Team Member 2: GUI & Interface Specialist

Your Role

You are responsible for creating all the visual interfaces using Tkinter - the login screen, main dashboard, and data entry forms.

Day 1 Tasks (6-8 hours)

Morning Session (3-4 hours)

1. Learn Tkinter Basics

Concepts to understand: - What is Tkinter and why it's perfect for desktop GUIs - Windows, frames, and widgets - Event handling (button clicks, form submissions) - Layout managers: pack(), grid(), place()

Quick Tutorial:

```
import tkinter as tk
from tkinter import ttk, messagebox

# Create a simple window
root = tk.Tk()
root.title("My First GUI")
root.geometry("400x300")

# Add widgets
label = tk.Label(root, text="Hello, Tkinter!", font=("Arial", 16))
label.pack(pady=20)

button = tk.Button(root, text="Click Me", command=lambda: messagebox.showinfo("Info", "
button.pack()

# Start the GUI
root.mainloop()
```

2. Study the Uploaded Materials

Look at these files from your materials: - **Building_Modern_UIs_with_CustomTkinter.pdf** - Modern GUI design - **Tkinter_1.ppt** and **tkinter_2_.pptx** - Basic Tkinter concepts

Key points: - Use frames to organize sections - Entry widgets for text input - Button widgets for actions - Listbox or Treeview for displaying records - messagebox for notifications

Afternoon Session (3-4 hours)

3. Design Your Interface Layout

Sketch or plan these screens: 1. **Login Window** - Username, password, login button, signup button 2. **Main Dashboard** - Menu, record list, action buttons 3. **Add/Edit Record Form** - Input fields, save/cancel buttons

4. Implement Your Code

File: gui_manager.py

```
"""
GUI Manager Module
Handles all graphical user interfaces for Smart Records System
Author: [Your Name] - Team Member 2
"""

import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
from typing import Optional, Callable
from datetime import datetime


class LoginWindow:
    """Login and Registration Window"""

    def __init__(self, on_login_success: Callable):
        """
        Initialize login window
        on_login_success: callback function(user_id, username) to call after successful
        """
        self.on_login_success = on_login_success
        self.window = tk.Tk()
        self.window.title("Smart Records System - Login")
        self.window.geometry("400x500")
        self.window.resizable(False, False)
```

```
        # Center the window
        self.center_window()

        # Configure colors
        self.bg_color = "#f0f0f0"
        self.accent_color = "#4CAF50"
        self.window.configure(bg=self.bg_color)

        self.create_login_interface()

    def center_window(self):
        """Center the window on screen"""
        self.window.update_idletasks()
        width = self.window.winfo_width()
        height = self.window.winfo_height()
        x = (self.window.winfo_screenwidth() // 2) - (width // 2)
        y = (self.window.winfo_screenheight() // 2) - (height // 2)
        self.window.geometry(f'{width}x{height}+{x}+{y}')

    def create_login_interface(self):
        """Create the login interface"""
        # Title Frame
        title_frame = tk.Frame(self.window, bg=self.accent_color, height=100
        title_frame.pack(fill='x')

        title_label = tk.Label(
            title_frame,
            text="Smart Records System",
            font=("Arial", 20, "bold"),
            bg=self.accent_color,
            fg="white"
        )
        title_label.pack(pady=30)

        # Main Frame
        main_frame = tk.Frame(self.window, bg=self.bg_color)
        main_frame.pack(expand=True, fill='both', padx=50, pady=30)

        # Username Field
        tk.Label(
            main_frame,
            text="Username:",
            font=("Arial", 12),
            bg=self.bg_color
        ).pack(anchor='w', pady=(0, 5))

        self.username_entry = tk.Entry(main_frame, font=("Arial", 12), width=30)
        self.username_entry.pack(pady=(0, 15))

        # Password Field
        tk.Label(
            main_frame,
```

```
            text="Password:",
            font=("Arial", 12),
            bg=self.bg_color
        ).pack(anchor='w', pady=(0, 5))

        self.password_entry = tk.Entry(main_frame, font=("Arial", 12), width=30, show="
        self.password_entry.pack(pady=(0, 25))

        # Bind Enter key to login
        self.password_entry.bind('<Return>', lambda e: self.login())

        # Login Button
        login_btn = tk.Button(
            main_frame,
            text="Login",
            font=("Arial", 12, "bold"),
            bg=self.accent_color,
            fg="white",
            width=25,
            height=2,
            cursor="hand2",
            command=self.login
        )
        login_btn.pack(pady=(0, 15))

        # Separator
        ttk.Separator(main_frame, orient='horizontal').pack(fill='x', pady=15)

        # Sign Up Button
        signup_btn = tk.Button(
            main_frame,
            text="Create New Account",
            font=("Arial", 11),
            bg=self.bg_color,
            fg=self.accent_color,
            cursor="hand2",
            border=0,
            command=self.show_signup_window
        )
        signup_btn.pack()

    def login(self):
        """Handle login button click"""
        username = self.username_entry.get().strip()
        password = self.password_entry.get()

        # Validation
        if not username or not password:
            messagebox.showerror("Error", "Please enter both username and password")
            return

        # Import database manager (will be provided by Team Member 1)
```

```
        try:
            from database_manager import DatabaseManager
            db = DatabaseManager()

            success, user_id, message = db.authenticate_user(username, password)

            if success:
                messagebox.showinfo("Success", message)
                self.window.destroy()
                self.on_login_success(user_id, username)
            else:
                messagebox.showerror("Login Failed", message)
                self.password_entry.delete(0, tk.END)

        except ImportError:
            # For testing without database
            messagebox.showwarning("Demo Mode", "Database not connected. Demo login suc
            self.window.destroy()
            self.on_login_success(1, username)
        except Exception as e:
            messagebox.showerror("Error", f"Login error: {str(e)}")

    def show_signup_window(self):
        """Show the signup window"""
        SignUpWindow(self.window)

    def run(self):
        """Start the GUI main loop"""
        self.window.mainloop()


class SignUpWindow:
    """Registration Window"""

    def __init__(self, parent):
        self.window = tk.Toplevel(parent)
        self.window.title("Create New Account")
        self.window.geometry("400x400")
        self.window.resizable(False, False)
        self.window.grab_set()  # Make it modal

        self.create_signup_interface()

    def create_signup_interface(self):
        """Create the signup interface"""
        # Title
        title_label = tk.Label(
            self.window,
            text="Create New Account",
            font=("Arial", 16, "bold")
        )
        title_label.pack(pady=20)
```

```
        # Form Frame
        form_frame = tk.Frame(self.window)
        form_frame.pack(padx=50, pady=10)

        # Full Name
        tk.Label(form_frame, text="Full Name:", font=("Arial", 11)).grid(row=0, column=
        self.fullname_entry = tk.Entry(form_frame, font=("Arial", 11), width=25)
        self.fullname_entry.grid(row=0, column=1, pady=10)

        # Username
        tk.Label(form_frame, text="Username:", font=("Arial", 11)).grid(row=1, column=0
        self.username_entry = tk.Entry(form_frame, font=("Arial", 11), width=25)
        self.username_entry.grid(row=1, column=1, pady=10)

        # Password
        tk.Label(form_frame, text="Password:", font=("Arial", 11)).grid(row=2, column=0
        self.password_entry = tk.Entry(form_frame, font=("Arial", 11), width=25, show="
        self.password_entry.grid(row=2, column=1, pady=10)

        # Confirm Password
        tk.Label(form_frame, text="Confirm Password:", font=("Arial", 11)).grid(row=3,
        self.confirm_entry = tk.Entry(form_frame, font=("Arial", 11), width=25, show="*
        self.confirm_entry.grid(row=3, column=1, pady=10)

        # Buttons Frame
        btn_frame = tk.Frame(self.window)
        btn_frame.pack(pady=20)

        signup_btn = tk.Button(
            btn_frame,
            text="Create Account",
            font=("Arial", 11, "bold"),
            bg="#4CAF50",
            fg="white",
            width=15,
            cursor="hand2",
            command=self.create_account
        )
        signup_btn.pack(side='left', padx=5)

        cancel_btn = tk.Button(
            btn_frame,
            text="Cancel",
            font=("Arial", 11),
            width=15,
            cursor="hand2",
            command=self.window.destroy
        )
        cancel_btn.pack(side='left', padx=5)

    def create_account(self):
```

```
        """Handle account creation"""
        fullname = self.fullname_entry.get().strip()
        username = self.username_entry.get().strip()
        password = self.password_entry.get()
        confirm = self.confirm_entry.get()

        # Validation
        if not all([fullname, username, password, confirm]):
            messagebox.showerror("Error", "All fields are required!")
            return

        if len(username) < 4:
            messagebox.showerror("Error", "Username must be at least 4 characters")
            return

        if len(password) < 6:
            messagebox.showerror("Error", "Password must be at least 6 characters")
            return

        if password != confirm:
            messagebox.showerror("Error", "Passwords do not match!")
            return

        # Try to create account
        try:
            from database_manager import DatabaseManager
            db = DatabaseManager()

            success, message = db.create_user(username, password, fullname)

            if success:
                messagebox.showinfo("Success", message)
                self.window.destroy()
            else:
                messagebox.showerror("Error", message)

        except ImportError:
            messagebox.showinfo("Demo", f"Account would be created for: {fullname}")
            self.window.destroy()
        except Exception as e:
            messagebox.showerror("Error", f"Signup error: {str(e)}")


class MainDashboard:
    """Main Application Dashboard"""

    def __init__(self, user_id: int, username: str):
        """
        Initialize main dashboard
        user_id: logged in user's ID
        username: logged in user's username
        """
```

```
        self.user_id = user_id
        self.username = username

        self.window = tk.Tk()
        self.window.title(f"Smart Records System - Welcome {username}")
        self.window.geometry("1000x600")

        # Center window
        self.center_window()

        self.create_dashboard()

    def center_window(self):
        """Center the window on screen"""
        self.window.update_idletasks()
        width = self.window.winfo_width()
        height = self.window.winfo_height()
        x = (self.window.winfo_screenwidth() // 2) - (width // 2)
        y = (self.window.winfo_screenheight() // 2) - (height // 2)
        self.window.geometry(f'{width}x{height}+{x}+{y}')

    def create_dashboard(self):
        """Create the main dashboard interface"""
        # Top Bar
        top_frame = tk.Frame(self.window, bg="#4CAF50", height=60)
        top_frame.pack(fill='x')
        top_frame.pack_propagate(False)

        tk.Label(
            top_frame,
            text=f"Welcome, {self.username}!",
            font=("Arial", 16, "bold"),
            bg="#4CAF50",
            fg="white"
        ).pack(side='left', padx=20)

        # Logout button
        logout_btn = tk.Button(
            top_frame,
            text="Logout",
            font=("Arial", 11),
            bg="white",
            command=self.logout
        )
        logout_btn.pack(side='right', padx=20)

        # Action Bar
        action_frame = tk.Frame(self.window, bg="#f5f5f5", height=60)
        action_frame.pack(fill='x')
        action_frame.pack_propagate(False)

        # Buttons
```

```
        tk.Button(
            action_frame,
            text="➕ Add Record",
            font=("Arial", 11, "bold"),
            bg="#4CAF50",
            fg="white",
            cursor="hand2",
            command=self.show_add_record
        ).pack(side='left', padx=10, pady=10)

        tk.Button(
            action_frame,
            text="🔄 Refresh",
            font=("Arial", 11),
            cursor="hand2",
            command=self.load_records
        ).pack(side='left', padx=5, pady=10)

        tk.Button(
            action_frame,
            text="📊 Generate Report",
            font=("Arial", 11),
            bg="#2196F3",
            fg="white",
            cursor="hand2",
            command=self.generate_report
        ).pack(side='left', padx=5, pady=10)

        # Search Frame
        search_frame = tk.Frame(action_frame, bg="#f5f5f5")
        search_frame.pack(side='right', padx=20)

        tk.Label(search_frame, text="Search:", bg="#f5f5f5", font=("Arial",

        self.search_var = tk.StringVar()
        self.search_var.trace('w', lambda *args: self.search_records())

        search_entry = tk.Entry(search_frame, textvariable=self.search_var,
        search_entry.pack(side='left')

        # Records Display Area
        display_frame = tk.Frame(self.window)
        display_frame.pack(expand=True, fill='both', padx=20, pady=20)

        # Create Treeview for records
        columns = ('ID', 'Title', 'Description', 'Category', 'Date', 'Status')
        self.tree = ttk.Treeview(display_frame, columns=columns, show='headings',

        # Define column headings
        self.tree.heading('ID', text='ID')
        self.tree.heading('Title', text='Title')
        self.tree.heading('Description', text='Description')
```

```
        self.tree.heading('Category', text='Category')
        self.tree.heading('Date', text='Date Added')
        self.tree.heading('Status', text='Status')

        # Define column widths
        self.tree.column('ID', width=50, anchor='center')
        self.tree.column('Title', width=200)
        self.tree.column('Description', width=300)
        self.tree.column('Category', width=120)
        self.tree.column('Date', width=150, anchor='center')
        self.tree.column('Status', width=100, anchor='center')

        # Add scrollbar
        scrollbar = ttk.Scrollbar(display_frame, orient='vertical', command=self.tree.y
        self.tree.configure(yscrollcommand=scrollbar.set)

        self.tree.pack(side='left', fill='both', expand=True)
        scrollbar.pack(side='right', fill='y')

        # Context menu for right-click
        self.tree.bind('<Button-3>', self.show_context_menu)
        self.tree.bind('<Double-1>', self.show_edit_record)

        # Bottom Status Bar
        status_frame = tk.Frame(self.window, bg="#e0e0e0", height=30)
        status_frame.pack(fill='x')
        status_frame.pack_propagate(False)

        self.status_label = tk.Label(
            status_frame,
            text="Ready",
            bg="#e0e0e0",
            font=("Arial", 9)
        )
        self.status_label.pack(side='left', padx=10)

        # Load initial records
        self.load_records()

    def load_records(self):
        """Load all records from database"""
        # Clear existing records
        for item in self.tree.get_children():
            self.tree.delete(item)

        try:
            from database_manager import DatabaseManager
            db = DatabaseManager()

            records = db.read_all_records(self.user_id)

            for record in records:
```

```
                # Format date
                date_str = record[4][:10] if len(record[4]) > 10 else record[4]

                self.tree.insert('', 'end', values=(
                    record[0],  # ID
                    record[1],  # Title
                    record[2][:50] + '...' if len(record[2]) > 50 else record[2],  # De
                    record[3],  # Category
                    date_str,   # Date
                    record[5]   # Status
                ))

            self.status_label.config(text=f"Loaded {len(records)} records")

        except ImportError:
            # Demo data
            demo_records = [
                (1, "Sample Record 1", "This is a demo record", "General", "2024-01-15"
                (2, "Sample Record 2", "Another demo", "Important", "2024-01-16", "Acti
            ]
            for record in demo_records:
                self.tree.insert('', 'end', values=record)
            self.status_label.config(text="Demo mode - 2 records")

        except Exception as e:
            messagebox.showerror("Error", f"Failed to load records: {str(e)}")

    def search_records(self):
        """Search records based on search box"""
        search_term = self.search_var.get().strip()

        # Clear existing
        for item in self.tree.get_children():
            self.tree.delete(item)

        try:
            from database_manager import DatabaseManager
            db = DatabaseManager()

            if search_term:
                records = db.search_records(self.user_id, search_term)
            else:
                records = db.read_all_records(self.user_id)

            for record in records:
                date_str = record[4][:10] if len(record[4]) > 10 else record[4]
                self.tree.insert('', 'end', values=(
                    record[0], record[1],
                    record[2][:50] + '...' if len(record[2]) > 50 else record[2],
                    record[3], date_str, record[5]
                ))
```

```
            self.status_label.config(text=f"Found {len(records)} records")

        except ImportError:
            self.load_records()  # Load demo data
        except Exception as e:
            messagebox.showerror("Error", f"Search failed: {str(e)}")

    def show_add_record(self):
        """Show add record dialog"""
        RecordFormWindow(self.window, self.user_id, callback=self.load_records)

    def show_edit_record(self, event=None):
        """Show edit record dialog"""
        selected = self.tree.selection()
        if not selected:
            messagebox.showinfo("Info", "Please select a record to edit")
            return

        item = self.tree.item(selected[0])
        record_data = item['values']

        RecordFormWindow(
            self.window,
            self.user_id,
            callback=self.load_records,
            edit_mode=True,
            record_id=record_data[0],
            existing_data={
                'title': record_data[1],
                'description': record_data[2],
                'category': record_data[3],
                'status': record_data[5]
            }
        )

    def show_context_menu(self, event):
        """Show right-click context menu"""
        # Select the item under cursor
        item = self.tree.identify_row(event.y)
        if item:
            self.tree.selection_set(item)

            menu = tk.Menu(self.window, tearoff=0)
            menu.add_command(label="Edit", command=self.show_edit_record)
            menu.add_command(label="Delete", command=self.delete_record)
            menu.post(event.x_root, event.y_root)

    def delete_record(self):
        """Delete selected record"""
        selected = self.tree.selection()
        if not selected:
            return
```

```
        item = self.tree.item(selected[0])
        record_id = item['values'][0]
        record_title = item['values'][1]

        confirm = messagebox.askyesno(
            "Confirm Delete",
            f"Are you sure you want to delete '{record_title}'?"
        )

        if confirm:
            try:
                from database_manager import DatabaseManager
                db = DatabaseManager()

                success, message = db.delete_record(record_id)

                if success:
                    messagebox.showinfo("Success", message)
                    self.load_records()
                else:
                    messagebox.showerror("Error", message)

            except ImportError:
                self.tree.delete(selected[0])
                messagebox.showinfo("Demo", "Record deleted (demo mode)")
            except Exception as e:
                messagebox.showerror("Error", f"Delete failed: {str(e)}")

    def generate_report(self):
        """Trigger report generation (will be handled by Team Member 4)"""
        try:
            from report_generator import ReportGenerator
            report_gen = ReportGenerator(self.user_id, self.username)
            report_gen.generate_full_report()
        except ImportError:
            messagebox.showinfo("Info", "Report generation module not yet integrated")
        except Exception as e:
            messagebox.showerror("Error", f"Report generation failed: {str(e)}")

    def logout(self):
        """Handle logout"""
        confirm = messagebox.askyesno("Logout", "Are you sure you want to logout?")
        if confirm:
            self.window.destroy()
            # Restart login window
            LoginWindow(lambda uid, uname: MainDashboard(uid, uname).run()).run()

    def run(self):
        """Start the main loop"""
        self.window.mainloop()
```

```
class RecordFormWindow:
    """Window for adding/editing records"""

    def __init__(self, parent, user_id: int, callback: Callable,
                 edit_mode: bool = False, record_id: int = None, existing_data: dict =
        """
        Initialize record form window
        edit_mode: True if editing existing record
        record_id: ID of record being edited
        existing_data: dict with existing record data
        """
        self.user_id = user_id
        self.callback = callback
        self.edit_mode = edit_mode
        self.record_id = record_id

        self.window = tk.Toplevel(parent)
        self.window.title("Edit Record" if edit_mode else "Add New Record")
        self.window.geometry("500x450")
        self.window.resizable(False, False)
        self.window.grab_set()

        self.create_form(existing_data)

    def create_form(self, existing_data: dict = None):
        """Create the form interface"""
        # Title
        title_text = "Edit Record" if self.edit_mode else "Add New Record"
        title_label = tk.Label(
            self.window,
            text=title_text,
            font=("Arial", 16, "bold")
        )
        title_label.pack(pady=20)

        # Form Frame
        form_frame = tk.Frame(self.window)
        form_frame.pack(padx=30, pady=10, fill='both', expand=True)

        # Title Field
        tk.Label(form_frame, text="Title:", font=("Arial", 11, "bold")).grid(row=0, col
        self.title_entry = tk.Entry(form_frame, font=("Arial", 11), width=35)
        self.title_entry.grid(row=0, column=1, pady=10, sticky='ew')

        # Description Field
        tk.Label(form_frame, text="Description:", font=("Arial", 11, "bold")).grid(row=
        self.description_text = scrolledtext.ScrolledText(form_frame, font=("Arial", 10
        self.description_text.grid(row=1, column=1, pady=10, sticky='ew')

        # Category Field
        tk.Label(form_frame, text="Category:", font=("Arial", 11, "bold")).grid(row=2,
```

```
        self.category_var = tk.StringVar()
        category_combo = ttk.Combobox(
            form_frame,
            textvariable=self.category_var,
            font=("Arial", 11),
            width=33,
            values=["General", "Important", "Personal", "Work", "Other"]
        )
        category_combo.grid(row=2, column=1, pady=10, sticky='ew')
        category_combo.set("General")

        # Status Field (only in edit mode)
        if self.edit_mode:
            tk.Label(form_frame, text="Status:", font=("Arial", 11, "bold")).grid(row=3
            self.status_var = tk.StringVar()
            status_combo = ttk.Combobox(
                form_frame,
                textvariable=self.status_var,
                font=("Arial", 11),
                width=33,
                values=["Active", "Inactive", "Completed"]
            )
            status_combo.grid(row=3, column=1, pady=10, sticky='ew')
            status_combo.set("Active")

        # Fill existing data if in edit mode
        if existing_data:
            self.title_entry.insert(0, existing_data.get('title', ''))
            self.description_text.insert('1.0', existing_data.get('description', ''))
            self.category_var.set(existing_data.get('category', 'General'))
            if self.edit_mode and 'status' in existing_data:
                self.status_var.set(existing_data['status'])

        form_frame.columnconfigure(1, weight=1)

        # Buttons Frame
        btn_frame = tk.Frame(self.window)
        btn_frame.pack(pady=20)

        save_btn = tk.Button(
            btn_frame,
            text="Save" if self.edit_mode else "Add Record",
            font=("Arial", 11, "bold"),
            bg="#4CAF50",
            fg="white",
            width=15,
            cursor="hand2",
            command=self.save_record
        )
        save_btn.pack(side='left', padx=5)

        cancel_btn = tk.Button(
```

```
            btn_frame,
            text="Cancel",
            font=("Arial", 11),
            width=15,
            cursor="hand2",
            command=self.window.destroy
        )
        cancel_btn.pack(side='left', padx=5)

    def save_record(self):
        """Save the record"""
        title = self.title_entry.get().strip()
        description = self.description_text.get('1.0', tk.END).strip()
        category = self.category_var.get()

        # Validation
        if not title:
            messagebox.showerror("Error", "Title is required!")
            return

        if not description:
            messagebox.showerror("Error", "Description is required!")
            return

        try:
            from database_manager import DatabaseManager
            db = DatabaseManager()

            if self.edit_mode:
                status = self.status_var.get()
                success, message = db.update_record(
                    self.record_id, title, description, category, status
                )
            else:
                success, message = db.create_record(
                    self.user_id, title, description, category
                )

            if success:
                messagebox.showinfo("Success", message)
                self.callback()  # Refresh the main list
                self.window.destroy()
            else:
                messagebox.showerror("Error", message)

        except ImportError:
            action = "updated" if self.edit_mode else "created"
            messagebox.showinfo("Demo", f"Record {action} (demo mode)")
            self.callback()
            self.window.destroy()
        except Exception as e:
            messagebox.showerror("Error", f"Save failed: {str(e)}")
```

```
# ============= MAIN PROGRAM =============
if __name__ == "__main__":
    def on_successful_login(user_id, username):
        """Callback after successful login"""
        MainDashboard(user_id, username).run()

    # Start with login window
    app = LoginWindow(on_successful_login)
    app.run()
```

Day 2 Tasks (4-6 hours)

Morning Session (2-3 hours)

5. Test Your Interface

Run your GUI:

```
python gui_manager.py
```

Test all features: - [ ] Login window displays correctly - [ ] Sign up window works - [ ] Main dashboard loads - [ ] Add record form works - [ ] Edit record (double-click) - [ ] Delete record (right-click) - [ ] Search functionality - [ ] Buttons respond correctly

Afternoon Session (2-3 hours)

6. Improve the Design

Add these enhancements: - Better colors and fonts - Icons on buttons (optional) - Tooltips for buttons - Loading indicators

7. Integration with Team

- Test with Team Member 1's database
- Help Team Member 4 add report button functionality
- Ensure smooth workflow

Checklist

Day 1

- [ ] Learn Tkinter basics
- [ ] Study provided materials
- [ ] Design interface layout
- [ ] Create login window
- [ ] Create main dashboard
- [ ] Create add/edit form
- [ ] Test basic functionality

Day 2

- [ ] Test all features thoroughly
- [ ] Fix any bugs
- [ ] Improve visual design
- [ ] Integrate with database
- [ ] Help with report button
- [ ] Prepare demo for presentation

Things to Search and Learn

1. **Tkinter Tutorial**: "Python Tkinter GUI tutorial"
2. **Treeview Widget**: "Tkinter Treeview table example"
3. **Modal Windows**: "Python Tkinter Toplevel modal window"
4. **Event Handling**: "Tkinter button click event"
5. **Layout Management**: "Tkinter pack vs grid vs place"

Common Issues & Solutions

Issue: Window doesn't center **Solution**: Call `center_window()` after `geometry()`

Issue: Entry field not clearing **Solution**: Use `.delete(0, tk.END)`

Issue: Treeview not displaying data **Solution**: Check `show='headings'` is set

Issue: Can't click buttons **Solution**: Check button `command` is set correctly

Your Contribution (for presentation)

"I designed and built all the user interfaces that you see. The login screen, main dashboard with the table view, and all the forms for adding and editing records. I made sure everything is user-friendly and intuitive to use."

Mark Breakdown for Your Work

- GUI Design & Functionality: 3 marks
- Code Quality & Documentation: 1 mark (shared)
- Total: 4 marks (27% of project)