```python
import datetime
import re

import streamlit as st
import pandas as pd
import csv
import os
import time
import matplotlib.pyplot as plt
import seaborn as sns
import random

from classes import * # import classes

st.set_page_config(
    page_title="Employee Performance Tracker",
    layout="wide"
)

# Initialize session state
if 'authenticated' not in st.session_state:
    st.session_state.authenticated = False
if 'current_user' not in st.session_state:
    st.session_state.current_user = None
if 'current_call' not in st.session_state:
    st.session_state.current_call = None
if 'workday_started' not in st.session_state:
    st.session_state.workday_started = False

DATA_DIR = "data"
os.makedirs(DATA_DIR, exist_ok=True)

# File paths
STAFF_FILE = os.path.join(DATA_DIR, "staff_details.csv")
CALLS_FILE = os.path.join(DATA_DIR, "call_details.csv")
TEAMS_FILE = os.path.join(DATA_DIR, "team_details.csv")
MANAGERS_FILE = os.path.join(DATA_DIR, "manager_details.csv")


# Initialize CSV files if they don't exist
def initialize_files():
    """Initialize CSV files with default data if they don't exist.

    Creates four CSV files (staff_details.csv, call_details.csv, team_details.csv,
    manager_details.csv) in the data directory with sample data if they don't already exist.
    """
    if not os.path.exists(STAFF_FILE):
        with open(STAFF_FILE, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['staff_id', 'first_name', 'last_name', 'manager_id', 'calls_taken',
                             'successful_calls', 'failed_calls', 'target_successful_calls',
                             'working_time_elapsed', 'avg_sat_score', 'status', 'team_id'])
            # Sample staff data
            writer.writerow(['101', 'John', 'Doe', '1', '2', '0', '0', '10', '0', '0.825', 'Free', '1'])
            writer.writerow(['102', 'Jane', 'Smith', '1', '1', '0', '0', '10', '0', '0.8', 'Free', '1'])
            writer.writerow(['201', 'Mike', 'Johnson', '2', '1', '0', '0', '10', '0', '0.6', 'Free', '2'])

    if not os.path.exists(CALLS_FILE):
        with open(CALLS_FILE, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['call_id', 'status', 'time_elapsed', 'sat_score', 'handler_id', 'date', 'team_id'])
            # Sample call data
            now = int(time.time())
            writer.writerow([f'{now}001', 'Completed', '120', '0.95', '101', '26/06/2025 15:36', '1'])
            writer.writerow([f'{now}002', 'Completed', '180', '0.7', '101', '19/07/2025 18:16', '1'])
            writer.writerow([f'{now}003', 'Completed', '90', '0.8', '102', '07/07/2025 08:07', '1'])
            writer.writerow([f'{now}004', 'Completed', '150', '0.6', '201', '10/07/2025 16:49', '2'])

    if not os.path.exists(TEAMS_FILE):
        with open(TEAMS_FILE, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['team_id', 'team_name', 'manager_id'])
            writer.writerow(['1', 'Customer Support East', '1'])
            writer.writerow(['2', 'Customer Support West', '2'])

    if not os.path.exists(MANAGERS_FILE):
        with open(MANAGERS_FILE, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['manager_id', 'manager_first_name', 'manager_last_name', 'staff_list'])
            writer.writerow([1, 'David', 'Cooper', [101, 102]])
            writer.writerow([2, 'Shirley', 'McDonald', [201]])


# Load data functions with class instantiation
@st.cache_data
def load_staff_data() -> tuple[pd.DataFrame, list[Staff]]:
    """Load staff data from CSV file and create Staff objects.

    Returns:
        tuple: A tuple containing:
            - pd.DataFrame: DataFrame with raw staff data
            - list[Staff]: List of Staff objects initialized with the data
    """
    df = pd.read_csv(STAFF_FILE)
    staff_objects = []
    for _, row in df.iterrows():
        staff = Staff(
            id=row['staff_id'],
            first_name=row['first_name'],
            last_name=row['last_name'],
            manager_id=row['manager_id'],
            calls_taken=row['calls_taken'],
            successful_calls=row['successful_calls'],
            failed_calls=row['failed_calls'],
            target_successful_calls=row['target_successful_calls'],
            working_time_elapsed=row['working_time_elapsed'],
            avg_sat_score=row['avg_sat_score'],
            status=row['status']
        )
        staff_objects.append(staff)
    return df, staff_objects

@st.cache_data
def load_calls_data() -> tuple[pd.DataFrame, list[Call]]:
    """Load call data from CSV file and create Call objects.

    Returns:
        tuple: A tuple containing:
            - pd.DataFrame: DataFrame with raw call data (includes datetime conversion)
            - list[Call]: List of Call objects initialized with the data
    """
    df = pd.read_csv(CALLS_FILE)
    df['datetime'] = pd.to_datetime(df['date'], format='%d/%m/%Y %H:%M')
    call_objects = []
    for _, row in df.iterrows():
        call = Call(
            id=row['call_id'],
            status=row['status'],
            time_elapsed=row['time_elapsed'],
            sat_score=row['sat_score'],
            handler_id=row['handler_id']
        )
        call.datetime = row['datetime']  # Add datetime to call object
        call_objects.append(call)
    return df, call_objects


@st.cache_data
def load_teams_data() -> pd.DataFrame:
    """Load team data from CSV file.

    Returns:
        pd.DataFrame: DataFrame containing team information
    """
    return pd.read_csv(TEAMS_FILE)


@st.cache_data
def load_managers_data() -> tuple[pd.DataFrame, list[Manager]]:
    """Load manager data from CSV file and create Manager objects.

    Returns:
        tuple: A tuple containing:
            - pd.DataFrame: DataFrame with raw manager data
            - list[Manager]: List of Manager objects initialized with the data
    """
    df = pd.read_csv(MANAGERS_FILE)
    manager_objects = []
    for _, row in df.iterrows():
        # Convert staff_list string to actual list
        staff_list = eval(row['staff_list']) if isinstance(row['staff_list'], str) else row['staff_list']
        manager = Manager(
            id=row['manager_id'],
            first_name=row['manager_first_name'],
            last_name=row['manager_last_name'],
            staff_list=staff_list
        )
        manager_objects.append(manager)
    return df, manager_objects


def authenticate(username, password):
    """Authenticate users based on username and password.

    Args:
        username: The username to authenticate (format: "staffX" or "managerX")
        password: The password to check (currently accepts "password" for all users)

    Returns:
        dict or None: A dictionary containing user information if authenticated,
                      None if authentication fails
    """
    staff_df, staff_objects = load_staff_data()
    teams_df = load_teams_data()
    manager_df, manager_objects = load_managers_data()

    for _, team in teams_df.iterrows():
        if username == f"manager{team['manager_id']}" and password == "password":
            # Find the correct manager object
            manager = next((m for m in manager_objects if m.id == team['manager_id']), None)
            if manager:
                return {
                    "id": manager.id,
                    "object": manager,
                    "role": "manager",
                    "team_id": team['team_id']
                }

    try:
        staff_id = int(username.replace("staff", ""))
        staff = next((s for s in staff_objects if s.id == staff_id), None)
        if staff and password == "password":
            return {
                "id": staff_id,
                "object": staff,
                "role": "staff",
                "team_id": staff_df[staff_df['staff_id'] == staff_id]['team_id'].iloc[0]
            }
    except:
        pass

    return None


# Login page
def login_page():
    """Render the login page and handle authentication."""
    st.title("Employee Performance Tracker")
    st.subheader("Login")

    with st.form("login_form"):
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")
        submitted = st.form_submit_button("Login")

        if submitted:
            user = authenticate(username, password)
            if user:
                st.session_state.authenticated = True
                st.session_state.current_user = user
                st.rerun()
            else:
                st.error("Invalid username or password")


# Staff Dashboard using Staff class methods
def staff_dashboard():
    """Render the staff dashboard with performance metrics and call handling functionality."""
    user = st.session_state.current_user
    staff = user["object"]
    st.title(f"Staff Dashboard - {staff.first_name} {staff.last_name}")

    # Quick links section (RS4)
    with st.expander("Quick Links"):
        col1, col2, col3 = st.columns(3)
        with col1:
```

```python
        st.page_link("https://www.google.com/", label="Customer Service Guide")
    with col2:
        st.page_link("https://www.google.com/", label="Technical Support Manual")
    with col3:
        st.page_link("https://www.google.com/", label="FAQ Database")

# Workday controls (RS5)
col1, col2 = st.columns(2)
with col1:
    if not st.session_state.workday_started:
        if st.button("Start Workday"):
            staff.start_workday()
            st.session_state.workday_started = True
            st.session_state.workday_start_time = time.time()
            st.success("Workday started!")
    else:
        work_time = time.time() - st.session_state.workday_start_time
        st.metric("Time Elapsed", f"{int(work_time // 3600)}h {int((work_time % 3600) // 60)}m")
        if st.button("End Workday"):
            total_time = staff.end_workday()
            st.session_state.workday_started = False

            # Update CSV
            staff_df, _ = load_staff_data()
            staff_df.loc[staff_df['staff_id'] == staff.id, 'working_time_elapsed'] = total_time
            staff_df.to_csv(STAFF_FILE, index=False)

            st.success(f"Workday ended! Total time: {int(total_time // 3600)}h {int((total_time % 3600) // 60)}m")

# Call simulation (RS6, RS7)
with col2:
    if st.session_state.workday_started:
        if st.session_state.current_call is None:
            if st.button("Simulate Incoming Call"):
                call_id = int(time.time())
                new_call = Call(
                    id=call_id,
                    status="Incoming",
                    time_elapsed=0,
                    sat_score=0,
                    handler_id=0
                )
                new_call.start_time = time.time()
                st.session_state.current_call = new_call
                st.rerun()
        else:
            call = st.session_state.current_call
            call_duration = time.time() - call.time_elapsed
            st.warning(f"Call in progress: {int(call_duration)} seconds")

            if st.button("End Call"):
                # Random satisfaction score between 0.5 and 1.0
                sat_score = round(random.uniform(0.5, 1.0), 1)
                staff.end_call(st.session_state.current_call, sat_score)
                current_call = st.session_state.current_call

                # Get current date/time in the correct format
                now = datetime.datetime.now().strftime('%d/%m/%Y %H:%M')

                # Update calls CSV
                calls_df, _ = load_calls_data()
                new_call_data = {
                    'call_id': current_call.id,
                    'status': current_call.status,
                    'time_elapsed': int(current_call.time_elapsed),
                    'sat_score': float(current_call.sat_score),
                    'handler_id': staff.id,
                    'date': now,
                    'team_id': user['team_id']
                }
                calls_df.loc[len(calls_df)] = new_call_data
                calls_df.to_csv(CALLS_FILE, index=False)

                st.session_state.current_call = None
                st.rerun()

# Performance metrics (RS1, RS3)
st.subheader("Performance Metrics")

calls_df, _ = load_calls_data()
staff_calls = calls_df[calls_df['handler_id'] == staff.id]
team_calls = calls_df[calls_df['team_id'] == user['team_id']]

if not staff_calls.empty:
    # Success rate pie chart (RS1)
    successful = len(staff_calls[staff_calls['sat_score'] >= 0.8])
    unsuccessful = len(staff_calls) - successful

    fig, ax = plt.subplots(1, 2, figsize=(12, 4))

    # Pie chart
    ax[0].pie([successful, unsuccessful],
             labels=['Successful', 'Unsuccessful'],
             autopct='%1.1f%%',
             colors=['#4CAF50', '#F44336'])
    ax[0].set_title('Your Call Success Rate')

    # Satisfaction trend line chart (RS3)
    daily_avg = staff_calls.groupby(staff_calls['datetime'].dt.date)['sat_score'].mean().reset_index()
    team_daily_avg = team_calls.groupby(team_calls['datetime'].dt.date)['sat_score'].mean().reset_index()

    ax[1].plot(daily_avg['datetime'], daily_avg['sat_score'], label='Your Score')
    ax[1].plot(team_daily_avg['datetime'], team_daily_avg['sat_score'], label='Team Average')
    ax[1].set_title('Satisfaction Score Trend')
    ax[1].set_xlabel('Date')
    ax[1].set_ylabel('Average Satisfaction Score')
    ax[1].legend()
    ax[1].grid(True)

    st.pyplot(fig)

# Update call history table:
if not staff_calls.empty:
    recent_calls = staff_calls.sort_values('datetime', ascending=False).head(5)
    recent_calls['duration'] = recent_calls['time_elapsed'].apply(lambda x: f"{x // 60}m {x % 60}s")
    recent_calls['status'] = recent_calls['sat_score'].apply(
        lambda x: "вњ… Successful" if x >= 0.8 else "вќЊ Unsuccessful")

    st.dataframe(recent_calls[['datetime', 'duration', 'sat_score', 'status']].rename(columns={
        'datetime': 'Time',
        'duration': 'Duration',
        'sat_score': 'Satisfaction Score',
        'status': 'Status'
        }), hide_index=True)


# Manager Dashboard using Manager class methods
def manager_dashboard():
    """Render the manager dashboard with team overview and staff management functionality."""
    user = st.session_state.current_user
    manager = user["object"]
    st.title(f"Manager Dashboard - {manager.first_name} {manager.last_name}")

    staff_df, staff_objects = load_staff_data()
    calls_df, _ = load_calls_data()
    teams_df = load_teams_data()
    manager_df, manager_objects = load_managers_data()

    team_staff = staff_df[staff_df['team_id'] == user['team_id']]
    team_calls = calls_df[calls_df['team_id'] == user['team_id']]

    st.subheader("Team Overview")

    # Team success rate (RM2)
    if not team_calls.empty:

        col1, col2= st.columns([2, 2])  # Adjusted relative widths

        with col1:
            # Time period selector - now properly contained within col1
            time_period = st.selectbox(
                "Time Period",
                ["All Time", "Today", "Last 7 Days", "Last 30 Days", "Last 90 Days"],
                key="time_period_selector"  # Added key to avoid duplicate widget issues
            )

            # Calculate filtered calls based on selected time period
            if time_period == "Today":
                cutoff_date = pd.to_datetime('today')
                filtered_calls = team_calls[team_calls['datetime'] >= cutoff_date]
            elif time_period == "Last 7 Days":
                cutoff_date = pd.to_datetime('today') - pd.Timedelta(days=7)
                filtered_calls = team_calls[team_calls['datetime'] >= cutoff_date]
            elif time_period == "Last 30 Days":
                cutoff_date = pd.to_datetime('today') - pd.Timedelta(days=30)
                filtered_calls = team_calls[team_calls['datetime'] >= cutoff_date]
            elif time_period == "Last 90 Days":
                cutoff_date = pd.to_datetime('today') - pd.Timedelta(days=90)
                filtered_calls = team_calls[team_calls['datetime'] >= cutoff_date]
            else:  # "All Time"
                filtered_calls = team_calls.copy()

            # Now create the pie chart with filtered data
            if not filtered_calls.empty:
                filtered_successful = len(filtered_calls[filtered_calls['sat_score'] >= 0.8])
                filtered_unsuccessful = len(filtered_calls) - filtered_successful

                fig, ax = plt.subplots()
                ax.pie([filtered_successful, filtered_unsuccessful],
                       labels=['Successful', 'Unsuccessful'],
                       autopct=lambda p: f'{p:.1f}%' if p > 0 else '',
                       colors=['#4CAF50', '#F44336'],
                       wedgeprops={'linewidth': 1, 'edgecolor': 'white'})

                ax.set_title(f'Team Success Rate\n({time_period}: {len(filtered_calls)} calls)')

                # centre_circle = plt.Circle((0, 0), 0.7, color='white', fc='white', linewidth=0)
                # ax.add_artist(centre_circle)

                st.pyplot(fig)

                success_rate = filtered_successful / len(filtered_calls) * 100 if len(filtered_calls) > 0 else 0
                st.metric("Success Rate",
                          f"{success_rate:.1f}%",
                          f"{filtered_successful} of {len(filtered_calls)} calls")
            else:
                st.warning(f"No call data available for {time_period.lower()}")

        with col2:
            # Team comparison chart
            st.write("**Team Comparison**")
            if len(teams_df) > 1:
                team_success = []
                for _, team in teams_df.iterrows():
                    team_calls = calls_df[calls_df['team_id'] == team['team_id']]
                    if len(team_calls) > 0:
                        success_rate = len(team_calls[team_calls['sat_score'] >= 0.8]) / len(team_calls)
                        team_success.append({
                            'Team': '.'.join(re.findall(r'\b(\w)\w*\b', team['team_name'])) + '.',
                            'Success Rate': success_rate
                        })

                    if not team_staff.empty and not team_calls.empty:
                        performance_data = []
                        for _, staff in team_staff.iterrows():
                            staff_calls = calls_df[calls_df['handler_id'] == staff['staff_id']]
                            if len(staff_calls) > 0:
                                success_rate = len(staff_calls[staff_calls['sat_score'] >= 0.8]) / len(staff_calls)
                                performance_data.append({
                                    'Staff ID': staff['staff_id'],
                                    'Name': f"{staff['first_name']} {staff['last_name']}",
                                    'Calls Taken': staff['calls_taken'],
                                    'Success Rate': success_rate,
                                    'Avg Satisfaction': staff['avg_sat_score']
                                })

                if team_success:
                    comparison_df = pd.DataFrame(team_success)
                    fig, ax = plt.subplots()
                    sns.barplot(data=comparison_df, x='Team', y='Success Rate', ax=ax)
                    ax.set_title('Team Comparison')
                    ax.set_ylim(0, 1)
                    st.pyplot(fig)

    # Top/worst performers (RM4)
    st.subheader("Performance Highlights")


    if performance_data:
        perf_df = pd.DataFrame(performance_data)
        perf_df = perf_df.sort_values(['Success Rate', 'Calls Taken', 'Avg Satisfaction'], ascending=False)

        col1, col2 = st.columns(2)

        with col1:
            st.write("Top Performers")
        # Apply green background to top performers
        st.dataframe(
            perf_df.head(3).style.map(
                lambda x: 'background-color: #95b36b',  # Mint green pastel
                subset=pd.IndexSlice[:, :]  # Apply to all cells
            ),
            hide_index=True
        )

    with col2:
        st.write("Need Improvement")
        # Apply red background to bottom performers and sort by Success Rate
        st.dataframe(
            perf_df.tail(3)[::-1].style.map(
                lambda x: 'background-color: #fab6b6',  # Pastel red
                subset=pd.IndexSlice[:, :]  # Apply to all cells
            ),
            hide_index=True
        )
else:
    st.info("No performance data available yet")


# Staff management section
st.subheader("Staff Management")

tab1, tab2, tab3, tab4 = st.tabs(["View Staff", "Add Staff", "Edit Staff", "Remove Staff"])

with tab1:
    # View staff details (RM1, RM9)
    selected_staff = st.selectbox(
        "Select Staff Member",
        team_staff.apply(lambda x: f"{x['staff_id']} - {x['first_name']} {x['last_name']}", axis=1)
    )

    if selected_staff:
        staff_id = int(selected_staff.split(" - ")[0])
        staff_details = team_staff[team_staff['staff_id'] == staff_id].iloc[0]

        st.write(f"**Name:** {staff_details['first_name']} {staff_details['last_name']}")
        st.write(f"**Calls Taken:** {staff_details['calls_taken']}")
        st.write(f"**Successful Calls:** {staff_details['successful_calls']}")
        st.write(f"**Failed Calls:** {staff_details['failed_calls']}")
        st.write(f"**Success Rate:** {staff_details['successful_calls'] / staff_details['calls_taken'] * 100:.1f}%"
                 if staff_details['calls_taken'] > 0 else "**Success Rate:** N/A")
        st.write(f"**Avg Satisfaction:** {staff_details['avg_sat_score']}")
        st.write(f"**Status:** {staff_details['status']}")

        # Staff call history
        staff_calls = calls_df[calls_df['handler_id'] == staff_id]
        if not staff_calls.empty:
            st.write("**Recent Calls:**")
            recent_calls = staff_calls.sort_values('datetime', ascending=False).head(5)
            recent_calls['duration'] = recent_calls['time_elapsed'].apply(lambda x: f"{x // 60}m {x % 60}s")
            recent_calls['status'] = recent_calls['sat_score'].apply(
                lambda x: "вњ… Successful" if x >= 0.8 else "вќЊ Unsuccessful")

            st.dataframe(recent_calls[['datetime', 'duration', 'sat_score', 'status']].rename(columns={
                'datetime': 'Time',
                'duration': 'Duration',
                'sat_score': 'Satisfaction Score',
                'status': 'Status'
            }), hide_index=True)

with tab2:
    # Add staff (RM6)
    with st.form("add_staff_form"):
        st.write("Add New Staff Member")
        staff_id = st.number_input("Staff ID", min_value=1, step=1)
        first_name = st.text_input("First Name")
        last_name = st.text_input("Last Name")
        target_calls = st.number_input("Target Successful Calls", min_value=1, value=10)

        submitted = st.form_submit_button("Add Staff")
        if submitted:
            if staff_id in staff_df['staff_id'].values:
                st.error("Staff ID already exists")
            else:
                # Create new Staff object
                new_staff = Staff(
                    id=staff_id,
                    first_name=first_name,
                    last_name=last_name,
                    manager_id=manager.id,
                    calls_taken=0,
                    successful_calls=0,
                    failed_calls=0,
                    target_successful_calls=target_calls,
                    working_time_elapsed=0,
                    avg_sat_score=0,
                    status='Free'
                )

                # Add to CSV
                new_row = {
                    'staff_id': staff_id,
                    'first_name': first_name,
                    'last_name': last_name,
                    'manager_id': manager.id,
                    'calls_taken': 0,
                    'successful_calls': 0,
                    'failed_calls': 0,
                    'target_successful_calls': target_calls,
                    'working_time_elapsed': 0,
                    'avg_sat_score': 0,
                    'status': 'Free',
                    'team_id': user['team_id']
                }
                staff_df.loc[len(staff_df)] = new_row
                staff_df.to_csv(STAFF_FILE, index=False)

                # Update manager's staff list
                manager.staff_list.append(staff_id)

                # Update manager CSV
                manager_df.loc[manager_df['manager_id'] == manager.id, 'staff_list'] = str(manager.staff_list)
                manager_df.to_csv(MANAGERS_FILE, index=False)

                st.success("Staff member added successfully!")
                st.rerun()

with tab3:
    # Edit staff (RM8)
```

```python
        staff_to_edit = st.selectbox(
            "Select Staff Member to Edit",
            team_staff.apply(lambda x: f"{x['staff_id']} - {x['first_name']} {x['last_name']}", axis=1),
            key="edit_select"
        )

        if staff_to_edit:
            staff_id = int(staff_to_edit.split(" - ")[0])
            staff_details = team_staff[team_staff['staff_id'] == staff_id].iloc[0]

            with st.form("edit_staff_form"):
                new_first = st.text_input("First Name", value=staff_details['first_name'])
                new_last = st.text_input("Last Name", value=staff_details['last_name'])
                new_target = st.number_input(
                    "Target Successful Calls",
                    min_value=1,
                    value=int(staff_details['target_successful_calls']))
                new_status = st.selectbox(
                    "Status",
                    ['Free', 'On Call', 'Lunch', 'Out of Office'],
                    index=['Free', 'On Call', 'Lunch', 'Out of Office'].index(staff_details['status']))

                submitted = st.form_submit_button("Update Staff")
                if submitted:
                    staff_df.loc[staff_df['staff_id'] == staff_id, 'first_name'] = new_first
                    staff_df.loc[staff_df['staff_id'] == staff_id, 'last_name'] = new_last
                    staff_df.loc[staff_df['staff_id'] == staff_id, 'target_successful_calls'] = new_target
                    staff_df.loc[staff_df['staff_id'] == staff_id, 'status'] = new_status
                    staff_df.to_csv(STAFF_FILE, index=False)
                    st.success("Staff details updated successfully!")
                    st.rerun()

    with tab4:
        # Remove staff (RM7)
        staff_to_remove = st.selectbox(
            "Select Staff Member to Remove",
            team_staff.apply(lambda x: f"{x['staff_id']} - {x['first_name']} {x['last_name']}", axis=1),
            key="remove_select"
        )

        if staff_to_remove:
            staff_id = int(staff_to_remove.split(" - ")[0])

            if st.button("Remove Staff"):
                if staff_id == manager.id:
                    st.error("You cannot remove yourself")
                else:
                    # Remove from CSV
                    staff_df = staff_df[staff_df['staff_id'] != staff_id]
                    staff_df.to_csv(STAFF_FILE, index=False)

                    # Update manager's staff list
                    if staff_id in manager.staff_list:
                        manager.staff_list.remove(staff_id)

                    st.success("Staff member removed successfully!")
                    st.rerun()


# Main app
def main():
    initialize_files()

    if not st.session_state.authenticated:
        login_page()
    else:
        if st.session_state.current_user['role'] == "manager":
            manager_dashboard()
        else:
            staff_dashboard()

        # Logout button
        st.sidebar.title("Account")
        if st.sidebar.button("Logout"):
            st.session_state.authenticated = False
            st.session_state.current_user = None
            st.session_state.current_call = None
            st.session_state.workday_started = False
            st.rerun()


if __name__ == "__main__":
    main()
```