EXAM HALL SEATING ARRANGEMENT SYSTEM

**SYNOPSIS**

The Automatic Exam Seating Arrangement System for Students is a software-based solution designed to streamline and optimize the process of seating arrangement for exams in college. This system leverages advanced algorithms and data analysis techniques to automate the tedious task of assigning seats to students based on various criteria. Traditionally, exam seating arrangements are manually handled, consuming significant time and effort from administrators. In our system aims to alleviate this burden by providing an efficient and accurate seating allocation process. By utilizing the power of technology, it ensures fairness, security, and convenience for both students and exam administrators. The system incorporates several key features. First, it allows administrators to define and input relevant parameters, such as the number of students, seating a capacity, and any special requirements. It then processes this information to generate optimal seating plans that satisfy constraints such as seating capacity, student preferences, and separation between certain individuals. It intelligently assigns seats to ensure fairness and prevent cheating while respecting any specific by the institution. The system also offers flexibility and adaptability. It can handle last-minute changes, such and student absences or new enrollments, and quickly generate revised seating arrangements.

**INTRODUCTION**

In today's rapidly changing world driven by advanced data management and automation, educational institutions face the challenge of efficiently organizing and allocating resources. Among these resources, seating arrangements for students in classrooms pose a significant logistical challenge. This project addresses these challenges by providing a simple yet effective solution designed to streamline the process for teachers and Vysya college administrators.

By using modern technology and innovative approaches, this project aims to transform the way seating arrangements are managed in learning environments. With a focus on user-friendly interfaces and intuitive workflows, the system empowers educators and administrators to create optimal seating arrangements with ease and efficiency.

In the past, assigning seats manually and dealing with paperwork was the norm. However, this project introduces a new modern age for educational institutions, using automation and digital tools to streamline the process of allocating seats. This advancement saves time and effort for everyone involved. While Excel and VBA macros have progressed to address certain needs, this project goes beyond by providing a user-friendly graphical interface. This interface improves accessibility and usability for administrators, teachers, and students, making the whole process smoother and more efficient.

**Objective:**

The primary objective is to optimize and streamline the allocation of seats to students, with the overarching goal of enhancing the overall efficiency and effectiveness of the seating arrangement process. By providing a user-friendly interface and implementing effective seating arrangements, the project aims to simplify seat allocation, making it more accessible and manageable for administrators, teachers, and students.

**Benefits:**

* Saves time and effort for educators.
* Ensures a fair and balanced distribution of students in classrooms.
* Provides an easy-to-use tool for effective seat management.

**SYSTEM SPECIFICATION**

**Hardware Configuration**

Minimum hardware requirements for this system are listed below:

* + System : Intel Core i5 or AMD Ryzen 5
  + HardDisk : 250 GB HDD
  + Monitor : 1280 x 720 pixels
  + RAM : 4GB
  + Keyboard : Standard Keyboard

**Software Specification**

Software requirements for this system are as listed

* + Operating System : Mostly Preferable (Any Windows)
  + Platform : Python Technology
  + Tool : Python 3.12/Visual Studio Code
  + Frontend : PyQt5,Tkinter,DataFrames
  + Backend : Python, Excel Sheet

**SYSTEM STUDY**

**EXISTING SYSTEM**

* The existing system in our college relies on Excel coupled with VBA macros for seating arrangements. This system typically involves manually inputting student data, room specifications, and seating preferences into Excel spreadsheets, with VBA macros automating certain aspects of the allocation process.
* The system allows for highly customizable seating arrangements tailored to specific exam formats, room layouts, and scheduling requirements
* VBA macros automate repetitive tasks, such as assigning students to seats based on predefined criteria and generating seating charts or reports. This automation enhances efficiency and reduces the time and effort required for manual seat allocation.
* The system's functionality may be dependent on individual expertise in Excel and VBA programming. Staff turnover or changes in personnel could disrupt the continuity and maintenance of the system, leading to potential inefficiencies or errors.

**Description:**

* While the existing system requires users to manually input student data and room specifications into Excel sheets, my project provides a graphical user interface (GUI) that simplifies this process. Instead of navigating complex spreadsheets and writing VBA macros, users can input data effortlessly using intuitive forms and menus.
* Furthermore, the existing system's reliance on VBA macros for automation poses challenges for users who lack programming skills or familiarity with Excel. In contrast, my project eliminates the need for VBA programming by offering built-in algorithms and automation features accessible through the GUI. This approach enhances accessibility and usability for all users, regardless of their technical expertise.
* Overall, my current project represents a significant improvement over the existing system by offering a modern, user-friendly, and automated solution for seating arrangement management in our college.

**Drawbacks:**

While the existing system with Excel and Macros/VBA may have served its purpose to some extent, it is not without its drawbacks:

1. **Limited Scalability:**

The existing system's reliance on Excel and VBA macros may limit its scalability, especially when dealing with large datasets or complex seating arrangements. As the volume of data increases, the performance of Excel may degrade, resulting in slower processing times and reduced efficiency.

1. **Dependency on Technical Expertise:**

Operating and maintaining the existing system requires a certain level of technical expertise, particularly in VBA programming.

1. **Complexity:**

Managing large datasets and seating configurations in Excel can be complex and cumbersome, particularly for administrators without advanced Excel skills.

**PROPOSED SYSTEM**

The proposed system is a modern, user-friendly software application designed to revolutionize the process of seating arrangement management in educational institutions. Unlike the existing system, which relies on manual data entry and limited automation through Excel and VBA macros, the proposed system offers a comprehensive solution with advanced features and enhanced usability.

**Description**

* Imagine a system where you can easily input exam details, such as the number of students and available rooms, using a familiar and intuitive interface. With just a few clicks, you can define seating preferences and let the system handle the rest.
* This system is all about automation. It takes care of assigning seats to students based on various factors like room capacity and special requirements, so you don't have to worry about manual calculations or errors keeping in.
* One of the best parts is its flexibility. Whether you're organizing a small classroom test or a large-scale exam, the system adapts seamlessly to your needs, making it suitable for any educational institution.
* Finally, this system offers a simple and user-friendly solution to the challenges faced in managing seating arrangements for exams in educational institutions. It's like having a smart assistant that handles all the complexities behind the scenes, leaving educators and administrators with a hassle-free experience.

**Features**

1. **Excel Data Processing:**

The system allows users to import student data from Excel files, ensuring compatibility with existing data sources.

1. **Efficient Data Allocation:**

Utilizing Python and pandas, the system processes and allocates student seats in a more efficient and optimized manner.

1. **Dynamic Scalability:**

The proposed system is designed to handle varying student numbers and classroom configurations, providing scalability for growing educational institutions.

1. **User-Friendly Interface:**

A Tkinter-based graphical user interface (GUI) makes the system accessible to users with varying technical expertise, reducing dependency on specialized skills.

1. **Error Reduction:**

Automation minimizes the risk of human errors, enhancing the accuracy of student seat allocation.

1. **Quick Generation of Seating Arrangements:**

The system generates seating arrangements swiftly, saving time for educators and administrators.

1. **Flexibility for Modifications:**

Changes in student enrollment or classroom setups can be easily accommodated through the system's adaptable design.

1. **Output in Excel Format:**

The final seating arrangement is exported to Excel, providing a familiar and widely used format for easy integration into existing workflows.

**MODULES**

* LOGIN PAGE
* SINUP PAGE
* HOME PAGE
* ROOM SPECIFICATION
* GENERATE REPORT

**SYSTEM DESIGN AND DEVELOPMENT**

**File Design**

The file design of the proposed system outlines the organization and structure of data files used during the student seat allocation process. This section provides an overview of the key files and their purposes.

**Input File**

Excel File (Student Data)

**Purpose:** The primary input file containing student data.

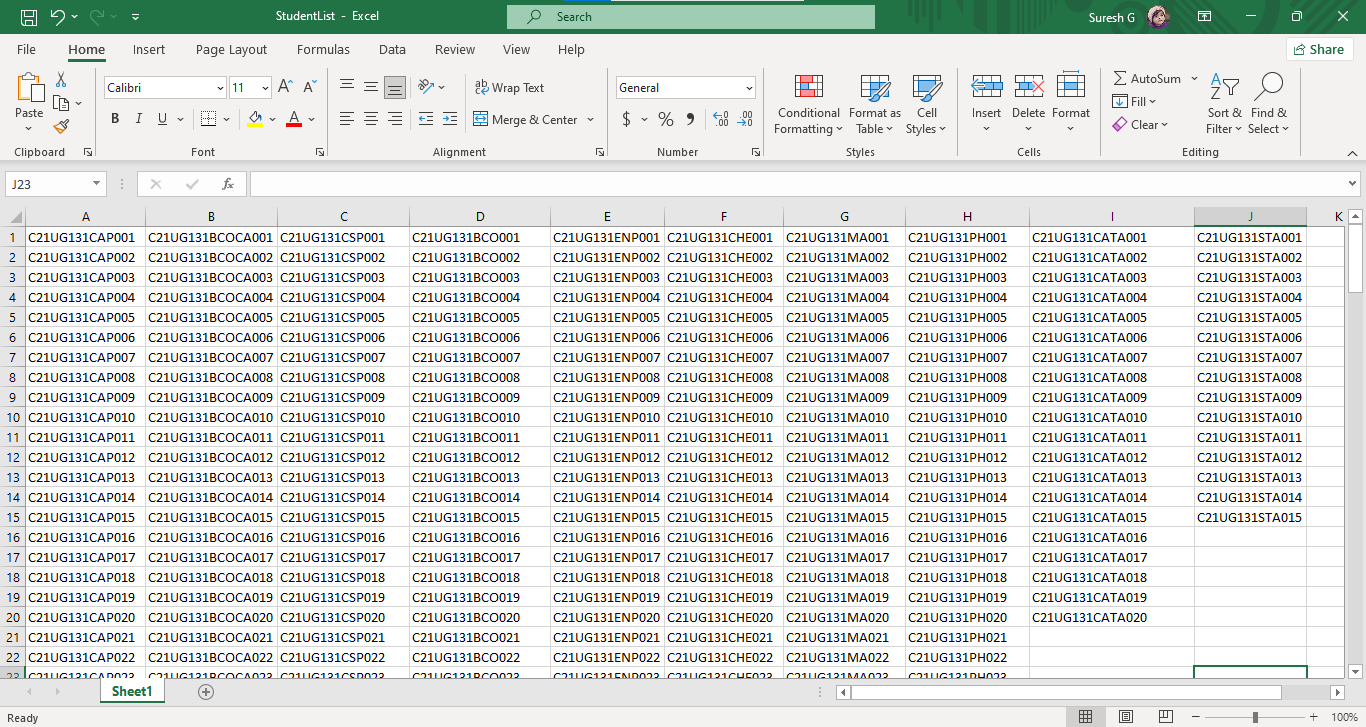
**Format:** Microsoft Excel (.xlsx)

**Structure:**

Each column represents the department of students.

Columns may include Students Register Number with decreasing ordered format.

**Design:**

****

**Output File:**

Excel File (Seating Arrangement)

**Purpose:** The output file storing the final seating arrangement.

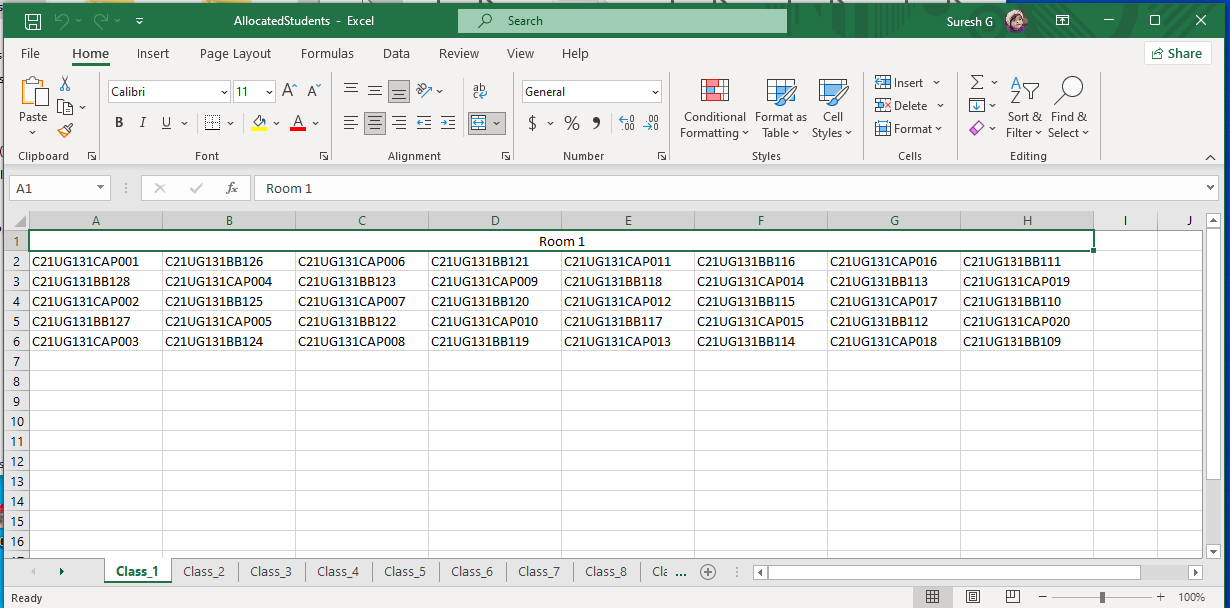
**Format:** Microsoft Excel (.xlsx)

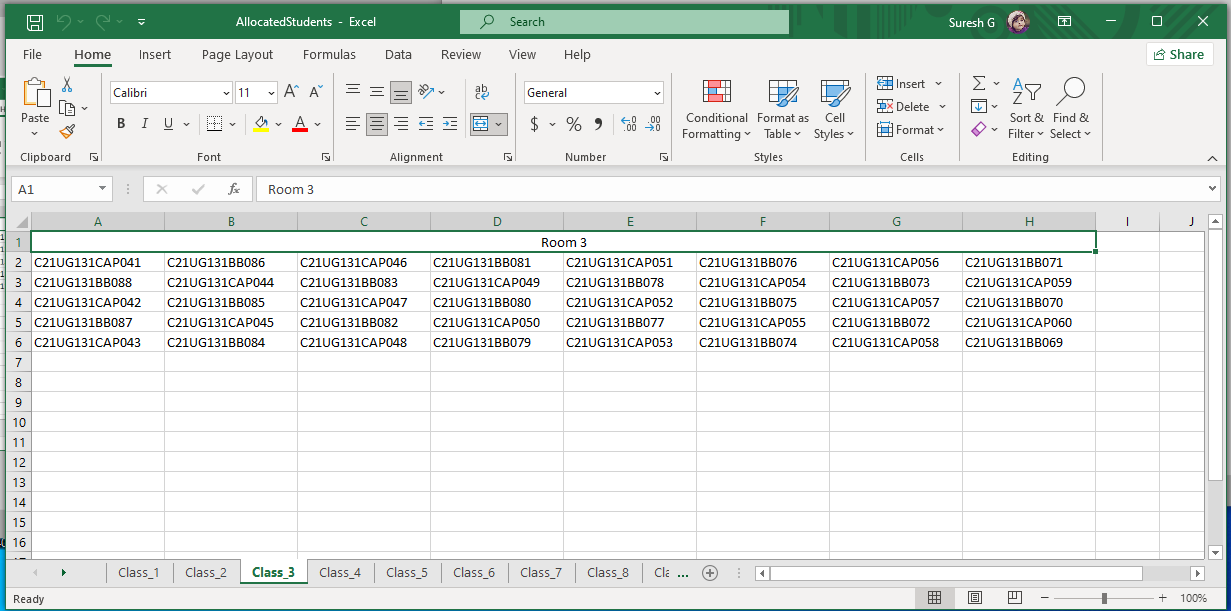
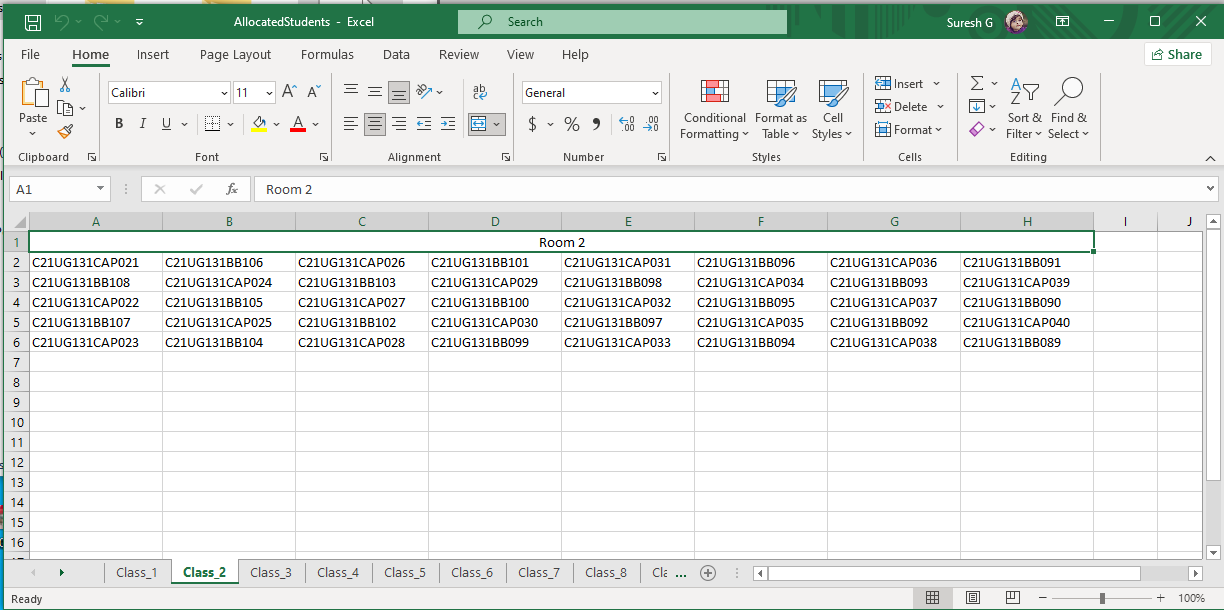
**Structure:**

Each sheet corresponds to a classroom.

Columns represent seat positions, and rows represent students.

**Constraints:** Follows the specified format for easy integration into existingworkflows.

**Design:**

****

**Internal Data Structures:**

Lists and DataFrames (Intermediate Data)

**Purpose:** Temporary data structures for efficient processing.

**Format:** Python lists and pandas DataFrames.

**Structure:** Dynamic storage of student data during allocation.

**SAMPLE CODING:**

**Main.py**

# main\_program.py

import sys

from PyQt5.QtWidgets import QApplication, QWidget

from design\_module import DesignModule

from allocation\_module import AllocationModule

class MainProgram(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.design\_module = DesignModule(self)

self.allocation\_module = AllocationModule(self)

self.init\_ui()

def init\_ui(self):

# Additional initialization if needed

pass

def generate\_excel\_output(self):

# Retrieve necessary information from the design module

file\_path = self.design\_module.file\_entry.text()

students\_per\_class = int(self.design\_module.class\_entry.text())

rows = int(self.design\_module.rows\_entry.text())

columns = int(self.design\_module.columns\_entry.text())

# Use the allocation module to perform the logic

l3, \_ = self.allocation\_module.read\_excel\_and\_generate\_output(file\_path, students\_per\_class)

self.allocation\_module.generate\_excel\_output(l3, students\_per\_class, rows, columns)

def main():

app = QApplication(sys.argv)

window = MainProgram()

window.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Login.py**

import tkinter as tk

from tkinter import Toplevel, Label, Button, Entry, Checkbutton, BooleanVar

from PIL import Image, ImageTk

import ast

def show\_custom\_error(title, message):

error\_dialog = Toplevel()

error\_dialog.title(title)

error\_dialog.iconbitmap(r"C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\logoico.ico")

error\_dialog.geometry('400x150+500+300')

error\_dialog.configure(bg='#c0392b') # Set background color to light red

error\_dialog.resizable(False, False)

Label(error\_dialog, text=message, fg='#f5f6fa', bg='#c0392b', font=('Arial', 14, 'bold')).pack(pady=5)

def close\_dialog():

error\_dialog.destroy()

# Resume background tasks here

Button(error\_dialog, text='OK', command=close\_dialog, bg='#0652DD', fg='white', font=('Arial', 14, 'bold')).pack(pady=30)

# Make the error dialog modal

error\_dialog.grab\_set()

# Wait until the error dialog is closed before continuing with the background tasks

error\_dialog.wait\_window()

def toggle\_password\_visibility():

global show\_password

show\_password = not show\_password

if show\_password:

code.config(show="")

else:

code.config(show="\*")

def signin():

username = user.get()

password = code.get()

if username == 'admin' and password == '1234':

screen = Toplevel()

screen.title("App")

screen.geometry('925x500+300+200')

screen.config(bg='white')

Label(screen, text='Hello Everyone', bg="#fff", font=('Calibri(Body)', 50, 'bold')).pack(expand=True)

screen.mainloop()

elif username != 'admin' and password != '1234':

show\_custom\_error("Incorrect", "Incorrect Username and Password")

elif password != '1234':

show\_custom\_error("Incorrect", "Incorrect Password")

elif username != 'admin':

show\_custom\_error("Incorrect", "Incorrect Password")

root = tk.Tk()

root.title('EXAM HALL SEATING ARRANGEMENT')

root.iconbitmap(r'C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\logoico.ico')

root.geometry('925x500+300+200')

root.configure(bg='#e84118') # Set background color to #e84118

root.resizable(False, False)

# Load and display the background image

try:

pil\_image = Image.open(r'C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\bg1.jpg')

img = ImageTk.PhotoImage(pil\_image)

label = Label(root, image=img)

label.place(x=0, y=0, relwidth=1, relheight=1)

except Exception as e:

print(f"An error occurred: {e}")

# Create a custom title bar

title\_bar = tk.Frame(root, bg='#1abc9c', height=30)

title\_bar.pack(fill='x')

# Create a label for the title

title\_label = tk.Label(title\_bar, text='Login Page', bg='#1abc9c', fg='white', font=('Helvetica', 12,'bold'))

title\_label.pack(side='left', padx=10)

# Create the frame

frame = tk.Frame(root, width=350, height=350, bg='white')

frame.place(x=480, y=70)

# Add heading label to the frame

heading = tk.Label(frame, text='Sign In', fg='#1abc9c', bg='white', font=('Microsoft Vahei UI L', 23, 'bold'))

heading.place(x=120, y=5)

###==================================================

# Create an Username Entry widget

def on\_enter\_user(e):

user.delete(0,'end')

def on\_leave\_user(e):

name=user.get()

if name=='':

user.insert(0,'Username')

user = tk.Entry(frame, width=25, fg='black', border=0, bg='white', font=('Microsoft Vahei UI Light', 11))

user.place(x=30, y=80) # Adjust the position as needed

user.insert(0,'Username')

user.bind('<FocusIn>',on\_enter\_user)

user.bind('<FocusOut>',on\_leave\_user)

tk.Frame(frame,width=295,height=2,bg='black').place(x=25,y=107)

###==================================================

# Create a Password Entry widget

def on\_enter\_code(e):

code.delete(0,'end')

def on\_leave\_code(e):

name=code.get()

if name=='':

code.insert(0,'Password')

show\_password = False

code = tk.Entry(frame, width=20, fg='black', border=0, bg='white', font=('Microsoft Vahei UI Light', 11), show="\*")

code.place(x=30, y=150) # Adjust the position as needed

code.insert(0,'Password')

code.bind('<FocusIn>',on\_enter\_code)

code.bind('<FocusOut>',on\_leave\_code)

tk.Frame(frame,width=295,height=2,bg='black').place(x=25,y=177)

##=============================================================

##=============================================================

tk.Button(frame, width=39, pady=7, text='Sign in', bg='#1abc9c', fg='white', border=0, command=signin).place(x=35, y=240)

label = tk.Label(frame, text="Don't have an account?", fg='black', bg='white', font=('Microsoft Vahei UI Light', 9))

label.place(x=75, y=290)

sign\_up = tk.Button(frame, width=6, text='Sign up', border=0, bg='white', cursor='hand2', fg='#1abc9c')

sign\_up.place(x=210, y=290)

# Checkbox for hiding and showing password

show\_password\_var = BooleanVar()

show\_password\_check = Checkbutton(frame, text="Show Password", variable=show\_password\_var, command=toggle\_password\_visibility, bg='white', fg='#1abc9c')

show\_password\_check.place(x=30, y=200)

root.mainloop()

**SignUp.py**

from tkinter import \*

from tkinter import messagebox

import ast

import tkinter as tk

from tkinter import Toplevel, Label, Button, Entry

from PIL import Image, ImageTk

window=Tk()

window.title("EXAM HALL SEATING ARRANGEMENT")

window.geometry('925x500+300+200')

window.configure(bg='#fff')

window.resizable(False,False)

window.iconbitmap(r'C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\logoico.ico')

def show\_custom\_error(title, message):

error\_dialog = Toplevel()

error\_dialog.title(title)

error\_dialog.iconbitmap(r"C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\logoico.ico")

error\_dialog.geometry('400x150+500+300')

error\_dialog.configure(bg='#c0392b') # Set background color to light red

error\_dialog.resizable(False, False)

Label(error\_dialog, text=message, fg='#f5f6fa', bg='#c0392b', font=('Arial', 14, 'bold')).pack(pady=5)

def close\_dialog():

error\_dialog.destroy()

# Resume background tasks here

Button(error\_dialog, text='OK', command=close\_dialog, bg='#0652DD', fg='white', font=('Arial', 14, 'bold')).pack(pady=30)

# Make the error dialog modal

error\_dialog.grab\_set()

# Wait until the error dialog is closed before continuing with the background tasks

error\_dialog.wait\_window()

def show\_custom\_info(title, message):

info\_dialog = Toplevel()

info\_dialog.title(title)

info\_dialog.iconbitmap(r"C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\logoico.ico")

info\_dialog.geometry('400x150+500+300')

info\_dialog.configure(bg='#1abc9c') # Set background color to light green

info\_dialog.resizable(False, False)

Label(info\_dialog, text=message, fg='#f5f6fa', bg='#1abc9c', font=('Arial', 14, 'bold')).pack(pady=5)

def close\_dialog():

info\_dialog.destroy()

# Resume background tasks here

Button(info\_dialog, text='OK', command=close\_dialog, bg='#0652DD', fg='white', font=('Arial', 14, 'bold')).pack(pady=30)

# Make the info dialog modal

info\_dialog.grab\_set()

# Wait until the info dialog is closed before continuing with the background tasks

info\_dialog.wait\_window()

def signup():

username = user.get()

password = code.get()

confirm\_password = confirm\_code.get()

if password == confirm\_password:

try:

file = open('datasheet.txt', 'r+')

d = file.read()

r = ast.literal\_eval(d)

dict2 = {username: password}

r.update(dict2)

file.truncate(0)

file.close()

file = open('datasheet.txt', 'w')

w = file.write(str(r))

show\_custom\_info('SignUp', 'Successfully Signed Up')

except:

file = open('datasheet.txt', 'w')

pp = str({'Username': 'Password'})

file.write(pp)

file.close()

else:

show\_custom\_error('Incorrect', 'Password should not match')

def sign():

window.destroy()

# Load and display the background image

try:

pil\_image = Image.open(r'C:\Users\sures\OneDrive\Desktop\Exam Seat\Assests\img\bg1.jpg')

img = ImageTk.PhotoImage(pil\_image)

label = Label(window, image=img)

label.place(x=0, y=0, relwidth=1, relheight=1)

except Exception as e:

print(f"An error occurred: {e}")

# Create a custom title bar

title\_bar = tk.Frame(window, bg='#1abc9c', height=30)

title\_bar.pack(fill='x')

# Create a label for the title

title\_label = tk.Label(title\_bar, text='SignUp Page', bg='#1abc9c', fg='white', font=('Helvetica', 12,'bold'))

title\_label.pack(side='left', padx=10)

#Create Frame

frame=Frame(window,width=350,height=390,bg='white')

frame.place(x=480,y=50)

heading = tk.Label(frame, text='Sign Up', fg='#1abc9c', bg='white', font=('Microsoft Vahei UI L', 23, 'bold'))

heading.place(x=120, y=5)

###-------------------------------------------------------------------------------------------------------------

def on\_enter(e):

user.delete(0,'end')

def on\_leave(e):

if user.get()=='':

user.insert(0,'Username')

user=Entry(frame,width=25,fg='black',border=0,bg='white',font=('Microsoft Vahei UI L', 11))

user.place(x=30,y=80)

user.insert(0,'Username')

user.bind('<FocusIn>',on\_enter)

user.bind('<FocusOut>',on\_leave)

Frame(frame,width=295,height=2,bg='black').place(x=25,y=107)

###-------------------------------------------------------------------------------------------------------------

def on\_enter(e):

code.delete(0,'end')

def on\_leave(e):

if code.get()=='':

code.insert(0,'Password')

code=Entry(frame,width=25,fg='black',border=0,bg='white',font=('Microsoft Vahei UI Light', 11))

code.place(x=30,y=150)

code.insert(0,'Password')

code.bind('<FocusIn>',on\_enter)

code.bind('<FocusOut>',on\_leave)

Frame(frame,width=295,height=2,bg='black').place(x=25,y=177)

###-------------------------------------------------------------------------------------------------------------

def on\_enter(e):

confirm\_code.delete(0,'end')

def on\_leave(e):

if confirm\_code.get()=='':

confirm\_code.insert(0,'Confirm Password')

confirm\_code=Entry(frame,width=25,fg='black',border=0,bg='white',font=('Microsoft Vahei UI Light', 11))

confirm\_code.place(x=30,y=220)

confirm\_code.insert(0,'Confirm Password')

confirm\_code.bind('<FocusIn>',on\_enter)

confirm\_code.bind('<FocusOut>',on\_leave)

Frame(frame,width=295,height=2,bg='black').place(x=25,y=247)

#--------------------------------------------------------------------------------

Button(frame,width=39,pady=7,text='Sign Up',bg='#1abc9c',fg='white',border=0,command=signup).place(x=35,y=280)

label=Label(frame,text='I have an account',fg='black',bg='white',font=('Microsoft Vahei UI Light', 9))

label.place(x=90,y=340)

signin=Button(frame,width=6,text='Sign in',border=0,bg='white',cursor='hand2',fg='#1abc9c',command=sign)

signin.place(x=190,y=340)

window.mainloop()

**Design\_module.py**

# design\_module.py

from PyQt5.QtWidgets import QLabel, QLineEdit, QPushButton, QVBoxLayout, QFileDialog

class DesignModule:

def \_\_init\_\_(self, parent):

self.parent = parent

self.init\_ui()

def init\_ui(self):

self.parent.setWindowTitle('Classroom Generator')

self.parent.setGeometry(100, 100, 400, 200)

self.file\_label = QLabel('Select Excel file:')

self.file\_entry = QLineEdit()

self.browse\_button = QPushButton('Browse', self.parent)

self.browse\_button.clicked.connect(self.browse\_file)

self.class\_label = QLabel('Students per Class:')

self.class\_entry = QLineEdit()

self.rows\_label = QLabel('Rows:')

self.rows\_entry = QLineEdit()

self.columns\_label = QLabel('Columns:')

self.columns\_entry = QLineEdit()

self.generate\_button = QPushButton('Generate Output', self.parent)

self.generate\_button.clicked.connect(self.parent.generate\_excel\_output)

layout = QVBoxLayout()

layout.addWidget(self.file\_label)

layout.addWidget(self.file\_entry)

layout.addWidget(self.browse\_button)

layout.addWidget(self.class\_label)

layout.addWidget(self.class\_entry)

layout.addWidget(self.rows\_label)

layout.addWidget(self.rows\_entry)

layout.addWidget(self.columns\_label)

layout.addWidget(self.columns\_entry)

layout.addWidget(self.generate\_button)

self.parent.setLayout(layout)

def browse\_file(self):

filename, \_ = QFileDialog.getOpenFileName(self.parent, 'Select Excel file', '/', 'Excel files (\*.xlsx);;All files (\*.\*)')

self.file\_entry.setText(filename)

**Allocation\_module.py**

# design\_module.py

from PyQt5.QtWidgets import QLabel, QLineEdit, QPushButton, QVBoxLayout, QFileDialog

class DesignModule:

def \_\_init\_\_(self, parent):

self.parent = parent

self.init\_ui()

def init\_ui(self):

self.parent.setWindowTitle('Classroom Generator')

self.parent.setGeometry(100, 100, 400, 200)

self.file\_label = QLabel('Select Excel file:')

self.file\_entry = QLineEdit()

self.browse\_button = QPushButton('Browse', self.parent)

self.browse\_button.clicked.connect(self.browse\_file)

self.class\_label = QLabel('Students per Class:')

self.class\_entry = QLineEdit()

self.rows\_label = QLabel('Rows:')

self.rows\_entry = QLineEdit()

self.columns\_label = QLabel('Columns:')

self.columns\_entry = QLineEdit()

self.generate\_button = QPushButton('Generate Output', self.parent)

self.generate\_button.clicked.connect(self.parent.generate\_excel\_output)

layout = QVBoxLayout()

layout.addWidget(self.file\_label)

layout.addWidget(self.file\_entry)

layout.addWidget(self.browse\_button)

layout.addWidget(self.class\_label)

layout.addWidget(self.class\_entry)

layout.addWidget(self.rows\_label)

layout.addWidget(self.rows\_entry)

layout.addWidget(self.columns\_label)

layout.addWidget(self.columns\_entry)

layout.addWidget(self.generate\_button)

self.parent.setLayout(layout)

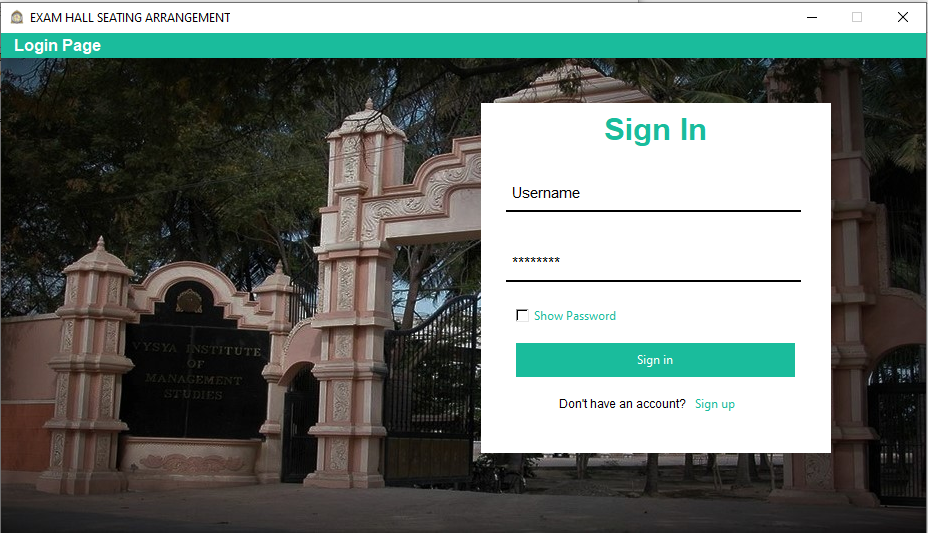
def browse\_file(self):

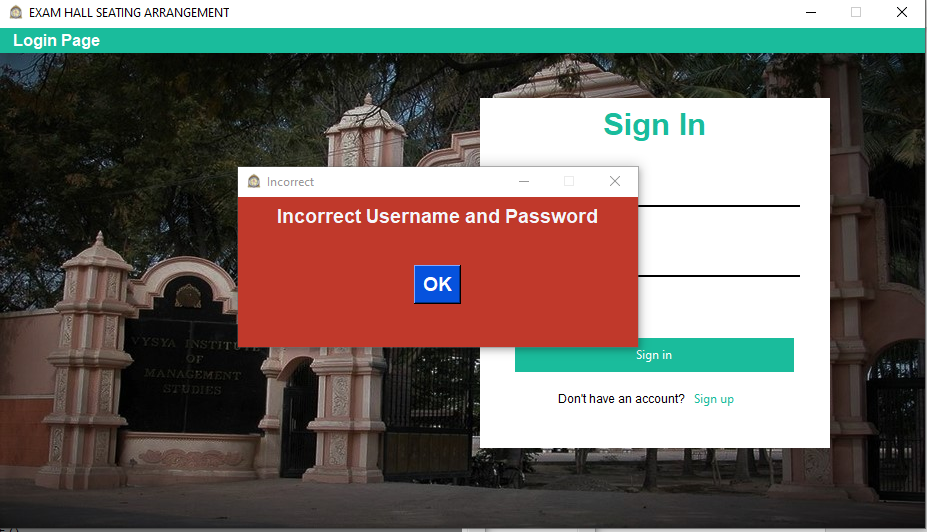
filename, \_ = QFileDialog.getOpenFileName(self.parent, 'Select Excel file', '/', 'Excel files (\*.xlsx);;All files (\*.\*)')

self.file\_entry.setText(filename)

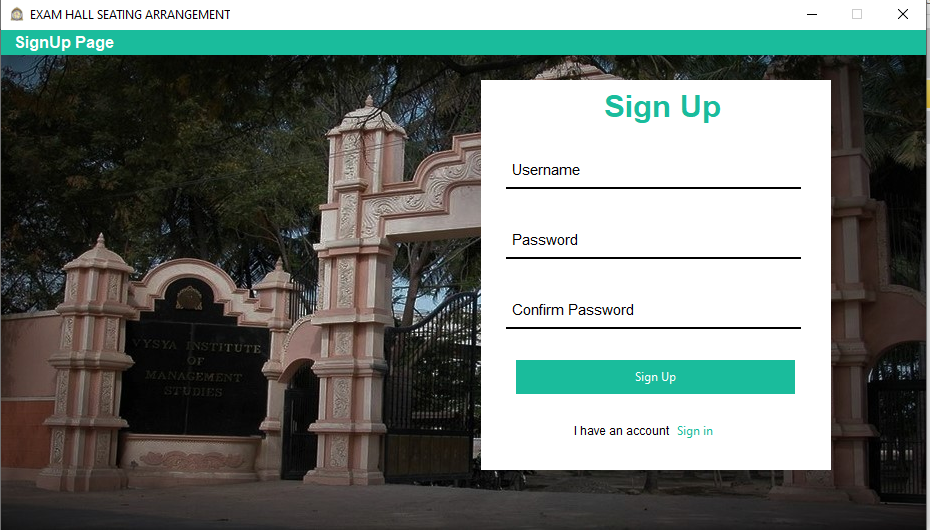
**SAMPLE INPUT AND OUTPUT**

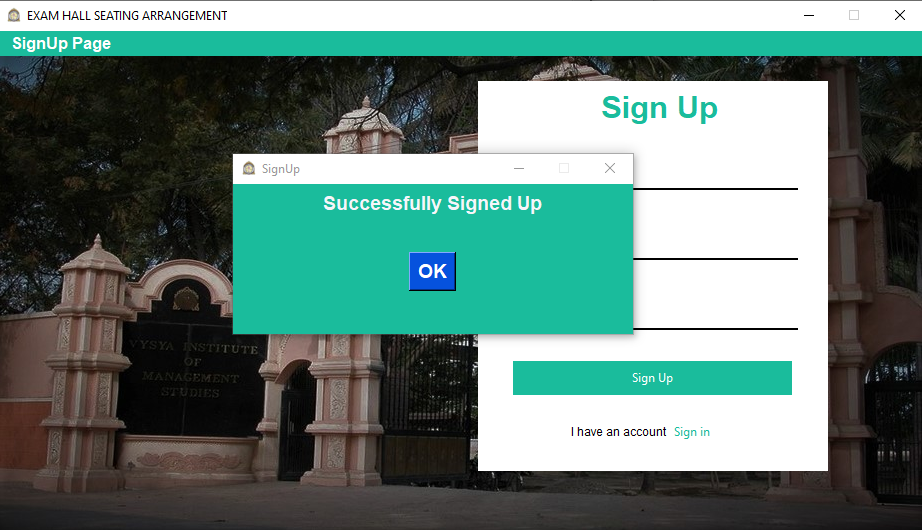
**Login Page**

****

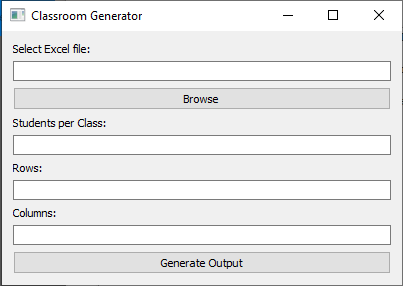
****

**SignUp Page**

****

****

**Home Page**

****