```
import tkinter as tk

from tkinter import ttk, messagebox, filedialog

from datetime import datetime, timedelta

import random

import hashlib

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

from matplotlib.figure import Figure

import csv


# Constants (Amazon-inspired color palette)

COLOR_PRIMARY = "#131A22"

COLOR_SECONDARY = "#232F3E"

COLOR_ACCENT = "#FF9900"

COLOR_LIGHT = "#FFFFFF"

COLOR_DARK = "#000000"

COLOR_CARD = "#37475A"


ADMIN_USERNAME = "admin"

ADMIN_PASSWORD_HASH = hashlib.sha256("admin123".encode()).hexdigest()


class LoginPage:

    def __init__(self, root):

        self.root = root

        self.root.title("Employee Management System - Login")

        self.root.geometry("400x500")

        self.root.configure(bg=COLOR_PRIMARY)

        self.root.resizable(False, False)

        self.create_widgets()
    def create_widgets(self):
        header = tk.Frame(self.root, bg=COLOR_PRIMARY, pady=30)
        header.pack(fill="x")
        tk.Label(header, text="EMPLOYEE MANAGEMENT", font=("Arial", 18, "bold"), fg=COLOR_ACCENT, bg=COLOR_PRIMARY).pack()
        tk.Label(header, text="SYSTEM LOGIN", font=("Arial", 12), fg=COLOR_LIGHT, bg=COLOR_PRIMARY).pack(pady=5)
        form_frame = tk.Frame(self.root, bg=COLOR_PRIMARY, padx=30, pady=30)
        form_frame.pack(expand=True, fill="both")
        tk.Label(form_frame, text="Username", font=("Arial", 10), fg=COLOR_LIGHT, bg=COLOR_PRIMARY, anchor="w").pack(fill="x", pady=(10, 5))
        self.username_entry = tk.Entry(form_frame, font=("Arial", 12), bg=COLOR_SECONDARY, fg=COLOR_LIGHT, insertbackground=COLOR_ACCENT, relief="flat")
        self.username_entry.pack(fill="x", ipady=8)
        tk.Label(form_frame, text="Password", font=("Arial", 10), fg=COLOR_LIGHT, bg=COLOR_PRIMARY, anchor="w").pack(fill="x", pady=(15, 5))
        self.password_entry = tk.Entry(form_frame, font=("Arial", 12), bg=COLOR_SECONDARY, fg=COLOR_LIGHT, insertbackground=COLOR_ACCENT, relief="flat", show="•")
        self.password_entry.pack(fill="x", ipady=8)
        self.remember_var = tk.IntVar()
        tk.Checkbutton(form_frame, text="Remember me", variable=self.remember_var, font=("Arial", 9), fg=COLOR_LIGHT, bg=COLOR_PRIMARY, selectcolor=COLOR_DARK, activebackground=COLOR_PRIMARY, activeforeground=COLOR_LIGHT).pack(pady=10, anchor="w")
        login_btn = tk.Button(form_frame, text="LOGIN", font=("Arial", 12, "bold"), bg=COLOR_ACCENT, fg=COLOR_DARK, bd=0, command=self.attempt_login)
        login_btn.pack(fill="x", ipady=10, pady=(10, 0))
        tk.Label(form_frame, text="Forgot password?", font=("Arial", 9), fg=COLOR_ACCENT, bg=COLOR_PRIMARY, cursor="hand2").pack(pady=10)
        self.root.bind('<Return>', lambda event: self.attempt_login())
    def attempt_login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()
        if not username or not password:
            messagebox.showerror("Error", "Please enter both username and password")
            return
        password_hash = hashlib.sha256(password.encode()).hexdigest()
        if username == ADMIN_USERNAME and password_hash == ADMIN_PASSWORD_HASH:
            self.on_login_success()
        else:
            messagebox.showerror("Login Failed", "Invalid username or password")
            self.password_entry.delete(0, tk.END)

    def on_login_success(self):
        self.root.destroy()
        root = tk.Tk()
        app = EmployeeManagementSystem(root)
        root.mainloop()

class EmployeeManagementSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Employee Management System - Dashboard")
        self.root.state('zoomed')
        self.root.configure(bg=COLOR_PRIMARY)
        self.employees = []
        self.attendance = {}
        self.salaries = {}
        self.generate_sample_data()
        self.create_header()
        self.create_search_bar()
        self.create_main_container()
        self.create_footer()
        self.show_employee_dashboard()

    def generate_sample_data(self):
        names = ["Dharma", "Venu", "Bala", "Chinmaiah", "Dhanujaya", "Biswajith", "Irfan",
"Sai"]
        join_date = datetime(2023, 1, 1)
        self.employees = []
        for i, name in enumerate(names):
            emp_id = f"EMP{1000 + i}"
            status = random.choice(["Active", "On Leave", "Terminated"])
            self.employees.append({
                "id": emp_id,
                "name": name,
                "position": random.choice(["Manager", "Developer", "HR", "Designer"]),
                "department": random.choice(["IT", "HR", "Finance", "Operations"]),
                "join_date": (join_date + timedelta(days=random.randint(0, 365))).strftime("%Y-
%m-%d"),
                "status": status,
                "salary": random.randint(30000, 90000)
            })
        for i in range(30):
            date = (datetime.now() - timedelta(days=i)).strftime("%Y-%m-%d")
            self.attendance[date] = {}
            for emp in self.employees:
                if emp["status"] == "Active":
                    self.attendance[date][emp["id"]] = random.choice(["Present", "Absent", "Late"])
        for emp in self.employees:
            present_days = sum(1 for date in self.attendance if emp["id"] in self.attendance[date]
and self.attendance[date][emp["id"]] == "Present")
            self.salaries[emp["id"]] = {
                "base": emp["salary"],
                "present_days": present_days,
                "deductions": random.randint(0, 5000),
                "bonus": random.randint(0, 3000),
                "allowances": random.randint(2000, 8000),
                "payment_status": random.choice(["Paid", "Pending", "Processing"])
            }

    def create_header(self):
        header = tk.Frame(self.root, bg=COLOR_PRIMARY, padx=20, pady=10)
        header.pack(fill="x")
        tk.Label(header, text="Employee Management", font=("Arial", 20, "bold"),
fg=COLOR_LIGHT, bg=COLOR_PRIMARY).pack(side="left")
        nav_frame = tk.Frame(header, bg=COLOR_PRIMARY)
        nav_frame.pack(side="right")
        nav_buttons = [
            ("Dashboard", self.show_employee_dashboard),
            ("Employees", self.show_employee_list),
            ("Attendance", self.show_attendance_view),
            ("Payroll", self.show_payroll_view),
            ("Reports", self.show_reports_view)
        ]
        for text, command in nav_buttons:
            btn = tk.Button(nav_frame, text=text, command=command, bg=COLOR_PRIMARY,
fg=COLOR_LIGHT, bd=0, font=("Arial", 10), activebackground=COLOR_SECONDARY)
            btn.pack(side="left", padx=5)

    def create_search_bar(self):
        search_frame = tk.Frame(self.root, bg=COLOR_PRIMARY, padx=20, pady=10)
        search_frame.pack(fill="x")
        self.search_var = tk.StringVar()
        search_entry = tk.Entry(search_frame, textvariable=self.search_var, font=("Arial", 12),
bg=COLOR_SECONDARY, fg=COLOR_LIGHT, insertbackground=COLOR_ACCENT, relief="flat")
        search_entry.pack(fill="x", expand=True, ipady=8)
        search_entry.bind("<Return>", lambda e: self.perform_search())
        btn_frame = tk.Frame(search_frame, bg=COLOR_PRIMARY)
        btn_frame.pack(fill="x", pady=(5, 0))
        search_buttons = [
            ("Employees", self.show_employee_list),
            ("Attendance", self.show_attendance_view),
            ("Payroll", self.show_payroll_view),
            ("Reports", self.show_reports_view)
        ]
        for text, command in search_buttons:
            btn = tk.Button(btn_frame, text=text, command=command, bg=COLOR_SECONDARY,
fg=COLOR_ACCENT, bd=0, font=("Arial", 9), padx=10, pady=3)
            btn.pack(side="left", padx=2)

    def create_main_container(self):
        self.main_container = tk.Frame(self.root, bg=COLOR_PRIMARY)
        self.main_container.pack(expand=True, fill="both", padx=20, pady=10)
    def create_footer(self):
        footer = tk.Frame(self.root, bg=COLOR_PRIMARY, padx=20, pady=10)
        footer.pack(fill="x", side="bottom")
        tk.Label(footer, text="© 2023 Employee Management System", fg=COLOR_LIGHT,
bg=COLOR_PRIMARY, font=("Arial", 9)).pack(side="left")
        tk.Label(footer, text=f"Last updated: {datetime.now().strftime('%Y-%m-%d %H:%M')}",
fg=COLOR_LIGHT, bg=COLOR_PRIMARY, font=("Arial", 9)).pack(side="right")

    def clear_main_container(self):
        for widget in self.main_container.winfo_children():
            widget.destroy()

    def perform_search(self):
        query = self.search_var.get().lower()
        if not query:
            self.show_employee_dashboard()
            return
        self.clear_main_container()
        emp_results = [emp for emp in self.employees if query in emp["name"].lower() or query
in emp["id"].lower()]
        attendance_results = []
        for date, records in self.attendance.items():
            for emp_id, status in records.items():
                emp = next((e for e in self.employees if e["id"] == emp_id), None)
                if emp and (query in emp["name"].lower() or query in emp["id"].lower() or query in
status.lower()):
                    attendance_results.append((date, emp_id, emp["name"], status))
        notebook = ttk.Notebook(self.main_container)
        notebook.pack(expand=True, fill="both")
        if emp_results:
            emp_frame = tk.Frame(notebook, bg=COLOR_PRIMARY)
            notebook.add(emp_frame, text=f"Employees ({len(emp_results)})")
            columns = ("ID", "Name", "Position", "Department", "Join Date", "Status")
            tree = self.create_treeview(emp_frame, columns)
            for emp in emp_results:
                tree.insert("", "end", values=(emp["id"], emp["name"], emp["position"],
emp["department"], emp["join_date"], emp["status"]))
        if attendance_results:
            att_frame = tk.Frame(notebook, bg=COLOR_PRIMARY)
            notebook.add(att_frame, text=f"Attendance ({len(attendance_results)})")
            columns = ("Date", "Employee ID", "Name", "Status")
            tree = self.create_treeview(att_frame, columns)
            for record in attendance_results:
                tree.insert("", "end", values=record)
        if not emp_results and not attendance_results:
            tk.Label(self.main_container, text="No results found", fg=COLOR_LIGHT,
bg=COLOR_PRIMARY, font=("Arial", 14)).pack(expand=True)


    def show_employee_dashboard(self):
        self.clear_main_container()
        cards_frame = tk.Frame(self.main_container, bg=COLOR_PRIMARY)
        cards_frame.pack(fill="x", pady=(0, 20))
        card1 = tk.Frame(cards_frame, bg=COLOR_CARD, padx=20, pady=15, relief="raised",
bd=1)
        card1.pack(side="left", expand=True, fill="both", padx=5)
        tk.Label(card1, text=f"{len(self.employees)}", font=("Arial", 24, "bold"),
fg=COLOR_ACCENT, bg=COLOR_CARD).pack()
```

```
        tk.Label(card1, text="Total Employees", font=("Arial", 10), fg=COLOR_LIGHT,
bg=COLOR_CARD).pack()

        active_count = len([e for e in self.employees if e["status"] == "Active"])

        card2 = tk.Frame(cards_frame, bg=COLOR_CARD, padx=20, pady=15, relief="raised",
bd=1)

        card2.pack(side="left", expand=True, fill="both", padx=5)

        tk.Label(card2, text=f"{active_count}", font=("Arial", 24, "bold"), fg="#4CAF50",
bg=COLOR_CARD).pack()

        tk.Label(card2, text="Active Employees", font=("Arial", 10), fg=COLOR_LIGHT,
bg=COLOR_CARD).pack()

        today = datetime.now().strftime("%Y-%m-%d")

        present_today = sum(1 for emp_id in self.attendance.get(today, {}) if
self.attendance[today][emp_id] == "Present")

        card3 = tk.Frame(cards_frame, bg=COLOR_CARD, padx=20, pady=15, relief="raised",
bd=1)

        card3.pack(side="left", expand=True, fill="both", padx=5)

        tk.Label(card3, text=f"{present_today}/{active_count}", font=("Arial", 24, "bold"),
fg="#FFC107", bg=COLOR_CARD).pack()

        tk.Label(card3, text="Present Today", font=("Arial", 10), fg=COLOR_LIGHT,
bg=COLOR_CARD).pack()

        activity_frame = tk.Frame(self.main_container, bg=COLOR_PRIMARY)

        activity_frame.pack(expand=True, fill="both")

        emp_frame = tk.Frame(activity_frame, bg=COLOR_PRIMARY)

        emp_frame.pack(side="left", expand=True, fill="both", padx=5)

        tk.Label(emp_frame, text="Recent Employees", font=("Arial", 12, "bold"),
fg=COLOR_LIGHT, bg=COLOR_PRIMARY).pack(anchor="w", pady=(0, 10))

        columns = ("ID", "Name", "Department", "Join Date")

        tree = self.create_treeview(emp_frame, columns)

        for emp in sorted(self.employees, key=lambda x: x["join_date"], reverse=True)[:5]:

            tree.insert("", "end", values=(emp["id"], emp["name"], emp["department"],
emp["join_date"]))
```

```
        att_frame = tk.Frame(activity_frame, bg=COLOR_PRIMARY)
        att_frame.pack(side="left", expand=True, fill="both", padx=5)
        tk.Label(att_frame, text="Recent Attendance", font=("Arial", 12, "bold"),
fg=COLOR_LIGHT, bg=COLOR_PRIMARY).pack(anchor="w", pady=(0, 10))
        columns = ("Date", "Name", "Status")
        tree = self.create_treeview(att_frame, columns)
        recent_dates = sorted(self.attendance.keys(), reverse=True)[:5]
        for date in recent_dates:
            for emp_id, status in self.attendance[date].items():
                emp = next((e for e in self.employees if e["id"] == emp_id), None)
                if emp:
                    tree.insert("", "end", values=(date, emp["name"], status))

    def show_employee_list(self):
        self.clear_main_container()
        top_frame = tk.Frame(self.main_container, bg=COLOR_PRIMARY)
        top_frame.pack(fill="x", pady=(0, 10))
        add_btn = tk.Button(top_frame, text="Add Employee", bg=COLOR_ACCENT,
fg=COLOR_DARK, font=("Arial", 10, "bold"), command=self.add_employee_dialog)
        add_btn.pack(side="left", padx=5)
        remove_btn = tk.Button(top_frame, text="Remove Employee", bg="#E53935",
fg=COLOR_LIGHT, font=("Arial", 10, "bold"), command=self.remove_selected_employee)
        remove_btn.pack(side="left", padx=5)

        # New: Export Button
        export_btn = tk.Button(top_frame, text="Export to CSV", bg=COLOR_SECONDARY,
fg=COLOR_LIGHT, font=("Arial", 10, "bold"), command=self.export_employee_data)
        export_btn.pack(side="right", padx=5)
        columns = ("ID", "Name", "Position", "Department", "Join Date", "Status", "Salary")
        tree = self.create_treeview(self.main_container, columns)
        for emp in self.employees:
            tree.insert("", "end", values=(emp["id"], emp["name"], emp["position"],
emp["department"], emp["join_date"], emp["status"], emp["salary"]))
        self.employee_tree = tree

    def export_employee_data(self):
        """Exports the current employee data to a CSV file."""
        if not self.employees:
            messagebox.showinfo("Export Data", "No employee data to export.")
            return

        file_path = filedialog.asksaveasfilename(
            defaultextension=".csv",
            filetypes=[("CSV files", "*.csv"), ("All files", "*.*")],
            title="Export Employee Data"
        )
        if not file_path:
            return  # User cancelled the dialog

        try:
            with open(file_path, 'w', newline='', encoding='utf-8') as file:
                writer = csv.writer(file)
                # Write header row
                headers = ["ID", "Name", "Position", "Department", "Join Date", "Status", "Salary"]
                writer.writerow(headers)
                # Write employee data
            for emp in self.employees:
                writer.writerow([
                    emp["id"],
                    emp["name"],
                    emp["position"],
                    emp["department"],
                    emp["join_date"],
                    emp["status"],
                    emp["salary"]
                ])
            messagebox.showinfo("Export Successful", f"Employee data exported
to:\n{file_path}")
        except Exception as e:
            messagebox.showerror("Export Error", f"Failed to export data: {e}")



    def add_employee_dialog(self):
        dialog = tk.Toplevel(self.root)
        dialog.title("Add Employee")
        dialog.configure(bg=COLOR_PRIMARY)
        dialog.geometry("300x350")
        fields = ["Name", "Position", "Department", "Join Date (YYYY-MM-DD)", "Status",
"Salary"]
        entries = {}
        for idx, field in enumerate(fields):
            tk.Label(dialog, text=field, bg=COLOR_PRIMARY, fg=COLOR_LIGHT).pack(anchor="w",
pady=(10 if idx == 0 else 5, 0))
            entry = tk.Entry(dialog, bg=COLOR_SECONDARY, fg=COLOR_LIGHT)
            entry.pack(fill="x", padx=10)
            entries[field] = entry
        def submit():
            name = entries["Name"].get()
            position = entries["Position"].get()
            department = entries["Department"].get()
            join_date = entries["Join Date (YYYY-MM-DD)"].get()
            status = entries["Status"].get()
            salary = entries["Salary"].get()
            if not all([name, position, department, join_date, status, salary]):
            messagebox.showerror("Error", "All fields are required", parent=dialog)
            return
            emp_id = f"EMP{1000 + len(self.employees)}"
            try:
            salary = int(salary)
            except ValueError:
            messagebox.showerror("Error", "Salary must be a number", parent=dialog)
            return
            self.employees.append({
            "id": emp_id,
            "name": name,
            "position": position,
            "department": department,
            "join_date": join_date,
            "status": status,
            "salary": salary
            })
            dialog.destroy()
            self.show_employee_list()
```

```
        tk.Button(dialog, text="Add", bg=COLOR_ACCENT, fg=COLOR_DARK,
command=submit).pack(pady=20)

    def remove_selected_employee(self):
        tree = self.employee_tree
        selected = tree.selection()
        if not selected:
            messagebox.showerror("Error", "Please select an employee to remove")
            return
        emp_id = tree.item(selected[0])["values"][0]
        self.employees = [emp for emp in self.employees if emp["id"] != emp_id]
        self.show_employee_list()

    def create_treeview(self, parent, columns):
        style = ttk.Style()
        style.configure("Treeview", background=COLOR_SECONDARY,
foreground=COLOR_LIGHT, fieldbackground=COLOR_SECONDARY, rowheight=28,
font=("Arial", 10))
        style.map("Treeview", background=[('selected', COLOR_ACCENT)],
foreground=[('selected', COLOR_DARK)])
        tree = ttk.Treeview(parent, columns=columns, show="headings", selectmode="browse")
        tree.pack(expand=True, fill="both")
        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, anchor="center", width=120)
        return tree

    def show_attendance_view(self):
        self.clear_main_container()
        tk.Label(self.main_container, text="Employee Attendance Summary", font=("Arial", 14,
"bold"),
                 fg=COLOR_ACCENT, bg=COLOR_PRIMARY).pack(anchor="w", pady=(0, 10))


        # Calculate attendance stats
        working_days = sorted(self.attendance.keys())
        total_working_days = len(working_days)
        columns = ("Employee ID", "Name", "Total Working Days", "Present", "Absent", "Late",
"Attendance %")
        tree = self.create_treeview(self.main_container, columns)


        for emp in self.employees:
            emp_id = emp["id"]
            present = absent = late = 0
            for date in working_days:
                status = self.attendance.get(date, {}).get(emp_id)
                if status == "Present":
                    present += 1
                elif status == "Absent":
                    absent += 1
                elif status == "Late":
                    late += 1
            attendance_percent = (present / total_working_days * 100) if total_working_days
else 0
            tree.insert("", "end", values=(
                emp_id, emp["name"], total_working_days, present, absent, late,
f"{attendance_percent:.1f}%"
            ))
```

```
def show_payroll_view(self):
    self.clear_main_container()
    tk.Label(self.main_container, text="Employee Payroll Details", font=("Arial", 14, "bold"),
             fg=COLOR_ACCENT, bg=COLOR_PRIMARY).pack(anchor="w", pady=(0, 10))

    # Add buttons for payroll actions
    btn_frame = tk.Frame(self.main_container, bg=COLOR_PRIMARY)
    btn_frame.pack(fill="x", pady=(0, 10))

    tk.Button(btn_frame, text="Generate Payslips", bg=COLOR_ACCENT, fg=COLOR_DARK,
              command=self.generate_payslips).pack(side="left", padx=5)
    tk.Button(btn_frame, text="Process Payroll", bg="#4CAF50", fg=COLOR_LIGHT,
              command=self.process_payroll).pack(side="left", padx=5)

    columns = ("Employee ID", "Name", "Basic Pay", "Allowances", "Deductions",
               "Bonus", "Net Pay", "Payment Status")
    tree = self.create_treeview(self.main_container, columns)

    for emp in self.employees:
        emp_id = emp["id"]
        salary_info = self.salaries.get(emp_id, {})

        # Get or calculate salary components
        basic = salary_info.get("base", emp["salary"])
        allowances = salary_info.get("allowances", 0)
        deductions = salary_info.get("deductions", 0)
        bonus = salary_info.get("bonus", 0)
        net_pay = basic + allowances + bonus - deductions
        payment_status = salary_info.get("payment_status", "Pending")

        tree.insert("", "end", values=(
            emp_id, emp["name"],
            f"₹{basic:,}",
            f"₹{allowances:,}",
            f"₹{deductions:,}",
            f"₹{bonus:,}",
            f"₹{net_pay:,}",
            payment_status
        ))

def generate_payslips(self):
    """Generate payslips for all employees"""
    selected_employees = self.get_selected_employees()
    if not selected_employees:
        messagebox.showinfo("Generate Payslips", "Generating payslips for all employees")
        selected_employees = self.employees

    for emp in selected_employees:
        emp_id = emp["id"]
        salary_info = self.salaries.get(emp_id, {})

        # Generate payslip content
        payslip_content = f"""
        === PAYSLIP ===
        Employee ID: {emp_id}
        Name: {emp['name']}
```

```
            Position: {emp['position']}
            Department: {emp['department']}
            Date: {datetime.now().strftime('%Y-%m-%d')}

            Earnings:
            - Basic Salary: ₹{salary_info.get('base', emp['salary']):,}
            - Allowances: ₹{salary_info.get('allowances', 0):,}
            - Bonus: ₹{salary_info.get('bonus', 0):,}

            Deductions:
            - Tax/Other: ₹{salary_info.get('deductions', 0):,}

            Net Pay: ₹{(salary_info.get('base', emp['salary']) +
                       salary_info.get('allowances', 0) +
                       salary_info.get('bonus', 0) -
                       salary_info.get('deductions', 0)):,}
            """

            # In a real app, you would save this to a file or database
            print(f"Generated payslip for {emp['name']}")

        messagebox.showinfo("Success", f"Generated payslips for {len(selected_employees)}
employees")

    def process_payroll(self):
        """Process payroll for selected employees"""
        selected_employees = self.get_selected_employees()
        if not selected_employees:
```

```
        if messagebox.askyesno("Confirm", "Process payroll for ALL employees?"):
            selected_employees = self.employees
        else:
            return

    total_amount = 0
    for emp in selected_employees:
        emp_id = emp["id"]
        salary_info = self.salaries.get(emp_id, {})
        net_pay = (salary_info.get('base', emp['salary']) +
                     salary_info.get('allowances', 0) +
                     salary_info.get('bonus', 0) -
                     salary_info.get('deductions', 0))
        total_amount += net_pay

        # Mark as paid (in a real app, you'd update database)
        salary_info["payment_status"] = "Paid"
        self.salaries[emp_id] = salary_info

    messagebox.showinfo("Payroll Processed",
                f"Processed payroll for {len(selected_employees)} employees\n"
                f"Total amount disbursed: ₹{total_amount:,}")
    self.show_payroll_view()  # Refresh the view

def get_selected_employees(self):
    """Helper method to get selected employees from treeview"""
    # In a real implementation, this would get selected rows from the treeview
    # For this demo, we'll return None to process all employees
```

```
        return None

    def show_reports_view(self):
        self.clear_main_container()

        # Main title
        tk.Label(self.main_container, text="Employee Reports Dashboard",
                 font=("Arial", 16, "bold"), fg=COLOR_ACCENT, bg=COLOR_PRIMARY).pack(pady=(0,
20))

        # Create a notebook for multiple report tabs
        notebook = ttk.Notebook(self.main_container)
        notebook.pack(expand=True, fill="both", padx=10, pady=10)

        # Tab 1: Department Distribution
        dept_frame = tk.Frame(notebook, bg=COLOR_PRIMARY)
        notebook.add(dept_frame, text="Department Stats")
        self.create_department_chart(dept_frame)

        # Tab 2: Salary Distribution
        salary_frame = tk.Frame(notebook, bg=COLOR_PRIMARY)
        notebook.add(salary_frame, text="Salary Analysis")
        self.create_salary_chart(salary_frame)

        # Tab 3: Attendance Trends
        att_frame = tk.Frame(notebook, bg=COLOR_PRIMARY)
        notebook.add(att_frame, text="Attendance Trends")
        self.create_attendance_chart(att_frame)
```

```
def create_department_chart(self, parent):
    """Create bar chart showing employee distribution by department"""
    # Calculate department counts
    dept_counts = {}
    for emp in self.employees:
        dept = emp["department"]
        dept_counts[dept] = dept_counts.get(dept, 0) + 1

    # Create figure
    fig = Figure(figsize=(6, 4), dpi=100, facecolor=COLOR_SECONDARY)
    ax = fig.add_subplot(111)
    ax.set_facecolor(COLOR_SECONDARY)

    # Customize colors and styles
    departments = list(dept_counts.keys())
    counts = list(dept_counts.values())
    colors = [COLOR_ACCENT, "#FFC107", "#4CAF50", "#2196F3", "#9C27B0"]

    # Create bar chart
    bars = ax.bar(departments, counts, color=colors[:len(departments)])
    ax.set_title('Employees by Department', color=COLOR_LIGHT, pad=20)
    ax.set_xlabel('Department', color=COLOR_LIGHT)
    ax.set_ylabel('Number of Employees', color=COLOR_LIGHT)

    # Customize appearance
    ax.tick_params(axis='x', colors=COLOR_LIGHT)
    ax.tick_params(axis='y', colors=COLOR_LIGHT)
    for spine in ax.spines.values():
        spine.set_color(COLOR_LIGHT)

    # Add value labels on bars
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{int(height)}', ha='center', va='bottom',
                color=COLOR_LIGHT, fontweight='bold')

    # Embed in Tkinter
    canvas = FigureCanvasTkAgg(fig, master=parent)
    canvas.draw()
    canvas.get_tk_widget().pack(expand=True, fill="both", padx=10, pady=10)

def create_salary_chart(self, parent):
    """Create bar chart showing salary distribution"""
    # Prepare salary data
    salaries = [emp["salary"] for emp in self.employees]
    names = [emp["name"] for emp in self.employees]

    # Create figure
    fig = Figure(figsize=(6, 4), dpi=100, facecolor=COLOR_SECONDARY)
    ax = fig.add_subplot(111)
    ax.set_facecolor(COLOR_SECONDARY)

    # Create horizontal bar chart
    y_pos = range(len(names))
    bars = ax.barh(y_pos, salaries, color=COLOR_ACCENT)
    ax.set_title('Employee Salaries', color=COLOR_LIGHT, pad=20)
    ax.set_yticks(y_pos)
    ax.set_yticklabels(names, color=COLOR_LIGHT)
    ax.set_xlabel('Salary (₹)', color=COLOR_LIGHT)

    # Customize appearance
    ax.tick_params(axis='x', colors=COLOR_LIGHT)
    for spine in ax.spines.values():
        spine.set_color(COLOR_LIGHT)

    # Add salary values
    for i, (salary, bar) in enumerate(zip(salaries, bars)):
        ax.text(bar.get_width() + 5000, bar.get_y() + bar.get_height()/2,
                f'₹{salary:,}', va='center', color=COLOR_LIGHT)

    # Embed in Tkinter
    canvas = FigureCanvasTkAgg(fig, master=parent)
    canvas.draw()
    canvas.get_tk_widget().pack(expand=True, fill="both", padx=10, pady=10)

def create_attendance_chart(self, parent):
    """Create bar chart showing attendance trends"""
    # Calculate attendance percentages
    emp_attendance = []
    working_days = sorted(self.attendance.keys())
    total_days = len(working_days)
```

```
        for emp in self.employees:
            present_days = sum(1 for date in working_days
                               if self.attendance[date].get(emp["id"]) == "Present")
            percentage = (present_days / total_days * 100) if total_days > 0 else 0
            emp_attendance.append((emp["name"], percentage))

        # Sort by attendance percentage
        emp_attendance.sort(key=lambda x: x[1])

        # Create figure
        fig = Figure(figsize=(6, 4), dpi=100, facecolor=COLOR_SECONDARY)
        ax = fig.add_subplot(111)
        ax.set_facecolor(COLOR_SECONDARY)

        # Create bar chart
        names = [x[0] for x in emp_attendance]
        percentages = [x[1] for x in emp_attendance]
        colors = ["#4CAF50" if p >= 90 else "#FFC107" if p >= 75 else "#E53935" for p in
percentages]

        bars = ax.bar(names, percentages, color=colors)
        ax.set_title('Attendance Percentage (Last 30 Days)', color=COLOR_LIGHT, pad=20)
        ax.set_ylabel('Attendance %', color=COLOR_LIGHT)
        ax.set_ylim(0, 100)

        # Customize appearance
        ax.tick_params(axis='x', colors=COLOR_LIGHT, rotation=45)
        ax.tick_params(axis='y', colors=COLOR_LIGHT)
        for spine in ax.spines.values():
            spine.set_color(COLOR_LIGHT)

        # Add percentage labels
        for bar in bars:
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                    f'{height:.1f}%', ha='center', va='bottom',
                    color=COLOR_LIGHT, fontweight='bold')

        # Embed in Tkinter
        canvas = FigureCanvasTkAgg(fig, master=parent)
        canvas.draw()
        canvas.get_tk_widget().pack(expand=True, fill="both", padx=10, pady=10)

if __name__ == "__main__":
    root = tk.Tk()
    LoginPage(root)
    root.mainloop()
```