SMART ELECTRONIC OMR ANSWER SHEET [SEOAS]

Student's Unit:
Code:

```
import tkinter as tk
from tkinter import messagebox
import socket

def show_name_entry():
    instruction_frame.pack_forget()
    name_entry_frame.pack()

def show_omr_page():
    name = name_entry.get().strip()
    roll_number = roll_entry.get()

    if not name or not name.isalpha() or not name.isupper():
        messagebox.showerror("Error", "Please enter a valid name with only uppercase block letters.")
    elif not roll_number or not roll_number.isdigit() or len(roll_number) != 10:
        messagebox.showerror("Error", "Please enter a valid 10-digit roll number.")
    else:
        instruction_frame.pack_forget()
        name_entry_frame.pack_forget()
        omr_frame.pack()
        start_timer()

def start_timer():
    global remaining_time
    update_timer_label()
    countdown()

def update_timer_label():
    hours = remaining_time // 3600
```

```
    minutes = (remaining_time % 3600) // 60
    seconds = remaining_time % 60
    timer_label.config(text="Time Left: {:02d}:{:02d}:{:02d}".format(hours, minutes, seconds))

def countdown():
    global remaining_time
    if remaining_time > 0:
        remaining_time -= 1
        update_timer_label()
        root.after(1000, countdown)
    else:
        messagebox.showinfo("Time's Up!", "The exam time is over.")
        submit_omr()

def select_option(question_index, option_index):
    option_button = option_buttons[question_index][option_index]
    current_bg_color = option_button.cget("bg")
    if current_bg_color == "green":
        option_button.config(bg="white", activebackground="white")
    else:
        for btn in option_buttons[question_index]:
            btn.config(bg="white", activebackground="white")
        option_button.config(bg="green", activebackground="green")

def toggle_flag(question_index):
    flag_button = flag_buttons[question_index]
    current_bg_color = flag_button.cget("bg")
    new_color = "yellow" if current_bg_color != "yellow" else "white"
    flag_button.config(bg=new_color, activebackground=new_color)

def submit_omr():
    submitted_responses = []
```

```
    for i, question_options in enumerate(option_buttons):
        selected_option = None
        for j, option_button in enumerate(question_options):
            if option_button.cget("bg") == "green":
                selected_option = options[j]
                break
        submitted_responses.append("Question {}: {}".format(i + 1, selected_option if selected_option else 'Not answered'))

    result_window = tk.Toplevel()
    result_window.title("OMR Sheet Results")

    response_label = tk.Label(result_window, text="Your Responses:", font=("Helvetica", 14, "bold"))
    response_label.pack()

    num_responses = len(submitted_responses)
    middle = num_responses // 2

    left_responses = submitted_responses[:middle]
    right_responses = submitted_responses[middle:]

    left_frame = tk.Frame(result_window)
    left_frame.pack(side=tk.LEFT, padx=5)

    right_frame = tk.Frame(result_window)
    right_frame.pack(side=tk.LEFT, padx=5)

    for response in left_responses:
        response_label = tk.Label(left_frame, text=response, font=("Helvetica", 10))
        response_label.pack()

    for response in right_responses:
```

```
        response_label = tk.Label(right_frame, text=response, font=("Helvetica", 10))
        response_label.pack()

    confirmation = messagebox.askyesno("Confirmation", "Are you sure you want to submit the exam?")

    if confirmation:
        name = name_entry.get().strip()
        roll_number = roll_entry.get()
        file_name = "{}_{}_exam_results.txt".format(name, roll_number)
        save_results(name, roll_number, submitted_responses)
        messagebox.showinfo("Success", "Exam submitted successfully!")
        root.destroy()

def save_results(name, roll_number, submitted_responses):
    file_name = "{}_{}_exam_results.txt".format(name, roll_number)

    with open(file_name, "w") as file:
        file.write("\n".join(submitted_responses) + "\n")

        serverMACAddress = "d8:3a:dd:32:6c:91"  # Supervisor's Bluetooth address
        port = 30 # Bluetooth port

    try:
        # Create a Bluetooth socket
        s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
        s.connect((serverMACAddress, port))  # Connect to the supervisor's device
        s.send(file_name.encode())
        with open(file_name, 'rb') as f:
            exam_data = f.read()  # Read the content of the exam file

        # Construct the data to be sent over Bluetooth
```

```
        data_to_send ="\n\nOMR Sheet Responses:\n" + '\n'.join(submitted_responses)

        s.send(data_to_send.encode())  # Send the data over Bluetooth

        print("File sent successfully")
    except Exception as e:
        print("Error:", e)


def create_keyboard(frame, entry1, entry2):
    keyboard_frame = tk.Frame(frame)
    keyboard_frame.pack()

    keys = [
        "1234567890",
        "QWERTYUIOP",
        "ASDFGHJKL",
        "ZXCVBNM",
    ]

    active_entry = entry1

    def set_active_entry(entry):
        nonlocal active_entry
        active_entry = entry

    def on_key_press(char):
        if char == "Backspace":
            current_text = active_entry.get()
            active_entry.delete(0, tk.END)
            active_entry.insert(0, current_text[:-1])
        elif char == "Space":
```

```
            active_entry.insert(tk.END, " ")
        elif char == "Enter":
            active_entry.insert(tk.END, "\n")
        else:
            active_entry.insert(tk.END, char)

    for row in keys:
        row_frame = tk.Frame(keyboard_frame)
        row_frame.pack()

        for char in row:
            char_button = tk.Button(row_frame, text=char, width=3, height=1, command=lambda c=char:
on_key_press(c))
            char_button.pack(side=tk.LEFT)

    special_keys = ["Space", "Backspace", "Enter"]
    special_key_frame = tk.Frame(keyboard_frame)
    special_key_frame.pack()

    for key in special_keys:
        key_button = tk.Button(special_key_frame, text=key, width=6, height=1, command=lambda
k=key: on_key_press(k))
        key_button.pack(side=tk.LEFT)

    switch_entry_button = tk.Button(keyboard_frame, text="Switch Entry", command=lambda:
set_active_entry(entry1 if active_entry == entry2 else entry2))
    switch_entry_button.pack()

root = tk.Tk()
root.title("OMR SHEET")
root.geometry("800x480")
root.configure(bg="#f0f0f0")
```

```
remaining_time = 3600


instruction_frame = tk.Frame(root, bg="#f0f0f0")
instruction_frame.pack(fill='both', expand=True)


instruction_label = tk.Label(instruction_frame, text="Read the instructions carefully and follow the guidelines below:", font=("Helvetica", 18, "bold"), bg="#f0f0f0", anchor='w', justify='left')
instruction_label.pack(anchor='w', padx=10, pady=10)


instructions = [
    "1. Read each question carefully.",
    "2. Select the correct answer (A, B, C, or D) for each question.",
    "3. Answer all questions.",
    "4. Review your answers before submitting.",
    "5. Manage your time to complete all questions within the allocated time.",
    "6. Use the provided space for rough work or additional information.",
    "7. Do not close the exam window until you have submitted your OMR Sheet.",
    "8. A question that has been answered and marked for review will be considered for evaluation"
]


for item in instructions:
    instruction_item = tk.Label(instruction_frame, text=item, font=("Helvetica", 18), bg="#f0f0f0", anchor='w', justify='left')
    instruction_item.pack(anchor='w', padx=20)


next_button = tk.Button(instruction_frame, text="Next", command=show_name_entry, font=("Helvetica", 16), bg="#4CAF50", fg="white")
next_button.pack(pady=10)


name_entry_frame = tk.Frame(root, bg="#f0f0f0")
name_label = tk.Label(name_entry_frame, text="Enter your name (block letters):", font=("Helvetica", 18), bg="#f0f0f0")
name_label.pack()
```

```
name_entry = tk.Entry(name_entry_frame, font=("Helvetica", 18))
name_entry.pack(pady=10)

roll_label = tk.Label(name_entry_frame, text="Enter your roll number (exactly 10 digits):", font=("Helvetica", 18), bg="#f0f0f0")
roll_label.pack()

roll_entry = tk.Entry(name_entry_frame, font=("Helvetica", 18))
roll_entry.pack(pady=10)

start_button = tk.Button(name_entry_frame, text="Start Exam", command=show_omr_page, font=("Helvetica", 16), bg="#4CAF50", fg="white")
start_button.pack(pady=20)

omr_frame = tk.Frame(root, bg="#f0f0f0")

num_questions = 50
num_columns = 5
questions_per_column = 14
options = ["A", "B", "C", "D"]

option_buttons = []
flag_buttons = []

for i in range(num_questions):
    if i % questions_per_column == 0:
        column_frame = tk.Frame(omr_frame, bg="#f0f0f0")
        column_frame.pack(side=tk.LEFT, padx=5)

    question_frame = tk.Frame(column_frame, bg="#f0f0f0")
    question_frame.pack(pady=5)

    label = tk.Label(question_frame, text="{}.".format(i + 1), font=("Helvetica", 10), bg="#f0f0f0")
```

```
    label.grid(row=0, column=0)

    option_buttons_question = []

    for j, option in enumerate(options):
        option_button = tk.Button(question_frame, text=option, command=lambda i=i, j=j:
select_option(i, j), width=1, height=1, font=("Helvetica", 8), bd=2)
        option_button.grid(row=0, column=j + 1, padx=2)

        option_buttons_question.append(option_button)

    flag_button = tk.Button(question_frame, text="Flag", command=lambda i=i: toggle_flag(i),
width=1, height=1, font=("Helvetica", 8), bd=2)
    flag_button.grid(row=0, column=num_columns + 1, padx=2)
    flag_buttons.append(flag_button)

    option_buttons.append(option_buttons_question)

timer_label = tk.Label(column_frame, text="Time Left: 1:00:00", font=("Helvetica", 12, "bold"),
bg="#f0f0f0")
timer_label.pack(pady=10)

submit_button = tk.Button(column_frame, text="Submit OMR", command=submit_omr,
font=("Helvetica", 12), bg="#4CAF50", fg="white", bd=2)
submit_button.pack(pady=20)

create_keyboard(name_entry_frame, name_entry, roll_entry)

root.mainloop()
```

Supervisor’s Unit:

Code:

```
import socket
import os
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QTableWidget, \
    QTableWidgetItem, QComboBox, QStackedWidget, QMessageBox, QInputDialog, QDialog, QVBoxLayout, QPushButton, \
    QGridLayout
from PyQt5.QtCore import Qt
from PyQt5.QtCore import Qt, QTimer, QDateTime
from datetime import datetime
import uuid
from datetime import datetime
import sys


class BluetoothSupervisor:
    def __init__(self):
        pass


    def send_start_command(self, supervisor_mac_address):
        port = 30
        size = 1024
        start_command = "START"


        try:
            s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
            s.connect((supervisor_mac_address, port))
            s.send(start_command.encode())
            print("Start command sent successfully")
            QMessageBox.information(None, "Success", "Start command sent successfully")
        except Exception as e:
```

```
            print("Error:", e)
            QMessageBox.warning(None, "Error", f"Error: {e}")
        finally:
            s.close()


    def receive_data(self, supervisor_mac_address):
        port = 30
        backlog = 1
        size = 1024
        s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM,
socket.BTPROTO_RFCOMM)
        s.bind((supervisor_mac_address, port))
        s.listen(backlog)
        try:
            client, address = s.accept()
            print("Connection accepted")


            received_data = b""
            while True:
                data = client.recv(size)
                if not data:
                    break
                received_data += data


            print("Data received:", received_data.decode())
            return received_data.decode()
        except Exception as e:
            print("Error:", e)
        finally:
            s.close()



    def run_bluetooth_code():
```

```
        host_mac_address = "D8:3A:DD:32:6C:91"
        port = 15
        backlog = 1
        size = 1024
        s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM,
socket.BTPROTO_RFCOMM)
        s.bind((host_mac_address, port))
        s.listen(backlog)
        try:
            client, address = s.accept()
            print("Connection accepted")

        # Generate a unique file name based on current timestamp
            file_name=client.recv(1024).decode()
            file_name=file_name.strip()
            current_time = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
            file_name = f"received_file_{current_time}.txt"

            with open(file_name, 'wb') as f:
                print("File opened")
            while 1:
                data = client.recv(size)
                if not data:
                    break
                f.write(data)
                print("File received")
                client.close()
        except Exception as e:
            print(e)
        finally:
            s.close()

class BluetoothRunner(QWidget):
```

```
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Bluetooth Code Runner")
        self.setGeometry(100, 100, 300, 100)

        layout = QVBoxLayout()

        self.run_button = QPushButton("Run Bluetooth Code")
        self.run_button.clicked.connect(self.run_bluetooth)

        layout.addWidget(self.run_button)
        self.setLayout(layout)

    def run_bluetooth(self):
        run_bluetooth_code()
        print("Bluetooth code executed successfully!")

class VirtualKeyboard(QDialog):
    def __init__(self, target_field):
        super().__init__()

        self.target_field = target_field

        self.setWindowTitle("Virtual Keyboard")
        self.setGeometry(40, 40, 40, 40)

        layout = QGridLayout(self)

        buttons = [
            "1", "2", "3", "4", "5", "6", "7", "8", "9", "0",
            "Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P",
```

```
            "A", "S", "D", "F", "G", "H", "J", "K", "L",
            "Z", "X", "C", "V", "B", "N", "M",
            "Space", "Back"
        ]

        positions = [(i, j) for i in range(5) for j in range(10)]

        for button_text, pos in zip(buttons, positions):
            button = QPushButton(button_text)
            button.clicked.connect(lambda _, text=button_text: self.handle_button_click(text))
            layout.addWidget(button, *pos)

            # Increase the size of all buttons
            button.setFixedSize(50, 40)
            button.setStyleSheet("font-size: 12pt;")

        vbox = QVBoxLayout()
        vbox.addLayout(layout)
        self.setLayout(vbox)

    def handle_button_click(self, button_text):
        current_text = self.target_field.text()

        if button_text == "Space":
            current_text += " "
        elif button_text == "Back":
            current_text = current_text[:-1] if current_text else current_text
        else:
            current_text += button_text

        self.target_field.setText(current_text)
```

```
class SupervisorApp(QWidget):
    def __init__(self):
        super().__init__()
        #self.init_ui()



        # Replace the placeholder with your actual file path
        self.file_path = r"/home/seaos/Downloads/student_data.txt"

        self.students_data = {}
        self.started_exam_students = []  # Keep track of students who have started the exam

        self.setWindowTitle("Supervisor Unit")
        self.setGeometry(100, 100, 800, 600)

        self.stacked_widget = QStackedWidget(self)

        self.page1 = QWidget()  # Create page1 instance
        self.page2 = QWidget()  # Create page2 instance

        self.label = QLabel("Supervisor Unit", self.page1)
        self.label.setStyleSheet("font-size: 20pt; font-weight: bold; color: #333")

        layout1 = QVBoxLayout(self.page1)
        layout1.addWidget(self.label, alignment=Qt.AlignCenter)
        layout1.setSpacing(5)

        center_label = QLabel("Centre No.", self.page1)
        center_label.setStyleSheet("font-size: 18pt; font-weight: bold;")

        block_label = QLabel("Block No.", self.page1)
        block_label.setStyleSheet("font-size: 18pt; font-weight: bold;")
```

```
        self.center_entry = QLineEdit(self.page1)
        self.center_entry.setReadOnly(True)
        self.center_entry.setStyleSheet("background-color: #eee; padding: 5px; font-size: 20px;")
        center_button = QPushButton("Keyboard", self.page1)
        center_button.clicked.connect(lambda _, field=self.center_entry: self.show_keyboard(field))
        center_button.setFixedSize(80, 40)

        self.block_entry = QLineEdit(self.page1)
        self.block_entry.setReadOnly(True)
        self.block_entry.setStyleSheet("background-color: #eee; padding: 5px; font-size: 20px;")
        block_button = QPushButton("Keyboard", self.page1)
        block_button.clicked.connect(lambda _, field=self.block_entry: self.show_keyboard(field))
        block_button.setFixedSize(80, 40)

        next_button = QPushButton("Next", self.page1)
        next_button.clicked.connect(self.show_second_page)
        next_button.setStyleSheet("background-color: #4CAF50; color: white; font-size: 16px; padding: 10px; border: none; border-radius: 5px;")

        layout1.addWidget(self.label, alignment=Qt.AlignCenter)
        layout1.addWidget(center_label)
        layout1.addWidget(self.center_entry)
        layout1.addWidget(center_button)
        layout1.addWidget(block_label)
        layout1.addWidget(self.block_entry)
        layout1.addWidget(block_button)
        layout1.addWidget(next_button, alignment=Qt.AlignRight)

        self.page2 = QWidget(self)
        entrance_label = QLabel("Entrance Exam", self.page2)
        entrance_label.setStyleSheet("font-size: 18pt; font-weight: bold; color: #333")
```

```
        center_block_label = QLabel(self.page2)
        center_block_label.setObjectName('center_block_label')

        self.table = QTableWidget(self.page2)
        self.table.setColumnCount(5)
        self.table.setHorizontalHeaderLabels(["Name", "Seat Number", "Absent", "Start Exam", "Fetch Data"])

        self.table.verticalHeader().setVisible(False)

        self.entries = []
        self.absent_status = [False] * 20

        self.start_buttons = []
        self.absent_comboboxes = []
        self.fetch_data_buttons = []

        with open(self.file_path, "r") as file:
            lines = file.readlines()
            for i, line in enumerate(lines):
                substrings = self.split_string(line.strip(), ",")
                name = substrings[0] if len(substrings) > 0 else ""
                seat_number = substrings[1] if len(substrings) > 1 else ""
                self.create_student_entry(name, seat_number, i)

        self.submit_button = QPushButton("Submit", self.page2)
        self.submit_button.clicked.connect(self.submit_exam)
        self.submit_button.setStyleSheet("background-color: #4CAF50; color: white; font-size: 16px; padding: 10px; border: none; border-radius: 5px;")

        back_button = QPushButton("Back", self.page2)
        back_button.clicked.connect(self.show_first_page)
```

```
        back_button.setStyleSheet("background-color: #2196F3; color: white; font-size: 16px; padding: 10px; border: none; border-radius: 5px;")

        layout2 = QVBoxLayout(self.page2)
        layout2.addWidget(back_button, alignment=Qt.AlignLeft)
        layout2.addWidget(entrance_label, alignment=Qt.AlignCenter)
        layout2.addWidget(center_block_label, alignment=Qt.AlignCenter)
        layout2.addWidget(self.table)
        layout2.addWidget(self.submit_button, alignment=Qt.AlignRight)

        self.stacked_widget.addWidget(self.page1)
        self.stacked_widget.addWidget(self.page2)

        layout = QVBoxLayout(self)
        layout.addWidget(self.stacked_widget)

        self.stacked_widget.setCurrentIndex(0)
        self.clock_label = QLabel("", self.page2)
        self.clock_label.setStyleSheet("font-size: 18pt; font-weight: bold; color: #333")
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_time)
        self.timer.start(1000)

        layout2.addWidget(self.clock_label, alignment=Qt.AlignCenter)  # Add clock label to the layout

    def update_time(self):
        # Get the current date and time in the Kolkata timezone
        current_datetime = QDateTime.currentDateTime()
        current_time = current_datetime.time().toString(Qt.DefaultLocaleLongDate)

        # Update the clock label with the current time
        self.clock_label.setText(f"Time: {current_time}")
```

```
    def split_string(self, input_string, delimiter):
        substrings = input_string.split(delimiter)
        return substrings

    def show_second_page(self):
        self.stacked_widget.setCurrentIndex(1)

        self.center_number = self.center_entry.text()
        self.block_number = self.block_entry.text()
        center_block_text = f"Center No.: {self.center_number}, Block No.: {self.block_number}"
        self.page2.findChild(QLabel, 'center_block_label').setText(center_block_text)

    def show_first_page(self):
        self.stacked_widget.setCurrentIndex(0)

    def show_keyboard(self, target_field):
        virtual_keyboard = VirtualKeyboard(target_field)
        virtual_keyboard.exec_()

    def create_student_entry(self, default_name, seat_number, index):
        self.table.insertRow(index)

        name_item = QTableWidgetItem(default_name)
        name_item.setFlags(name_item.flags() ^ 2)  # Set item as non-editable (Qt.ItemIsEditable)

        seat_item = QTableWidgetItem(seat_number)

        # Create a combobox for "Absent" status
        absent_combobox = QComboBox(self)
        absent_combobox.addItems(["No", "Yes"])
        absent_combobox.currentIndexChanged.connect(lambda _, i=index:
self.toggle_start_exam_button(i))
        self.absent_comboboxes.append(absent_combobox)
```

```
        # Create a button for "Start Exam"
        start_exam_button = QPushButton("Start Exam", self)
        start_exam_button.clicked.connect(lambda _, i=index: self.start_exam_individual(i))
        start_exam_button.setEnabled(not self.absent_status[index])  # Enable only if not absent
        self.start_buttons.append(start_exam_button)

        # Create a button for "Fetch Data"
        fetch_data_button = QPushButton("Fetch Data", self)
        fetch_data_button.clicked.connect(lambda _, i=index: self.fetch_data_for_student(i))
        fetch_data_button.setEnabled(not self.absent_status[index])  # Enable only if not absent
        self.fetch_data_buttons.append(fetch_data_button)

        self.table.setItem(index, 0, name_item)
        self.table.setItem(index, 1, seat_item)
        self.table.setCellWidget(index, 2, absent_combobox)
        self.table.setCellWidget(index, 3, start_exam_button)
        self.table.setCellWidget(index, 4, fetch_data_button)

        # Set the size of the "Name" column to fit the content
        self.table.resizeColumnToContents(0)

    def toggle_start_exam_button(self, student_index):
        # Disable "Start Exam" button if student is marked as absent or already started
        self.start_buttons[student_index].setEnabled(
            self.absent_comboboxes[student_index].currentText() == "No" and student_index not in
self.started_exam_students
        )

        # Disable "Fetch Data" button if student is marked as absent

self.fetch_data_buttons[student_index].setEnabled(self.absent_comboboxes[student_index].currentTe
xt() == "No")
```

```
    def start_exam_individual(self, student_index):
        if not self.absent_status[student_index]:
            self.started_exam_students.append(student_index)
            self.start_buttons[student_index].setEnabled(False)  # Disable the button after starting the exam
            QMessageBox.information(self, "Exam Started",
                                    f"{self.table.item(student_index, 0).text()}'s exam has started.")
            # Send Bluetooth command to enable the next button on the other device
            supervisor_mac_address = "E4:5F:01:8B:B4:85"  # MAC address of the supervisor unit
            bluetooth_supervisor.send_start_command(supervisor_mac_address)
        else:
            QMessageBox.warning(self, "Absent Student",
                                f"{self.table.item(student_index, 0).text()} is marked as absent. Exam cannot be started.")

    def fetch_data_for_student(self, student_index):

        supervisor_mac_address = "D8:3A:DD:32:6C:91"  # MAC address of the supervisor unit
        received_data = bluetooth_supervisor.receive_data(supervisor_mac_address)
        QMessageBox.information(self, "Fetch Data", f"Data Received: {received_data}")

    def submit_exam(self):
        present_students_data = []  # Store data of present students

        for student_index, is_absent in enumerate(self.absent_status):
            if not is_absent:  # If student is present
                student_name = self.table.item(student_index, 0).text()
                seat_number = self.table.item(student_index, 1).text()
                absent_status = self.absent_comboboxes[student_index].currentText()

                student_info = f"Name: {student_name}, Seat Number: {seat_number}, Absent: {absent_status}"
```

```
                present_students_data.append(student_info)

        # Save present students' data to a text file
        file_name = "present_students_data.txt"
        with open(file_name, "w") as file:
            file.write("\n".join(present_students_data))

        # Display summary message
        total_students = len(self.absent_status)
        present_students = total_students - sum(self.absent_status)
        absent_students = sum(self.absent_status)

        result_message = f"Exam Submitted Successfully!\nTotal Students: {total_students}\nPresent Students: {present_students}\nAbsent Students: {absent_students}"
        QMessageBox.information(self, "Exam Summary", result_message)

        # Disable all widgets in the current page's layout
        self.disable_widgets_in_layout(self.page2.layout())

    def disable_widgets_in_layout(self, layout):
        # Disable all widgets within a layout
        for i in range(layout.count()):
            item = layout.itemAt(i)
            if isinstance(item, QWidgetItem):
                widget = item.widget()
                if widget:
                    widget.setEnabled(False)
                else:
                    self.disable_widgets_in_layout(item.layout())

bluetooth_supervisor = BluetoothSupervisor()

if __name__ == "__main__":
```

```
app = QApplication(sys.argv)
window = SupervisorApp()
window.show()
app.exec_()
```