Project Object Oriented Visual Programming

Movie Ticket Management Java Application

Name : Abdurrahman Khairi

PU ID : 001202300163

Class : Informatics 4

Lecturer :

Ms. Deffa Rahadian

MAJOR INFORMATICS

FACULTY COMPUTING

PRESIDENT UNIVERSITY

ACADEMIC YEAR 2023/2024

TABLE OF CONTENTS

TABLE OF IMAGES

BAB I
INTRODUCTION

1.1 BACKGROUND

In today's dynamic world, where time is a precious commodity, the demand for efficient and convenient solutions has permeated every aspect of our lives. The entertainment industry, particularly the world of cinema, is no exception to this trend. Movie enthusiasts eagerly seek seamless and hassle-free ways to indulge in their passion for captivating cinematic experiences.

However, the traditional methods of movie ticket booking often fall short of meeting these expectations. The manual process of purchasing tickets at cinema counters can be time-consuming and inconvenient, often leading to long queues and frustration among eager moviegoers. Additionally, the lack of comprehensive and readily available movie information on traditional ticketing platforms often leaves users struggling to make informed decisions about their cinematic choices.

To address these shortcomings and elevate the movie-watching experience, innovative solutions are urgently needed. The development of a user-friendly, informative, and feature-rich movie ticketing application has emerged as a promising approach to revolutionize the way we access and enjoy the magic of cinema.

1.2 PROBLEM STATEMENT

The current movie ticketing landscape is plagued by several fundamental issues that hinder the overall user experience:

a. **Manual Booking System:** The archaic process of purchasing movie tickets manually at cinema counters remains a significant inconvenience, often resulting in lengthy queues

b. and wasted time for movie enthusiasts. This outdated system fails to align with the fast-paced nature of modern life, where efficiency and convenience are paramount.
c. **Lack of Comprehensive Movie Information:** Traditional ticketing platforms often lack the depth and breadth of movie information that users seek to make informed decisions about their cinematic choices. Essential details such as synopses, trailers, ratings, and user reviews are often scarce or inaccessible, leaving users with limited insights into the films they consider watching.
d. **Limited Movie Selection:** The range of movies available on traditional ticketing platforms often falls short of meeting the diverse preferences of moviegoers. The lack of variety restricts users' options and hinders their ability to discover films that align with their interests and tastes.

These challenges collectively create a frustrating and unsatisfactory experience for movie enthusiasts, diminishing the overall enjoyment of their cinematic endeavors.

1.3 OBJECTIVE

Driven by the desire to transform the movie-watching experience, this project aims to develop a user-friendly, informative, and feature-rich movie ticketing application that addresses the shortcomings of traditional ticketing methods. The application will empower users to seamlessly book tickets, access comprehensive movie information, and discover a diverse range of films. Here's a breakdown of the key objectives:

a. **Simplify Movie Ticket Booking:** The application will streamline the ticket booking process, enabling users to purchase tickets quickly and conveniently.
b. **Enhance Movie Information Accessibility:** Provide users with detailed synopses, trailers, ratings, and reviews to make informed choices.
c. **Expand Movie Selection:** Offer a vast selection of movies from various genres and countries, catering to diverse preferences.
d. **Increase User Convenience and Satisfaction:** Ultimately, create a seamless and enjoyable movie-watching experience through streamlined booking, rich information, and diverse film choices.

e. PROJECT BENEFITS

The implementation of this innovative movie ticketing application will bring forth a multitude of benefits for users, including:

a. **Simplified Movie Ticket Booking:** The application will streamline the ticket booking process, enabling users to purchase tickets quickly and conveniently from the comfort of their own devices. This eliminates the need for lengthy queues at cinema counters, saving users valuable time and effort.

b. **Comprehensive Movie Information:** The application will serve as a comprehensive resource for movie information, providing users with detailed synopses, engaging trailers, informative ratings, and insightful user reviews. This wealth of information will empower users to make informed decisions about their cinematic choices, ensuring they select films that align with their preferences and expectations.

c. **Wide Variety of Films:** The application will offer an extensive selection of movies from various genres and countries, catering to the diverse tastes and interests of movie enthusiasts. This wide array of choices will enable users to discover new films, explore different cinematic styles, and expand their horizons in the world of cinema.

d. **Enhanced User Convenience and Satisfaction:** By addressing the limitations of traditional ticketing methods, the application will significantly enhance user convenience and satisfaction. The streamlined booking process, comprehensive movie information, and diverse film selection will collectively create a seamless and enjoyable movie-watching experience for all users.

f. APPLICATION

This versatile movie ticketing application will cater to a broad spectrum of users, including:

a. **Movie Enthusiasts:** Avid moviegoers who seek convenient and hassle-free ways to book tickets and access comprehensive movie information.
b. **Families:** Families seeking a fun and engaging way to enjoy movies together, with the ease of booking tickets and selecting films that appeal to all ages.
c. **Students:** Students on a budget who appreciate the convenience and affordability of booking tickets through the application.
d. **General Users:** Individuals who enjoy watching movies and appreciate the wide variety of films available on the application.

The application's accessibility across various devices, including smartphones, tablets, and computers, will further enhance its usability and broaden its reach, allowing users to access its features anytime, anywhere.

g. LINK APPLICATION

Github :

https://github.com/abdurrahmankhairii/laravel_project.git

BAB II
THEORITICAL FRAMEWORK

2.1 JAVA NETBEANS

2.1.1 Rationale for Selecting Java NetBeans

The decision to utilize Java NetBeans as the primary development environment for this final project was driven by several compelling factors:

a. **Robustness and Versatility:** Java is a widely recognized and mature programming language renowned for its object-oriented paradigm, platform independence, and extensive security features. Its versatility makes it suitable for developing a wide range of applications, including complex enterprise systems.
b. **NetBeans IDE's Comprehensive Features:** NetBeans IDE offers a comprehensive suite of tools and functionalities that streamline the development process, enhancing productivity and efficiency. Its intuitive graphical user interface, code completion capabilities, and debugging tools significantly expedite the development cycle.
c. **Seamless Integration with MySQL Database:** MySQL is a popular open-source relational database management system known for its reliability, scalability, and ease of use. NetBeans IDE provides seamless integration with MySQL, enabling effortless data management and manipulation within the application.

2.1.2 Key Advantages of Java NetBeans

The combination of Java and NetBeans IDE offers a multitude of advantages for developing this movie ticketing application:

a. **Simplified Development:** Java's object-oriented programming approach and NetBeans IDE's intuitive tools facilitate efficient coding and rapid prototyping.
b. **Robust Application:** Java's inherent security features and NetBeans IDE's debugging tools ensure the application's stability and reliability.

c. **Scalable Solution:** Java's scalability and NetBeans IDE's modular design enable the application to accommodate future growth and enhancements.
d. **Cross-Platform Compatibility:** Java's platform independence ensures the application runs seamlessly across various operating systems, expanding its reach.

2.1.3 Specific Features of NetBeans IDE 21

NetBeans IDE 21 provides a plethora of features that significantly enhance the development experience:

a. **Intuitive Graphical User Interface:** The user-friendly interface simplifies navigation and access to development tools.
b. **Advanced Code Completion:** Intelligent code completion suggestions streamline coding and reduce errors.
c. **Refactoring Tools:** Powerful refactoring tools facilitate code restructuring and maintainability.
d. **Integrated Debugging Tools:** Comprehensive debugging tools aid in identifying and resolving software defects.
e. **Version Control Integration:** Seamless integration with version control systems like Git promotes collaboration and code management.

2.1.4 MySQL Connector 8.0.25 Integration

MySQL Connector 8.0.25 plays a crucial role in enabling effective database interaction:

a. **Database Connectivity:** Establishes a robust connection between the application and the MySQL database.
b. **Data Manipulation:** Facilitates efficient data retrieval, insertion, updates, and deletions within the application.
c. **Database Administration:** Provides tools for managing database schema, users, and permissions.

2.2 DATABASE CONNECTION

2.2.1 AdminUser

a. MySQL Code

Jalankan perintah SQL pada basis data mydatabase:

```
CREATE TABLE adminuser (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

Image 2.2.1.1 mySQL Code Table AdminUser

b. MySQL Table

		user_id	username	password
☐	Ubah Salin Hapus	1	khairiadmin	adminpassword
☐	Ubah Salin Hapus	3	vikoadmin	vikoadmin
☐	Ubah Salin Hapus	4	ragiladmin	ragiladmin
☐	Ubah Salin Hapus	5	mamdeffa	mamdeffa

Image 2.2.1.2 Database Table AdminUser

c. Connection Java Netbeans

```
public static void insertAdminOrNormalUser(String username, String pass, int idx) {
    try {
        Connection myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert;
        if (idx == 1) {
            insert = "INSERT INTO mydatabase.adminuser (username, password) VALUES ('" + username + "', '" + pass + "');";
        } else {
            insert = "INSERT INTO mydatabase.normaluser (username, password) VALUES ('" + username + "', '" + pass + "');";
        }
        myStmt.executeUpdate(insert);
        myConn.close();
    } catch (SQLException exc) {
        System.out.println("Nothing found");
    }
}
```

Image 2.2.1.3 Database Connection Insert Admin or Normal User

```java
public static boolean adminOrNormalExists(String username, int idx) {
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert;
        if (idx == 1) {
            insert = "SELECT * FROM mydatabase.adminuser WHERE username = '" + username + "';";
        } else {
            insert = "SELECT * FROM mydatabase.normaluser WHERE username = '" + username + "';";
        }
        ResultSet myRs = myStmt.executeQuery(insert);
        boolean found = false;
        while (myRs.next()) {
            found = true;
        }
        myConn.close();
        return found;
    } catch (SQLException exc) {
        System.out.println("Nothing found");
        return false;
    } finally {
        try {
            myConn.close();
        } catch (SQLException ex) {
            Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Image 2.2.1.4 Database Connection Admin or Normal Exist

```java
public static boolean validAdminOrNormal(String username, String password, int idx) {
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/moviedatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert;
        if (idx == 1) {
            insert = "SELECT * FROM mydatabase.adminuser WHERE username = '" + username + "';";
        } else {
            insert = "SELECT * FROM mydatabase.normaluser WHERE username = '" + username + "';";
        }
        ResultSet myRs = myStmt.executeQuery(insert);
        boolean found = false;
        String cur = "";
        while (myRs.next()) {
            found = true;
            cur = myRs.getString("password");
        }
        myConn.close();
        return password.equals(cur);
    } catch (SQLException exc) {
        System.out.println("Nothing found");
        return false;
    } finally {
        try {
            myConn.close();
        } catch (SQLException ex) {
            Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Image 2.2.1.5 Database Connection Validation Admin or Normal User

```
159     public static void deleteAdminOrNormal(String username, int idx) {
160         Connection myConn = null;
161         try {
162             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
163             Statement myStmt = myConn.createStatement();
164             String insert;
165             if (idx == 1) {
166                 insert = "DELETE FROM mydatabase.adminuser WHERE username = '" + username + "';";
167             } else {
168                 insert = "DELETE FROM mydatabase.normaluser WHERE username ='" + username + "';";
169             }
170             //System.out.println(insert);
171             myStmt.executeUpdate(insert);
172             myConn.close();
173         } catch (SQLException exc) {
174             System.out.println("Nothing found");
175         } finally {
176             try {
                 myConn.close();
178             } catch (SQLException ex) {
179                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
180             }
181         }
182     }
```

Image 2.2.1.6 Database Connection Delete Admin or Normal User

2.2.2 *NormalUser*

a. MySQL Code

Jalankan perintah SQL pada basis data mydatabase:

```
CREATE TABLE normaluser (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL
);
```

Image 2.2.2.1 mySQL Code Database Table NormalUser

b. MySQL Table

		user_id	username	password
☐	Ubah Salin Hapus	2	moshe	password
☐	Ubah Salin Hapus	3	ragil	ragil
☐	Ubah Salin Hapus	4	vikoadrian	vikoadrian
☐	Ubah Salin Hapus	5	mamdeffa	mamdeffa
☐	Ubah Salin Hapus	6	abdur	password

Image 2.2.2.2 Database Table NormalUser

c. Connection Java Netbeans

```
21
22      public static void insertAdminOrNormalUser(String username, String pass, int idx) {
23          try {
24              Connection myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
25              Statement myStmt = myConn.createStatement();
26              String insert;
27              if (idx == 1) {
28                  insert = "INSERT INTO mydatabase.adminuser (username, password) VALUES ('" + username + "', '" + pass + "');";
29              } else {
30                  insert = "INSERT INTO mydatabase.normaluser (username, password) VALUES ('" + username + "', '" + pass + "');";
31              }
32              myStmt.executeUpdate(insert);
33              myConn.close();
34          } catch (SQLException exc) {
35              System.out.println("Nothing found");
36          }
37      }
38
```

Image 2.2.2.7 Database Connection Insert Admin or Normal User

```java
public static boolean adminOrNormalExists(String username, int idx) {
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert;
        if (idx == 1) {
            insert = "SELECT * FROM mydatabase.adminuser WHERE username = '" + username + "';";
        } else {
            insert = "SELECT * FROM mydatabase.normaluser WHERE username = '" + username + "';";
        }
        ResultSet myRs = myStmt.executeQuery(insert);
        boolean found = false;
        while (myRs.next()) {
            found = true;
        }
        myConn.close();
        return found;
    } catch (SQLException exc) {
        System.out.println("Nothing found");
        return false;
    } finally {
        try {
            myConn.close();
        } catch (SQLException ex) {
            Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Image 2.2.2.8 Database Connection Admin or Normal Exist

```java
public static boolean validAdminOrNormal(String username, String password, int idx) {
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/moviedatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert;
        if (idx == 1) {
            insert = "SELECT * FROM mydatabase.adminuser WHERE username = '" + username + "';";
        } else {
            insert = "SELECT * FROM mydatabase.normaluser WHERE username = '" + username + "';";
        }
        ResultSet myRs = myStmt.executeQuery(insert);
        boolean found = false;
        String cur = "";
        while (myRs.next()) {
            found = true;
            cur = myRs.getString("password");
        }
        myConn.close();
        return password.equals(cur);
    } catch (SQLException exc) {
        System.out.println("Nothing found");
        return false;
    } finally {
        try {
            myConn.close();
        } catch (SQLException ex) {
            Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Image 2.2.2.9 Database Connection Validation Admin or Normal User

```
159     public static void deleteAdminOrNormal(String username, int idx) {
160         Connection myConn = null;
161         try {
162             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
163             Statement myStmt = myConn.createStatement();
164             String insert;
165             if (idx == 1) {
166                 insert = "DELETE FROM mydatabase.adminuser WHERE username = '" + username + "';";
167             } else {
168                 insert = "DELETE FROM mydatabase.normaluser WHERE username ='" + username + "';";
169             }
170             //System.out.println(insert);
171             myStmt.executeUpdate(insert);
172             myConn.close();
173         } catch (SQLException exc) {
174             System.out.println("Nothing found");
175         } finally {
176             try {
178             } catch (SQLException ex) {
179                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
180             }
181         }
182     }
```

Image 2.2.2.10 Database Connection Delete Admin or Normal User

2.2.3 *BannedUser*

a. MySQL Code

Jalankan perintah SQL pada basis data mydatabase:

```
CREATE TABLE banneduser (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE
);
```

Image 2.2.3.1 mySQL Code Database Table BannedUser

b. MySQL Table

	user_id	username
Ubah Salin Hapus	2	anaknakal
Ubah Salin Hapus	1	ernest
Ubah Salin Hapus	4	inibanned
Ubah Salin Hapus	3	pelanggar

Image 2.2.3.2 Database Table BannedUser

c. Connection Java Netbeans

```
112
113     public static void addBannedUser(String name) {
114         Connection myConn = null;
115         try {
116             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
117             Statement myStmt = myConn.createStatement();
118             String insert = "INSERT INTO mydatabase.banneduser (username) VALUES ('" + name + "');";
119             myStmt.executeUpdate(insert);
120             myConn.close();
121         } catch (SQLException exc) {
122             System.out.println("Nothing found");
123         } finally {
124             try {
126                 myConn.close();
126             } catch (SQLException ex) {
127                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
128             }
129         }
130     }
```

Image 2.2.3.3 Database Connection Add Banned User

```
131
132     public static boolean bannedUser(String username) {
133         Connection myConn = null;
134         try {
135             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
136             Statement myStmt = myConn.createStatement();
137             String insert = "SELECT * FROM mydatabase.bannedUser WHERE username = '" + username + "';";
138 
139             ResultSet myRs = myStmt.executeQuery(insert);
140             boolean found = false;
                String cur = "";
142             while (myRs.next()) {
143                 found = true;
144             }
145             myConn.close();
146             return found;
147         } catch (SQLException exc) {
148             System.out.println("Nothing found");
149             return false;
150         } finally {
151             try {
                    myConn.close();
153             } catch (SQLException ex) {
154                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
155             }
156         }
157     }
```

Image 2.2.3.4 Database Connection Banned User

2.2.4 *MovieDetails*

a. MySQL Code

Jalankan perintah SQL pada basis data mydatabase:

```sql
CREATE TABLE moviedetails (
  movie_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL UNIQUE,
  genre VARCHAR(255) NOT NULL,
  director VARCHAR(255) NOT NULL,
  length FLOAT NOT NULL,
  rating FLOAT NOT NULL,
  url VARCHAR(255) NOT NULL
);
```

Image 2.2.4.1 mySQL Code Database Table MovieDetails

b. MySQL Table

	movie_id	name	genre	director	length	rating	url
Ubah Salin Hapus	2	THE AVENGERS	ACTION	ANTHONY RUSSO	213	4.9	https://www.google.com/url?sa=i&url=https%3A%2F%2F...
Ubah Salin Hapus	4	THE MARVELS	ACTION	NIA DACOSTA	105	4.75	D:KuliahMs. Deffa RahadianFINALMovieDatabase□uildc...
Ubah Salin Hapus	6	THE NUN	HORROR	CORIN HARDY	96	9.2	D:KuliahMs. Deffa RahadianFINALMovieDatabase□uildc...
Ubah Salin Hapus	7	MOANA	ANIMATION	JOHN MUSKER	107	4.6	D:KuliahMs. Deffa RahadianFINALMovieDatabase□uildc...

Image 2.2.4.2 Database Connection Table MovieDetails

c. Connection Java Netbeans

```
184     public static void addMovie(String name, String genre, String director, String length, String rating, String url) {
185         if (url.length() == 0) {
186             url = "nothing";
187         }
188         Connection myConn = null;
189         try {
190             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
191             Statement myStmt = myConn.createStatement();
192             String insert = "INSERT INTO mydatabase.moviedetails (name, genre, director, length, rating, url) VALUES ('"
                        + name + "','" + genre + "','" + director + "','" + Float.parseFloat(length) + "','" + Float.parseFloat(rating)
194                     + "', '" + url + "');";
195             //System.out.println(insert);
196             myStmt.executeUpdate(insert);
197             myConn.close();
198         } catch (SQLException exc) {
199             System.out.println("Nothing found");
200         } finally {
201             try {
                    myConn.close();
203             } catch (SQLException ex) {
204                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
205             }
206         }
207     }
```

Image 2.2.4.3 Database Connection Add Movie

```
209     public static boolean checkMovie(String s) {
210         Connection myConn = null;
211         try {
212             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
213             Statement myStmt = myConn.createStatement();
214             String insert;
215             insert = "SELECT * FROM mydatabase.moviedetails WHERE name = '" + s + "';";
216             ResultSet myRs = myStmt.executeQuery(insert);
217             boolean found = false;
218             while (myRs.next()) {
219                 found = true;
220             }
221             myConn.close();
222             return found;
223         } catch (SQLException exc) {
224             System.out.println("Nothing found");
225             return false;
226         } finally {
227             try {
228                 myConn.close();
229             } catch (SQLException ex) {
230                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
231             }
232         }
233     }
```

Image 2.2.4.4 Database Connection Check Movie

```
260     public static void deleteMovieOrRequest(String s, int idx) {
261         Connection myConn = null;
262         try {
263             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
264             Statement myStmt = myConn.createStatement();
265             String insert;
266             if (idx == 1) {
267                 insert = "DELETE FROM mydatabase.moviedetails WHERE name = '" + s + "';";
268             } else {
269                 insert = "DELETE FROM mydatabase.movierequest WHERE moviename = '" + s + "';";
270             }
271             System.out.println(insert);
272             myStmt.executeUpdate(insert);
273             myConn.close();
274         } catch (SQLException exc) {
275             System.out.println("Nothing found");
276         } finally {
277             try {
278                 myConn.close();
279             } catch (SQLException ex) {
280                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
281             }
282         }
283     }
```

Image 2.2.4.5 Database Connection Delete Movie and Request

```java
public static String getMovieDetails(String s) {
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert = "SELECT * FROM mydatabase.moviedetails WHERE name = '" + s + "';";
        ResultSet myRs = myStmt.executeQuery(insert);
        String cur = "";
        while (myRs.next()) {
            cur += myRs.getString("name");
            cur += ",";
            cur += myRs.getString("genre");
            cur += ",";
            cur += myRs.getString("director");
            cur += ",";
            cur += String.valueOf(myRs.getString("length"));
            cur += ",";
            cur += String.valueOf(myRs.getString("rating"));
            cur += ",";
            cur += myRs.getString("url");
        }
        myConn.close();
        return cur;
    } catch (SQLException exc) {
        System.out.println("Nothing found");
    } finally {
        try {
            myConn.close();
        } catch (SQLException ex) {
            Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    return "";
}
```

Image 2.2.4.6 Database Connection Get Show Movie

```java
public static String getGenreOrDirector(String name, int type) {
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert = "SELECT * FROM mydatabase.moviedetails;";
        ResultSet myRs = myStmt.executeQuery(insert);
        String cur = "";
        boolean add = false;
        while (myRs.next()) {
            if ((type == 1 && myRs.getString("genre").equals(name))
                    || (type == 2 && myRs.getString("director").equals(name))) {
                if (add) {
                    cur += "+";
                }
                cur += myRs.getString("name");
                cur += ",";
                cur += myRs.getString("genre");
                cur += ",";
                cur += myRs.getString("director");
                cur += ",";
                cur += String.valueOf(myRs.getString("length"));
                cur += ",";
                cur += String.valueOf(myRs.getString("rating"));
                cur += ",";
                cur += myRs.getString("url");
                add = true;
            }
        }
        myConn.close();
        if (cur.length() == 0) {
            cur = "NOTHING";
        }
        return cur;
    } catch (SQLException exc) {
        System.out.println("Nothing found");
    } finally {
```

Image 2.2.4.7 Database Connection Get Show Genre or Director

```java
public static String getRating(float start, float end) {
    //System.out.println(start + " " + end);
    if (start > end) {
        float x = end;
        end = start;
        start = x;
    }
    Connection myConn = null;
    try {
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
        Statement myStmt = myConn.createStatement();
        String insert = "SELECT * FROM mydatabase.moviedetails;";
        ResultSet myRs = myStmt.executeQuery(insert);
        String cur = "";
        boolean add = false;
        while (myRs.next()) {
            float f = Float.parseFloat(myRs.getString("rating"));
            if (f >= start && f <= end) {
                if (add) {
                    cur += "+";
                }
                cur += myRs.getString("name");
                cur += ",";
                cur += myRs.getString("genre");
                cur += ",";
                cur += myRs.getString("director");
                cur += ",";
                cur += String.valueOf(myRs.getString("length"));
                cur += ",";
                cur += String.valueOf(myRs.getString("rating"));
                cur += ",";
                cur += myRs.getString("url");
                add = true;
            }
        }
        myConn.close();
```

Image 2.2.4.8 Database Connection Get Show Rating

2.2.5 *MovieRequest*

a. MySQL Code

Jalankan perintah SQL pada basis data mydatabase:

```
CREATE TABLE movierequest (
  request_id INT AUTO_INCREMENT PRIMARY KEY,
  moviename VARCHAR(255) NOT NULL UNIQUE
);
```

Image 2.2.5.1 mySQL Code Database Table MovieRequest

b. MySQL Table

	request_id	moviename
Ubah Salin Hapus	2	ANNABEL
Ubah Salin Hapus	1	BIG MOUSE
Ubah Salin Hapus	4	BOBOIBOY GALAXY
Ubah Salin Hapus	7	JALANGKUNG
Ubah Salin Hapus	5	SIKSA KUBUR
Ubah Salin Hapus	6	SIKSA NERAKA
Ubah Salin Hapus	3	UP

Image 2.2.5.2 Database Table MovieRequest

c. Connection Java Netbeans

```
39    public static void addMovieRequest(String name) {
40        try {
              Connection myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
42            Statement myStmt = myConn.createStatement();
43            String insert = "INSERT INTO mydatabase.movierequest (moviename) VALUES ('" + name + "');";
44            myStmt.executeUpdate(insert);
45            myConn.close();
46        } catch (SQLException exc) {
47            System.out.println("Nothing found");
48        }
49    }
```

Image 2.2.5.3 Database Connection Add Request Movie

```
235     public static boolean movieRequestExists(String name) {
236         Connection myConn = null;
237         try {
238             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
239             Statement myStmt = myConn.createStatement();
240             String insert = "SELECT * FROM mydatabase.movierequest WHERE moviename = '" + name + "';";
241             ResultSet myRs = myStmt.executeQuery(insert);
242             boolean found = false;
243             while (myRs.next()) {
244                 found = true;
245             }
246             myConn.close();
247             return found;
248         } catch (SQLException exc) {
249             System.out.println("Nothing found");
250             return false;
251         } finally {
252             try {
                    myConn.close();
254             } catch (SQLException ex) {
255                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
256             }
257         }
258     }
```

Image 2.2.5.4 Database Connection Movie Request Exist

```
260     public static void deleteMovieOrRequest(String s, int idx) {
261         Connection myConn = null;
262         try {
263             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
264             Statement myStmt = myConn.createStatement();
265             String insert;
266             if (idx == 1) {
267                 insert = "DELETE FROM mydatabase.moviedetails WHERE name = '" + s + "';";
268             } else {
269                 insert = "DELETE FROM mydatabase.movierequest WHERE moviename = '" + s + "';";
270             }
271             System.out.println(insert);
272             myStmt.executeUpdate(insert);
273             myConn.close();
274         } catch (SQLException exc) {
275             System.out.println("Nothing found");
276         } finally {
277             try {
                    myConn.close();
279             } catch (SQLException ex) {
280                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
281             }
282         }
283     }
```

Image 2.2.5.5 Database Connection Delete Movie or Request Movie

```
320     public static String requestedMovie() {
321         Connection myConn = null;
322         try {
323             myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDatabase", "root", "");
324             Statement myStmt = myConn.createStatement();
325             String insert = "SELECT * FROM mydatabase.movierequest;";
326             ResultSet myRs = myStmt.executeQuery(insert);
327             String cur = "";
328             boolean add = false;
329             while (myRs.next()) {
330                 if (add) {
331                     cur += ",";
332                 }
333                 cur += myRs.getString("moviename");
334                 add = true;
335             }
336             myConn.close();
337             return cur;
338         } catch (SQLException exc) {
339             System.out.println("Nothing found");
340         } finally {
341             try {
343                 myConn.close();
344             } catch (SQLException ex) {
345                 Logger.getLogger(database.class.getName()).log(Level.SEVERE, null, ex);
346             }
347         }
348         return "";
    }
```

Image 2.2.5.6 Database Connection Get Show Requested Movie

BAB III

OBJECT ORIENTED PROGRAMMING PILLARS

3.1 ENCAPSULATION

Encapsulation promotes data protection by bundling data (attributes) and the methods that operate on that data within a single unit called a class. This concept ensures data integrity and controlled access, safeguarding the application's core functionality.

```
public class user {

    private String username, password;

    public user(String user, String pass) {
        this.username = user;
        this.password = pass;
    }

    public user() {
        this.username = null;
        this.password = null;
    }
```

Image 3.1.1 Encapsulation

In the provided code, there is a class named user that demonstrates the concept of encapsulation. The username and password variables are declared as private, which means they can only be accessed within the user class. The toString() method is an example of polymorphism, as it is overridden to provide a specific implementation for the user class.

However, the code does not demonstrate the concepts of inheritance and abstraction.

Inheritance is the mechanism by which classes in Java, Python, and other OOP languages inherits the attribute of other classes. Abstraction means to only show the necessary details to the user and hide the irrelevant information.

In the user class, the username and password variables are encapsulated, meaning they are hidden from other classes and can only be accessed through the methods of the user class.

The toString() method is an example of polymorphism, as it is overridden to provide a specific implementation for the user class.

The code does not demonstrate the concept of inheritance, as there is no parent class from which the user class inherits attributes or methods. The code also does not demonstrate the concept of abstraction, as it does not hide irrelevant details from the user.

3.2 INHERITANCE

Inheritance allows us to create new classes (subclasses) that inherit properties and functionalities from existing classes (superclasses). This promotes code reusability and enables the creation of specialized classes that extend the functionalities of more general classes.

```
/**
 *
 * @author Abdurrahman Khairi
 */
public class addAdmin extends javax.swing.JFrame {
public class addMovie extends javax.swing.JFrame {
public class adminForm extends javax.swing.JFrame {
public class adminLoginForm extends javax.swing.JFrame {
public class banUser extends javax.swing.JFrame {
public class deleteMovieForm extends javax.swing.JFrame {
public class editMovie extends javax.swing.JFrame {
 */
public class editMovieForm extends javax.swing.JFrame {
public class removeAdmin extends javax.swing.JFrame {
public class requestedMovie extends javax.swing.JFrame {
public class addReview extends javax.swing.JFrame {
public class contactUs extends javax.swing.JFrame {
public class homePage extends javax.swing.JFrame {
public class normalUser extends user {
public class registeruserForm extends javax.swing.JFrame {
public class requestMovie extends javax.swing.JFrame {
public class searchByDirector extends javax.swing.JFrame {
public class searchByGenre extends javax.swing.JFrame {
public class searchByName extends javax.swing.JFrame {
public class searchByRating extends javax.swing.JFrame {
public class searchMovie extends javax.swing.JFrame {
public class userForm extends javax.swing.JFrame {
public class adminOrNormal extends javax.swing.JFrame {
public class allMovie extends javax.swing.JFrame {
public class newShowMovie extends javax.swing.JFrame {
public class showMovie extends javax.swing.JFrame {
```

Image 3.2.1 Inheritance

Explanation Inheritance, for example in addAmin :

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and behavior from a parent class. In the provided Java code, the addAdmin class extends the javax.swing.JFrame class, which is an example of inheritance.

The addAdmin class is the child class, and JFrame is the parent class. By extending JFrame, the addAdmin class automatically inherits all the properties and behavior of JFrame, such as the ability to create a window, set its size and title, and handle events. This allows the addAdmin class to focus on its specific functionality, which is to add a new admin user to the system.

The addAdmin class has a constructor that takes no arguments, which calls the initComponents() method to initialize the user interface components and then sets the default close operation for the window using the setDefaultCloseOperation() method.

The setDefaultCloseOperation() method is inherited from JFrame, and it allows the addAdmin class to set the default close operation for the window without having to define its own implementation. This is an example of how the addAdmin class uses inheritance to inherit behavior from JFrame.

By extending JFrame, the addAdmin class automatically inherits many other properties and behavior, such as the ability to set the window size, location, and title, handle events, and more. This reduces the amount of code that needs to be written in the addAdmin class and makes it easier to maintain and extend.

3.3 POLYMORPHISM

Polymorphism refers to the ability of objects to respond differently to the same method call. This flexibility allows for creating generic methods that can handle various object types, enhancing code adaptability and maintainability.

```
public class user {

    private String username, password;

    public user(String user, String pass) {
        this.username = user;
        this.password = pass;
    }

    public user() {
        this.username = null;
        this.password = null;
    }

    @Override //
    public String toString() {
        return this.username + " " + this.password;
    }

}
```

Image 3.3.1 Polymorphism

toString() is a method inherited from the Object class, which is the superclass of all classes in Java. The toString() method is used to convert an object into a string representation.

By default, the toString() method will produce a string representation of the object that includes the class name and memory address. However, in this case, we want to change the string representation of the user object to a concatenation of username and password.

This is done by overriding the toString() method in the user class. The use of the @Override annotation indicates that we are overriding a method from the superclass.

The use of overriding in this case allows us to get a string representation that is more suitable for our needs. Additionally, by overriding the toString() method, we can avoid writing redundant code to print the attributes of the user object.

3.4 ABSTRACTION

Abstraction is one of the fundamental concepts in object-oriented programming (OOP) that allows us to focus on the essential features of an object while ignoring the irrelevant details. In Java, abstraction can be achieved through abstract classes and interfaces.

An abstract class is a class that cannot be instantiated, but it can be subclassed. An abstract class can contain both abstract and concrete methods. Abstract methods are methods without a body, and they are meant to be overridden by the subclasses.

```
18  public class normalUser extends user {
19
20      protected static boolean normalCheck(String name, String pwd) {
21          File f = new File("D:\\Kuliah\\Ms. Deffa Rahadian\\Project Movie\\Movie Database\\MovieDatabase\\txt\\src/normalUser.txt");
22          FileReader fr = null;
23          BufferedReader br = null;
24          try {
25              fr = new FileReader(f);
26              br = new BufferedReader(fr);
27              String line;
28              while ((line = br.readLine()) != null) {
29                  String Temp[] = line.split(",");
30                  String userName = Temp[0];
31                  String password = Temp[1];
32                  if (userName.equals(name) && password.equals(pwd)) {
33                      return true;
34                  }
35              }
36          } catch (FileNotFoundException ex) {
37              Logger.getLogger(adminUser.class.getName()).log(Level.SEVERE, null, ex);
38          } catch (IOException ex) {
39              Logger.getLogger(adminUser.class.getName()).log(Level.SEVERE, null, ex);
40          } finally {
41              try {
42                  br.close();
43              } catch (IOException ex) {
44                  Logger.getLogger(adminUser.class.getName()).log(Level.SEVERE, null, ex);
45              }
46          }
47          return false;
48      }
```

Image 3.4.1 Abstraction

The Abstraction is present in the normalUser class, which is a subclass of the user class. The normalUser class provides a specific implementation of the user class for normal users, while abstracting away the details of how the user data is stored.

The abstraction is present in the following methods of the normalUser class:

normalCheck() method (lines 14-42): This method takes a username and password as parameters and checks whether a normal user with the given username and password exists in the system. It does this by reading the normalUser.txt file, which contains a list of normal users and their passwords. If a matching user is found, the method returns true; otherwise, it returns false.

BAB IV

SYSTEM DESIGN

4.1 MAIN APPLICATION

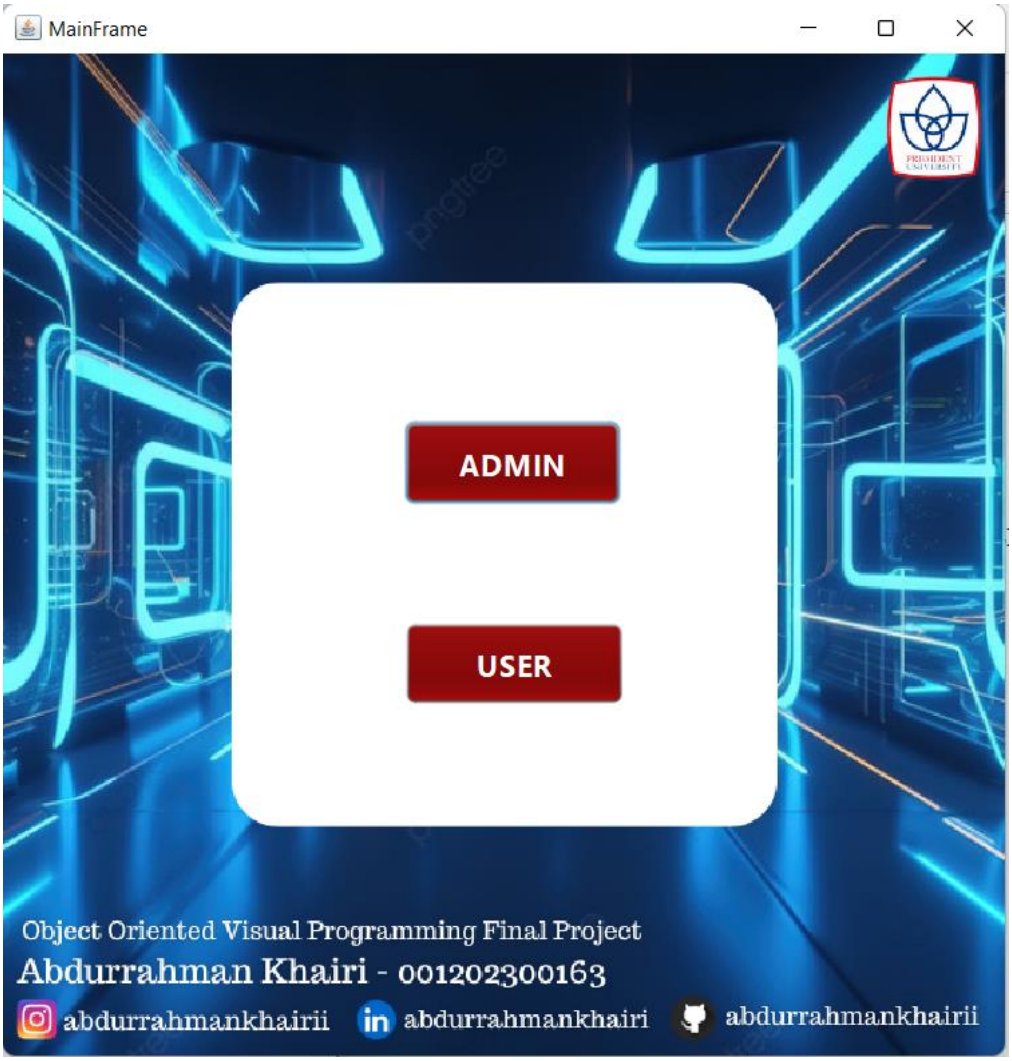

Image 4.1.1 Main Display Application

4.2 ADMIN

4.2.1 adminLogin

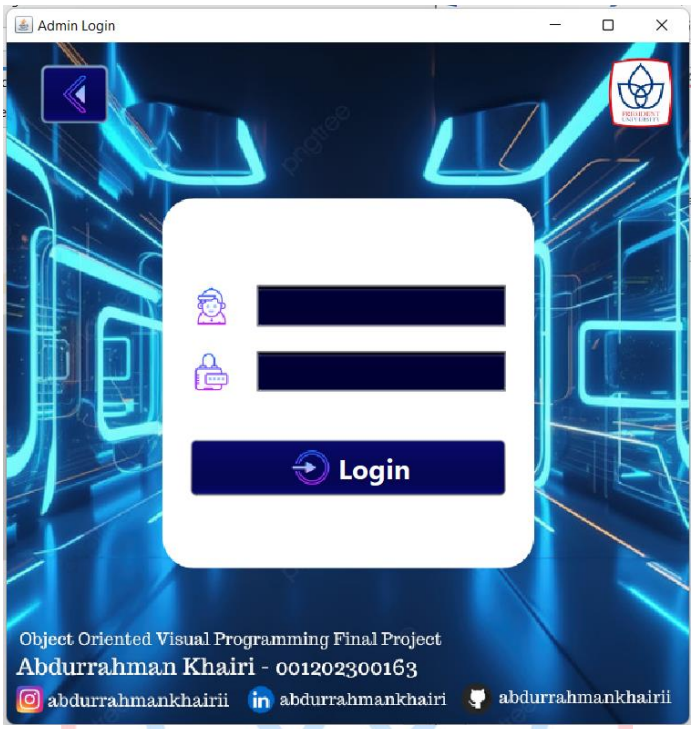

Image 4.2.1.1 Image View Admin Login Form

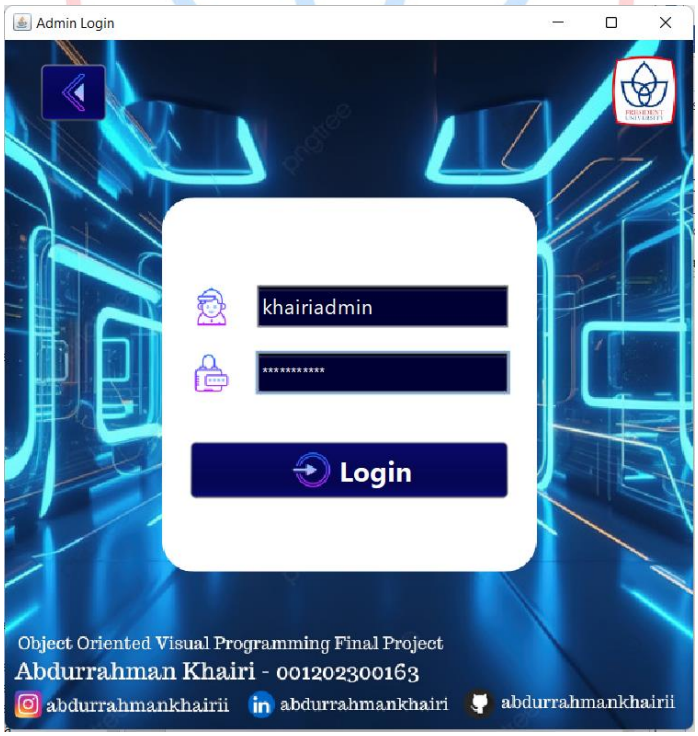

Image 4.2.1.2 Input Admin Form

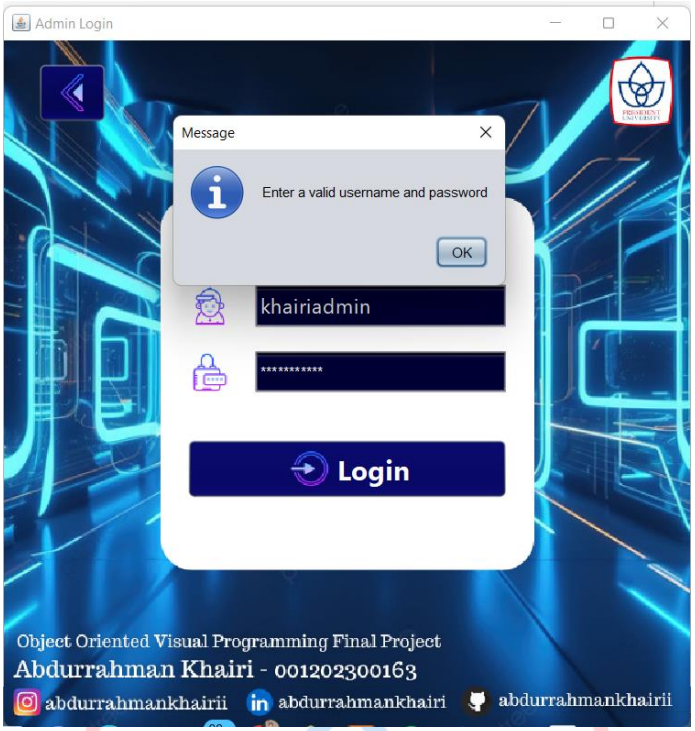

Image 4.2.1.3 Wrong Login Admin Form

4.2.2 adminForm

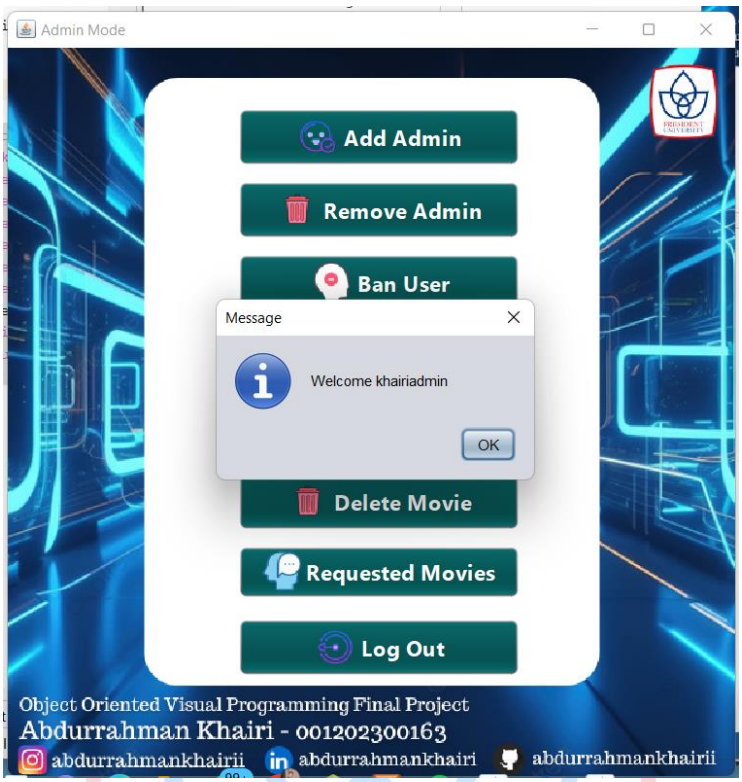

Image 4.2.2.1 Image Login Admin Form

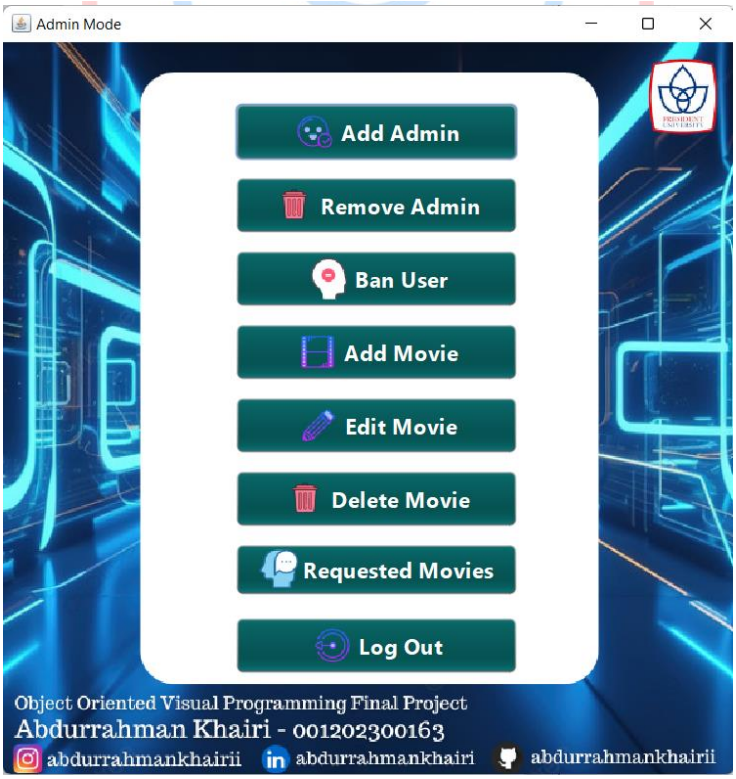

Image 4.2.2.2 View Admin Form

4.2.3 addAdmin

Image 4.2.3.1 Form Add Admin

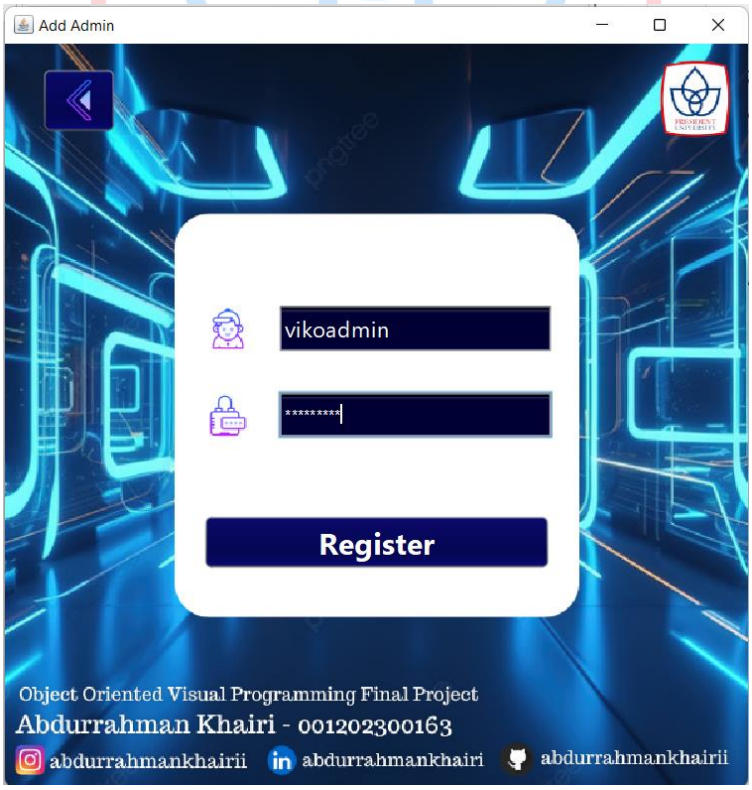

Image 4.2.3.2 Input Add Admin Form

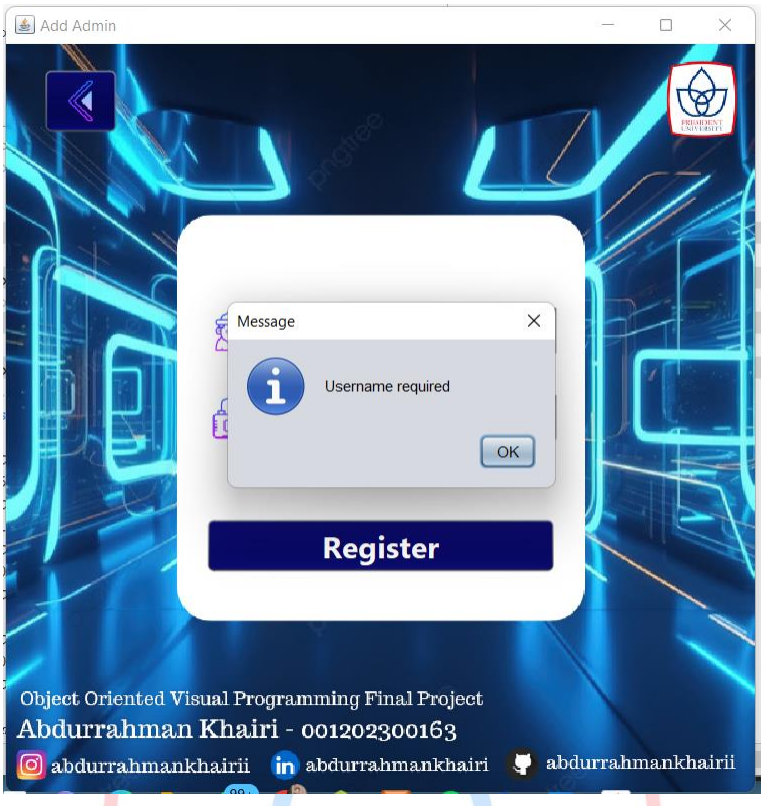

Image 4.2.3.3 Username and Password Required

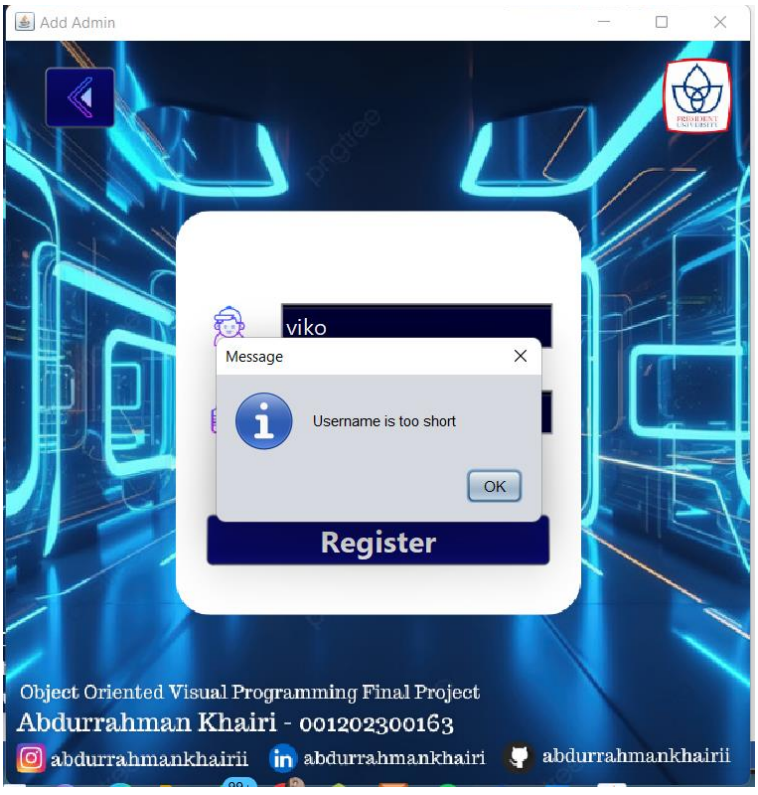

Image 4.2.3.4 Username and Password Too Short

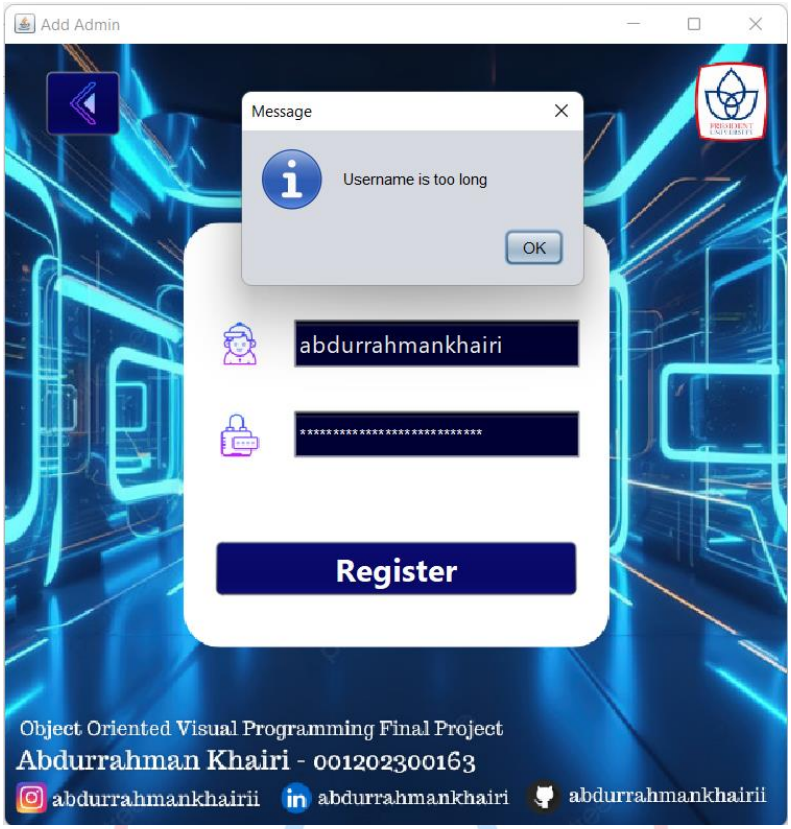

Image 4.2.3.5 Username and Password Too Long

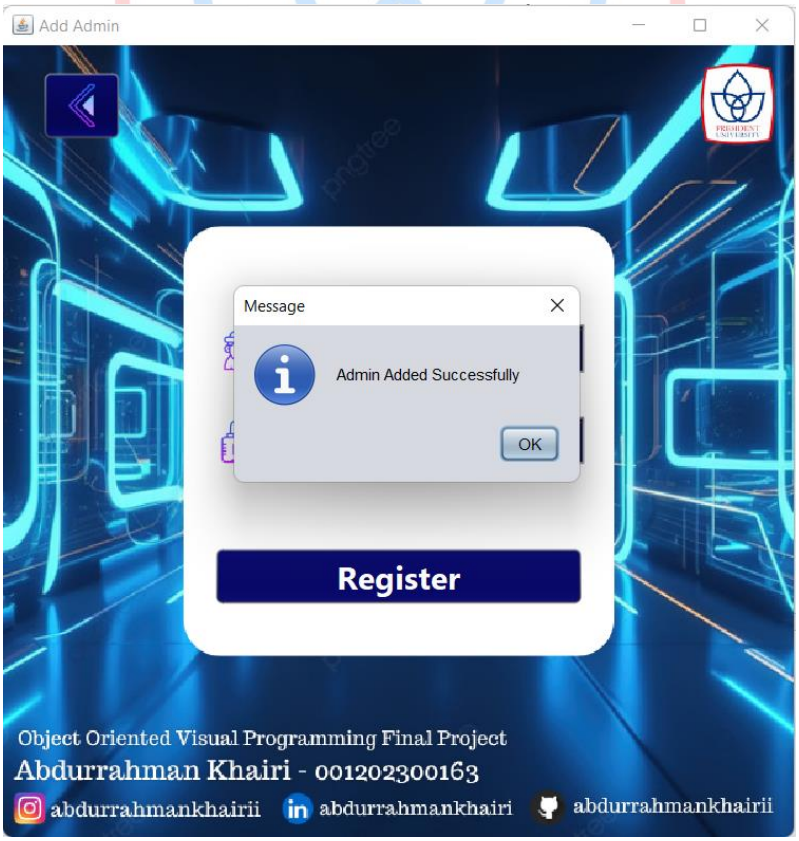

Image 4.2.3.6 Admin Added Successfully

4.2.4 RemoveAdmin

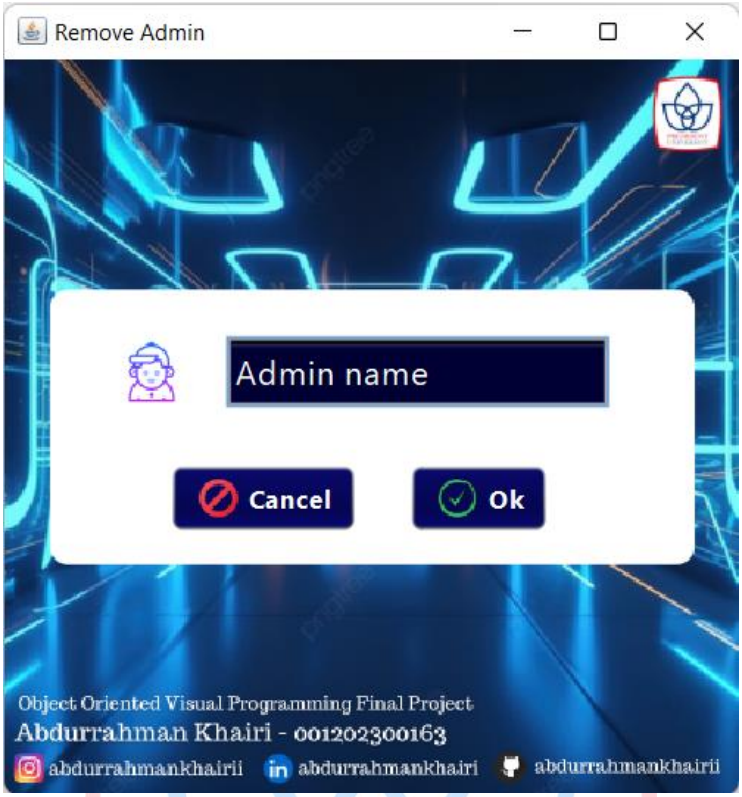

Image 4.2.4.1 View Remove Admin Form

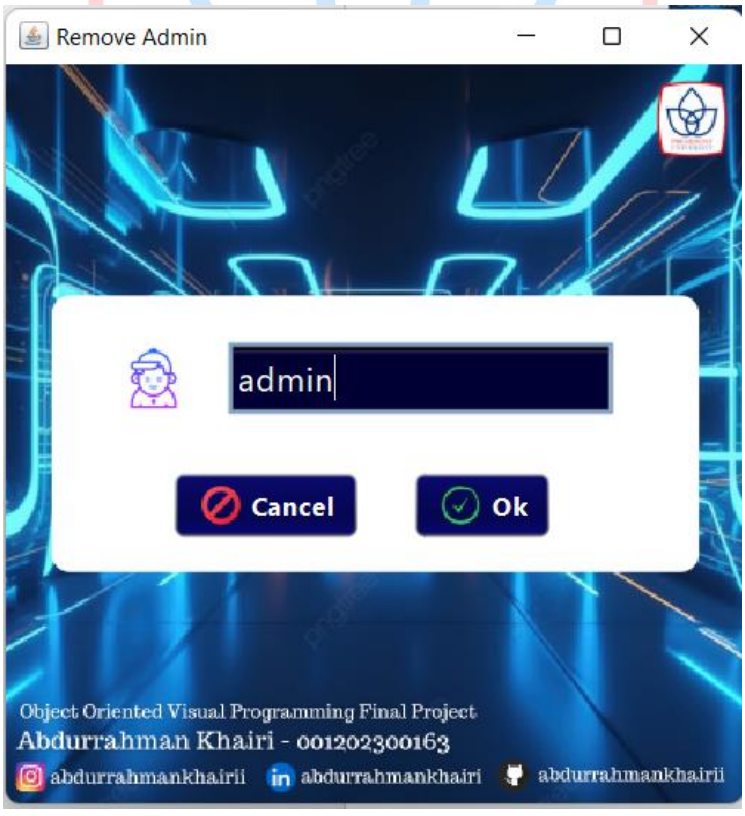

Image 4.2.4.2 Input Remove Admin Form

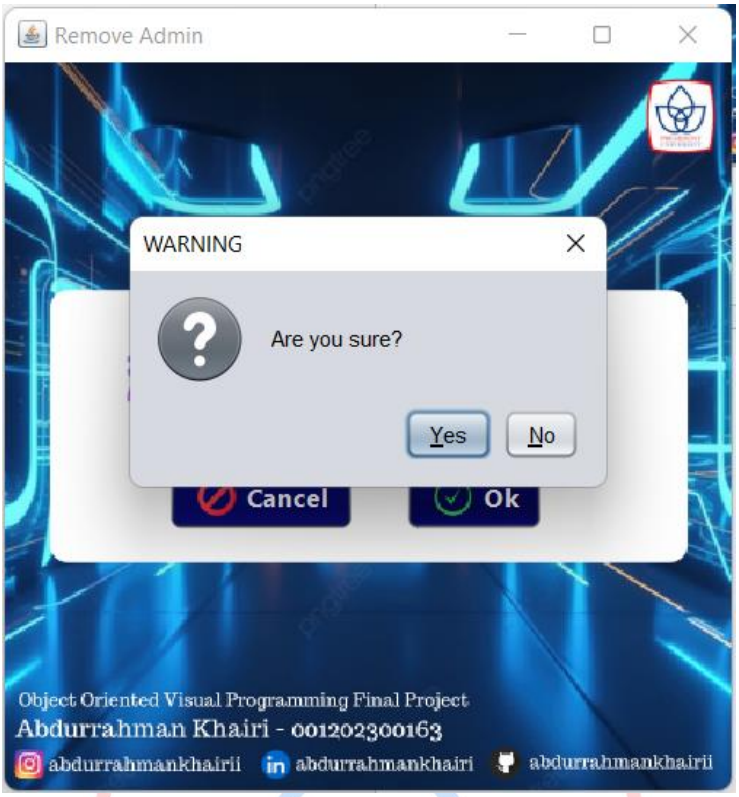

Image 4.2.4.3 Cancel Remove Admin

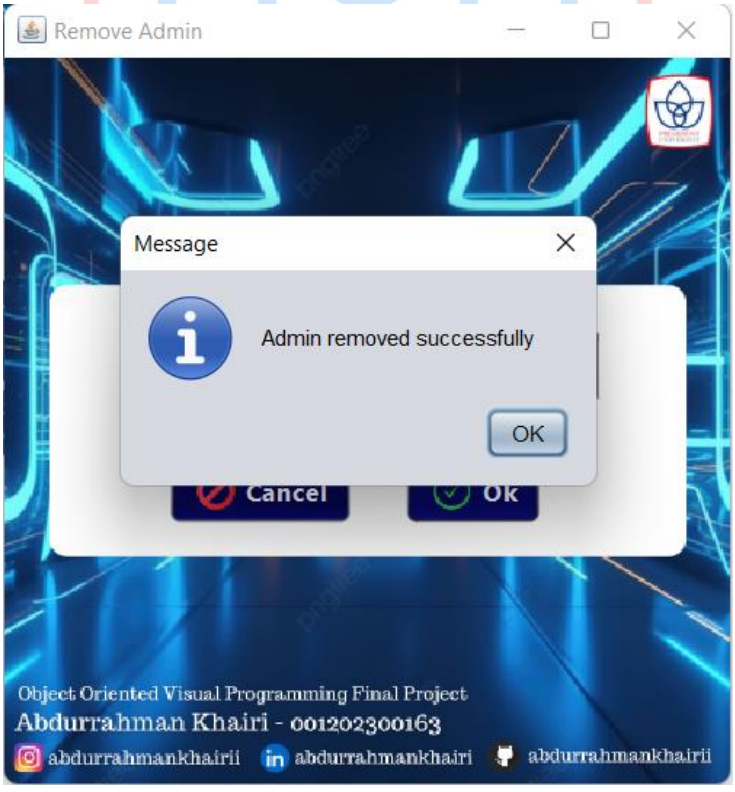

Image 4.2.4.4 Admin Removed Successfully

4.2.5 BanUser

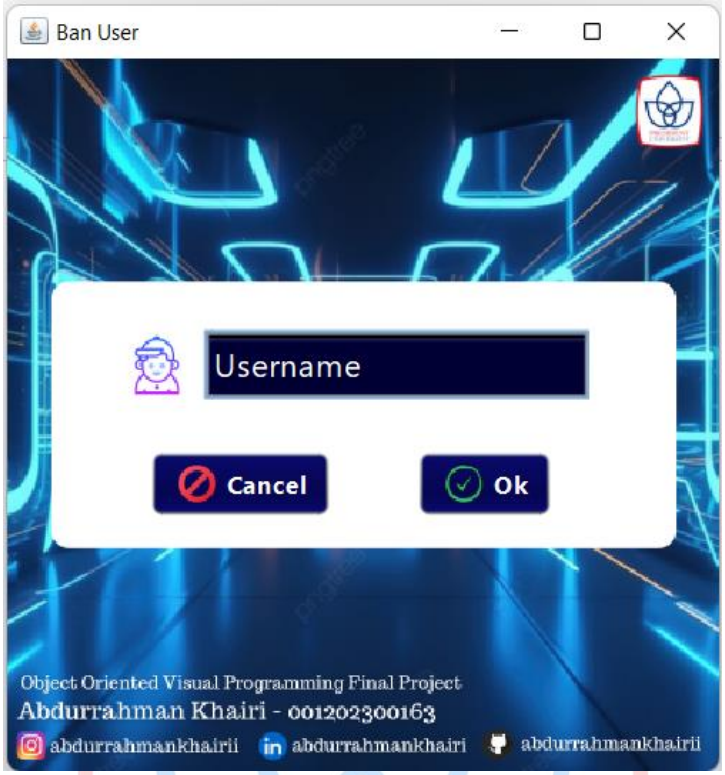

Image 4.2.5.2 View Ban User Form

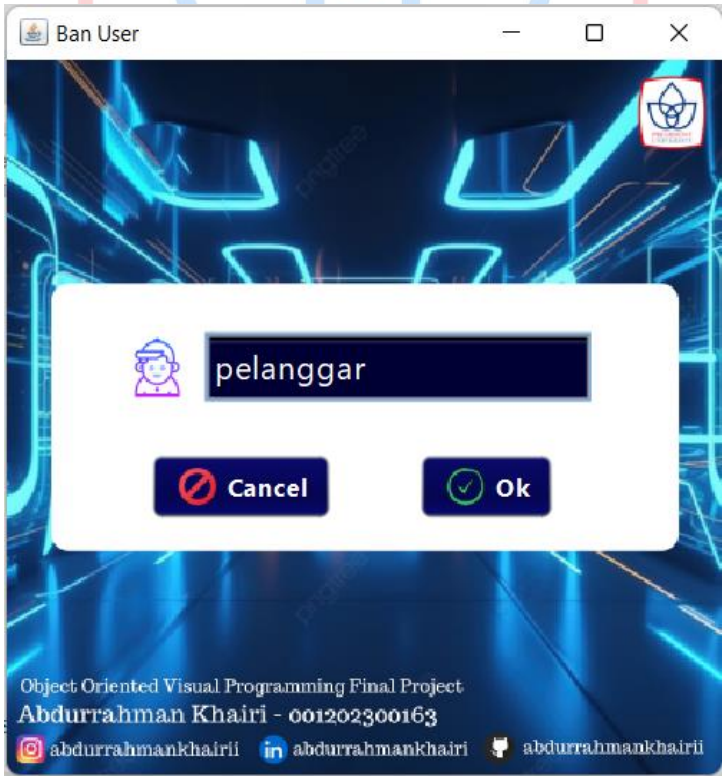

Image 4.2.5.3 Input Ban User Form

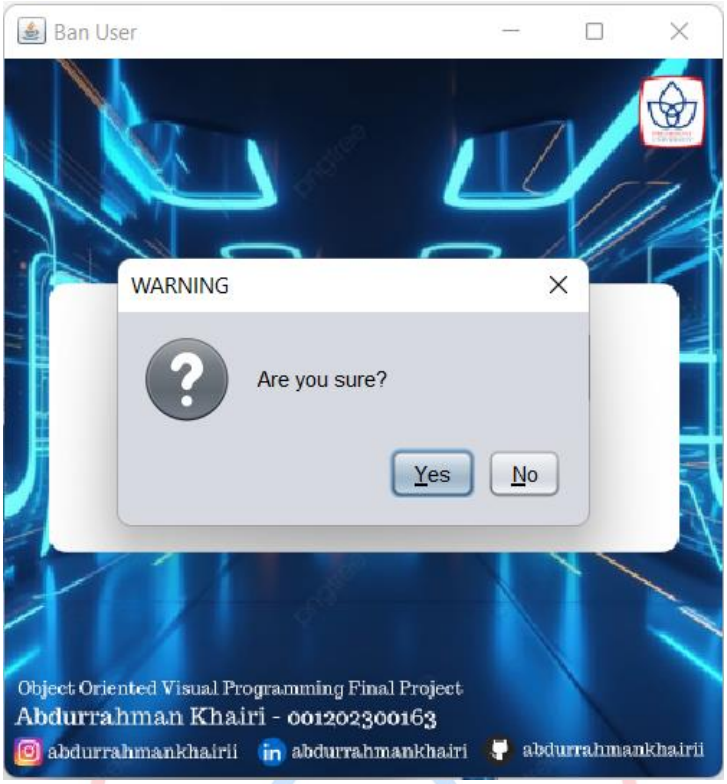

Image 4.2.5.4 Cancel Ban User Form

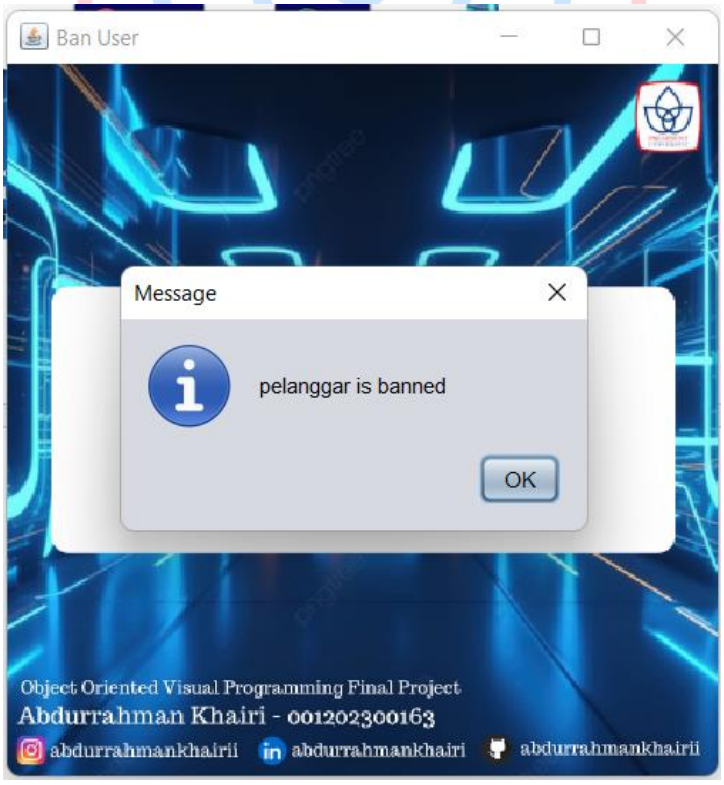

Image 4.2.5.5 Banned User

4.2.6 addMovie

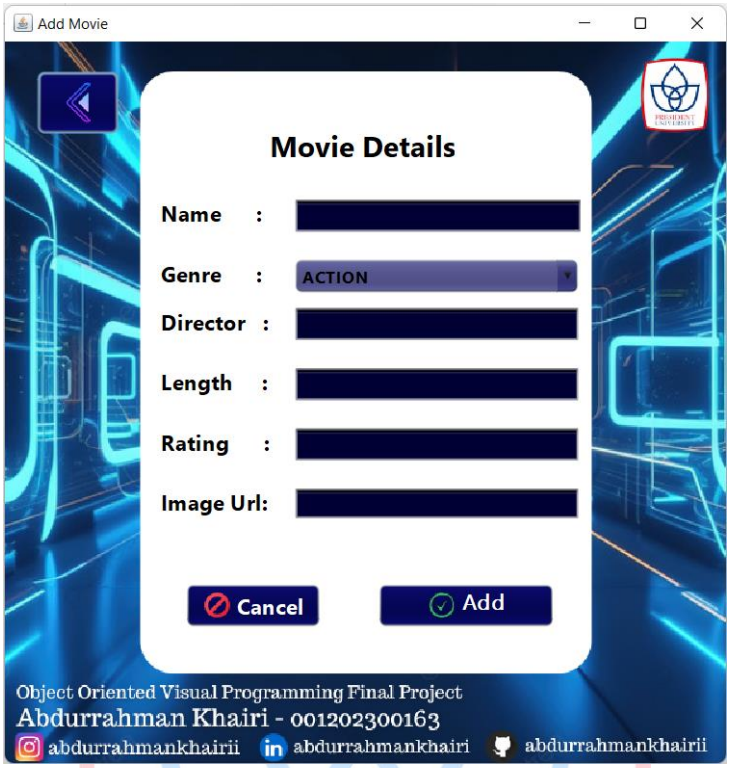

Image 4.2.6.1 View Add Movie Form

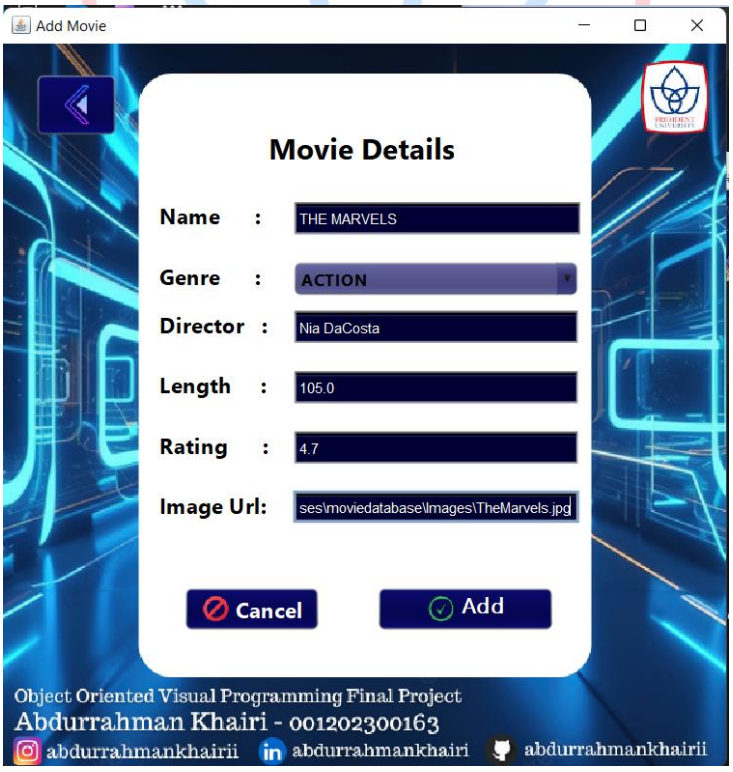

Image 4.2.6.2 Input Add Movie Form

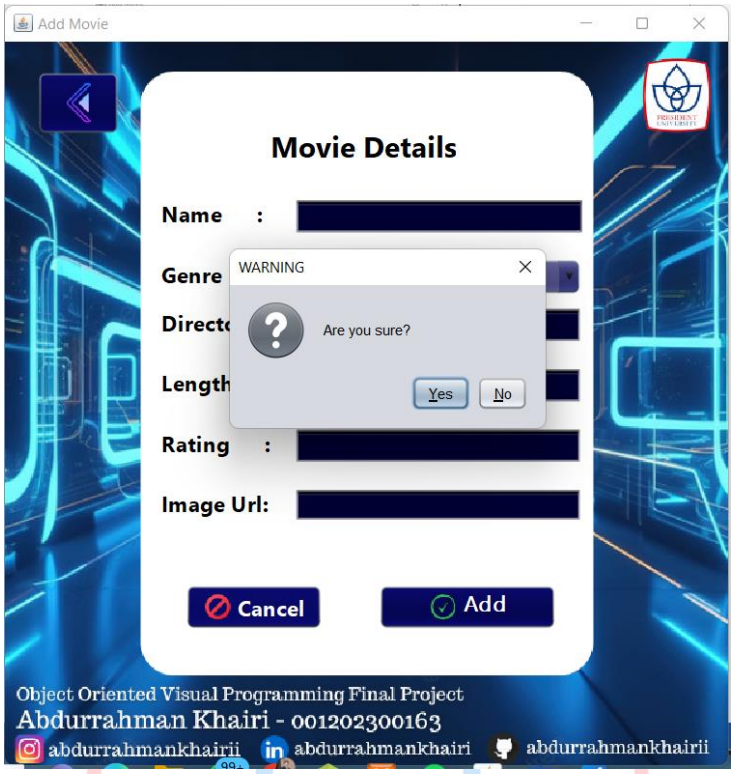

Image 4.2.6.3 Cancel Add View Form

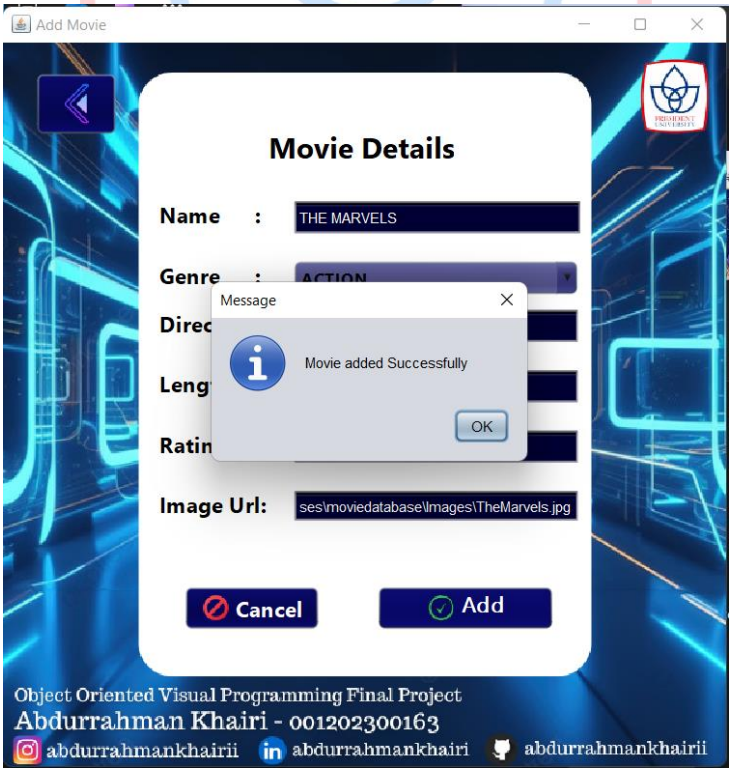

Image 4.2.6.4 Movie Added Successfully

4.2.7 editMovie

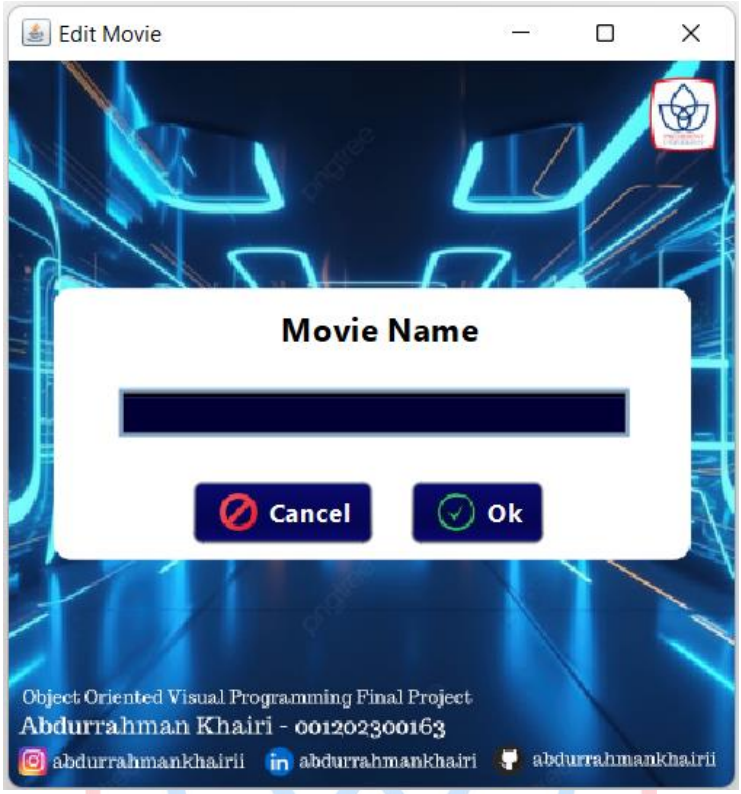

Image 4.2.7.1 View Edit Movie Form

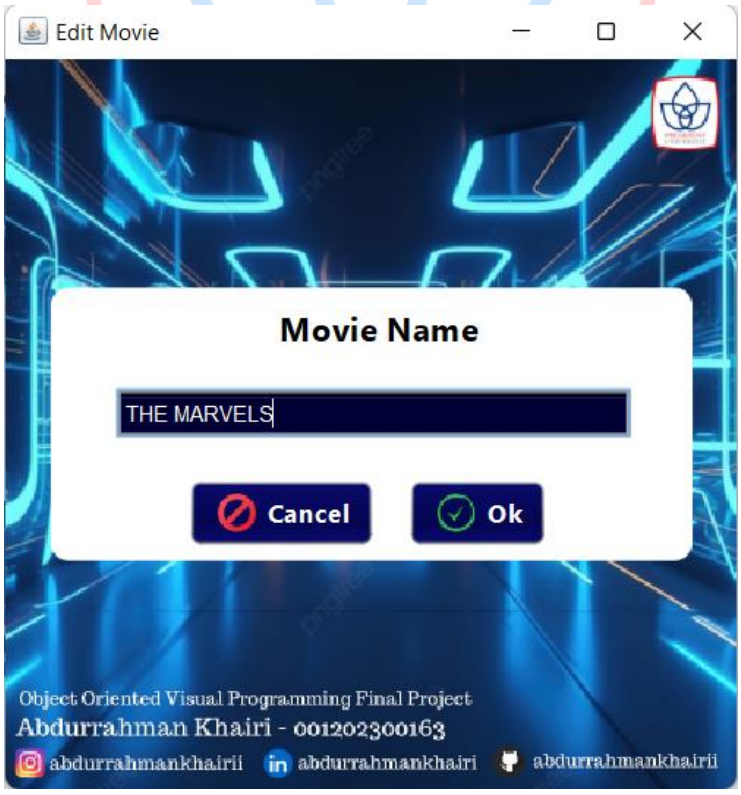

Image 4.2.7.2 Input Edit Movie Form

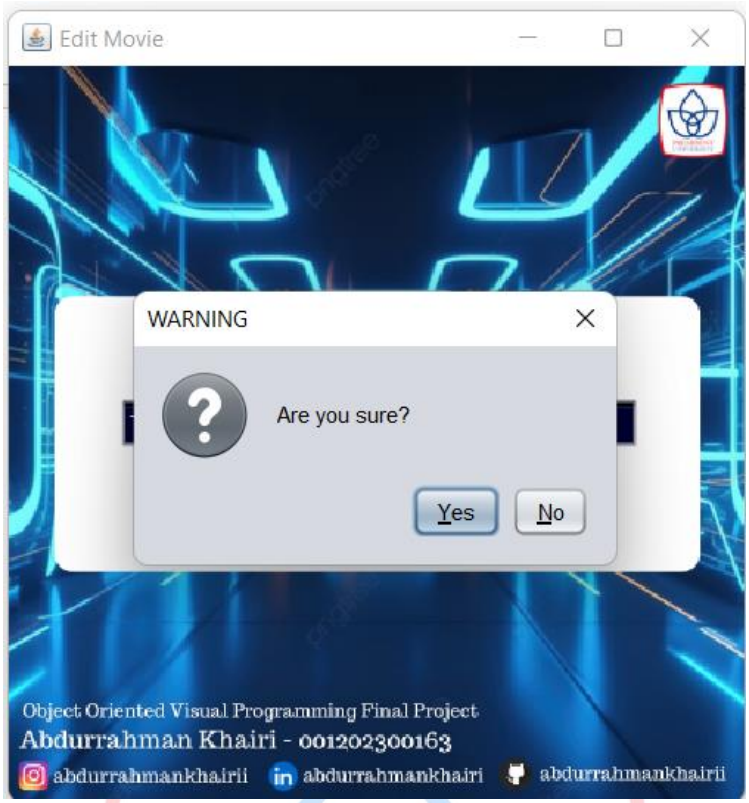

Image 4.2.7.3 Cancel Edit Movie Form

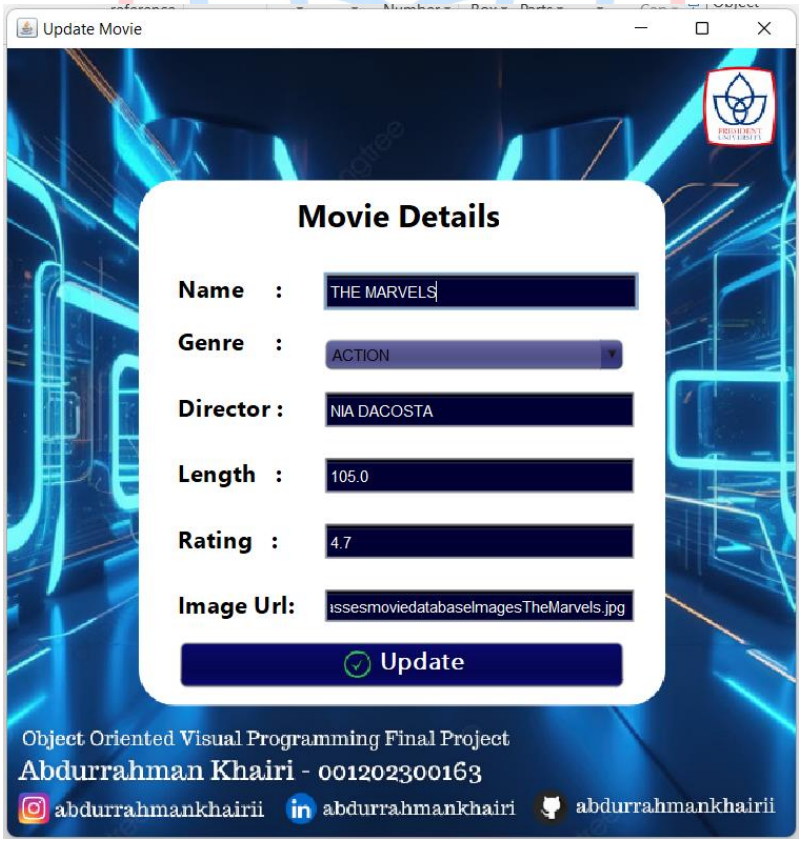

Image 4.2.7.4 View Update Movie Form

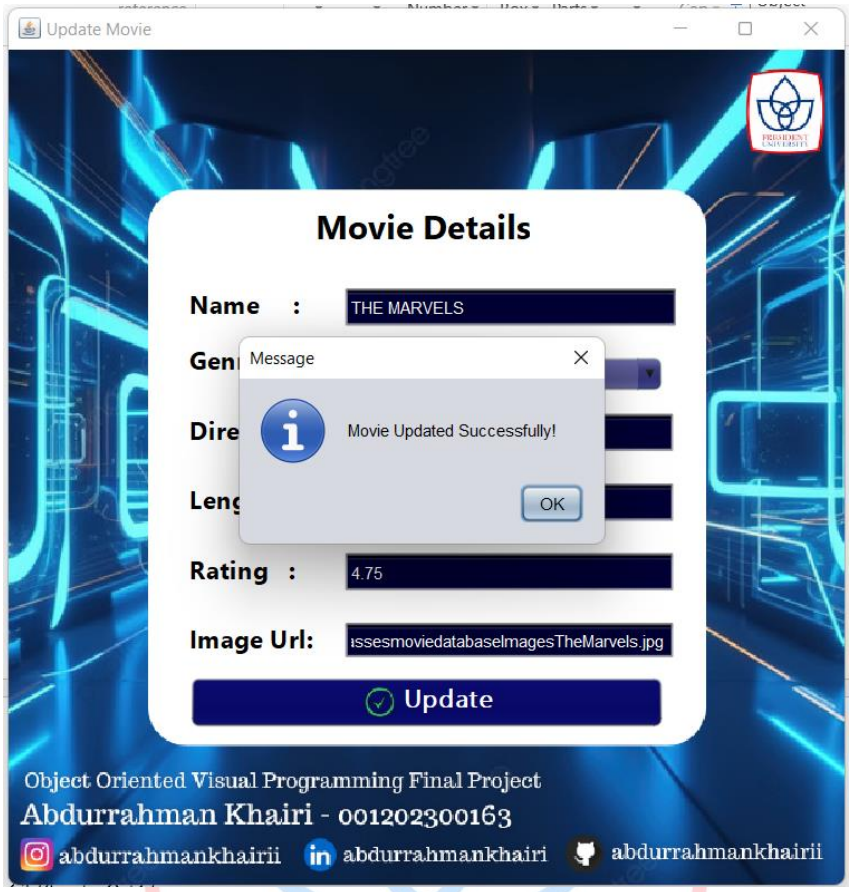

Image 4.2.7.5 Movie Updated Successfully

4.2.8 deleteMovie

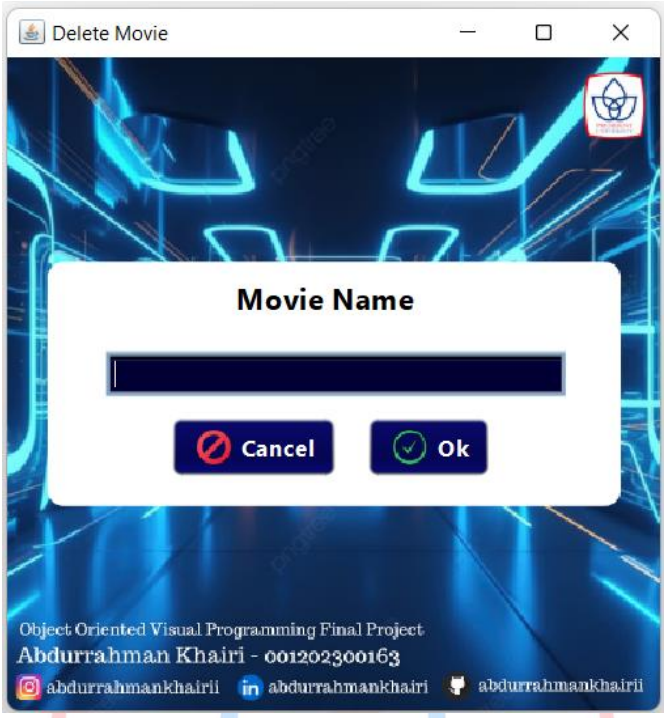

Image 4.2.8.1 View Delete Movie Form

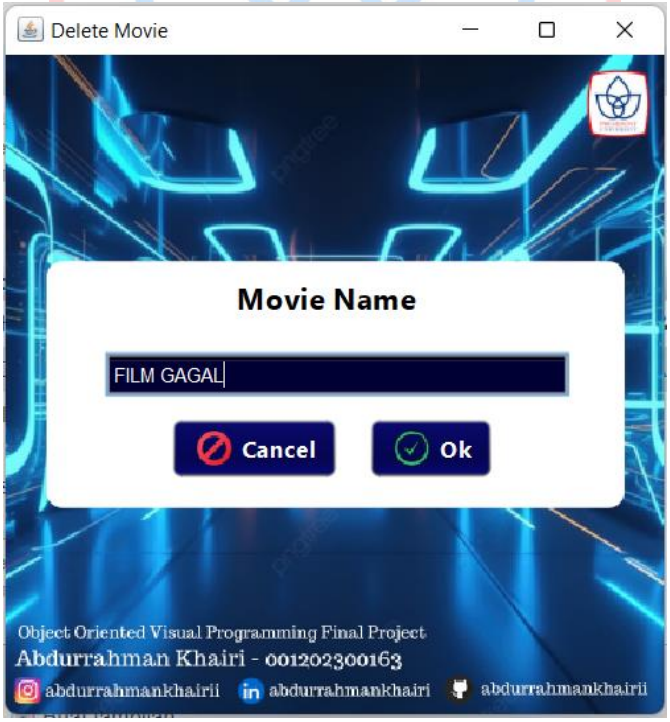

Image 4.2.8.2 Input Delete Movie Form

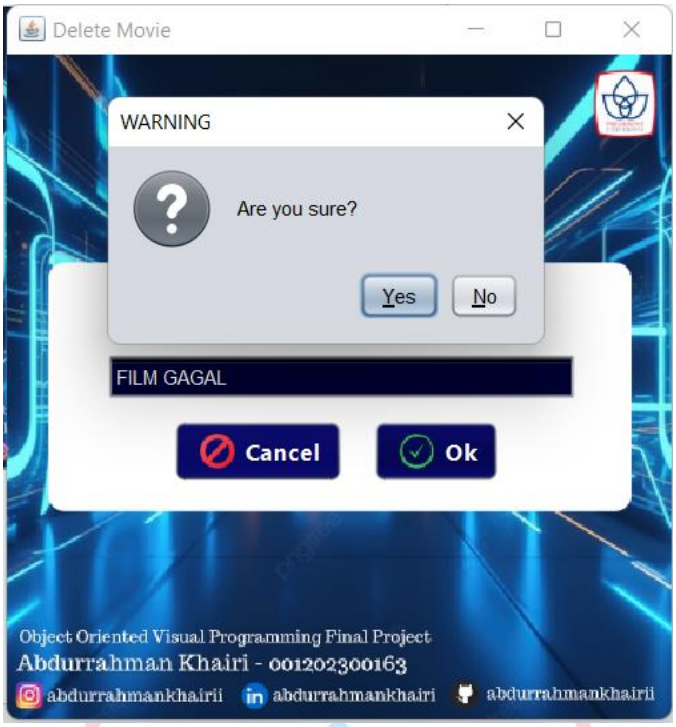

Image 4.2.8.3 Cancel Delete Movie Form

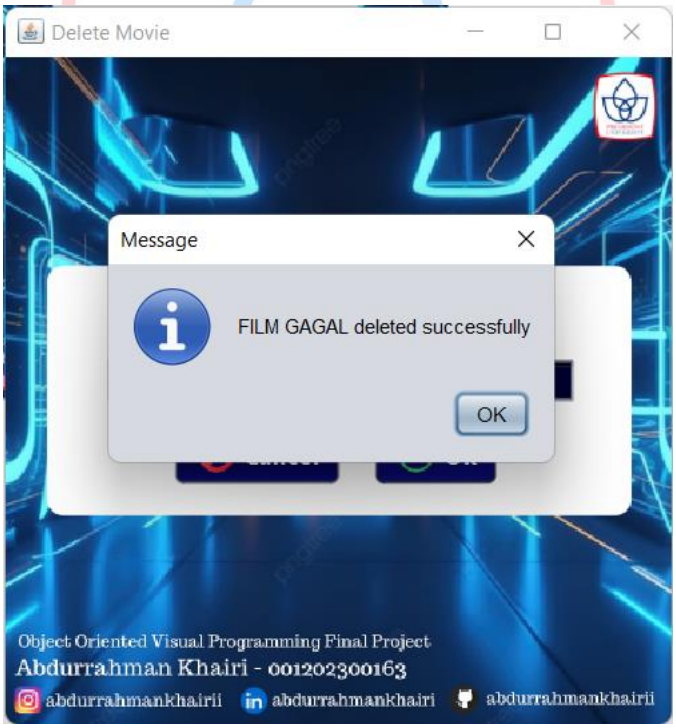

Image 4.2.8.4 Movie Deleted Successfully

4.2.9 requestedMovie

Image 4.2.9.1 View First Requested Movie

Image 4.2.9.2 View Last Request Movie

4.2.10 logOutAdmin

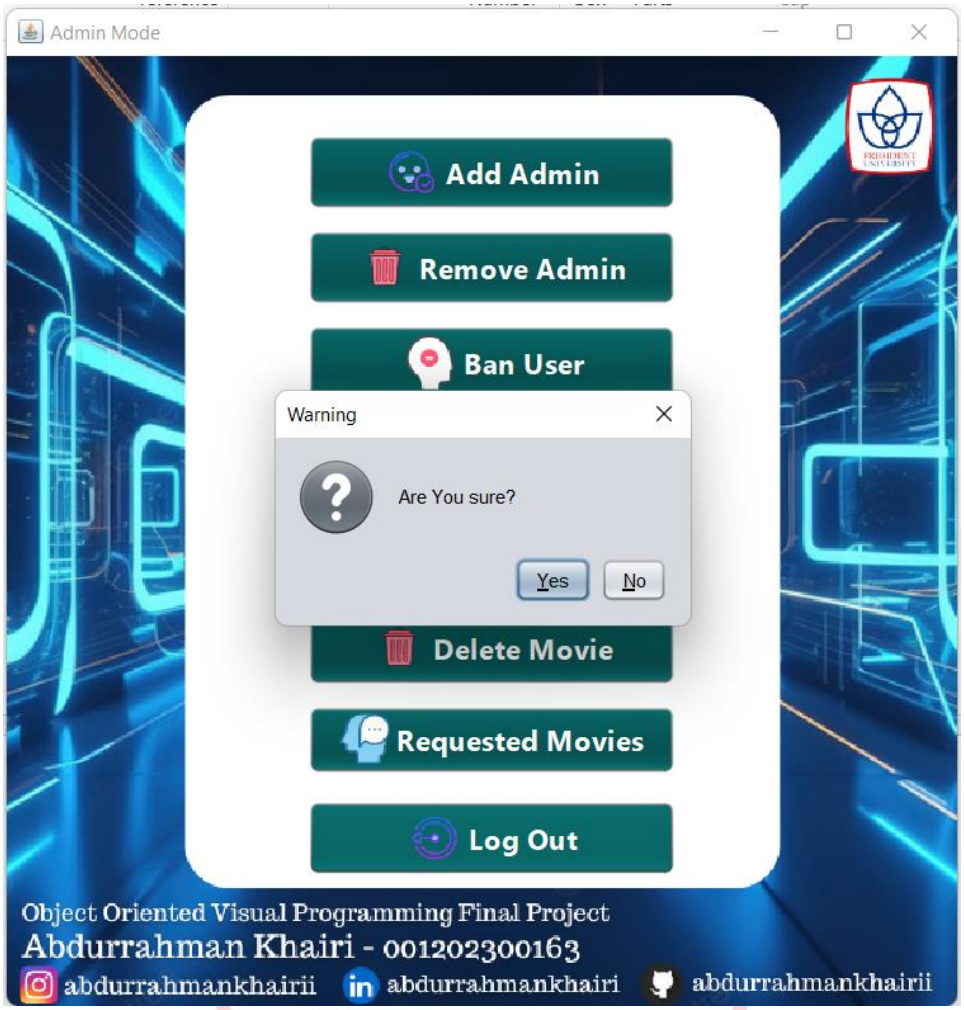

Image 4.2.10.1 LogOut Admin

4.3 USER

4.3.1 UserLogin

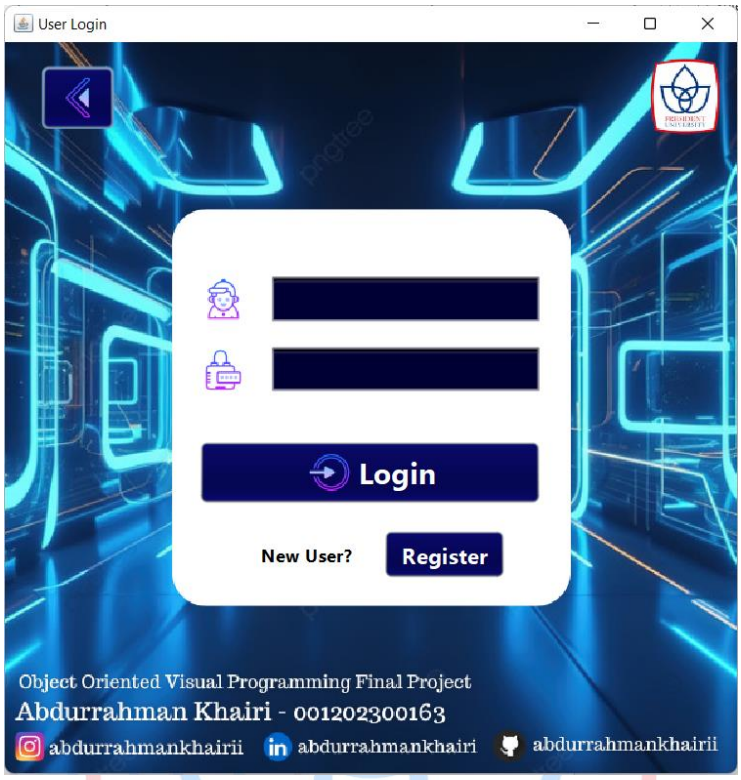

Image 4.3.1.1 View LogIn User Form

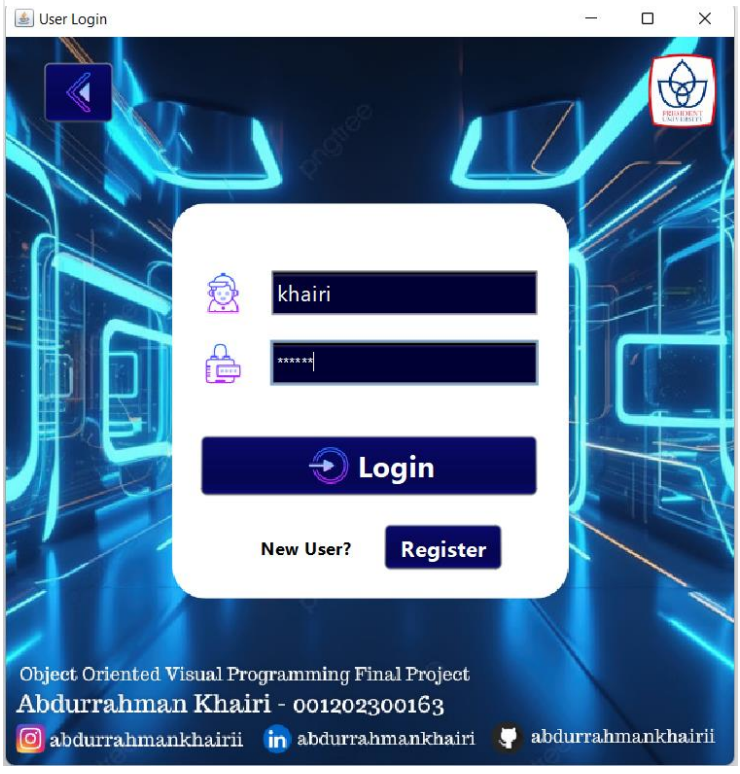

Image 4.3.1.2 Input Login User Form

4.3.2 RegisterUser

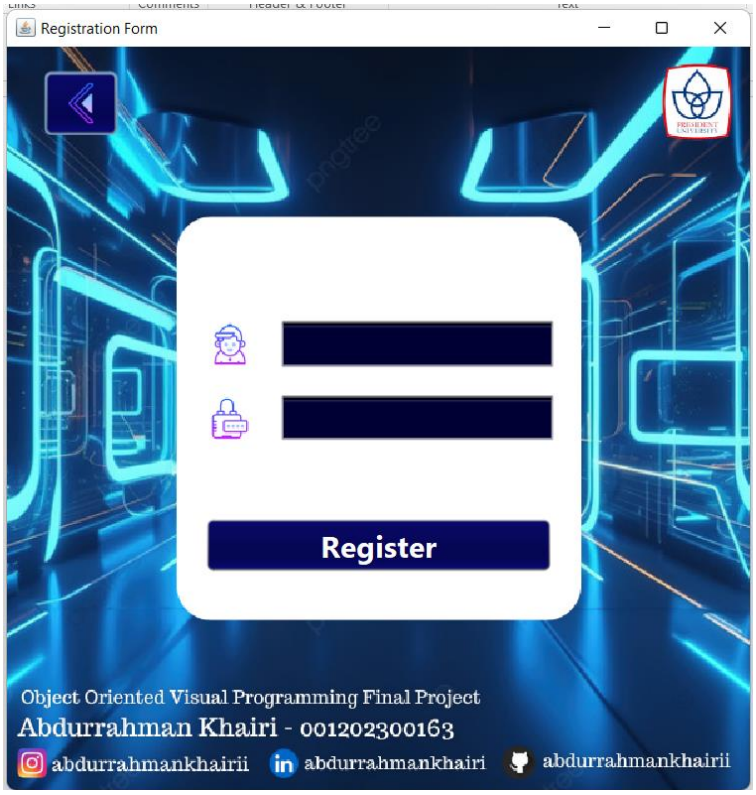

Image 4.3.2.1 View Regiestration User Form

Image 4.3.2.2 Input Registration User Form

Image 4.3.2.3 Registration User Successfully

4.3.3 UserForm

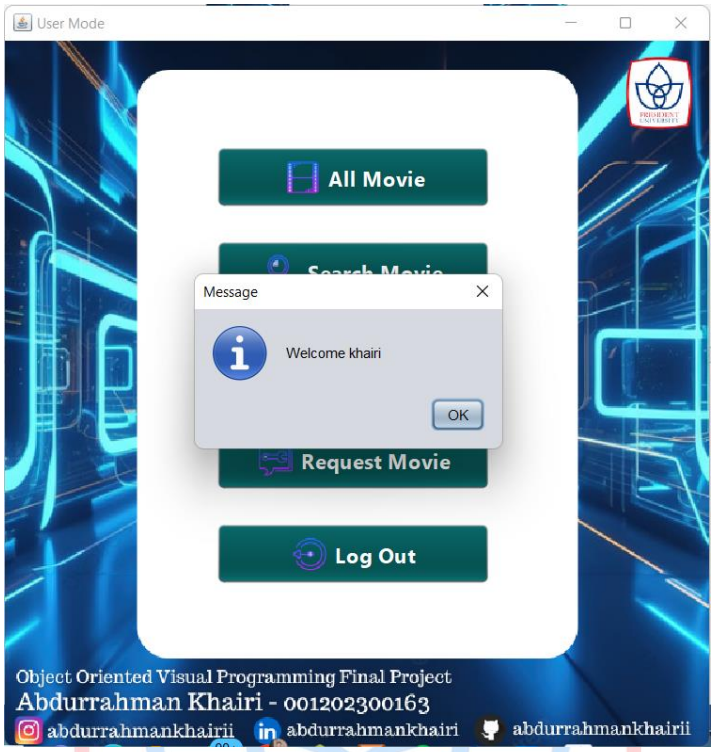

Image 4.3.3.1 LogIn User Form

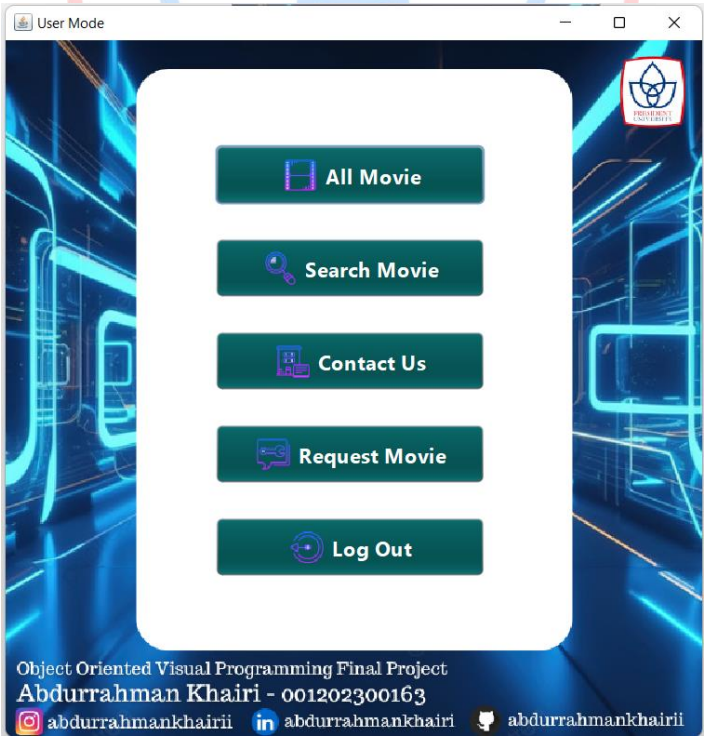

Image 4.3.3.2 View User Form

4.3.4 AllMovie

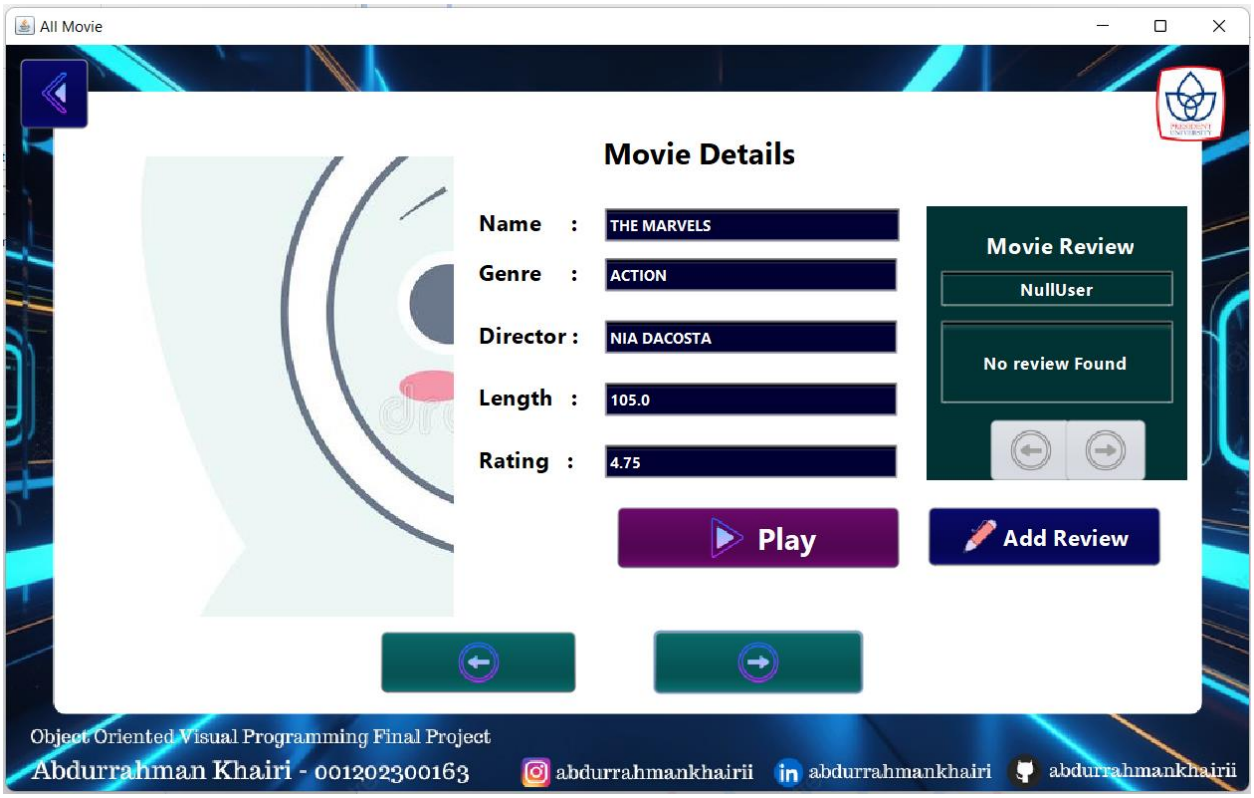

Image 4.3.4.1 View Display All Movie Details

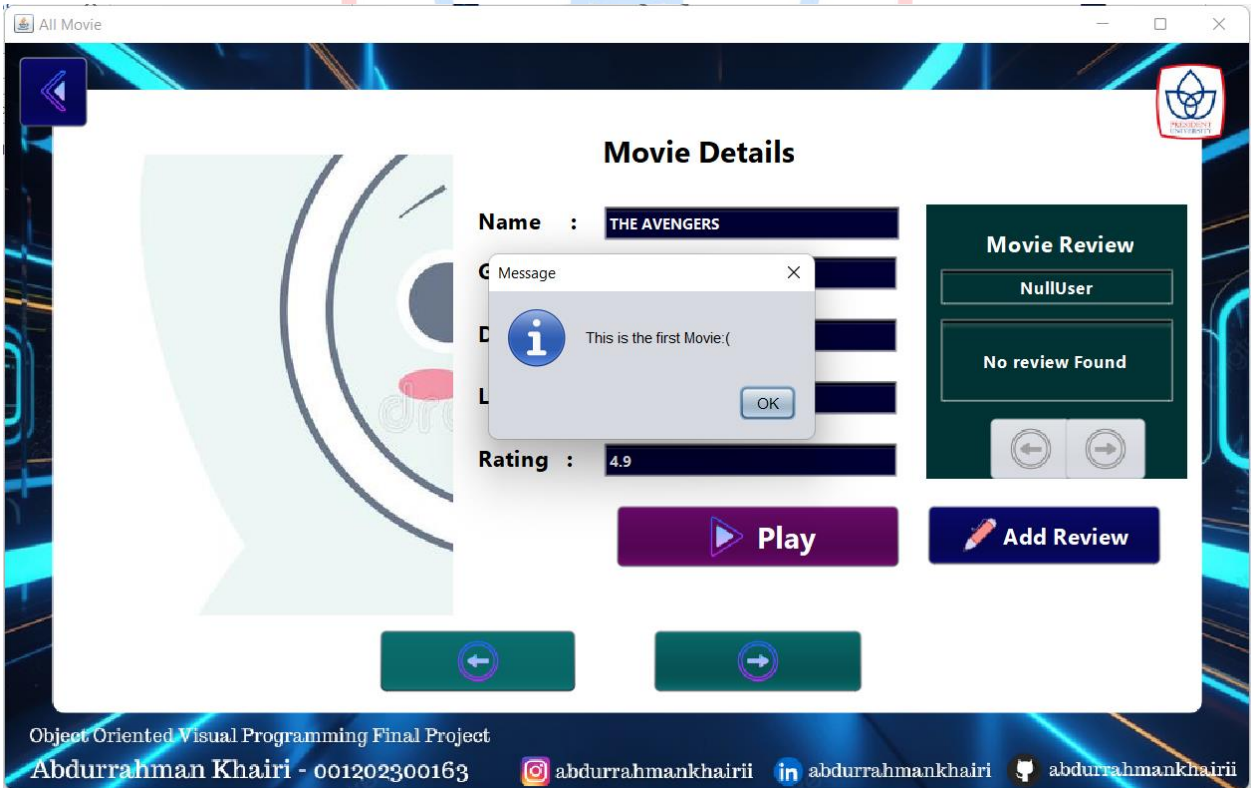

Image 4.3.4.2 View First Movie Details

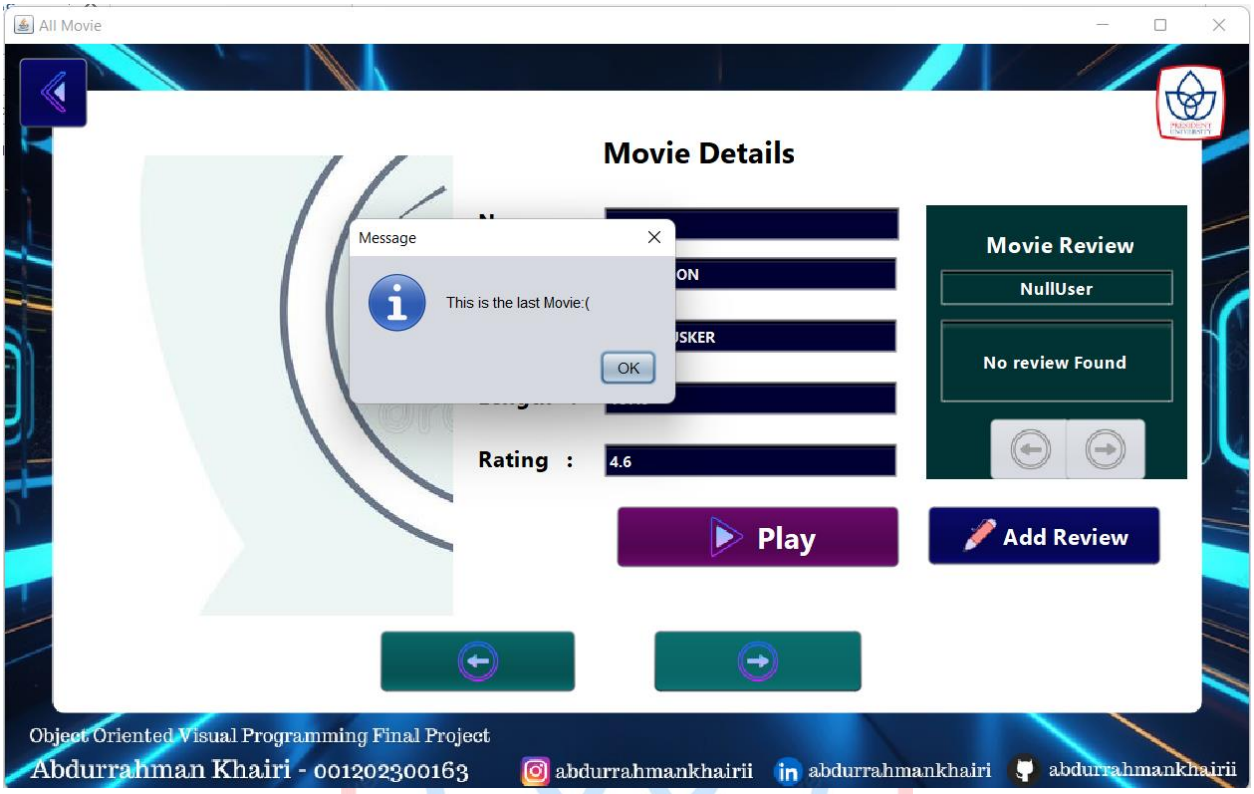

Image 4.3.4.3 View Last Movie Details

4.3.5 AddReview

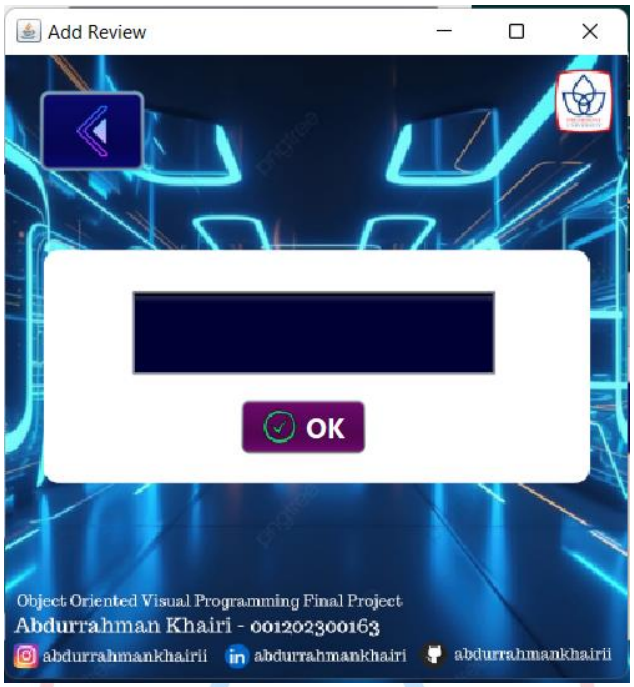

Image 4.3.5.1 View Add Review Form

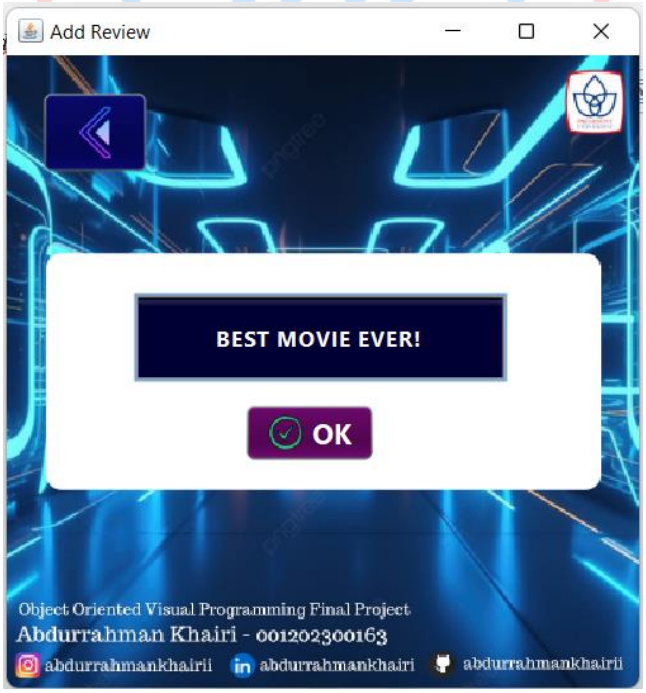

Image 4.3.5.2 Input Add Review Form

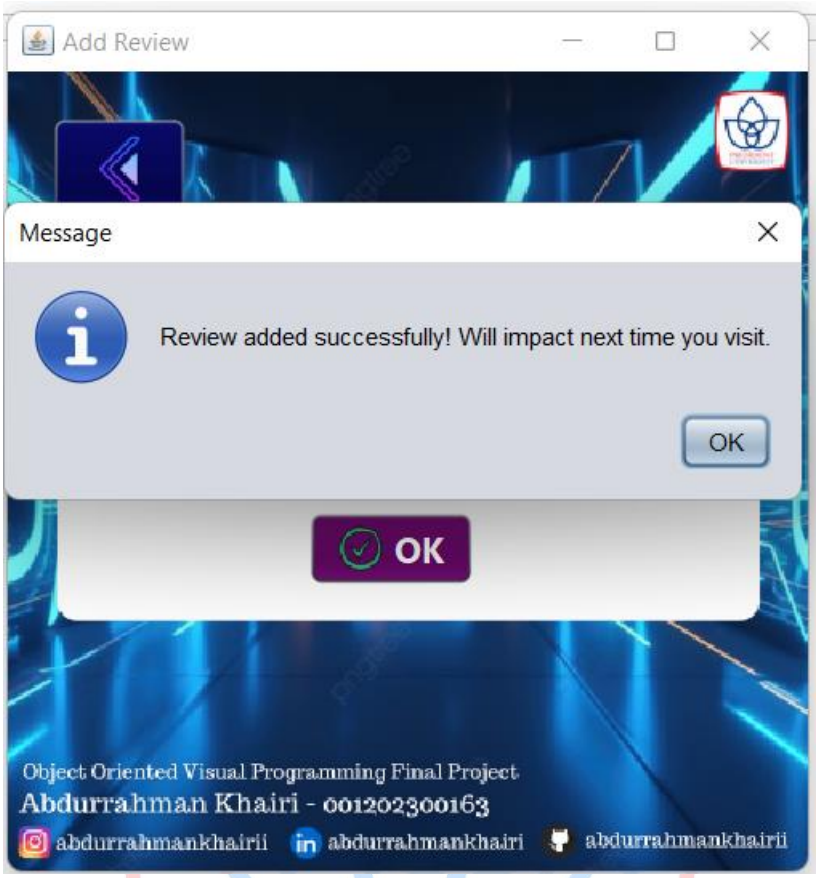

Image 4.3.5.3 Review Added Successfully

4.3.6 SearchMovie

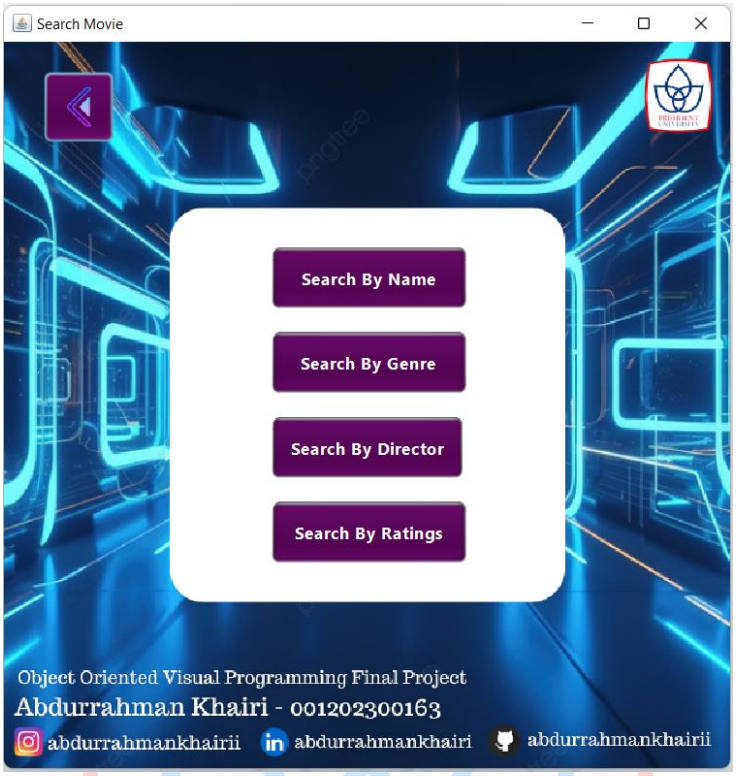

Image 4.3.6.1 View Search Movie Form

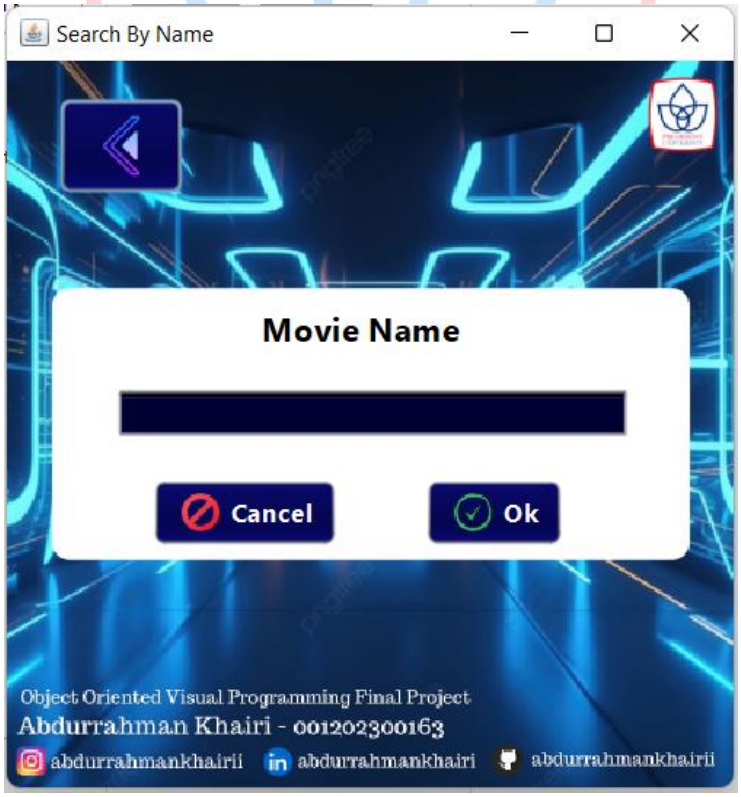

Image 4.3.6.2 View Serch Movie By Name Form

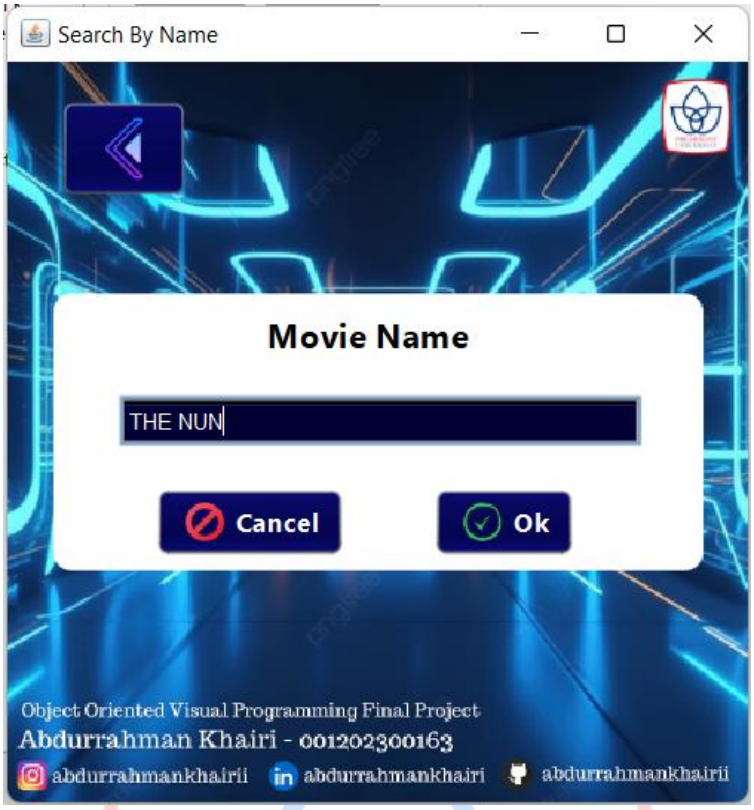

Image 4.3.6.3 Input Serach Movie By Name Form

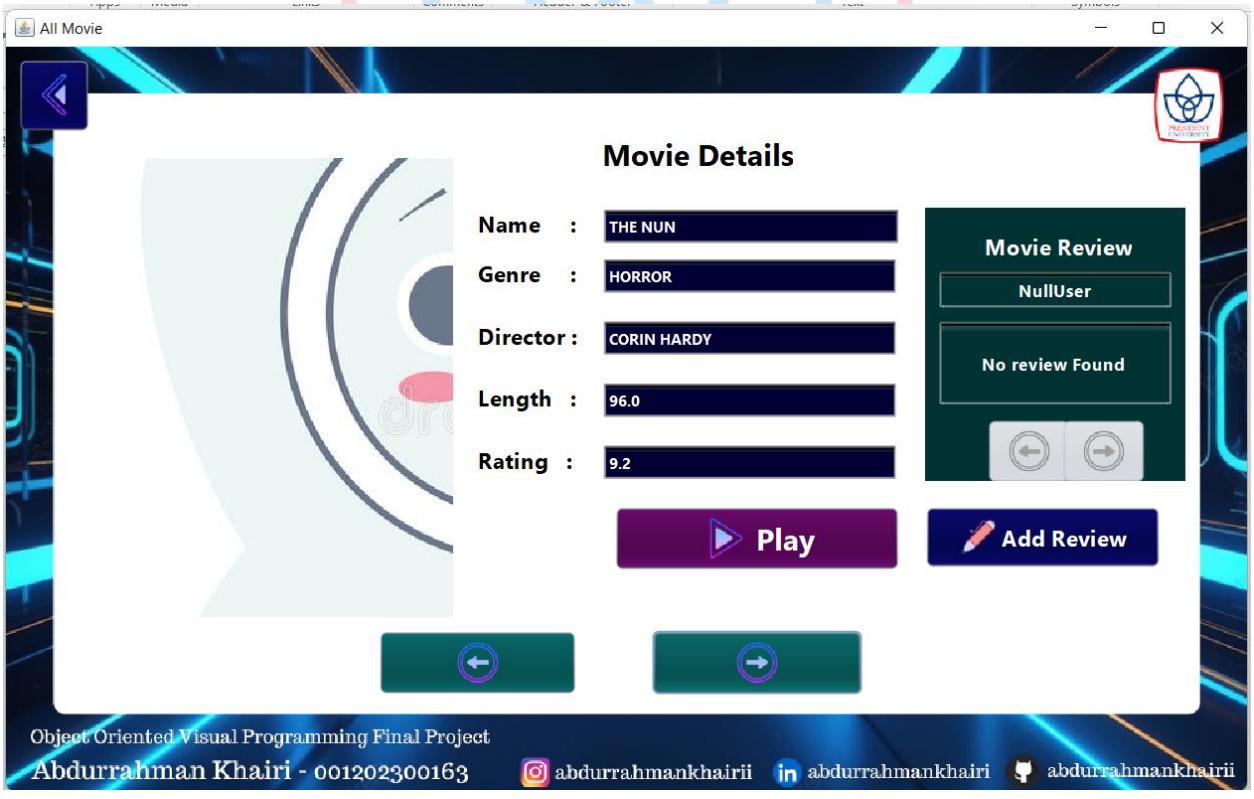

Image 4.3.6.4 Display Movie Details By Searching Name Form

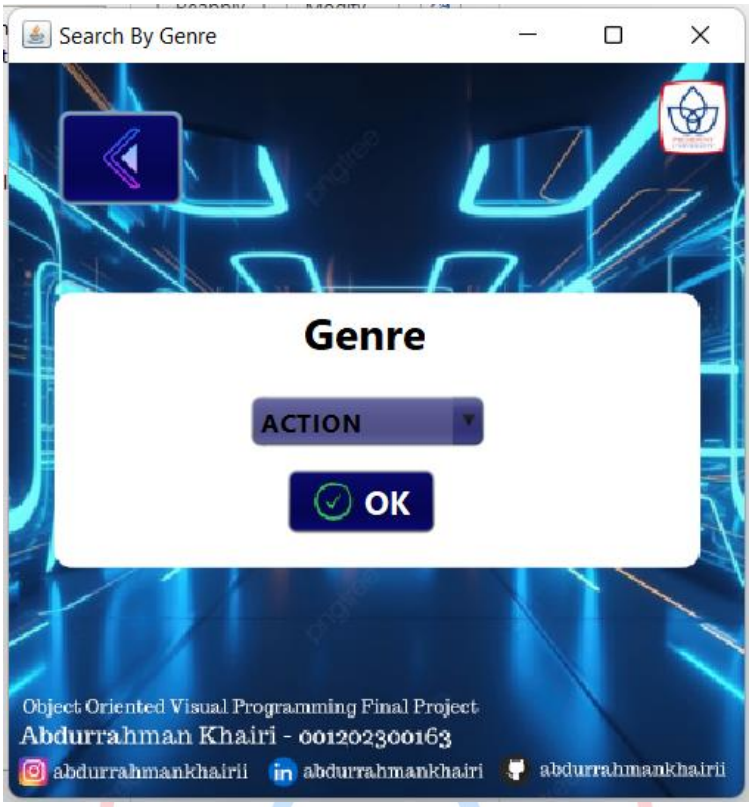

Image 4.3.6.5 View Search Movie By Genre Form

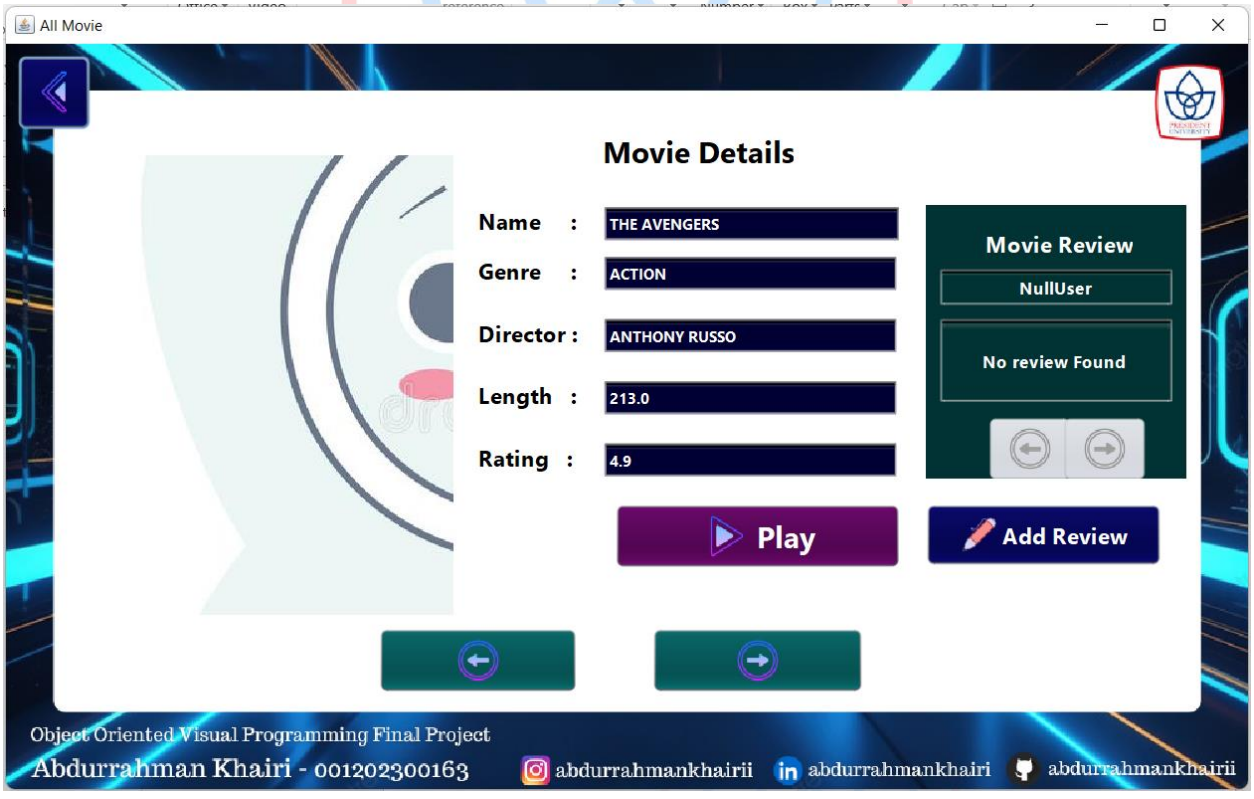

Image 4.3.6.6 Display Movie Details By Seaching Genre Form

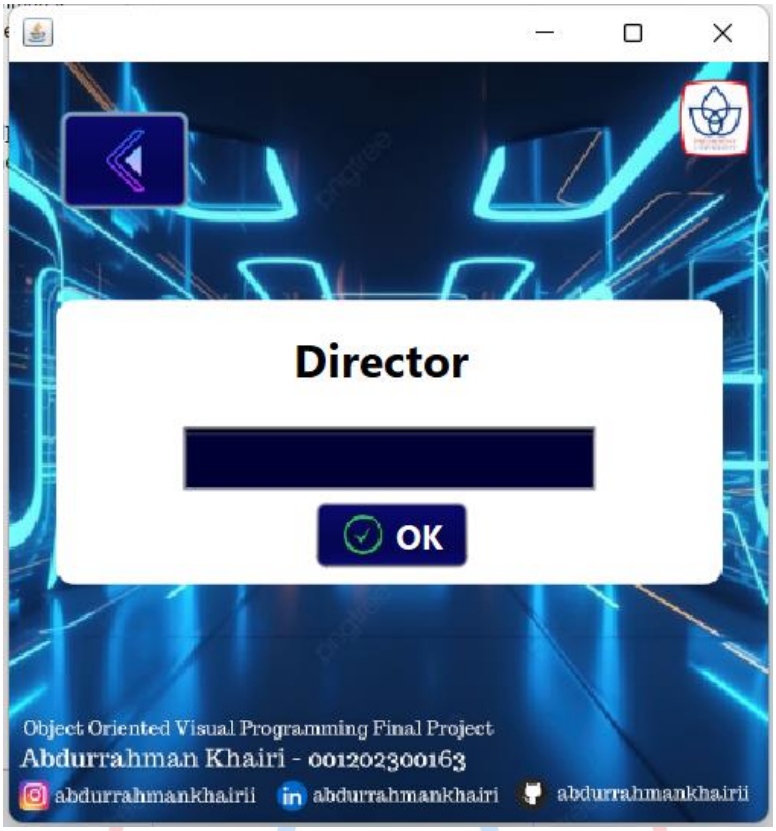

Image 4.3.6.7 View Search Movie By Director Form

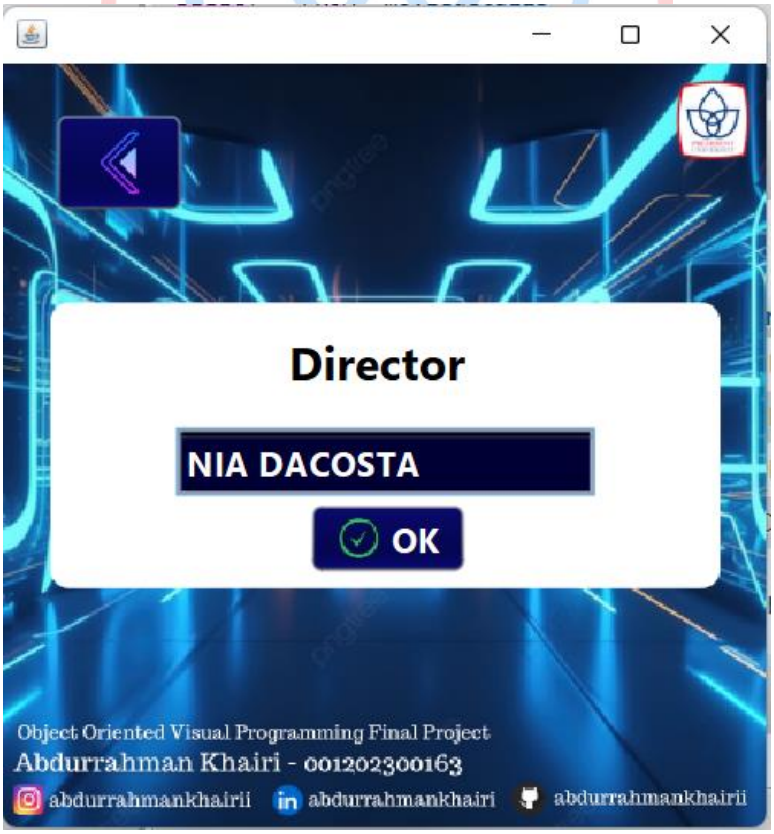

Image 4.3.6.8 Input Search Movie By Director Form

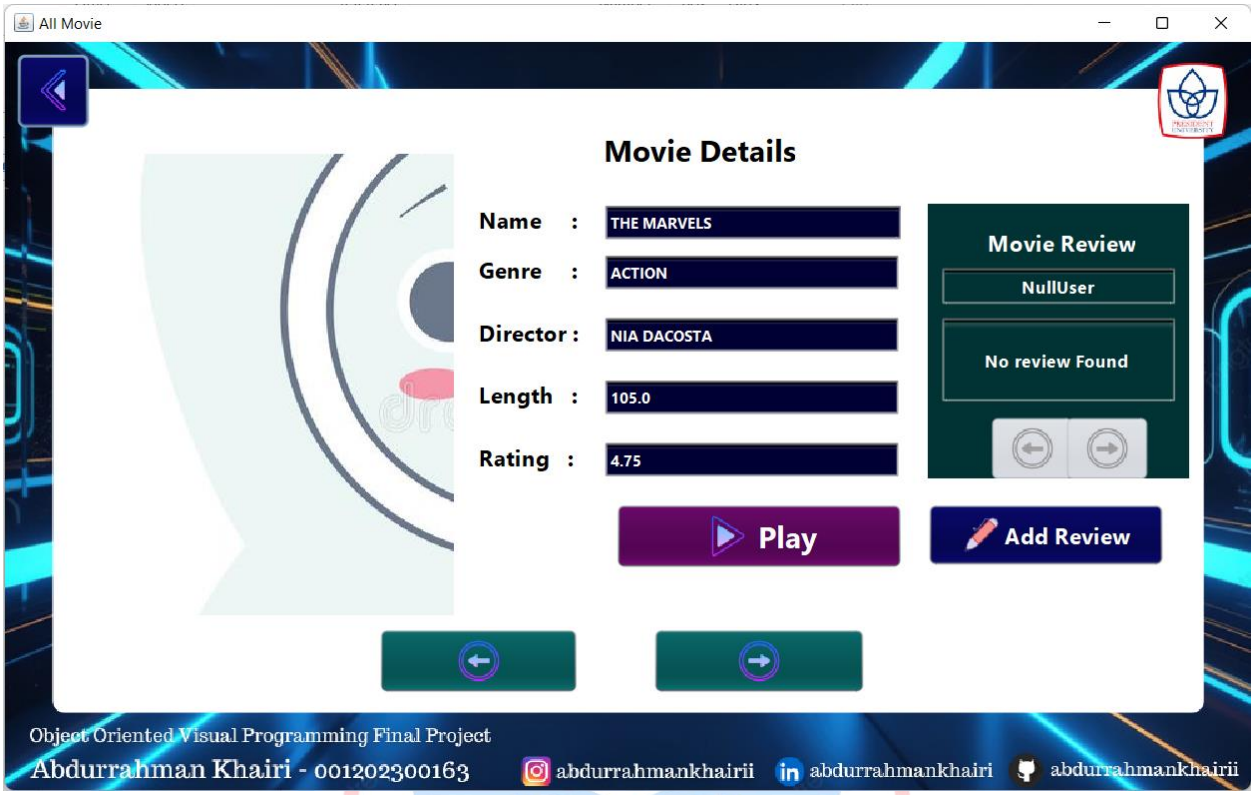

Image 4.3.6.9 Display Movie Details By Searching Director Form

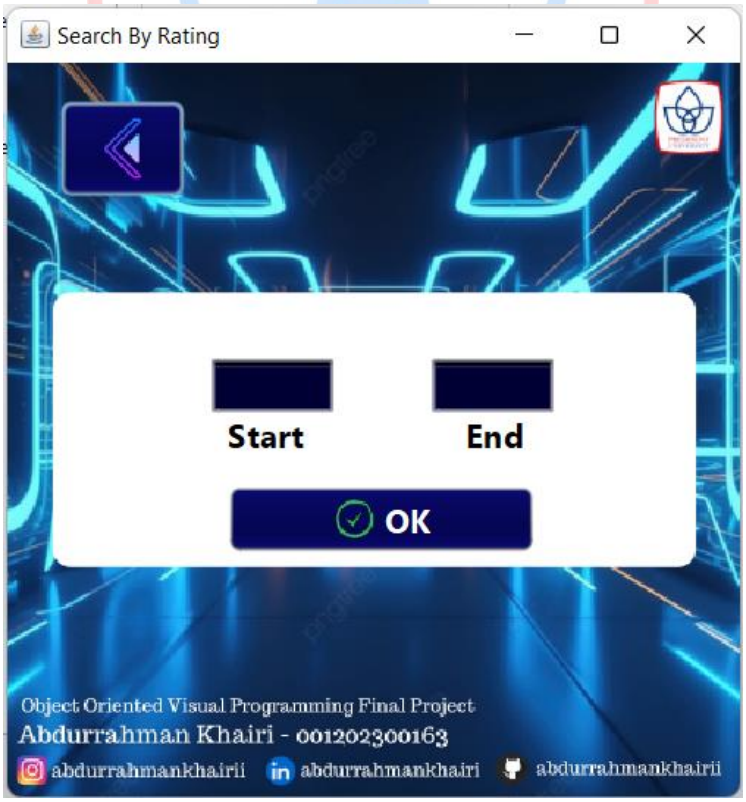

Image 4.3.6.10 View Search Movie By Rating Form

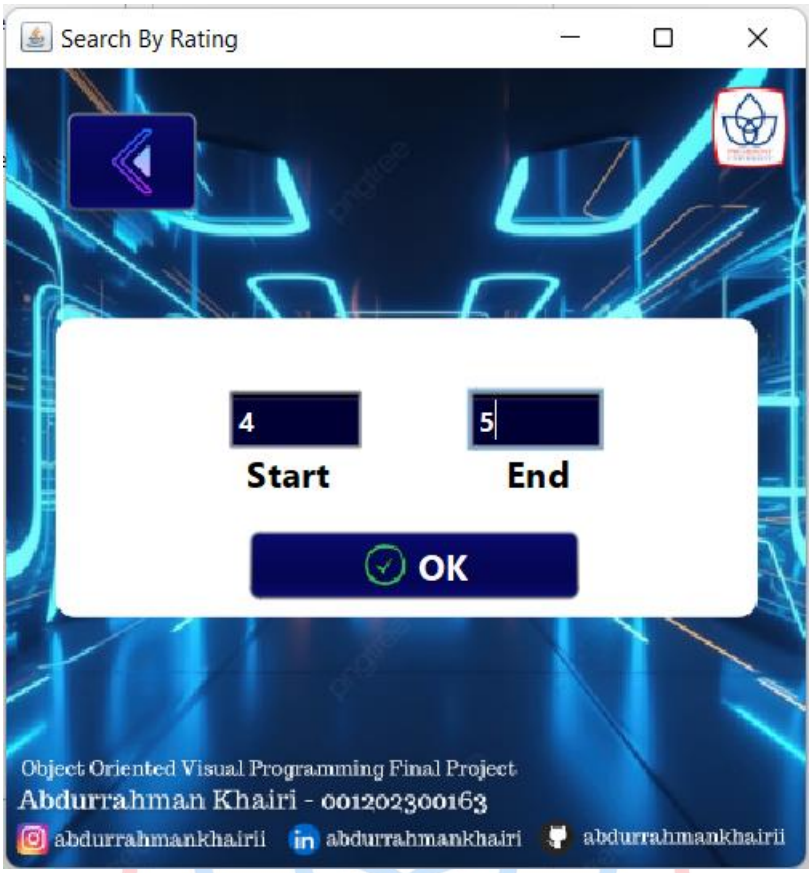

Image 4.3.6.11 Input Search Movie By Rating Form

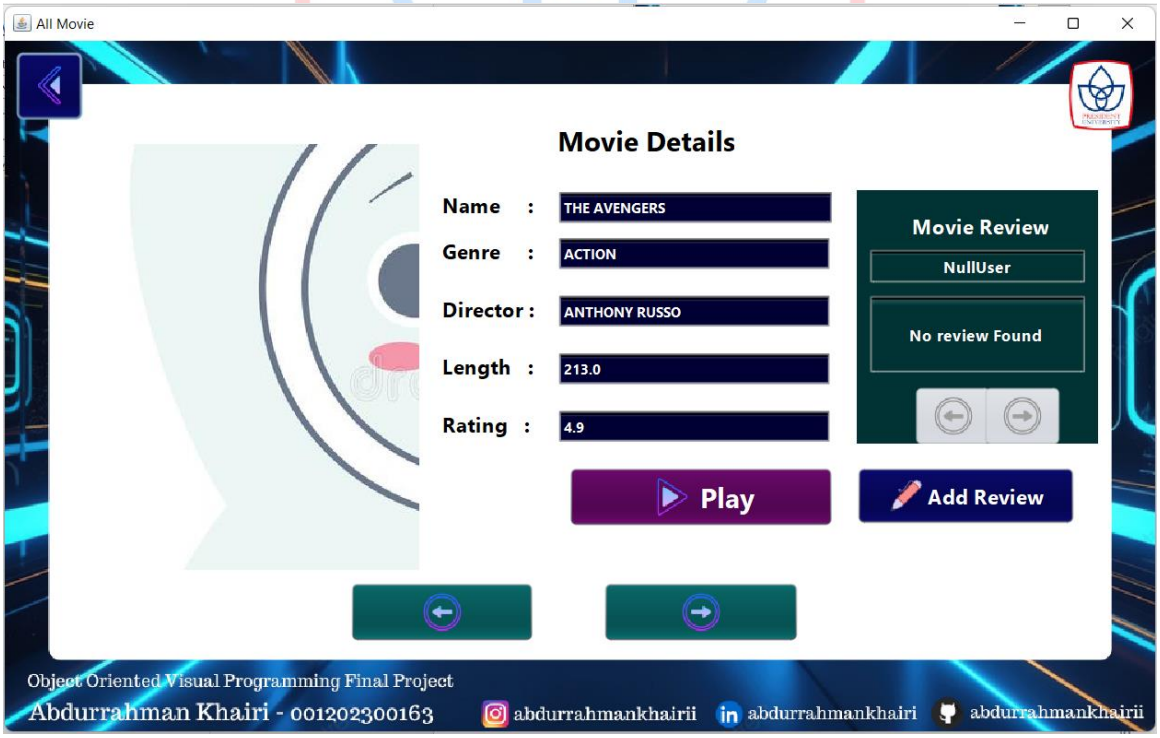

Image 4.3.6.12 Display Movie Details By Searching Rating Form

4.3.7 ContactUs

Image 4.3.7.1 View Contact Us

4.3.8 RequestMovie

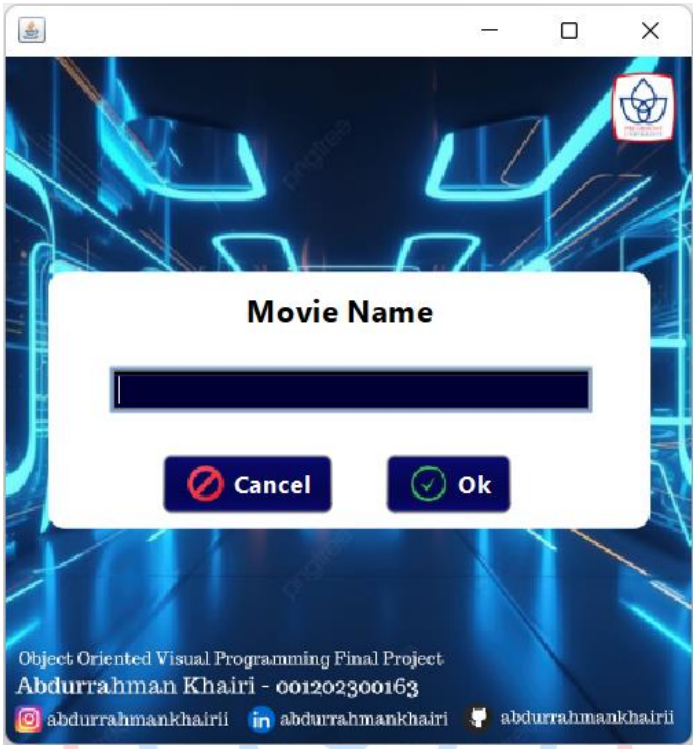

Image 4.3.8.1 View Request Movie Form

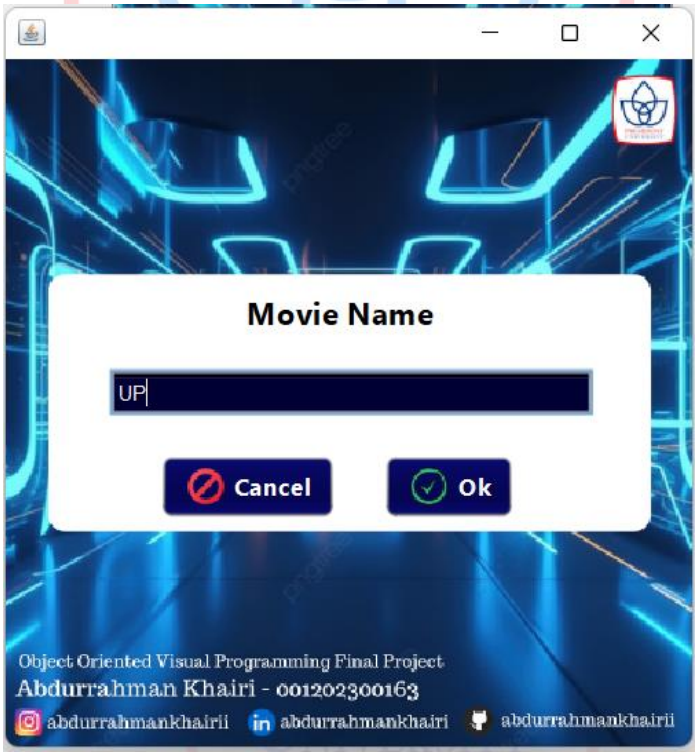

Image 4.3.8.2 Input Request Movie Form

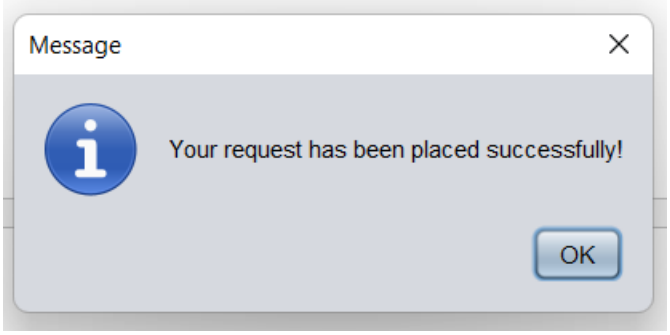

Image 4.3.8.3 Request Send Successfully

4.3.9 LogOutUser

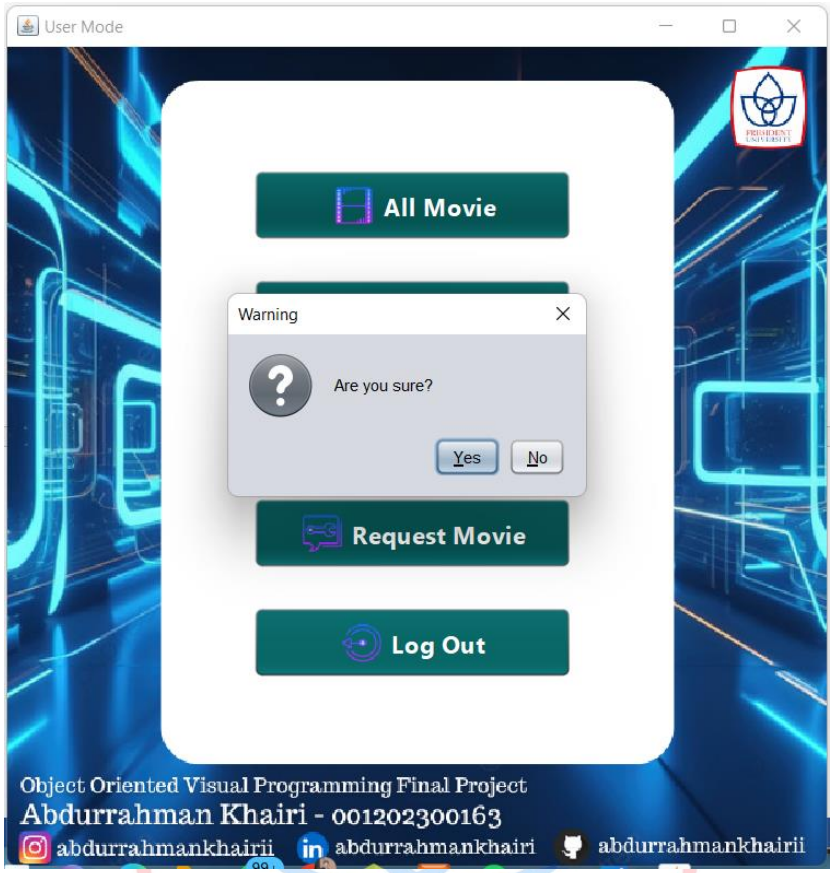

Image 4.3.8.4 LogOut User

BAB V
CONCLUSION

5.1 CONCLUSION

In conclusion, the development of a movie ticketing application using Java NetBeans and a MySQL database offers a comprehensive solution to streamline the movie-watching experience. This user-friendly application empowers users to effortlessly book tickets, access detailed movie information, and discover a wide variety of films.

The application addresses the shortcomings of traditional ticketing methods by providing a convenient and efficient platform for booking movie tickets. NetBeans IDE's intuitive interface and Java's object-oriented programming capabilities facilitate a seamless development process, resulting in a robust and scalable application.

5.2 BENEFITS

The implementation of this movie ticketing application offers a multitude of benefits for users, including:

a. **Simplified Movie Ticket Booking**: Users can conveniently book movie tickets from their devices, eliminating the need to queue at cinema counters.
b. **Comprehensive Movie Information**: The application provides users with detailed information about movies, including synopses, trailers, ratings, and user reviews, empowering them to make informed choices.
c. **Diverse Movie Selection**: Users can explore a vast selection of movies from various genres and countries, catering to diverse preferences and cinematic tastes.
d. **Enhanced User Convenience and Satisfaction**: By streamlining the booking process, offering rich movie information, and providing a diverse film selection, the application elevates the overall movie-watching experience for users.

5.3 FUTURE RECOMMENDATION

Future enhancements can be explored to further improve the application's functionality and user experience. These may include:

a. **Integration with Online Payment Gateways**: Implementing secure online payment gateways will allow users to seamlessly purchase tickets within the application.
b. **Advanced Search and Filtering Options**: Adding advanced search and filtering functionalities based on genre, release date, director, cast, and user ratings will enable users to refine their movie searches and discover films that align with their specific preferences.
c. **Seat Selection Functionality**: Integrating seat selection capabilities will empower users to choose their preferred seats during the booking process.
d. **Personalized Recommendations**: Implementing a recommendation system based on user preferences and viewing history will personalize the movie discovery experience for users.
e. **Integration with Social Media Platforms**: Connecting the application with social media platforms will allow users to share upcoming movies, reviews, and recommendations with friends and family.

By incorporating these potential enhancements, the movie ticketing application can evolve into a comprehensive and user-centric platform, further transforming the way users discover, book tickets for, and enjoy movies.