A MINOR PROJECT REPORT ON

EVENT REGISTRATION & VERIFICATION USING QR CODE

Submitted in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS
To
Guru Gobind Singh Indraprastha University, Delhi

Under the Guidance of:
Dr. Brahampal Singh

Submitted by:
Sharad Jain
BCA-V Sem
04020602018

Session 2018 – 2021

TRINITY INSTITUTE OF PROFESSIONAL STUDIES

To Whom It May Concern

I **Sharad Jain**, Enrolment No. **04020602018** from BCA-V Sem of the Trinity Institute of Professional Studies, Delhi hereby declare that the Minor Project Report entitled **"Event Registration & Verification Using QR Code"** at **Trinity Institute Of Professional Studies** is an original work and the same has not been submitted to any other Institute for the award of any other degree.

Date: Signature of the Student

Certified that the Project Report submitted in partial fulfilment of Bachelor of Computer Applications (BCA) to be awarded by G.G.S.I.P. University, Delhi by **Sharad Jain**, Enrolment No. **04020602018** has been completed under my guidance and is Satisfactory.

Date: Signature of the Guide

                                                                         Name of the Guide:
                                                                         Dr. Brahampal Singh
                                                                         Designation: Associate Professor

ii

Acknowledgement

Abstract

Every institution organizes events and meetings. It is a major task to organize participants, manage reports, registrations, messages, make groups, collect fees etc. There are multiple societies and clubs in every institution and all of them organize a variety of events throughout the year, sometimes at the same time. Their publicity messages and posts are often ignored or lost in the sea. Further, the hassle of making groups or participants, arranging meetings to get payment, circulating messages for any changes, reminding participants etc., often leads to chaos and confusion.

We offer a solution which would prevent the struggle of managing participants and reducing the work force required, making it possible to focus their attention on other matters.

The purpose of this application is to automate existing manual system with the help of computerized equipment and full-fledged software.

This report discusses the project developed and implemented by us along with the challenges we faced during the development period of the project.

Table of Contents

Table of Figures

List of Tables

List of Abbreviations

S. No.	Abbreviated Name	Full Name
1.	ERVQR	Event Registration and Verification using QR (Project Title)
2.	QR	Quick Response
3.	TIPS	Trinity Institute of Professional Studies
4.	SRS	Software Requirement Specification
5.	ID	Identification
6.	DFD	Data Flow Diagram
7.	ER	Entity Relationship
8.	TCs	Test Cases
9.	Regex	Regular expression

1. Introduction

Every organization, big or small, organizes events and has challenges to overcome, people, and operations to manage smoothly. From attendees to the location to the ambiance, there are many different areas in need of careful management. This application is being developed to override the problems prevailing the present manual systems. This system is being designed, keeping in mind the event requirements. This will reduce the struggle of maintaining groups, records of participants and payments, etc. Moreover, it will provide all the features with a user-friendly interface. The collection will be understandable and straightforward.

We offer a solution which would prevent the struggle of managing participants and reducing the work force required, making it possible to focus their attention on other matters. The purpose of this application is to automate existing manual system with the help of computerized equipment and full-fledged software.

This report discusses the project developed and implemented by us along with the challenges we faced during the development period of the project.

1.1 Idea and Purpose

We suggest an event particular app which will act as a common platform for serving different societies and clubs of an organization for easy and effective event management. This application will bridge the gap between the organizers and the attendees. The application will reduce the manual work significantly and allow easy management of participation.

1.2 Problem Definition

During our fest, workshop, seminars, society events it was observed that there were lot of coordination problems and everything was being handled manually. There was no proper system to handle the incoming participation, event information management and report management.

Also, the manual handling of events led to a lot of disturbance in the classrooms as the students would request to leave the class for organising events, making announcements, designing posters, attending meetings etc. This would also disturb the faculty and thus they would not support the missing of the classes. Even the society coordinators would find it difficult to manage multiple events, participation.

1.3 Existing Methods

Presently, the following methods are used to manage publicity and participations:

1. Flooding messages on WhatsApp and making groups of participants on WhatsApp to circulate messages require recording contact numbers of everyone and making sure that they read the previous messages.
2. Repetitive announcements in classrooms – often not possible to make in all classes.
3. Posters on notice boards – cluttering noticeboards with multiple posters, important notices go unnoticed.
4. Arranging meetings with the organizers to give payments – unreliable and causes confusion as people are often in different places.

1.4 Motivation

This system application's motivation arose due to the current manual system problems, some of which are:

1. Very narrow span of attention towards long messages on WhatsApp.
2. Infrequent announcements disturb the environment of classes.
3. Wastage of resources in the form of paper and personnel.
4. Very time-consuming.
5. Difficulty in steady coordination and scheduling of different activities.

As participants and organizers, we feel that these problems cause confusion and a bad reputation.

1.5 Suggested Solutions

We suggest the following solutions to the above-discussed problems:

1. A common platform for organizers to handle records of the event.
2. Auto-ticket generation (QR).
3. A central database for storage of records.

2. System Requirement Analysis

2.1 Project Plan

The project management plan has been broken down in the following parts:

2.1.1. Stakeholders and Expectations:

Technical Team: Have ready access to individuals with the authority to make decisions regarding events.

Project Manager: Have an application in the form of a desktop application.

Client: Gain an application which event organizers can use easily and enhance their customer's experience.

2.1.2. Project Priorities and Degrees of Freedom:

The main focus of the project is to develop an event management system on time and within budget. Further we aim to conduct a comprehensive testing of the system to detect bugs and defects. Another priority area is proper allocation of resources and team members to the various roles and responsibilities that arise during the software development.

2.1.3. Approach

We will be following an Incremental approach for this project. The first iteration will focus on basic functionality of the application and subsequent iterations will depend upon that and incorporate more features as time allows based on their priority and importance to the whole application.

2.1.4. Assumptions

- Server terminals are available and functional when needed.
- Feasibility of interfacing the ERVQR application with the other applications.
- The professional societies and clubs will adopt this desktop application meant for smooth running of their events.

2.1.5. Success Criteria

The project will be considered a success if the team delivers an operational prototype at the end of the semester with the earlier mentioned features.

2.1.6. Risks and Obstacles to Success

The risks and hurdles that might occur during the development of the portal and operations of the application include-net speed, network connectivity issues (because our system depends on a reliable and fast connected network to operate), a secure and trusted medium of communication between users and societies and other risks including improper use of the server, power failures etc.

2.1.7. Scope

This project will consist of creating an application of event management based upon the cultural, technical societies and student chapters of different college/universities. The project will:

- Allow students to register for events and societies.
- Allow societies to create and manage events and their registrations.
- Allow students to enter an event with just the use of QR code.
- Allow administrator to manage societies.

2.2 Software Requirement Specifications

Software Requirement Specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide. It establishes the basis for an agreement between customers and the software providers on what the software product is to do and what it is not expected to do so that there is no room for confusion in the future. If used appropriately, SRS can help prevent software project failure.

2.2.1. Functional Requirements

Our proposed system has the following requirements:

1. The system requires storing the information about a new participant being registered and, most notably, the event and its organizers.

2. The system needs to help the internal staff manage the entries of the database and keep information on activities.

3. The system needs to update, delete, and modify the records.

4. The system needs to maintain quality and quantity records.

5. The system requires the verification and authentication process of users.

6. The system needs to provide a QR code as identification to each participant at the time of registration.

7. The system should be able to provide as easy to read record of all participants and events.

8. Every participant should be able to enter an event just by showing the QR code that was provided at the time of registration.

9. System needs to maintain security to prevent unauthorized modification of data.

2.2.2. Non-Functional Requirements

<u>Performance Requirements</u>

- The system needs to be reliable.
- If unable to process the request then appropriate error message.
- Windows are loaded within few seconds.

<u>Security Requirements</u>

- After entering the password and user-id the user can access his profile.
- The details of user must be safe and secure.
- Sharing of details.
- Unauthorized modification of records must be prevented.

<u>Safety Requirements</u>

- The details need to be maintained properly.
- Users must be authenticated.

2.3 Tools and Technologies

The following tools and technologies are expected to be used in development. Further may be added as the operations are implemented.

2.3.1. Development Language

2.3.1.1.   Python 3.8

Python is a general-purpose programming language that can be used on any modern computer operating system. It can be used for processing text, numbers, images, scientific data and just about anything else you might save on a computer. It is used daily in the operations of the Google search engine, the video-sharing website YouTube, NASA and the New York Stock Exchange. These are but a few of the places where Python plays important roles in the success of the business, government, and non-profit organizations; there are many others.

2.3.1.2.   Pip 20.2.4

pip is a de facto standard package-management system used to install and manage software packages written in Python. Many packages can be found in the default source for packages and their dependencies — Python Package Index. Most distributions of Python come with pip preinstalled.

2.3.2. Development Platforms

2.3.2.1.   PyCharm 2020.20

PyCharm is an Integrated Development Environment (IDE) used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django.

2.3.2.2.   Atom 1.52.0 x64

Atom is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in Node.js, and embedded Git Control, developed by GitHub. Atom is a desktop application built using web technologies. Most of the extending packages have free software licenses and are community-built and maintained. Atom is based on Electron (formerly known as Atom Shell), a framework that enables cross-platform desktop applications using Chromium and Node.js. It is written in CoffeeScript and Less.

2.3.2.3.   MySQL Workbench 8 CE

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modelling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.

2.3.3. Database

2.3.3.1. *MySQL 8.0.19*

MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL). SQL is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use. MySQL is an essential part of almost every open source PHP application. Good examples for PHP & MySQL-based scripts are WordPress, Joomla! and Drupal.

2.3.3.2. *Excel*

Microsoft Excel is a spreadsheet developed by Microsoft for Windows, macOS, Android and iOS. It features calculation, graphing tools, pivot tables, and a macro programming language called Visual Basic for Applications. It has been a very widely applied spreadsheet for these platforms, especially since version 5 in 1993, and it has replaced Lotus 1-2-3 as the industry standard for spreadsheets. Excel forms part of the Microsoft Office suite of software.

2.3.4. Hosting Service

2.3.4.1. *PythonAnywhere*

PythonAnywhere is an online integrated development environment (IDE) and web hosting service based on the Python programming language. It provides in-browser access to server-based Python and Bash command-line interfaces, along with a code editor with syntax highlighting. Program files can be transferred to and from the service using the user's browser. Web applications hosted by the service can be written using any WSGI-based application framework.

2.3.5. Libraries

Library	Version
certifi	2020.11.8
chardet	3.0.4
click	7.1.2
et-xmlfile	1.0.1
Flask	1.1.2
Flask-Cors	3.0.9
idna	2.10
itsdangerous	1.1.0
jdcal	1.4.1
Jinja2	2.11.2
MarkupSafe	1.1.1
mysql-connector-python	8.0.22
numpy	1.19.3
opencv-contrib-python	4.4.0.46
opencv-python	4.4.0.46
openpyxl	3.0.5
Pillow	8.0.1
protobuf	3.13.0
PyQRCode	1.2.1
pyzbar	0.1.8
requests	2.24.0
six	1.15.0
urllib3	1.25.11
Werkzeug	1.0.1
Tkinter	8.6
re	2.2.1
hashlib	20081119
datetime	4.3

Table 1: List of Libraries

2.4 Hardware Requirements

The following hardware requirements are recommended to be fulfilled in order to run this software.

1. CPU: Intel core i3 3rd Generation / AMD FX-6100
2. RAM: 2 GB
3. GPU: Integrated Graphics
4. Storage: 1 GB
5. Camera: Any 3MP camera

2.5 Software Requirements

The following software requirements are recommended to be fulfilled in order to run this software.

1. OS: Any Operating System
2. Database: MySQL
3. Programming Language: Python

3. System Feasibility Study

3.1 Feasibility Study

An Assessment of the feasibility of the project.

3.1.1. Economic Feasibility

The project is economically feasible as it works with functions with low-cost services such as laptops and desktops.

3.1.2. Technical Feasibility

The current project is technically feasible as the application requires:

1. Any python supported IDE
2. Server-Side Services
3. GUI development tools

All these are readily available and can be successfully deployed on any available computer.

3.1.3. Behavioural Feasibility

The application is behaviourally feasible since it requires no technical guidance; all the modules are user friendly.

3.1.4. Operational Feasibility

The application is operationally feasible as:

1. Complete GUI-Base, which is user friendly.
2. Inputs to be taken are self-explanatory.
3. The system cuts down the load and cost of clients by high margins.

4. System Design

4.1 Use Case

Use Case Model

The Use Case Model describes the proposed functionality of our system. The diagram represents a discrete unit of interaction between the user (society or a new member) and the application. This interaction is a single unit of meaningful work, such as Create Event or View Event Details.

The project's use case model consists of the following type of actors:

- Main Actors – Participant

- Supporting Actors – Admin and User

The primary modules (different functions) are:

- User Registration

- Admin / User Login

- Add Participant

- Register Participant in Events

- Add / Remove Events

- Mark Entry of Participant

- Report Generation

- Quick Response (QR) Code

The model is given below:
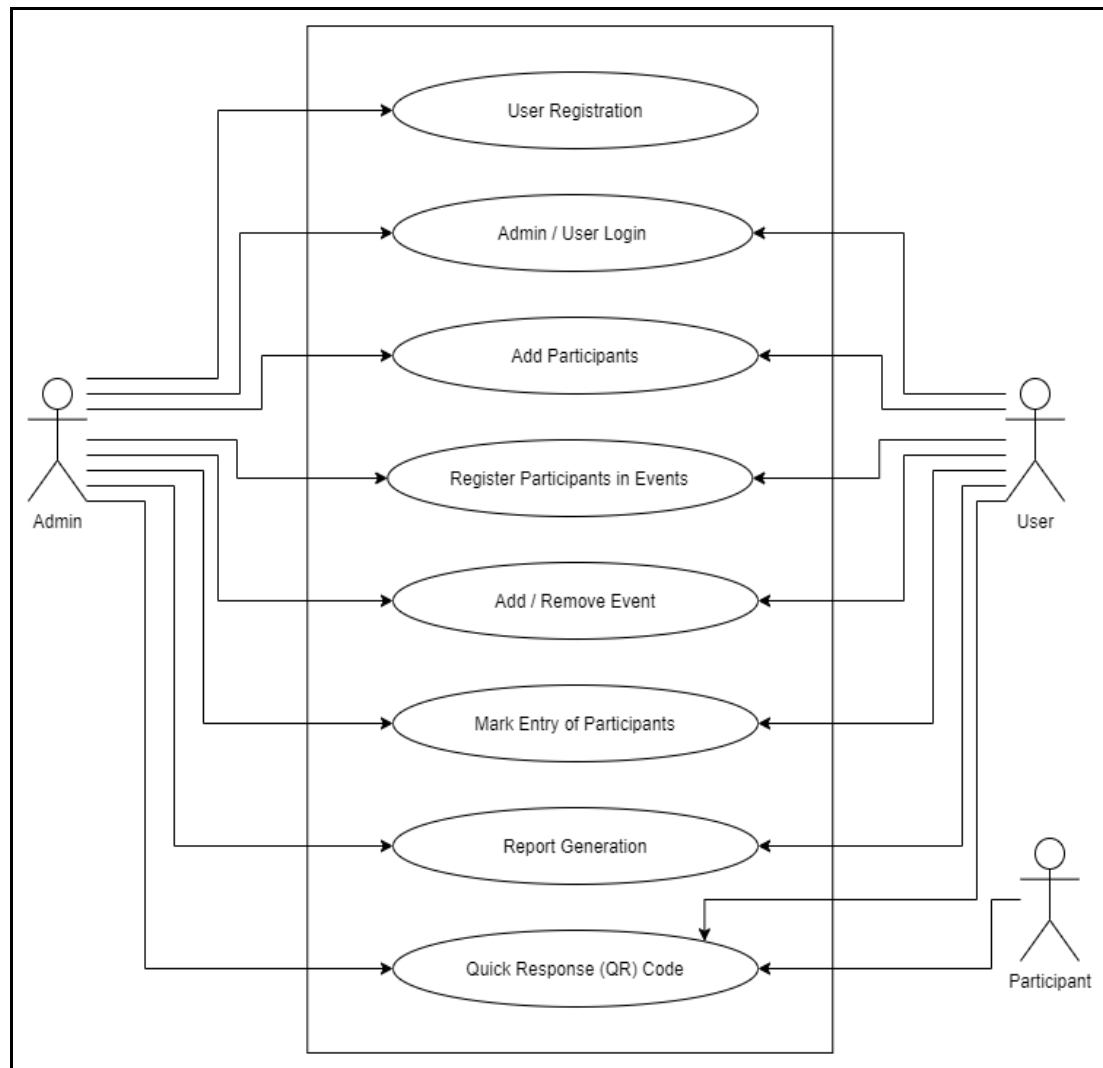
Figure 1: Use Case Diagram

The project has the following use case specifications:

4.1.1. Use Case: User Registration

Description: This use case takes care of registration of users (Organizers) in the MySQL database and gives them access to the application by signing in using their user id (email) and password.

Note: Only Administrator is allowed to add new users.

Primary Flow:

1. User information is entered by an administrator.
2. The information is sent to the database to register the user.
3. The user provides his email and password.
4. The user is authenticated.
5. The new user can access the application.

Alternate Flow at: The user is already registered.

1. The administrator is warned that the email is already registered.
2. The administrator may register another user or retry with another email id.

Error Flow E1: The user credentials are wrong.

1. The administrator is told to recheck the credentials.
2. The administrator may try again.

Error Flow E2: The server or internet is not working.

1. The administrator is told to recheck internet connection.
2. The administrator may try again.

4.1.2.   Use Case: Admin / User Login

Description: This use case takes care of login of users (Organizers), check their credentials in MySQL database and gives them access to the application by signing in using their user id (email) and password.

Primary Flow:

1. User enters their email id and password.
2. The information is sent to the database.
3. The database returns the permission level.
4. The user is authenticated.
5. The user has access to the application.

Error Flow E1: The user credentials are wrong.

1. The user is told to recheck the credentials.
2. The user may try again.

Error Flow E2: The server or internet is not working.

1. The user is told to recheck internet connection.
2. The user may try again.

4.1.3. Use Case: Add Participants

Description: This use case takes care of registration of participants (customers/clients of user) in the MySQL database and gives them a unique QR code as participant ID.

Primary Flow:
1. Participant provides his/her information and events.
2. Participant information is entered by user.
3. The information is sent to the database to register the participant.
4. The participant is given a QR code.
5. The participant is registered.

Alternate Flow 1: The Participant is already registered in all given events.
1. The user is warned that the participant is already registered.
2. The user informs the participant.

Alternate Flow 2: The Participant is already registered in any 1 of 2 given events.
1. The user is warned that the participant is already registered in one of the events and registration is complete for the other event.
2. The user informs the participant.

Error Flow E1: The participant information is wrong.
1. The user is told to recheck the information.
2. The user may try again.

Error Flow E2: The server or internet is not working.
1. The user is told to recheck internet connection.
2. The user may try again.

4.1.4. Use Case: Register Participant in Events

Description: This use case takes care of registration of already registered participants (customers/clients of user) in events in the MySQL database.

Primary Flow:

1. Participant provides QR code and events.
2. Participant information is entered by user.
3. The information is sent to the database.
4. The participant is registered in events provided.

Alternate Flow 1: The Participant is already registered in all given events.

1. The user is warned that the participant is already registered.
2. The user informs the participant.

Alternate Flow 2: The Participant is already registered in any 1 of 2 given events.

1. The user is warned that the participant is already registered in one of the events and registration is complete for the other event.
2. The user informs the participant.

Error Flow E1: The participant QR code is wrong.

1. The user is told to recheck the QR code.
2. The user may try again.

Error Flow E2: The server or internet is not working.

1. The user is told to recheck internet connection.
2. The user may try again.

4.1.5. Use Case: Add / Remove Events

Description: This use case takes care of adding and removing events from the MySQL database.

Primary Flow:

1. User enters event information.
2. The information is sent to the database.
3. Event is added in database.

Alternate Flow 1: The event is already registered.

1. The user is warned that the event is already registered.
2. The user may try again.

Error Flow E1: The event information is wrong.

1. The user is told to recheck the event information.
2. The user may try again.

Error Flow E2: The server or internet is not working.

1. The user is told to recheck internet connection.
2. The user may try again.

4.1.6. Use Case: Mark Entry of Participants

Description: This use case takes care of marking entry participants (customers/clients of user) in events in the MySQL database. Participant are only provided entry once in an event.

Primary Flow:

1. Participant provides QR cod.
2. Participant QR code is entered by user.
3. The information is sent to the database.
4. The participant is provided entry in the event.

Alternate Flow 1: The Participant is already registered provided entry on that QR code.

1. The user is warned that the participant has already Entered.
2. Entry is denied to the participant.

Error Flow E1: The participant QR code is wrong.

1. The user is told to recheck the QR code.
2. The user may try again.

Error Flow E2: The server or internet is not working.

1. The user is told to recheck internet connection.
2. The user may try again.

4.1.7. Use Case: Report Generation

Description: This use case takes care of generation of report from the data in MySQL database. Report is generated for all participants along with events they are participating in.

Primary Flow:

1. User requests to make a report.
2. The request is sent to the database.
3. A report is generated and user is informed.

Error Flow E1: The server or internet is not working.

1. The user is told to recheck internet connection.
2. The user may try again.

4.1.8. Use Case: Quick Response (QR) Code

Description: This use case takes care of generation of QR code for participants (customers/clients of user) identification.

Primary Flow:

1. Participant is registered.
2. A QR code is generated for the user.
3. QR code is provided to participant.

Alternate Flow 1: Unable to register participant.

1. The user is warned that the participant is not registered.
2. No QR code is generated
3. The user informs the participant.

Error Flow E1: The server or internet is not working.

1. The user is told to recheck internet connection.
2. The user may try again.

4.2 Flow Chart

A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.

Flow Chart Components:

1. Process

Rectangle - One step in the process. The step is written inside the box. Usually, only one arrow goes out of the box.

2. Flowline (Arrowhead)
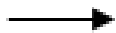
Arrow - Direction of flow from one step or decision to another.

3. Decision

Diamond - Decision based on a question. The question is written in the diamond. More than one arrow goes out of the diamond, each one showing the direction the process takes for a given answer to the question. (Often the answers are "yes" and "no.")

4. Terminal

Circle or oval - Indicates the beginning and ending of a program or sub-process. Represented as a stadium, oval or rounded (fillet) rectangle. They usually contain the word "Start" or "End", or another phrase signalling the start or end of a process, such as "submit inquiry" or "receive product".

5. Input / Output

Shows a conditional operation that determines which one of the two paths the program will take. The operation is commonly a yes/no question or true/false test. Represented as a diamond.

6. Data File or Database
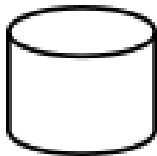
Data represented by a cylinder (disk drive).

7. Internal Storage

File represents storage of a file in internal storage.
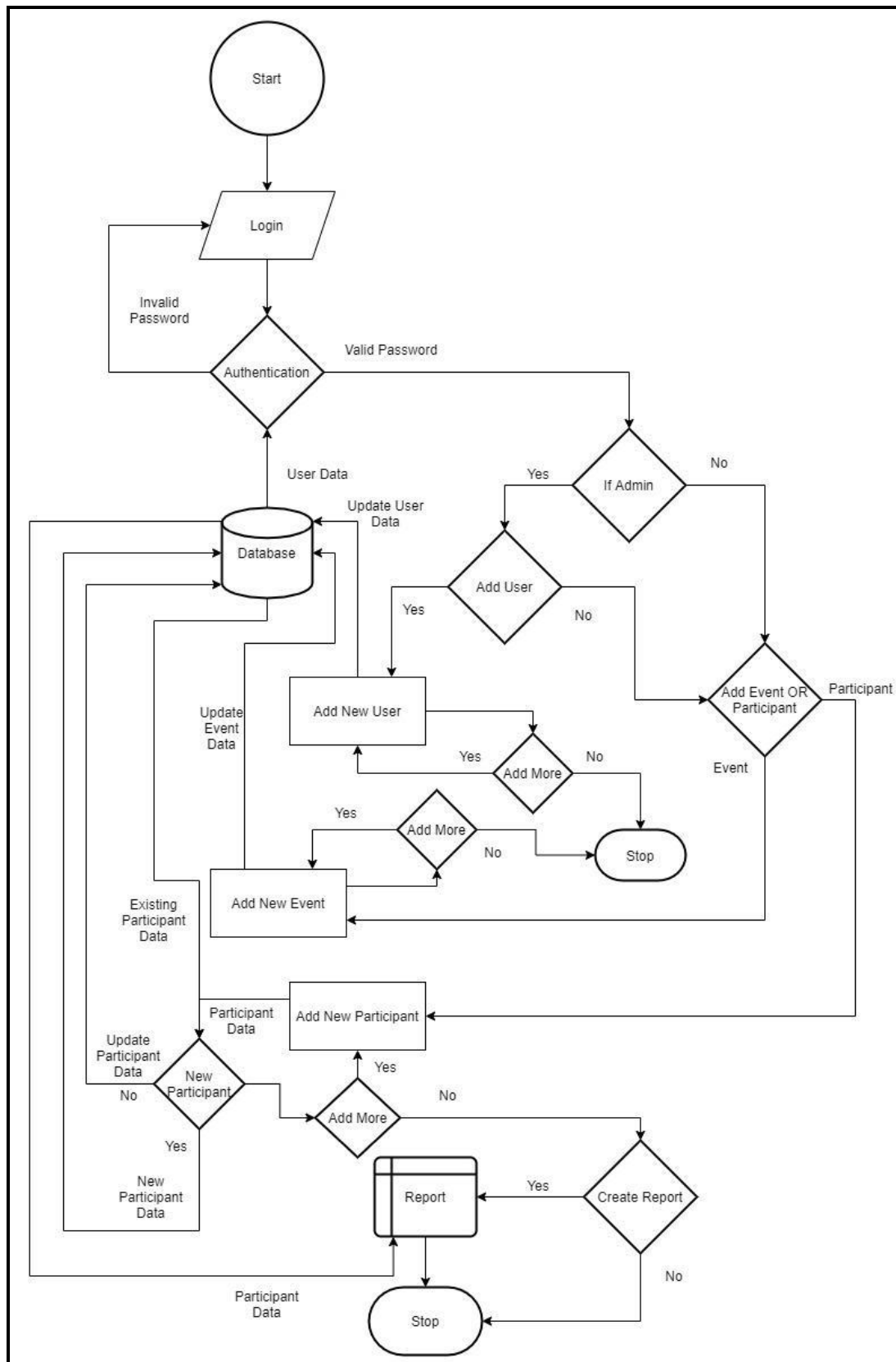
Figure 2: Flow Chart Diagram

4.3 Entity Relationship Diagram:

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let's have a look at a simple ER diagram to understand this concept.

ER Diagram Components:

1. Entity

Rectangle: An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

2. Key Attribute
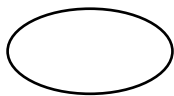
Oval: An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram.

3. Relationship
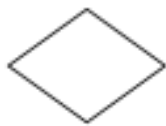
Diamond: A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities.

4. Flowline (Arrowhead)

Arrow - Direction of relational flow between objects.
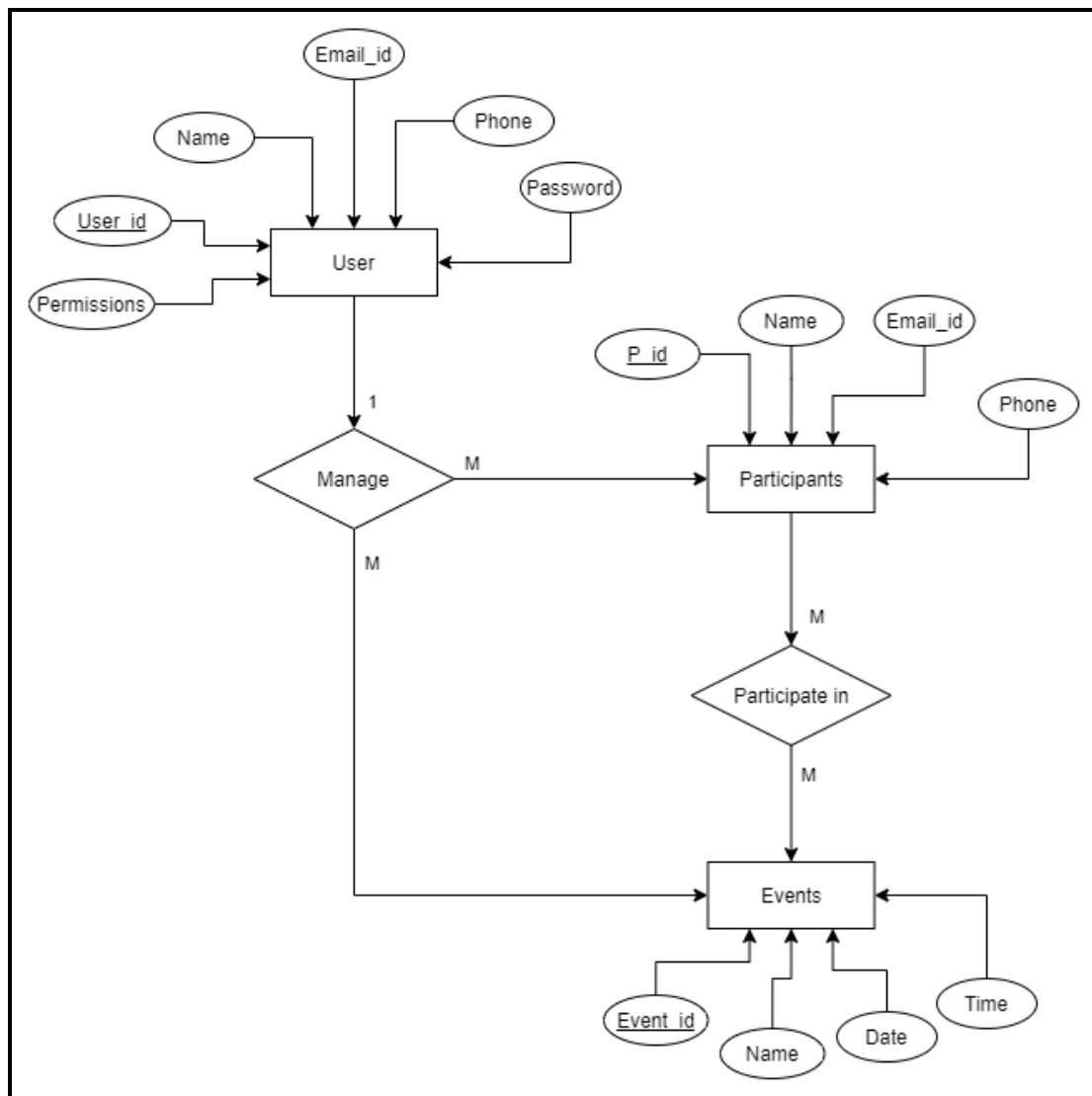
Figure 3: Entity Relationship Diagram

4.4 Database Tables Diagram:

1. User table:

This table stores data of user (Admins & Organizers) login credentials.

It contains attributes like:

- The user ID attribute

  It is the primary key of type int which cannot be left empty and hence it auto increments which contains ID of users.

- The username attribute

  It is of type varchar which cannot be left empty and can contain complex usernames of users up to a length of 50 alphanumeric values.

- The email ID attribute

  It is of type varchar which cannot be left empty and contains email ID of user to be used for verification/authentication during login.

- The phone attribute

  It is of type varchar which cannot be left empty and can contain phone number of users from any region up to 15 digits.

- The password attribute

- It is of type varchar which cannot be left empty and contains password of users in an encrypted format.

- The permission attribute

  It is of type int which cannot be left empty and contains the rights of the user to check if they are admin or normal user.

2. Participants table:

This table stores data of participants who have registered for various events.

It contains attributes like:

- The participant ID attribute
  It is the primary key of type int which cannot be left empty and hence it auto increments which contains ID of participants.
- The name attribute
  It is of type varchar which cannot be left empty and can contain names of participants up to a length of 50 alphanumeric values.
- The email ID attribute
  It is of type varchar which cannot be left empty and contains email ID of participant for later use.
- The phone number attribute
  It is of type varchar which cannot be left empty and can contain phone number of participants from any region up to 15 digits.

3. Events table:

This table stores information regarding events that will be organized in which participants can register.

It contains attributes like:

- The event ID attribute

  It is the primary key of type int which cannot be left empty and hence it auto increments which contains ID of events being organized.

- The name attribute

  It is of type varchar which cannot be left empty contains names of events being organized.

- The date attribute

  It is of type date which cannot be left empty contains the date of the event to be organized.

- The time attribute

  It is of type time which cannot be left empty contains the time of the event to be organized.

4. Registration table:

This table stores information of participant's registration in one or more events and weather they have entered in that event or not.

It contains attributes like:

- The registration ID attribute

  It is the primary key of type int which cannot be left empty and hence it auto increments which contains registration ID of relation between events and participants.

- The participant ID attribute

  It is the first foreign key obtained from participant table to link participants to their registered events.

- The event ID attribute

  It is the second foreign key obtained from events table to link participants to their registered events.

- The present attribute

It is of type int which cannot be left empty and contains the status of participant's entry into registered events.
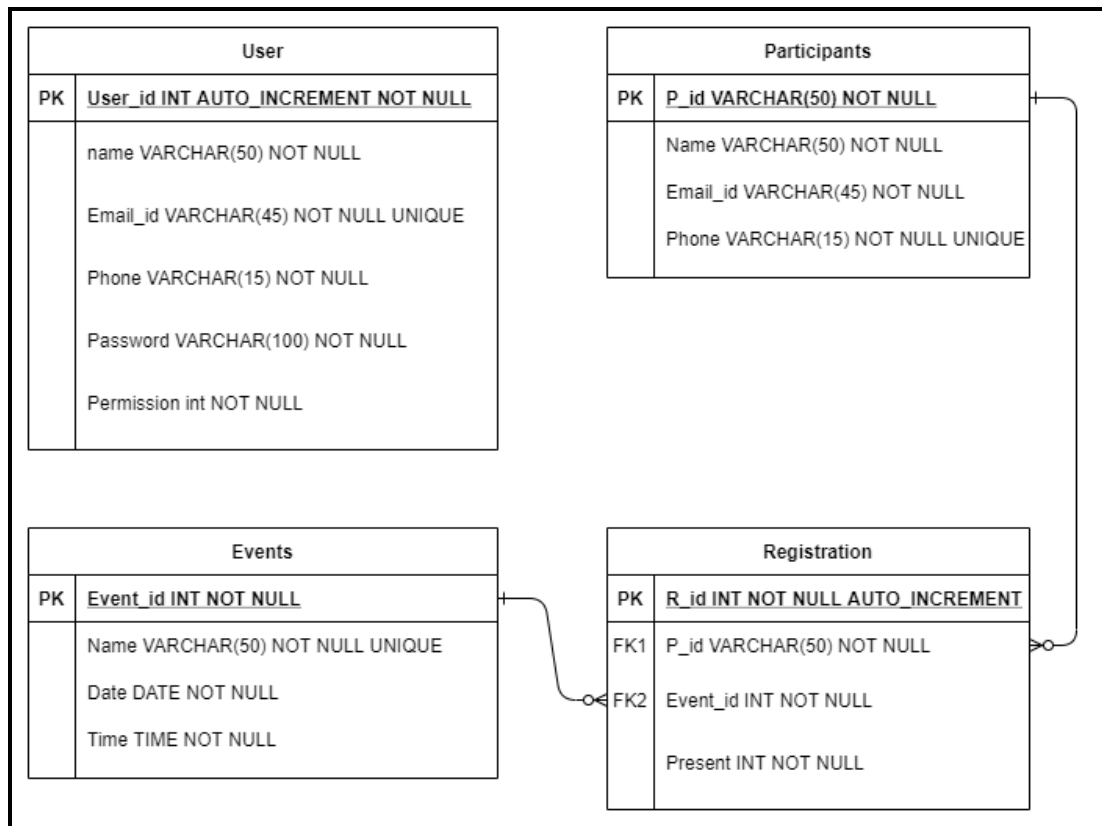
Figure 4: Entity Relationship Table Diagram

4.5 Data Flow Diagram (DFD):

4.5.1 Level 0:

Inputs:

1. User login

   Credentials entered by user to login & access the software.

2. User data

   Data provided by individuals to Admin to get login credentials.

3. Event data

   Data added by user about events being organized.

4. Participant data

   Data provided by participants to register themselves in various events.

Outputs:

1. Events

   Information about the event being organized and participants registered in it.

2. Participants

   Information about participant and the event they are registered in.

3. Entry

   Information to & after validating a participant to enter an event.
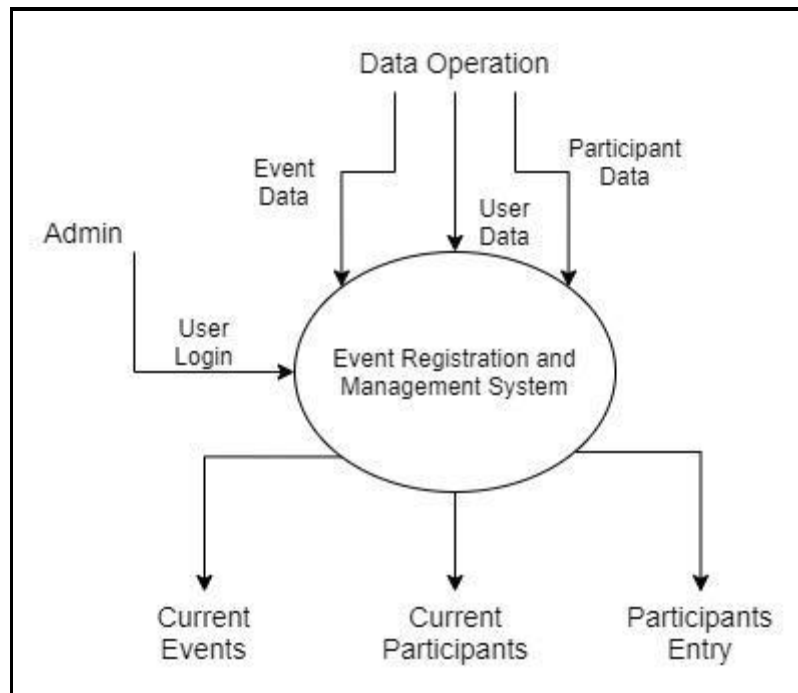
Figure 5: DFD Level 0 Diagram

4.5.2 Level 1:

1. Login

   Takes user input and verifies, validates & authenticates and logs them in as Admin or regular user.

2. Event registration / QR Generation

   The user enters the data provided by a participant to register them in any event of their choice and provides them QR code.

3. Event verification / QR Verification

   The user validates and admits a participant by the QR code provided by them during the event day.

4. Event management

   The user adds details of events to be organized during the first use so that participants can register in them.

5. Registry management

   The data of participant is linked to the event in which they have been registered to create their entrance records.

6. Report

   The user can generate a report of participant registry at any point of use which is in excel format.

7. Login as admin

   If the credentials provided by user is that of an admin, they have access to all the above abilities and to add more users of the software.
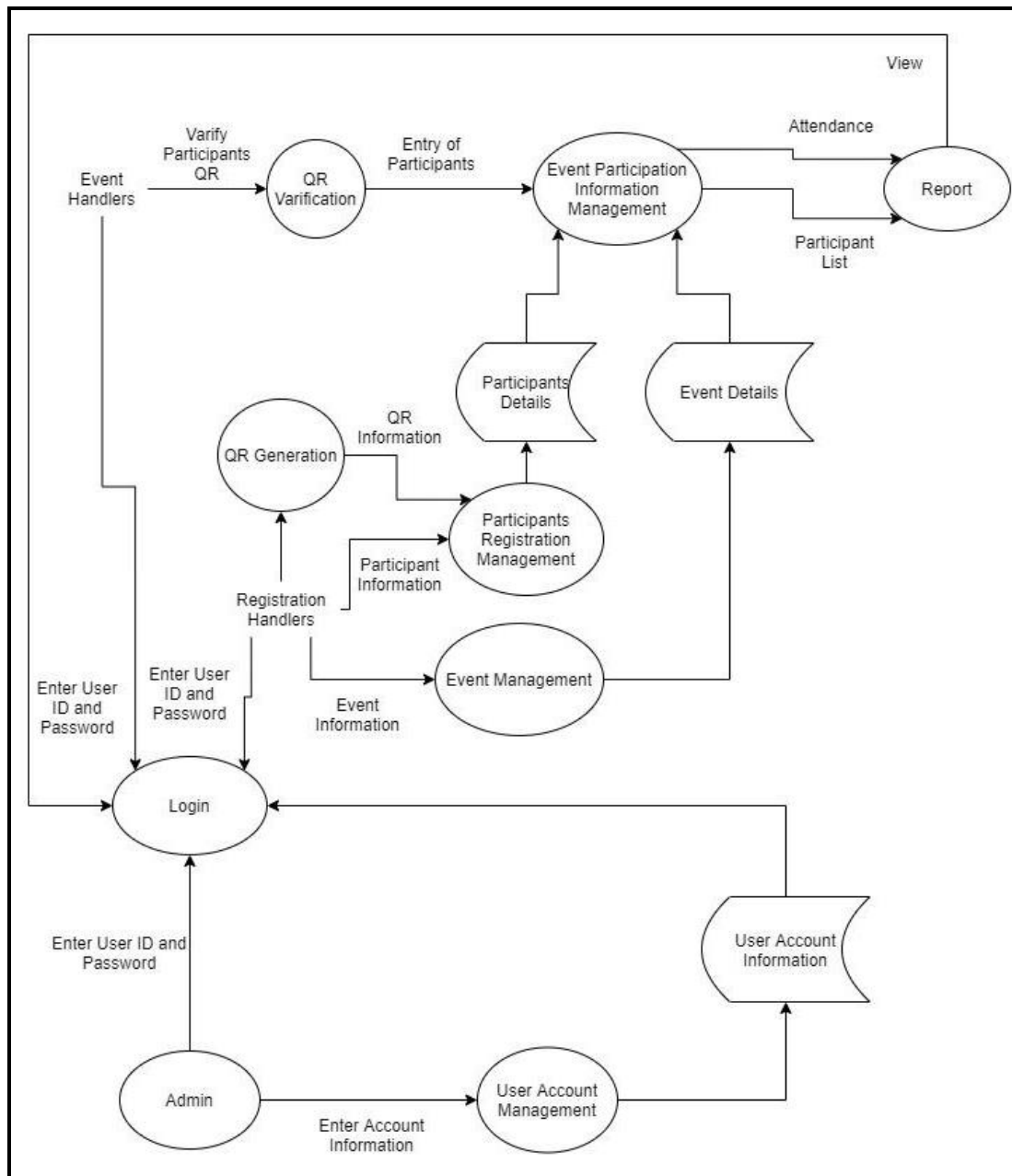
Figure 6: DFD Level 1 Diagram

4.5.3 Level 2:

4.5.3.1 User Account Information Management:

The admin manages:

- User account information.
- They can add more users, it can be more admins or handlers.
- They can update details of existing users.
- They can also remove existing users.
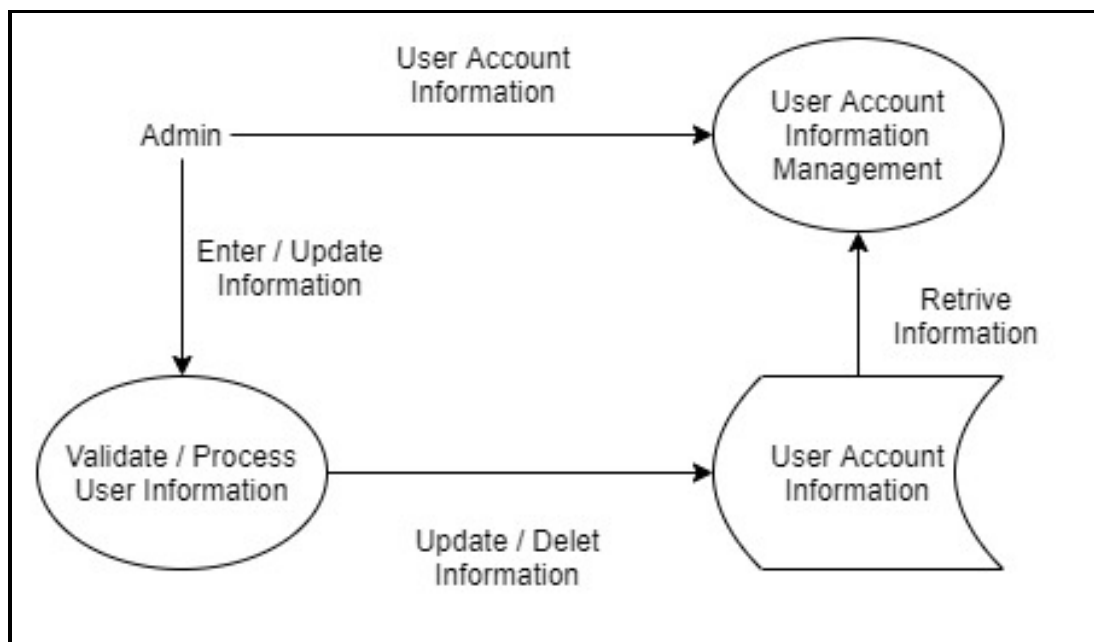
Figure 7: DFD Level 2 User Account Information Management

4.5.3.2 Login:

It validates user data against the data store to log in a user as:

- Admin.
- Event handler.
- Registration handler.
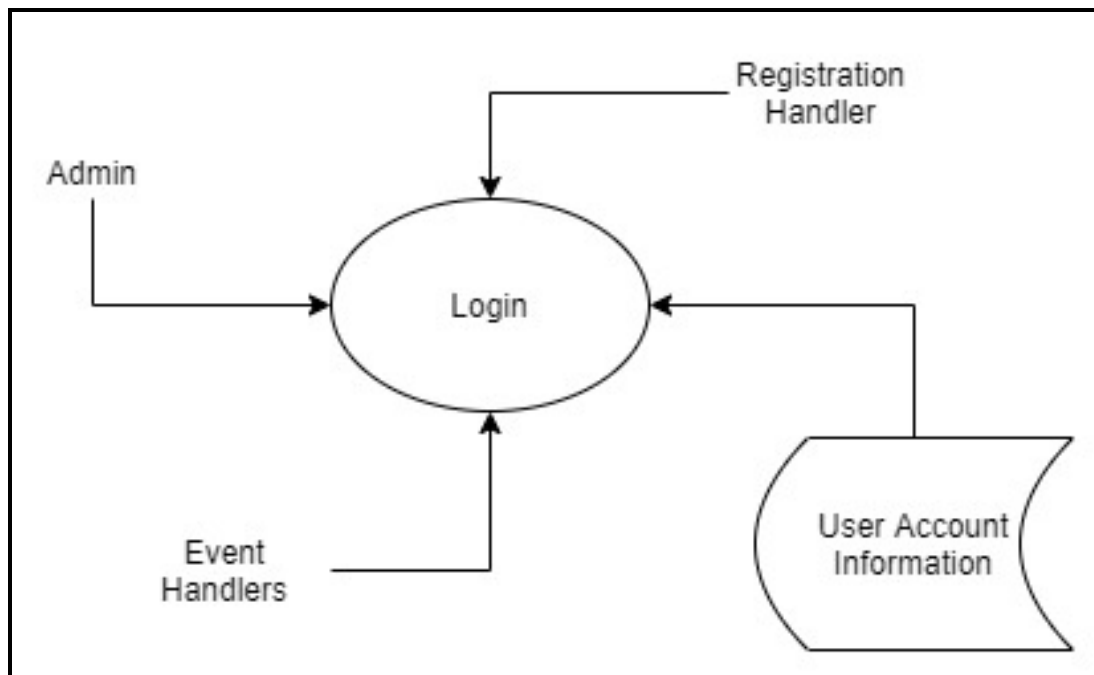
Figure 8: DFD Level 2 Login

4.5.3.3 Participant Details Management:

The registration handler manages:

- Participant data store.
- Registering participants to events.
- Providing QR code for participants.
- Re-register existing participants to more events.
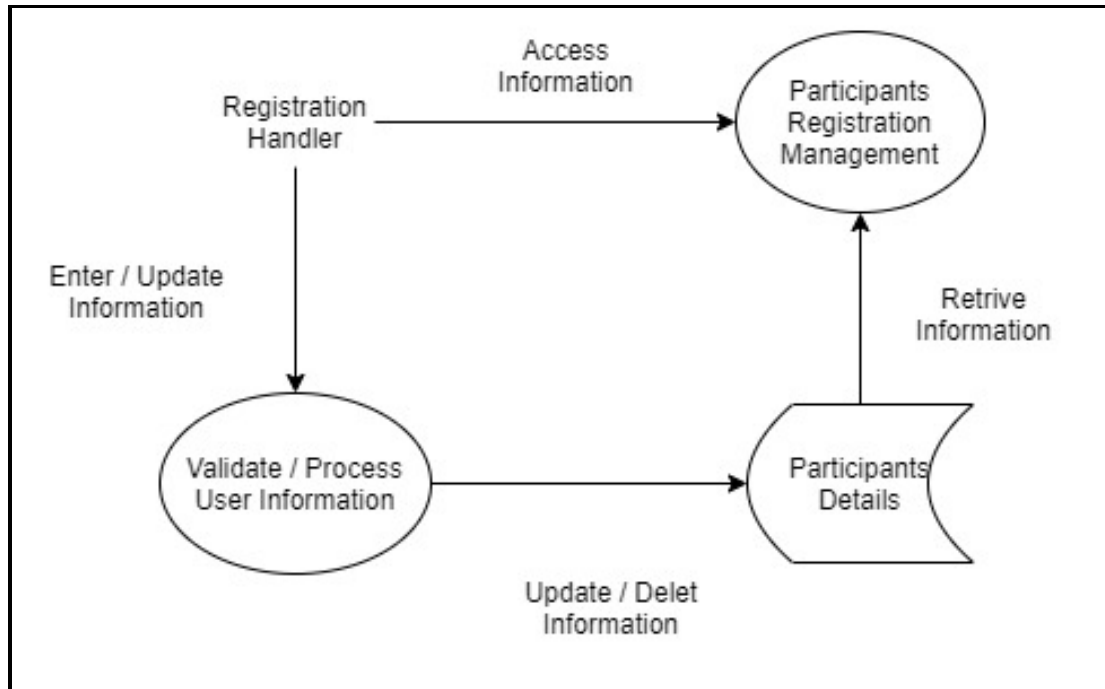
Figure 9: DFD Level 2 Participant Details Management

4.5.3.4 Event Details Management:

The registration handler manages:

- Event data store.
- Marking participant entry in an event.
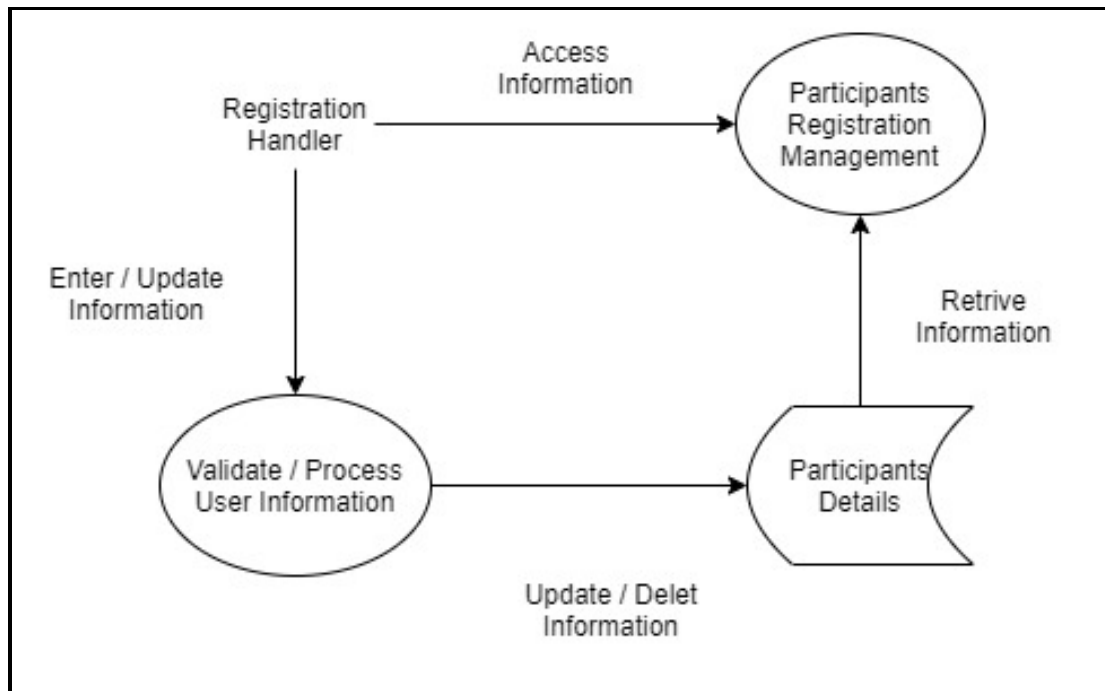
Figure 10: DFD Level 2 Event Details Management

4.5.3.5 Event Participation Information Management:

The event handler manages:

- Participant data store.
- Registering participants to events.
- Providing QR code for participants.
- Re-register existing participants to more events.
- Marking entry of participants.
- Generate Report of whole event.
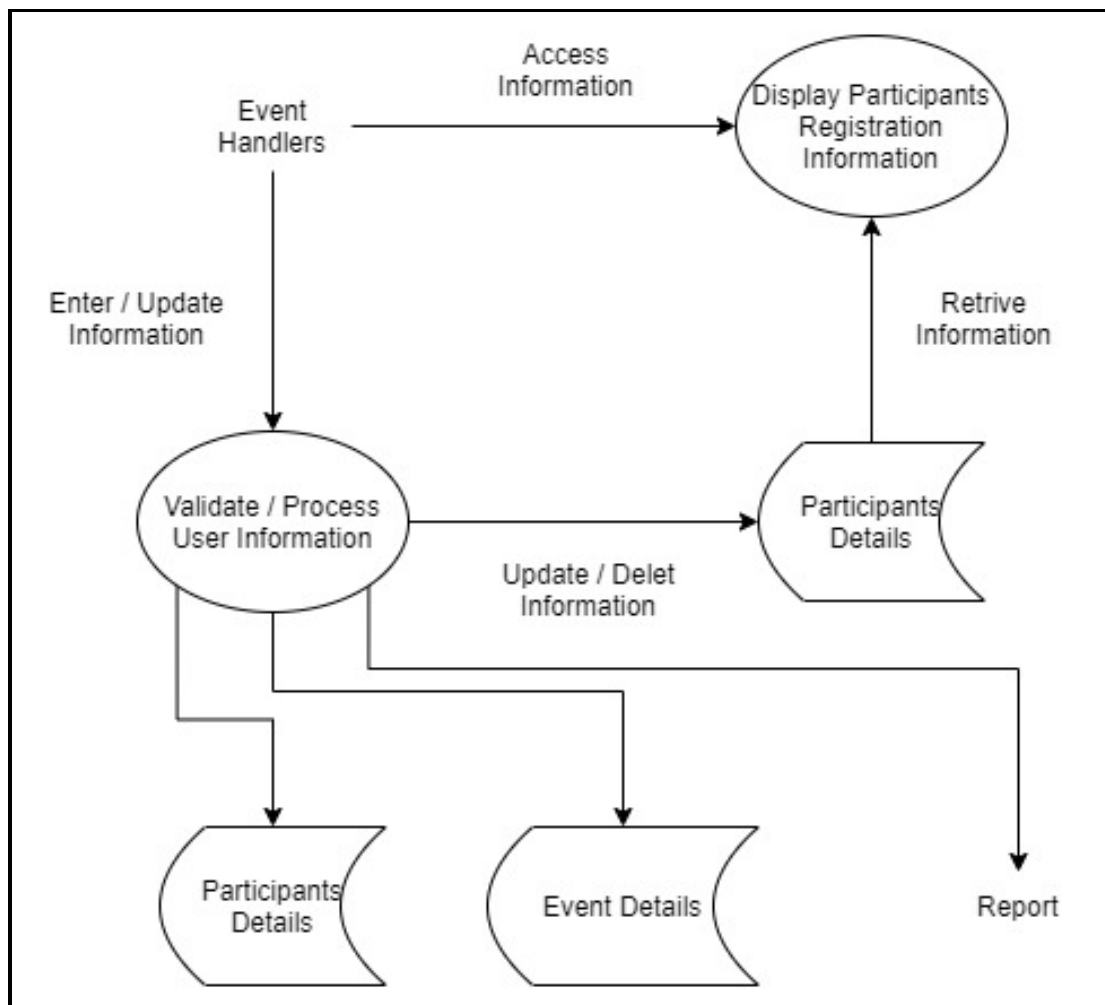
Figure 11: DFD Level 2 Event Participation Information Management

5. System Development

5.1 Frontend

5.1.1 Script "config.py"

```python
screen1 = "none"
screen1_5 = "none"
screen2 = "none"
screen3 = "none"
screen4 = "none"
screen5 = "none"


screen1geo = 0
screen1_5geo = 0
screen2geo = 0
screen3geo = 0
screen4geo = 0
screen5geo = 0
screen6geo = 0
screen7geo = 0


perm_entry = "none"
```

5.1.2 Script "ERVQR.py"

```python
# importing libraries & modules
import os
import re
import cv2
import time
import config
import random
import platform
import pyqrcode
import numpy as np
import tkinter as tk
import frontend_api as fi
import pyzbar.pyzbar as pyzbar
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import ImageTk, Image
from openpyxl import Workbook, load_workbook


# check OS to set GUI size
def chkos():
    """

    This function identifies the Operating system being used
    To set the geometry of the Windows in this app.
    """


    oname = platform.system()
```

```
if oname == "Windows":
    config.screen1geo = "430x300"
    config.screen1_5geo = "490x145"
    config.screen2geo = "500x310"
    config.screen3geo = "360x125"
    config.screen4geo = "585x458"
    config.screen5geo = "690x470"
    config.screen6geo = "420x190"
    config.screen7geo = "300x250"

elif oname == "Linux":
    config.screen1geo = "375x300"
    config.screen1_5geo = "390x145"
    config.screen2geo = "520x310"
    config.screen3geo = "320x125"
    config.screen4geo = "680x458"
    config.screen5geo = "770x460"
    config.screen6geo = "495x190"
    config.screen7geo = "360x270"

else:
    config.screen1geo = "375x300"
    config.screen1_5geo = "390x145"
    config.screen2geo = "520x310"
    config.screen3geo = "320x125"
    config.screen4geo = "250x258"
    config.screen5geo = "760x460"
    config.screen6geo = "455x190"
    config.screen7geo = "300x300"
```

```python
# code for QR scanner
def scanner():
    """

    This function is a module that manages the aspects of a QR
    Scanner using various sub modules.


    :return: it returns the data obtained from the QR Code.
    """


    # start device camera
    cap = cv2.VideoCapture(0)
    cap.set(3, 640)
    cap.set(4, 480)
    time.sleep(2)


    # find content of QR
    def decode(im: object) -> object:
        """

        This function decodes image given by camera to isolate
        QR code present in it.


        :param im: it is the image obtained from the device
        camera.
        :return: it returns the data obtained by decoding the
        image.
        """


        decodedObjects = pyzbar.decode(im)
        return decodedObjects


    # font for hull
    font = cv2.FONT_HERSHEY_SIMPLEX
```

```python
# scan the QR
def app():
    """
    This function uses the device camera to scan the QR
    shown to it &
    helps save the QR if user wants to.


    :return: it returns the data obtained by the scanner.
    """


    decodedObject = ""
    while cap.isOpened():
        ret, frame = cap.read()
        im = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        decodedObjects = decode(im)
        for decodedObject in decodedObjects:
            points = decodedObject.polygon
            if len(points) > 4:
                hull = cv2.convexHull(np.array(
                    [point for point in points],
                    dtype=np.float32))
                hull = list(map(tuple, np.squeeze(hull)))
            else:
                hull = points

            n = len(hull)
```

```python
            for j in range(0, n):
                # while scanning
                cv2.line(frame, hull[j], hull[(j+1) % n],
                        (255, 0, 0), 3)
                x = decodedObject.rect.left
                y = decodedObject.rect.top
                cv2.putText(frame, str(decodedObject.data)
                    , (x, y), font, 1, (0, 255, 255), 2,
                    cv2.LINE_AA)


        try:
            barCode = str(decodedObject.data)
            bar = barCode
        except:
            bar = ""


        cv2.imshow('frame', frame)
        key = cv2.waitKey(1)
        # binding q key as shortcut to close camera
        if key & 0xFF == ord('q') or key == 27:
            cap.release()
            cv2.destroyAllWindows()
            break

        # binding s key as shortcut to save image in front
        # of camera
        elif key & 0xFF == ord('s'):
            if (bar != "") or (len(bar) != 0):
                iname = "./scan/" + bar + ".png"
            else:
                iname = "./scan/" + str(random.randint(1,
                    101)) + ".png"
```

```python
            cv2.imwrite(iname, frame)
            messagebox.showinfo("INFO", "QR Saved")


        # check if any QR detected
        try:
            barCode = str(decodedObject.data)
            bar = barCode
            return bar
        except:
            messagebox.showerror("ALERT", "No QR Detected")
            return 0


    # start scanner
    bar = app()
    # close camera
    cap.release()
    cv2.destroyAllWindows()
    return bar


# QR Scanner code
def QRScan():
    """

    This function is a module that manages aspects of
    participant entry
    using serveral sub modules.
    """
```

```python
# call scanner
def callscan():
    """
    This function calls the QR Scanner to scan the QR of
    participants.
    """

    resp = scanner()
    if resp == 0:
        screen7.focus_force()
    else:
        qrpid.set(resp)
        screen7.focus_force()


# entry marker
def marker():
    """
    This function validates the QR provided by a
    participant and marks their entry
    by matching it against data in the server.
    """

    try:
        if (qrpid.get() != "") and (qreve.get() != ""):
            uid = qrpid.get()[2:-1]
            eve = qreve.get()
            resp = fi.mark_entry(p_id=uid, event=eve)
```

```python
            if resp == 0:
                messagebox.showerror("ALERT", "No
                    participant registered with this QR
                    ID in this event")
            elif resp == 1:
                messagebox.showinfo("Success",
                    "Participant entry marked.
                    \nEntry Granted")
            else:
                messagebox.showerror("ALERT", "Participant
                    already entered. \nEntry Denied")
            screen7.focus_force()
            qrpid.set("")
        elif qrpid.get() != "":
            messagebox.showerror("ALERT", "No Event
                selected")
        elif qreve.get() != "":
            messagebox.showerror("ALERT","No QR ID found")
        else:
            messagebox.showerror("ALERT",
                "Fields Incomplete")
    except:
        messagebox.showerror("ALERT", "Unable to connect
            to the server")
    finally:
        screen7.focus_force()
```

```python
# GUI of entry marker
screen7 = Toplevel(config.screen4)
screen7.title("Event Entry Management")
screen7.geometry(config.screen7geo)
screen7.resizable(False, False)
screen7.config(background=colr)
screen7.focus_force()
icon = PhotoImage(file="./resc/qr-code-scan.png")
screen7.iconphoto(False, icon)


try:
    evts = fi.get_events()
    qrpid = StringVar()
    qreve = StringVar()
    label = Label(screen7, text="Participant Entry",
        bg=colr, font=("Times New Roman", 20, 'bold'))
    label.configure(foreground="white", anchor="center")
    label.grid(row=1, column=1, padx=20, pady=(20, 15),
        columnspan=3)
    label = Label(screen7, width=15, text="Select Event :
        ", bg=colr)
    label.configure(foreground="white")
    label.grid(row=3, column=1, padx=5, pady=10)
    screen7.entry1 = ttk.Combobox(screen7, width=17,
        textvariable=qreve, state="readonly")
    screen7.entry1.grid(row=3, column=2, padx=5, pady=10,
        columnspan=1)
    screen7.entry1['values'] = evts
    screen7.entry1.current()
    label = Label(screen7, width=15, text="QR ID : ",
        bg=colr)
    label.configure(foreground="white")
    label.grid(row=4, column=1, padx=5, pady=10)
```

```python
        screen7.entryname = Entry(screen7, width=20,
            textvariable=qrpid)
        screen7.entryname.grid(row=4, column=2, padx=5,
            pady=10, columnspan=2)
        buton = Button(screen7, width=15, text="Scan(ctrl+s)",
            command=callscan)
        buton.grid(row=5, column=1, padx=15, pady=(20, 0),
            columnspan=1)

        # binding Ctrl + s key as shortcut to open scanner
        screen7.bind("<Control-s>",
            lambda event=None: buton.invoke())
        button = Button(screen7, width=15, text="Mark (↵)",
            command=marker)
        button.grid(row=5, column=2, padx=15, pady=(20, 0),
            columnspan=1)

        # binding Enter key as shortcut to mark the entry
        screen7.bind('<Return>', lambda event=None:
            button.invoke())
        label = Label(screen7, width=35, text="Press 'Esc' or
            'q' to close camera \nPress 's' to save image.",
            bg=colr)
        label.configure(foreground="#767777")
        label.grid(row=6, column=1, padx=5, pady=10,
            columnspan=2)
    except:
        messagebox.showerror("ALERT", "Unable to connect to
            the server")
    finally:
        screen7.focus_force()
```

```python
        # set focus to management window on closing
        config.screen4.focus_force()



# code to register & generate QR for participant
def QRP():
    """
    This function is a module that manages aspects of
    participant registration
    using serveral sub modules.
    """


    # call scanner
    def callscan():
        """
        This function calls the QR Scanner to register
        existing participant in another event.
        """


        resp = scanner()
        if resp == 0:
            config.screen5.focus_force()
        else:
            qrID.set(resp)
            config.screen5.focus_force()

    # code for GUI of QR generator
    def QRGen():
        """
        This function loads the GUI of participant
        registration window.
        """
```

```python
gcolor = "#161a2d"
config.screen5 = Toplevel(config.screen4)
config.screen5.title("QR Generator")
config.screen5.geometry(config.screen5geo)
config.screen5.resizable(False, False)
config.screen5.config(background=gcolor)
icon = PhotoImage(file="./resc/laptop.png")
config.screen5.iconphoto(False, icon)
try:
    evts = fi.get_events()
    label = Label(config.screen5, text="Event
        Registration", bg=gcolor,
        font=("Times New Roman", 20, 'bold'))
    label.configure(foreground="white",
        anchor="center")
    label.grid(row=0, column=2, padx=5, pady=5,
        columnspan=4)
    label = Label(config.screen5, text="Enter all
        details or QR-ID of participant", bg=gcolor)
    label.configure(foreground="white")
    label.grid(row=1, column=1, padx=5, pady=10,
        columnspan=3)
    label = Label(config.screen5, text="Enter Name : "
        , bg=gcolor)
    label.configure(foreground="white")
    label.grid(row=2, column=1, padx=5, pady=10)
    config.screen5.entryname = Entry(config.screen5,
        width=30, textvariable=qrName)
    config.screen5.entryname.grid(row=2, column=2,
        padx=5, pady=10, columnspan=2)
    config.screen5.entryname.focus_set()
    label = Label(config.screen5, text="Enter Phno : "
        , bg=gcolor)
```

```python
label.configure(foreground="white")
label.grid(row=3, column=1, padx=5, pady=10)
config.screen5.entryphno = Entry(config.screen5,
    width=30, textvariable=qrphno)
config.screen5.entryphno.grid(row=3, column=2,
    padx=5, pady=10, columnspan=2)
label = Label(config.screen5, text="Enter Email :"
    , bg=gcolor)
label.configure(foreground="white")
label.grid(row=4, column=1, padx=5, pady=10)
config.screen5.entrymail = Entry(config.screen5,
    width=30, textvariable=qrmail)
config.screen5.entrymail.grid(row=4, column=2,
    padx=5, pady=10, columnspan=2)
label = Label(config.screen5, text="QR ID : ",
    bg=gcolor)
label.configure(foreground="white")
label.grid(row=5, column=1, padx=5, pady=10)
config.screen5.entry = Entry(config.screen5,
    width=15, textvariable=qrID)
config.screen5.entry.grid(row=5, column=2, padx=10
    , pady=10, columnspan=1, sticky='w')
sbtn = Button(config.screen5, width=10, text="Scan
    (ctrl + s)", command=callscan)
sbtn.grid(row=5, column=3, padx=5, pady=10,
     sticky='e')

# binding Ctrl + s key as shortcut to open scanner
config.screen5.bind("<Control-s>",
    lambda event=None: sbtn.invoke())
ttk.Separator(config.screen5,
    orient=HORIZONTAL).grid(column=1, row=6,
    columnspan=3, sticky='ew')
```

```python
label = Label(config.screen5, text="1st Event
    Name : ", bg=gcolor)
label.configure(foreground="white")
label.grid(row=7, column=1, padx=5, pady=10)
label = Label(config.screen5, text="2nd Event
    Name : ", bg=gcolor)
label.configure(foreground="white")
label.grid(row=8, column=1, padx=5, pady=10)
config.screen5.entry1 = ttk.Combobox(
    config.screen5, width=27,textvariable=qrevent1
    , state="readonly")
config.screen5.entry1.grid(row=7, column=2,
    padx=5, pady=10, columnspan=2)
config.screen5.entry1['values'] = evts
config.screen5.entry1.current()
config.screen5.entry2 = ttk.Combobox(
    config.screen5, width=27,
    textvariable=qrevent2, state="readonly")
config.screen5.entry2.grid(row=8, column=2,
     padx=5, pady=10, columnspan=2)
config.screen5.entry2['values'] = evts
config.screen5.entry2.current()
label = Label(config.screen5, text="QR Code : ",
    bg=gcolor)
label.configure(foreground="white")
label.grid(row=9, column=1, padx=5, pady=10)
button = Button(config.screen5, width=10,
    text="Generate (↵)", command=QRCodeGenerate)
button.grid(row=9, column=2, padx=5, pady=10,
    columnspan=1)
# binding Enter key as shortcut to generate QR
config.screen5.bind('<Return>', lambda event=None:
    button.invoke())
```

```python
        buton = Button(config.screen5, width=10,
            text="Clear (ctrl + r)", command=QRClear)
        buton.grid(row=9, column=3, padx=5, pady=10,
            columnspan=1)
        # binding Ctrl + r key as shortcut to clear fields
        config.screen5.bind("<Control-r>",
            lambda event=None: buton.invoke())
        config.screen5.imageLabel = Label(config.screen5,
            background=gcolor)
        config.screen5.imageLabel.grid(row=2, column=4,
            rowspan=9, columnspan=3, padx=(10, 5), pady=10
            )
        image = Image.open("./resc/wait.png")
        image = image.resize((350, 350), Image.ANTIALIAS)
        image = ImageTk.PhotoImage(image)
        config.screen5.imageLabel.config(image=image)
        config.screen5.imageLabel.photo = image

    except:
        messagebox.showerror("ALERT", "Unable to connect
            to the server")

    finally:
        config.screen5.focus_force()
```

```python
# reload wait image
def ld():
    """
    This function loads the waiting screen image after
    fields are cleared.
    """

    image = Image.open("./resc/wait.png")
    image = image.resize((350, 350), Image.ANTIALIAS)
    image = ImageTk.PhotoImage(image)
    config.screen5.imageLabel.config(image=image)
    config.screen5.imageLabel.photo = image


# code for clearing values of GUI fields
def QRClear():
    """
    This function clears the fields in participant
    registration window.
    """

    config.screen5.entryname.focus_set()
    qrName.set("")
    qrphno.set("")
    qrmail.set("")
    qrevent1.set("")
    qrevent2.set("")
    qrID.set("")
    image = Image.open("./resc/done.png")
    image = image.resize((350, 350), Image.ANTIALIAS)
    image = ImageTk.PhotoImage(image)
    config.screen5.imageLabel.config(image=image)
    config.screen5.imageLabel.photo = image
    config.screen5.after(500, ld)
```

```python
# code to generate QR with participant data
def QRCodeGenerate():
    """
    This function generates QR Code for participant after
    validating the information provided by them.
    """
    if (qrName.get() != '') and (qrphno.get() != '') and
    (qrmail.get() != '') and (qrevent1.get() != ''):
        if len(qrphno.get()) == 10:
            try:
                rege = '^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]
                    \w{2,3}$'
                if re.search(rege, qrmail.get()):
                    content = qrName.get().lower() +
                        "-" + qrphno.get()
                    i = QRdatamgSQL(content)
                    if i == 0:
                        messagebox.showerror("ALERT",
                            "Internal error in
                            registration")
                    elif i == 2:
                        messagebox.showinfo("Pay Attention
                            ", "Participant already
                            registered in event 1.
                            \nRegistration for event 2
                            complete")
                    elif i == 3:
                        messagebox.showinfo("Pay Attention
                            ", "Participant already
                            registered in event 2.
                            \nRegistration for event 1
                            complete")
```

```python
                elif i == 4:
                    messagebox.showerror("ALERT",
                        "Participant already
                        registered in provided events
                        \nRegistration Aborted")
                elif i == 1:
                    qrGenerate = pyqrcode.create(
                        content)
                    qrCodePath = './data/'
                    qrCodeName = qrCodePath +
                        qrphno.get() + ".png"
                    qrGenerate.png(qrCodeName,
                        scale=10)
                    image = Image.open(qrCodeName)
                    image = image.resize((350, 350),
                        Image.ANTIALIAS)
                    image = ImageTk.PhotoImage(image)
                    config.screen5.imageLabel.config(
                        image=image)
                    config.screen5.imageLabel.photo =
                        image
                    QRdatamgXL()
                config.screen5.focus_force()
            else:
                messagebox.showerror("ALERT", "Invalid
                    Email ID")
                config.screen5.focus_force()
                config.screen5.entrymail.focus_set()
        except:
            messagebox.showerror("ALERT", "Error in
                Generating QR or Phone NaN")
            config.screen5.focus_force()
            config.screen5.entryphno.focus_set()
```

```python
        else:
            messagebox.showerror("ALERT", "Invalid Phone
                Number")
            config.screen5.focus_force()
            config.screen5.entryphno.focus_set()
    elif (qrID.get() != '') and (qrevent1.get() != ''):
        autofil()
    else:
        messagebox.showerror("ALERT", "Fields Incomplete")
        config.screen5.focus_force()


# code for autofill
def autofil():
    """
    This function registers an existing participant in
    another event.
    """


    uid = qrID.get()[2:-1]
    eve = [qrevent1.get(), qrevent2.get()]
    if qrevent2.get() == "":
        eve = [qrevent1.get()]
    try:
        resp = fi.add_part(p_id=uid, name="", email_id="",
phone="", events=eve)
        if resp == 0:
            messagebox.showerror("ALERT", "No Registration
                found on this QR ID \nRegistration Aborted
                ")
        elif resp == 2:
            messagebox.showinfo("Pay Attention",
                "Participant already registered in event 1
                \nRegistration for event 2 complete")
```

```python
        elif resp == 3:
            messagebox.showinfo("Pay Attention",
                "Participant already registered in event 2
                \nRegistration for event 1 complete")
        elif resp == 4:
            messagebox.showerror("ALERT",
                "Participant already registered in
                provided events. \nRegistration Aborted")
        else:
            messagebox.showinfo("Success", "Registration
                Completed")
    except:
        messagebox.showerror("ALERT", "Unable to connect
            to the server")
    finally:
        config.screen5.focus_force()


# code to add participant data to excel sheet
def QRdatamgXL():
    """
    This function adds the participant data to an Excel
    sheet in local machine
    as soon as the participant data is added to the
    server.
    """


    # path for excel file
    path = "./data/regdata.xlsx"
    try:
        wb = load_workbook(path)
    except FileNotFoundError:
        wb = Workbook(path)
        wb.save(path)
```

```python
        except PermissionError:
            messagebox.showerror("Alert", "File access denied.
                \nClose the excel sheet to fix this issue.")
    wb = load_workbook(path)
    sheet = wb.active
    row = (qrName.get(), qrphno.get(), qrmail.get(),
        qrevent1.get(), qrevent2.get())
    sheet.append(row)
    wb.save(path)


# code to add participant data to SQL
def QRdatamgSQL(content: str) -> int:
    """
    This function contacts the server to add participant
    data in the database.

    :param content: it contains the data to be used as
    participant ID from the QR Code.
    :return: it returns the response provided by the
    server, to check if data has been stored.
    """

    eve = [qrevent1.get(), qrevent2.get()]
    if qrevent2.get() == "":
        eve = [qrevent1.get()]
    try:
        resp = fi.add_part(p_id=content,name=qrName.get(),
            email_id=qrmail.get(), phone=qrphno.get(),
            events=eve)
        return resp
    except:
        messagebox.showerror("ALERT", "Unable to connect
            to the server")
```

```python
        finally:
            config.screen5.focus_force()


    qrName = StringVar()
    qrphno = StringVar()
    qrmail = StringVar()
    workbook = Workbook()
    qrevent1 = StringVar()
    qrevent2 = StringVar()
    qrID = StringVar()
    QRGen()



# code to maintain event list
def eventmgm():
    """
    This function is a module that loads the GUI of event
    management window &
    contains sub modules to manage aspects of events.
    """


    # clear fields
    def clrevent():
        """
        This function clears all the fields in the event
        management window.
        """


        adevent.focus_set()
        evename.set("")
        evedate.set("")
        evetime.set("")
```

```python
# clear placeholder of date field
def clrdt(event: object):
    """
    This function is used to monitor widget selection to
    clear the placeholder value.

    :param event: checks mouse interrupt.
    """

    evedate.set("")


# clear placeholder of time field
def clrti(event: object):
    """
    This function is used to monitor widget selection to
    clear the placeholder value.

    :param event: checks mouse interrupt.
    """

    evetime.set("")


# add events to database
def addevent():
    """
    This function contacts the server to add an event data
    & performs validation
    on the data provided & stops if event already exists.
    """

    if (evename.get() != "") and (evedate.get() != "") and
    (evetime.get() != ""):
        try:
```

```python
        resp = fi.add_event(name=evename.get(),
            date=evedate.get(), time=evetime.get())
        if resp == 0:
            messagebox.showerror("ALERT", "Another
                event entry with same name already
                exists")
            adevent.focus_set()
        elif resp == 2:
            messagebox.showerror("ALERT", "Wrong date
                format: \nCorrect format: YYYY-MM-DD")
            adevedt.focus_set()
        elif resp == 3:
            messagebox.showerror("ALERT", "Wrong Time
                format: \nCorrect format: HH:MM")
            adeveti.focus_set()
        else:
            messagebox.showinfo("Success", "Event
                added successfully")
    except:
        messagebox.showerror("ALERT", "Unable to
                connect to the server")
    finally:
        screen6.focus_force()
else:
    messagebox.showerror("ALERT", "Fields Incomplete")
    screen6.focus_force()
```

```python
# remove events from database
def remevent():
    """
    This function contacts the server to remove an event
    from the database by
    validating the information & stops the process if
    information is invalid
    or if any participant is registered to the event.
    """

    if (evename.get() != "") and (evedate.get() != "") and
    (evetime.get() != ""):
        try:
            resp = fi.remove_event(name=evename.get(),
                date=evedate.get(), time=evetime.get())
            if resp == 0:
                messagebox.showerror("ALERT", "Someone is
                    registered in this event")
                adevent.focus_set()
            elif resp == 1:
                messagebox.showinfo("Success", "Event
                    removed successfully")
            elif resp == 2:
                messagebox.showerror("ALERT", "Wrong date
                    format: \nCorrect format: YYYY-MM-DD")
                adevedt.focus_set()
            elif resp == 3:
                messagebox.showerror("ALERT", "Wrong Time
                    format: \nCorrect format: HH:MM")
                adeveti.focus_set()
            elif resp == 4:
                messagebox.showerror("ALERT", "Invalid
                    event details")
```

```python
        except:
            messagebox.showerror("ALERT", "Unable to
                connect to the server")
        finally:
            screen6.focus_force()
    else:
        messagebox.showerror("ALERT", "Fields Incomplete")
        screen6.focus_force()


# GUI code for event manager
screen6 = Toplevel(config.screen4)
screen6.title("Event Manager")
screen6.geometry(config.screen6geo)
screen6.resizable(False, False)
screen6.config(background="green")
icon = PhotoImage(file="./resc/team-management.png")
screen6.iconphoto(False, icon)
screen6.focus_force()
evename = StringVar()
evetime = StringVar()
evedate = StringVar()
label = Label(screen6, text="Event Management",
    bg="green", font=("Times New Roman", 20, 'bold'))
label.configure(foreground="white", anchor="center")
label.grid(row=0, column=1, padx=(17, 0), pady=(10, 15),
    columnspan=4)
lbl = Label(screen6, text="Event Name", bg="green")
lbl.configure(foreground="white")
lbl.grid(row=1, column=1, padx=(40, 5), pady=5,
    columnspan=1)
adevent = Entry(screen6, width='17', textvariable=evename)
adevent.grid(row=1, column=2, padx=5, pady=5, columnspan=1
)
```

```python
lbl = Label(screen6, text="Event Date", bg="green")
lbl.configure(foreground="white")
lbl.grid(row=2, column=1, padx=(40, 5), pady=5,
    columnspan=1)
adevedt = Entry(screen6, width='17', textvariable=evedate)
adevedt.insert(0, 'YYYY-MM-DD')
adevedt.bind("<FocusIn>", clrdt)
adevedt.grid(row=2, column=2, padx=5, pady=5, columnspan=1
)
lbl = Label(screen6, text="Event Time", bg="green")
lbl.configure(foreground="white")
lbl.grid(row=3, column=1, padx=(40, 5), pady=5,
    columnspan=1)
adeveti = Entry(screen6, width='17', textvariable=evetime)
adeveti.insert(0, 'HH:MM')
adeveti.bind("<FocusIn>", clrti)
adeveti.grid(row=3, column=2, padx=5, pady=5, columnspan=1
)
bnt = Button(screen6, text="Clear (ctrl + r)",
    command=clrevent, width=18)
bnt.grid(row=1, column=3, padx=5, pady=5, columnspan=2)
nbt = Button(screen6, text="Add Event (ctrl + a)",
    command=addevent, width=18)
nbt.grid(row=2, column=3, padx=5, pady=5, columnspan=2)
tnb = Button(screen6, text="Remove Event (ctrl + d)",
    command=remevent, width=18)
tnb.grid(row=3, column=3, padx=5, pady=5, columnspan=2)
adevent.focus_set()

# binding Ctrl + a key as shortcut to add event
screen6.bind("<Control-a>",
    lambda event=None: nbt.invoke())
```

```python
    # binding Ctrl + d key as shortcut to remove event
    screen6.bind("<Control-d>",
        lambda event=None: tnb.invoke())
    # binding Ctrl + r key as shortcut to clear fields
    screen6.bind("<Control-r>",
        lambda event=None: bnt.invoke())


    # code to monitor event management window close event
    def on_closing():
        """
        This function monitors event management window close
        event &
        reopens tasks window.
        """


        config.screen4.deiconify()
        screen6.destroy()
    screen6.protocol("WM_DELETE_WINDOW", on_closing)


# code to generate report
def report_gen():
    """
    This function is a module that contacts the server to
    generate report of
    participants & their attendance in various events.
    """


    try:
        rep = fi.get_report()
        # path to report excel file
        p = "./data/report.xlsx"
```

```python
        try:
            wb = load_workbook(p)
        except FileNotFoundError:
            wb = Workbook(p)
            wb.save(p)
        try:
            wb = load_workbook(p)
            sheet = wb.active
            sheet.delete_cols(1, 20)
            sheet.delete_rows(1, 1000)
            row = ("S.No.", "QR ID", "Name",
                "E-mail", "Phone no.")
            sheet.append(row)
            wb.save(p)
            count = 1
            for i in rep:
                row = (count, i[0], i[1], i[2], i[3])
                col = 6
                e_count = 1
                count += 1
                sheet.append(row)
                for x in i[4].split(","):
                    sheet.cell(row=1, column=col).value =
                        "Event " + str(e_count)
                    sheet.cell(row=count, column=col).value =
                        x[:-1]
                    sheet.cell(row=1, column=col + 1).value =
                        "E-" + str(e_count) + " Entry"
                    sheet.cell(row=count, column=col+1).value
                        = x[-1].replace("1", "Not Entered"
                        ).replace("2", "Entered")
                    col += 2
                    e_count += 1
```

```python
            wb.save(p)
            messagebox.showinfo("Success", "Report Generated
                successfully \nPath: " + p)
        except PermissionError:
            messagebox.showerror("Alert", "File access denied.
                \nClose the excel sheet OR Run program as
                Administrator to fix this issue.")
    except:
        messagebox.showerror("ALERT", "Unable to connect to
            the server")
    finally:
        # set focus to management window on closing
        config.screen4.focus_force()


# code to manage user tasks
def mgm_page():
    """
    This function loads the GUI of main tasks window of the
    app & guides its users of its purpose,
    It contains the participant registration, participant
    entry, event management & report generation modules.
    """


    # GUI for organizer management
    config.screen3.withdraw()
    config.screen4 = config.screen4
    config.screen4 = Toplevel(config.screen3)
    config.screen4.title("Select")
    config.screen4.geometry(config.screen4geo)
    config.screen4.resizable(False, False)
    config.screen4.config(background=colr)
    icon = PhotoImage(file="./resc/process.png")
```

```python
    config.screen4.iconphoto(False, icon)
    config.screen4.focus_force()
    label = Label(config.screen4, text="Event Registration &
        Verification Using QR", bg=colr, fg="white",
        font=("Times New Roman", 20, 'bold'))
    label.grid(row=1, column=1, padx=5, pady=(20, 30),
        columnspan=3)
    label = Label(config.screen4, text="Participant
        Registration", bg=colr, fg="white",
        font=("Times New Roman", 12, 'bold'))
    label.grid(row=2, column=1, padx=(30, 40), pady=15,
        columnspan=1)
    btn = Button(config.screen4, width=15, borderwidth=0,
        text="Registry (ctrl + g)", command=QRP)
    btn.grid(row=3, column=1, padx=(30, 40), pady=10,
        columnspan=1)
    label = Label(config.screen4, text="Register participants
        in one or more \nevents & Generate QR code,
        \nunique for everyone", bg=colr, fg="white")
    label.grid(row=4, column=1, padx=(30, 40), pady=10,
        columnspan=1)
    label = Label(config.screen4, text="Participant
        Verification", bg=colr, fg="white",
font=("Times New Roman", 12, 'bold'))
    label.grid(row=2, column=3, padx=50, pady=15, columnspan=1
    )
    bnt = Button(config.screen4, width=15, borderwidth=0,
        text="Entry (ctrl + s)", command=QRScan)
    bnt.grid(row=3, column=3, padx=50, pady=10, columnspan=1)
    label = Label(config.screen4, text="Verify and mark
        participant's entry, \nusing the QR code provided,
        \nfor each event", bg=colr, fg="white")
```

```python
label.grid(row=4, column=3, padx=50, pady=10, columnspan=1
)
ttk.Separator(config.screen4, orient=HORIZONTAL).grid(
    column=1, row=5, columnspan=3, sticky='ew')
ttk.Separator(config.screen4, orient=HORIZONTAL).grid(
    column=2, row=2, rowspan=9, sticky='ns')
label = Label(config.screen4, text="Event Management",
    bg=colr, fg="white", font=("Times New Roman", 12,
    'bold'))
label.grid(row=6, column=1, padx=(30, 40), pady=15,
    columnspan=1)
tbn = Button(config.screen4, width=15, borderwidth=0,
    text="Manage (ctrl + e)", command=eventmgm)
tbn.grid(row=7, column=1, padx=(30, 40), pady=10,
    columnspan=1)
label = Label(config.screen4, text="Add and remove events
    to be organized, \nalong with their date and time",
    bg=colr, fg="white")
label.grid(row=8, column=1, padx=(30, 40), pady=10,
    columnspan=1)
label = Label(config.screen4, text="Report Generator",
    bg=colr, fg="white", font=("Times New Roman", 12,
    'bold'))
label.grid(row=6, column=3, padx=50, pady=15, columnspan=1
)
ttbn = Button(config.screen4, width=15, borderwidth=0,
    text="Report (ctrl + r)", command=report_gen)
ttbn.grid(row=7, column=3, padx=5, pady=10, columnspan=1)
label = Label(config.screen4, text="Generate report for
    all events and \nparticipants along with their
    details", bg=colr, fg="white")
label.grid(row=8, column=3, padx=50, pady=10, columnspan=1
)
```

```python
# binding Ctrl + g key as shortcut to open participant
# adding window
config.screen4.bind("<Control-g>",
    lambda event=None: btn.invoke())


# binding Ctrl + s key as shortcut to open participant
# entry window
config.screen4.bind("<Control-s>",
    lambda event=None: bnt.invoke())


# binding Ctrl + e key as shortcut to open event
# management window
config.screen4.bind("<Control-e>",
    lambda event=None: tbn.invoke())


# binding Ctrl + r key as shortcut to generate report
config.screen4.bind("<Control-r>",
    lambda event=None: ttbn.invoke())


# code to monitor main tasks window / app close event
def on_closing():
    """
    This function monitors the close event main tasks
    window of the app.
    it also terminates the entire app.
    """

    config.screen1.destroy()
config.screen4.protocol("WM_DELETE_WINDOW", on_closing)
```

```python
# GUI & code for login & signup
def main_page():
    """
    This function is a module that loads the GUI of login
    window and contains
    user login & registration modules, which are managed by
    sub modules.
    """


    # code to clear login data fields after successful login
    def clrlogin():
        """
        This function clears the fields of login window
        """


        username_verify.set("")
        password_verify.set("")
        mgm_page()

    # code to organizer management
    def register_user():
        """
        This function manages user registration success
        window.
        """


        # code to monitor screen1 close event
        def on_closing():
            """
            This function monitors the Registration window
            close event.
            """
```

```python
        config.screen2.destroy()
        config.screen1.deiconify()
    config.screen2.protocol("WM_DELETE_WINDOW", on_closing
)


# GUI for user add success
def disab():
    """
    This function sets the GUI of user registration
    success window.
    """

    config.screen1_5 = Toplevel(config.screen1)
    config.screen1_5.title("Success")
    config.screen1_5.geometry(config.screen1_5geo)
    config.screen1_5.resizable(False, False)
    config.screen1_5.config(background="green")
    config.screen1_5.focus_force()

    # code to call login success screen
    def calllog():
        """
        This function monitors the user Registration
        window close event & clears the fields
        present in it & reopens the admin login
        window.
        """

        username.set("")
        emailid.set("")
        phno.set("")
        password.set("")
        config.screen3.deiconify()
```

```python
        config.screen1_5.destroy()
        config.screen2.destroy()
        adminlogin()


    label = Label(config.screen1_5, text="",
        bg="green")
    label.grid(row=1, column=1)
    label = Label(config.screen1_5,
        text="Registeration Success", width='30',
        bg="green", font=("Times New Roman", 20,
        'bold'))
    label.configure(foreground="white")
    label.grid(pady=5, row=2, column=1, columnspan=1)
    bttn = Button(config.screen1_5, text="OK (↵)",
    width="15", command=calllog)
    bttn.grid(pady=5, row=3, column=1, columnspan=1)
    # binding Enter key as shortcut to proceed
    config.screen1_5.bind('<Return>',
        lambda event=None: bttn.invoke())

disab()
```

```python
# registry data validation
def valinp():
    """
    This function validates the data provided by admin to
    add more users,
    and sends the validated data to the server.
    """


    if (username.get() != "") and (emailid.get() != "")
    and (phno.get() != "") and (password.get() != "")
    and (config.perm_entry.get() != "Select"):
        if len(phno.get()) == 10:
            rege = '^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]
                \w{2,3}$'
            if re.search(rege, emailid.get()):
                if re.fullmatch(r'[A-Za-z0-9@#$%^&+=]
                {8,}', password.get()):
                    perm = config.perm_entry.get()
                    perm = 2 if perm == "Admin" else 1
                    try:
                        resp = fi.add_user(
                            name=username.get(),
                            email_id=emailid.get(),
                            password=password.get(),
                            phone=int(phno.get()),
                            perm=perm)
                        if resp == 0:
                            messagebox.showerror("ALERT",
                                "User email already
                                exists")
                            config.screen2.focus_force()
                        else:
                            register_user()
```

```python
                    except:
                        messagebox.showerror("ALERT",
                            "Unable to connect to the
                            server")
                    finally:
                        config.screen2.focus_force()
                else:
                    messagebox.showerror("ALERT",
                        "Password not Strong")
                    config.screen2.focus_force()
            else:
                messagebox.showerror("ALERT", "Invalid
                    Email")
                config.screen2.focus_force()
        else:
            messagebox.showerror("ALERT", "Invalid Phone
                Number")
            config.screen2.focus_force()
    else:
        messagebox.showerror("ALERT", "Fields Incomplete")
        config.screen2.focus_force()


# GUI code for adding user
def register():
    """
    This function sets the GUI of user registration
    window.
    """

    config.screen3.withdraw()
    config.screen2 = Toplevel(config.screen1)
    config.screen2.title("Register")
    config.screen2.geometry(config.screen2geo)
```

```python
config.screen2.resizable(False, False)
config.screen2.config(background=colr)
icon = PhotoImage(file="./resc/add.png")
config.screen2.iconphoto(False, icon)
config.screen2.focus_force()
labl = Label(config.screen2, text="Please enter user
    information", width="30", bg=colr)
labl.configure(foreground="white", font=("Times New
    Roman", 20, 'bold'))
labl.grid(row=1, column=1, padx=5, pady=5,
    columnspan=2)
labl = Label(config.screen2, text="User Name",
    width='30', bg=colr)
labl.configure(foreground="white")
labl.grid(row=2, column=1, padx=5, pady=5,
    columnspan=1)
username_entry = Entry(config.screen2,
    textvariable=username)
username_entry.grid(row=3, column=1, padx=5, pady=5,
    columnspan=1)
username_entry.focus_set()
labl = Label(config.screen2, text="Email ID",
    width='30', bg=colr)
labl.configure(foreground="white")
labl.grid(row=2, column=2, padx=5, pady=5,
    columnspan=1)
emailid_entry = Entry(config.screen2,
    textvariable=emailid)
emailid_entry.grid(row=3, column=2, padx=5, pady=5,
    columnspan=1)
labl = Label(config.screen2, text="Phone Number",
    width='30', bg=colr)
labl.configure(foreground="white")
```

```python
        labl.grid(row=4, column=1, padx=5, pady=5,
            columnspan=1)
        phno_entry = Entry(config.screen2, textvariable=phno)
        phno_entry.grid(row=5, column=1, padx=5, pady=5,
            columnspan=1)
        labl = Label(config.screen2, text="Password",
            width='30', bg=colr)
        labl.configure(foreground="white")
        labl.grid(row=4, column=2, padx=5, pady=5,
            columnspan=1)
        password_entry = Entry(config.screen2, show="*",
            textvariable=password)
        password_entry.grid(row=5, column=2, padx=5, pady=5,
            columnspan=1)
        labl = Label(config.screen2, text="", width="30",
            bg=colr)
        labl.grid(row=6, column=1, padx=5, pady=5,
            columnspan=2)
        labl = Label(config.screen2, text="Permission : ",
            width='30', bg=colr)
        labl.configure(foreground="white")
        labl.grid(row=7, column=1, padx=5, pady=5,
            columnspan=1)
        config.perm_entry = ttk.Combobox(config.screen2,
            textvariable=rights, width="17",
            values=["Select", "Admin", "User"],
            state="readonly")
        config.perm_entry.current(0)
        config.perm_entry.grid(row=7, column=2, columnspan=1,
            pady=5)
        labl = Label(config.screen2, text="", bg=colr)
        labl.grid(row=8, column=1, columnspan=2)
```

```python
    regbtn = Button(config.screen2, text="Sumbit (↵)",
        width='18', command=valinp)
    regbtn.grid(row=9, column=1, padx=5, pady=5,
        columnspan=2)


    # binding Enter key as shortcut to proceed
    config.screen2.bind('<Return>', lambda event=None:
        regbtn.invoke())


    # code to monitor screen2 close event
    def on_closing():
        """
        This function monitors the user registration
        window close event &
        it reopens the admin login success window.
        """


        config.screen3.deiconify()
        config.screen2.destroy()
    config.screen2.protocol("WM_DELETE_WINDOW", on_closing
    )

# GUI if user is admin
def adminlogin():
    """
    This functions sets the GUI if user is an admin.
    """


    config.screen3.geometry(config.screen3geo)
    label = Label(config.screen3, text="Login Success",
        width='30', bg="green")
    label.configure(foreground="white", font=("Times New
        Roman", 16, 'bold'))
```

```python
        label.grid(row=1, column=1, pady=5, columnspan=1)
        bttnn = Button(config.screen3, text="OK (↵)",
            width="15", command=clrlogin)
        bttnn.grid(row=2, column=1, pady=5, columnspan=1)
        bttn = Button(config.screen3, text="Add User(ctrl+a)",
            width="15", command=register)
        bttn.grid(row=3, column=1, pady=5, columnspan=1)

        # binding Enter key as shortcut to proceed
        config.screen3.bind('<Return>', lambda event=None:
            bttnn.invoke())

        # binding Ctrl + a key as shortcut to access user
        # adding screen
        config.screen3.bind("<Control-a>",
            lambda event=None: bttn.invoke())

    # GUI if user is user
    def userlogin():
        """
        This function sets the GUI if the user is an
        organizer.
        """

        config.screen3.geometry(config.screen3geo)
        label = Label(config.screen3, text="", bg="green")
        label.grid(row=1, column=1, pady=5)
        label = Label(config.screen3, text="Login Success",
            width='30', bg="green")
        label.configure(foreground="white", font=("Times New
            Roman", 16, 'bold'))
        label.grid(row=2, column=1, pady=5)
```

```python
    bttn = Button(config.screen3, text="OK (↵)",
        width="10", command=clrlogin)
    bttn.grid(row=3, column=1, pady=5)
    # binding Enter key as shortcut to proceed
    config.screen3.bind('<Return>', lambda event=None:
        bttn.invoke())


# code for GUI & user details verification
def login_verify():
    """
    This function sets the GUI for login success screen &
    it sends the login credentials to be verified &
    authenticated to the server.
    """

    config.screen1.withdraw()
    config.screen3 = Toplevel(config.screen1)
    config.screen3.title("Info")
    config.screen3.geometry(config.screen3geo)
    config.screen3.resizable(False, False)
    config.screen3.config(background="green")
    config.screen3.focus_force()
    icon = PhotoImage(file="./resc/check.png")
    config.screen3.iconphoto(False, icon)
```

```python
    # code to monitor screen3 close event
    def on_closing():
        """
        This function monitors login success screen close
        event, and reopens login window.
        It is the equivalent of logging out.
        """


        clrlogin()
        config.screen1.deiconify()
        config.screen3.destroy()
    config.screen3.protocol("WM_DELETE_WINDOW", on_closing
    )

    try:
        resp = fi.login(uid=username_verify.get(),
            password=password_verify.get())
        if resp == 2:
            adminlogin()
        elif resp == 1:
            userlogin()
        elif resp == 0:
            on_closing()
            messagebox.showerror("ALERT", "Invalid
                User/password")
            username_entry1.focus_set()
        else:
            on_closing()
            messagebox.showerror("ALERT", "Invalid User")
            username_entry1.focus_set()
    except:
        on_closing()
        messagebox.showerror("ALERT", "No Internet")
```

```python
        username_entry1.focus_set()


# check if fields are complete
def chk_login_verify():
    """
    This function ensures that the fields in login
    screen are completed.
    """

    if (username_verify.get() != "") and (
    password_verify.get() != ""):
        login_verify()
    elif username_verify.get() == "":
        messagebox.showerror("ALERT", "Username Field
            Incomplete")
        username_entry1.focus_set()
    elif password_verify.get() == "":
        messagebox.showerror("ALERT", "Password Field
            Incomplete")
        password_entry1.focus_set()
    else:
        messagebox.showerror("ALERT", "Fields Incomplete")
        username_entry1.focus_set()

# code for login GUI
config.screen1 = Tk()
config.screen1.title("Login")
config.screen1.geometry(config.screen1geo)
config.screen1.config(background=colr)
config.screen1.resizable(False, False)
icon = PhotoImage(file="./resc/login.png")
config.screen1.iconphoto(False, icon)
username = StringVar()
```

```python
password = StringVar()
emailid = StringVar()
phno = StringVar()
rights = StringVar()
username_verify = StringVar()
password_verify = StringVar()
label = Label(text="", bg=colr)
label.grid(row=1, column=1)
label = Label(text="Please Enter your Login \nInformation"
    , width='30', bg=colr)
label.configure(foreground="white", font=("Times New Roman
    ", 18, 'bold'))
label.grid(row=2, column=1, padx=5, pady=5, columnspan=1)
label = Label(text="User Email ID : ", width='30', bg=colr
)
label.configure(foreground="white")
label.grid(row=4, column=1, padx=25, pady=5, columnspan=1)
username_entry1 = Entry(width="21",
    textvariable=username_verify)
username_entry1.grid(row=5, column=1, padx=5, pady=5,
    columnspan=1)
username_entry1.focus_set()
label = Label(text="Password : ", width='30', bg=colr)
label.configure(foreground="white")
label.grid(row=6, column=1, padx=5, pady=5, columnspan=1)
password_entry1 = Entry(width='21', show="*",
    textvariable=password_verify)
password_entry1.grid(row=7, column=1, padx=5, pady=5,
    columnspan=1)
label = Label(text="", bg=colr)
label.grid(row=8, column=1)
btnn = Button(text="Login (↵)", width="18",
    command=chk_login_verify)
```

```python
        btnn.grid(row=9, column=1, padx=5, pady=5, columnspan=1)
        # binding Enter key as shortcut to login
        config.screen1.bind('<Return>', lambda event=None:
            btnn.invoke())


        # monitor app close
        def on_closing(event: object):
            """
            This function is used to monitor app close event,
            it binds the escape to it.


            :param event: checks keyboard interrupt.
            """


            sys.exit()
        # binding Escape key as shortcut to close app
        config.screen1.bind('<Escape>', on_closing)
        config.screen1.mainloop()


# securing code from import exploits
if __name__ == "__main__":
    # setting a main color theme
    colr = "#1c44a5"


    # checking OS to set GUI geometry
    chkos()


    # start program by calling login module
    main_page()
```

5.1.3 Script "frontend_api.py"

```python
import datetime
import requests
import hashlib
# SHA hash algorithms.

# LIST OF FUNCTIONS
# security(password)
# login (uid, passw)
# add_user (name, email_id, password, phone, perm)
# add_part (p_id, name, e_id, phone, events)
# add_event (name, date, time)
# get_events ()
# mark_entry (p_id, event)
# remove_event (name, date, time)
# get_report ()

url = " https://KKSJmp.pythonanywhere.com "
user_id = ""
upassw = ""
timeout = 5


def login(uid: str, password: str) -> int:
    """
    This function is used to verify user login credentials by
    communicating with the server.

    :param uid: User ID used to login.
    :param password: Password of the user.
    :return:"1"/"2" Password match, "0" Wrong password,
    "-1" User dose not exists.
    """
```

```python
    global upassw, user_id
    password = security(password)
    r = requests.post(url + "/login", json={"id": uid,
        "password": password}, timeout=timeout)
    if r.json()["body"]["permission"] >= 1:
        upassw = password
        user_id = uid
    return r.json()["body"]["permission"]


def add_user(name: str, email_id: str, password: str,
phone: str, perm: int) -> int:
    """
    Send request to add a new user to the server.

    :param name: Name of the user.
    :param email_id: E-mail ID of user.
    :param password: Password of user.
    :param phone: Phone number of user.
    :param perm: Level of access provided to the user
    ("2" for admin, "1" for user).
    :return: "1" Added, "0" Error (Already exists).
    """

    password = security(password)
    r = requests.post(url + "/add_user", json={"name": name,
        "email_id": email_id, "password": password, "phone":
        phone, "permission": perm, "uid": user_id,
        "upassw": upassw}, timeout=timeout)
    return r.json()["body"]["response"]
```

```python
def add_part(p_id: str, name: str, email_id: str, phone: str,
events: [str]) -> int:
    """
    Send request to add a new participant to the server.


    :param p_id: ID of participant.
    :param name: Name of participant.
    :param email_id: E-mail ID of participant.
    :param phone: Phone number of participant.
    :param events: List of events.
    :return: "0" some error OR no registration for this
    participant, "1" success, ("2"/"3"/"4") event (1/2/both)
    registered for this participant.
    """


    r = requests.post(url + "/add_part", json={"p_id": p_id,
        "name": name, "email_id": email_id, "phone": phone,
        "events": events, "uid": user_id, "upassw": upassw},
        timeout=timeout)
    return r.json()["body"]["response"]
```

```python
def add_event(name: str, date: str, time: str) -> int:
    """
    Send request to add a event to server.

    :param name: Name of event.
    :param date: Date of event (Format: YYYY-MM-DD).
    :param time: Time of event (Format: HH:MM).
    :return: "1" success, "0" event name exists.
    """

    try:
        datetime.datetime.strptime(date, '%Y-%m-%d')
    except ValueError:
        return 2
    try:
        datetime.datetime.strptime(time, '%H:%M')
    except ValueError:
        return 3
    time = time + ":00"
    r = requests.post(url + "/add_event", json={"name": name,
        "date": date, "time": time, "uid": user_id,
        "upassw": upassw}, timeout=timeout)
    return r.json()["body"]["response"]
```

```python
def get_events():
    """
    Send request to server for list of events.


    :return: List of events.
    """


    r = requests.get(url + "/get_events", timeout=timeout)
    return r.json()["body"]["response"]


def mark_entry(p_id, event):
    """
    Send request to server to mark participant with "p_id" as
    present in specified event "event".

    :param p_id: Participant ID.
    :param event: Event in which entry will be marked.
    :return: "0" Not Registered, "1" Registered, "2" Already
    entered.
    """

    r = requests.post(url + "/mark_entry", json={"p_id": p_id,
        "event": event, "uid": user_id, "upassw": upassw},
        timeout=timeout)
    return r.json()["body"]["response"]
```

```python
def remove_event(name, date, time):
    """
    Sends request to remove an event to the server.

    :param name: Name of the event.
    :param date: Date of the event (Format: YYYY-MM-DD).
    :param time: Time of the event (Format: HH:SS).
    :return: "1" success, "0" event participant registered,
    "4" wrong event details.
    """

    try:
        datetime.datetime.strptime(date, '%Y-%m-%d')
    except ValueError:
        return 2
    try:
        datetime.datetime.strptime(time, '%H:%M')
    except ValueError:
        return 3
    time = time + ":00"
    r = requests.post(url + "/remove_event", json={"name":
        name, "date": date, "time": time, "uid": user_id,
        "upassw": upassw}, timeout=timeout)
    return r.json()["body"]["response"]
```

```python
def get_report():
    """
    Send request to get the report of all the participants
    along with events they are registered in.

    :return: List of tuple containing registry of each
    participant.
    """

    r = requests.get(url + "/get_report", timeout=timeout)
    return r.json()["body"]["response"]


def security(password: str) -> str:
    """
    This function will hash the "password" and return the
    hash.

    :param password: String which will be hashed.
    :return: Hash of the string.
    """

    password = password[::-1]
    password = hashlib.sha224(password.encode()).hexdigest()
    password = password[::-1]
    password = hashlib.sha256(password.encode()).hexdigest()
    password = password[::-1]
    return password
```

5.2 Backend

5.2.1 Script "backend_api.py"

```python
from flask import Flask, request, jsonify, abort
from flask_cors import CORS
import manage_op as op


app = Flask(__name__)
CORS(app)


# LIST OF FUNCTIONS
# login
# add_user
# add_part
# add_event
# get_events
# mark_entry
# remove_event
# get_report


@app.route('/')
def hello_world():
    """
    It is demo function to test weather server is working or
    not.

    :return: A static string.
    """


    return 'Hello from Flask! This is a test site'
```

```python
@app.route('/login', methods=["Post"])
def login():
    """
    This function will verify credentials.
    It requires user id and hashed password.

    :return: "1"/"2" Password match, "0" Wrong password,
    "-1" User dose not exists in JSON format.
    """


    req_data = request.get_json()
    id = req_data["id"]
    passw = req_data["password"]

    # "1"/"2" Password match, "0" Wrong password,
    # "-1" User dose not exists
    permission = op.login(uid=id, passw=passw)

    return jsonify({
        "method": "POST",
        "headers": {
            "content-type": "application/json"
        },
        "body": {
            "permission": permission
        }
    })
```

```python
@app.route('/add_user', methods=["Post"])
def add_user():
    """
    Handles the request to add a user.
    It requires user details, their login credentials and
    operator authentication.

    :return: "1" if added, "0" Error (User already exists) in
    JSON format.
    """

    req_data = request.get_json()
    response = "Custom Response"

    e_id = req_data["email_id"]
    passw = req_data["password"]
    phone = req_data["phone"]
    name = req_data["name"]
    perm = req_data["permission"]
    uid = req_data["uid"]
    upassw = req_data["upassw"]

    if op.login(uid=uid, passw=upassw) == 2:
        # "1" if added, "0" if exists
        response = op.add_user(name=name, email_id=e_id,
                               phone=phone, perm=perm,
                               password=passw)
    else:
        abort(404)
```

```python
    return jsonify({
        "method": "POST",
        "headers": {
            "content-type": "application/json"
        },
        "body": {
            "response": response
        }
    })


@app.route('/add_part', methods=["Post"])
def add_part():
    """
    Handles the request to add a participant.
    It requires participant details, events list and operator
    authentication.

    :return: "0" some error / no registration for this
    participant, "1" success, ("2"/"3"/"4") event (1/2/both)
    registered for this participant in JSON format.
    """

    req_data = request.get_json()
    response = "Custom Response"
    e_id = req_data["email_id"]
    events = req_data["events"]
    p_id = req_data["p_id"]
    phone = req_data["phone"]
    name = req_data["name"]
    uid = req_data["uid"]
    upassw = req_data["upassw"]
```

```python
        if op.login(uid=uid, passw=upassw) >= 1:
            # "0" some error / no registration for this
            # participant, "1" success,
            # ("2"/"3"/"4") event (1/2/both) registered for
            # this participant
            response = op.add_part(p_id=p_id, name=name,
                                   email=e_id, phone=phone,
                                   events=events)
        else:
            abort(404)


        return jsonify({
            "method": "POST",
            "headers": {
                "content-type": "application/json"
            },
            "body": {
                "response": response
            }
        })


@app.route('/add_event', methods=["Post"])
def add_event():
    """
    Handles the request to add an event.
    It requires event details and operator authentication.


    :return: "1" success, "0" event name exists in JSON
    format.
    """
```

```python
req_data = request.get_json()
response = "Custom Response"


date = req_data["date"]
time = req_data["time"]
name = req_data["name"]
uid = req_data["uid"]
upassw = req_data["upassw"]


if op.login(uid=uid, passw=upassw) >= 1:
    # "1" success, "0" event name exists
    response = op.add_event(name=name, date=date,
                            time=time)
else:
    abort(404)


return jsonify({
    "method": "POST",
    "headers": {
        "content-type": "application/json"
    },
    "body": {
        "response": response
    }
})
```

```python
@app.route('/get_events')
def get_events():
    """
    Handles the request for list of events.


    :return: List of events in JSON format.
    """


    # list of tupple(event_id and name)
    response = op.get_events()


    return jsonify({
        "method": "POST",
        "headers": {
            "content-type": "application/json"
        },
        "body": {
            "response": response
        }
    })


@app.route('/mark_entry', methods=["Post"])
def mark_entry():
    """
    Handles the request to mark entry of a participant in a
    event.
    It requires participant ID, event name and operator
    authentication.


    :return: "0" Not Registered, "1" Registered, "2"
    Participant already entered in JSON format.
    """
```

```python
req_data = request.get_json()
response = "Custom Response"


p_id = req_data["p_id"]
event = req_data["event"]
uid = req_data["uid"]
upassw = req_data["upassw"]


if op.login(uid=uid, passw=upassw) >= 1:
    # "0" Not Registered, "1" Registered, "2" Entered
    response = op.mark_entry(p_id=p_id, event=event)
else:
    abort(404)


return jsonify({
    "method": "POST",
    "headers": {
        "content-type": "application/json"
    },
    "body": {
        "response": response
    }
})
```

```python
@app.route('/remove_event', methods=["Post"])
def remove_event():
    """
    Handles the request to remove an event.
    It requires event name and operator authentication.
    :return: "1" success, "0" event participant registered,
    "4" wrong event details in JSON format.
    """
    req_data = request.get_json()
    response = "Custom Response"
    date = req_data["date"]
    time = req_data["time"]
    name = req_data["name"]
    uid = req_data["uid"]
    upassw = req_data["upassw"]

    if op.login(uid=uid, passw=upassw) >= 1:
        # "1" success, "0" event participant registered,
        # "4" wrong event details
        response = op.remove_event(name=name, date=date,
                                   time=time)
    else:
        abort(404)

    return jsonify({
        "method": "POST",
        "headers": {
            "content-type": "application/json"
        },
        "body": {
            "response": response
        }
    })
```

```python
@app.route('/get_report')
def get_report():
    """
    Handles the request to remove an event.
    It requires event name and operator authentication.

    :return: List of participant and registered events in JSON
    format.
    """


    # list of tupple(event_id and name)
    response = op.get_report()


    return jsonify({
        "method": "POST",
        "headers": {
            "content-type": "application/json"
        },
        "body": {
            "response": response
        }
    })


if __name__ == "__main__":
    app.run()
```

5.2.2 Script "manage_op.py"

```python
import database as db_init
import db_operations as db

# LIST OF FUNCTIONS
# login (uid, passw)
# add_user (name, email_id, password, phone, perm)
# add_part (p_id, name, e_id, phone, events)
# event_registry (p_id, events)
# add_event (name, date, time)
# get_events ()
# mark_entry (p_id, event_id)
# remove_event (name, date, time)
# get_report ()


def login(uid: str, passw: str) -> int:
    """
    Process the login data for database operation.

    :param uid: User ID.
    :param passw: Hash of password of user.
    :return: "1"/"2" Password match, "0" Wrong password,
    "-1" User dose not exists.
    """

    try:
        users = db.get_user()
    except:
        db_init.db_init()
        users = db.get_user()
```

```python
    for i in users:
        if i[0] == uid:
            # User email exists
            if passw == i[1]:
                # "1"/"2" Password match
                return i[2]
            # "0" Wrong password
            return 0
    # "-1" User dose not exists
    return -1


def add_user(name: str, email_id: str, password: str,
phone: str, perm: int) -> int:
    """
    Request database to add new user data.

    :param name: Name of user.
    :param email_id: E-mail ID of user.
    :param password: Hash of password of new user.
    :param phone: Phone number of new user.
    :param perm: Permission level provided to user.
    :return: "1" if added, "0" Error (user already exists).
    """

    success = db.add_user(name=name, email=email_id,
                          phone=phone, passw=password,
                          perm=perm)
    # "1" if added, "0" if exists
    return success
```

```python
def add_part(p_id: str, name: str, email: str, phone: str, eve
nts: str) -> int:
    """
    Process participant data to add into database.

    :param p_id: Participant ID.
    :param name: Participant name.
    :param email: Participant E-mail ID.
    :param phone: Participant phone number.
    :param events: List of events in which participant needs
    to register.
    :return: "0" some error, "1" success, ("2"/"3"/"4") event
    (1/2/both) registered for this participant.
    """

    if name == "":
        resp = db.check_part(p_id)
        if resp == 0:
            # No registration for this participant
            return 0
    else:
        # Registering participant
        status = db.add_participant(
            p_id=p_id, name=name, email=email, phone=phone)
        if status == 0:
            # User exists
            p_id = db.get_pid(phone)

    # "0" some error, "1" success, ("2"/"3"/"4") event
    # (1/2/both) registered for this participant
    return event_registry(p_id=p_id, events=events)
```

110

```python
def event_registry(p_id: str, events: str) -> int:
    """

    Process data to register participant in each event.


    :param p_id: Participant ID.
    :param events: List of events in which participant needs
    to register.
    :return: "0" some error, "1" success, ("2"/"3"/"4") event
    (1/2/both) registered for this participant.
    """


    ex_event = []
    for i in events:
        # Registering participant in selected events
        e_id = db.get_event_id(i)
        resp = db.get_reg(p_id=p_id, event_id=e_id[0][0])
        if resp == 0:
            resp = db.add_reg(p_id=p_id, event_id=e_id[0][0])
            if resp == 0:
                return 0
            ex_event.append(0)
        else:
            ex_event.append(1)


    # "0" some error, "1" success, ("2"/"3"/"4") event
    # (1/2/both) registered for this participant
    if len(ex_event) == 2:
        if ex_event[0] == 1 and ex_event[1] == 1:
            return 4
        elif ex_event[0] == 1:
            return 2
        elif ex_event[1] == 1:
            return 3
```

```python
        elif ex_event[0] == 1:
            return 4
        return 1


def add_event(name: str, date: str, time: str) -> int:
    """
    Request database to add new event.

    :param name: Name of event.
    :param date: Date of event.
    :param time: Time of event.
    :return: "1" success, "0" event name exists.
    """

    resp = db.add_event(name=name, date=date, time=time)
    # "1" success, "0" event name exists
    return resp


def get_events() -> [str]:
    """
    Request database for list of events.

    :return: List of events.
    """

    events = db.get_events()
    # list of tupple(name)
    return events
```

```python
def mark_entry(p_id: str, event: str) -> int:
    """
    Mark entry of a participant into to the event.

    :param p_id: Participant ID.
    :param event: Event name.
    :return: "0" Not Registered, "1" Registered, "2" Entered.
    """

    event_id = db.get_event_id(event)

    resp = db.get_reg(p_id=p_id, event_id=event_id[0][0])
    if resp == 1:
        db.mark_entry(p_id=p_id, event_id=event_id[0][0])
    # "0" Not Registered, "1" Registered, "2" Entered
    return resp


def remove_event(name: str, date: str, time: str) -> int:
    """
    Remove an event from database.

    :param name: Name of event.
    :param date: Date of event.
    :param time: Time of event.
    :return: "1" success, "0" event participant registered,
    "4" wrong event details.
    """

    resp = db.remove_event(name=name, date=date, time=time)
    # "1" success, "0" event participant registered, "4" wrong
    # event details
    return resp
```

```python
def get_report() -> [[str]]:
    """
    Return the list of participant data and events they are
    registered in.

    :return: List of participant data.
    """

    return db.get_report()
```

5.2.3 Script "db_operations.py"

```python
import mysql.connector


# LIST OF FUNCTIONS
# sql_connect ()
# add_event (name, date, time)
# add_participant (p_id, name, email, phone)
# add_user (name, email, phone, passw, perm)
# add_reg (p_id, event_id)
# get_event_id (name)
# get_events ()
# get_user ()
# get_reg (p_id, event_id)
# get_pid (phone)
# remove_event (name, date, time)
# get_report ()
# mark_entry (p_id, event_id)
# check_part (p_id)


def sql_connect():
    """
    Make a connection to the database.

    :return: database name, Cursor object of database,
    connector of database.
    """

    mydb = mysql.connector.connect(
        host="KKSJmp.mysql.pythonanywhere-services.com",
        user="KKSJmp",
        password="********"
    )
```

```python
    mycursor = mydb.cursor()
    db_name = "minor_db"
    return db_name, mycursor, mydb


def add_event(name: str, date: str, time: str) -> int:
    """
    Command database to add an event.

    :param name: Name of event.
    :param date: Date of event.
    :param time: Time of event.
    :return: "1" success, "0" event name exists.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    # Date "yyyy-mm-dd" Time "hh:mm:ss"
    try:
        sql = "INSERT INTO events (name, date, time)
            VALUES (%s, %s, %s)"
        val = (name, date, time)
        mycursor.execute(sql, val)

        mydb.commit()
        return 1
    except:
        # Name already exists
        return 0
```

```python
def add_participant(p_id: str, name: str, email: str,
phone: str) -> int:
    """
    Command database to add a new participant.

    :param p_id: Participant ID.
    :param name: Participant name.
    :param email: Participant E-mail ID.
    :param phone: Participant phone number.
    :return: "0" some error, "1" success.
    """


    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    try:
        sql = "INSERT INTO participants (p_id, name, email_id,
            phone) VALUES (%s, %s, %s, %s)"
        val = (p_id, name, email, phone)
        mycursor.execute(sql, val)

        mydb.commit()
        return 1
    except:
        # Participant already exists
        return 0
```

```python
def add_user(name: str, email: str, phone: str, passw: str,
perm: int) -> int:
    """
    Command database to add a new user.

    :param name: Name of user.
    :param email: E-mail ID of user.
    :param passw: Hash of password of new user.
    :param phone: Phone number of new user.
    :param perm: Permission level provided to user.
    :return: "1" if added, "0" Error (user already exists).
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)
    phone = str(phone)
    passw = str(passw)

    try:
        sql = "INSERT INTO user (name, email_id, phone,
            password, permission) VALUES (%s, %s, %s, %s, %s)"
        val = (name, email, phone, passw, perm)
        mycursor.execute(sql, val)

        mydb.commit()
        return 1
    except:
        # E-mail already exists
        return 0
```

```python
def add_reg(p_id: str, event_id: str) -> int:
    """
    Command database to register a participant in an event.

    :param p_id: Participant ID.
    :param event_id: Event ID.
    :return: "0" some error, "1" success.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    try:
        sql = "INSERT INTO registration (p_id, event_id,
            present) VALUES (%s, %s, %s)"
        val = (p_id, event_id, 1)
        mycursor.execute(sql, val)
        mydb.commit()
        return 1
    except:
        # p_id or event_id incorrect
        return 0
```

```python
def get_event_id(name: str) -> int:
    """
    Retrieve event ID from database using the event name.

    :param name: Event name.
    :return: Event ID.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    sql = "SELECT event_id FROM events WHERE name = \""
        + name + "\""
    mycursor.execute(sql)
    myresult = mycursor.fetchall()
    return myresult


def get_events() -> [str]:
    """
    Retrieve List of events from database.

    :return: List of events.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    mycursor.execute("SELECT name FROM events")
    myresult = mycursor.fetchall()
    return myresult
```

120

```python
def get_user() -> [[str]]:
    """
    Retrieve e-mail id, password and permission level of user
    from database.

    :return: List of e-mail id, password and permission level
    of all users.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    mycursor.execute("SELECT email_id, password, permission
    FROM user")
    myresult = mycursor.fetchall()
    return myresult


def get_reg(p_id: str, event_id: str) -> int:
    """
    Retrieve entry detail of a participant in a particular
    event.

    :param p_id: Participant ID.
    :param event_id: Event ID.
    :return: "1" Not Entered, "2" Entered, "0" Dose not exist.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)
    event_id = str(event_id)
```

```python
    try:
        sql = "SELECT present FROM registration WHERE p_id = \
            %s AND event_id = %s"
        val = (p_id, event_id)
        mycursor.execute(sql, val)
        myresult = mycursor.fetchall()
        return myresult[0][0]
    except:
        # Dose not exist
        return 0


def get_pid(phone: str) -> str:
    """
    Retrieve participant id of a participant from phone number

    :param phone: Phone number of participant.
    :return: Participant ID.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)
    phone = str(phone)

    mycursor.execute("SELECT p_id FROM participants WHERE
    phone = " + phone)
    myresult = mycursor.fetchall()
    return myresult[0][0]
```

```python
def remove_event(name: str, date: str, time: str) -> int:
    """
    Command database to delete an event.

    :param name: Name of event.
    :param date: Date of event.
    :param time: Time of event.
    :return: "1" success, "0" event participant registered,
    "4" wrong event details.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    # Date "yyyy-mm-dd" Time "hh:mm:ss"
    try:
        sql = "DELETE FROM events WHERE name = %s AND date = \
            %s AND time = %s"
        val = (name, date, time)
        mycursor.execute(sql, val)
        mydb.commit()

        if mycursor.rowcount == 0:
            # no change
            return 4
        return 1
    except:
        # Someone is registered
        return 0
```

```python
def get_report() -> [[str]]:
    """
    Retrieve participant details and events they registered in
    from database.

    :return: List of participant details.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    sql = "SELECT `" + db_name + "`.`participants`.*,
        GROUP_CONCAT(`" + db_name + "`.`events`.name, `" +
        db_name + "`.`registration`.present) as \"events\"
        FROM ((`" + db_name + "`.`participants` INNER JOIN `"
        + db_name + "`.`registration` ON `" + db_name +
        "`.`participants`.p_id = `" + db_name +
        "`.`registration`.p_id) INNER JOIN `" + db_name +
        "`.`events` ON `" + db_name + "`.`events`.event_id
        = `" + db_name + "`.`registration`.event_id)
        group by p_id;"
    mycursor.execute(sql)
    myresult = mycursor.fetchall()
    return myresult


def mark_entry(p_id: str, event_id: str):
    """
    Command database to mark a participant entry in a event.

    :param p_id: Participant ID.
    :param event_id: Event ID.
    """
```

```python
    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)


    sql = "UPDATE `" + db_name + "`.`registration` SET
        `present` = '2' WHERE p_id = %s AND event_id = %s"
    val = (p_id, event_id)
    mycursor.execute(sql, val)
    mydb.commit()


def check_part(p_id: str) -> int:
    """
    Retrieve all details of participant from database.

    :param p_id: Participant ID.
    :return: List of details of participant.
    """

    db_name, mycursor, mydb = sql_connect()
    mycursor.execute("USE " + db_name)

    sql = "SELECT * FROM participants WHERE p_id = \"" +
        p_id + "\""
    mycursor.execute(sql)
    mycursor.fetchall()

    return mycursor.rowcount
```

5.2.4 Script "database.py"

```python
import mysql.connector
import hashlib
# SHA hash algorithms.


def db_init():
    """
    This function initializes the database and all the tables
    required in it.
    First Administrator is automatically added.
    Tables: User, Participant, Events, Registration.
    """

    mydb = mysql.connector.connect(

        host="KKSJmp.mysql.pythonanywhere-services.com",
        user="KKSJmp",
        password="********"
    )

    db_name = 'minor_db'

    mycursor = mydb.cursor()

    mycursor.execute("SHOW DATABASES")

    # Creating Database
    ls = []
    for x in mycursor:
        ls.append(x[0])
```

```python
if db_name in ls:
    # DB present
    mycursor.execute("USE " + db_name)
else:
    # new db
    mycursor.execute("CREATE DATABASE " + db_name)
    # DB created
    mycursor.execute("USE " + db_name)


# Creating User Table
mycursor.execute("SHOW TABLES")
user_table = "user"


ls = []
for x in mycursor:
    ls.append(x[0])


if user_table not in ls:
    # User not present
    mycursor.execute("CREATE TABLE `" + db_name +
    "`.`user` (`User_id` INT AUTO_INCREMENT NOT NULL,
    `name` VARCHAR(50) NOT NULL,`Email_id` VARCHAR(45)
    NOT NULL,`Phone` VARCHAR(15) NOT NULL,
    `Password` VARCHAR(100) NOT NULL,
    `Permission` INT NOT NULL,PRIMARY KEY (`User_id`),
    UNIQUE INDEX `Email_id_UNIQUE` (`Email_id` ASC),
    UNIQUE INDEX `User_id_UNIQUE` (`User_id` ASC));"
    )

    sql = "INSERT INTO user (User_id, name, Email_id,
    Phone, Password, Permission)
    VALUES (%s, %s, %s, %s, %s, %s)"
```

```python
        password = "*********"
        password = password[::-1]
        password=hashlib.sha224(password.encode()).hexdigest()
        password = password[::-1]
        password=hashlib.sha256(password.encode()).hexdigest()
        password = password[::-1]
        val = (1, "Admin", "admin@****.com", 1234567890,
        password, 2)
        mycursor.execute(sql, val)
        mydb.commit()

# Creating Participant Table
mycursor.execute("SHOW TABLES")
part_table = "participants"


ls = []
for x in mycursor:
    ls.append(x[0])


if part_table not in ls:
    # part not present
    mycursor.execute("CREATE TABLE `" + db_name +
    "`.`participants` (`p_id` VARCHAR(50) NOT NULL,
    `name` VARCHAR(50) NOT NULL,`Email_id` VARCHAR(45)
    NOT NULL,`Phone` VARCHAR(15) NOT NULL,
    PRIMARY KEY (`p_id`), UNIQUE INDEX `phone_UNIQUE`
    (`phone` ASC),UNIQUE INDEX `idparticipants_UNIQUE`
    (`p_id` ASC));"
    )
```

```python
# Creating Events Table
mycursor.execute("SHOW TABLES")
event_table = "events"


ls = []
for x in mycursor:
    ls.append(x[0])


if event_table not in ls:
    # event not present
    mycursor.execute("CREATE TABLE `" + db_name +
    "`.`events` (`event_id` INT NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(50) NOT NULL,`date` DATE NOT NULL,
    `time` TIME NOT NULL, PRIMARY KEY (`event_id`),
    UNIQUE INDEX `name_UNIQUE` (`name` ASC),
    UNIQUE INDEX `event_id_UNIQUE` (`event_id` ASC));"
    )


# Creating Registration Table
mycursor.execute("SHOW TABLES")
reg_table = "registration"


ls = []
for x in mycursor:
    ls.append(x[0])
```

```python
if reg_table not in ls:
    # reg not present
    mycursor.execute("CREATE TABLE `" + db_name +
    "`.`registration` (`r_id` INT NOT NULL AUTO_INCREMENT,
    `p_id` VARCHAR(50) NOT NULL,`event_id` INT NOT NULL,
    `present` INT NOT NULL, PRIMARY KEY (`r_id`),
    UNIQUE INDEX `r_id_UNIQUE` (`r_id` ASC),
    INDEX `p_id_idx` (`p_id` ASC),
    INDEX `event_id_idx` (`event_id` ASC),
    CONSTRAINT `p_id`  FOREIGN KEY (`p_id`)  REFERENCES
    `" + db_name + "`.`participants` (`p_id`)
    ON DELETE NO ACTION  ON UPDATE NO ACTION,
    CONSTRAINT `event_id`  FOREIGN KEY (`event_id`)
    REFERENCES `" + db_name + "`.`events` (`event_id`)
    ON DELETE NO ACTION  ON UPDATE NO ACTION);"
    )
```

5.2.5 WSGI configuration file: kksjminorproject_pythonanywhere_com_wsgi.py

```python
# This file contains the WSGI configuration required
# to serve up your web application at
# http://<your-username>.pythonanywhere.com/
# It works by setting the variable 'application' to a
# WSGI handler of some description.
#
# The below has been auto-generated for your Flask project

import sys

# add your project directory to the sys.path
project_home = '/home/KKSJmp/mysite'
if project_home not in sys.path:
    sys.path = [project_home] + sys.path

# import flask app but need to call it "application" for
# WSGI to work
from backend_api import app as application  # noqa
```

5.3 Database

5.3.1 Database Script

```
-- Show version of mysql components
SHOW VARIABLES LIKE "%version%";


-- DEVELOPMENT COMPONENTS
-- +------------------------+------------------------------+
-- | Variable_name          | Value                        |
-- +------------------------+------------------------------+
-- | immediate_server_version| 999999                      |
-- | innodb_version         | 8.0.19                       |
-- | original_server_version | 999999                      |
-- | protocol_version       | 10                           |
-- | slave_type_conversions |                              |
-- | tls_version            | TLSv1,TLSv1.1,TLSv1.2,TLSv1.3 |
-- | version                | 8.0.19                       |
-- | version_comment        | MySQL Community Server - GPL |
-- | version_compile_machine | x86_64                      |
-- | version_compile_os     | Win64                        |
-- | version_compile_zlib   | 1.2.11                       |
-- +------------------------+------------------------------+

-- PRODUCTION COMPONENTS
-- +------------------------+--------------------+
-- | Variable_name          | Value              |
-- +------------------------+--------------------+
-- | innodb_version         | 5.6.48             |
-- | protocol_version       | 10                 |
-- | slave_type_conversions |                    |
-- | version                | 5.6.48-log         |
-- | version_comment        | Source distribution |
-- | version_compile_machine | x86_64            |
-- | version_compile_os     | Linux              |
-- +------------------------+--------------------+


-- Create a database.
CREATE DATABASE minor_DB;


-- Show all databases.
SHOW DATABASES;

```

```sql
-- Use "minor_db" database.
USE minor_DB;


-- Create user table
-- It is used to store user details.
CREATE TABLE `user` (
    `User_id` INT AUTO_INCREMENT NOT NULL,
    `name` VARCHAR(50) NOT NULL,
    `Email_id` VARCHAR(45) NOT NULL,
    `Phone` VARCHAR(15) NOT NULL,
    `Password` VARCHAR(100) NOT NULL,
    `Permission` INT NOT NULL,
    PRIMARY KEY (`User_id`),
    UNIQUE INDEX `Email_id_UNIQUE` (`Email_id` ASC),
    UNIQUE INDEX `User_id_UNIQUE` (`User_id` ASC)
);


-- Display list of tables.
SHOW TABLES;


-- Dispaly data of all tables.
SELECT * FROM user;
SELECT * FROM participants;
SELECT * FROM events;
SELECT * FROM registration;


-- Display login credentials of all users.
SELECT email_id, password, permission FROM user;
```

133

```sql
-- Display details of participants and events they are
registered in.
SELECT `participants`.*, GROUP_CONCAT(`events`.name,
`registration`.present) as "events" FROM ((`participants`
INNER JOIN `registration` ON
`participants`.p_id = `registration`.p_id)
INNER JOIN `events` ON
`events`.event_id = `registration`.event_id)
group by p_id;


-- Create participants table
-- It is used to store participants details.
CREATE TABLE `participants` (
    `p_id` VARCHAR(50) NOT NULL,
    `name` VARCHAR(50) NOT NULL,
    `Email_id` VARCHAR(45) NOT NULL,
    `Phone` VARCHAR(15) NOT NULL,
    PRIMARY KEY (`p_id`),
    UNIQUE INDEX `phone_UNIQUE` (`phone` ASC),
    UNIQUE INDEX `idparticipants_UNIQUE` (`p_id` ASC)
);


-- Create events table
-- It is used to store event details.
CREATE TABLE `events` (
    `event_id` INT NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(50) NOT NULL,
    `date` DATE NOT NULL,
    `time` TIME NOT NULL,
    PRIMARY KEY (`event_id`),
    UNIQUE INDEX `name_UNIQUE` (`name` ASC),
    UNIQUE INDEX `event_id_UNIQUE` (`event_id` ASC)
);
```

```sql
-- Create registration table
-- It is used to store participant ID and event ID for each
registration along with entry of participants.
CREATE TABLE `registration` (
    `r_id` INT NOT NULL AUTO_INCREMENT,
    `p_id` VARCHAR(50) NOT NULL,
    `event_id` INT NOT NULL,
    `present` INT NOT NULL,
    PRIMARY KEY (`r_id`),
    UNIQUE INDEX `r_id_UNIQUE` (`r_id` ASC),
    INDEX `p_id_idx` (`p_id` ASC),
    INDEX `event_id_idx` (`event_id` ASC),
    CONSTRAINT `p_id`
        FOREIGN KEY (`p_id`)
        REFERENCES `minor_db`.`participants` (`p_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    CONSTRAINT `event_id`
        FOREIGN KEY (`event_id`)
        REFERENCES `minor_db`.`events` (`event_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

6. System Implementation

6.1 Login

- Opening/starting screen of login
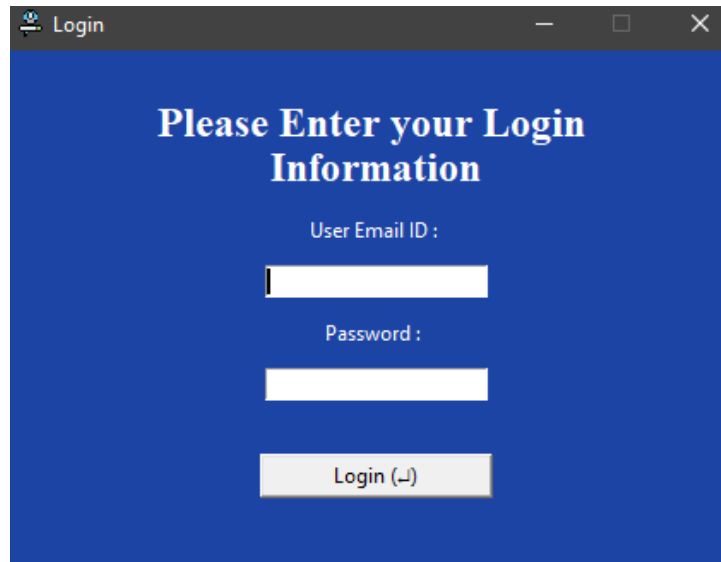
Figure 12: Login Screen

- Login attempt without internet
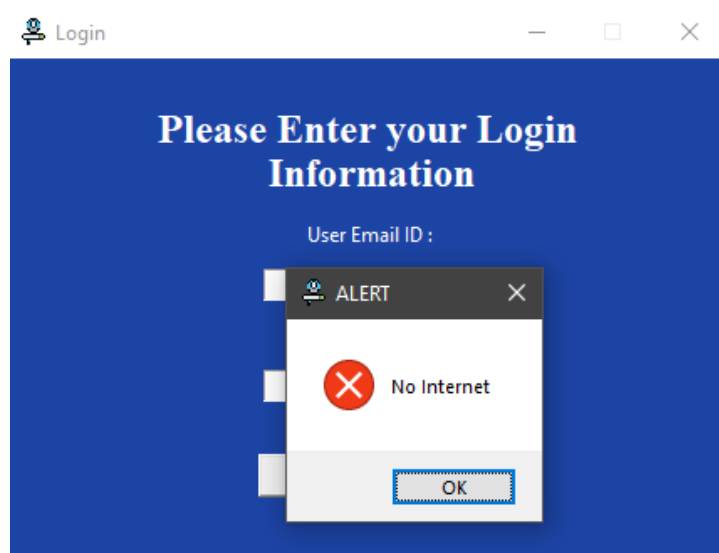
Figure 13: Login Server Error

- Login attempt with fields incomplete
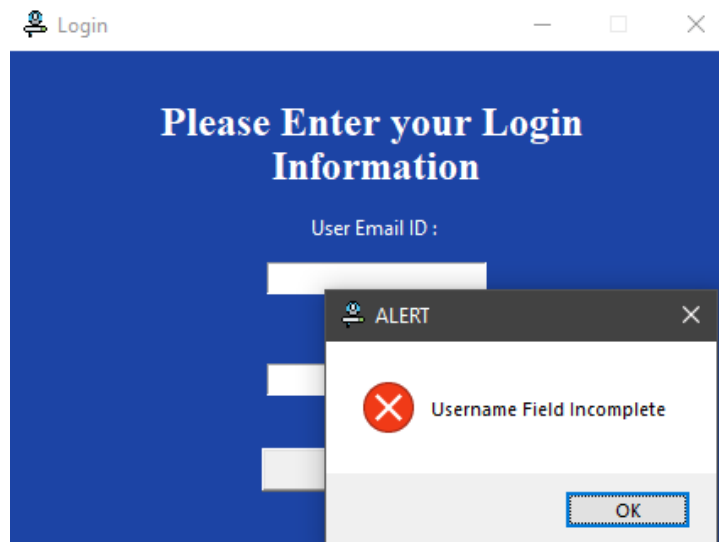
Figure 14: Login Username Field Incomplete
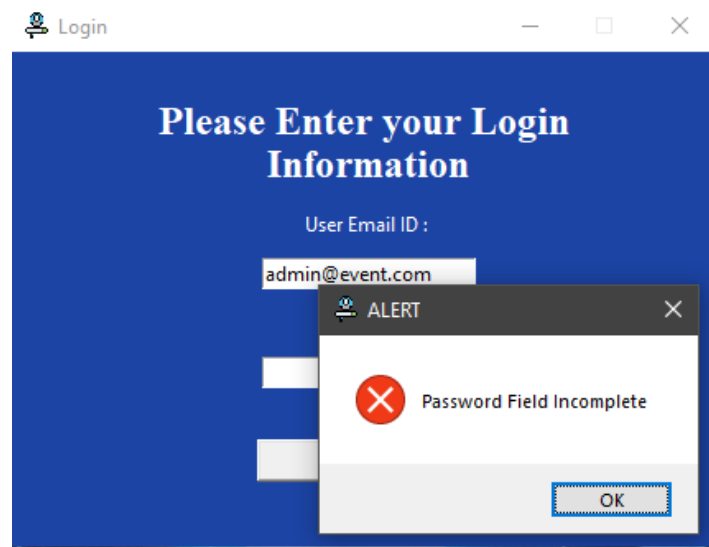
Figure 15: Login Password Field Incomplete

- Login attempt with invalid credentials
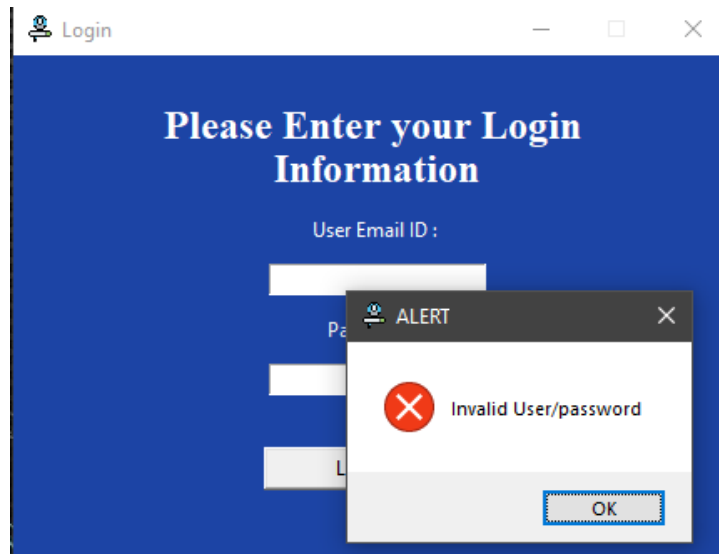
Figure 16: Login Invalid Credentials

138

6.2 Login Success

- Admin login success
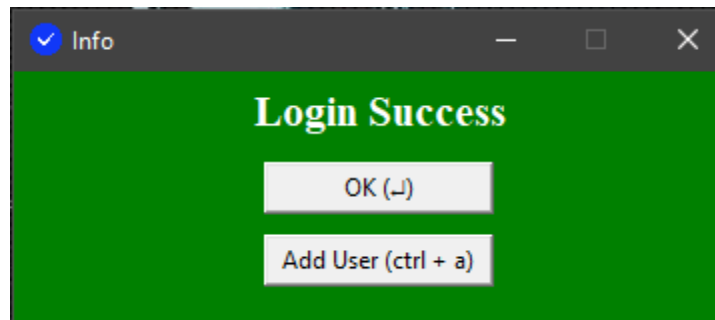
Figure 17: Admin Login Success
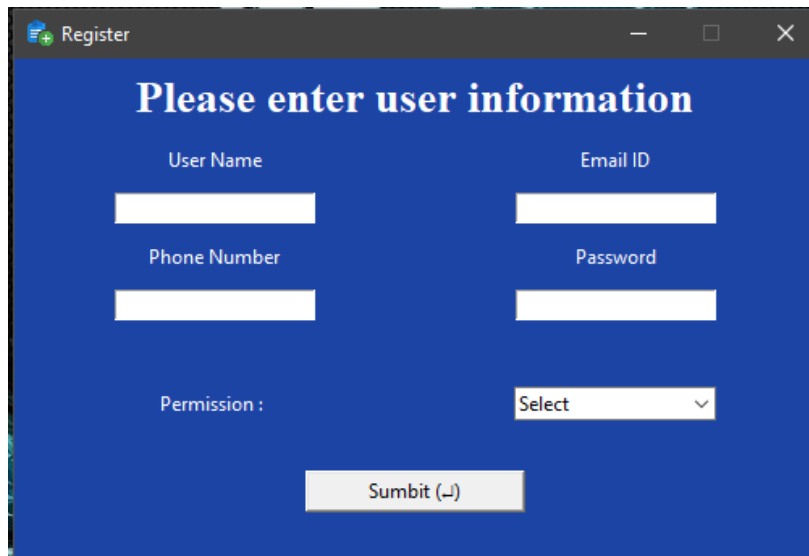
- User login success
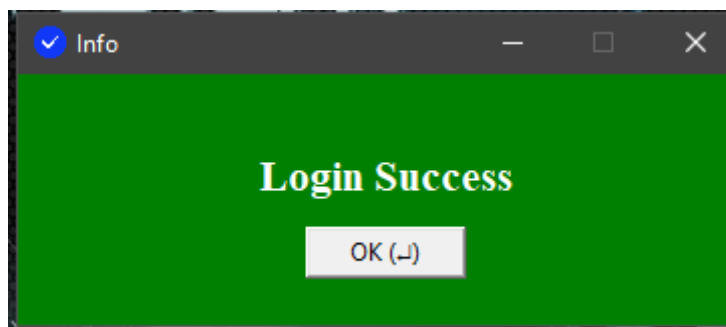
Figure 18: User Login Success

6.3 User Registration
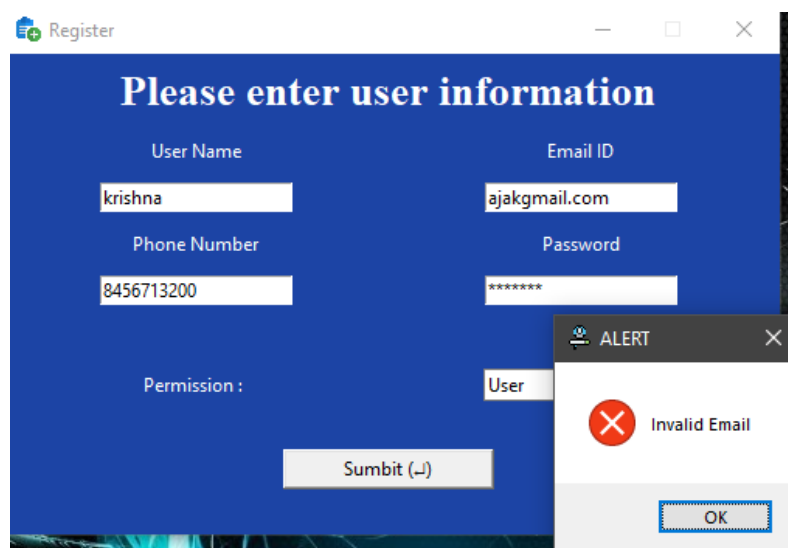
- Add user (admin feature)

Figure 19: Add User

- Attempt to adding user with invalid email ID

Figure 20: Add User Invalid E-mail

140

- Attempt to adding user with invalid phone number
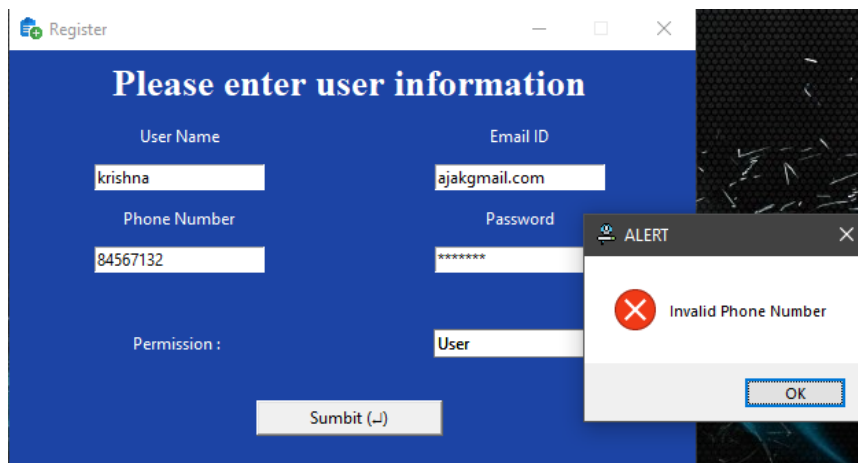
Figure 21: Add User Invalid Phone Number

- Attempt to adding user without selecting rights
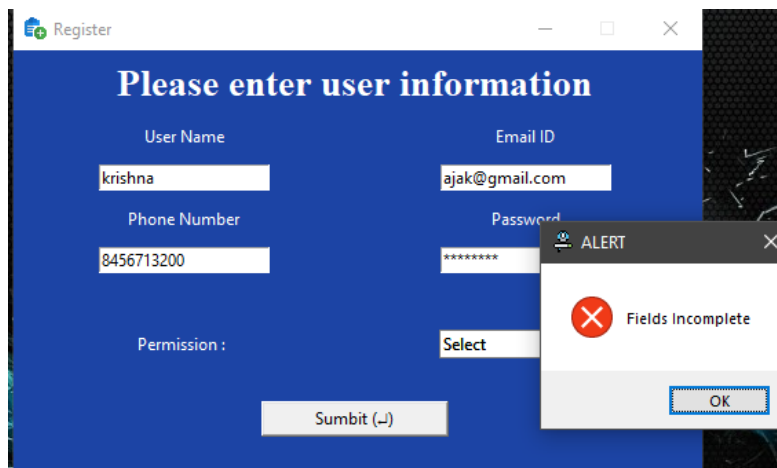
Figure 22: Add User Fields Incomplete

141

- Attempt to adding user without strong password
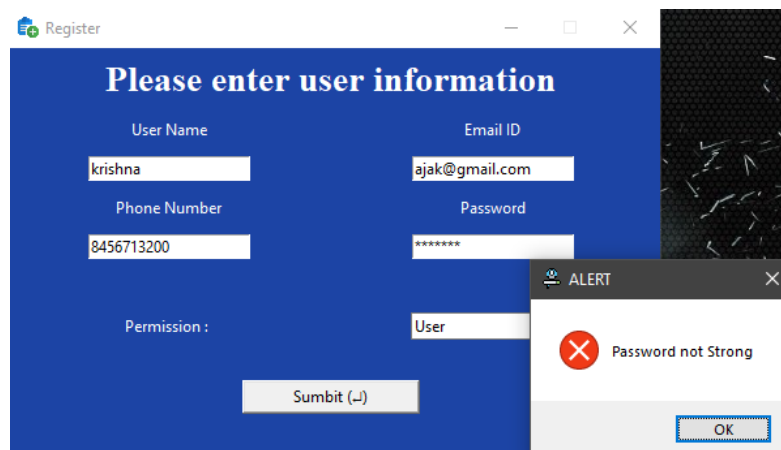
Figure 23: Add User Invalid Password

- Attempt to re-add existing user
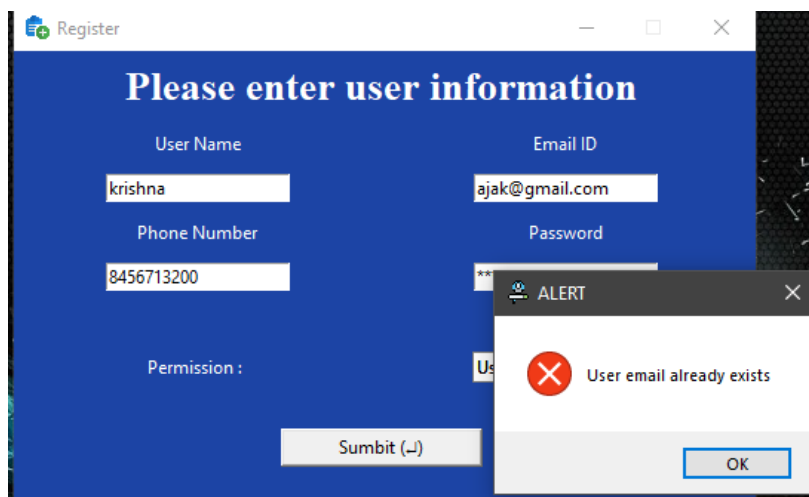
Figure 24: Add User Error-User Exists

- User added successful
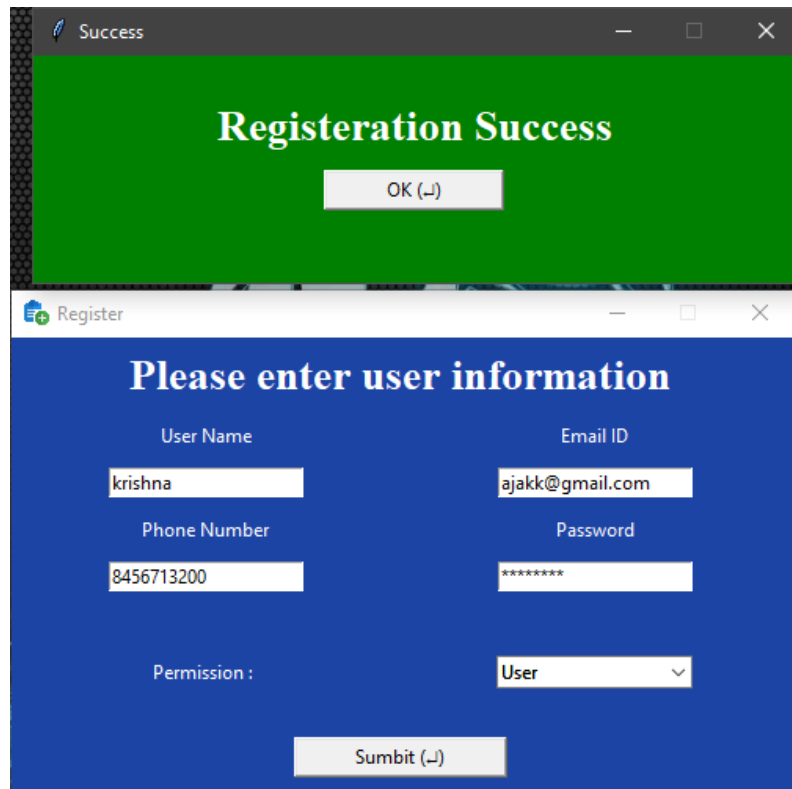
Figure 25: Add User Successful

6.4 Main Window

Figure 26: Main Window

6.5 Event Management Window

- Event management window
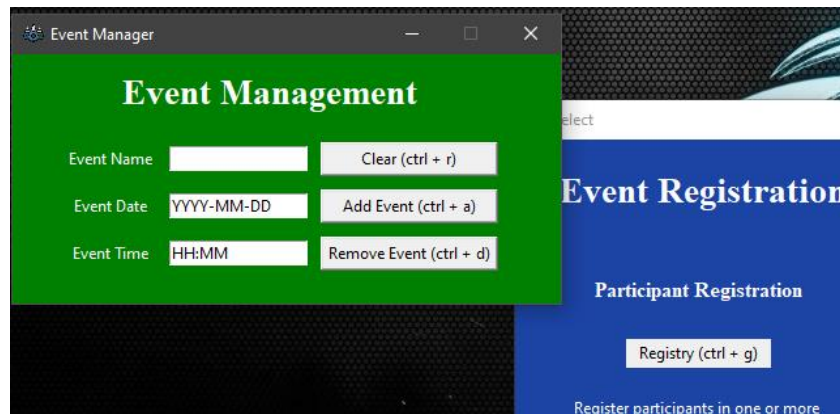
Figure 27: Event Management Window

- Attempt to adding event with incorrect date format
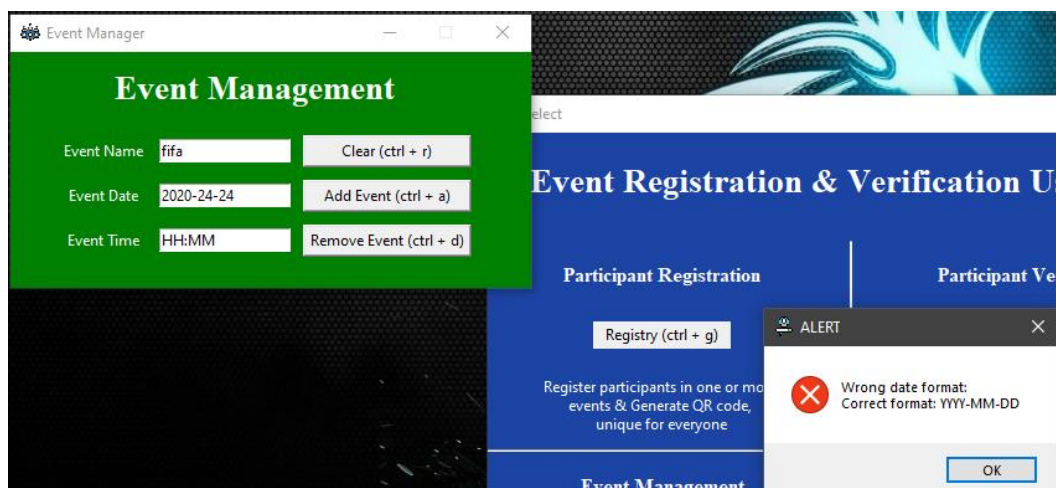
Figure 28: Wrong Date Format

- Attempt to adding event with incorrect time format
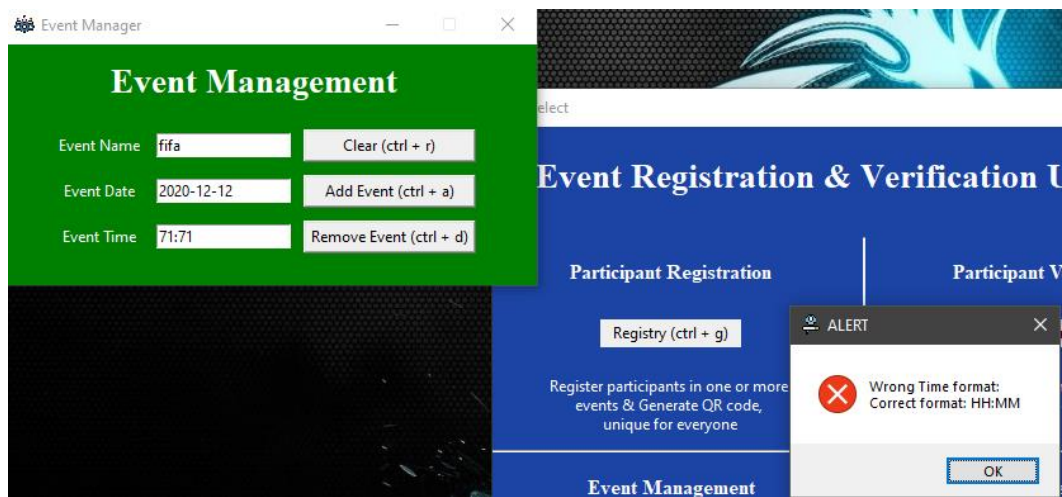
Figure 29: Wrong Time Format

- Event Added Successfully
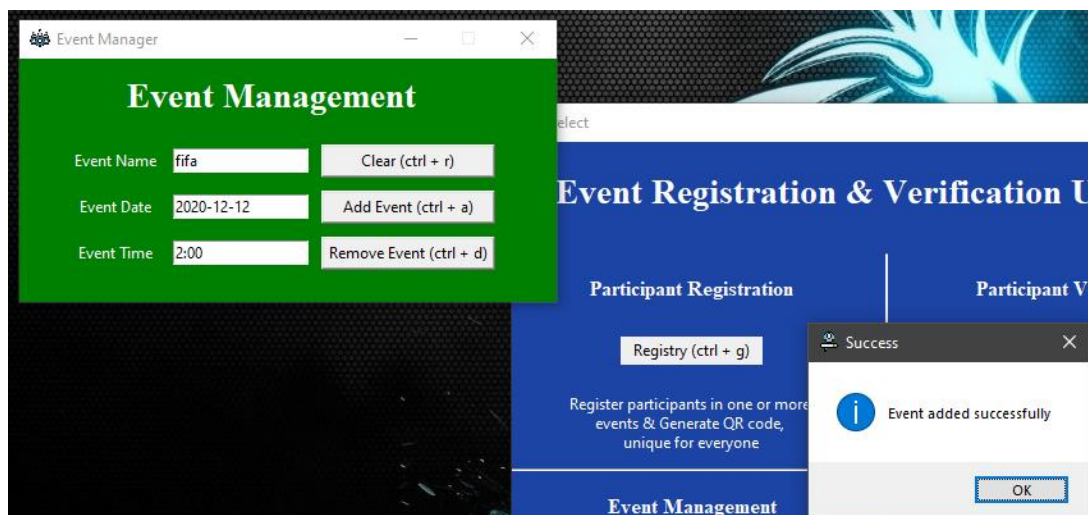
Figure 30: Event Added Successfully

- Attempt to re-add existing event
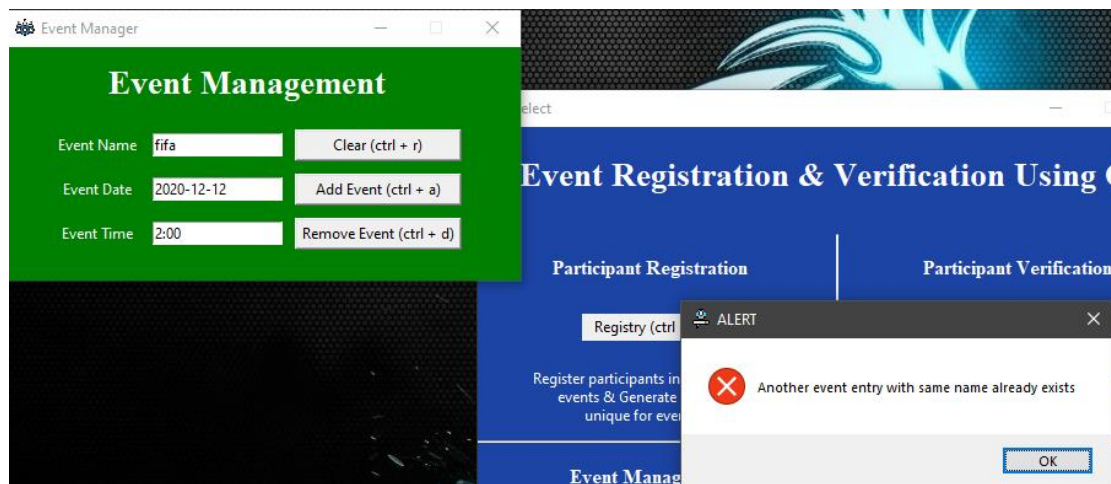
Figure 31: Error Event Already Exists

- Removing existing event
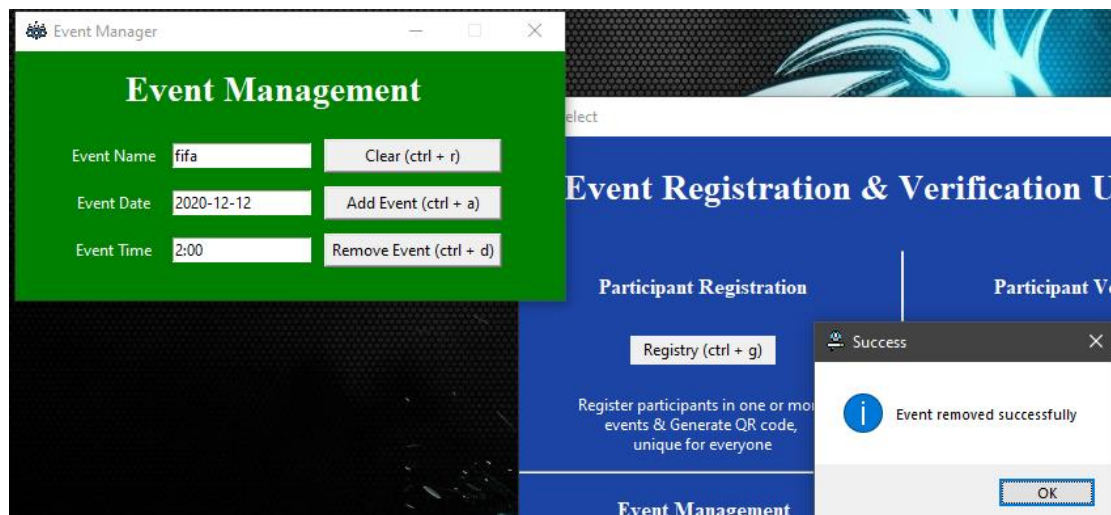
Figure 32: Event Removed Successfully

- Attempt to remove non-existing event
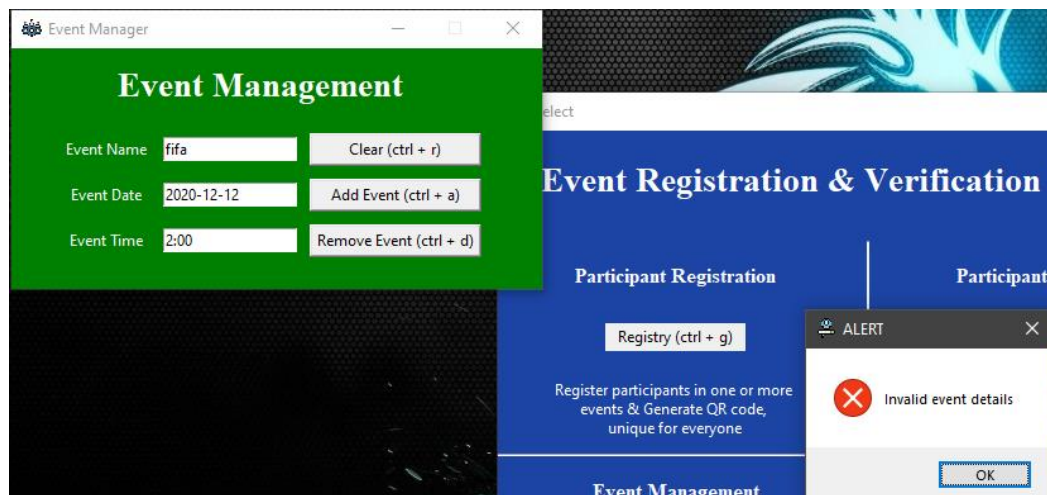
Figure 33: Error Event does not exist

- Attempt to remove event with participants registered in it
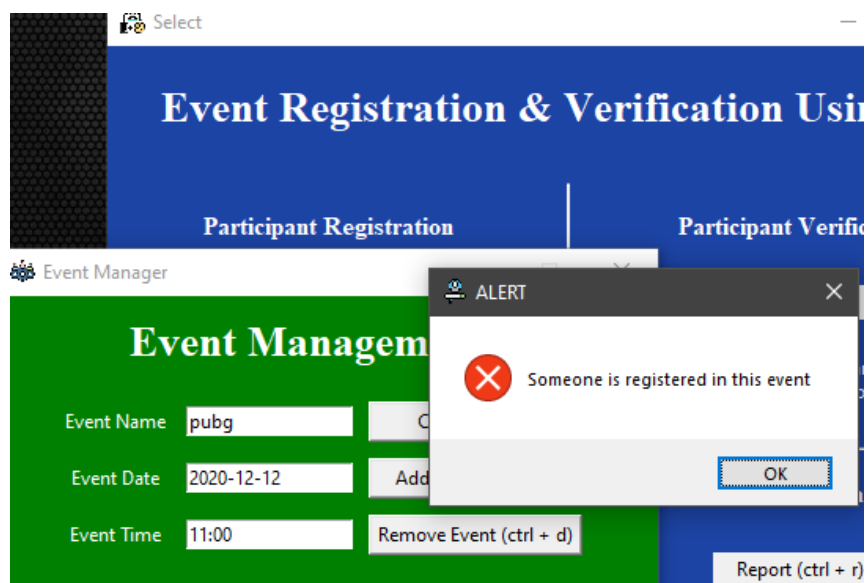
Figure 34: Error-Event cannot be removed
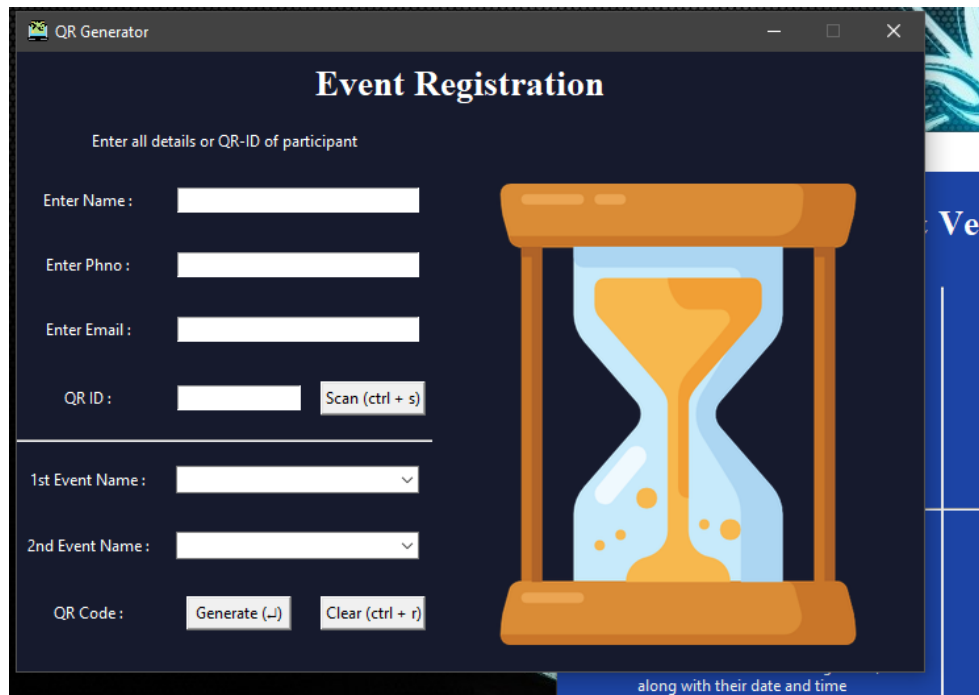
- Clear fields
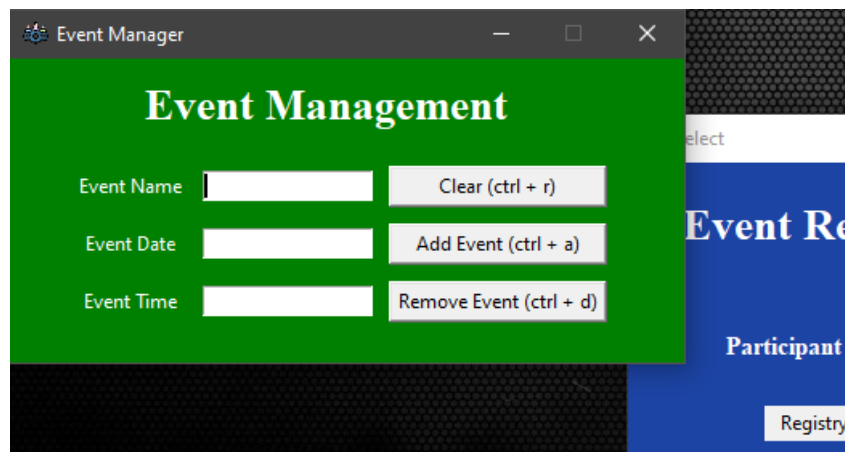
Figure 35: Cleared Event Fields

6.6 Participant Registration Window

- Registration / QR Generation window

Figure 36: Participant Registration Window

- Attempt to adding participant with invalid phone number

Figure 37: Invalid Participant Phone Number

- Attempt to adding participant with invalid email ID
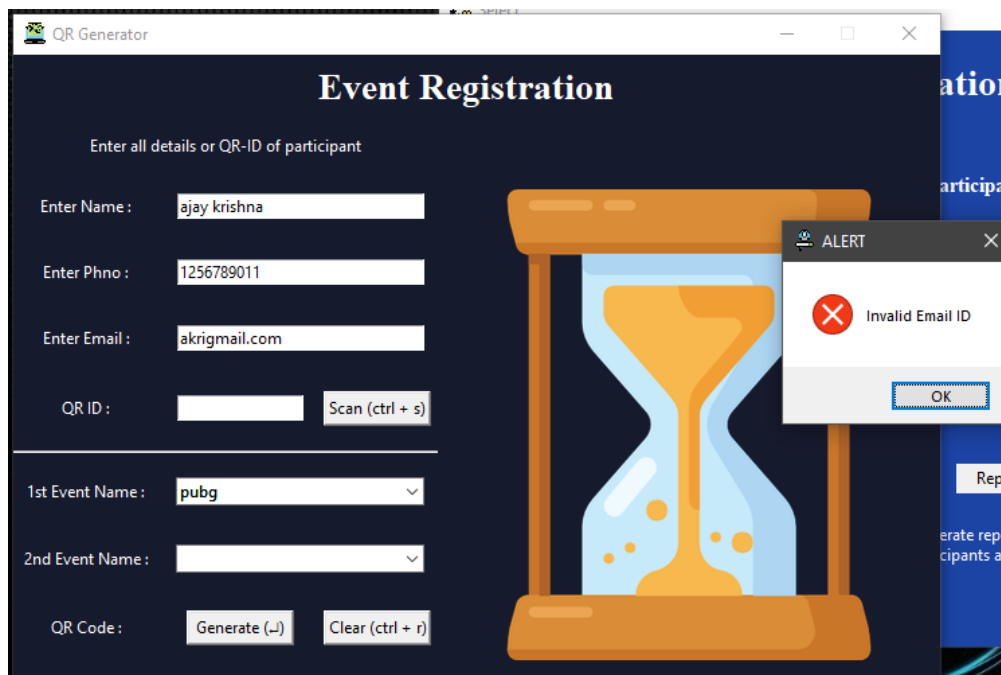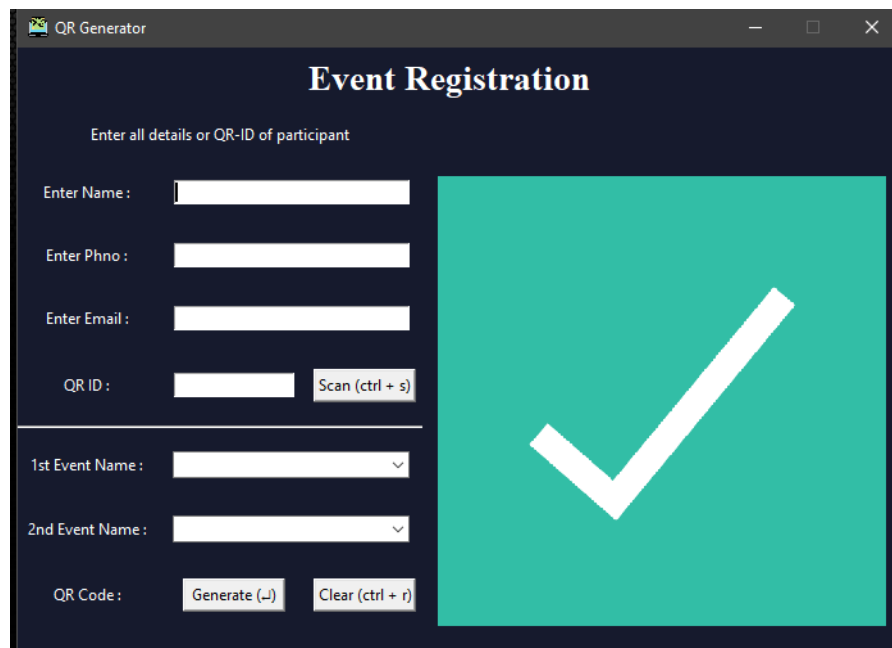
Figure 38: Invalid Participant E-mail ID

- Adding participants

Figure 39: Participant Added Successfully

151
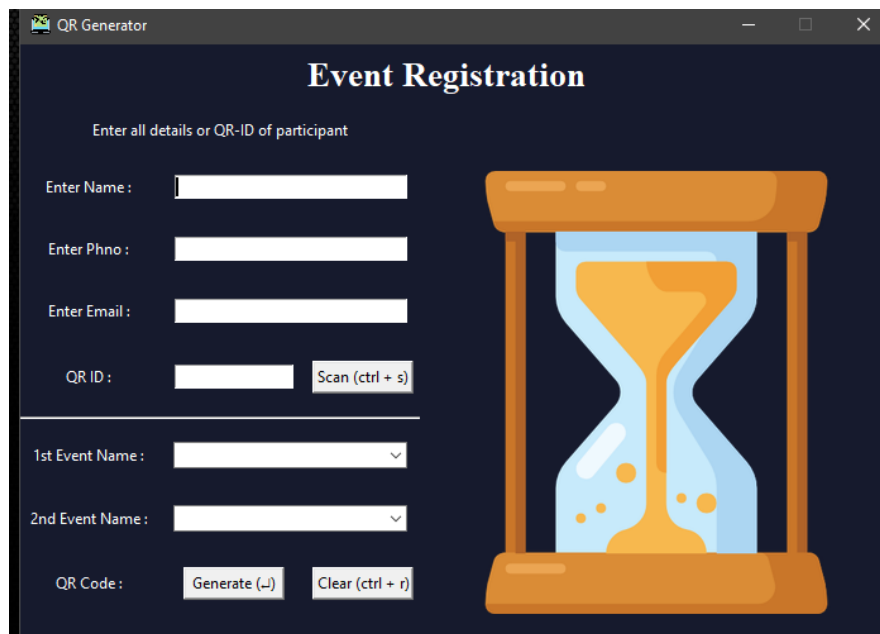
- Clear fields

Figure 40: Cleared Participant fields 1

Figure 41: Cleared Participant fields 2

- Attempt to re-add participant without new data
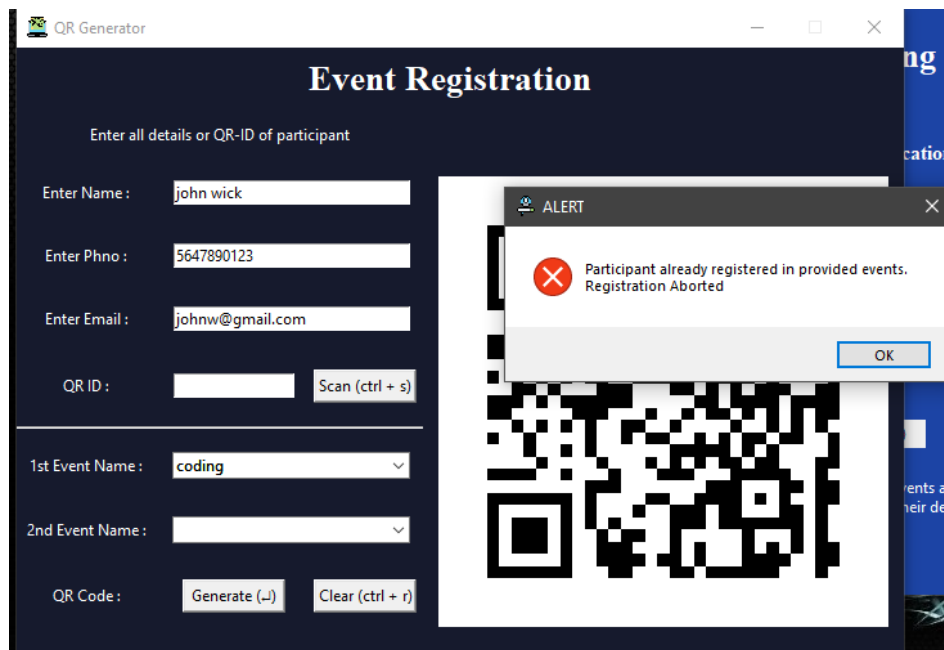
Figure 42: Error-Participant Already Registered in this Event

- Re-adding existing participants to new events
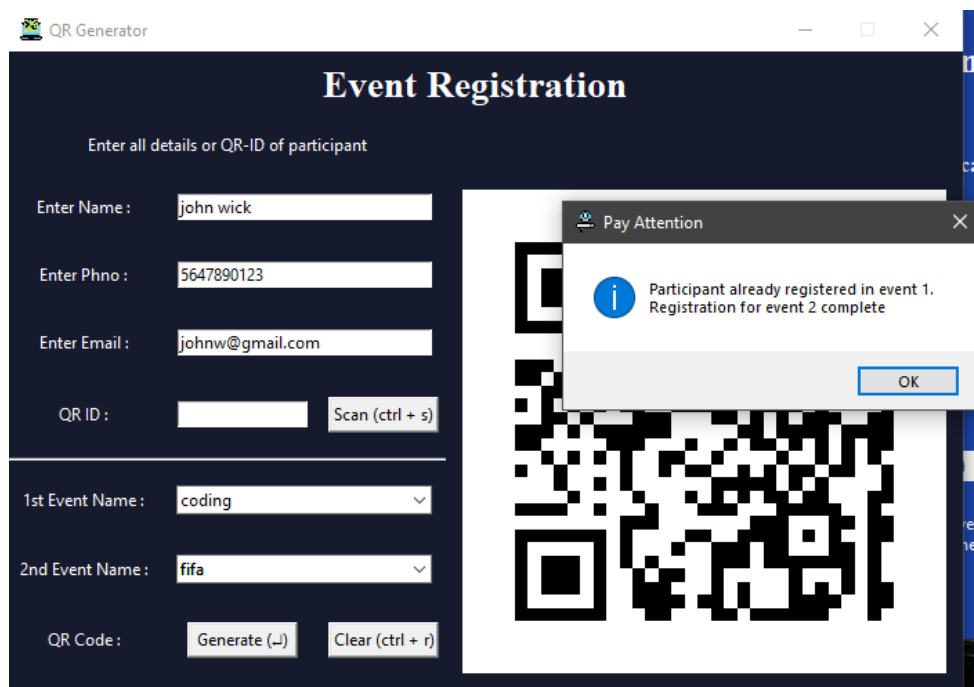
Figure 43: Participant Registered in one of the provided events

- Re-adding existing participants to new events using QR
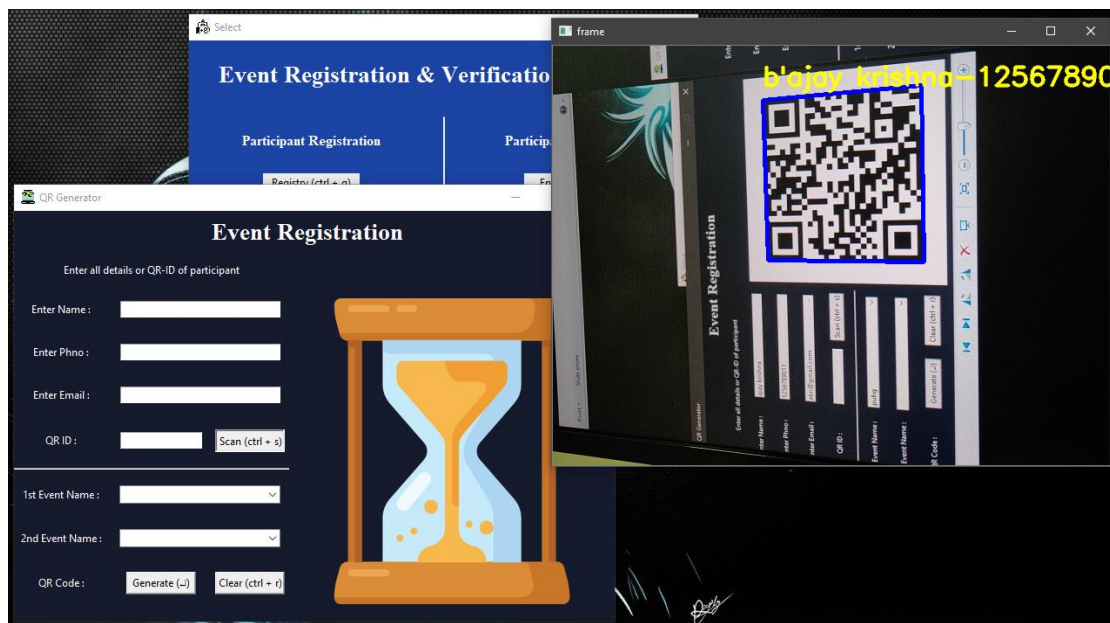
Figure 44: Scanning Existing participant QR code

Figure 45: Display QR code value
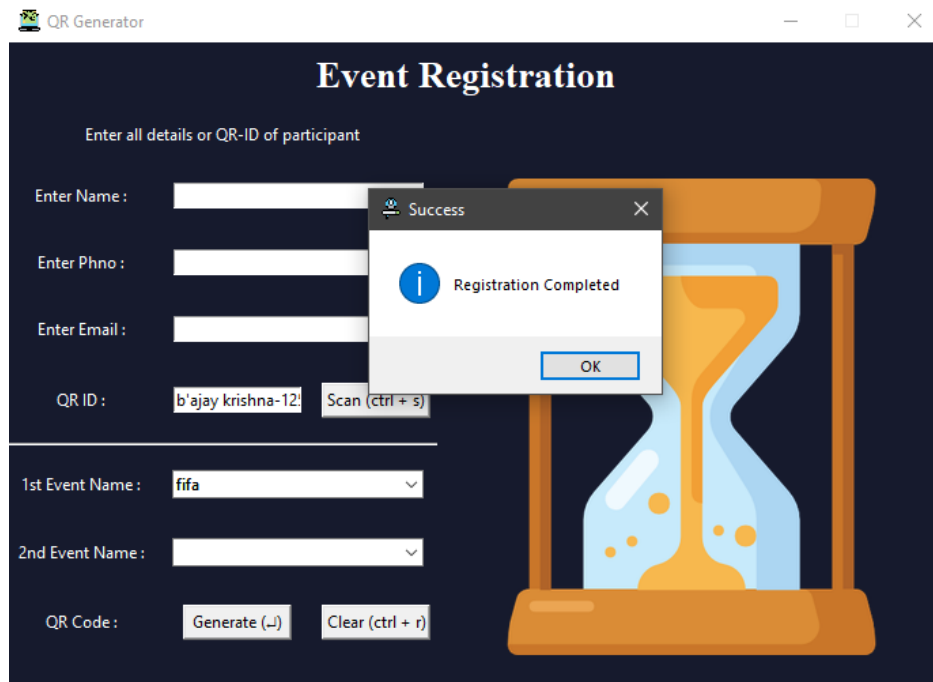
Figure 46: Participant Registration Successful

6.7 Report Generation

- Report generation

Figure 47: Report Generated Successfully

- Report

Figure 48: Report Excel File

6.8 Participant Entry

- Participant verification & entry window
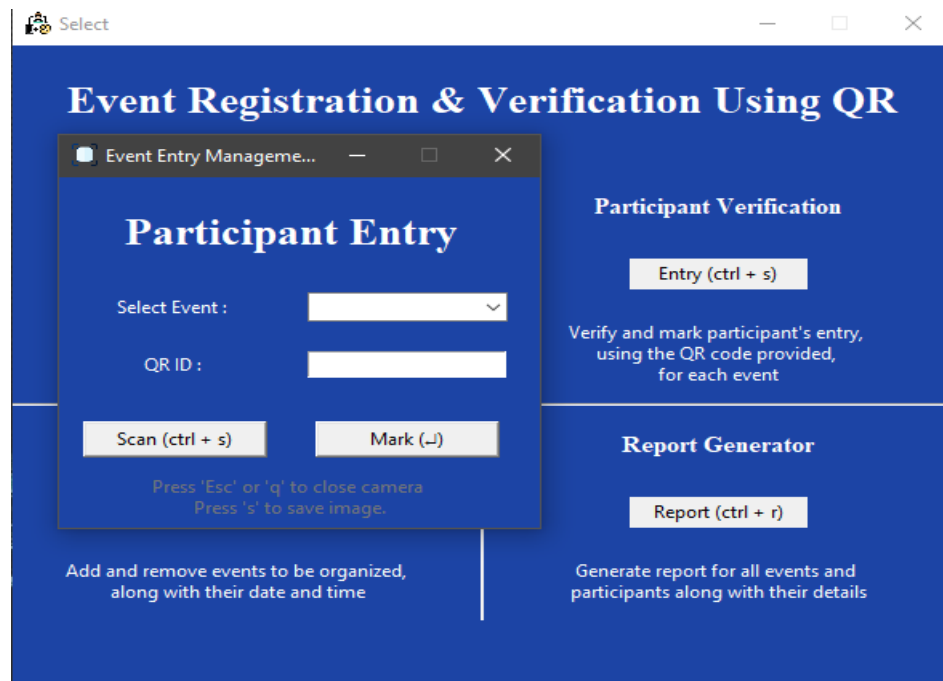
Figure 49: Participant Entry Management

- Attempt to enter with incomplete fields
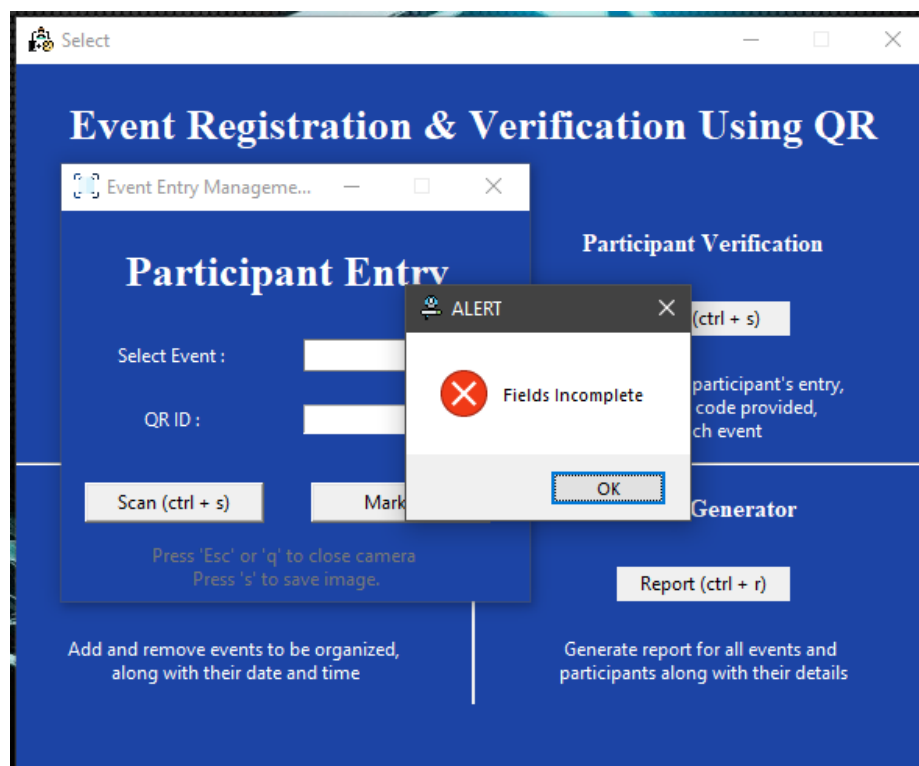
Figure 50: Participant Entry Fields Incomplete

- Scanning & saving a QR
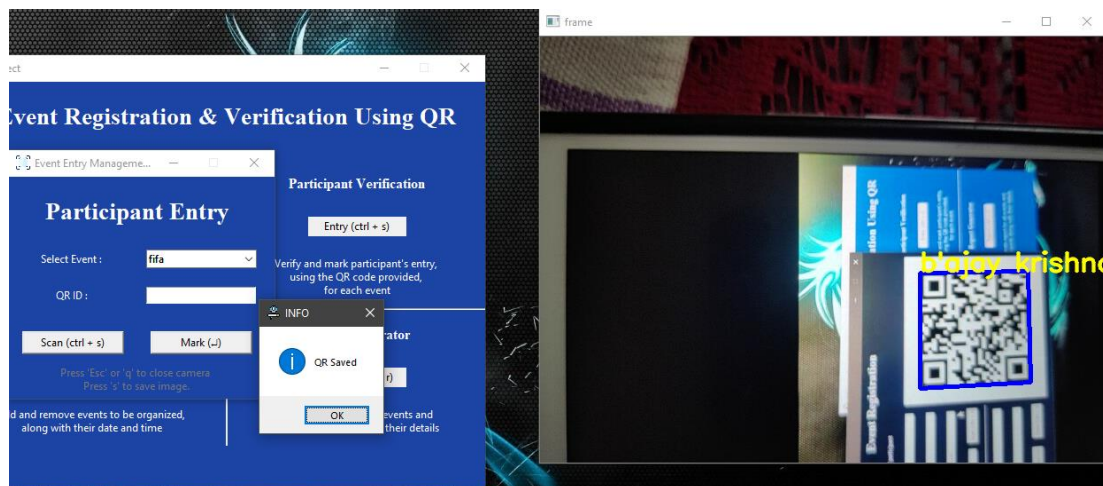
Figure 51: QR Scanned & Saved

- If no QR is found
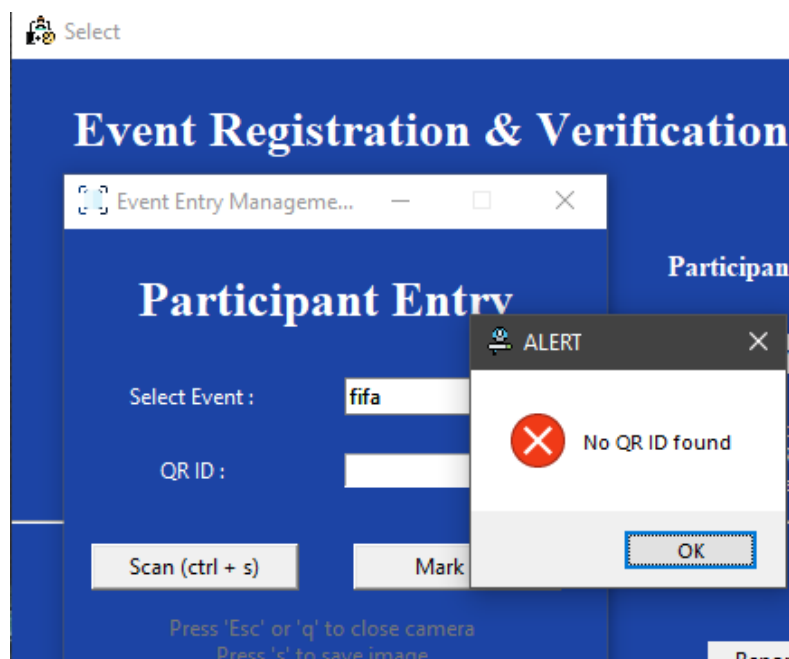
Figure 52: No QR Detected

- If QR is found
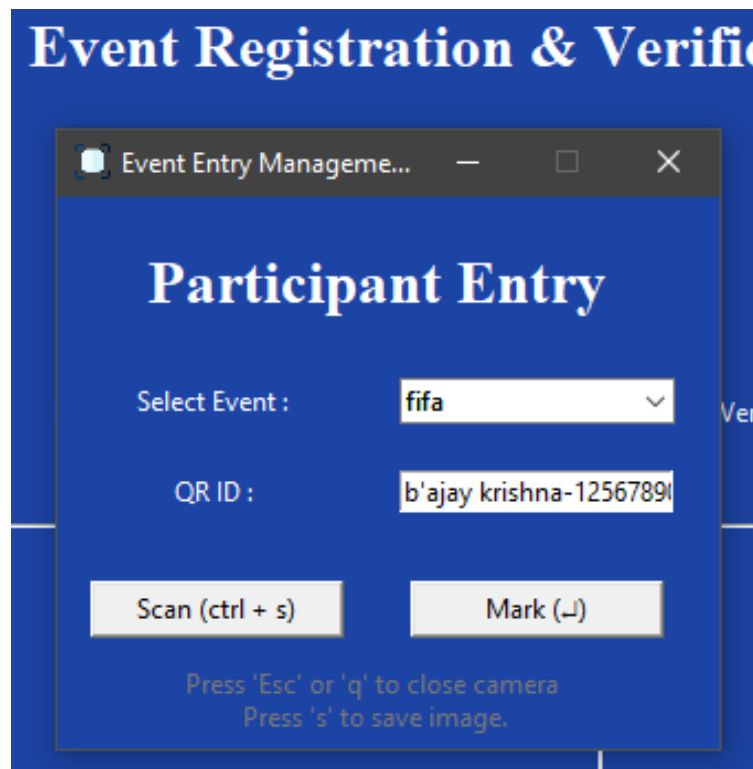
Figure 53: QR code read successfully

- If QR is invalid or not for selected event
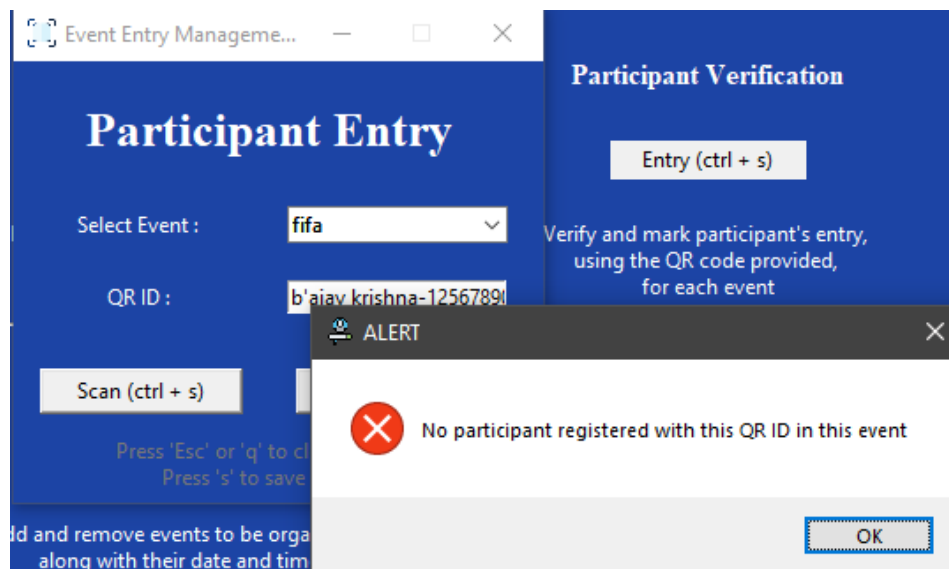
Figure 54: Invalid QR code

- Successful entry
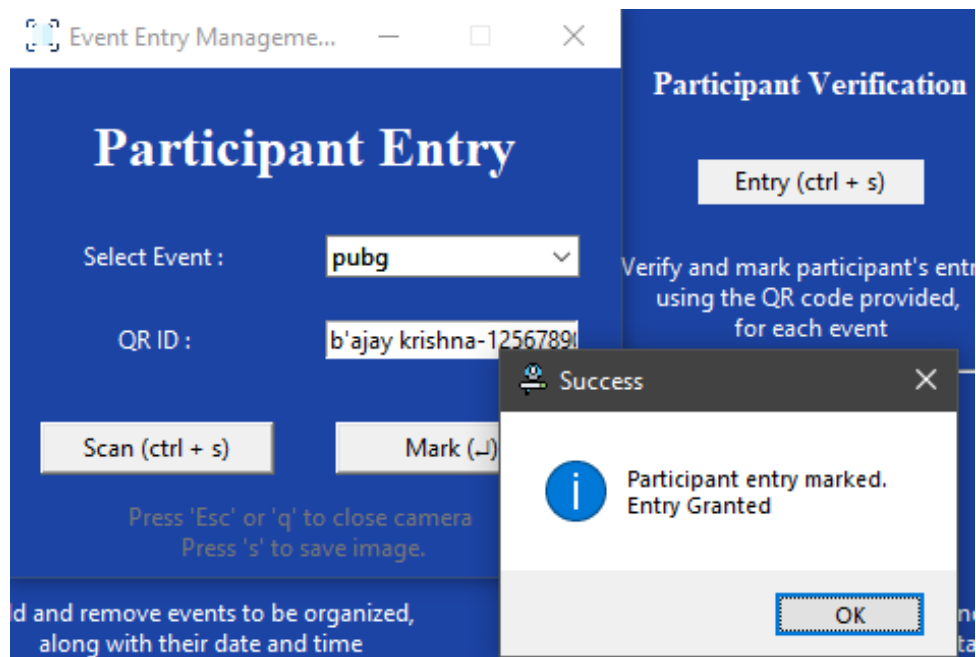
Figure 55: Participant Entry Successful

- Attempt to reuse QR maliciously
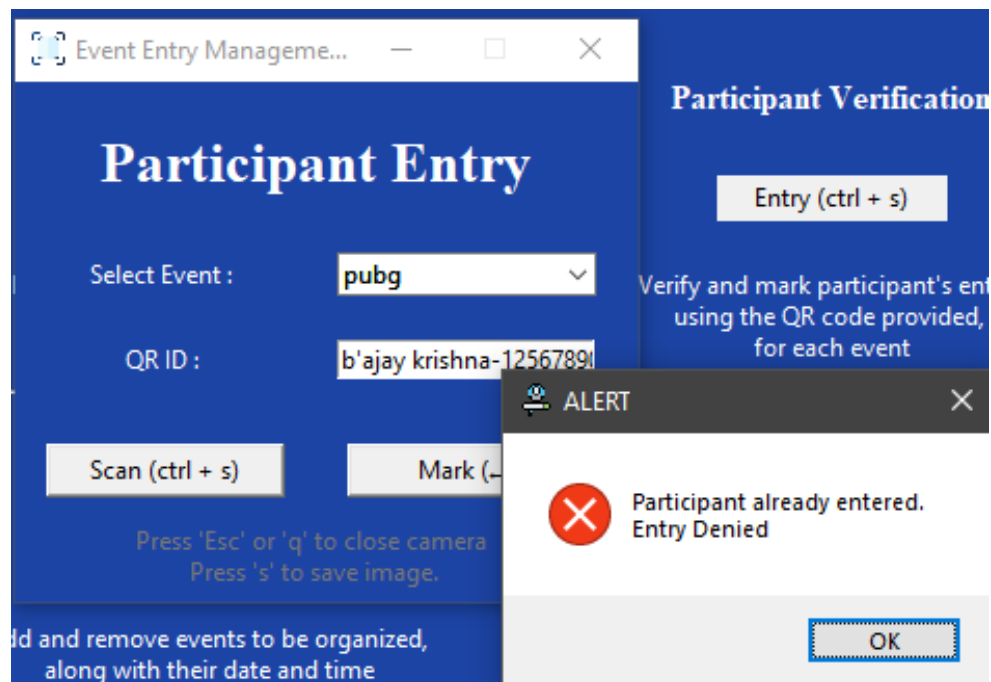
Figure 56: Error-QR code being reused

- After entry has been granted
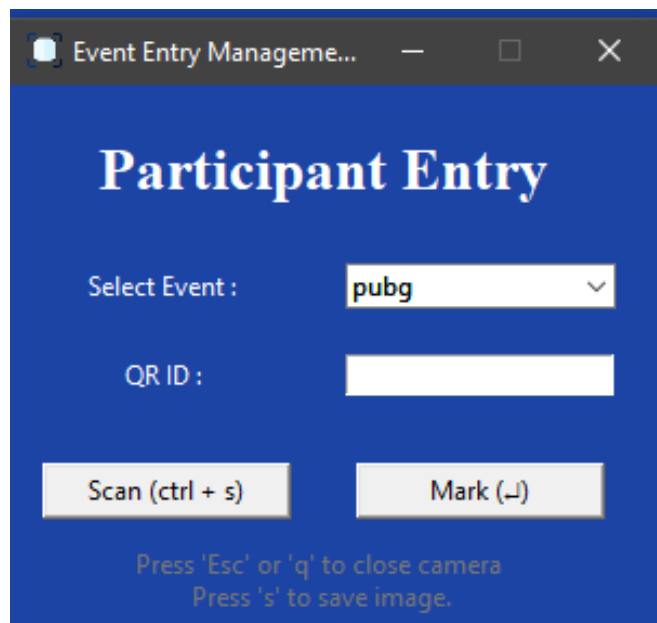
Figure 57: After granting entry

6.9 Server

- Error in communication with server at any point
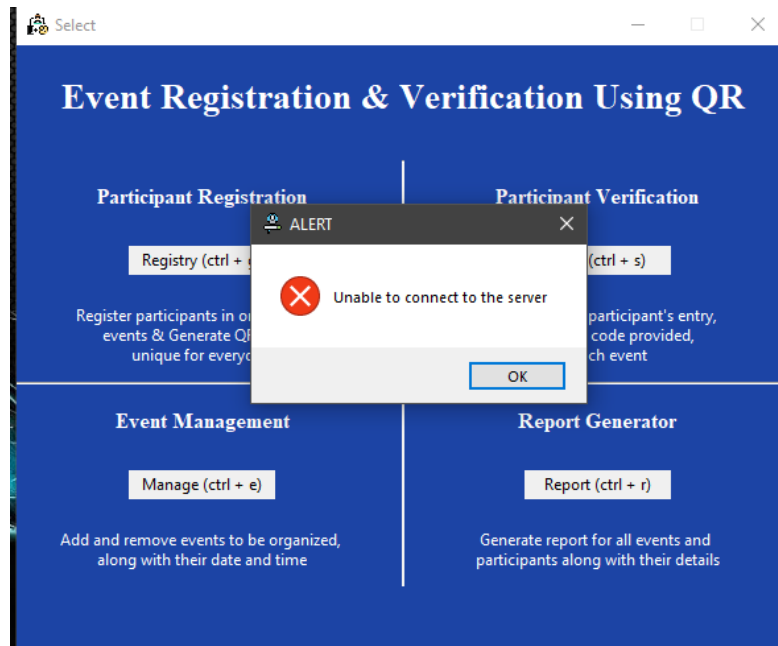
Figure 58: Network Error
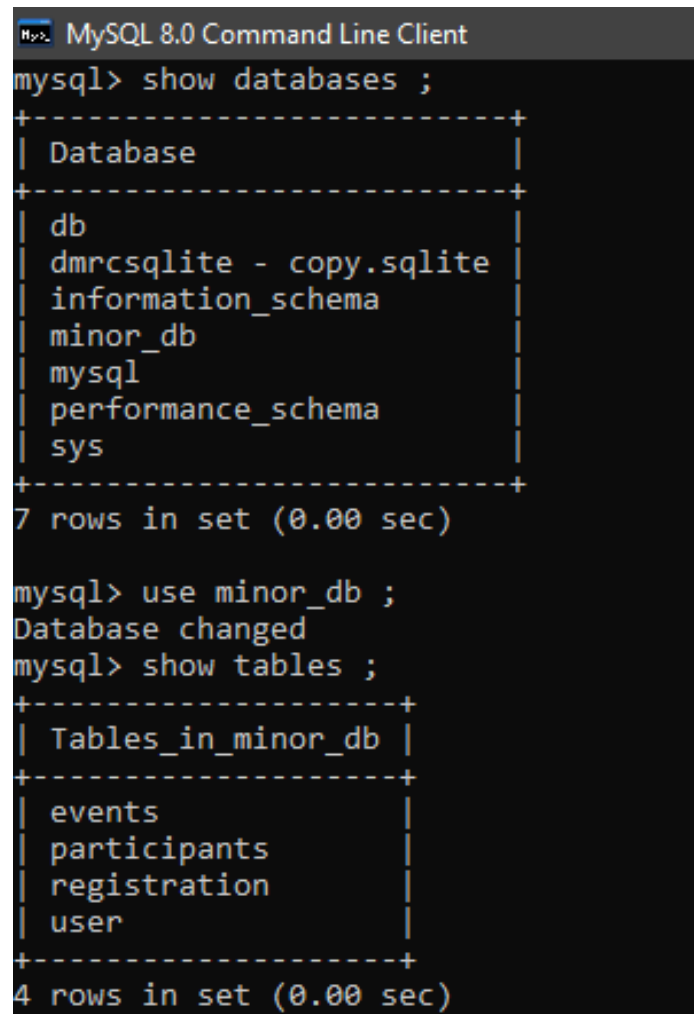
- Server handling during any process

Figure 59: Server Log

6.10 Database

- Database

Figure 60: Database and Tables

- User table

Figure 61: User Table Description

Figure 62: User Table Data

164

- Events table

Figure 63: Event Table Description

Figure 64: Event Table Data

165

- Participants table

Figure 65: Participants Table Description

Figure 66: Participants Table Data

166

- Registration table

Figure 67: Registration Table Description

Figure 68: Registration Table Data

7. System Testing

Testing is the final stage of the application development which plays a vital role in the process of creating high-quality software. Desktop Application testing is a type of software testing that focuses on desktop applications. It involves complete testing of a desktop-based system before going live which helps address issues before the system is revealed to the public.

The aspects taken into consideration while carrying out testing involve functional and non-functional areas (the basic functionality of the application), security of the application, usability and performance monitoring, ability to adapt to the multitude of desktops, devices, and operating systems.

Figure 69: Testing Phases

The following different types of testing have been performed to check the functionality, performance and overall efficiency of our application:

7.1 System Testing

We performed system (compatibility) testing to test our application with the supported software and hardware configurations.

We performed the following:

- OS Configuration
- Server Configuration

7.1.1. **Cross-platform testing:**

It allows evaluating the work of the application in different OS:

- Windows
- iOS/Mac OS
- Linux

7.1.2. **Server testing:**

We verified that our application worked equally well and efficiently when receives request from several sources simultaneously.

7.2 Module Testing

We checked each module in our application separately to test its functionality, connection and inter-dependence on other modules.

Modules	Description	%TCs Executed	%TCs Passed	TCs Pending	Priority
User Registration	Register admin or user of software.	100%	100%	0	MEDIUM
Admin / User Login	Login of user or admin.	100%	100%	0	HIGH
Add Participant	Add participant.	100%	100%	0	HIGH
Register Participant in Events	Register participant in events.	100%	100%	0	HIGH
Add / Remove Events	Add or remove events from database.	100%	100%	0	HIGH
Mark Entry of Participant	Mark entry of a participant in an event.	100%	100%	0	HIGH
Report Generation	Generate report of all participants with details.	100%	100%	0	MEDIUM
QR Code Generation	Generate QR code for participants.	100%	100%	0	HIGH
Scan QR code	Scan QR code for participant verification.	100%	100%	0	MEDIUM

Table 2: Module Testing

170

7.3 User Testing

We have tested our system with some students and people unfamiliar with software.

7.4 Functionality Testing

We performed functional testing to ensure that each function of our application operated in conformance with the requirement specification. Basically, functionality testing is a quality assurance (QA) process and a type of black-box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (unlike white-box testing).

1. Buttons and Textboxes Testing, where we verified
   - Correctness of buttons.
   - Checking buttons lead to the desired windows.
   - If there are windows that are not referenced.
   - There are no broken buttons.
   - Inputs are being read correctly from textboxes.

2. Testing for all windows, where we verified
   - The input data validity.
   - Allowed values for the data field.
   - Invalid input values for the data field.
   - Options for forms in which deletion or any other modification of data is possible.

3. Code Validation, where we verified
   - Code syntax errors.

7.5 Interface Testing

We performed interface testing to verify that the graphical user interface of our application meets the specifications. Basically, Interface testing ensures that all interactions between the web server and application interfaces are running smoothly.

This includes checking the communication processes as well as making sure that error messages are displayed correctly. Further, interruptions by the user as well as the server also needs to be tested for correct handling. Some of the verifications we performed are:

- Compliance with the standards of graphical interfaces.
- Design elements evaluation: layout, colors, fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links.
- Testing with different screen resolutions.
- Testing the graphical user interface on target devices i.e. desktops.

8. Conclusion and Future Scope

Our project is an initiative to satisfy the problems faced in manual event management system. It includes problems in managing the entries, payments, attendee tickets, managing and coordinating the services provided by the event, venue details and time etc. The application would provide an automatic, user friendly and time saving system to organise event and manage participations.

8.1 Applications

- **Can be used in any institution to manage the events effectively.**

This application provides an easy and efficient solution for managing events with least human intervention. This would provide an ease of managing the events and provide a common and uniform platform for the same for the entire university.

- **Can be used in schools as well for management of events that require no attending fees.**

Apart from use in a university, this system also finds utility in schools where multiple academic, extra-curricular and traditional events happen requiring attendance of large number of students at once. This application would help the concerned school authorities to organise events effectively.

8.2 Future Work

- **Chat-Group Module**

A chat-group can be created that would help in society group discussion instead of using different Chat Application like WhatsApp.

- **Implementing Security Techniques**

Further multiple security and testing techniques can be applied to improve the functionality and security of the application.

- **SMS Facility**

SMS facility of upcoming events and reminders can also be provided along with the email service.

- **Website Integration**

As of now, this project is a desktop application. The participants have to reach out to the registration desk to register themselves. In the near future, this application could be redesigned as a web-based application to make online registrations possible. This will reduce the waiting time of participants standing in line to register. This will make the registration process easier for the participant and encourage more people to participate.

References

[1] Scanning QR code-OpenCV with python

https://pysource.com/2019/02/28/scanning-qr-code-opencv-with-python/

[2] Create and read QR code with Python in 3 minutes (OpenCV and qrcode)

https://medium.com/@stasinskipawel/create-and-read-qr-code-with-python-in-3-minutes-opencv-and-qrcode-38ecc3a6258a

[3] Openpyxl

https://pypi.org/project/openpyxl/

[4] Python Regex

https://www.w3schools.com/python/python_regex.asp

[5] Iconphoto () method in Tkinter | Python

https://www.geeksforgeeks.org/iconphoto-method-in-tkinter-python/

[6] How do I share global variables across modules?

http://effbot.org/pyfaq/how-do-i-share-global-variables-across-modules.htm

[7] Tkinter: Catching Window Deletion

https://effbot.org/zone/tkinter-window-destroy.htm

[8] Responses

https://www.flaskapi.org/#responses

[9] Requests

https://www.flaskapi.org/#requests

[10]   MySQL GROUP_CONCAT () function

https://www.w3resource.com/mysql/aggregate-functions-and-grouping/aggregate-functions-and-grouping-group_concat.php

[11]   Python - GUI Programming (Tkinter)

https://www.tutorialspoint.com/python/python_gui_programming.htm

[12]   MySQL Tutorial

https://www.javatpoint.com/mysql-tutorial

[13]   OpenCV Tutorials

https://docs.opencv.org/master/d9/df8/tutorial_root.html

[14]   Python OpenCV | cv2.putText() method

https://www.geeksforgeeks.org/python-opencv-cv2-puttext-method/

[15]   Resizing window doesn't resize contents in tkinter

https://stackoverflow.com/questions/17598475/resizing-window-doesnt-resize-contents-in-tkinter