A Project Report

On

"EVENT MANAGEMENT SYSTEM"

Submitted to

CHHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY

BHILAI

Bachelor of Technology

In

Computer Science & Engineering

specialization in

Artificial intelligence and machine learning

By

SABAJ KHAN

MD DAWAR KHAN

MOHAMMAD AYAN KHAN

BHESHEJ NETAM

Under the Guidance of

(Dr). MEGHA MISHRA

Associate Professor (CSE)

Department of Computer Science Engineering

Shri Shankaracharya Technical Campus, Junwani, Bhilai

DECLARATION

We solemnly declare the report of the project work entitled "Face Recognition Based Attendance System" is based on our own work carried **out** during my study under **(Dr.) Megha Mishra, Associate Professor (CSE)** department of **SSTC,** Bhilai.

I assert that the statements made, and conclusion drawn are an outcome of the project work. I further declare that to the best of my knowledge and belief that the report does not contain any part of any work which has been submitted for the award of any other degree / diploma / certificate in this university or any other university.

Sabaj khan
Roll No: 301410923086
Enrollment No: CD7708

MD Dawar khan
Roll No: 301410923061
Enrollment No:CD7730

Mohammad Ayan khan
Roll No: 301410923125
Enrollment No:CD8907

Bheshej netam
Roll No: 301410923037
Enrollment No: CD7706

CERTIFICATE

This is to certify the report of the project submitted is an outcome of the project work entitled:"

Event management system" carried out by

Sabaj khan bearing roll no: **301410923086, MD Dawar khan bearing** roll no: **301410923061, Mohammad Ayan khan** bearing roll no: **301410923125, Bheshej netam** bearing roll no: **301410923037** and carried out under my guidance and supervision for the award of Degree in Bachelor of Technology in Computer Science and Engineering (SSTC) of Chhattisgarh Swami Vivekanand Technical University, Bhilai (CG).

To the best of my knowledge the report:

- Embodies the work of the candidate himself/herself.
- Has duly been completed.
- Fulfils the requirements of Ordinance relating to the BE degree of the University.
- Is up to the desired standards for the purpose of which it is submitted.

Dr. Abha Choubey	**(Dr.) Megha Mishra**
Professor	**Associate Professor (CSE)**
HOD CSE(Specialization)	
(Signature of HOD with seal)	(Signature of the Guide)

The project work mentioned above hereby recommended and forwarded for examination and evaluation.

Date:

CERTIFICATE BY THE EXAMINERS

This is to certify that the project work entitled "**Event Management System**"
Submitted by

Sabaj khan	Roll no. 301410923086	Enrollment no. CD7708
MD Dawar khan	Roll no. 301410923061	Enrollment no.CD7730
Mohammad Ayan khan	Roll no. 301410923125	Enrollment no.CD8907
Bheshej Netam	Roll no. 301410923037	Enrollment no. CD7706

"Has been examined by the undersigned as a part of the examination for the award of Bachelor of Engineering degree in Computer Science & Engineering of Chhattisgarh Swami Vivekanand Technical University, Bhiai.

Internal Examiner

Date: 05-05-2025

External Examiner Date

Date: 18-07-2025

ACKNOWLEDGEMENT

Working on the project was a great experience for us. There were moments of anxiety when we could not solve our problem for several days. But we enjoyed every bit of the process and are thankful to all the people associated with us during this period.

We convey our sincere thanks to our project guide (**Dr.) Megha Mishra,** Associate Professor for providing tats all sorts of facilities. His support and guidance helped us to carry out the project. We owe a great department of his gratitude for his constant advice. support, cooperation and encouragement throughout the project.

We would like to express our deep gratitude to our respected (**Dr.) Megha Mishra** for her help and support. We also pay special thanks for his helpline solution and comments enriched by her experience.

We would also like to express our deep gratitude to our college management **Shri I.P. Mishra, Chairman (Shri Gangajali Education** Society, **Bhilai), Mrs. Jaya Mishra, President (Shri Gangajali Education Society**, **Bhilai), Dr. P.B. Deshmukh, Director, SSTC and Dr. Samta Gajbhiye** (**H.O.D. CSE Department (Specialization)** for providing an educational ambience. It will be our pleasure to acknowledge, utmost cooperation and valuable suggestions from time to time given by our staff members of' our department to whom we owe our entire computer knowledge and also we would like to thank all those persons who have directly or indirectly helped us by providing books and computer peripherals and other necessary amenities which helped in the development of this project.

SABAJ KHAN
MD DAWAR KHAN
MOHAMMAD AYAN KHAN
BHESHEJ NETAM

INDEX

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Event management has always played a crucial role in organizing and coordinating various activities in academic, corporate, and social environments. With the increasing frequency and complexity of events, manual methods of managing event-related data are becoming obsolete and inefficient. The traditional paper-based approach often leads to disorganized information, participant mismanagement, and delayed communications. As digital transformation advances in all areas, there is a pressing need for robust and user-friendly event management systems that can handle these challenges with ease and efficiency.

The **Event Management System** developed as part of this project is a desktop application built using Java SE (Standard Edition) and MySQL for backend storage. It is designed to streamline the process of creating, managing, and participating in events through an interactive graphical user interface (GUI). This system serves as a centralized platform where administrators can create new events by specifying details such as event name, date, location, and description. At the same time, users can register themselves and enroll in events of their interest using their basic information like name and email.

The system architecture is modular, ensuring a clear separation between data handling (via the MySQL database), business logic (managed by the Java service layer), and the presentation layer (implemented using Java Swing GUI). This layered design allows easy scalability and maintenance, making it adaptable for future improvements. Moreover, the use of Java provides platform independence, enabling deployment across different operating systems such as Windows, Linux, or macOS.

In educational institutes and organizations where numerous events are conducted regularly—such as seminars, workshops, and cultural programs—the need for an efficient digital solution is paramount. The Event Management System not only reduces administrative workload but also enhances the user experience by providing clear, real-time information and confirmations. It includes features such as listing all created events, viewing participants, and automated error checking to avoid duplicate or invalid entries.

1.2 SCOPE

The **Event Management System** is developed with the primary objective of addressing the operational and logistical challenges faced in organizing and managing events, particularly in academic and institutional settings. However, its design, functionality, and modular nature make it versatile enough to be extended for broader use cases in various industries such as corporate organizations, non-profit groups, conference organizers, and even personal event planners.

This system enables users to seamlessly create, update, and manage event-related data through an intuitive Java-based graphical user interface. Administrators can input critical event information including the name, date, venue, and a detailed description, which is then stored in a MySQL database for persistence and reliability. On the other hand, users can register with basic details like name and email address and enroll themselves for events of interest. The system ensures that these tasks are executed efficiently, reducing dependency on paper-based processes and eliminating common issues such as duplicate records, manual entry errors, and inefficient communication.

The scope of this project is not limited to a single use case or domain. It is built to handle a wide variety of event types such as academic seminars, webinars, technical workshops, student registrations for cultural fests, and community gatherings. With minor modifications, the same system can be customized to manage corporate training sessions, team-building activities, public campaigns, or private events like birthday parties and weddings.

This application is also intended to be a scalable solution. The current implementation focuses on core functionalities such as event creation, user registration, and participant-event mapping. However, the architecture allows easy extension of features like admin-user role segregation, automated email confirmations, advanced filtering of events, real-time event capacity tracking, and even analytics to evaluate event success rates or participation trends.

Additionally, the system is suitable for small to medium-scale organizations that require a lightweight yet effective tool without the overhead of large-scale enterprise software. Its offline nature, running as a standalone Java application, makes it especially useful in institutions with limited internet access or infrastructure.

1.3 PROBLEM STATEMENT

In today's fast-paced and increasingly digital environment, the manual management of events has become highly inefficient and error-prone. Educational institutions, businesses, and various organizations frequently organize events such as seminars, workshops, conferences, training programs, and recreational activities. Traditionally, the process of organizing such events involves numerous manual tasks including event planning, maintaining attendance lists, communicating with participants, and handling registration forms. These activities, when performed without digital assistance, often lead to a wide range of challenges that compromise the effectiveness and overall experience of the event. Some of the major issues encountered in manual event management systems include the loss of participant data, human errors in record keeping, difficulty in managing large numbers of attendees, and the absence of real-time updates or confirmations. Moreover, tracking attendance, generating reports, and ensuring smooth communication between organizers and participants becomes a cumbersome task without an automated system in place. As events grow in scale and frequency, these issues multiply, resulting in inefficiencies that hinder productivity and organization.

Another common problem is the lack of a centralized platform to store and retrieve event-related information. Without a systematic database, organizers often face difficulties in accessing past records, evaluating event participation, or maintaining consistency across events. In academic environments, for instance, it becomes challenging for departments to monitor student engagement or participation history without a structured system. This lack of transparency and structure can lead to decreased participation and dissatisfaction among users.

Furthermore, the absence of an interactive user interface makes it difficult for potential participants to browse, understand, and register for events conveniently. Physical registration methods or informal communication (e.g., emails, word-of-mouth) are often inadequate in ensuring full outreach or timely responses, especially when dealing with a large user base.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 SYSTEM REQUIREMENT

2.1.1 HARDWARE REQUIREMENTS

- Intel Core i3 3rd gen processor or later.
- 512 MB disk space.
- 512 MB RAM.

2.1.2 SOFTWARE REQUIREMENTS

- Operating System: Windows/Linux/MacOS
- Java JDK 8 or higher
- MySQL Server
- IntelliJ IDEA / Eclipse IDE - MySQL Workbench

2.2 SYSTEM FEATURES

- Create new events with name, date, location, and description.
- View all created events in a table format.
- Register new users with name and email.
- Register users to specific events.
- Error handling for invalid inputs.
- Confirmation dialogs for successful operations.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

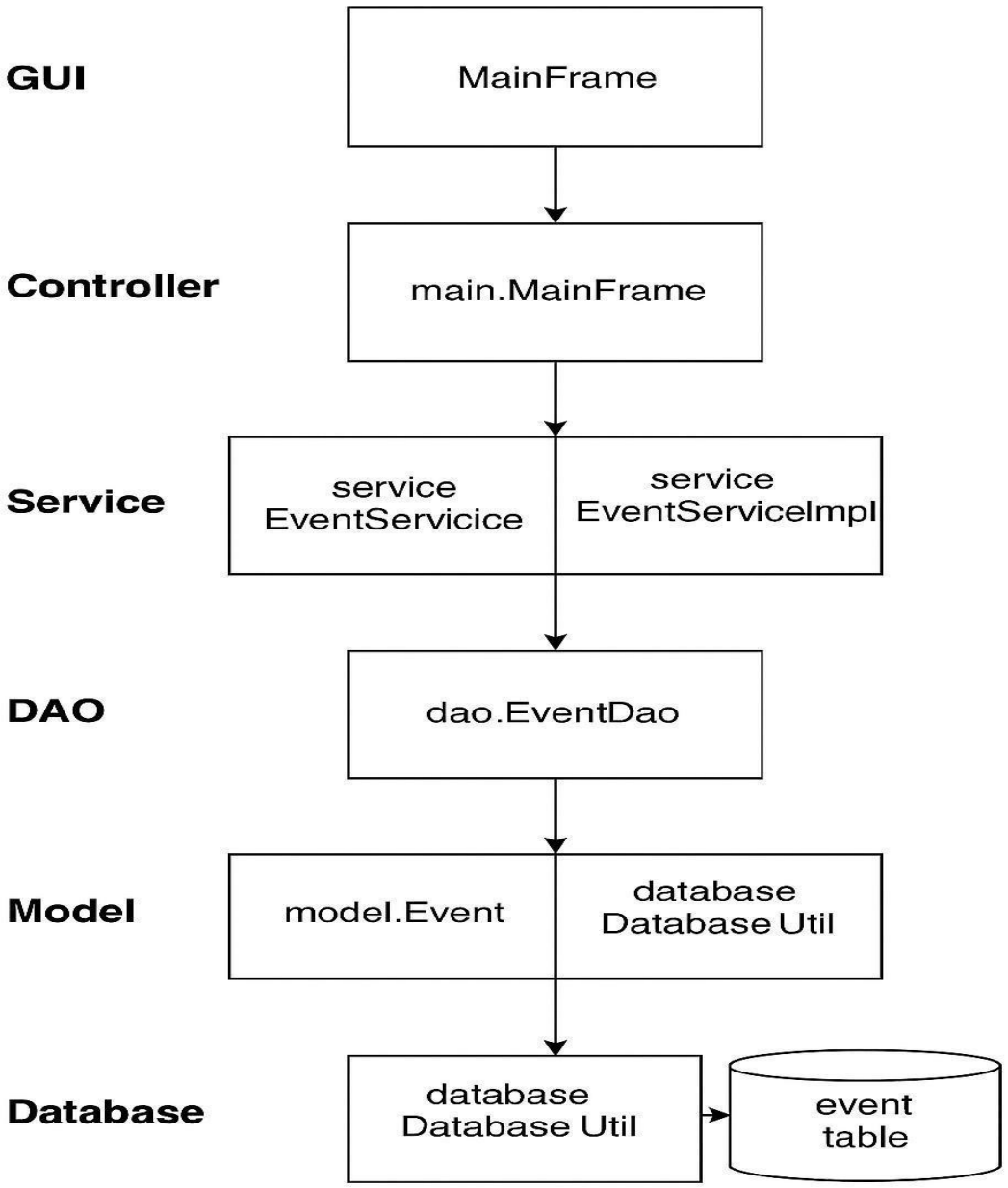

Fig : System Architecture for Event Management System

.

3.2 MODULES IN THE SYSTEM

- **Event Management System** is designed using a modular architecture to ensure clarity, scalability, and ease of maintenance. Each module represents a specific functionality within the system, allowing for separation of concerns and better organization of the codebase. The modular approach also facilitates future expansion by enabling developers to add or modify features without affecting the entire system. The core modules integrated into the application are described below:

- This is the central module of the system that handles all operations related to events. It allows administrators to create new events by providing necessary details such as the event name, date, location, and a descriptive summary. The module is designed to ensure data integrity with input validation mechanisms, preventing the creation of incomplete or duplicate event entries. Events are stored in the MySQL database for persistent access, and each event is assigned a unique identifier to distinguish it during the registration and retrieval processes. The system also allows viewing all stored events in a tabular format through the GUI, aiding in quick reference and management.

- The User Registration module is responsible for enrolling participants into the system. It captures essential user information such as name and email address through a clean, form-based interface. To maintain data accuracy and prevent redundancy, the module includes validations like checking for email format and uniqueness. Once registered, user data is stored in the backend database and can be accessed later for participation in events. This module ensures that only valid and properly registered users are eligible to register for events, which also aids in traceability and report generation.

- This module facilitates the process of linking users with events. Registered users can select an event they wish to join by entering their email and the corresponding event ID. The system checks the validity of the user and event before allowing the registration to proceed. If a user does not exist in the system, a prompt will guide them to register first. The registration details are then recorded in a junction table that maps users to events, enabling efficient tracking of who is participating in which event. This module plays a critical role in managing attendance and generating participant lists.

- These three modules form the backbone of the **Event Management System**. Their interaction enables a smooth flow of data and ensures an effective user experience for both administrators and participants. Future enhancements such as user authentication, role-based access control, notifications, and analytics can be integrated by building upon these well-defined core modules.

3.3 USE CASE DIAGRAM

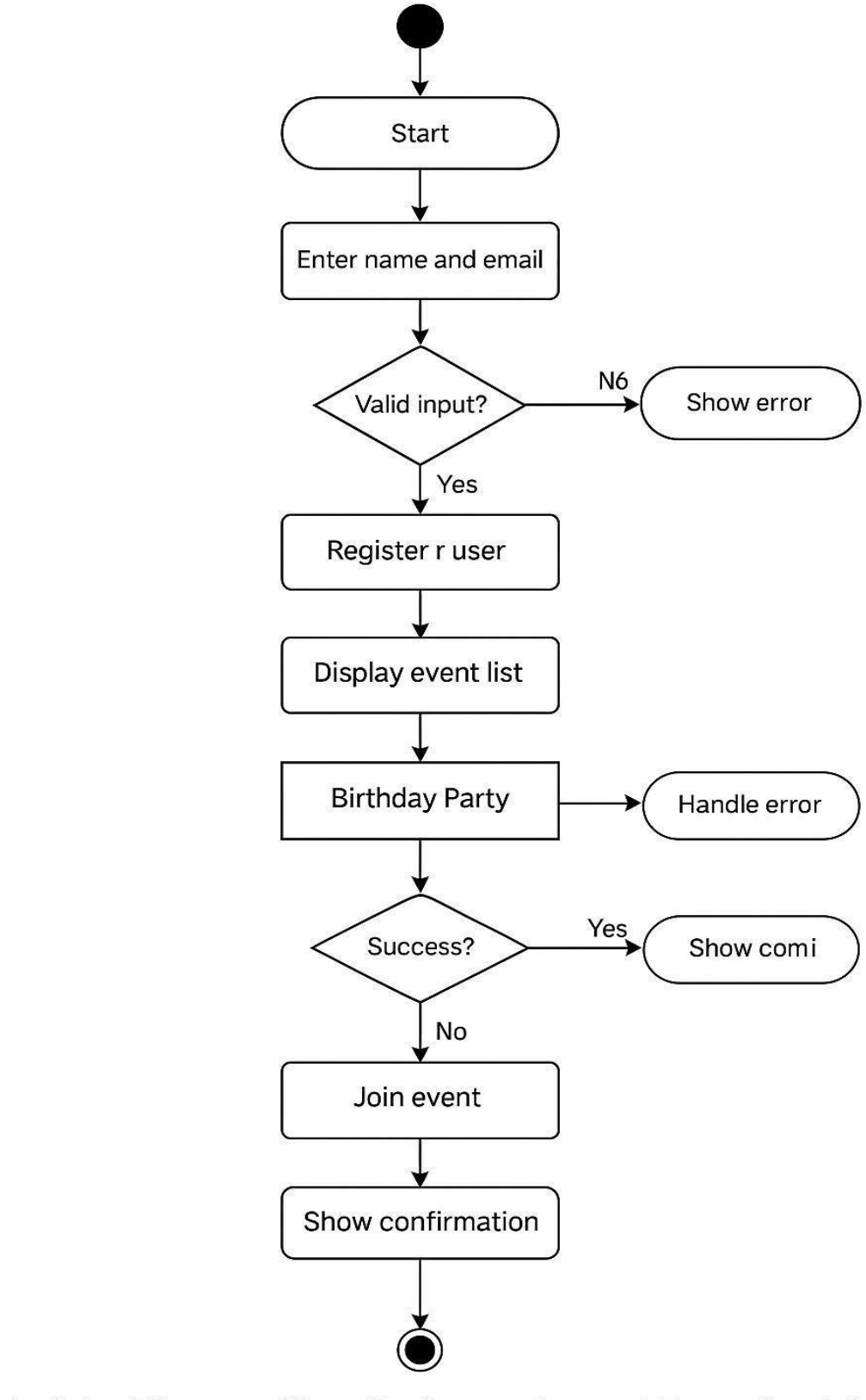

Fig : Use Case Diagram for Event Management System by Joining a Birthday party event

3.4 ACTIVITY DIAGRAM

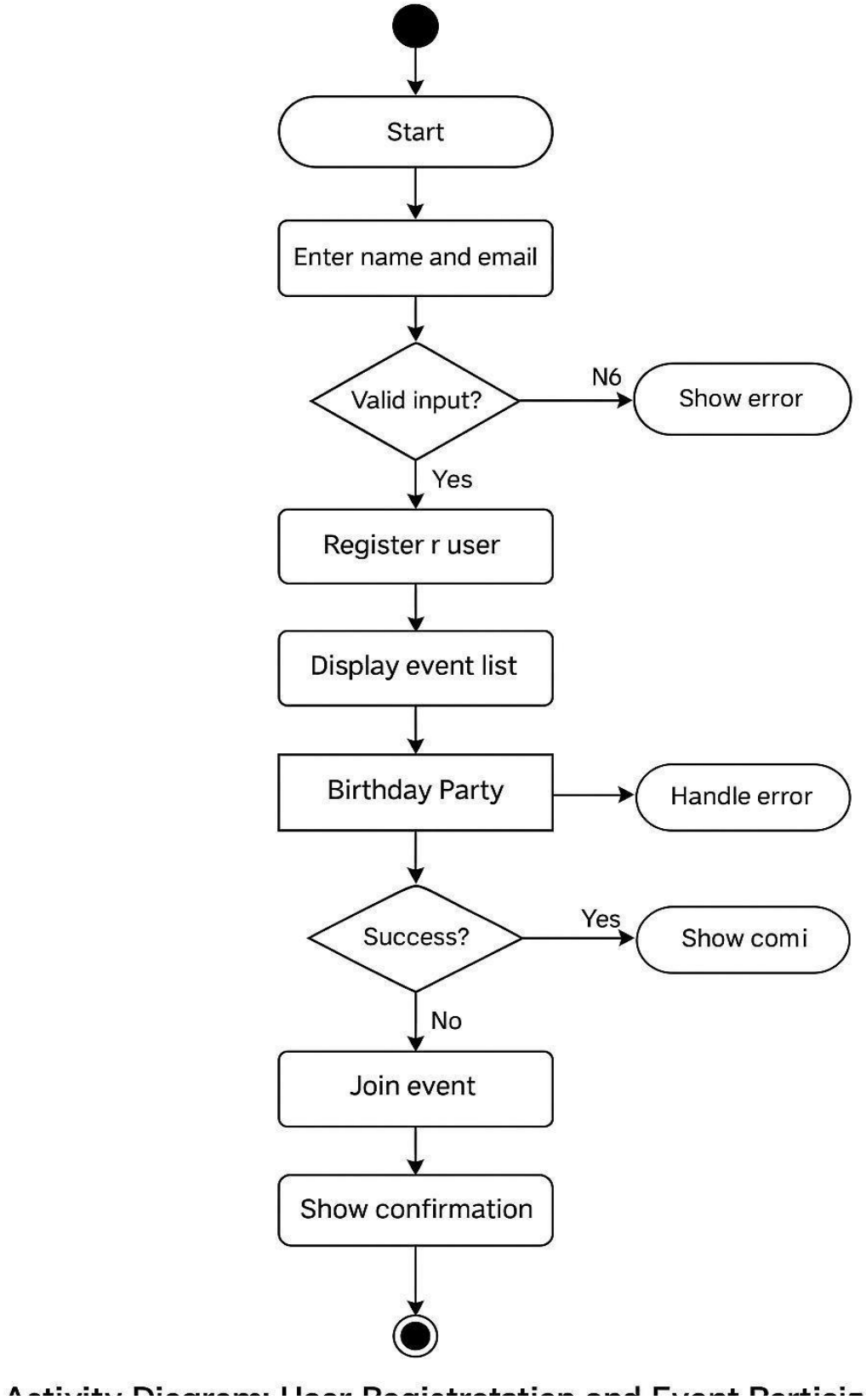

Fig : Internal Working Diagram Using Joing Event named Birthday Party event

CHAPTER 4
IMPLEMENTATION

4.1 CODE SNIPPETS

Java Code Snippets

Inside src package

- **Inside main package:**

MainFrame.java - package main;

```
import javax.swing.*; import
java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MainFrame extends JFrame {

    public MainFrame() {
        setTitle("Event Management System");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null); // Center window

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(5, 1, 10, 10));

        JButton createEventButton = new JButton("Create Event");
        JButton viewEventsButton = new JButton("View Events");
        JButton registerUserButton = new JButton("Register User");
        JButton registerToEventButton = new JButton("Register to Event");
        JButton exitButton = new JButton("Exit");

        panel.add(createEventButton);
panel.add(viewEventsButton);         panel.add(registerUserButton);
panel.add(registerToEventButton);         panel.add(exitButton);

        add(panel);

        // Action Listeners
        createEventButton.addActionListener(e -> {
            new gui.CreateEventFrame().setVisible(true);
        });

        viewEventsButton.addActionListener(e -> {
            new gui.ViewEventsFrame().setVisible(true);
        });
```

```
        registerUserButton.addActionListener(e -> {
            new gui.RegisterUserFrame().setVisible(true);
        });

        registerToEventButton.addActionListener(e -> {
            new gui.RegisterToEventFrame().setVisible(true);
        });

        exitButton.addActionListener(e -> System.exit(0));
    }

    public static void main(String[] args) {
SwingUtilities.invokeLater(() -> {
            new MainFrame().setVisible(true);
        });
    }
}
```

- **Inside GUI package:**

CreateEventFrame.java

```
package gui;

import service.EventService;

import javax.swing.*;
import java.awt.*;

public class CreateEventFrame extends JFrame {

    private JTextField nameField, dateField, locationField;
private JTextArea descriptionArea;
    private EventService eventService = new EventService();

    public CreateEventFrame() {         setTitle("Create
New Event");
        setSize(400, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new GridLayout(5, 2, 10, 10));

        // Components
        panel.add(new JLabel("Event Name:"));
nameField = new JTextField();
        panel.add(nameField);
```

```
        panel.add(new JLabel("Event Date (yyyy-mm-dd):"));

        dateField = new JTextField();

        panel.add(dateField);


        panel.add(new JLabel("Location:"));
locationField = new JTextField();
panel.add(locationField);          panel.add(new
JLabel("Description:"));

descriptionArea = new JTextArea();

        panel.add(new JScrollPane(descriptionArea));


        JButton createButton = new JButton("Create Event");
panel.add(createButton);


        add(panel);


        // Action Listener

        createButton.addActionListener(e -> {
String name = nameField.getText();

            String date = dateField.getText();

            String location = locationField.getText();

            String description = descriptionArea.getText();


            if (name.isEmpty() || date.isEmpty() || location.isEmpty() ||

description.isEmpty()) {

                JOptionPane.showMessageDialog(this, "❌ All fields are required!",

"Error", JOptionPane.ERROR_MESSAGE);

            } else {

                eventService.createEvent(name, date, location, description);
JOptionPane.showMessageDialog(this, "☑ Event created  successfully!", "Success",
JOptionPane.INFORMATION_MESSAGE);

dispose(); // Close window after creation

            }

        });

    }

}
```

ViewEventsFrame- package
gui;

import model.Event; import

service.EventService; import

javax.swing.*;

import javax.swing.table.DefaultTableModel; import

java.awt.*;

import java.util.List;

public class ViewEventsFrame extends JFrame {

```
    private EventService eventService = new EventService();
private JTable eventsTable;

    public ViewEventsFrame() {
setTitle("View All Events");         setSize(600,
400);          setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
setLocationRelativeTo(null);

        // Table Model
        DefaultTableModel tableModel = new DefaultTableModel();
tableModel.addColumn("Event ID");
tableModel.addColumn("Name");         tableModel.addColumn("Date");
tableModel.addColumn("Location");
        tableModel.addColumn("Description");

        // Load Events
        List<Event> events = eventService.getAllEvents();
        for (Event event : events) {
            tableModel.addRow(new Object[]{
event.getEventId(),
event.getName(),                    event.getDate(),
event.getLocation(),
                    event.getDescription()
            });
        }

        eventsTable = new JTable(tableModel);
      JScrollPane scrollPane = new JScrollPane(eventsTable); add(scrollPane,
              BorderLayout.CENTER);
    }
}
```

RegisterUserFrame.java-

```
package gui;

import service.UserService;

import javax.swing.*;
import java.awt.*;

public class RegisterUserFrame extends JFrame {

    private JTextField nameField, emailField;     private
UserService userService = new UserService();

    public RegisterUserFrame() {
setTitle("Register New User");
        setSize(400, 300);
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new GridLayout(3, 2, 10, 10));

        // Components
        panel.add(new JLabel("Name:"));         nameField
= new JTextField();
        panel.add(nameField);

        panel.add(new JLabel("Email:"));
emailField = new JTextField();
        panel.add(emailField);

        JButton registerButton = new JButton("Register");
panel.add(registerButton);

      add(panel);
     // Action Listener
registerButton.addActionListener(e -> {
String name = nameField.getText();
            String email = emailField.getText();

            if (name.isEmpty() || email.isEmpty()) {
                JOptionPane.showMessageDialog(this, "❌ All fields are required!",
"Error", JOptionPane.ERROR_MESSAGE);
            } else {
                userService.registerUser(name, email);
                JOptionPane.showMessageDialog(this, "☑ User registered
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
dispose(); // Close window after registration
            }
        });
    }
}
```

RegisterToEventFrame.java- package gui;

```
import service.RegistrationService;
import service.UserService;

import javax.swing.*;
import java.awt.*;

public class RegisterToEventFrame extends JFrame {

    private JTextField emailField, eventIdField;     private UserService userService =
new UserService();      private RegistrationService registrationService = new
RegistrationService();
```

```
    public RegisterToEventFrame() {
setTitle("Register User to Event");         setSize(400,
300);          setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
setLocationRelativeTo(null);
      JPanel panel = new JPanel(new GridLayout(3, 2, 10, 10));

        // Components
        panel.add(new JLabel("User Email:"));
emailField = new JTextField();
        panel.add(emailField);

        panel.add(new JLabel("Event ID:"));
eventIdField = new JTextField();
        panel.add(eventIdField);

        JButton registerButton = new JButton("Register");
panel.add(registerButton);

        add(panel);

        // Action Listener
        registerButton.addActionListener(e -> {
            String email = emailField.getText();
            String eventIdStr = eventIdField.getText();

            if (email.isEmpty() || eventIdStr.isEmpty()) {
                JOptionPane.showMessageDialog(this, "❌ All fields are required!",
"Error", JOptionPane.ERROR_MESSAGE);
            } else {
try {
                    int eventId = Integer.parseInt(eventIdStr);
int userId = userService.getUserIdByEmail(email);

                    if (userId == -1) {
                        JOptionPane.showMessageDialog(this, "❌ User not found!
Please register first.", "Error", JOptionPane.ERROR_MESSAGE);
                    } else {
                        registrationService.registerUserToEvent(userId, eventId);
JOptionPane.showMessageDialog(this, "☑ User registered to  event
successfully!", "Success", JOptionPane.INFORMATION_MESSAGE);
                        dispose(); // Close window after registration
                    }
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(this, "❌ Event ID must be a
number.", "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
```

```
        });
    }
}
```

- **Inside service package:**

EventService.java- package service;

```
import dao.EventDAO;
import model.Event;

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

public class EventService {
    private EventDAO eventDAO = new EventDAO();

    public void createEvent(String name, String dateStr, String location, String
description) {
        try {
            Date date = Date.valueOf(dateStr);
            Event event = new Event(name, date, location, description);
eventDAO.addEvent(event);
        } catch (Exception e) {
            System.out.println("❌ Invalid event data: " + e.getMessage());
        }
    }
    public List<Event> getAllEvents() {
        return eventDAO.getAllEvents();
    }
}
```

UserService.java- package service;

```
import dao.UserDAO;
import model.User;

public class UserService {
    private UserDAO userDAO = new UserDAO();

    public void registerUser(String name, String email) {
        if (email.contains("@")) {
            User user = new User(name, email);
userDAO.addUser(user);
```

```
        } else {
            System.out.println("❌ Invalid email address.");
        }
    }

    public int getUserIdByEmail(String email) {
        return userDAO.getUserIdByEmail(email);
    }
}
```

RegisterService.java-

```
package service;

import dao.RegistrationDAO;
import model.Registration;

public class RegistrationService {
    private RegistrationDAO registrationDAO = new RegistrationDAO();

    public void registerUserToEvent(int userId, int eventId) {
if (userId <= 0 || eventId <= 0) {
System.out.println("❌ Invalid user or event ID.");             return;
}
        Registration registration = new Registration(userId, eventId);
registrationDAO.registerUserToEvent(registration);
    }
}
```

- **Inside model package:**

Event.java-

```
package model;

import java.sql.Date;

public class Event {
private int eventId;
private String name;
private Date date;
private String location;
    private String description;

    public Event(int eventId, String name, Date date, String location, String
description) {
        this.eventId = eventId;
this.name = name;         this.date
= date;         this.location =
location;
        this.description = description;
    }
```

```
    public Event(String name, Date date, String location, String description) {
this.name = name;         this.date = date;         this.location = location;
        this.description = description;
    }
    // Getters and Setters
    public int getEventId() { return eventId; }
    public void setEventId(int eventId) { this.eventId = eventId; }

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

    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }

    public String getLocation() { return location; }
    public void setLocation(String location) { this.location = location; }

    public String getDescription() { return description; }     public
void setDescription(String description) { this.description = description; }
}
```

User.java

```
package model;

public class User {
private int userId;
private String name;
    private String email;

    public User(int userId, String name, String email) {
        this.userId = userId;
this.name = name;
        this.email = email;
    }

    public User(String name, String email) {
this.name = name;
        this.email = email;
    }

    // Getters and Setters
    public int getUserId() { return userId; }
    public void setUserId(int userId) { this.userId = userId; }

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

```
    public String getEmail() { return email; }

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

}
```

Registration.java- package model; public class Registration {

```
private int regId;     private
int userId;

    private int eventId;


    public Registration(int regId, int userId, int eventId) {

this.regId = regId;         this.userId = userId;

        this.eventId = eventId;

    }


    public Registration(int userId, int eventId) {

        this.userId = userId;

        this.eventId = eventId;

    }


    // Getters and Setters     public int
getRegId() { return regId; }

    public void setRegId(int regId) { this.regId = regId; }


    public int getUserId() { return userId; }

    public void setUserId(int userId) { this.userId = userId; }


    public int getEventId() { return eventId; }

    public void setEventId(int eventId) { this.eventId = eventId; }

}
```

- **Inside dao package:**

EventDAO.java

```
package dao;


import model.Event; import
java.sql.*; import
java.util.ArrayList;

import java.util.List;

import database.DatabaseConnection;


public class EventDAO {


    public void addEvent(Event event) {

        String sql = "INSERT INTO events (name, date, location, description)

VALUES (?, ?, ?, ?)";
```

```
        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, event.getName());
stmt.setDate(2, event.getDate());             stmt.setString(3, event.getLocation());
            stmt.setString(4, event.getDescription());

            stmt.executeUpdate();
            System.out.println("☑ Event added successfully!");

        } catch (SQLException e) {
e.printStackTrace();
        }
    }

    public List<Event> getAllEvents() {
List<Event> events = new ArrayList<>();
        String sql = "SELECT * FROM events";

        try (Connection conn = DatabaseConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Event event = new Event(
rs.getInt("event_id"),
rs.getString("name"),                         rs.getDate("date"),
rs.getString("location"),
                        rs.getString("description")
                );
                events.add(event);
            }

        } catch (SQLException e) {
e.printStackTrace();
        }
        return events;
    }
}
```

UserDAO.java- package dao;

import model.User; import java.sql.*;

import database.DatabaseConnection;

```
public class UserDAO {

    public void addUser(User user) {
```

```
        String sql = "INSERT INTO users (name, email) VALUES (?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, user.getName());
stmt.setString(2, user.getEmail());             stmt.executeUpdate();
            System.out.println("☑ User added successfully!");

        } catch (SQLException e) {
e.printStackTrace();
        }
    }

    public int getUserIdByEmail(String email) {
        String sql = "SELECT user_id FROM users WHERE email = ?";         try
(Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {
stmt.setString(1, email);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getInt("user_id");
            }

        } catch (SQLException e) {
e.printStackTrace();
        }
        return -1;
    }
}
```

RegistrationDAO.java-

```
package dao;

import model.Registration; import

java.sql.*;

import database.DatabaseConnection;

public class RegistrationDAO {

    public void registerUserToEvent(Registration reg) {
        String sql = "INSERT INTO registrations (user_id, event_id) VALUES (?,  ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, reg.getUserId());
            stmt.setInt(2, reg.getEventId());
```

```
            stmt.executeUpdate();
            System.out.println("☑ Registration successful!");

        } catch (SQLException e) {
e.printStackTrace();
        }
    }
}
```

- **Inside database package:**

DatabaseConnection.java

```
package database;

import java.sql.Connection; import
java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/eventdb";
private static final String USER = ""; // apna DB username
    private static final String PASSWORD = ""; // apna DB password

    public static Connection getConnection() {
try {
            Connection conn = DriverManager.getConnection(URL, USER,  PASSWORD);
            System.out.println("☑ Database connected successfully!");
return conn;
        } catch (SQLException e) {
            System.out.println("❌ Failed to connect to database.");
e.printStackTrace();             return null;
        }
    }
}
```

MySQl WorkBench /cmd code snippets

EventManagementSystem.sql- (same level as src)

```
-- Database: EventManagementSystem

CREATE DATABASE IF NOT EXISTS EventManagementSystem;
USE EventManagementSystem;

-- Table: users
```

```
CREATE TABLE IF NOT EXISTS users (     id INT
AUTO_INCREMENT PRIMARY KEY,     name
VARCHAR(100) NOT NULL,

    email VARCHAR(100) UNIQUE NOT NULL

);


-- Table: events

CREATE TABLE IF NOT EXISTS events (     id INT
AUTO_INCREMENT PRIMARY KEY,     name VARCHAR(100)
NOT NULL,     date

DATE NOT NULL,     location VARCHAR(150),

    description TEXT

);


-- Table: registrations

CREATE TABLE IF NOT EXISTS registrations (

    id INT AUTO_INCREMENT PRIMARY KEY,
user_id INT,      event_id INT,

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,

    FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE

);
```

4.2 Inside lib Folder (same level as src) **mysql-connector-j-9.3.0.jar**

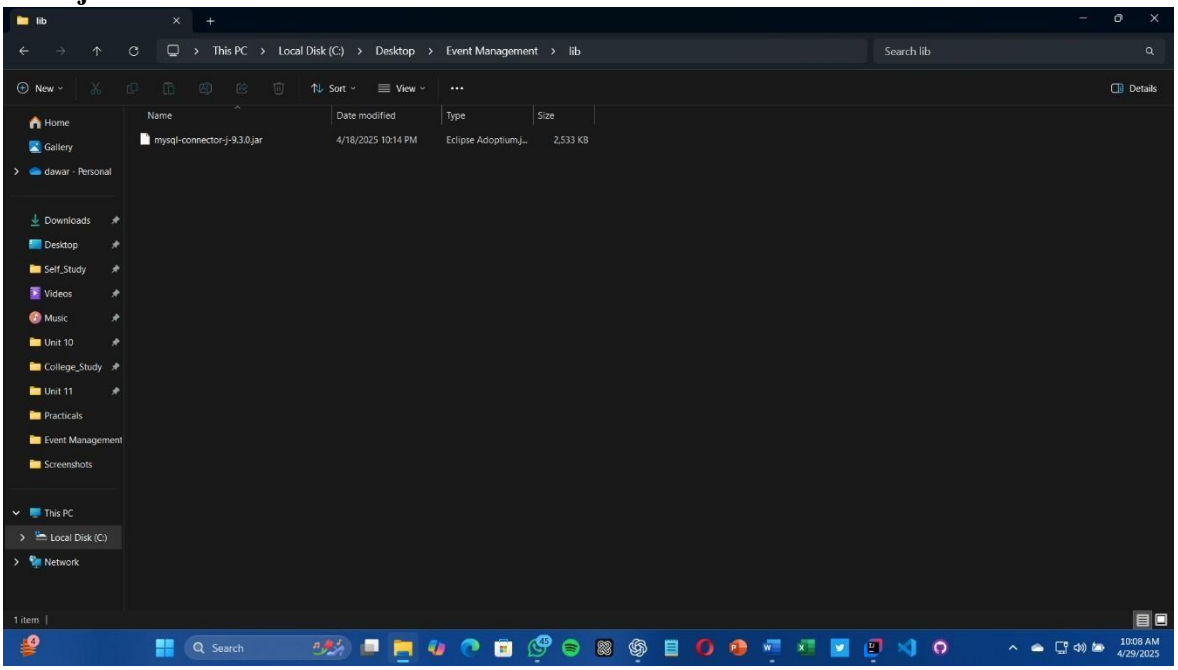

Fig4.2.1 : Download MySql Driver from Internet and Add .jar file of MySql Driver inside lib folder (same level as src) of to connect with Database

4.3 Check all Java files Working or not

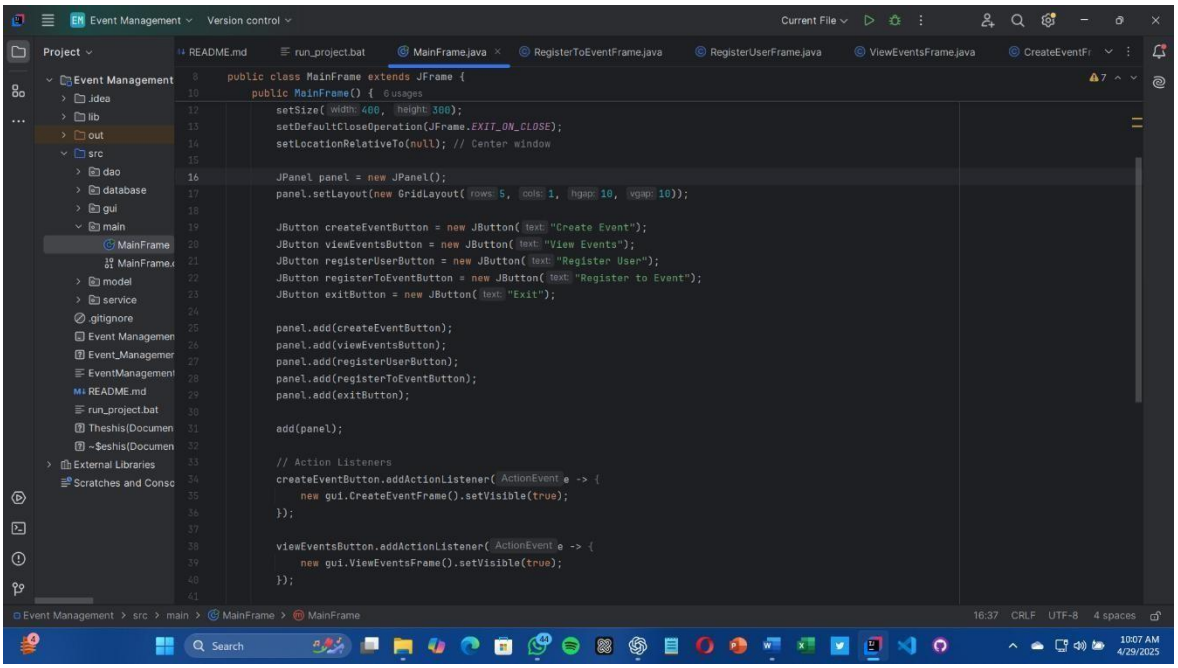

Fig4.3.2: Check all necessary .java files are ready and work properly

Compile and Run Java Application

Open the Java Project

In Terminal

Cd src

To Compile : javac -encoding UTF-8 -cp ".;../lib/mysqlconnector-j-9.3.0.jar" main/MainFrame.java gui/*.java dao/*.java service/*.java model/*.java database/*.java **To Run:** java -cp ".;../lib/mysql-connector-j-9.3.0.jar"
main.MainFrame

4.4 Making Extra Files (same level as src)

Make run_project.bat-

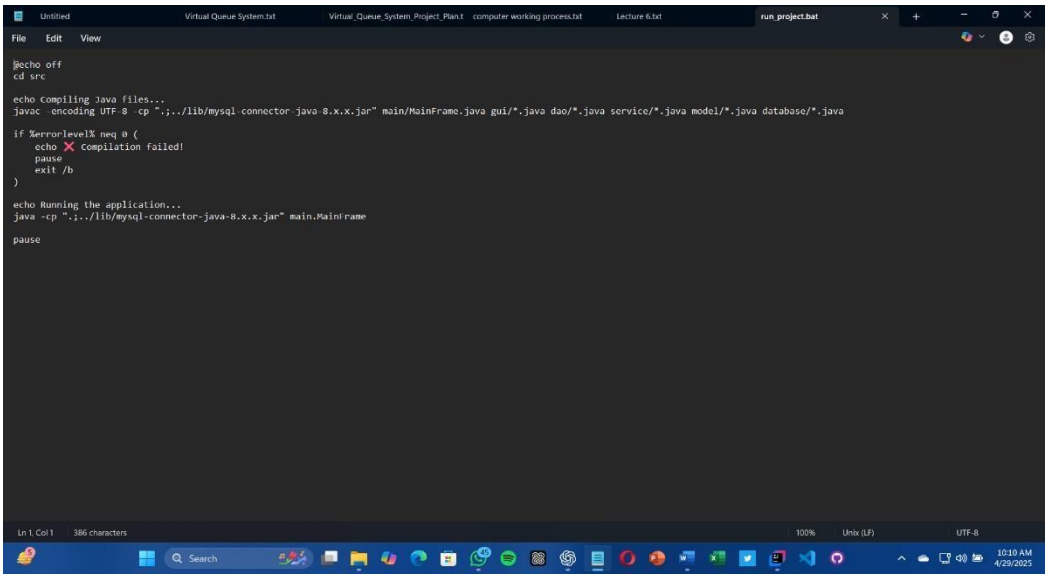

Fig4.4.3 : Make .bat file of named run_project to run in single time via double clicking on this file in file manager

4.4.1Make Readme and EventManagement.docx-

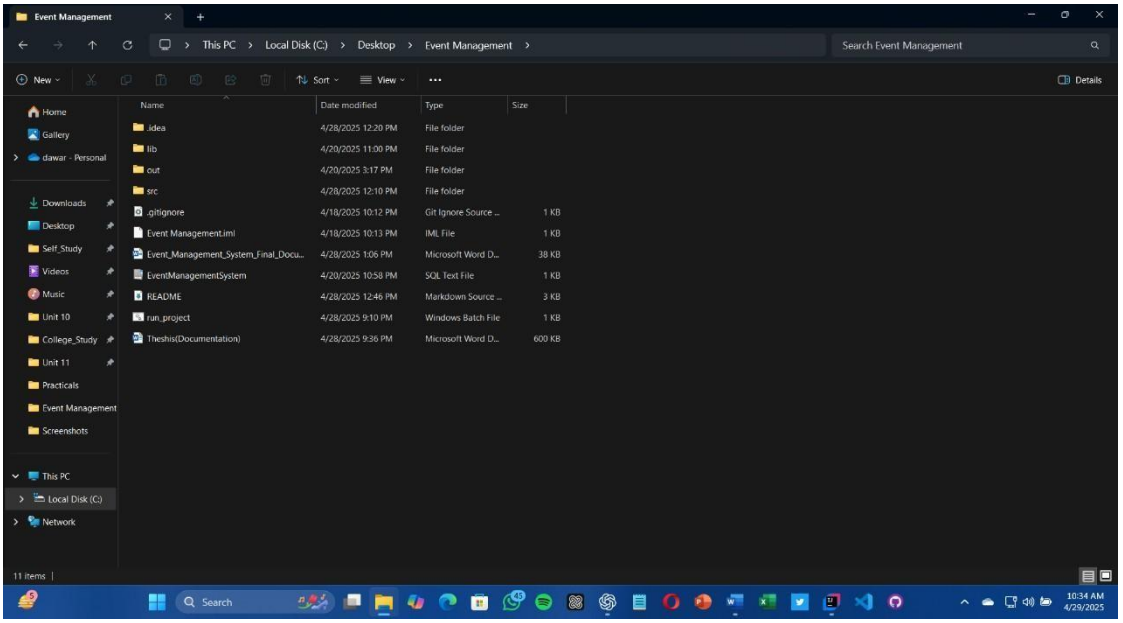

Fig4.4.1.4 : Ensure all files and folders are placed inside root directory name Event Management System

4.5 Testing The Appication

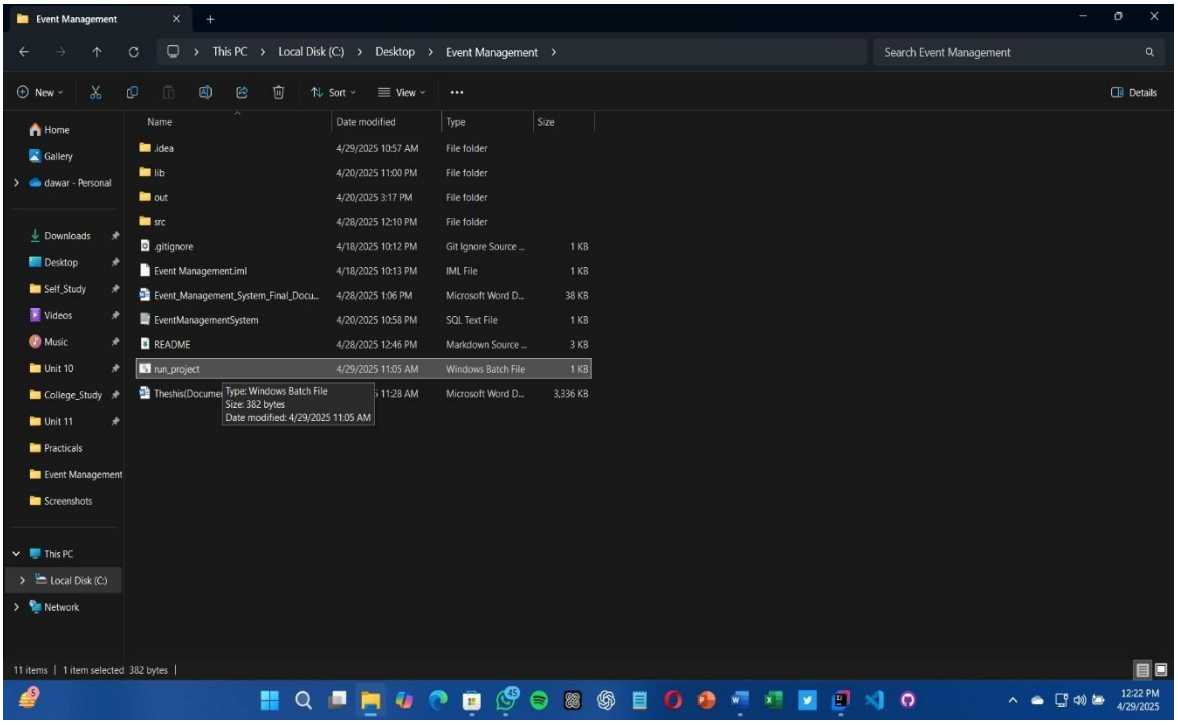

Fig4.5.5: Double Click the run_project.bat file to open the java application

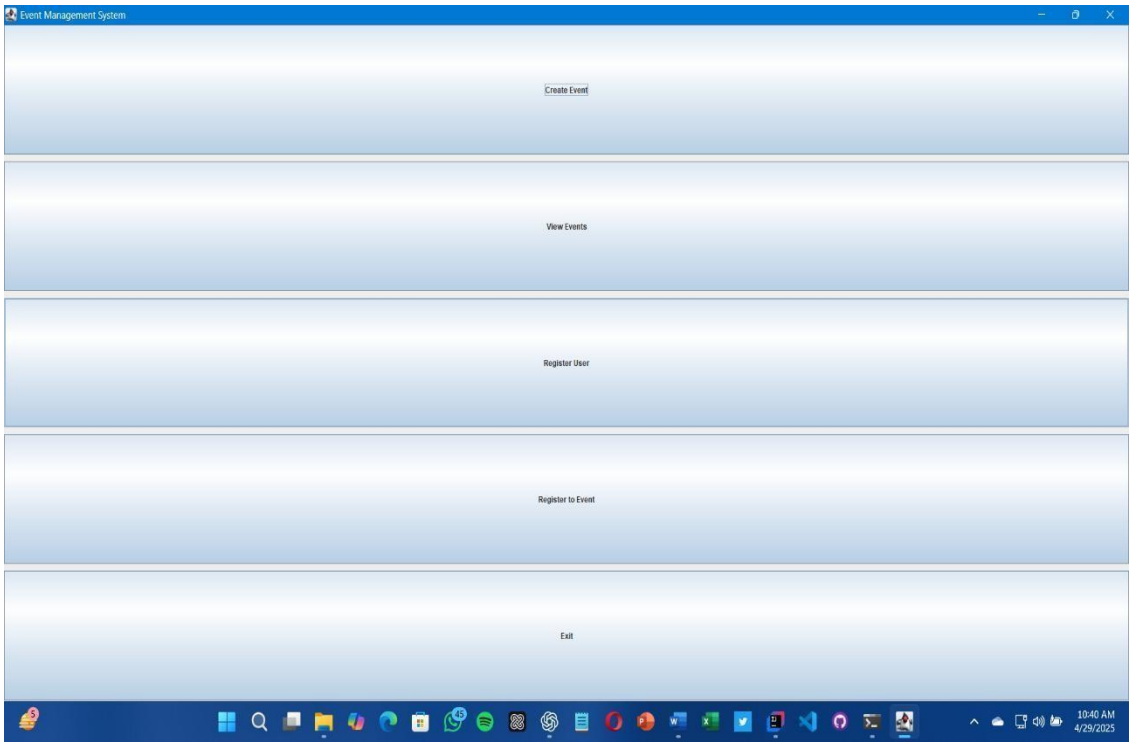

Fig4.5.6 : User Home View of application

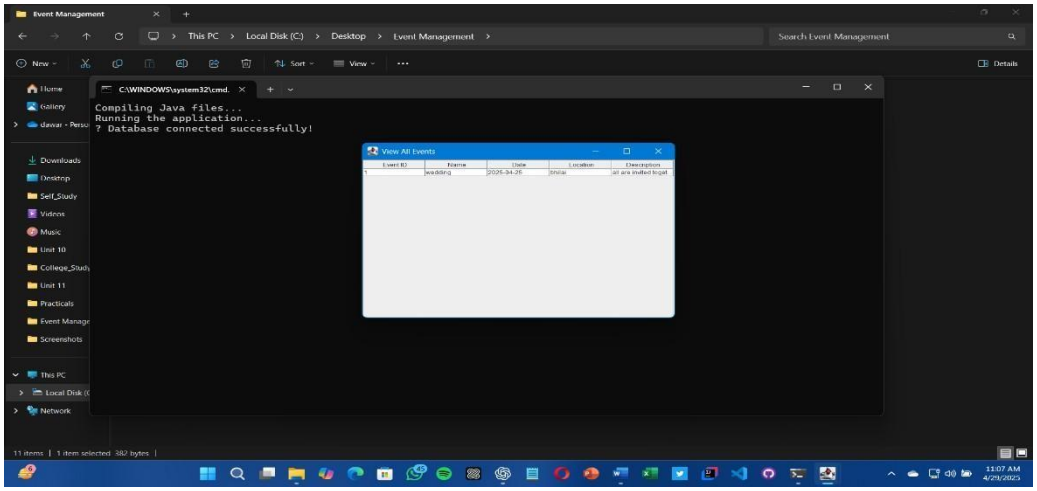

Fig4.5.7 : Viewing available events and capturing Event id by any User

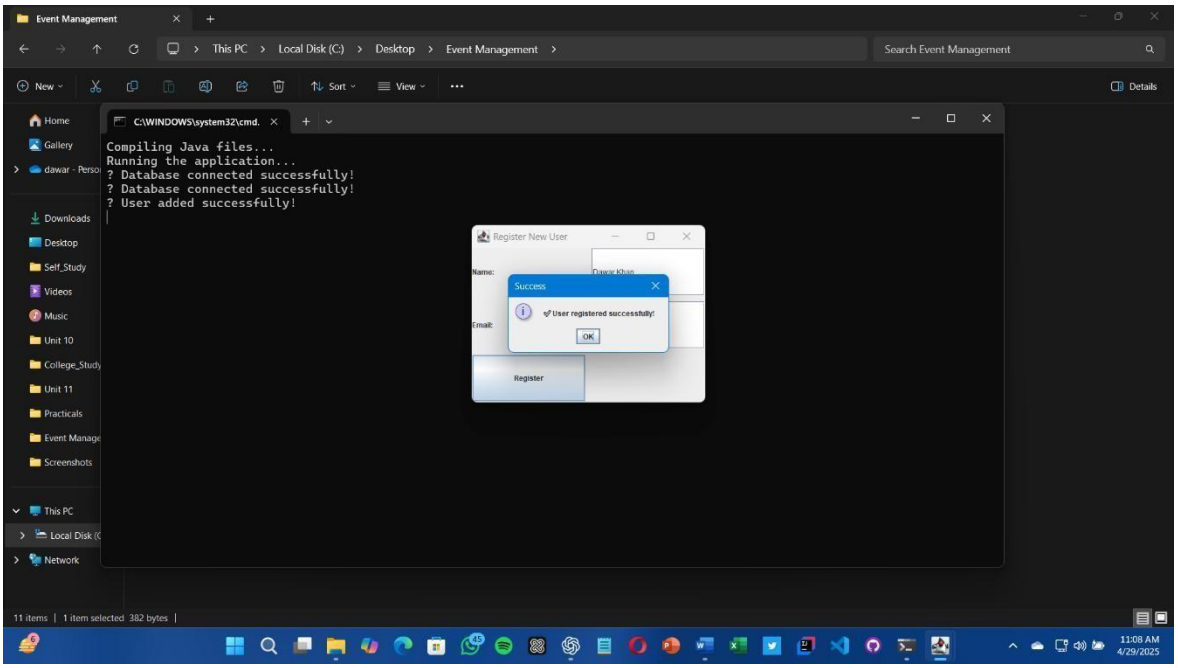

Fig 4.5.8: Registering in the application to join Event

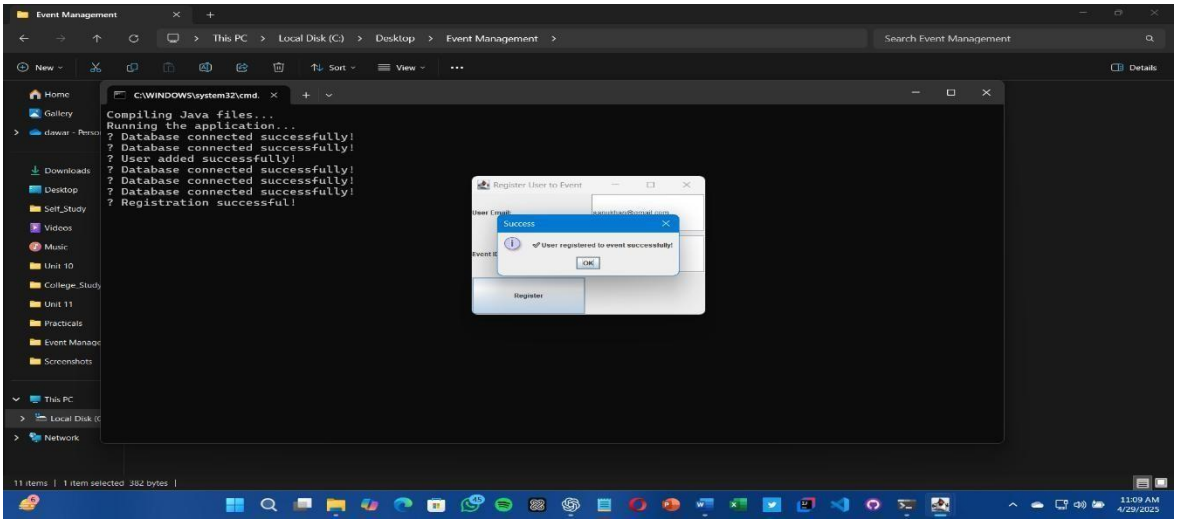

Fig 4.5.9: Registering in the event successfully

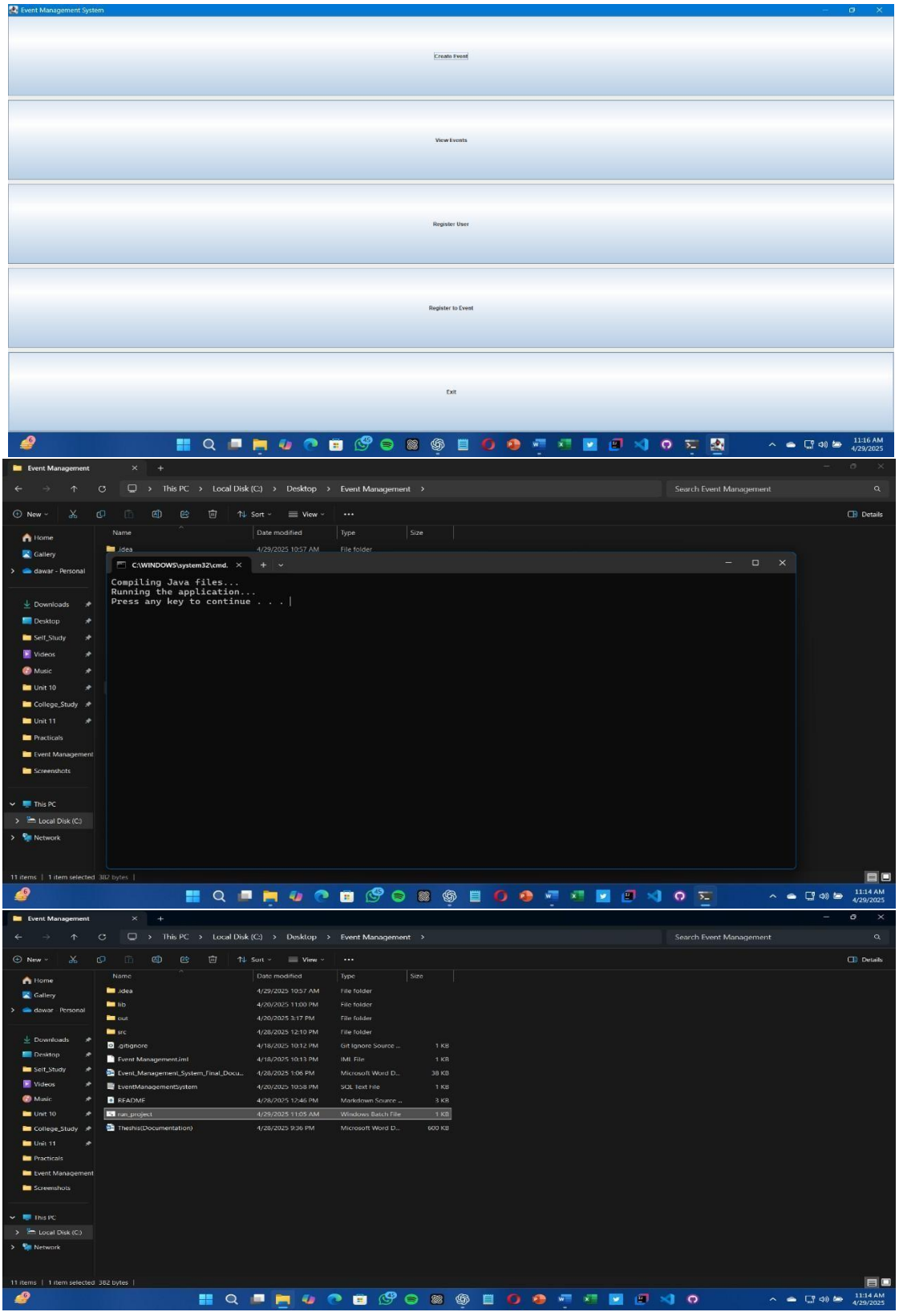

Fig4.6: Quitting the Application on clicking Exit Button and come back to our root directory

CHAPTER 5

CONCLUSION

5.1 CONCLUSION

The **Event Management System** developed as part of this project effectively demonstrates how modern software technologies can be applied to automate and simplify the complex processes associated with organizing events. The system, implemented using Java SE and MySQL, provides a structured and user-friendly interface for managing event creation, participant registration, and event listings. Through modular design and a graphical user interface, the application offers a seamless experience for both administrators and users.

The successful integration of GUI components using Java Swing has allowed the creation of intuitive windows for event operations such as creating new events, registering users, viewing scheduled events, and registering participants to specific events. This approach reduces dependency on paper-based records and enhances accessibility and usability, especially in environments with limited digital infrastructure. The modularity of the codebase, with clear separations between data access, business logic, and presentation layers, further contributes to the system's maintainability and scalability.

One of the key achievements of the project is the effective use of the **MySQL database**, which provides persistent storage of data. This ensures that all event and user information is securely stored, retrievable, and modifiable without the risk of data loss. The system also incorporates basic validation and error-handling mechanisms to alert users about incorrect or missing input, ensuring data consistency and integrity throughout the operations.

The application caters primarily to educational institutions and small-to-medium organizations, offering a foundational tool that can easily be enhanced with advanced features. While the current scope includes core functionalities like event creation, user registration, and event participation, the project lays the groundwork for further development in areas such as role-based access, email notifications, event analytics, and web or mobile interfaces.

5.2 FUTURE SCOPE

- While the current implementation of the **Event Management System** fulfills its primary objective of simplifying event organization and participant management, there is considerable potential for future enhancement and expansion. One major area for improvement is the inclusion of **user authentication** through role-based login systems, distinguishing between administrators and general users. This would enable secure access control and allow administrators to manage backend operations while users can focus on registration and viewing events.
- The existing desktop-based system could be transformed into a **web-based or mobile application**, enabling greater accessibility and wider usage across various platforms and devices. This shift would allow real-time updates, remote access, and integration with third-party services like calendars, emails, and payment gateways.
- Another significant addition would be **automated notifications**, such as email or SMS alerts, to confirm registrations or send event reminders. Incorporating **data analytics** to monitor participant trends, event popularity, and engagement metrics could also add value for organizers in future planning.
- Lastly, a visually enhanced, modern **GUI using frameworks like JavaFX or web technologies** would greatly improve the user experience. These upgrades will not only extend the system's capabilities but also make it more robust, scalable, and applicable to a broader range of event management needs.

REFERENCES

REFERENCES

[1] Sh Java: The Complete Reference by Herbert Schildt.
[2] MySQL Documentation.
[3] Java Swing Tutorials.
[4] JDBC API Documentation.