the start such as data types, variables, conditional, loops, and eventually later on advanced to OOP. We have learned OOP back in our first semester using Python programming language, but we dive deeper to the world of OOP in this second semester course. We were taught basic about OOP, 4 pillars of OOP, how OOP works in Java, how to implement it in Java, how to design classes, creating UML diagram, abstraction, and many more. Just like the first semester, we need to create a final project implementing what we have learned during our class and we give presentation plus demonstration of our program to the lecturer.

For me, I didn't think about this project until around 1 month before the submission date. I spend like almost a week to think what I want to make and searching what kind of framework I want to work with. Initially, I want to make this project using Springboot framework and make a web application but after watching some tutorial, I find the quite hard to implements since I need to also code the front end of the web using HTML, CSS, or other similar language. After that I switched my idea to make a GUI application and I tried to use Swing, but since Swing is quite old fashioned so I move on to a newer framework called JavaFX. It is a lot easier using the JavaFX than Springboot because I can drag and drop the application without coding. The idea of making the GUI application as a read/watch list application actually just came to my mind in the process of making the application.

I started doing this project in the middle of May 2021 and I use Apache Netbeans as my IDE for this project. This project will be uploaded in my GitHub account, https://github.com/EuphosiouX/FinalOOP.git

Image 1 – Apache Netbeans IDE

(https://www.andreszsogon.com/wp-content/uploads/logo_apache_netbeans_cordova.png)

2. Overview

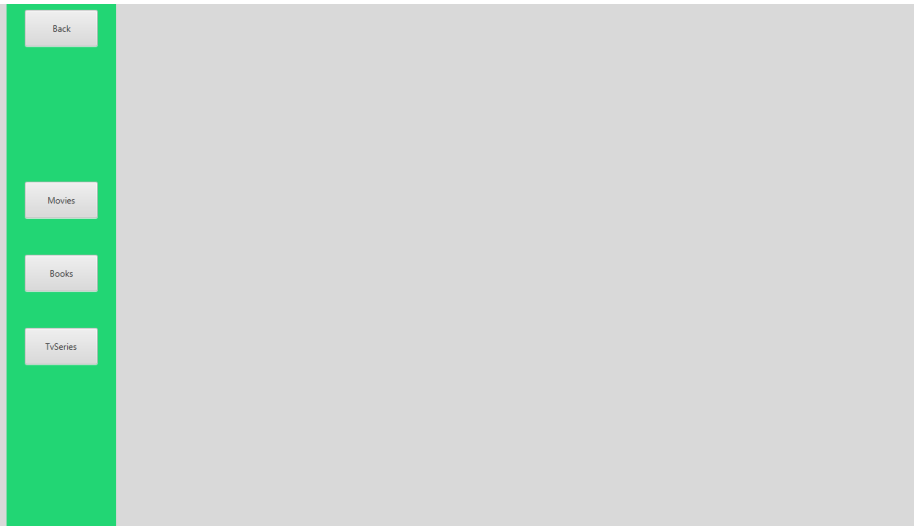


Image 2 – Application's dashboard

I don't have a title or name for this GUI application so I just call it "Read and Watch List Application" and it is an application where users can input movies, books, or tvseries to record them. This application can be used by people in various age range and this application might be useful to them and provide good services.

This GUI application is created in Java 8 using JavaFX framework, Gluon Scene Builder app to build the application appearance, MySQL for the database, and MySQL connector.

Image 3 -Java logo

(https://spzone-simpleprogrammer.netdna-ssl.com/wp-content/uploads/2016/11/Untitled-1-6-1024x576.png)

Image 4 – JavaFX logo

(https://bangness.net/wp-content/uploads/2019/01/JavaFX_Logo.png)

Image 5 – MySQL logo

(https://upload.wikimedia.org/wikipedia/id/a/a9/MySQL.png)

Image 6 – Gluon Scene Builder Logo

(https://gluonhq.com/wp-content/uploads/2015/02/SceneBuilderLogo-300x300.png)

3. Code Flow

The codes and classes for this program are all categorized into packages of their own category so there are a lot of packages in this project. For example controllers class are all in the same package, entertainments class with its child in the same package, and others. For the UML class diagram, I use simple classes since some of my class have a lot of variables and methods that it will not fit into the diagram but I also provided the complete diagram. The complete diagram have incomplete and incorrect relationship between class so it is used only to see what variables and methods are inside the classes and please use the simple diagram to see the relationship between classes. Also, the color on the lines and some classes is only to make it easier to see the lines and serve no other purposes. For clearer UML diagram, .png files are available in the GitHub link provided.

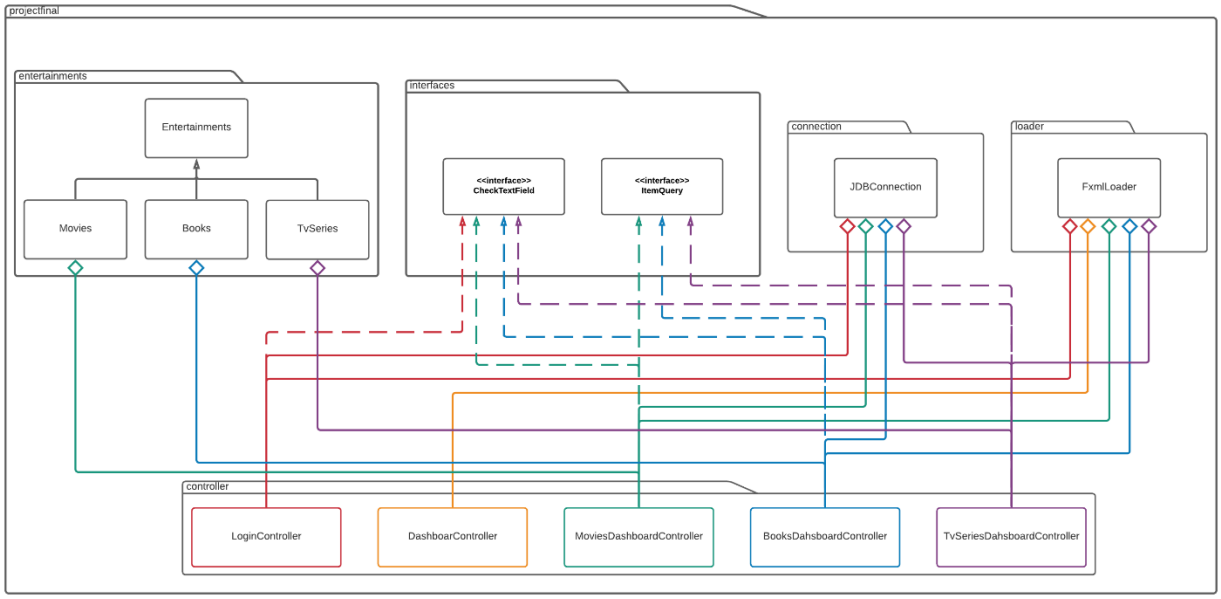


Image 7 – Simple UML Diagram

(simpleDiagram.png)

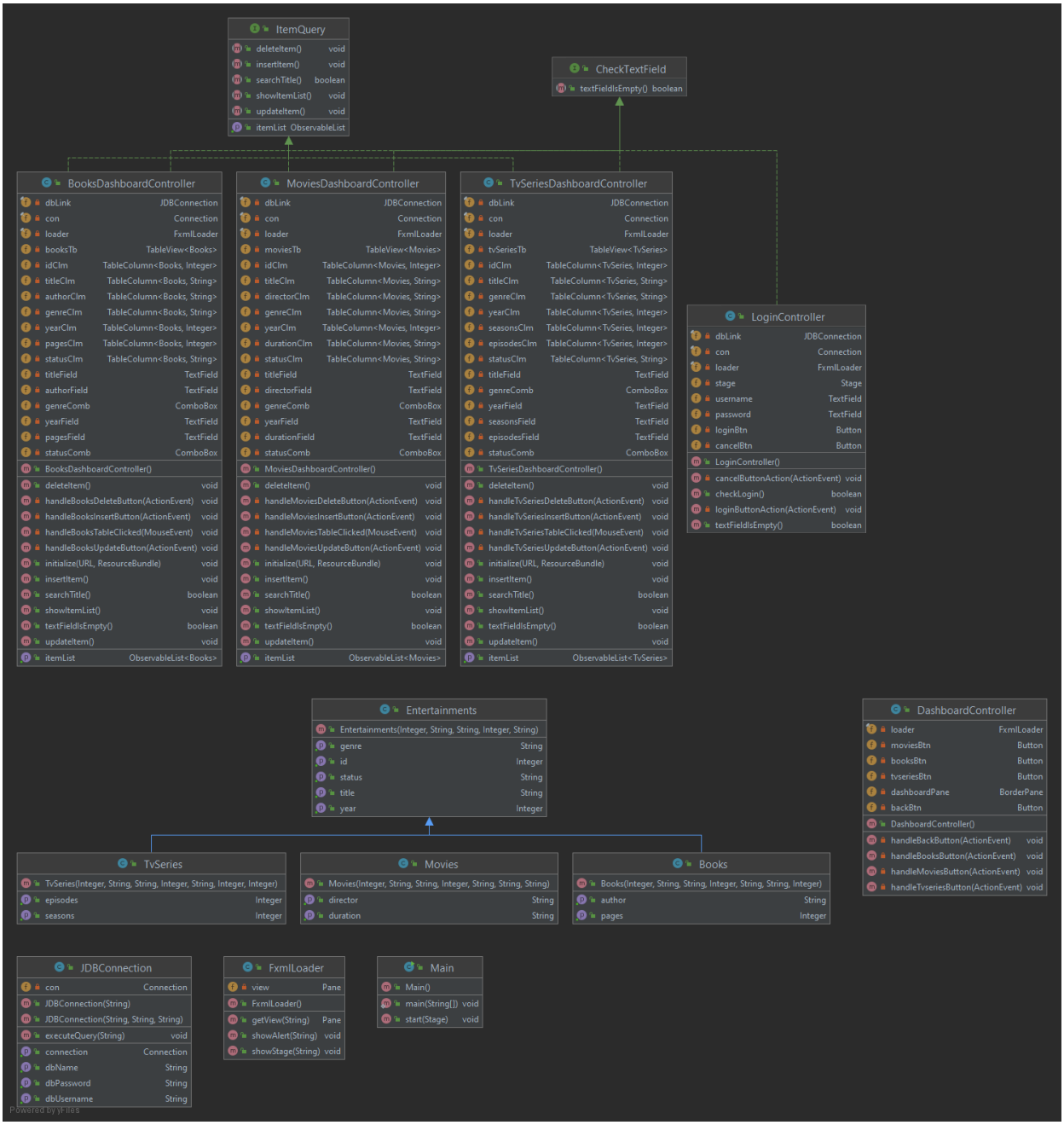


Image 8 – Complete diagram

(completeDiagram.png)

4. Visual Design

The visual design for this application is simple and straightforward using solid color for the appearance

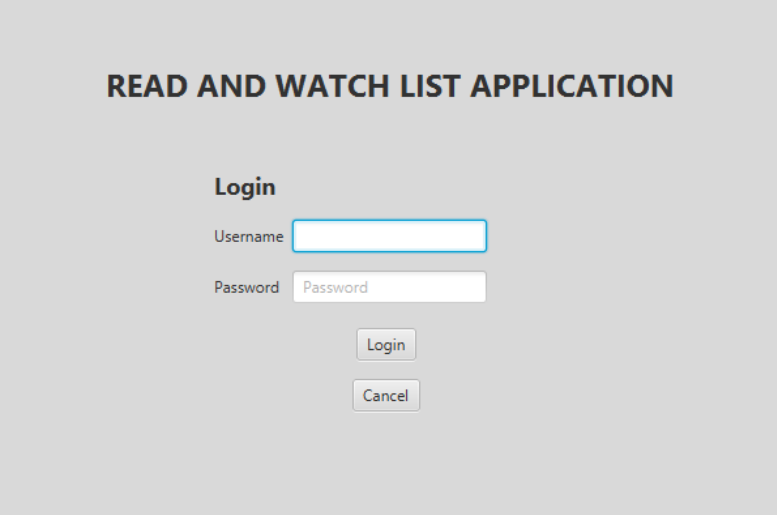


Image 9 – Login menu

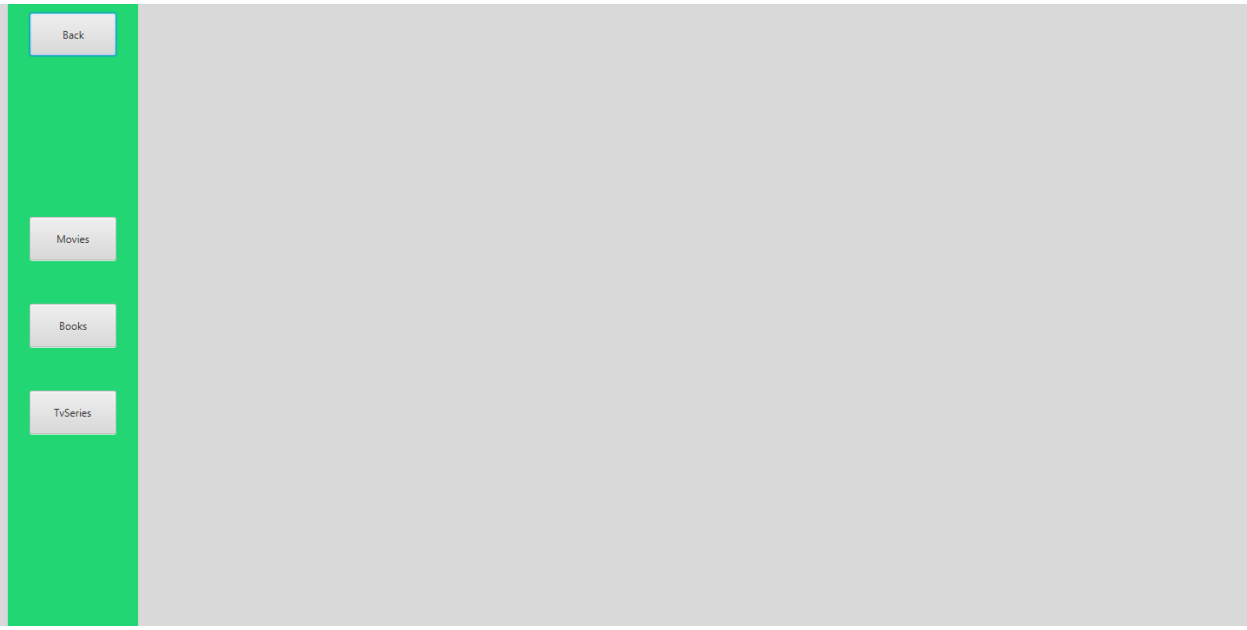


Image 10 – Application's dashboard

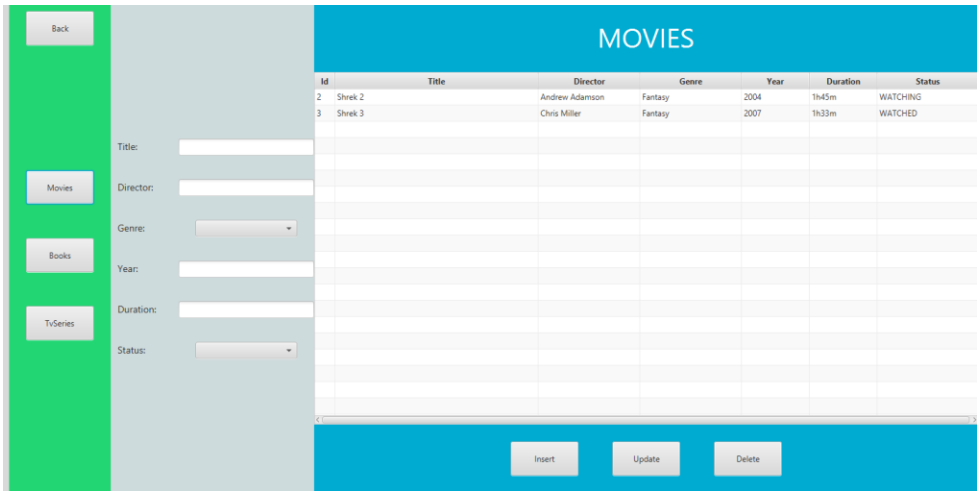


Image 11 – Movies dashboard

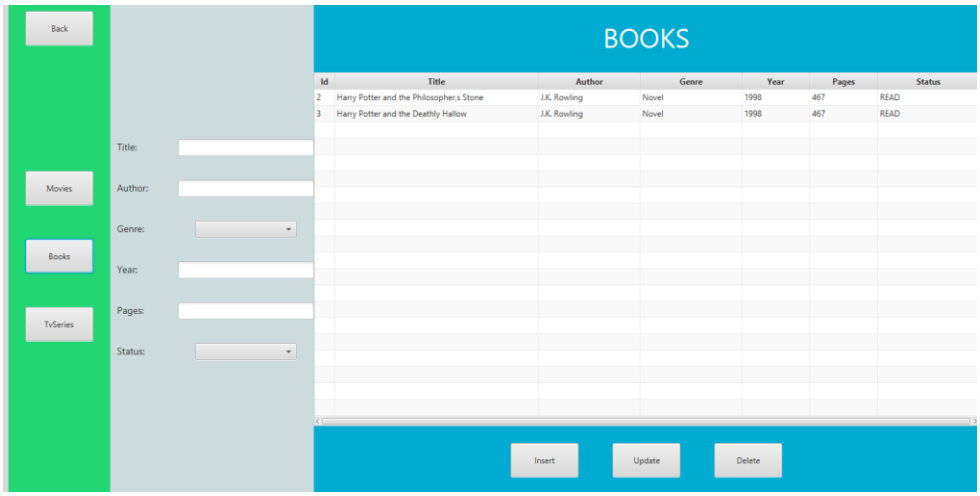


Image 12 – Books dashboard

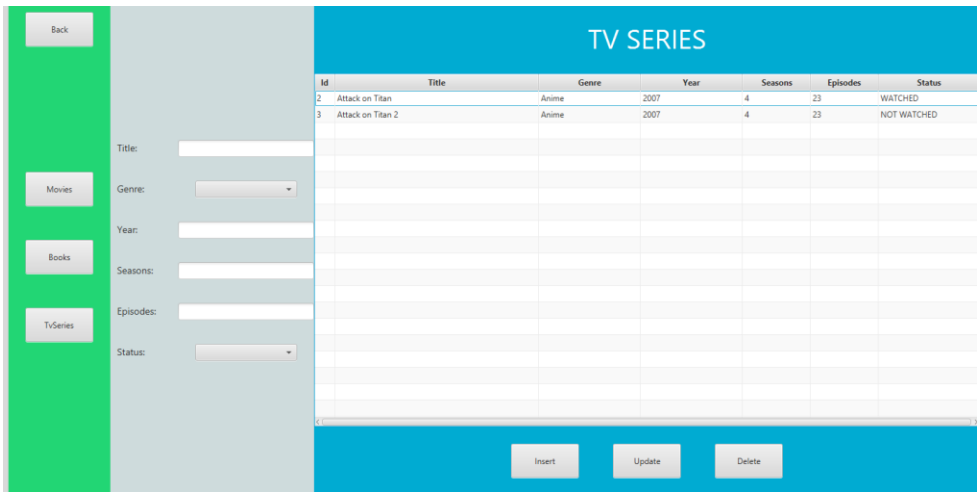


Image 13 – TvSeries dashboard

Implementation

1. Main

```java
// base package
package projectfinal;

// Importing required module, libary, and package
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

// Main class extends Application
public class Main extends Application {

    // Start the JavaFX application
    @Override
    public void start(Stage stage) throws Exception {
        // Create root fxml
        Parent root = FXMLLoader.load(getClass().getResource("/projectfinal/fxml/Login.fxml"));
        // Set stage
        Scene scene = new Scene(root);
        stage.setScene(scene);
        // Show stage
        stage.show();
    }

    public static void main(String[] args) {
        // Launch application
        launch(args);
    }
}
```

Firstly, I created the main class for this project and this class extends from the JavaFX Application class. To start the application, I need to implement the start() from the Application to start the application. In the start(), I set the parent root of the initial fxml file that I want to load for this case it is the Login.fxml file. Then I create a new Scene object of root and set the scene to the current stage. Finally, I can show the stage and launch the application.

2. Interfaces

1. CheckTextField

```java
// interfaces package
package projectfinal.interfaces;

// ChectTextField interface
public interface CheckTextField {
    // Declare methods
    public boolean textFieldIsEmpty();
}
```

CheckTextField interface that has textFieldIsEmpty method that will be used later on by different class.

2. ItemQuery

```java
// interfaces package
package projectfinal.interfaces;

// Importing required module, libary, and package
import javafx.collections.ObservableList;

// ItemQuery interface
public interface ItemQuery {
    // Declare methods
    public ObservableList getItemList();
    public void showItemList();
    public void insertItem();
    public void updateItem();
    public void deleteItem();
    public boolean searchTitle();
}
```

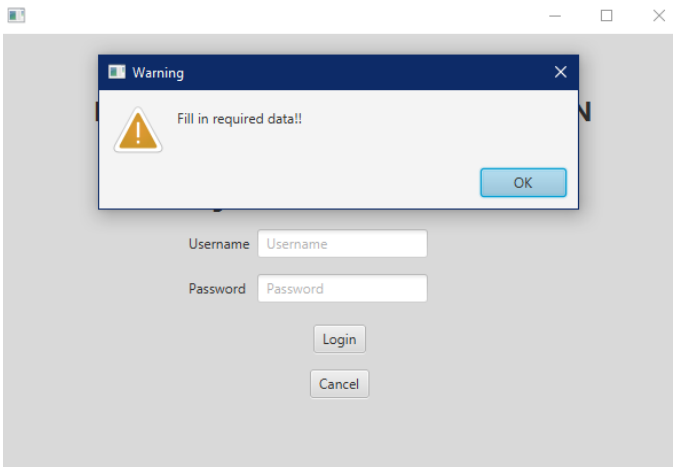
ItemQuery interface that has getItemList, showItemList, insertItem, updateItem, deleteItem, and searchTitle methods that will be used later on by different class.

3. Classes

1. FxmlLoader

```java
// classes.loader package
package projectfinal.classes.loader;

// Importing required module, libary, and package
import java.io.IOException;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;

// FxmlLoader class
public class FxmlLoader {

    // Set class variables
    private Pane view;


    // Method to get view of the provided file URL
    public Pane getView(String file) throws IOException {
        view = FXMLLoader.load(getClass().getResource(file));
        return view;
    }

    // Method to show the stage of the provided file URL
    public void showStage(String file) throws IOException {
        Parent root = FXMLLoader.load(getClass().getResource(file));
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();
    }

    // Method to show alert with the provided message
    public void showAlert(String message){
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.setTitle("Warning");
        alert.setContentText(message);
        alert.setHeaderText(null);
        alert.showAndWait();
    }
}
```

I created this FxmlLoader class to handle the application's GUI and fxml files. I created 3 method in this class, the first one is getView() and this method is used for loading the Pane view of the fxml URL file provided and return the view so it can be used later on to show other fxml inside an fxml. Similar to getView(), showStage() is also used to load other fxml file with the provided URL but the differences is that this fxml file is saved as Stage object instead of Pane object and this method also directly show the new Stage and return nothing. This showStage() is used to open another fxml file in other window instead of loading it inside another fxml file like getView(). The last method that I created in this FxmlLoader file is showAlert() and it is a method that takes an error message of type String and the purpose of this method is to show an alert pop up window when something wrong happen for example if the user doesn't fill all of the required data, it will show a pop up saying "Fill in required data!!".



2. JDBConnection

```java
// classes.connection package
package projectfinal.classes.connection;

// Importing required module, libary, and package
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

// JDBConnection class
public class JDBConnection {

    // Set class variables
    private String dbName;
    private String dbUsername;
    private String dbPassword;
    private Connection con;

    // Constructor 1
    public JDBConnection(String dbName, String dbUsername, String dbPassword) {
        this.dbName = dbName;
        this.dbUsername = dbUsername;
        this.dbPassword = dbPassword;
    }

    // Constructor 2
    public JDBConnection(String dbName) {
        this.dbName = dbName;
        // Set default username and password if not provided
        this.dbUsername = "root";
        this.dbPassword = "";
    }
```

Next. I created this JDBConnection class and it is a class that will handle the connection to a database(for this project I use MySQL). There are 4 variables in this class which is dbName, dbUsername, and dbPassword with all of them being a String data type and con variable with Connection data type from sql.Connection. There are 2 constructor in this class and the first one takes dbName, dbUsername, and dbPassword and the second one which only takes dbName as its parameter and use “root” as the default dbUsername and “” as the default dbPassword.

```java
// DbName getter
public String getDbName() {
    return dbName;
}

// DbUsername getter
public String getDbUsername() {
    return dbUsername;
}

// DbPassword getter
public String getDbPassword() {
    return dbPassword;
}

// Method to connect to desired database
public Connection getConnection(){
    try{
        // Try to connect to the databse
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/" + getDbName(), getDbUsername(), getDbPassword());
        return con;
    }
    catch(Exception ex){
        // Display error if can't connect to database
        System.out.println("Error: "+ex.getMessage());
        ex.printStackTrace();
        return null;
    }
}
```

Here is the getter of the class and the first 3 getter for dbName, dbUsername, and dbPassword is the generic getter where it just return back its own variable but for the getConnection(), I use a try and catch statement. First, the program will try to connect to the database using DriverManager.getConnection("jdbc:mysql://localhost:3306/{database name}", {username}, {password}). The port number may change in different machine but it is using port 3306 in my machine. Return the connection if it is successful but if there is something wrong when connecting to the database, it will catch what is the exception and print out the error message with the stack trace and return nothing.

```java
    // Method to execute the provided query
    public void executeQuery(String query){
        // Declare st variable
        Statement st;
        try{
            // Create statement
            st = con.createStatement();
            // Execute the query
            st.executeUpdate(query);
        }
        // Catch exception if query is not correct
        catch(Exception ex){
            // Print stack trace
            ex.printStackTrace();
        }
    }
}
```

The last method in this JDBConnection class is executeQuery(). The purpose of this method is to execute a query given in the parameter to edit the table in the database. Query is a syntax used by SQL to access and show data in a SQL database system. How this method work is

it will try to create a statement and executing the query provided in the parameter. If the query is eligible, SQL will understand them and run them to edit the database but if the query is not eligible or wrong, it will catch the exception and print the stack trace.

3. Entertainments

```java
// classes.entertainments package
package projectfinal.classes.entertainments;

// Entertainments class
public class Entertainments {

    // Set class variables
    private Integer id;
    private String title;
    private String genre;
    private Integer year;
    private String status;

    // Constructor
    public Entertainments(Integer id, String title, String genre, Integer year, String status) {
        this.id = id;
        this.title = title;
        this.genre = genre;
        this.year = year;
        this.status = status;
    }

    // Id getter
    public Integer getId() {
        return id;
    }

    // Title getter
    public String getTitle() {
        return title;
    }

    // Genre getter
    public String getGenre() {
        return genre;
    }

    // Year getter
    public Integer getYear() {
        return year;
    }

    // Status getter
    public String getStatus() {
        return status;
    }
}
```

Next, I created the Entertainments class and this class will be the parent class of Movies, Books, and TvSeries class and these class later on will contain the variables that will be inserted to the table database later on. This entertainments class have 5 variables, id, title, genre, year, and status with a constructor taking every variable as its parameters. I also implement the getter for each variable.

4. Movies, Books, and TvSeries

```java
// classes.entertainments package
package projectfinal.classes.entertainments;

// Movies class extends Entertainments
public class Movies extends Entertainments{

    // Set class variables
    private String director;
    private String duration;

    // Constructor
    public Movies(Integer id, String title, String genre, Integer year, String status, String director, String duration) {
        super(id, title, genre, year, status);
        this.director = director;
        this.duration = duration;
    }

    // Director getter
    public String getDirector() {
        return director;
    }

    // Duration getter
    public String getDuration() {
        return duration;
    }
}
```

```java
// classes.entertainments package
package projectfinal.classes.entertainments;

// Books class extends Entertainments
public class Books extends Entertainments{

    // Set class variables
    private String author;
    private Integer pages;

    // Constructor
    public Books(Integer id, String title, String genre, Integer year, String status, String author, Integer pages) {
        super(id, title, genre, year, status);
        this.author = author;
        this.pages = pages;
    }

    // Author getter
    public String getAuthor() {
        return author;
    }

    // Pages getter
    public Integer getPages() {
        return pages;
    }
}
```

```java
// classes.entertainments package
package projectfinal.classes.entertainments;

// TvSeries class extends Entertainments
public class TvSeries extends Entertainments{

    // Set class variables
    private Integer seasons;
    private Integer episodes;

    // Constructor
    public TvSeries(Integer id,  String title, String genre, Integer year, String status, Integer seasons, Integer episodes) {
        super(id, title, genre, year, status);
        this.seasons = seasons;
        this.episodes = episodes;
    }

    // Seasons getter
    public Integer getSeasons() {
        return seasons;
    }

    // Episodes getter
    public Integer getEpisodes() {
        return episodes;
    }

}

```

As mentioned before, Movies, Books, and TvSeries class are a child class of Entertainments class so they extends from Entertainments. Each of them have two additional variables inside them, director and duration for Movies, author and pages for Books, and finally seasons and episodes for TvSeries. I also include getter for each variable in each class.

4. Controller

1. LoginController

```java
// controller package
package projectfinal.controller;

// Importing required module, libary, and package
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import projectfinal.classes.loader.FxmlLoader;
import projectfinal.classes.connection.JDBConnection;
import projectfinal.interfaces.CheckTextField;

// LoginController class implements CheckTextField
public class LoginController implements CheckTextField{

    // Create new JDBConnection object from projectfinal.classes.connection package
    private final JDBConnection dbLink = new JDBConnection("finalproject");
    // Call getConnection() method from JDBConnection and assign it to con variable
    private final Connection con = dbLink.getConnection();
    // Create new FxmlLoader object from  projectfinal.classes.loader package
    private final FxmlLoader loader = new FxmlLoader();
    // Declare stage variable
    private Stage stage;

    // All FXML components id/variable
    @FXML
    private TextField username;
    @FXML
    private TextField password;
    @FXML
    private Button loginBtn;
    @FXML
    private Button cancelBtn;
```

This is a class called loginController and just like its name, the purpose of this class is to handle and control the fxml and application's appearance and it is also the same for other controller. First, I imported all the necessary library, module, and package and this class will implements CheckTextField interface. This class also have objects from other classes that have mentioned before and that is from JDBConnection to get connection to the database and from FxmlLoader to handle loading other fxml files. There is also variable calles stage with Stage data type and I created all of the fxml related variables.

```java
    // FXML function to handle action if 'Login' button is pressed
    @FXML
    private void loginButtonAction(ActionEvent event) throws IOException {
        // Check if textFieldIsEmpty() == false
        if (!textFieldIsEmpty())
            // Check if checkLogin()) == true
            if(checkLogin()){
                // Show Dashboard.fxml by calling showStage() from FxmlLoader
                loader.showStage("/projectfinal/fxml/Dashboard.fxml");
                // Close current FXML
                stage = (Stage) loginBtn.getScene().getWindow();
                stage.close();
            }
            else{
                // Throw alert
                loader.showAlert("Invalid username / password!!");
            }
        else{
            // Throw alert
            loader.showAlert("Fill in required data!!");
        }
    }

    // FXML function to handle action if 'Cancel' button is pressed
    @FXML
    private void cancelButtonAction(ActionEvent event) {
        // Close current FXML
        stage = (Stage) cancelBtn.getScene().getWindow();
        stage.close();
    }
```

This next two methods that handle action of certain button if pressed. The first method is to handle login button action and it will check if the text field empty or not and if it is empty, throw an alert and if it is not empty, check if username and password exist in table called “users” or not and if it doesn’t exist, throw another alert. After checking the user’s username and password and if they are correct, close current fxml and open Dashboard.fxml. The second method is to close the application if cancel button is pressed.

```java
    // Function to check if username and password are available in the database or no
    public boolean checkLogin(){
        // Set query
        String query = "SELECT count(1) FROM users WHERE username = '" + username.getText() + "' AND password = '" + password.getText() + "'";
        // Declare st and rs variable
        Statement st;
        ResultSet rs;

        // Try the query
        try{
            // Create statement
            st = con.createStatement();
            // Execute the query
            rs = st.executeQuery(query);
            // While there is/are column(s) in the table
            while(rs.next()){
                // Check if value in first column in database == 1
                if(rs.getInt(1) == 1){
                    return true;
                }
            }
        }
        // Catch exception if query is not correct
        catch(Exception ex){
            // Print stack trace
            ex.printStackTrace();
        }
        return false;
    }

    // Function that override from CheckTextField in projectfinal.interfaces
    // Function to check if text field is empty or not
    @Override
    public boolean textFieldIsEmpty() {
        return username.getText().isEmpty() == true && password.getText().isEmpty() == true;
    }
}
```

This next two methods is basically the methods to check if username and password exist in the table and check if input fields empty or not respectively. The textFieldIsEmpty method overrides and implements the CheckTextField interface.

2. DashboardController

```java
// controller package
package projectfinal.controller;

// Importing required module, libary, and package
import java.io.IOException;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;
import projectfinal.classes.loader.FxmlLoader;

// DashboardController class
public class DashboardController {

    // Create new FxmlLoader object from  projectfinal.classes.loader package
    private final FxmlLoader loader = new FxmlLoader();

    // All FXML components id/variable
    @FXML
    private Button moviesBtn;
    @FXML
    private Button booksBtn;
    @FXML
    private Button tvseriesBtn;
    @FXML
    private BorderPane dashboardPane;
    @FXML
    private Button backBtn;
    @FXML
```

This is the class that control the dashboard and the class variables is similar to LoginController like it have fxml related variables and loader from FxmlLoader but the differences is that this class doesn't implement any interface, DashboardController doesn't need an object from JDBConnection like in LoginController since this class doesn't need any database connection , and this class doesn't need Stage.

```
    @FXML
    // FXML function to handle action if 'Movies' button is pressed
    private void handleMoviesButton(ActionEvent event) throws IOException {
        // Show MoviesDashboard.fxml inside Dashboard.fxml
        Pane view = loader.getView("/projectfinal/fxml/MoviesDashboard.fxml");
        dashboardPane.setCenter(view);
    }

    // FXML function to handle action if 'Books' button is pressed
    @FXML
    private void handleBooksButton(ActionEvent event) throws IOException {
        // Show BooksDashboard.fxml inside Dashboard.fxml
        Pane view = loader.getView("/projectfinal/fxml/BooksDashboard.fxml");
        dashboardPane.setCenter(view);
    }

    // FXML function to handle action if 'Tvseries' button is pressed
    @FXML
    private void handleTvseriesButton(ActionEvent event) throws IOException{
        // Show TvSeriesDashboard.fxml inside Dashboard.fxml
        Pane view = loader.getView("/projectfinal/fxml/TvSeriesDashboard.fxml");
        dashboardPane.setCenter(view);
    }

    // FXML function to handle action if 'Back' button is pressed
    @FXML
    private void handleBackButton(ActionEvent event) throws IOException {
        // Close current FXML
        Stage stage = (Stage) backBtn.getScene().getWindow();
        stage.close();
        // Show Login.fxml by calling showStage() from FxmlLoader
        loader.showStage("/projectfinal/fxml/Login.fxml");
    }
}
```

These methods are to handle actions of each button pressed. If movies button is pressed, open MoviesDashboard.fxml to the current fxml, If books button is pressed, open BooksDashboard.fxml to the current fxml, If tvseries button is pressed, open TvSeriesDashboard.fxml to the current fxml, and go back to login menu if back button is pressed.

3. MoviesDashboardController

```java
// controller package
package projectfinal.controller;

// Importing required module, libary, and package
import ...21 lines

// MoviesDashboardController class implements Initializable, CheckTextField, and Item Query
public class MoviesDashboardController implements Initializable, CheckTextField, ItemQuery {

    // Create new JDBConnection object from projectfinal.classes.connection package
    private final JDBConnection dbLink = new JDBConnection("finalproject");
    // Call getConnection() method from JDBConnection and assign it to con variable
    private final Connection con = dbLink.getConnection();
    // Create new FxmlLoader object from  projectfinal.classes.loader package
    private final FxmlLoader loader = new FxmlLoader();

    // All FXML components id/variable
    @FXML
    private TableView<Movies> moviesTb;
    @FXML
    private TableColumn<Movies, Integer> idClm;
    @FXML
    private TableColumn<Movies, String> titleClm;
    @FXML
    private TableColumn<Movies, String> directorClm;
    @FXML
    private TableColumn<Movies, String> genreClm;
    @FXML
    private TableColumn<Movies, Integer> yearClm;
    @FXML
    private TableColumn<Movies, String> durationClm;
    @FXML
    private TableColumn<Movies, String> statusClm;
    @FXML
    private TextField titleField;
    @FXML
    private TextField directorField;
    @FXML
    private ComboBox genreComb;
    @FXML
    private TextField yearField;
    @FXML
    private TextField durationField;
    @FXML
    private ComboBox statusComb;
```

Just like LoginController, this class implements CheckTextField interface, have object from JDBConnection and FxmlLoader, and have fxml related variables, but this class also implements additional interfaces, which is Initializable and ItemQuery.

```java
    // FXML function to handle action if 'Insert' button is pressed
    @FXML
    private void handleMoviesInsertButton(ActionEvent event) {
        // Check if textFieldIsEmpty() == true
        if(textFieldIsEmpty()){
            // Throw alert
            loader.showAlert("Fill in required data!!");
        }
        // Check if searchTitle() == true
        else if(searchTitle()){
            // Throw alert
            loader.showAlert("Title already exist!!");
        }
        else{
            // Insert item into table by calling insertItem()
            insertItem();
            // Throw alert
            loader.showAlert("Item successfully inserted");
        }
    }

    // FXML function to handle action if 'Update' button is pressed
    @FXML
    private void handleMoviesUpdateButton(ActionEvent event) {...17 lines }

    // FXML function to handle action if 'Delete' button is pressed
    @FXML
    private void handleMoviesDeleteButton(ActionEvent event) {...18 lines }

    // FXML function to handle action if a row in the table is clicked
    @FXML
    private void handleMoviesTableClicked(MouseEvent event) {
        // Set text fields with the item available in the row
        Movies movies = moviesTb.getSelectionModel().getSelectedItem();
        titleField.setText(movies.getTitle());
        directorField.setText(movies.getDirector());
        yearField.setText("" + movies.getYear());
        durationField.setText(movies.getDuration());
    }
```

These are the method that will handle clicked action for insert, update, delete button, and table row. After the insert button is pressed, throw alert if text fields are empty and throw alert if title already existed in the table. For update and delete button, instead of throwing alert if title exist, it will throw alert if title NOT existed in the table. If any row in the table is pressed, it will print the value of each column back to the text fields.

```java
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // Set items in 'Genre' combo box
        ObservableList<String> genreList = FXCollections.observableArrayList("Action", "Drama", "Animated", "Sci-fi", "Fantasy");
        genreComb.setItems(genreList);
        // Set items in 'Status' combo box
        ObservableList<String> statusList = FXCollections.observableArrayList("NOT WATCHED", "WATCHING", "WATCHED");
        statusComb.setItems(statusList);
        // Show the table by calling showItemList()
        showItemList();
    }
```

This is the method that initialize and do actions when the fxml first showed in the screen and the actions are setting the combo box value and show the table.

```java
// Function that override from CheckTextField in projectfinal.interfaces
// Function to check if text field is empty or not
@Override
public boolean textFieldIsEmpty() {
    return titleField.getText().isEmpty() == true || directorField.getText().isEmpty() == true
            || yearField.getText().isEmpty() == true || durationField.getText().isEmpty() == true;
}
```

This is a method that will return true if there are any empty text field.

```java
@Override
public ObservableList<Movies> getItemList(){
    // Create movie list
    ObservableList<Movies> movieList = FXCollections.observableArrayList();
    // Set query
    String query = "SELECT * FROM movies";
    // Declare st and rs variable
    Statement st;
    ResultSet rs;

    // Try the query
    try{
        // Create statement
        st = con.createStatement();
        // Execute the query
        rs = st.executeQuery(query);
        // While there is/are column(s) in the table
        while(rs.next()){
            // Create new Movies object from projectfinal.classes.classes package
            Movies movies = new Movies(rs.getInt("id"), rs.getString("title"), rs.getString("genre"), rs.ge
            // add movies to movie list
            movieList.add(movies);
        }
    }
    // Catch exception if query is not correct
    catch(Exception ex){
        // Print stack trace
        ex.printStackTrace();
    }
    return movieList;
```

This method is to get item list from the table and how it works is first, create movieList variable of type Observable list. Next, create a query string and statement and execute that try to execute the string query statement. If there are rows in the table, create movies object with inputted data in the text field as its value then add them to the observable list and return it. If the string query is not correct catch the exception and print stack trace.

```java
@Override
public void showItemList(){
    // Create list by calling getItemList()
    ObservableList<Movies> list = getItemList();
    // Set cell value factory of each table's cell
    idClm.setCellValueFactory(new PropertyValueFactory<>("id"));
    titleClm.setCellValueFactory(new PropertyValueFactory<>("title"));
    directorClm.setCellValueFactory(new PropertyValueFactory<>("director"));
    genreClm.setCellValueFactory(new PropertyValueFactory<>("genre"));
    yearClm.setCellValueFactory(new PropertyValueFactory<>("year"));
    durationClm.setCellValueFactory(new PropertyValueFactory<>("duration"));
    statusClm.setCellValueFactory(new PropertyValueFactory<>("status"));
    // Set items in the table
    moviesTb.setItems(list);
}
```

This method is to show the item list from the table and how it works is first, create a list of item by calling getItemList(). After that, set cell value factory of each table column with the corresponding column name and then set the items into the table.

```
216         @Override
217         public void insertItem(){
218             // Set query
219             String query = "INSERT INTO movies VALUES (NULL, '" + titleField.getText() + "','" + dire
220                     + genreComb.getSelectionModel().getSelectedItem().toString() + "'," + year
221             // Execute the query by calling executeQuery() from JDBConnection
222             dbLink.executeQuery(query);
223             // Show the table by calling showItemList()
224             showItemList();
225         }
226 
227         // Function that override from ItemQuery in projectfinal.interfaces
228         // Function to update items in the table
229         @Override
230         public void updateItem(){
231             // Set query
232             String query = "UPDATE  movies SET title  = '" + titleField.getText() + "', director = '"
233                     yearField.getText() + ", duration = '" + durationField.getText() + "', status = '
234             // Execute the query by calling executeQuery() from JDBConnection
235             dbLink.executeQuery(query);
236             // Show the table by calling showItemList()
237             showItemList();
238         }
239 
240         // Function that override from ItemQuery in projectfinal.interfaces
241         // Function to delete items from the table
242         @Override
243         public void deleteItem(){
244             // Set query
245             String query = "DELETE FROM movies WHERE title = '" + titleField.getText() + "'";
246             // Execute the query by calling executeQuery() from JDBConnection
247             dbLink.executeQuery(query);
248             // Show the table by calling showItemList()
249             showItemList();
250         }
```

Insert, update, and deleteItem methods are actually the same the only difference is the string query in each method. How it works is that it will edirectly execute the string query and show the changes into the table.

```
254         @Override
255         public boolean searchTitle() {
256             // Set query
257             String query = "SELECT count(1) FROM movies WHERE title = '" + titleField.getText() + "'";
258             // Declare st and rs variable
259             Statement st;
260             ResultSet rs;
261 
262             // Try the query
263             try{
264                 // Create statement
265                 st = con.createStatement();
266                 // Execute the query
267                 rs = st.executeQuery(query);
268                 // While there is/are column(s) in the table
269                 while(rs.next()){
270                     // Check if value in first column in databse == 1
271                     if(rs.getInt(1) == 1){
272                         return true;
273                     }
274                 }
275             }
276             // Catch exception if query is not correct
277             catch(Exception ex){
278                 // Print stack trace
279                 ex.printStackTrace();
280             }
281             return false;
282         }
283     }
```

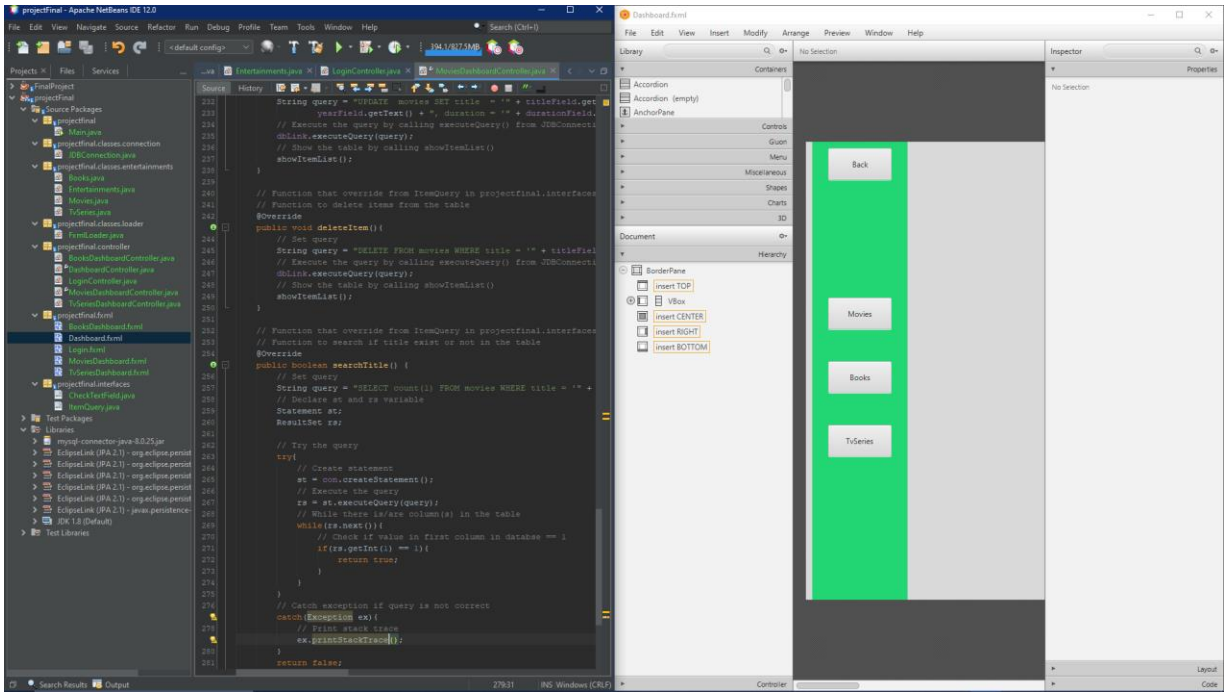
Lastly, this is the method to search title and return true if title exist in the table and return false if it doesn't

4. Books and TvSeriesDashboardController

For BooksDashboardController and TvSeriesDashboardController class, they are exactly the same with the MoviesControllerClass and what's different is that in get item list, instead of creating new Movies object it will create Books or TvSeries object instead and the string query will be different because in MoviesDashboardController, the string query will edit "movies" table but it will edit "books" table in BooksDashboardController, and it will edit "tvseries" table in TvSeriesDashboardController.

Evidence Of working Program

Every screenshot is taken from my IDE and taken by myself



Summary and Self Reflection

This project is a fun project to do since I also learn more about databases and SQL language. It is also a challenging project for me to work on but it is actually enjoyable and it allows me to understand more about object oriented programming in Java language.

Resource and Reference

Reference for some of the code and tutorial for this project:

https://www.youtube.com/watch?v=CGWRwpeihE8&t=283s

https://www.youtube.com/watch?v=ejwzueIZo70

https://www.youtube.com/watch?v=rKv8eavrAio

https://www.youtube.com/watch?v=HJC_JxpHTeU

https://www.youtube.com/watch?v=5yQbt6lYRqk&t=708s

https://www.youtube.com/watch?v=J0IE5LRyzx8