EVENT MANAGEMENT SYSTEM

OBJECT ORIENTED PARADIGM

DECEMBER 31, 2024

GROUP: "WITH OOP"

INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD

DEPARTMENT OF COMPUTER SCIENCE AND IT

PROJECT : EVENT MANAGEMENT SYSTEM

COURSE :OBJECT ORIENTED PARADIGM

COURSE CODE :CS 211

PROFESSOR :DR MUHAMMAD NADEEM

SUBMITTED BY:

• **MUHAMMAD IRFAN**	**4953-FOC/BSCS/F23**	**SECT: A**
• **TALAL GUL**	**4956-FOC/BSCS/F23**	**SECT: A**
• **HALAR KHAN**	**4957-FOC/BSCS/F23**	**SECT: A**
• **WAHAB EJAZ**	**5033-FOC/BSCS/F23**	**SECT: C**

INDEX

SOFTWARE REQUIREMENT SPECIFICATION(SRS)

TABLE OF CONTENT

CHAPTER 1

1. Introduction:

1.1 Purpose:

The purpose of this document is to provide a detailed Software Requirements Specification (SRS) for the Event Management System (EMS). This system is designed to facilitate the creation, registration, and management of events by both admin and customers. The primary objective of this software is to provide an intuitive platform for admins to manage event details and registrations, while offering customers an easy way to browse and register for events.

The EMS system will encompass the following features:

- **Event Management**: Admins can create ,edit, and delete events, as well as view events.
- **Customer Registration** :Customers can view events ,register for events, and check their registration status.

1.2 Document Intended For:

This document is intended for the following audiences:

- **Designers**: The design team will use this document to create wireframes, system architecture, and user interface designs.
- **Developers**: The development team will refer to this document for coding guidelines, system functionality, and file handling requirements.
- **Testers**: The testing team will use this document to create test cases, ensuring that the system meets specified requirements and adheres to testing standards.
- **Stakeholders**: Business managers and stakeholders will use this document to ensure that the system fulfills business requirements, aligns with goals, and meets key objectives.

1.3 Project Scope:

The Event Management System (EMS) will enable:

- **Admin Functions**:
    - Create, view, update, and delete events.
    - View and manage customer registrations.

Customer Functions:

- Browse available events, and view detailed event information.
- Register for events, provide personal details, and select tickets or registration types.
- View and update registration status.

1.4 Tools and Technologies Used:

- **Programming Language**: C++ is chosen for the core backend development due to its efficiency and performance.
- **Development IDE**: Visual Studio Code and Microsoft Visual Studio are the chosen development environments to support C++ development.
- **Data Storage**: The system will use file handling techniques, such as text files or binary files, to store event details, customer registrations, and other system data. This allows for data retrieval without requiring a traditional database.
- **Operating System Compatibility**: The system is expected to be compatible with Windows, Linux, and MacOS.

1.5 Assumptions:

- All team members possess the necessary skills in C++ programming, file handling, and software development.
- The system will run on computers with at least 2GB RAM, 1GB free disk space, and the ability to run command-line applications.
- File handling will be efficient and optimized to handle large volumes of data with minimal latency.

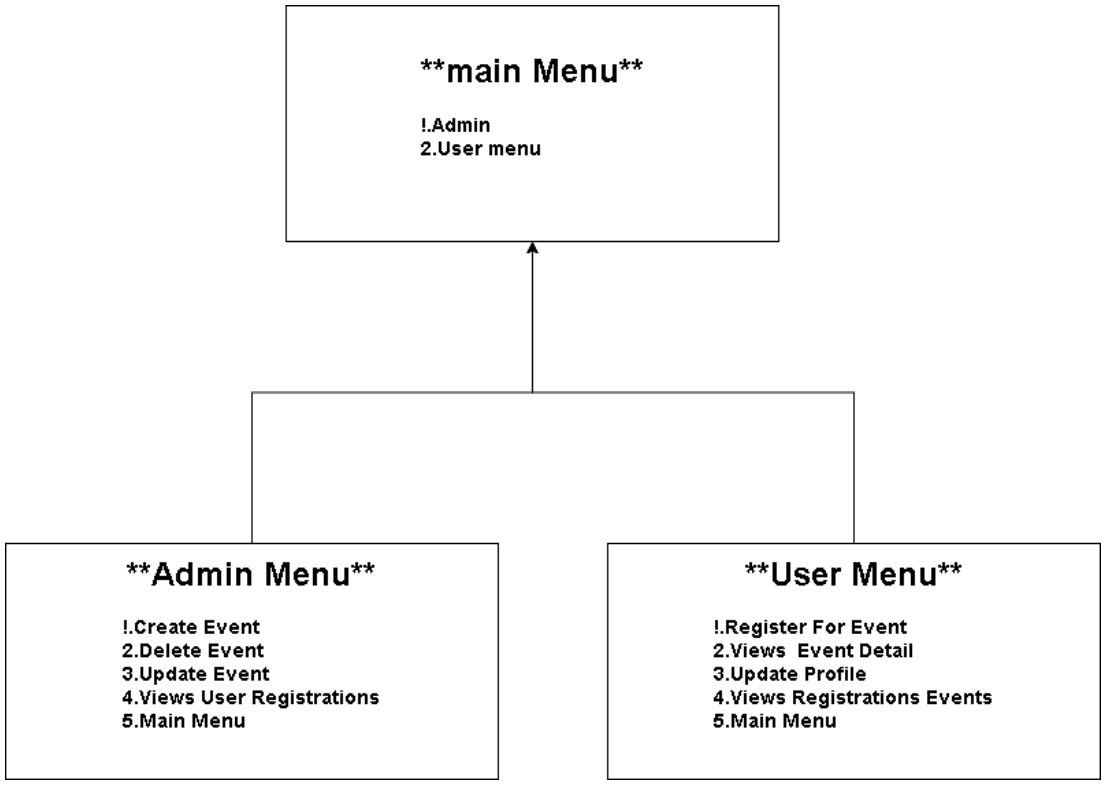
CHAPTER 2

2. Customers:

2.1 Targeted Customers:

- **Organizations**: The Event Management System (EMS) is primarily intended for organizations that need an efficient platform for managing events, such as event organizers, businesses, universities, and companies. These organizations will use the system to create, manage, and promote events, and they will benefit from features like event categorization and data analytics.
- **Individuals**: Customers who wish to register for events, track their registration, and manage their participation will also be key stakeholders. This category includes general attendees, event participants, and potential customers looking for entertainment, cultural, or professional events.

2.2 User Demographics:



- **Admin Users**:
    - Event managers, coordinators, and event organizers within organizations, including businesses, universities, and government bodies.
    - Targeted users are typically business owners, marketing managers, or event managers.
    - These users will seek to simplify event management tasks, track registrations, and generate reports on event participation.
- **Customer Users**:
    - General public and participants seeking to register for events, track their registrations, and receive updates on upcoming events.
    - These users are likely students, professionals, families, or individuals seeking leisure and entertainment activities.

CHAPTER 3

3. Functionality:

3.1 Overview of System Functionality:

The Event Management System will provide the following key functionalities:

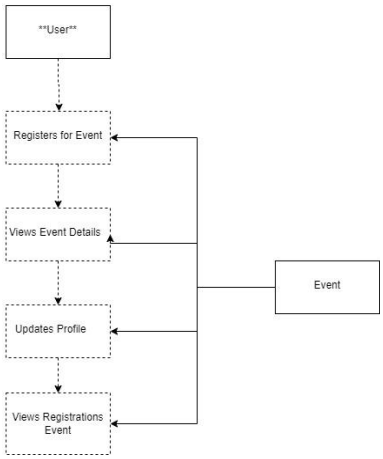
- **Admin Functions**:
  - **Event Management**: Admins can create, view, update, and delete events and manage event dates, locations, and descriptions.
  - **Customer Management**: Admins can view, modify, and delete customer registrations and generate reports.
  - **Event Analytics**: Admins can view reports on event popularity, number of registrations, and revenue generated by events.
- **Customer Functions**:
  - **Event Browsing**: Customers can browse available events and view event details.
  - **Event Registration**: Customers can register for events, select tickets, and provide personal details for event registration.
  - **Registration Status**: Customers can check their registration status and update their profile information, such as contact details.

3.2 User Interaction:

- **Admin Interaction**:
  - Admins will interact with the system via a text-based interface accessible via the command line or graphical command-line interface.
  - Admins can add, update, or delete events and access registration data.
- **Customer Interaction**:
  - Customers will interact with the system via a command-line interface for browsing and registering for events.
  - Customers can access event information, register, and track their registration status through command-line interactions or graphical user interfaces, if available.

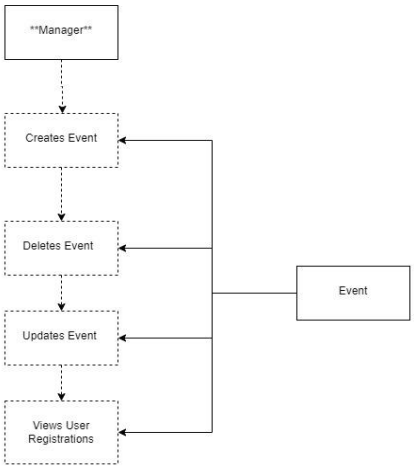
User Use Cases:

1. Register for Event
2. View Event Details
3. Update Profile
4. View Registered Events



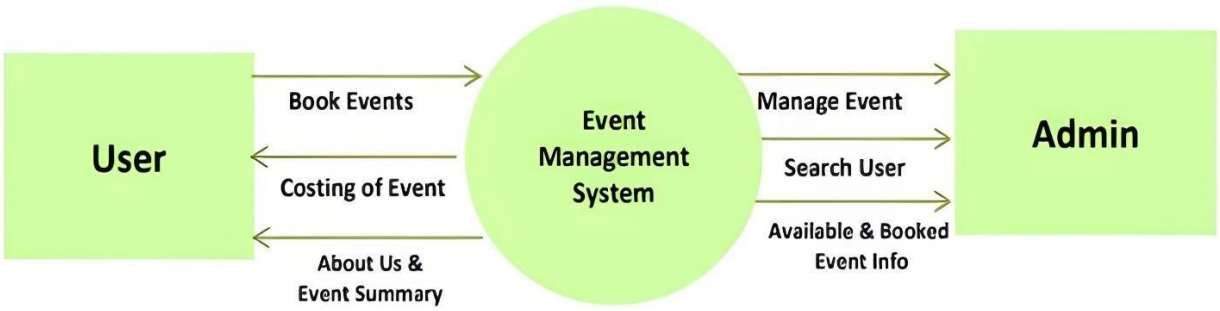
Admin Use Cases:

1. Creates Event
2. Deletes Event
3. Updates Event
4. Views User Registrations



3.3 Data Flow:

- **Admin Data Flow**:
  - Admins add, update, or delete events, assign event categories, and manage registrations. Data related to event details and registrations are stored in separate files.
- **Customer Data Flow**:
  - Customers access event listings, register for events, and track their registration status. Customer data, event details, and registration status are stored in text or binary files for efficient data management.



CHAPTER 4

4. Development Responsibilities:

4.1 Communication:

- **Project Manager (WahabEjaz)**:
  - Coordinates between the development team, testing team, and stakeholders.
  - Manages project timelines, budget, and resource allocation.
  - Ensures alignment of the project with stakeholder needs and business objectives.

4.2 Designing:

- **Design Team (Including Halar Khan and Irfan Shah)**:
  - Responsible for creating wireframes, designing system architecture, and ensuring the user interface (UI) and user experience (UX) are intuitive for both admins and customers.
  - Develops UI/UX mockups, conducts usability testing, and creates diagrams for event flow and data interaction.
  - Ensures that the system design is scalable, modular, and maintainable.

4.3 Coding and Testing:

- **Development Team (WahabEjaz, HalarKhan, Irfan Shah, and Talal Gul)**:
  - Writes the system code in C++.
  - Implements file handling techniques for event data, user registrations, and event statistics.
  - Ensures that code quality standards are maintained, and the system adheres to coding best practices.
- **Testing Team (Talal Gul and Irfan Shah)**:
  - Develops test cases based on the SRS and performs validation to ensure the system's functionality.
  - Conducts stress testing, load testing, and functional testing to validate system stability, performance, and usability.

4.4 Database Handling:

- Since the system uses file handling for storage, the development team will handle file-based data management, ensuring efficient data access and retrieval.

- Implement efficient file input/output operations to handle large volumes of event and customer data without delays.

4.5 Deployment and Maintenance:

- **Deployment Team (WahabEjaz, Halar Khan, Irfan Shah, and Talal Gul)**:
  - Responsible for deploying the EMS software to different environments (Windows, Linux, MacOS).
  - Ensures system performance and handles any issues related to the deployment.
- **Ongoing Maintenance**:
  - Ongoing maintenance, including bug fixes, security patches, and system updates, will be handled by the development team.
  - The deployment team will also handle system upgrades and ensure data is regularly backed up to avoid data loss.

CHAPTER 5

5. User Class and Characteristics:

5.1 Admin Class:

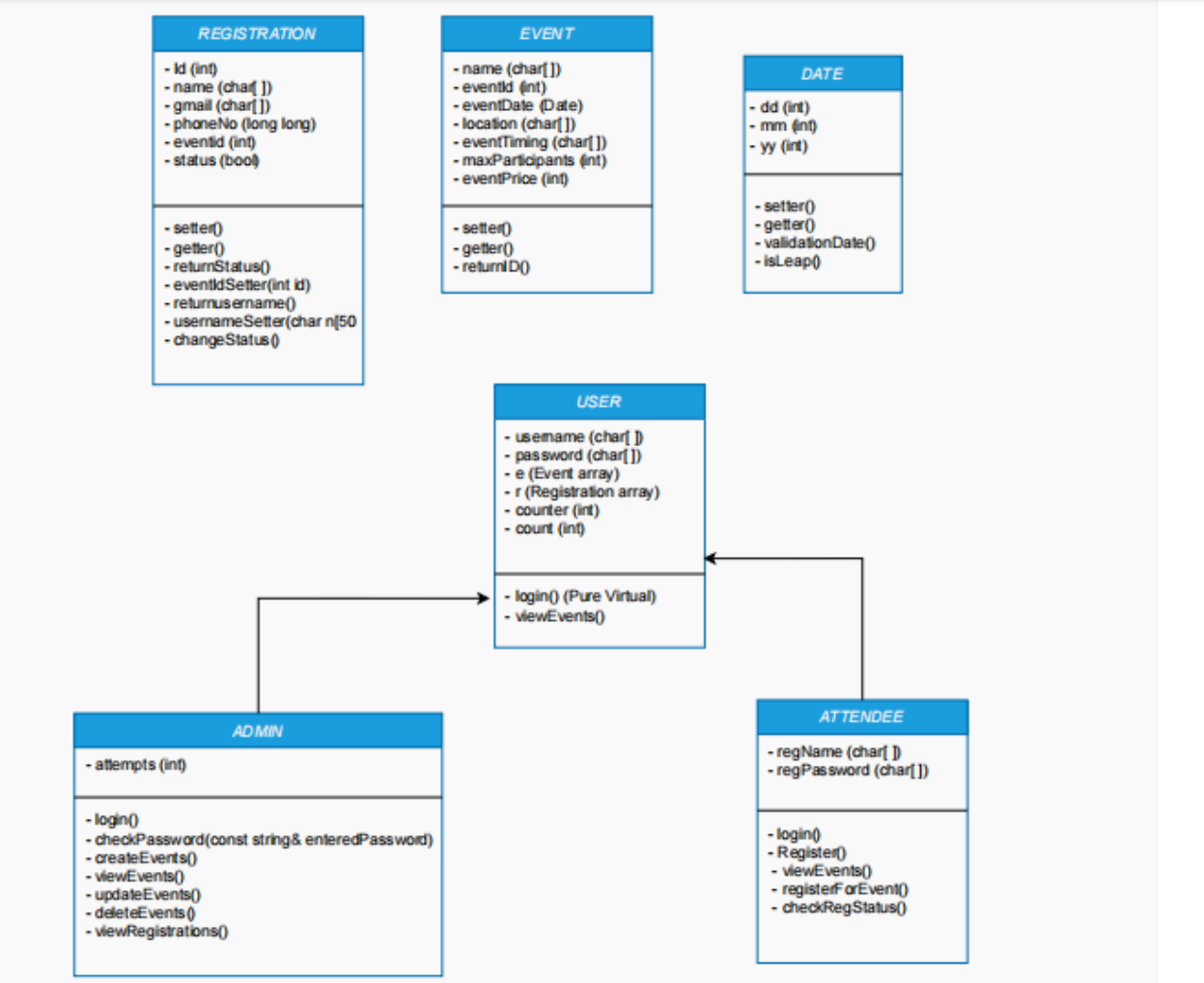
- **Attributes**:
  - **Username**: Unique identifier for the admin.
  - **Password**: Secure password stored in hashed format.
  - **Permissions**: Admins have full control over event management, customer registrations, and system settings.
- **Actions by Admin**:
  - **Create Event**:
    - Admin can create events, specifying event name, description, date, time, location, and registration fees.
  - . **Browse Events**:
    - Amin can view all available events, view event details.
  - **Update Event**:
    - Admin can modify event details, such as event description, date, venue, or time.
  - **Delete Event**:
    - Admin can remove events from the system, ensuring that outdated events do not clutter the database.
  - **Manage User Registrations**:
    - Admin can view customer registrations, modify registration details, and delete unwanted registrations.
  - **Event Categories Management**:
    - Admin can assign events to categories such as cultural, corporate, or academic, ensuring events are well-organized.

5.2 Customer Class:

- **Attributes**:
  - **Username**: Unique identifier for the customer.
  - **Password**: Secure password stored in hashed format.
  - **Registered Events**: List of events the customer has registered for.
- **Actions by Customer**:
  - **Browse Events**:

- Customers can view all available events, filter by category, view event details, and select events to register for.

- **Register for Events**:
  - Customers can register by entering their personal details and event preferences.
- **View Registration Status**:
  - Customers can view the status of their registration, including payment status and event details.



CHAPTER 6

6. System Features:

6.1 Functional Requirements:

- **Event Management**:
  - The system must allow the admin to create, update, delete, and view events with detailed information such as name, date, time, location, and category.
  - Admins must be able to categorize events and assign event categories, such as corporate, cultural, or academic.
  - Admins should be able to view detailed statistics for each event, such as registration count, revenue generated, and registration status.
- **Customer Registration**:
  - The system must allow customers to register for events by providing necessary details (name, email, phone number, ticket type, etc.).
  - The system should send a confirmation email or message to customers once their registration is successfully completed.
  - Customers must be able to view and modify their registration details.
  - Customers must be able to view all available events, search by categories, and filter based on event details.
- **Data Management**:
  - The system must use file handling for event and customer data storage. The system should support storing events and registrations in text files or binary files.
  - Data must be organized and accessible through structured file formats to ensure efficient data retrieval and storage.
  - The system should allow backups to be created regularly to prevent data loss.

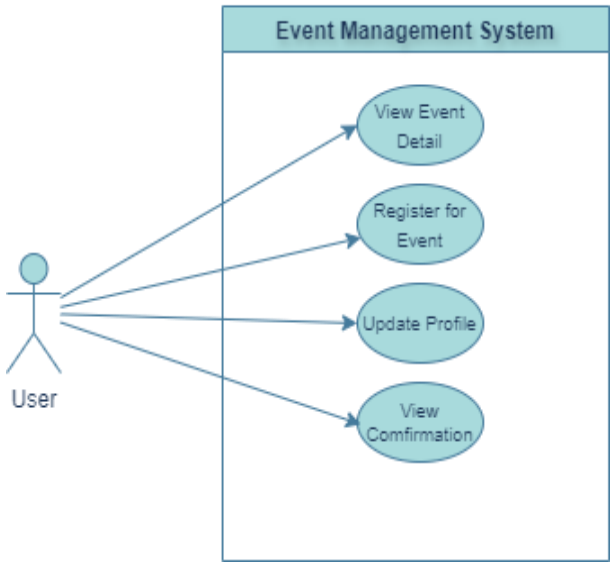
CHAPTER 7

7.FUNCTIONAL REQUIREMENT

7.1 User Interface:

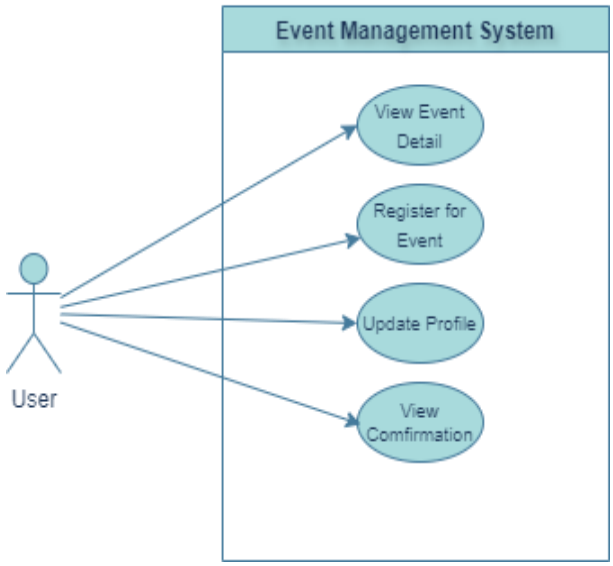
- **Admin Interface**:
  - The admin interface should provide the following features:
    - A command-line or graphical user interface to manage events.
    - An option to view event statistics, including the number of registrations, event revenue, and user participation.
    - A method to manage customer data, including viewing, updating, and deleting registrations.
    - A system for managing event categories.

Case Diagram of USer



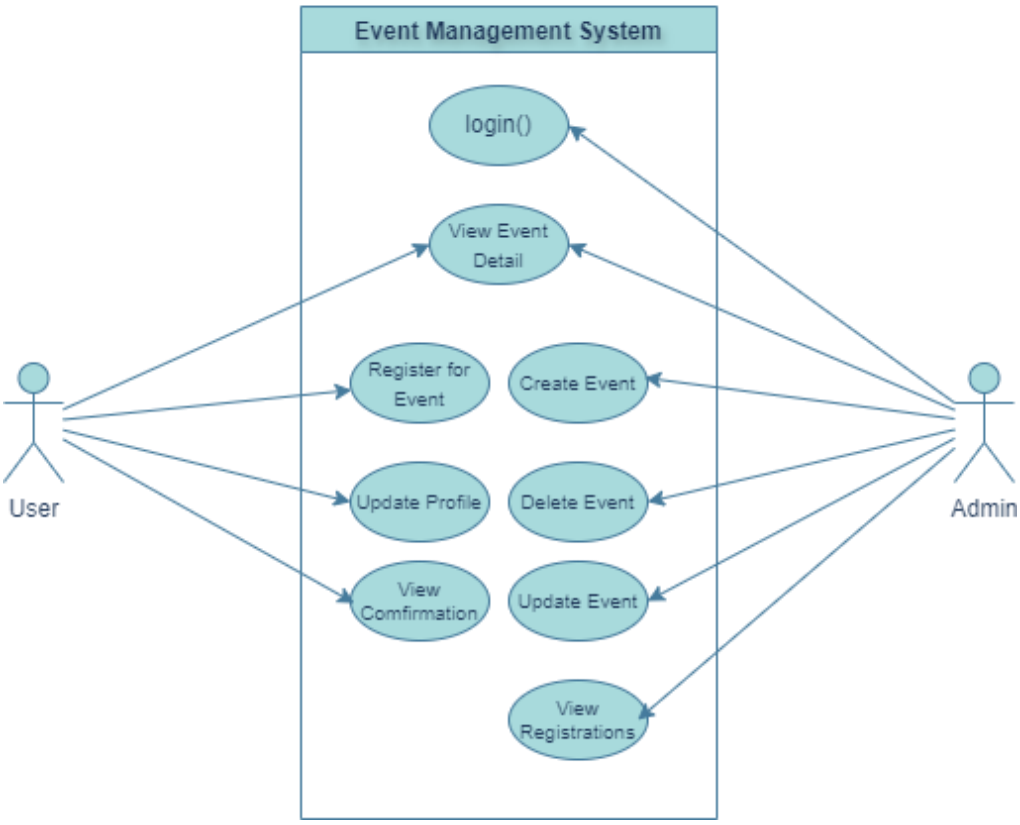
- **Customer Interface**:
    - The customer interface should be simple and easy to use, offering the following features:
        - A method to browse events, view event details, and filter events based on categories.
        - A registration form for customers to sign up for events, providing their contact details and ticket preferences.
        - A user profile page where customers can view and update their registration details.

Case Diagram of USer



Event Management System

case Diagram



7.2 Hardware Interface:

- The system should be compatible with standard desktop and laptop hardware, with the following minimum specifications:
  - **CPU**: Dual-core processor or equivalent.
  - **RAM**: At least 2GB RAM.
  - **Disk Space**: 500MB free disk space for installation.
  - **Display**: A screen capable of displaying at least 800x600 resolution.

7.3 Software Interface:

- The Event Management System (EMS) should interact with the operating system's file system for data storage and retrieval.
- The system should be compatible with the following platforms:
    - **Windows**: Windows 7 or higher.
    - **Linux**: Any distribution supporting C++ applications.
    - **MacOS**: Version 10.12 or higher.

7.4 Communication Interface:

- The communication will be through text-based interface(e.g. Command prompt).
- The system will not require any external communication interfaces such as network connectivity for basic functionality.
- For future versions, the integration with email services for sending registration confirmations and reminders can be added.

CHAPTER 8

8. Non Functional Requirements:

8.1 Performance Requirements:

- The system should handle a large number of events and registrations without significant performance degradation.
- The system should be able to process event registrations in real time, ensuring immediate registration confirmation.

8.2 Safety Requirements:

- All data entered by users (events, registrations) should be **persisted in files** securely.
- **Backup** of event data should be created regularly to ensure data integrity.

8.3 Security Requirements:

- All user data (such as passwords and personal information) should be securely stored and transmitted using encryption.
- The system should have an authentication mechanism to ensure that only authorized admins can access sensitive event data.
- Customer registration information must be stored securely to prevent unauthorized access.

8.4 Software Quality Attributes:

- **Usability Requirements**:
  - The system must have an intuitive interface, both for admins and customers, allowing users to navigate and interact with the system without difficulty.
  - The system should be easy to install and deploy on Windows, Linux, and MacOS platforms.
- **Reliability Requirements**:
  - The system should be stable and reliable, with minimal downtime.
  - Regular backups should be created automatically to ensure that critical data is not lost.
- **Maintainability Requirements**:
  - The system should be modular to allow for easy updates and maintenance.
  - Code should be well-documented, following industry standards, to make future maintenance and enhancement easier.

CHAPTER 9

9. Conclusion:

The **Event Management System (EMS)** is a desktop-based software application designed to facilitate the creation, registration, and management of events for both admins and customers. This document provides a detailed Software Requirements Specification (SRS) that outlines the system's functionality, design, and operational requirements, ensuring alignment between stakeholders and the development team.

The system will allow **admins** to create, update, delete, and manage events, as well as monitor customer registrations and event statistics. **Customers**, on the other hand, will be able to browse, register, and view event details, along with updating their registration status.

The software will be developed in **C++** using file handling for data storage, ensuring the system runs efficiently on Windows, Linux, and macOS platforms. It will have a simple text-based user interface for interaction. The system's non-functional requirements emphasize performance, usability, security, and scalability, ensuring it performs well even with large volumes of events and users.

The document specifies various components, including user classes (admin and customer), system features, platform requirements, and testing procedures. Additionally, it outlines the roles and responsibilities of each team involved in the development process, including designers, developers, and testers. Furthermore, the system's modular architecture ensures ease of maintenance and future upgrades.

In conclusion, this SRS document serves as the foundation for developing an efficient and user-friendly Event Management System that meets the needs of organizations and individual event participants. It ensures that all functional and non-functional requirements are well-defined, minimizing misunderstandings and delays in the software development lifecycle.

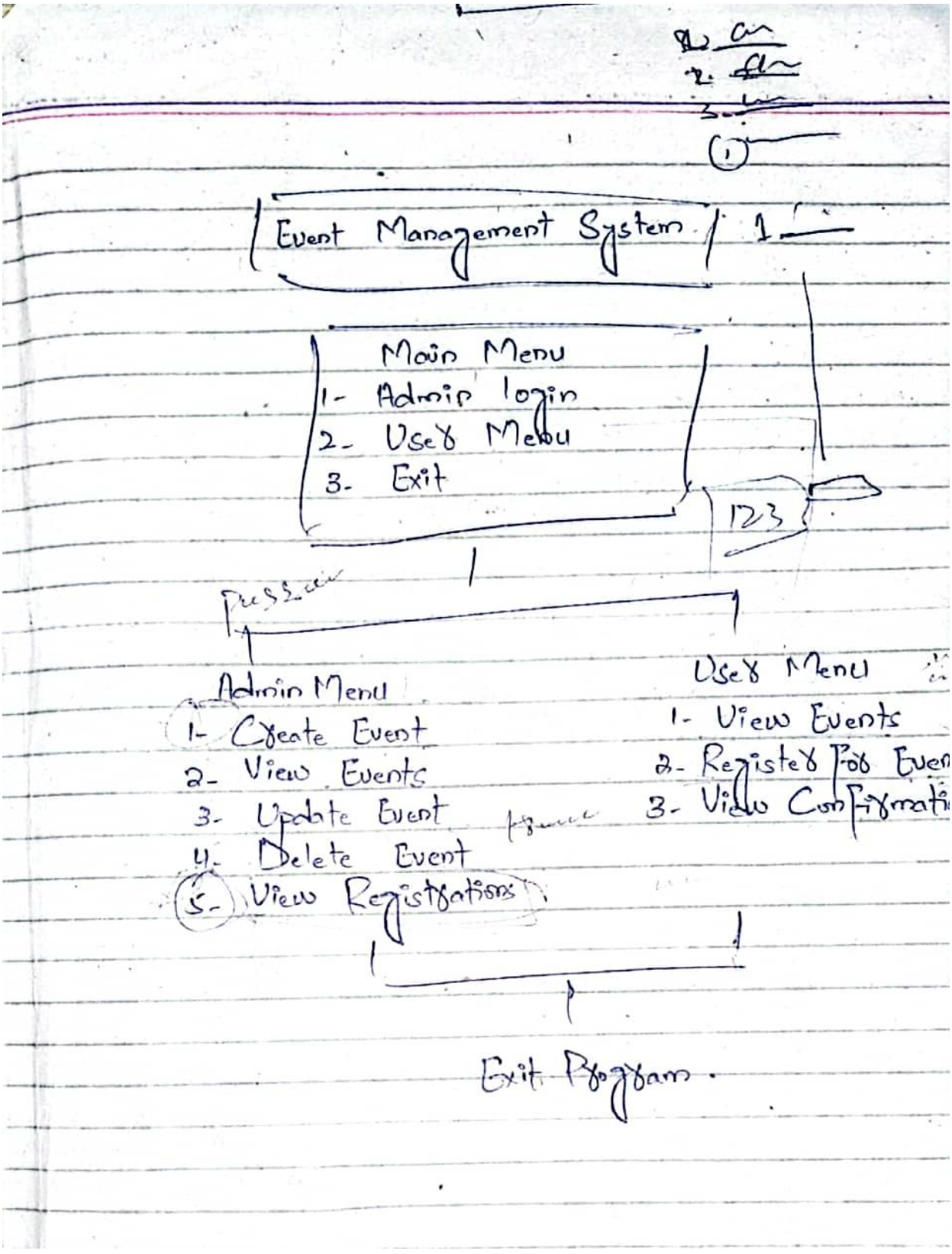
ANALYSICS

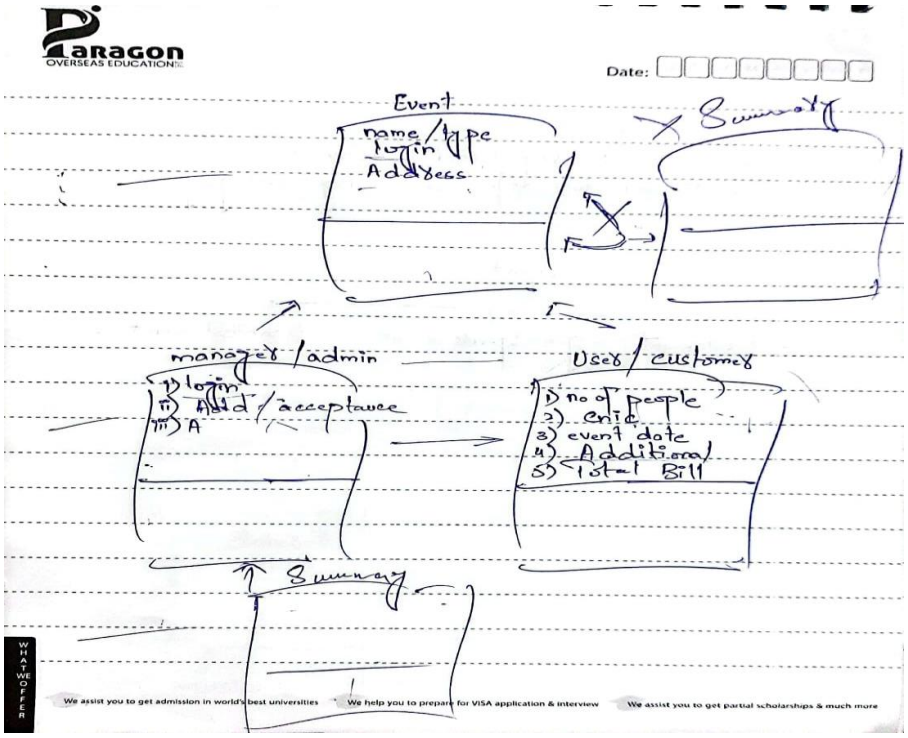
AND

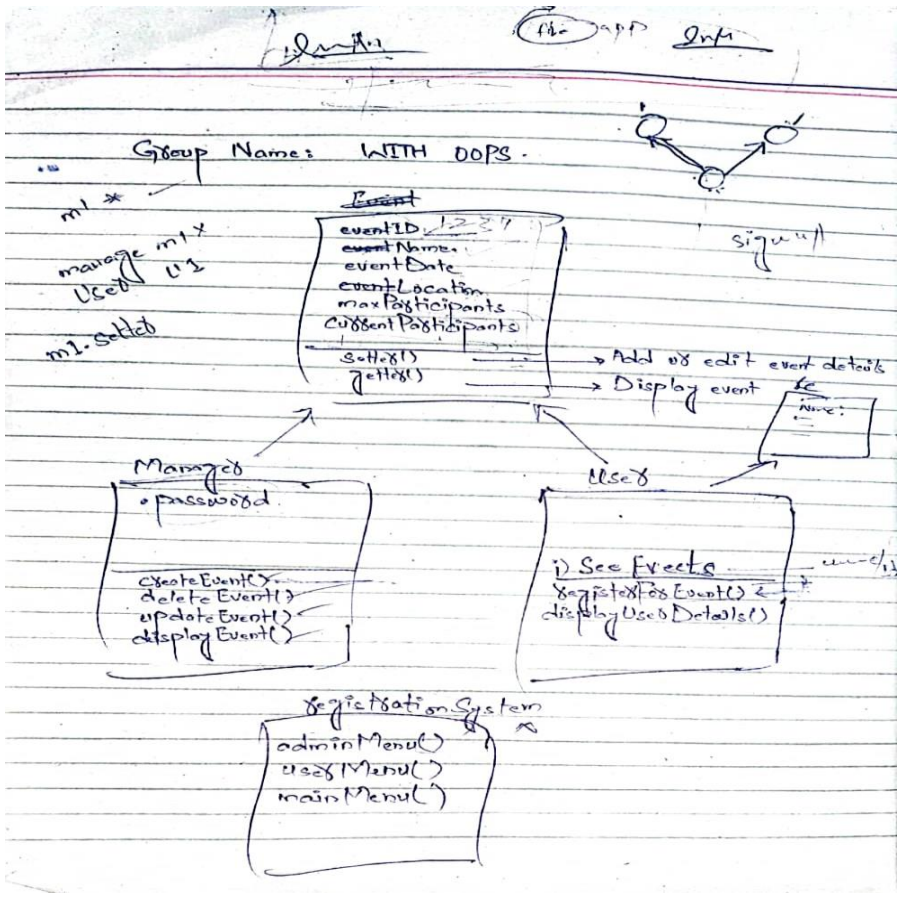
SOMEPAPERWORK

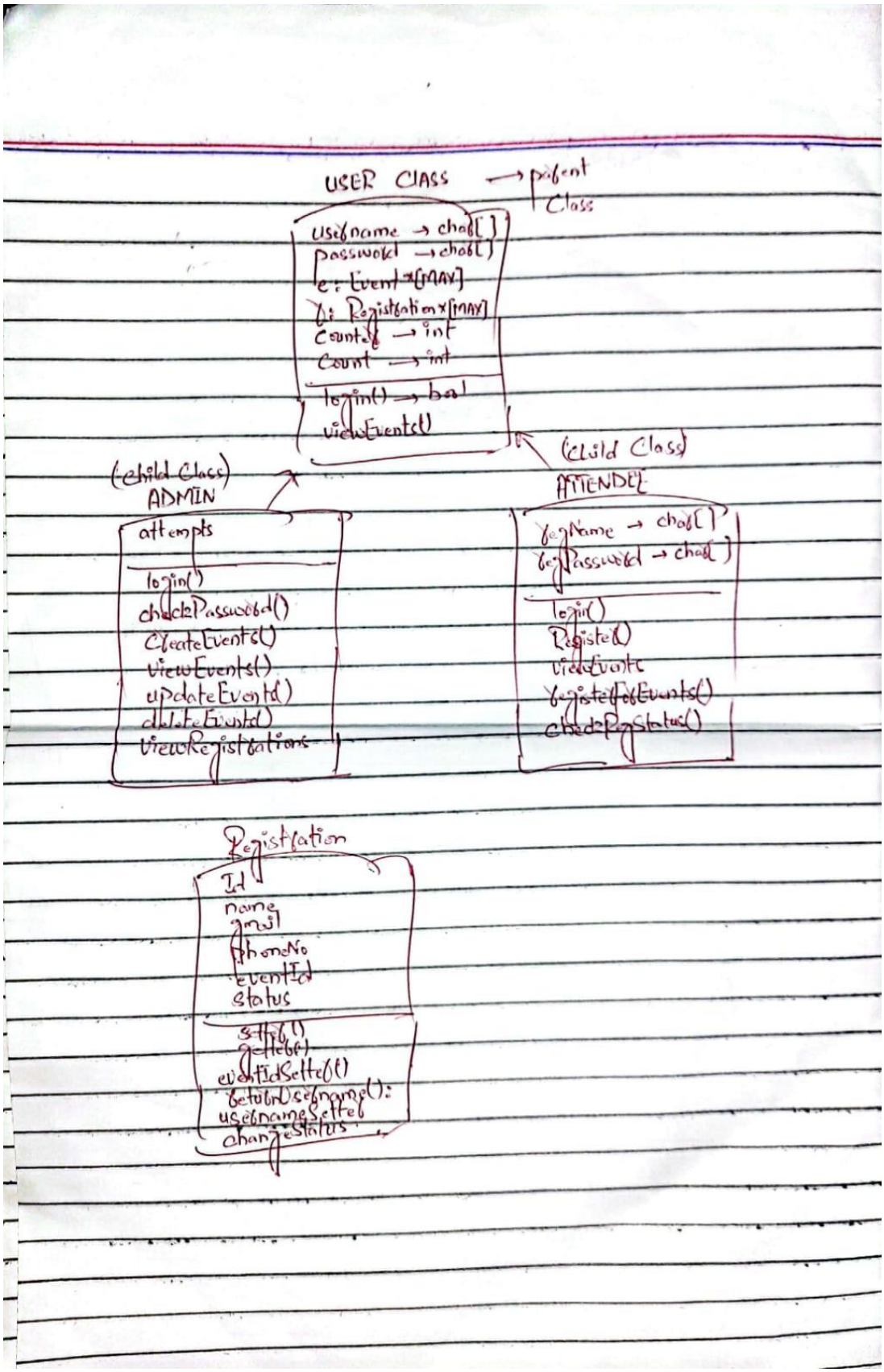
i) name
ii) cnic
iii) Address → Registration
iv) Date →
v) venue / place
   - system
   - user
vi) cost
vii) contact
viii) advanced payment
ix) no of people (MAX.)
x) login / registration
xi) photographer
xii) caterer
xiii) menu
xiv) Summary
xv) Additional
   - music
   - stage
   - movi
   - Car
   - photography

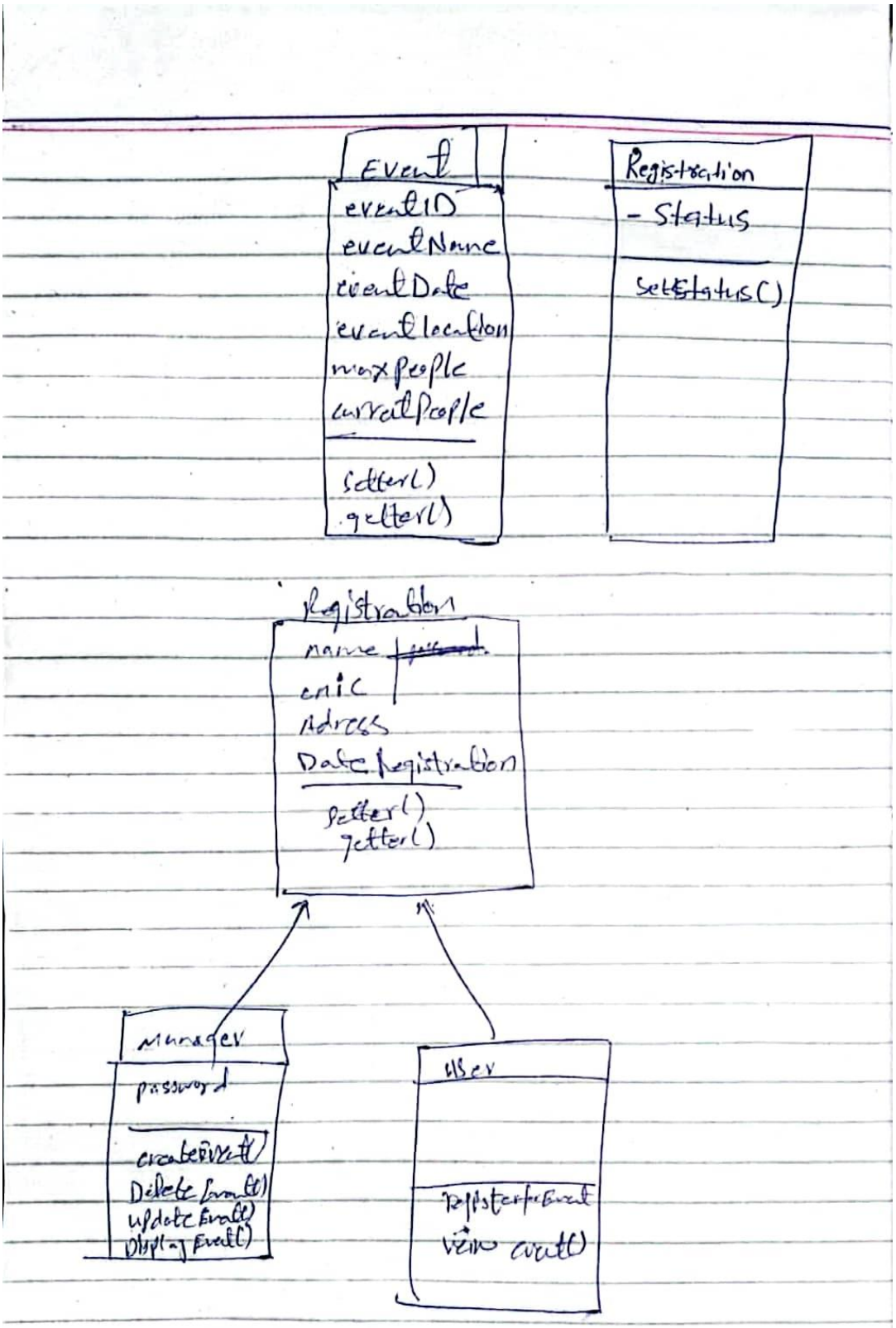
(x) Slot.
Slot 1: 9:00 – 11:00
Slot 2: 12:00 – 14:00
Slot 3: 15:00 – 17:00


Event Management System
Main Menu
1- Admin login
2- User Menu
3- Exit
123
Admin Menu
1- Create Event
2- View Events
3- Update Event
4- Delete Event
5- View Registrations
User Menu
1- View Events
2- Register For Even
3- View Confirmati
Exit Program.

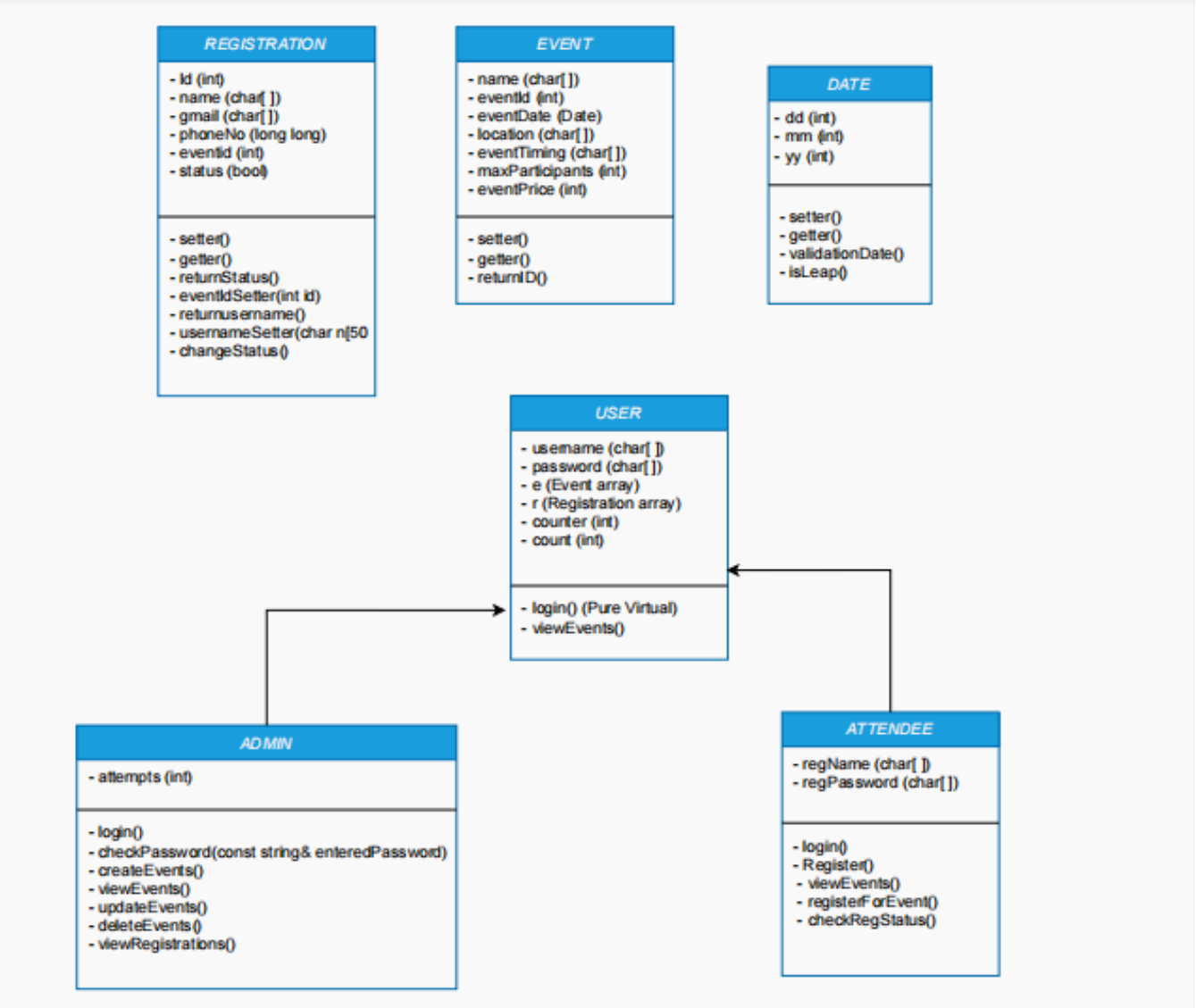

Paragon
OVERSEAS EDUCATION
Date:
Event
name/type
Login
Address
Summary
manager/admin
Login
Add/acceptance
A
User/customer
1) no of people
2) cnic
3) event date
4) Additional
5) Total Bill
Summary
WHAT WE OFFER
We assist you to get admission in world's best universities
We help you to prepare for VISA application & interface
We assist you to get partial scholarships & much more


Group Name: WITH OOPS.
Event
eventID
eventName
eventDate
eventLocation
maxParticipants
currentParticipants
setter()
getter()
Add or edit event details
Display event
Signup
Manager
password
createEvent()
deleteEvent()
updateEvent()
displayEvent()
User
i) See Events
registerForEvent()
displayUserDetails()
RegistrationSystem
adminMenu()
userMenu()
mainMenu()


USER CLASS
parent Class
username → char[ ]
password → char[ ]
e: Event *[MAX]
r: Registration *[MAX]
CountE → int
Count → int
login() → bool
viewEvents()
(child Class)
ADMIN
attempts
login()
checkPassword()
CreateEvents()
viewEvents()
updateEvent()
deleteEvent()
viewRegistrations
(child Class)
ATTENDEE
regName → char[ ]
regPassword → char[ ]
login()
Register()
viewEvents
registerForEvents()
checkRegStatus()
Registration
Id
name
gmail
phoneNo
eventId
status
setId()
getId()
eventIdSetter()
getUserUsername():
usernameSetter()
changeStatus


Event
eventID
eventName
eventDate
eventlocation
maxPeople
currentPeople
setter()
getter()
Registration
- Status
setStatus()
Registration
name
cnic
Adress
Date_Registration
setter()
getter()
Manager
password
createEvent()
DeleteEvent()
updateEvent()
DisplayEvent()
User
RegisterforEvent
viewEvent()

CLASS DIAGRAM


REGISTRATION
- Id (int)
- name (char[ ])
- gmail (char[ ])
- phoneNo (long long)
- eventId (int)
- status (bool)
- setter()
- getter()
- returnStatus()
- eventIdSetter(int id)
- returnusername()
- usernameSetter(char n[50
- changeStatus()
EVENT
- name (char[ ])
- eventId (int)
- eventDate (Date)
- location (char[ ])
- eventTiming (char[ ])
- maxParticipants (int)
- eventPrice (int)
- setter()
- getter()
- returnID()
DATE
- dd (int)
- mm (int)
- yy (int)
- setter()
- getter()
- validationDate()
- isLeap()
USER
- username (char[ ])
- password (char[ ])
- e (Event array)
- r (Registration array)
- counter (int)
- count (int)
- login() (Pure Virtual)
- viewEvents()
ADMIN
- attempts (int)
- login()
- checkPassword(const string& enteredPassword)
- createEvents()
- viewEvents()
- updateEvents()
- deleteEvents()
- viewRegistrations()
ATTENDEE
- regName (char[ ])
- regPassword (char[ ])
- login()
- Register()
- viewEvents()
- registerForEvent()
- checkRegStatus()

CODE SAMPLE

```cpp
//EVENT MANAGEMENT SYSTEM
/*The purpose of this project(Event Management System), is to streamline the planning,
 coordination, and execution of events by efficiently managing various event details and partici
 //Programmers: Halar Khan,Wahab Ejaz,Muhammad Irfan,Talal Gul
 //Compiler: gcc
 //Date 1-Jan-2025
#include <iostream>
#include <string>
#include <limits>
#include <fstream>
#include <iomanip>
#include <cctype>
#include <regex>
#include <windows.h>
#include <thread>
#include <chrono>

using namespace std;

const int MAX = 500;


// Function Prototypes
int getConsoleWidth();
void printCentered(const string& text);
void displayTimer(int seconds);
bool validInput();
bool isLeapYear(int year);
bool isValidDate(int day, int month, int year);

// ------------------- Utility Functions -------------------

// Get console width for centering text
int getConsoleWidth() {
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    if (GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi)) {
        return csbi.dwSize.X;
    }
    return 80; // Default width if unable to retrieve
}

// Print centered text in console
void printCentered(const string& text) {
    int consoleWidth = getConsoleWidth();
    int padding = (consoleWidth - text.length()) / 2;
    if (padding > 0) {
        cout << string(padding, ' ') << text << "\n";
    } else {
        cout << text << "\n";
    }
}

// Display a countdown timer
void displayTimer(int seconds) {
    for (int i = seconds; i > 0; i--) {
        cout << "\rPlease wait " << i << " seconds before trying again... " << flush;
        this_thread::sleep_for(chrono::seconds(1));
    }
    cout << "\rYou can now try logging in again." << endl << endl;
}
```

```cpp
// Validate user input
bool validInput() {
    if (cin.good()) {
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return true;
    }
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << "Invalid input. Please try again." << endl;
    return false;
}

// Check if a year is a Leap year
bool isLeapYear(int year) {
    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}

// Validate date
bool isValidDate(int day, int month, int year) {
    if (month < 1 || month > 12) {
        cout << "Invalid month! Please enter a value between 1 and 12." << endl;
        return false;
    }
    if ((month == 4 || month == 6 || month == 9 || month == 11) && day > 30) {
        cout << "Invalid day for the given month!" << endl;
        return false;
    }
    if (month == 2) {
        if (day > (isLeapYear(year) ? 29 : 28)) {
            cout << "Invalid day for February!" << endl;
            return false;
        }
    } else if (day < 1 || day > 31) {
        cout << "Invalid day! Please enter a value between 1 and 31." << endl;
        return false;
    }
    if (year < 1900 || year > 2026) {
        cout << "Invalid year! Please enter a value between 1900 and 2026." << endl;
        return false;
    }
    return true;
}

bool validationName(const string& name) {
    if (name.length() < 3 || name.length() > 20) {
        cout << "Invalid name!" << endl;
        return true;
    }
    return false;
}

bool validationAddress(const string& address) {
    if (address.length() < 5 || address.length() > 30) {
        cout << "Invalid address!" << endl;
        return true;
    }
    return false;
}

bool isValidGmail(const char* gmail) {
    // Regular expression for Gmail format
```

```
    regex gmailPattern("^[a-zA-Z0-9._]+@gmail\\.com$");

    // Check if the input matches the pattern
    return regex_match(gmail, gmailPattern);
}

bool isValidPhoneNumber(const char* phoneNo) {
    // Regular expression for phone number format
     regex phonePattern("^\\+?92[ -]?(\\(?\\d{3}\\)?)[ -]?\\d{7}$");

    // Check if the input matches the pattern
    return regex_match(phoneNo, phonePattern);
}
// -------------------- Class Definition --------------------

class Date {
    int dd, mm, yy;

public:
    // Constructor
    Date() : dd(1), mm(1), yy(1900) {}

    // Member Functions
    void setter();
    void getter() const;
};

// Setter: Input and validate date
void Date::setter() {
    do {
        cout << "Enter Event Date (dd): ";
        cin >> dd;
        cout << "Enter Event Month (mm): ";
        cin >> mm;
        cout << "Enter Event Year (yy): ";
        cin >> yy;
    } while (!isValidDate(dd, mm, yy) || !validInput());
}

// Getter: Display the date
void Date::getter() const {
    cout<< "\t\t\tEvent Date is  : " << dd << "/" << mm << "/" << yy ;
}

class Event
{
protected:
    char Name[50];
    int eventId;
    Date eventDate;
    char location[50];
    char eventTiming[60];
    int maxParticipants;
    int eventPrice;
public:
    void setter();
    void getter();
    int returnID();

};
```

```
184: void Event::setter()
185: {
186:     do{
187:     cout<<"Enter Event ID: ";
188:     cin>>eventId;
189:     }while (!validInput());
190:
191:     do {
192:     cout << "Enter Event Name: ";
193:     cin.getline(Name, 50);
194:     } while (validationName(Name));
195:
196:     eventDate.setter();
197:
198:     do {
199:     cout << "Enter Event Location: ";
200:     cin.getline(location, 50);
201:     } while (validationAddress(location));
202:
203:     do {
204:     cout << "Enter Event Timing(HH:MM): ";
205:     cin.getline(eventTiming, 60);
206:     if (!eventTiming[0] || strlen(eventTiming) > 60 || !strchr(eventTiming, ':'))
207:     cout << "Invalid input. Please enter a valid event timing.\n";
208:     } while (!eventTiming[0] || strlen(eventTiming) > 60 || !strchr(eventTiming, ':'));
209:
210:
211:     do{
212:     cout << "Enter Num Max Participants: ";
213:     cin >> maxParticipants;
214:     }while (!validInput());
215:
216:     do{
217:     cout << "Enter Event ticket Price: ";
218:     cin >> eventPrice;
219:     }while (!validInput());
220: }
221:
222: void Event :: getter()
223: {
224:     cout << "\n\t\t\tEvent ID       : " << eventId << endl;
225:     cout << "\t\t\tEvent Name     : " << Name << endl;
226:     // Assuming the Date class has a getter method to print the event date.
227:     eventDate.getter();
228:     cout << "\n\t\t\tEvent Location : " << location << endl;
229:     cout << "\t\t\tEvent Timing   : " << eventTiming << endl;
230:     cout << "\t\t\tEvent Capacity : " << maxParticipants <<" People"<< endl;
231:     cout << "\t\t\tTicket Price   : $" << eventPrice << endl;
232:     cout<<"--------------------------------------------------------------------------------"<<endl;
233: }
234:
235:
236: int Event::returnID()
237: {
238:     return eventId;
239: }
240: void save2disk(Event* ee[], int nn);
241: void readFfile(Event* ee[], int& nn);
242:
243: class Registration
244: {
```

```
    int Id;
    char name[50];
    char gmail[60];
    char phoneNo[20];
    int eventid;
    bool status=false;
public:
    void getter();
    void setter();
    void eventIdSetter(int id);
    char* returnusername();
    void usernameSetter(char n[50]);
    void changeStatus();
    bool returnStatus();
};

void Registration::setter()
{
    do{
    cout<<"Enter your Id: ";
    cin>>Id;
    }while (!validInput());

    do {
    cout<<"Enter Gmail: ";
    cin.getline(gmail,60);
    if(!isValidGmail(gmail))
    cout << "Invalid Gmail. Plz enter a valid Gmail address in standard format!" << endl;
    } while (!isValidGmail(gmail));

    do {
    cout<<"Enter Your Contact No: ";
    cin.getline(phoneNo,20);
    if (!isValidPhoneNumber(phoneNo)) {
    cout << "Invalid phone number. Please enter a valid phone number in Pakistani format (e.g., +92 300 1
    }
    } while (!isValidPhoneNumber(phoneNo));
}

void Registration::getter()
{
    cout<<"\n\t\t\tEvent Id     : "<<eventid;
    cout<<"\n\t\t\tUser Id      : "<<Id;
    cout<<"\n\t\t\tUser Name    : "<<name;
    cout<<"\n\t\t\tUser gmail   : "<<gmail;
    cout<<"\n\t\t\tUser phone NO: "<<phoneNo;
    if(status==true)
        cout<<"\n\n\t\t\t****Appliction Approved**** ";
    else
        cout<<"\n\n\t\t\tAppliction Pending..";
    cout<<"\n_____ "<<endl;

}

void Registration::eventIdSetter(int id)
{
    eventid=id;
}

char* Registration::returnusername()
{
```

```
306:
307:     return name;
308: }
309:
310: void Registration::usernameSetter(char n[50])
311: {
312:     strcpy(name, n);
313: }
314:
315: void Registration::changeStatus()
316: {
317:     status=true;
318: }
319:
320: bool Registration::returnStatus()
321: {
322:     return status;
323: }
324:
325:
326: void clearEventArray(Event* e[], int count);
327: void clearRegistrationArray(Registration* r[], int count);
328: void saveApplications(Registration* r[],int nn);
329: void loadApplications(Registration* r[],int& nn);
330:
331: void clearEventArray(Event* e[], int count) {
332:     for (int i = 0; i < count; i++) {
333:         delete e[i]; // Delete each dynamically allocated event
334:         e[i] = nullptr; // Reset the pointer to nullptr to avoid dangling pointer
335:     }
336: }
337:
338: void clearRegistrationArray(Registration* r[], int count) {
339:     for (int i = 0; i < count; i++) {
340:         delete r[i]; // Delete each dynamically allocated registration
341:         r[i] = nullptr; // Reset the pointer to nullptr to avoid dangling pointer
342:     }
343: }
344: // Base Class
345: class User {
346: protected:
347:     char username[50];
348:     char password[50]="admin123";
349:     Event* e[MAX];
350:     Registration* r[MAX];
351:     int counter=0,count=0;
352:
353: public:
354:     virtual bool login() = 0; // Pure virtual function
355:     void viewEvents();
356: };
357:
358: void User::viewEvents()
359: {
360:     readFfile(e,count);
361:     for(int i=0;i<count;i++){
362:         cout<<"\t\t\t\t\t|  "<<i+1<<"  |"<<endl;
363:         cout<<"\t\t\t\t\t ------\n";
364:         e[i]->getter();
365:     }
366:
```

```
    // Clear the dynamically allocated memory for events after viewing
    clearEventArray(e, count);
}



class Admin : public User
{
protected:
    int count = 0;
public:
    int attempts = 0;
    bool login();
    bool checkPassword(const string& enteredPassword);
    void createEvents();
    void viewEvents();
    void updateEvents();
    void deleteEvents();
    void viewRegistrations();
};

bool Admin::checkPassword(const string& enteredPassword)
{
    return enteredPassword == password;
}


bool Admin::login()
{
    printCentered("_____________");
    printCentered(" ****LOGIN**** ");
    printCentered("_____________");

    string enteredPassword;
    int attempts = 0;
    const int maxAttempts = 3;
    bool isAuthenticated = false;

    do
    {
        while (attempts < maxAttempts)
        {
            cout << "Enter password: ";
            cin >> enteredPassword;

            if (checkPassword(enteredPassword))
            {
                cout << endl;
                cout << ":) Login successful!" << endl;
                isAuthenticated = true; // Successful login
                int width = getConsoleWidth();
                for (int i = 0; i < width; i++) cout << "-";
                cout << endl;
                break;
            }
            else
            {
                attempts++;
                cout << endl;
                cout << "Incorrect Password. Attempts left: " << (maxAttempts - attempts) << endl;
            }
```

```
        }

        if (!isAuthenticated && attempts >= maxAttempts)
        {
            cout << "Too many failed attempts. Please wait 30 seconds before trying again.\n";
            displayTimer(30);
            attempts = 0; // Reset attempts after the timer
        }

    } while (!isAuthenticated);

    return true; // Return true if login is successful
}

void Admin::createEvents() {
    readFfile(e, count);
    cout << "-------------------------\n";
    char ch;
    do
    {
        if (count < MAX) {
            e[count] = new Event;  // Allocate memory for a new event
            e[count]->setter();   // calling Event setter
            count++;
        }
        else {
            cout << "Cannot add more events. Maximum limit reached." << endl;
            break;
        }
        cout << "Do you want add another Event(Y/N)? : ";
        cin >> ch;
        if ((ch != 'Y') && (ch != 'y'))
            break;
    } while (true);
    save2disk(e, count);
    // Clear the dynamically allocated memory for events after saving to disk
    clearEventArray(e, count);
}

void Admin::viewEvents()
{
    User::viewEvents();
}


void Admin::updateEvents()
{
    readFfile(e, count);
    int ID, found = 0;
    cout << "Enter Event ID to Update: ";
    cin >> ID;
    for (int i = 0; i < count; i++)
    {
        if (e[i]->returnID() == ID)
        {
            cout << "Updating Event....\n";
            e[i]->setter();
            found = 1;
            cout << " Event Updated successfull.\n";
        }
    }
```

```
489:
490:     if (found == 0)
491:         cout << "Event not found.";
492:
493:     ofstream outfile;
494:     outfile.open("Events.DAT", ios::out | ios::binary);
495:     if (!outfile)
496:     {
497:         cout << "Couldn't open file.\n";
498:         return;
499:     }
500:     outfile.write((char*)&count, sizeof(int));
501:     // Loop through each object and save its type and data
502:     for (int i = 0; i < count; i++)
503:     {
504:         outfile.write((char*)e[i], sizeof(Event));
505:     }
506:     outfile.close();
507:     cout << "Data saved successfully!\n";
508:
509:     // Clear the dynamically allocated memory for events after viewing
510:     clearEventArray(e, count);
511: }
512:
513: void Admin::deleteEvents()
514: {
515:     readFfile(e, count);
516:     int ID, found = 0;
517:     cout << "Enter Event ID to Delete: ";
518:     cin >> ID;
519:     for (int i = 0; i < count; i++)
520:     {
521:         if (e[i]->returnID() == ID)
522:         {
523:             cout << "Deleting Event....";
524:             e[i] = e[i + 1];
525:             count--;
526:             found = 1;
527:             cout << " Event Deleted successfull.";
528:         }
529:     }
530:
531:     if (found == 0)
532:         cout << "Event not found.";
533:
534:     ofstream outfile;
535:     outfile.open("Events.DAT", ios::out | ios::binary);
536:     if (!outfile)
537:     {
538:         cout << "Couldn't open file.\n";
539:         return;
540:     }
541:     outfile.write((char*)&count, sizeof(int));
542:     // Loop through each object and save its type and data
543:     for (int i = 0; i < count; i++)
544:     {
545:         outfile.write((char*)e[i], sizeof(Event));
546:     }
547:     outfile.close();
548:     cout << "Data saved successfully!\n";
549:
```

```
    // Clear the dynamically allocated memory for events after viewing
    clearEventArray(e, count);
}



void Admin::viewRegistrations()
{
    loadApplications(r,counter);
    char ch;
    for(int i=0;i<counter;i++)
    {
        cout<<"\t\t\tApplication # "<<i+1<<endl;
        r[i]->getter();
    }
    for (int j=0;j<counter;j++)
    {   if(r[j]->returnStatus()==false)
        {
            cout<<endl;
            printCentered("*APPROVE APPLICATIONS*");
            cout<<"_____________________________________________________________________________\n";
            cout<<"\t\t\tApplication # "<<j+1<<endl;
            r[j]->getter(); // Display registration details
            cout<<"Enter[Y/y] to approve Application: ";
            cin>>ch;
            if(ch=='y'||ch=='Y')
            r[j]->changeStatus();
        }
    }
    saveApplications(r,counter);
    // clear dynamically allocated memory
    clearRegistrationArray(r, counter);
}


void save2disk(Event* ee[], int nn) {
    ofstream outfile("Events.DAT", ios::out | ios::binary);
    if (!outfile) {
        cout << "Couldn't open file OR No Events available.\n";
        return;
    }

    outfile.write((char*)&nn, sizeof(nn)); // Save the number of events
    for (int i = 0; i < nn; i++) {
        outfile.write((char*)ee[i], sizeof(Event));
    }

    outfile.close();
    cout << "Data saved successfully!\n";
}


void readFfile(Event* ee[], int& nn) {
    ifstream infile("Events.DAT", ios::in | ios::binary);
    if (!infile) {
        cout << "Couldn't open file OR No Events available.\n";
        return;
    }

    infile.read((char*)&nn, sizeof(nn)); // Read the number of events
    for (int i = 0; i < nn; i++) {
```

```
        ee[i] = new Event; // Allocate memory for each Event object
        infile.read((char*)ee[i], sizeof(Event));
    }

    infile.close();
}


//Drived class from User class
class Attendee :public User
{
    char regName[50];
    char regPassword[50];
public:
    bool login();
    void Register();
    void viewEvents();
    void registerForEvent();
    void checkRegStatus();
};

bool Attendee::login()
{
    printCentered("_____________");
    printCentered(" ****LOGIN**** ");
    bool isAuthenticated=false;
    cout<<"Enter Username:";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cin.getline(username,50);
    cout<<"Enter Password:";
    cin.getline(password,50);
    ifstream infile("users.dat", ios::in | ios::binary);
    if (!infile) {
        cout << "User not Registered.\n";
    }


    while (infile.read((char*)regName, sizeof(regName))) {
        infile.read((char*)regPassword, sizeof(regPassword));

        // Check if the entered credentials match the ones in the file
        if (strcmp(username, regName) == 0 && strcmp(password, regPassword) == 0) {
            isAuthenticated = true;
            cout << "Login successful! Welcome, " << username << "!\n";
            break;
        }
    }

    infile.close();
    if (!isAuthenticated) {
        cout << "Login Failed! Wrong Password or User doesn't Exist.. " << "!\n";
        return false;
    }
    return true;
}

void Attendee::Register()
{
    printCentered("_____________________");
    printCentered("   ****REGISTER****   ");
    printCentered("_____________________");
```

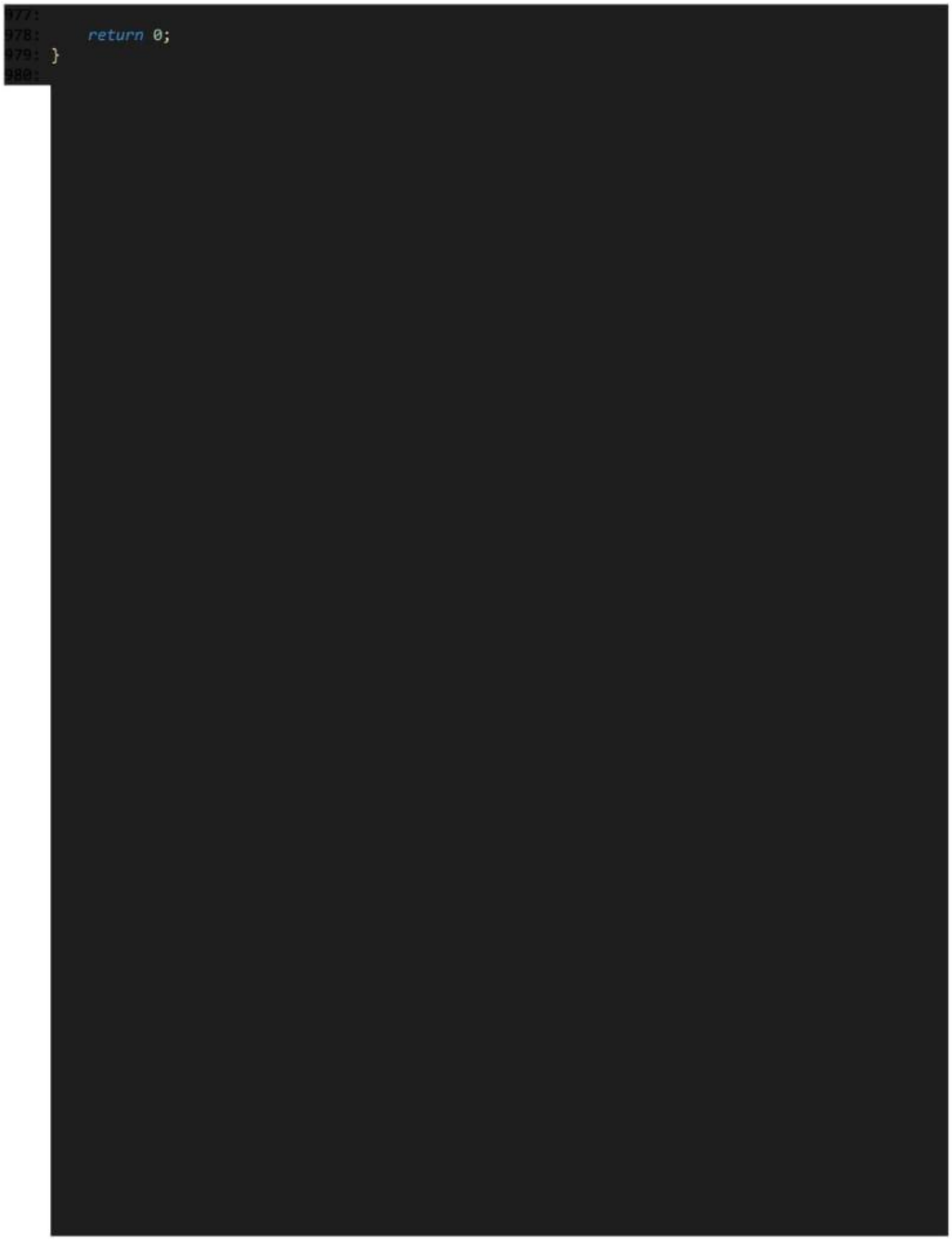
```
672:
673:     cout<<"Enter Username For Account Registration:";
674:     cin.ignore(numeric_limits<streamsize>::max(), '\n');
675:     cin.getline(regName,50);
676:     cout<<"Enter Password For Account Registration:";
677:     cin.getline(regPassword,50);
678:
679:     // Open a file in append mode to store username and password
680:     ofstream outfile("users.dat", ios::app | ios::binary);
681:     if (!outfile) {
682:         cout << "Error opening file for writing.\n";
683:         return;
684:     }
685:
686:     outfile.write((char*)regName, sizeof(regName));       // Store username
687:     outfile.write((char*)regPassword, sizeof(regPassword)); // Store password
688:
689:     outfile.close();
690:     cout << "Account registered successfully and saved to file.\n";
691: }
692:
693: void Attendee::registerForEvent()
694: {
695:     User::viewEvents();
696:     loadApplications(r,counter);
697:     readFfile(e,count);
698:     int id;
699:     char ch;
700:     bool exsists=false;
701:     do
702:     {
703:         cout<<"Enter Event Id to Register:";
704:         cin>>id;
705:         for(int j=0;j<count;j++){
706:
707:             if(e[j]->returnID()==id){
708:                 cout<<"Event Exsists...\n";
709:                 exsists=true;
710:                 r[counter]=new Registration;
711:                 r[counter]->eventIdSetter(id);
712:                 r[counter]->usernameSetter(username);
713:                 r[counter]->setter();
714:                 counter++;
715:             }
716:         }
717:
718:         if(exsists==false)
719:             cout<<"Event doesn't Exsist.\n";
720:
721:         cout<<"Do you want to register for another Event: ";
722:         cin>>ch;
723:     }while(ch!='N'&&ch!='n');
724:     saveApplications(r,counter);
725:     // clear dynamically allocated memory
726:     clearRegistrationArray(r, counter);
727:     clearEventArray(e, count);
728: }
729:
730: void Attendee::checkRegStatus()
731: {
732:     loadApplications(r,counter);
```

```
    bool found=false;
    for (int i=0;i<counter;i++)
    {
        if (strcmp(username, r[i]->returnusername()) == 0) {
            r[i]->getter(); // Display registration details
            found = true;
        }
    }
    if(!found)
    cout<<"\n***NO Applications Under Username: "<<username<<endl;
    // clear dynamically allocated memory
    clearRegistrationArray(r, counter);
}

void Attendee::viewEvents()
{
    User::viewEvents();
}


void saveApplications(Registration* r[],int nn)
{
     ofstream outfile("applications.dat", ios::out | ios::binary);
    if (!outfile) {
        cout << "Couldn't open file for writing.\n";
        return;
    }

    outfile.write((char*)&nn, sizeof(nn)); // Save the number of events
    for (int i = 0; i < nn; i++) {
        // Write individual fields using getters
       outfile.write((char*)r[i],sizeof(Registration));
    }

    outfile.close();
    cout << "Data saved successfully!\n";
}

void loadApplications(Registration* r[],int& nn)
{
    ifstream infile("applications.dat", ios::in | ios::binary);
    if (!infile) {
        cout << "Couldn't open file for reading.\n";
        return;
    }

    infile.read((char*)&nn, sizeof(nn)); // Read the number of events
    for (int i = 0; i < nn; i++) {
        r[i] = new Registration; // Allocate memory for each Event object
        infile.read((char*)r[i],sizeof(Registration));
    }
    infile.close();

}

void adminMenu() {
    int admChoice;
    Admin A;
    if (A.login()) {
        do {
            printCentered("==========================");
```

```
            printCentered("| 1. CREATE EVENTS         |");
            printCentered("| 2. VIEW EVENTS           |");
            printCentered("| 3. UPDATE EVENTS         |");
            printCentered("| 4. DELETE AN EVENT       |");
            printCentered("| 5. VIEW REGISTRATIONS    |");
            printCentered("| 6. RETURN TO MAIN MENU   |");
            printCentered("| 7. EXIT                  |");
            printCentered("============================");
            cout << "Enter your choice: ";

            // Validate the input for admChoice
            cin >> admChoice;

            if (cin.fail()) {
                cout << "Invalid input! Please enter a number between 1 and 8." << endl;
                cin.clear(); // Clear the error flag
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest of the invalid in
                continue; // Skip the rest of the loop and prompt again
            }

            switch (admChoice) {
                case 1:
                    A.createEvents();
                    break;
                case 2:
                    A.viewEvents();
                    break;
                case 3:
                    A.updateEvents();
                    break;
                case 4:
                    A.deleteEvents();
                    break;
                case 5:
                    A.viewRegistrations();
                    break;
                case 6:
                   cout << "Returning to main menu...\n";
                    return;  // Return to the main menu
                case 7:
                    cout << "Exiting from the program...\n";
                    exit(0);  // Exit the program
                default:
                    cout << "Invalid choice! Please enter a valid option between 1 and 8.\n";
                    break;
            }
        } while (true);
    } else {
        cout << "Invalid login credentials. Access denied.\n";
    }
}


void attendeeMenu() {
    int attenChoice;
    Attendee E;
    bool exitAttendeeMenu = false;
    int attenChoice2;
    do {
        printCentered("========================");
        printCentered("|     1. REGISTER       |");
```

```
        printCentered("|     2. LOGIN          |");
        printCentered("|     3. RETURN TO MENU |");
        printCentered("=======================");
        cout << "Enter your choice: ";

        // Validate the input for attenChoice
        cin >> attenChoice;

        if (cin.fail()) {
            cout << "Invalid input! Please enter a number (1, 2, or 3)." << endl;
            cin.clear(); // Clear the error flag
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest of the invalid input
            continue; // Skip the rest of the loop and prompt again
        }

        switch (attenChoice) {
            case 1:
                E.Register();
                break;
            case 2:
                if (E.login()) {
                    do {
                        printCentered("================================");
                        printCentered("| 1. VIEW EVENTS               |");
                        printCentered("| 2. APPLY FOR EVENT           |");
                        printCentered("| 3. CHECK APPLICATION STATUS  |");
                        printCentered("| 4. RETURN TO MAIN MENU       |");
                        printCentered("================================");
                        cout << "Enter your choice: ";

                        // Validate the input for attenChoice2
                        cin >> attenChoice2;

                        if (cin.fail()) {
                            cout << "Invalid input! Please enter a number (1, 2, 3, or 4)." << endl;
                            cin.clear(); // Clear the error flag
                            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest of th
                            continue; // Skip the rest of the loop and prompt again
                        }

                        switch (attenChoice2) {
                            case 1:
                                E.viewEvents();
                                break;
                            case 2:
                                E.registerForEvent();
                                break;
                            case 3:
                                E.checkRegStatus();
                                break;
                            case 4:
                                cout << "Returning To Main Menu...\n";
                                exitAttendeeMenu = true;
                                break;
                            default:
                                cout << "Invalid input. Please enter a valid choice." << endl;
                        }
                    } while (!exitAttendeeMenu);
                }
                break;
            case 3:
```

```
                cout << "EXITING...\n";
                exitAttendeeMenu = true;
                break;
            default:
                cout << "Invalid input. Please enter a valid choice." << endl;
        }
    } while (!exitAttendeeMenu);
}


int main() {
    int mainChoice;
    cout << left;
    printCentered("|-------------------------------------|");
    printCentered("|        EVENT MANAGEMENT SYSTEM      |");
    printCentered("|-------------------------------------|");
    int width = getConsoleWidth();
    for (int i = 0; i < width; i++) cout << "_";
    cout << endl;

    do {
        printCentered("|              MAIN MENU              |");
        for (int i = 0; i < width; i++) cout << "_";
        cout << endl;
        printCentered("===========================");
        printCentered("|       1. ORGANIZER       |");
        printCentered("|       2. ATTENDEE        |");
        printCentered("|       2. EXIT            |");
        printCentered("===========================");
        cout << "Enter your choice: ";

        // Check if the input is a valid integer
        cin >> mainChoice;

        // Validate if the user entered a valid integer
        if (cin.fail()) {
            cout << "Invalid input! Please enter a number (1, 2, or 3)." << endl;

            // Clear the error flag
            cin.clear();
            // Ignore the invalid input
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

        } else {
            // Handle the valid input
            switch (mainChoice) {
                case 1:
                    adminMenu();
                    break;
                case 2:
                    attendeeMenu();
                    break;
                case 3:
                    cout << "EXITING From Program..\n";
                    break;
                default:
                    cout << "Invalid choice. Please enter 1, 2, or 3.\n";
            }
        }

    } while (mainChoice != 3);
```


977:
978: return 0;
979: }
980:

OUTPUT SCREEN


C:\Users\Dell\Desktop\sardar\EEEEEEEEE1.exe
MAIN MENU
1. ORGANIZER
2. ATTENDEE
Enter your choice: 1
****LOGIN****
Enter password:


Enter Num Max Particpants: 20
Enter Event ticket Price: 20
Event Updated successfull.
Data saved successfully!
1. CREATE EVENTS
2. VIEW EVENTS
3. UPDATE EVENTS
4. DELETE AN EVENT
5. VIEW REGISTRATIONS
6. RETURN TO MAIN MENU
7. EXIT
Enter your choice: 2
1
Event ID : 1113
Event Name : EXPO
Event Date is : 12/5/2025
Event Location : IIUI CS DEPARTMENT
Event Timing : 3:00PM
Event Capacity : 200 People
Ticket Price : $0
2
Event ID : 1212
Event Name : Hiking
Event Date is : 23/4/2025
Event Location : Trale 3
Event Timing : 9:00
Event Capacity : 20 People
Ticket Price : $20
1. CREATE EVENTS
2. VIEW EVENTS
3. UPDATE EVENTS
4. DELETE AN EVENT
5. VIEW REGISTRATIONS
6. RETURN TO MAIN MENU
7. EXIT
Enter your choice: 5
Application # 1
Event Id : 1113
User Id : 43203
User Name : HALAR_01
User gmail : Halar@gmail.com
User phone NO: +923063400970
****Appliction Approved****
Data saved successfully!


1. CREATE EVENTS
2. VIEW EVENTS
3. UPDATE EVENTS
4. DELETE AN EVENT
5. VIEW REGISTRATIONS
6. RETURN TO MAIN MENU
7. EXIT
Enter your choice: 6
Returning to main menu...
MAIN MENU
1. ORGANIZER
2. ATTENDEE
2. EXIT
Enter your choice: 1
****LOGIN****
Enter password: admin123
:) Login successful!
1. CREATE EVENTS
2. VIEW EVENTS
3. UPDATE EVENTS
4. DELETE AN EVENT
5. VIEW REGISTRATIONS
6. RETURN TO MAIN MENU
7. EXIT
Enter your choice: 6
Returning to main menu...
MAIN MENU
1. ORGANIZER
2. ATTENDEE
2. EXIT
Enter your choice: 2
1. REGISTER
2. LOGIN
3. RETURN TO MENU
Enter your choice: 2
****LOGIN****
Enter Username:HALAR_01
Enter Password:khan.halar
Login successful! Welcome, HALAR_01!
Enter Username:HALAR_01
Enter Password:khan.halar
Login successful! Welcome, HALAR_01!
1. VIEW EVENTS
2. APPLY FOR EVENT
3. CHECK APPLICATION STATUS
4. RETURN TO MAIN MENU
Enter your choice: 2
1
Event ID : 1113
Event Name : EXPO
Event Date is : 12/5/2025
Event Location : IIUI CS DEPARTMENT
Event Timing : 3:00PM
Event Capacity : 200 People
Ticket Price : $0
2
Event ID : 1212
Event Name : Hiking
Event Date is : 23/4/2025
Event Location : Trale 3
Event Timing : 9:00
Event Capacity : 20 People
Ticket Price : $20
Enter Event Id to Register:1212
Event Exsists..
Enter your Id: 43203
Enter Gmail: halar000@gmail.com
Enter Your Contact No: +923063400970
Do you want to register for another Event: n
Data saved successfully!
1. VIEW EVENTS
2. APPLY FOR EVENT
3. CHECK APPLICATION STATUS
4. RETURN TO MAIN MENU
Enter your choice: 3
Event Id : 1113
User Id : 43203
User Name : HALAR_01
User gmail : Halar@gmail.com
User phone NO: +923063400970
****Appliction Approved****
Event Id : 1212
User Id : 43203


Event Id : 1212
User Id : 43203
User Name : HALAR_01
User gmail : halar0000@gmail.com
User phone NO: +923063400970
Appliction Pending..
1. VIEW EVENTS
2. APPLY FOR EVENT
3. CHECK APPLICATION STATUS
4. RETURN TO MAIN MENU
Enter your choice: 4
Returning To Main Menu...
MAIN MENU
1. ORGANIZER
2. ATTENDEE
2. EXIT
Enter your choice: 1
****LOGIN****
Enter password: admin123
:) Login successful!
1. CREATE EVENTS
2. VIEW EVENTS
3. UPDATE EVENTS
4. DELETE AN EVENT
5. VIEW REGISTRATIONS
6. RETURN TO MAIN MENU
7. EXIT
Enter your choice: 5
Application # 1
Event Id : 1113
User Id : 43203
User Name : HALAR_01
User gmail : Halar@gmail.com
User phone NO: +923063400970
****Appliction Approved****


Application # 2
Event Id : 1212
User Id : 43203
User Name : HALAR_01
User gmail : halar0000@gmail.com
User phone NO: +923063400970
Appliction Pending..
APPROVE APPLICATIONS
Application # 2
Event Id : 1212
User Id : 43203
User Name : HALAR_01
User gmail : halar0000@gmail.com
User phone NO: +923063400970
Appliction Pending..
Enter[Y/y] to approve Application: y
Data saved successfully!
1. CREATE EVENTS
2. VIEW EVENTS
3. UPDATE EVENTS
4. DELETE AN EVENT
5. VIEW REGISTRATIONS
6. RETURN TO MAIN MENU
7. EXIT
Enter your choice: 4
Enter Event ID to Delete: 1212
Deleting Event.... Event Deleted successfull.Data saved successfully!
Enter your choice: 2
1
Event ID : 1113
Event Name : EXPO
Event Date is : 12/5/2025
Event Location : IIUI CS DEPARTMENT
Event Timing : 3:00PM
Event Capacity : 200 People
Ticket Price : $0

REPORT

Project Report

Event Management System

Introduction

The Event Management System project was developed as a semester assignment to enhance the process of organizing and participating in events. This system was implemented using Object-Oriented Programming principles in C++ and enables efficient management of events and registrations. The team, named 'WITH OOP,' consists of Wahab Ejaz, Muhammad Irfan, Talal Gul, and Halar Khan. The purpose of the project was to apply theoretical concepts to build a real-world application that supports multiple roles, including administrators and attendees.

Methodology

The development process involved several stages:

1. **Requirement Analysis:** Understanding the functionalities required for event management.
2. **Design:** Structuring the program with classes to encapsulate different functionalities.
3. **Implementation:** Writing and testing C++ code with a focus on object-oriented design.
4. **Testing:** Performing manual testing for scenarios such as event creation, user registration, and data storage.
5. **Finalization:** Refining code and addressing edge cases.

Implementation

Classes:

- **Event:** Handles details like ID, name, date, location, timing, maximum participants, and ticket price.
- **Registration:** Manages user registration details such as ID, username, email, phone number, and linked event ID.
- **User (Base Class):** Provides login functionality and a mechanism to view all events. It has two derived classes:
    - **Admin:** Allows for event management tasks like creating, updating, deleting, and viewing registrations.

- **Attendee:** Supports functionalities for user registration, applying for events, and checking the application status.

Key Functions:

- **login()** (Admin and Attendee): Handles authentication. Admin credentials are validated against a stored password, while attendee credentials are checked from a file.
- **createEvents()** (Admin): Creates events and saves them to a binary file.
- **viewEvents()** (Admin, Attendee): Displays event details stored in the system.
- **updateEvents()** (Admin): Updates details of an existing event.
- **deleteEvents()** (Admin): Deletes an event from the system.
- **viewRegistrations()** (Admin): Shows all user registrations.
- **Register()** (Attendee): Enables new user registration.
- **registerForEvent()** (Attendee): Allows users to register for specific events.
- **checkRegStatus()** (Attendee): Checks the user's registration status.

Utility Functions:

- **Validation:** Ensures proper input for names, dates, emails, and phone numbers.
- **Persistent Storage:** Uses binary files (Events.DAT, applications.dat) to store data efficiently.

Results and Analysis

The project met its goals effectively by implementing:

- A functional event management system with binary file persistence.
- Role-specific functionalities, such as event management for administrators and event application for attendees.
- Intuitive user interaction through a menu-driven console interface.

Performance testing verified robust operations for event creation, updates, deletions, and data retrieval. Data storage using binary files ensured efficiency and integrity, with smooth handling of invalid inputs.

Challenges and Solutions

- **File Management:**

- Challenge: Maintaining data persistence across program runs.
- Solution: Binary file operations to read and write events and registration details.

- **Input Validation:**
  - Challenge: Ensuring data consistency and correctness.
  - Solution: Custom validation functions for critical inputs like email and dates.
- **Dynamic Memory Management:**
  - Challenge: Avoiding memory leaks due to dynamic allocations.
  - Solution: Utility functions (clearEventArray, clearRegistrationArray) to deallocate memory safely.

Conclusion

The Event Management System successfully demonstrated the practical implementation of object-oriented programming concepts. It supports essential functionalities for both administrators and attendees, fulfilling the project objectives. Future improvements could include adding GUI support, payment integration, and real-time notifications to provide an even better user experience. Overall, the project provided a solid foundation in software design and development practices.

REFERENCES

Our Senior (GPT)

- Provided guidance, suggestions, and explanations throughout the project development process. Assisted in understanding key OOP concepts and improving code structure.

Robert Lafore – *Object-Oriented Programming in C++*

- Served as a foundational resource for understanding Object-Oriented Programming principles, syntax, and best practices in C++.

Dr. Muhammad Nadeem*(OOP Teacher, International Islamic University Islamabad)*

- Played a crucial role in building a strong foundation of Object-Oriented Programming concepts throughout the semester, enabling effective implementation in the project.

GitHub Copilot

- Assisted in generating code snippets, debugging, and offering coding suggestions to optimize the program efficiently.

Sohail Iqbal

- Helped in testing the project, identifying minor bugs, and providing valuable feedback for improvements.

*****THE END*****