RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CB23332
SOFTWARE ENGINEERING LAB

Laboratory Record Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

RAJALAKSHMI NAGAR, THANDALAM – 602-105

BONAFIDE CERTIFICATE

NAME:_____ **REGISTER NO.:** _____

ACADEMIC YEAR: 2024-25 **SEMESTER:** III **BRANCH:**_____B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner External Examiner

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND BUSINESS SYSTEMS

REGULATION 2023

CB23332 – SOFTWARE ENGINEERING – LAB MANUAL

S.No.	Date	INDEX	Page No.	Signature
1		Writing the complete problem statement		
2		Writing the software requirement specification document		
3		Entity relationship diagram		
4		Data flow diagrams at level 0 and level 1		
5		Use case diagram		
6		Activity diagram of all use cases.		
7		State chart diagram of all use cases		
8		Sequence diagram of all use cases		
9		Collaboration diagram of all use cases		
10		Assign objects in sequence diagram to classes and make class diagram		
11		Sample Code and screenshots		

EXERCISE NO. 1

AIM: To prepare a **Problem Statement** for an Event Management System in a College Campus.

ALGORITHM:

1. Define the problem clearly, focusing on the issues in the current event management process.
2. Provide a high-level, non-technical description of the problem.
3. Emphasize the need for improvement in the existing system.
4. Highlight the impact of inefficient event management on various stakeholders.
5. Ensure the problem statement addresses **what** needs to be done, not **how** it will be done.

INPUT:

- Issues with the current event management process.
- Broad expectations for the new system from stakeholders (Organizers, Attendees, Admins).
- Initial requirements gathered from key users.
- Overview of the existing system challenges.

OUTPUT:

Problem:

Managing events on the college campus is currently a manual, decentralized process, which leads to inefficiencies in event organization, attendee registration, and communication. Organizers lack a unified platform to manage all event logistics, attendees experience difficulty registering and staying updated on events, and admins face challenges in monitoring event statuses and providing support.

Background:

The college hosts a wide variety of events, such as seminars, cultural programs, technical workshops, and guest lectures. The current event management system is fragmented, requiring organizers to handle different aspects—like registration, communication, and logistics—separately. Attendees often miss important event updates, and admins struggle to oversee and provide necessary services for all events. This creates delays, confusion, and inconsistent event experiences.

The event management process lacks automation and coordination, resulting in poor user experience, inefficient resource allocation, and time delays. Without an integrated system, organizers manually handle registrations, ticketing, and communications, while attendees do not have a centralized platform to track events or receive real-time updates.

Relevance:

Effective event management is crucial for the smooth operation of events on a college campus. A centralized event management system will:

- Enhance the event creation process for organizers.
- Simplify the registration and ticketing process for attendees.

- Provide real-time updates on events.
- Allow admins to oversee multiple events and offer service and support as required. This solution will result in better communication between stakeholders, reduce administrative burdens, and improve the overall user experience.

Objectives:

The primary objective of this project is to develop a comprehensive Event Management System for the college campus that streamlines event planning, registration, and execution processes. Specific objectives include:

1. **For Organizers:**
    - Allow organizers to create events with customizable options, including venue selection, time, date, and event-specific requirements.
    - Provide tools for managing participants, schedules, and resources more efficiently.
    - Implement an intuitive user interface that enables organizers to modify event details on the go.
    - Provide a system to send automatic reminders and notifications to registered attendees.
2. **For Attendees:**
    - Enable attendees to register for events through a centralized platform with real-time updates and notifications.
    - Implement a seamless ticketing system for paid or free events.
    - Offer attendees the ability to view event details, schedules, and updates in real-time through the system.
3. **For Administrators:**
    - Provide administrators with tools to manage multiple events simultaneously.
    - Create a dashboard that offers real-time insights into events, enabling better decision-making and resource allocation.
    - Enable administrators to provide event support services, resolve conflicts, and ensure smooth execution.
    - Allow easy communication between organizers and the administrative team.
4. **General:**
    - Develop a centralized database to store event details, user information, and related data for quick retrieval and analysis.
    - Implement a robust notification system for all stakeholders to keep them informed about changes or updates in events.

Result:

The problem statement has been successfully created, clearly outlining the existing challenges in event management on a college campus and proposing a centralized solution to enhance efficiency and communication among all stakeholders.

EXERCISE NO. 2

AIM: To do requirement analysis and develop Software Requirement Specification Sheet (SRS) for an Event Management System for a College Campus.

1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to develop an **Event Management System** for college campuses, aimed at automating and simplifying the process of planning, organizing, and managing events. The system will provide functionalities for event creation, participant registration, ticketing, notifications, and logistics management. The main stakeholders include event organizers, attendees, and administrators.

1.2 DOCUMENT CONVENTIONS

This document uses the following abbreviations:

- **DB**: Database
- **GUI**: Graphical User Interface
- **API**: Application Programming Interface

1.3 INTENDED AUDIENCE AND READING SUGGESTIONS

This SRS is intended for the following audience:

- **Event Organizers** who are responsible for creating and managing events.
- **Attendees** who register and participate in events.
- **Administrators** who oversee the overall event operations, including logistics and technical support.

1.4 PROJECT SCOPE

The **Event Management System** will streamline the process of organizing events, from creating an event to managing attendee registrations. The system will allow organizers to post event details, sell tickets, and provide real-time updates to attendees. For administrators, the system will offer tools to manage multiple events, monitor resource allocation, and handle logistical needs. The platform will include features like user authentication, notification systems, and reporting tools.

1.5 REFERENCES

- “Software Engineering” by Ian Sommerville
- College Event Management guidelines
- Existing event management solutions (e.g., Eventbrite, Meetup)

---

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

The **Event Management System** will operate as a web-based platform accessible by both desktop and mobile devices. It will integrate with the college's existing student and staff databases to streamline registration and notifications.

- **Event Details**: Include information such as event name, date, location, ticket availability, description, and category (e.g., academic, cultural, sports).
- **User Registration**: Users can register for events and purchase tickets online.
- **Notification System**: Users will receive reminders and updates via email or SMS.

2.2 PRODUCT FEATURES

Key features include:

- **Event Creation and Management**: Organizers can create events, set ticket limits, and define event categories.
- **User Registration**: Attendees can view upcoming events, register, and purchase tickets.
- **Admin Dashboard**: Provides tools for monitoring events, sending notifications, and handling event logistics.
- **Notification System**: Sends automatic alerts to attendees and organizers about event updates or changes.
- **Reports**: Generates summaries for post-event analysis (e.g., attendance, feedback).

2.3 USER CLASSES AND CHARACTERISTICS

- **Organizers**: Faculty or students responsible for managing events. They will have access to create, update, and manage events.
- **Attendees**: Students or external participants who can register, view, and attend events.
- **Administrators**: Users with elevated privileges to manage event logistics, solve issues, and ensure the platform's smooth functioning.

2.4 OPERATING ENVIRONMENT

The system will operate on:

- **Server**: College cloud-based hosting or campus data center.
- **Client**: Any modern browser (Chrome, Firefox, etc.) on Windows, MacOS, iOS, or Android.
- **Database**: SQL or NoSQL database to store event, user, and registration data.

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

- **Compliance**: The system must comply with data privacy and security policies.
- **Language**: The front-end will use HTML, CSS, and JavaScript, while the back-end will use Python or PHP.
- **Database Integrity**: Must ensure the reliability of user and event data, using constraints to avoid conflicts.

2.6 ASSUMPTIONS AND DEPENDENCIES

- **Internet Access**: The platform assumes stable internet access for event registration and management.
- **User Accounts**: Attendees and organizers will require valid college credentials to access the platform.

---

3. SYSTEM FEATURES

3.1 FUNCTIONALITY

The core functions of the Event Management System include:

- **Create Event**: Organizers can specify event details (title, description, venue, etc.).
- **User Registration**: Attendees can register for events and receive confirmation.
- **Ticket Management**: For ticketed events, attendees can view available slots and make purchases.
- **Notification**: Automatic emails or SMS messages will notify users about event status and changes.
- **Admin Features**: Administrators can monitor ongoing events, handle logistics, and access reporting tools.

3.2 EXTERNAL INTERFACE REQUIREMENTS

3.2.1 USER INTERFACES

- **Web Interface**: The system's primary interface will be a user-friendly web application accessible by browsers.
- **Mobile Interface**: The system will also offer a mobile-optimized version.

3.2.2 HARDWARE INTERFACES

- **Client Devices**: Desktops, laptops, tablets, and smartphones.
- **Server**: The server hosting the platform must have sufficient storage and processing capacity to handle concurrent users.

3.2.3 SOFTWARE INTERFACES

- **Operating Systems**: Windows, macOS, Linux, iOS, Android.
- **Database**: SQL (MySQL or PostgreSQL) for storing event and user data.
- **Web Technologies**: HTML5, CSS3, JavaScript for the front end, and Python or PHP for the back end.

3.2.4 COMMUNICATION INTERFACES

- **Internet Protocols**: HTTPS for secure data transmission.

---

4. NON-FUNCTIONAL REQUIREMENTS

4.1 PERFORMANCE REQUIREMENTS

- **Response Time**: The system should provide a response to user queries within 2 seconds.
- **Availability**: 99.9% uptime is required to ensure users can access the system at all times.

4.2 SAFETY REQUIREMENTS

- **Data Backup**: Regular backups should ensure the integrity of the event data.
- **Error Handling**: Any system errors must be reported and handled without affecting ongoing event operations.

4.3 SECURITY REQUIREMENTS

- **User Authentication**: Secure login for organizers, attendees, and administrators.
- **Data Encryption**: All sensitive data (e.g., user credentials, payment details) must be encrypted using industry-standard algorithms.

4.4 SOFTWARE QUALITY ATTRIBUTES

- **Usability**: The system should be easy to use for all user classes, including those unfamiliar with technology.
- **Maintainability**: The platform should support updates for new features or bug fixes.
- **Scalability**: The system must handle increasing numbers of events and users as the college population grows.

RESULT: The Software Requirement Specification (SRS) for the Event Management System for a college campus has been successfully developed, addressing all required features and constraints.

EXERCISE NO. 3

AIM:

To draw the Entity Relationship Diagram for the Event Management System.

ALGORITHM:

1. **Mapping of Regular Entity Types**
    - Identify the main entities involved in the system:
        - Organizers
        - Attendees
        - Admins
        - Events
2. **Mapping of Weak Entity Types**
    - Determine if there are any weak entities (e.g., Event Details may not exist independently without Events). In this case, there are no weak entities as all identified entities can exist independently.
3. **Mapping of Binary 1:1 Relation Types**
    - Identify relationships where each entity is related to only one instance of another entity.
    - no 1:1 relationships are defined in this case.
4. **Mapping of Binary 1: N Relationship Types**
    - Define relationships where one entity can relate to multiple instances of another entity:
        - Organizers create and manage multiple Events (1:N).
        - One Admin can manage multiple Events (1:N)
5. **Mapping of Binary M : N Relationship Types**
    - Define relationships where many instances of one entity can relate to many instances of another entity:
        - Attendees can register for multiple Events, and each Event can have multiple Attendees (M:N).

INPUT:

- **Entities:**
    - **Organizers**
    - **Attendees**
    - **Admins**

    - **Events**
- **Entity Relationship Matrix:**
    - **Organizers ↔ Events**: 1

(One Organizer can create many Events)

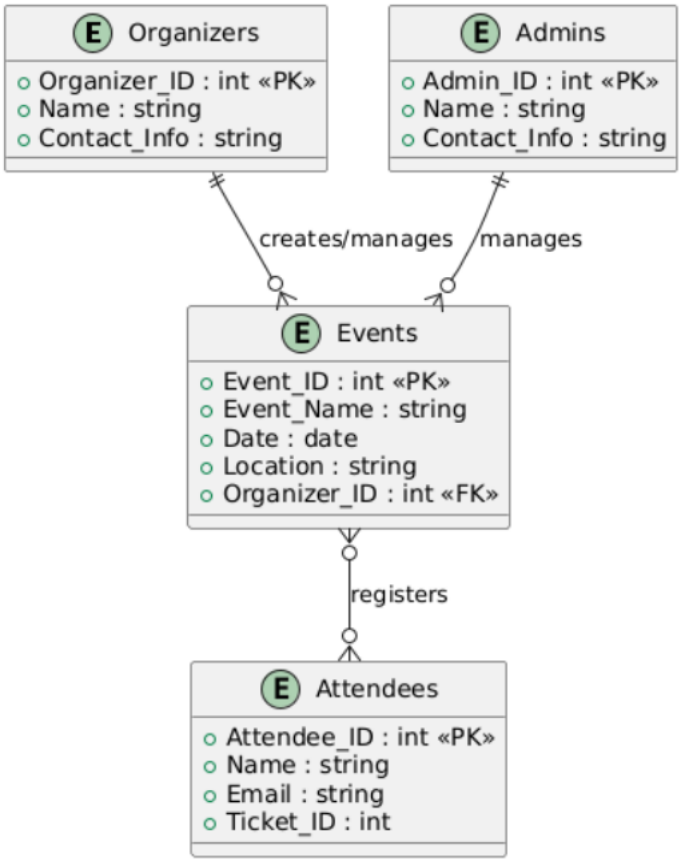
    - **Events ↔ Attendees**: M

(Many Attendees can register for many Events)

- **Primary Keys:**
    - Organizer_ID (for Organizers)
    - Attendee_ID (for Attendees)
    - Admin_ID (for Admins)
    - Event_ID (for Events)
- **Attributes:**
    - **Organizers**:
        - Organizer_ID
        - Name
        - Contact_Info
    - **Attendees**:
        - Attendee_ID
        - Name
        - Email
        - Ticket_ID
    - **Admins**:
        - Admin_ID
        - Name
        - Contact_Info
    - **Events**:
        - Event_ID
        - Event_Name
        - Date
        - Location
        - Organizer_ID (FK)
- **Mapping of Attributes with Entities:**

- **Organizers** → [Organizer_ID, Name, Contact_Info]
- **Attendees** → [Attendee_ID, Name, Email, Ticket_ID]
- **Admins** → [Admin_ID, Name, Contact_Info]
- **Events** → [Event_ID, Event_Name, Date, Location, Organizer_ID]

OUTPUT:



RESULT:

The entity relationship diagram was made successfully by following the steps described above, effectively illustrating the relationships and attributes relevant to the Event Management System.

EXERCISE NO. 4

AIM:

To draw the Data Flow Diagram for the Event Management System and list the application modules.

ALGORITHM:

1. **Open Tool:** Use Visual Paradigm or Lucidchart.
2. **Select Template:** Choose a DFD template.
3. **Name DFD:** Title it appropriately (e.g., "Event Management System DFD").
4. **Add External Entities:** Include:
    - Organizers
    - Attendees
    - Admins
5. **Add Processes:** Include:
    - Create Event
    - Register for Event
    - Manage Events
    - View Event Details
6. **Add Data Stores:** Include:
    - Event Database
    - Attendee Database
    - Organizer Database
7. **Add Data Flow:** Connect entities and processes with labeled arrows.
8. **Customize DFD:** Use colors and fonts for clarity.
9. **Add Title and Share:** Include a title and share your DFD.

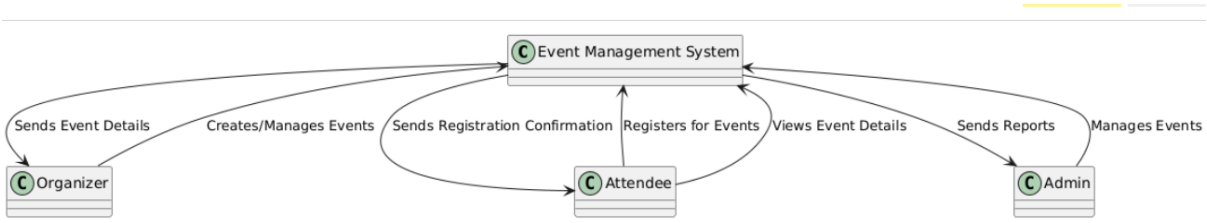
INPUT:

- **Processes:** Create Event, Register for Event, Manage Events, View Event Details
- **Datastores:** Event Database, Attendee Database, Organizer Database
- **External Entities:** Organizers, Attendees, Admins

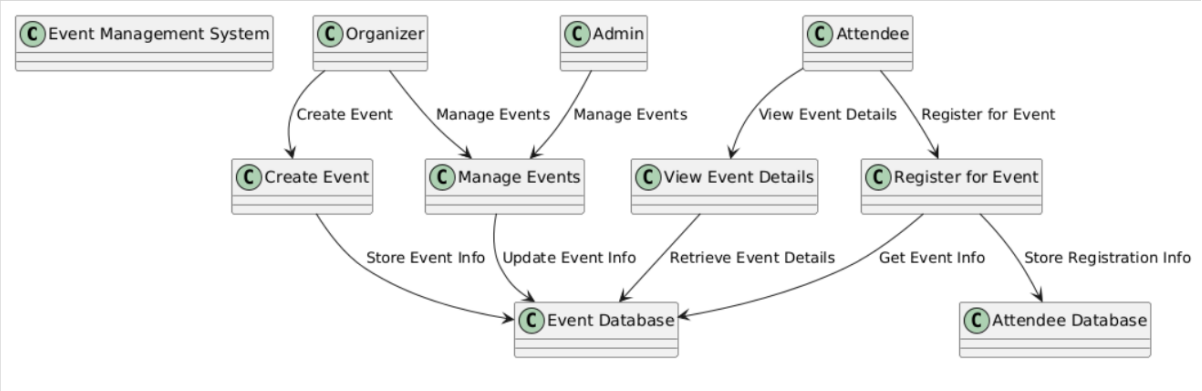
MODULES IN THE APPLICATION:

1. **Event Management Module**
2. **Registration Module**
3. **Admin Module**

4. **Notification Module**
5. **Reporting Module**



OUTPUT :



Result: The Data Flow diagram was made successfully by following the steps described above.

EXERCISE NO. 5

AIM:

To Draw the Use Case Diagram for the **Event Management System**.

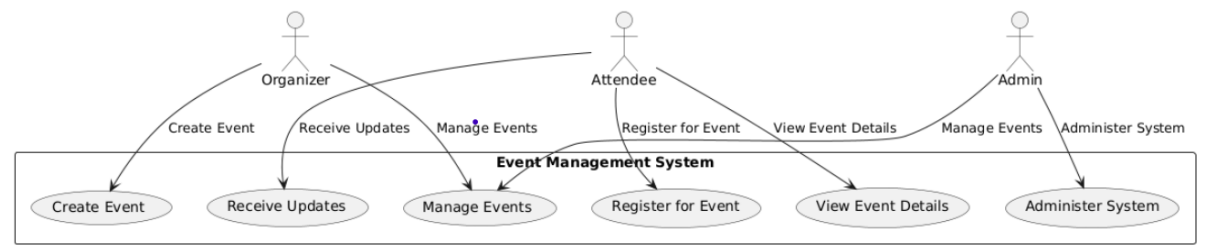
ALGORITHM:

1. **Identify Actors**:
    - Identify the primary users interacting with the system .
2. **Identify Use Cases**:
    - List the main tasks the actors perform within the system (e.g., Create Event, Register for Event).
3. **Connect Actors and Use Cases**:
    - Draw lines connecting each actor to the use cases they interact with.
4. **Add System Boundary**:
    - Encapsulate the use cases within a boundary that represents the event management system.
5. **Define Relationships**:
    - Specify include or extend relationships between use cases if applicable.
6. **Review and Refine**:
    - Ensure that all important tasks and interactions are captured accurately.
7. **Validate**:
    - Validate the diagram by checking it against the system's requirements

INPUTS:

- **Actors**: Organizer, Attendee, Admin
- **Use Cases**: Create Event, Manage Events, Register for Event, View Event Details, Receive Updates, Administer System
- **Relations**: Direct connections between actors and use cases

OUTPUT:



Result: The use case diagram has been created successfully by following the steps given.

EXERCISE NO. 6

AIM:

To Draw the Activity Diagram for the Event Management System.

ALGORITHM:

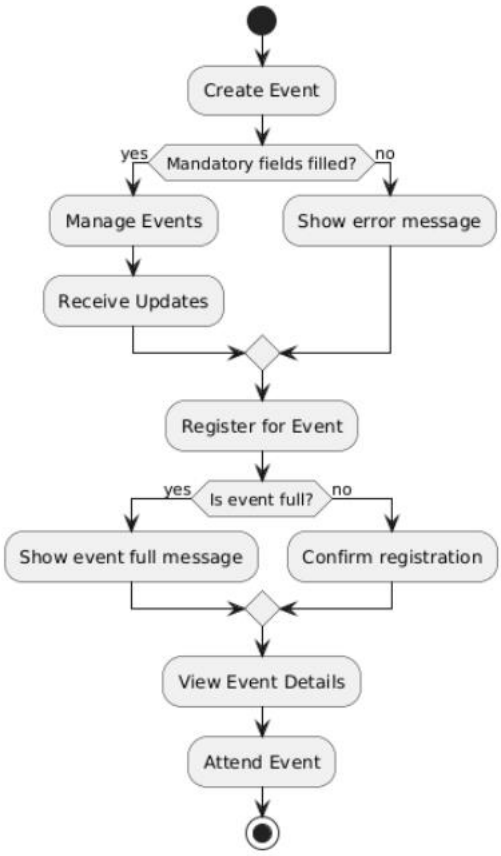
1. **Identify Initial State and Final States**:
    - Define the starting point (initial state) of the activity and the end point (final state) where the process concludes.
2. **Identify Intermediate Activities Needed**:
    - List the main activities that occur between the initial and final states (e.g., creating an event, registering for an event, etc.).
3. **Identify Conditions or Constraints**:
    - Determine any conditions or decision points that affect the flow of activities (e.g., is the event full?).
4. **Draw the Diagram with Appropriate Notations**:
    - Use standard notations like ovals for initial and final states, rectangles for activities, diamonds for decision points, and arrows to show the flow of control.

INPUTS:

- **Activities**:
    - Create Event
    - Manage Events
    - Register for Event
    - View Event Details
    - Receive Updates
    - Administer System
- **Decision Points**:
    - Is the event full?
    - Is the attendee registered?
- **Guards**:
    - If the event is not full, allow registration.
    - If the attendee is registered, allow access to event details.
- **Parallel Activities**:
    - Organizers can manage multiple events while attendees register for events.
- **Conditions**:

- Ensure all mandatory fields are filled before an event is created.
- Check for any scheduling conflicts when creating an event.

OUTPUT:



RESULT : The Activity diagram has been created successfully by following the steps given.

EXERCISE NO. 7

AIM:

To Draw the State Chart Diagram for the Event Management System.

ALGORITHM:

1. **Identify Important Objects to be Analyzed**:
    - Determine the key objects in the Event Management System (e.g., Event, Organizer, Attendee).
2. **Identify the States**:
    - List the possible states of each object throughout its lifecycle. For example:
        - **Event**: Planned, Ongoing, Completed, Canceled.
        - **Organizer**: Active, Inactive, Managing Events.
        - **Attendee**: Registered, Attending, Completed Attendance.
3. **Identify the Events**:
    - Define the events that trigger state changes (e.g., Event Created, Event Started, Event Completed, Event Canceled).

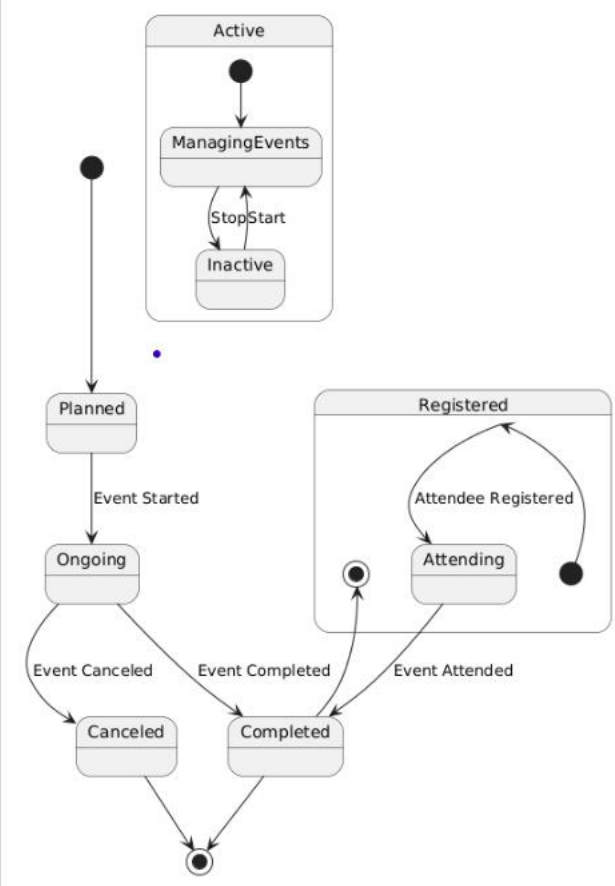
INPUTS:

- **Objects**:
    - Event
    - Organizer
    - Attendee
- **States**:
    - **Event**:
        - Planned
        - Ongoing
        - Completed
        - Canceled
    - **Organizer**:
        - Active
        - Inactive
        - Managing Events
    - **Attendee**:
        - Registered
        - Attending

- Completed Attendance

- **Events**:
  - Event Created
  - Event Started
  - Event Completed
  - Event Canceled
  - Attendee Registered
  - Attendee Attended

OUTPUT :



Result: The State Chart diagram has been created successfully by following the steps given.

EXERCISE NO. 8

AIM: To Draw the Sequence Diagram for the Event Management System.

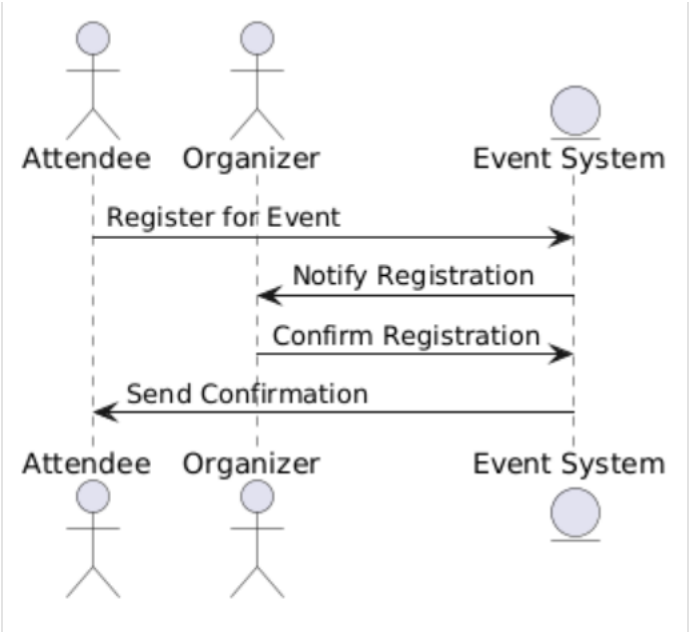
ALGORITHM:

1. Identify the Scenario (e.g., Attendee registering for an event).
2. List the Participants (Organizer, Attendee, Event System).
3. Define Lifelines for each participant.
4. Arrange Lifelines vertically.
5. Add Activation Bars to represent active periods.
6. Draw Messages between participants (e.g., registration requests).
7. Include Return Messages after processing.
8. Indicate Timing and Order of events.
9. Include Conditions and Loops (if applicable).
10. Consider Parallel Execution (if applicable).
11. Review and Refine the diagram for clarity.
12. Add Annotations and Comments for better understanding.
13. Document Assumptions and Constraints.
14. Use a tool (like PlantUML or Lucidchart) to create a neat sequence diagram.

INPUTS:

- Objects participating in the interaction:
    - Organizer
    - Attendee
    - Event System
- Message flows among the objects:
    - Registration request
    - Confirmation response
- The sequence of messages:
    - Attendee -> Event System: Register for Event
    - Event System -> Organizer: Notify Registration
    - Organizer -> Event System: Confirm Registration
    - Event System -> Attendee: Send Confirmation

Output :



Result: The Sequence diagram has been created successfully by following the steps given.

EXERCISE NO. 9

AIM: To Draw the Collaboration Diagram for the Event Management System.

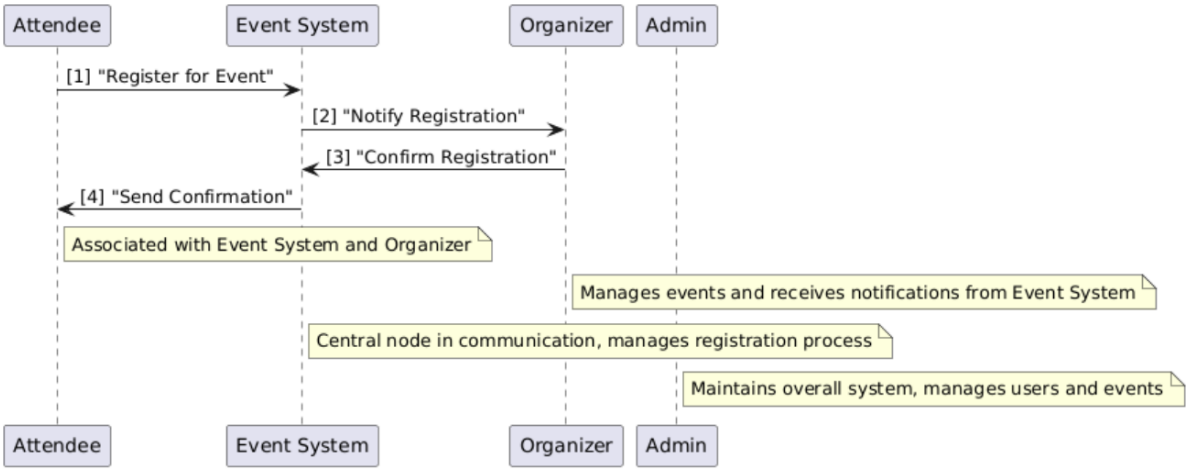
ALGORITHM:

1. **Identify Objects/Participants**:
    - List the key participants (e.g., Organizer, Attendee, Event System, Admin).
2. **Define Interactions**:
    - Determine how these participants interact with each other.
3. **Add Messages**:
    - Document the messages exchanged between participants, including the order of messages.
4. **Consider Relationships**:
    - Define the relationships (associations) between the objects in the diagram.
5. **Document the Collaboration Diagram**:
    - Create a neat and organized diagram.
    - Include any relevant explanations or annotations to clarify interactions.

INPUTS:

- **Objects**:
    - Organizer
    - Attendee
    - Event System
    - Admin
- **Message Flows**:
    - The specific messages sent between participants (e.g., "Register for Event," "Confirm Registration").
- **Sequence of Messages**:
    - The order in which messages are exchanged.
- **Object Organization**:
    - How objects relate to one another.

OUTPUT:



Result: The Collaboration diagram has been created successfully by following the steps given.

EXERCISE NO. 10

AIM: To Draw the Class Diagram for the Event Management System.

ALGORITHM:

1. **Identify Classes**:
    - Determine the main classes required for the system, such as Event, Organizer, Attendee, and Admin.
2. **List Attributes and Methods**:
    - Define the attributes (data members) and methods (functions) for each class.
3. **Identify Relationships**:
    - Determine how classes interact with each other (e.g., associations, inheritance).
4. **Create Class Boxes**:
    - Design individual boxes for each class.
5. **Add Attributes and Methods**:
    - Include the attributes and methods within the respective class boxes.
6. **Draw Relationships**:
    - Connect classes using lines to represent their relationships.
7. **Label Relationships**:
    - Clearly label the type of relationships (e.g., one-to-many, inheritance).
8. **Review and Refine**:
    - Check for any inconsistencies and make necessary adjustments.
9. **Use Tools for Digital Drawing**

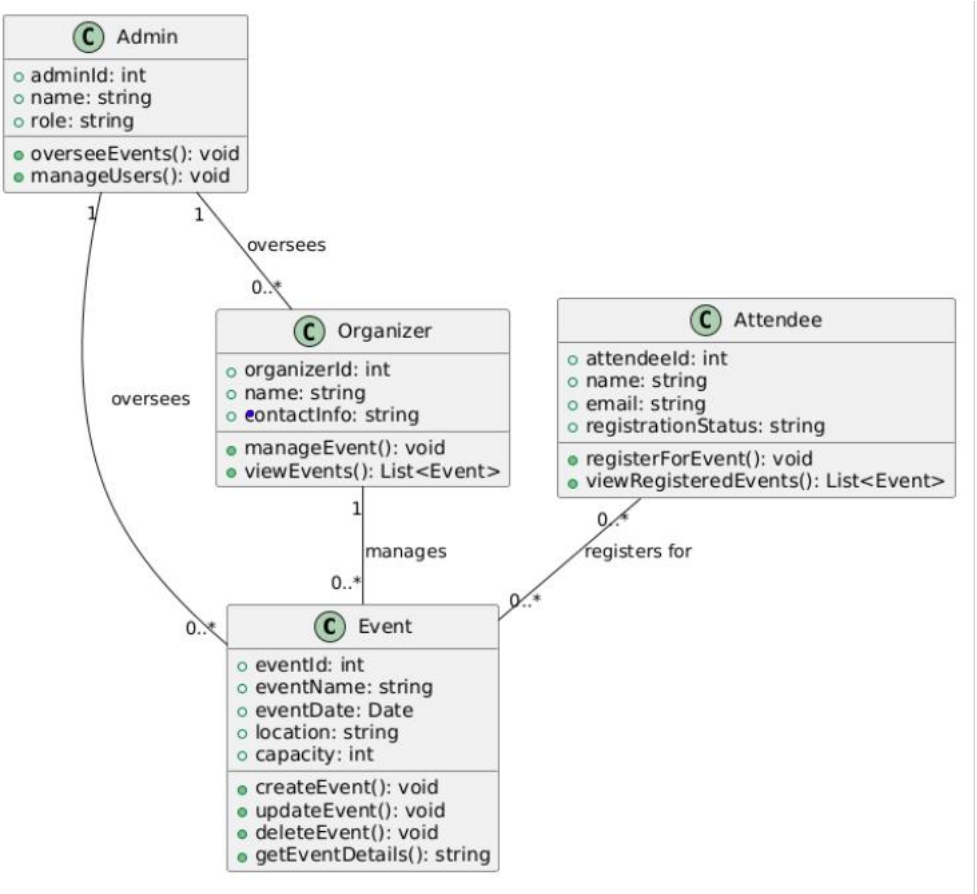
INPUTS:

1. **Class Names**:
    - Event
    - Organizer
    - Attendee
    - Admin
2. **Attributes**:
    - **Event**: eventId, eventName, eventDate, location, capacity
    - **Organizer**: organizerId, name, contactInfo

- **Attendee**: attendeeId, name, email, registrationStatus
- **Admin**: adminId, name, role

3. **Methods**:
   - **Event**: createEvent(), updateEvent(), deleteEvent(), getEventDetails()
   - **Organizer**: manageEvent(), viewEvents()
   - **Attendee**: registerForEvent(), viewRegisteredEvents()
   - **Admin**: overseeEvents(), manageUsers()
4. **Visibility Notation**:
   - + for public
   - - for private
   - # for protected

OUTPUT :



Result: The Class diagram has been created successfully by following the steps given.

SAMPLE CODE :

Here is the sample code representing the deployment of mvp version of campus event management system :

```
import streamlit as st
import pandas as pd
import uuid

# Initialize session states
if 'event_db' not in st.session_state:
    st.session_state.event_db = pd.DataFrame(columns=[
        'Event ID', 'Event Name', 'Description', 'Date', 'Location', 'Status', 'Organizer ID'
    ])
if 'user_db' not in st.session_state:
    st.session_state.user_db = {
        "admin": {"password": "admin123", "role": "admin"},
        "organizer1": {"password": "orgpassword", "role": "organizer"},
        "attendee1": {"password": "attendee123", "role": "attendee"},
    }

# Helper functions
def create_event(event_name, description, date, location, organizer_id):
    event_id = str(uuid.uuid4())[:8]
    new_event = {
        "Event ID": event_id,
        "Event Name": event_name,
        "Description": description,
        "Date": date,
        "Location": location,
        "Status": "Scheduled",
        "Organizer ID": organizer_id
```

```
    }
    st.session_state.event_db = pd.concat([st.session_state.event_db,
pd.DataFrame([new_event])], ignore_index=True)
    st.success(f"Event '{event_name}' created successfully!")

# Login Page for different roles
def login_page():
    st.title("Welcome to Campus Event Management System", anchor="top")
    st.markdown(
        """
        <style>
        .login-title {font-size: 24px; color: #4CAF50; font-weight: bold;}
        .login-form {background-color: #f4f4f4; border-radius: 8px; padding: 20px; box-shadow: 0
4px 8px rgba(0, 0, 0, 0.1);}
        .login-button {background-color: #4CAF50; color: white; padding: 10px 20px; border-
radius: 8px; cursor: pointer;}
        .login-button:hover {background-color: #45a049;}
        </style>
        """,
        unsafe_allow_html=True
    )

    role = st.radio("Select your role", ["Admin", "Organizer", "Attendee"])

    username = st.text_input(f"Enter {role} Username")
    password = st.text_input("Enter Password", type="password")

    login_button = st.button("Login", key="login_button", use_container_width=True)

    if login_button:
        if username in st.session_state.user_db and
st.session_state.user_db[username]["password"] == password:
            role_assigned = st.session_state.user_db[username]["role"]
            if role_assigned == role.lower():
```

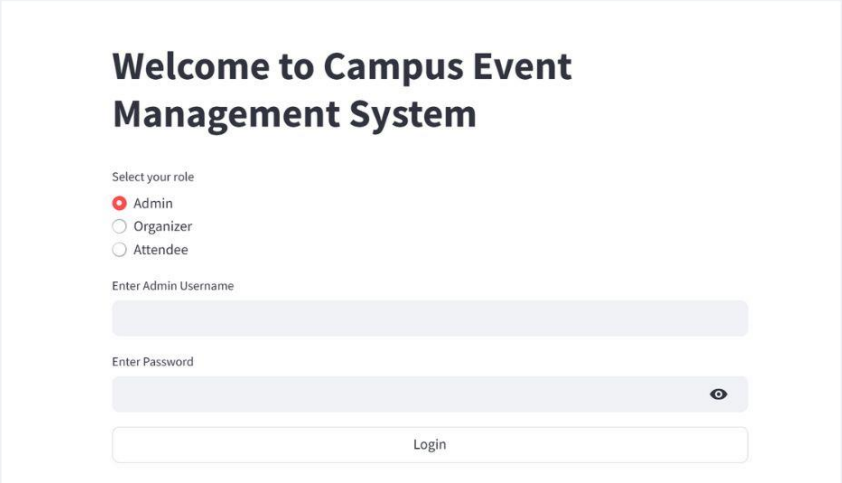
```
                st.session_state.logged_in_user = username
                st.session_state.user_role = role_assigned
                st.success(f"Welcome, {username} ({role_assigned.capitalize()})!")
                return True
            else:
                st.error("Invalid role for the provided username!")
        else:
            st.error("Invalid username or password!")

    return False

# Event Creation Page (for Organizer)
def create_event_page():
    st.title("Create New Event")
    with st.form(key='create_event_form'):
        event_name = st.text_input("Event Name")
        description = st.text_area("Event Description")
        date = st.date_input("Event Date")
        location = st.text_input("Event Location")
        submit_button = st.form_submit_button("Create Event")

        if submit_button:
            if not event_name or not description or not location:
                st.error("Please fill out all fields!")
            else:
                organizer_id = st.session_state.logged_in_user  # Get organizer ID from logged-in user
                create_event(event_name, description, date, location, organizer_id)

# View Events Page (for Attendee and Organizer)
def view_events_page():
    st.title("Upcoming Events")
    if st.session_state.event_db.empty:
```

```
        st.warning("No events scheduled yet.")
    else:
        st.dataframe(st.session_state.event_db)

# Register for Event Page (for Attendee)
def register_for_event_page():
    st.title("Event Registration")
    event_ids = st.session_state.event_db['Event ID'].tolist()
    if not event_ids:
        st.warning("No events available for registration.")
        return

    selected_event_id = st.selectbox("Select Event to Register", event_ids)
    if selected_event_id:
        event_info = st.session_state.event_db[st.session_state.event_db['Event ID'] ==
selected_event_id].iloc[0]
        st.write(f"**Event Name:** {event_info['Event Name']}")
        st.write(f"**Description:** {event_info['Description']}")
        st.write(f"**Date:** {event_info['Date']}")
        st.write(f"**Location:** {event_info['Location']}")

        with st.form(key='register_form'):
            name = st.text_input("Your Name")
            email = st.text_input("Your Email")
            register_button = st.form_submit_button("Register")

            if register_button:
                if not name or not email:
                    st.error("Please provide your name and email.")
                else:
                    st.success(f"Successfully registered for '{event_info['Event Name']}'!")
def admin_panel():
    st.title("Admin Panel")
```

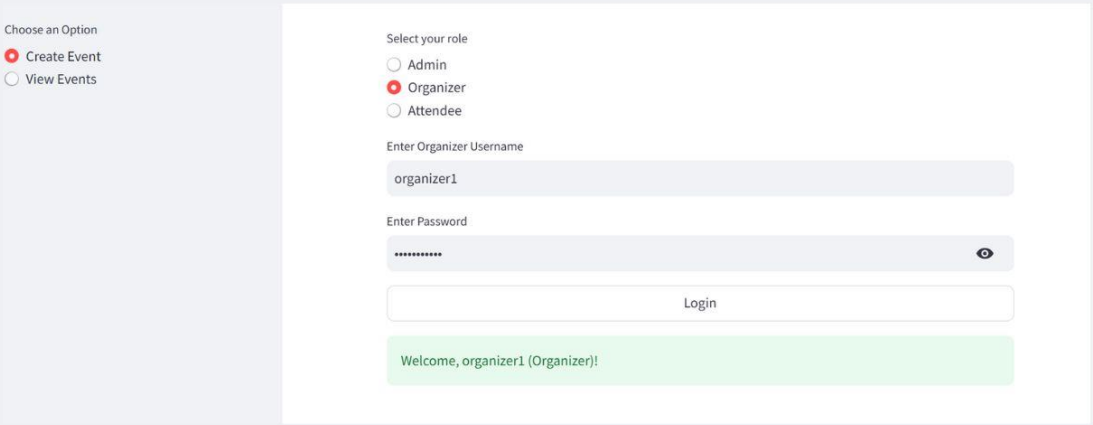
```
    st.write("Welcome to the Admin Panel!")
    st.dataframe(st.session_state.event_db)

# Main Streamlit Application
def main():
    if 'logged_in_user' not in st.session_state:
        if not login_page():
            return
    if st.session_state.user_role == "admin":
        admin_panel()
    elif st.session_state.user_role == "organizer":
        user_choice = st.sidebar.radio("Choose an Option", ["Create Event", "View Events"])
        if user_choice == "Create Event":
            create_event_page()
        elif user_choice == "View Events":
            view_events_page()
    elif st.session_state.user_role == "attendee":
        user_choice = st.sidebar.radio("Choose an Option", ["View Events", "Register for
Event"])
        if user_choice == "View Events":
            view_events_page()
        elif user_choice == "Register for Event":
            register_for_event_page()

if __name__ == "__main__":
    main()
```

OUTPUT :

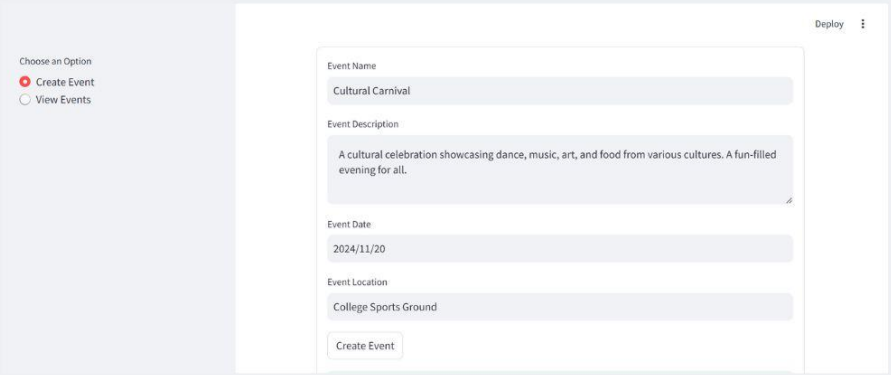
Landing page



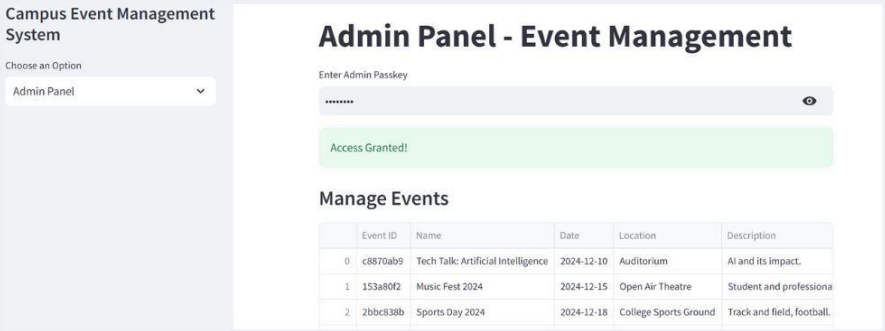
Login page



Organizers create and manage events



Administrators page



Attendees page -To browse and register up for events