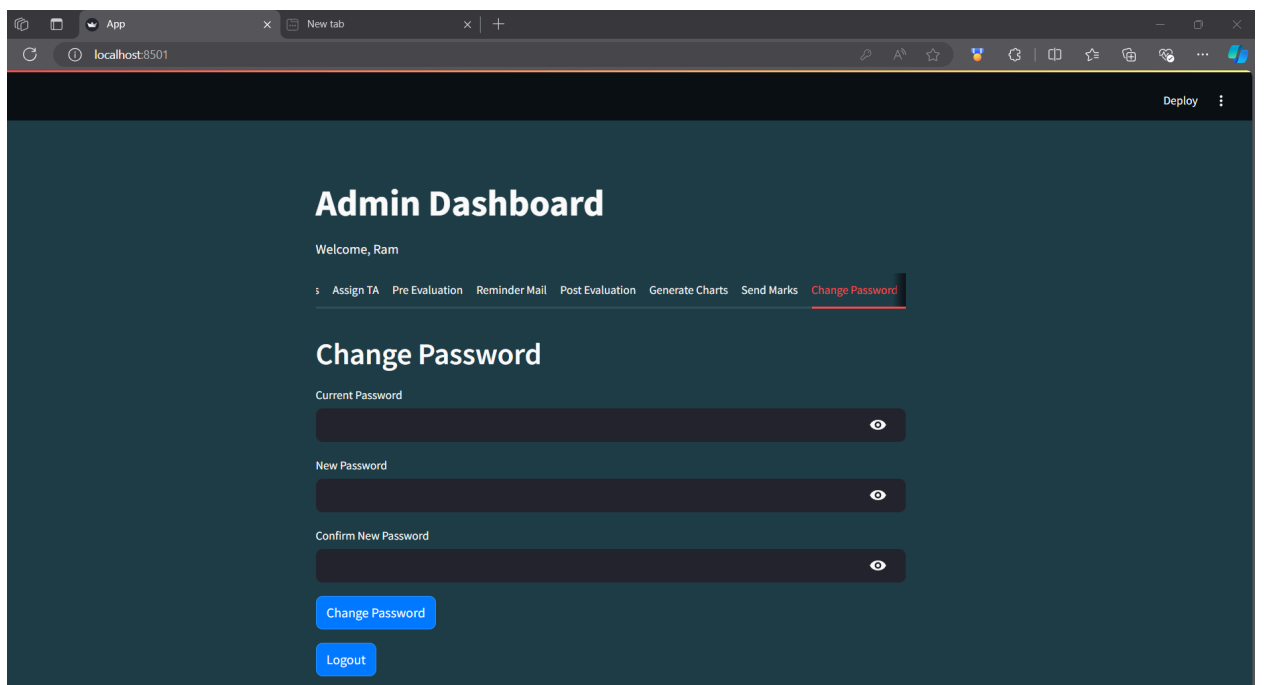
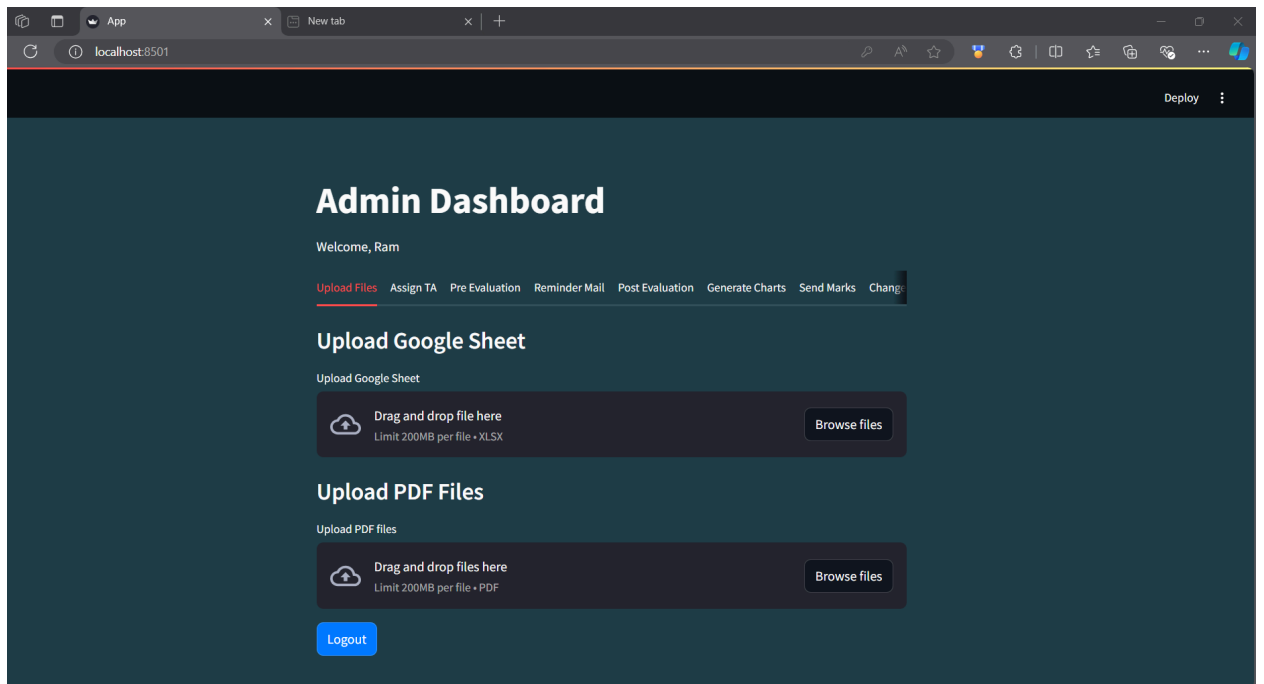
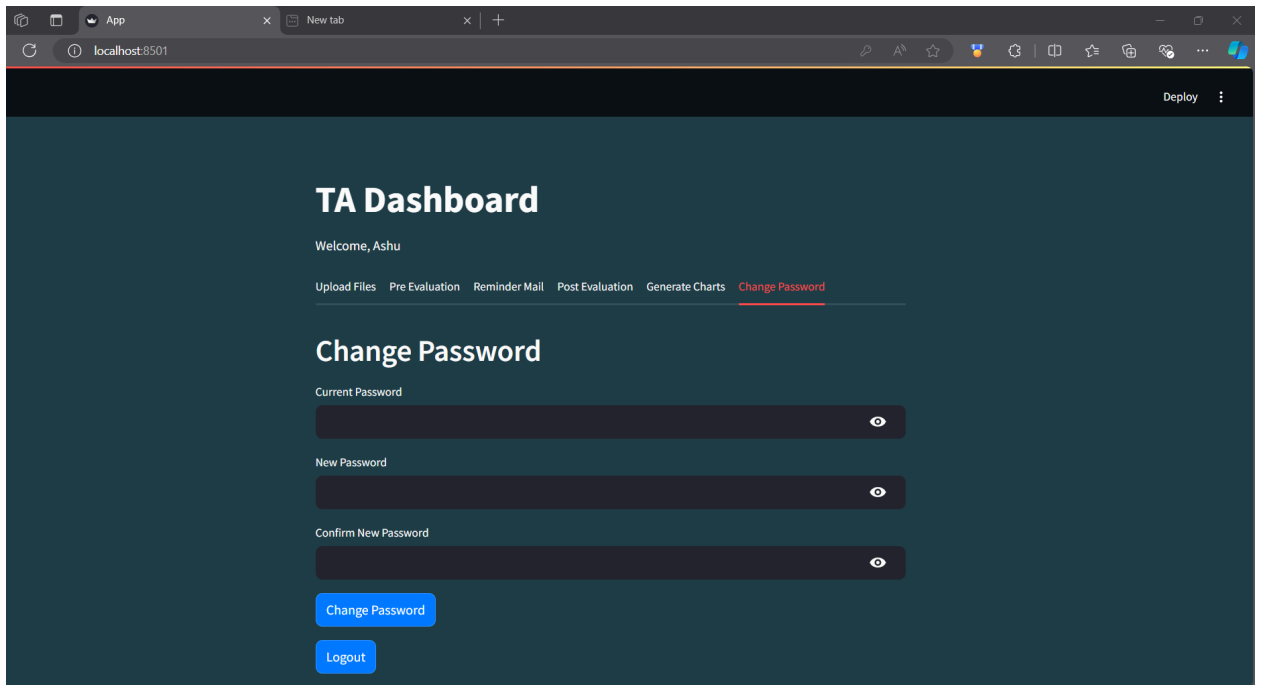
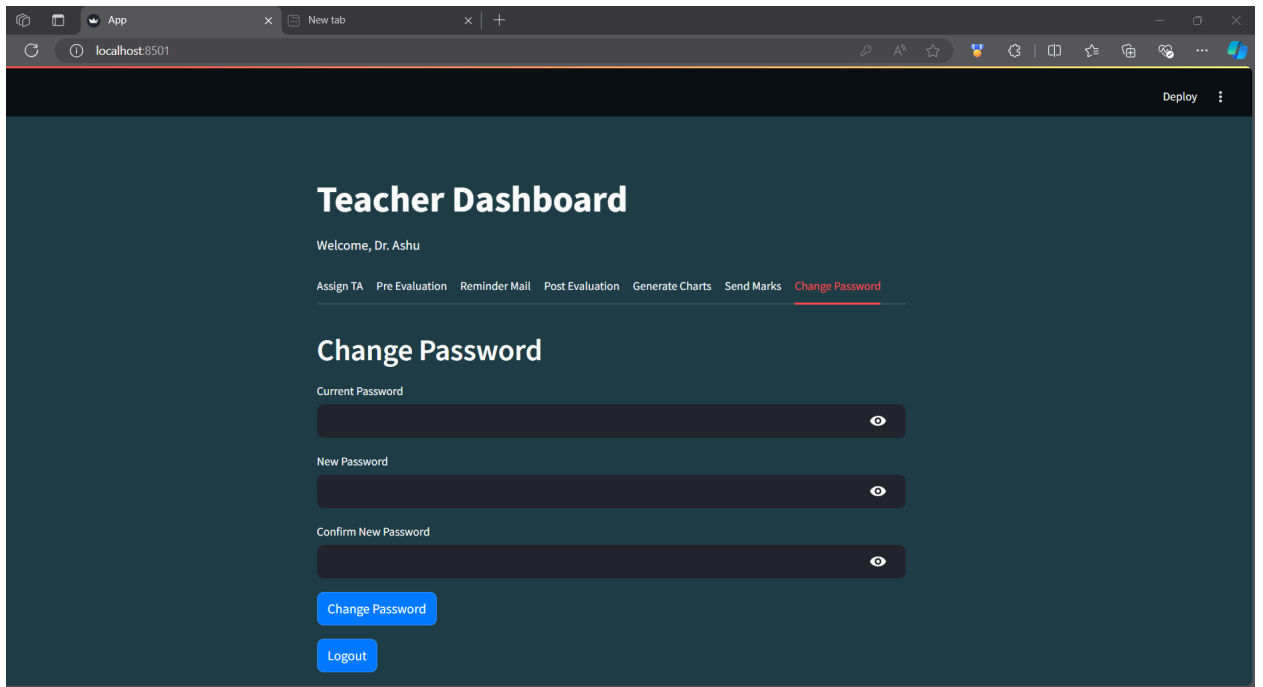


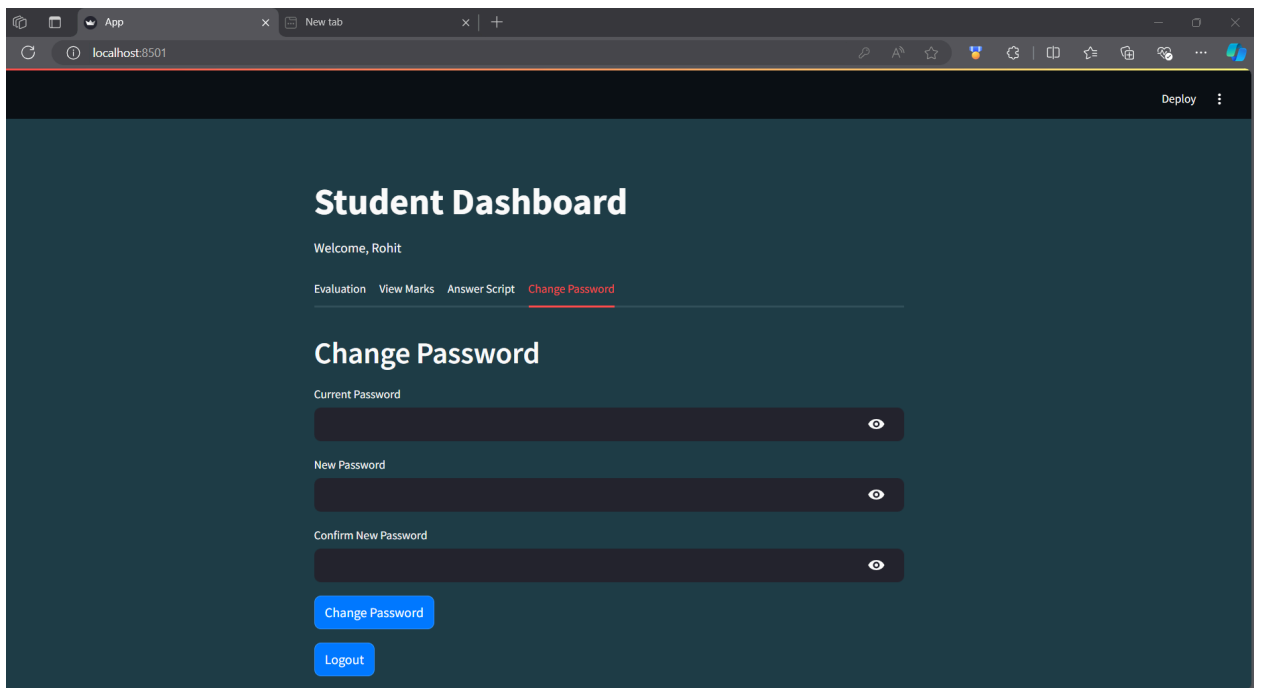
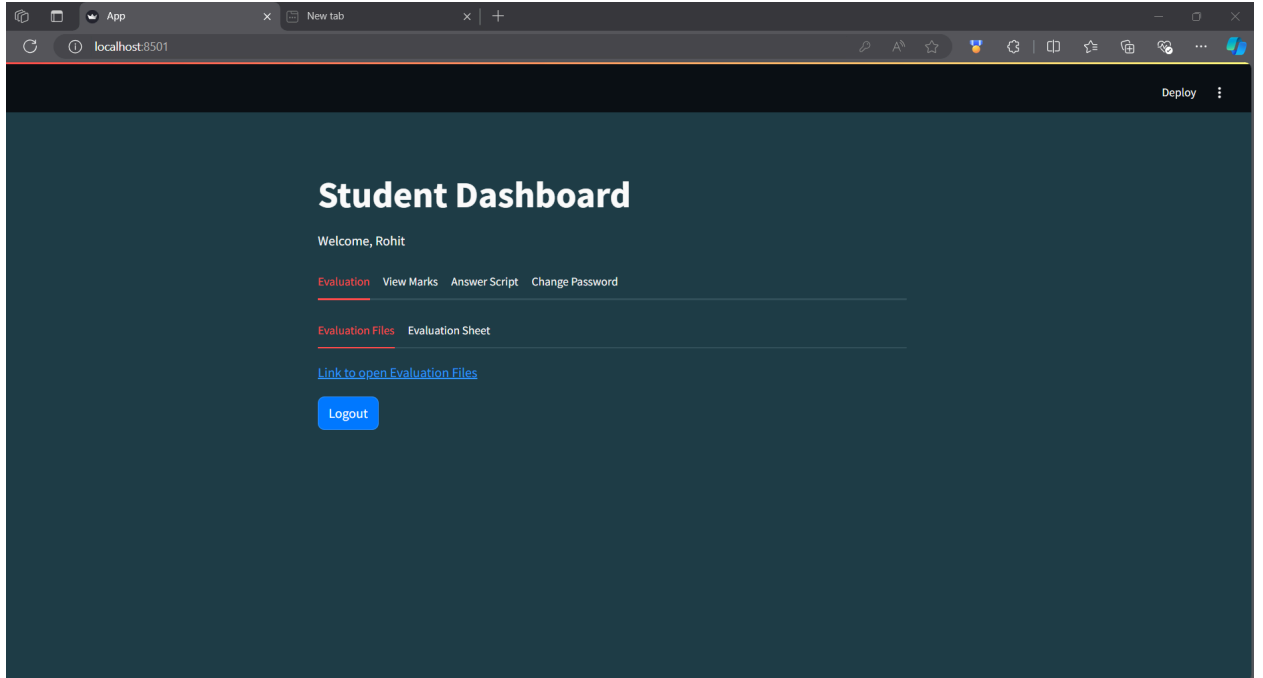
Peer Evaluation System UI/UX

Screenshots of the UI/UX design: -

- The changes from the today's code are reflected below: -







Code: -

1. Python: -

```
import io
import re
import time
import bcrypt
import gspread
import requests
import streamlit as st
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseUpload
from googleapiclient.http import MediaIoBaseDownload
from oauth2client.service_account import ServiceAccountCredentials

# Google Sheets and Google Drive setup
SCOPE = [
    "https://spreadsheets.google.com/feeds",
    "https://www.googleapis.com/auth/drive"
]
CREDENTIALS_FILE = "D:/ROHIT IIT/Peer
Evaluation/peer-evaluation-sem1-e2fcf8b5fc27.json"
SHEET_NAME = "UserRoles"

# Initialize connection to Google Sheets
def connect_to_google_sheets():
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
    client = gspread.authorize(creds)
    sheet = client.open(SHEET_NAME).sheet1
    return sheet

# Google Drive authentication
def authenticate_drive():
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
    service = build('drive', 'v3', credentials=creds)
```

```
return service
```

```
# Fetch users from Google Sheets
```

```
def get_users_from_sheets():  
    sheet = connect_to_google_sheets()  
    records = sheet.get_all_records()  
    return records
```

```
# Add new user to Google Sheets with role auto-assignment
```

```
def register_user(username, password):  
    sheet = connect_to_google_sheets()
```

```
    # Check if the email contains numeric values (assumed to be student)
```

```
    if re.search(r'\d', username):
```

```
        role = "Student"
```

```
    else:
```

```
        role = "Teacher"
```

```
    # Hash the password before saving
```

```
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
```

```
    new_user = [username, hashed_password.decode('utf-8'), role]
```

```
    #new_user = [username, password, role]
```

```
    sheet.append_row(new_user)
```

```
    return role
```

```
# Update role from Student to TA (only for Teachers)
```

```
def update_role_to_ta(username):
```

```
    sheet = connect_to_google_sheets()
```

```
    records = sheet.get_all_records()
```

```
    for i, user in enumerate(records, start=2): # start=2 to account for 1-based index in
```

```
Google Sheets
```

```
        if user['username'] == username and user['role'] == 'Student':
```

```
            sheet.update_cell(i, 3, 'TA') # Assuming role is in column 3
```

```
            return True
```

```
    return False
```

```

# Verify user credentials
def login(username, password, users):
    for user in users:
        if user['username'] == username:
            # Check if the password matches the stored hash
            if bcrypt.checkpw(password.encode('utf-8'), user['password'].encode('utf-8')):
                st.session_state["login_status"] = True
                st.session_state["role"] = user["role"]
                st.session_state["username"] = username
                st.session_state["page"] = "dashboard"
                st.session_state["message"] = None
                return
            else:
                st.session_state["message"] = "Incorrect password"
                return
    st.session_state["message"] = "Incorrect username or password"

```

```

# Logout function
def logout():
    st.session_state["login_status"] = False
    st.session_state["role"] = None
    st.session_state["username"] = None
    st.session_state["page"] = "login"
    st.session_state["message"] = "Logged out successfully"

```

```

# Function to change password
def change_password(username, current_password, new_password):
    sheet = connect_to_google_sheets()
    records = sheet.get_all_records()

    # Find the user in the records
    for i, user in enumerate(records, start=2): # start=2 for 1-based indexing (Google Sheets)
        if user['username'] == username:
            # Check if the current password matches the stored hash
            if bcrypt.checkpw(current_password.encode('utf-8'),
user['password'].encode('utf-8')):

```

```

        # Hash the new password
        hashed_new_password = bcrypt.hashpw(new_password.encode('utf-8'),
bcrypt.gensalt()).decode('utf-8')

```

```

        # Update the password in the sheet
        sheet.update_cell(i, 2, hashed_new_password)
        return True # Password changed successfully
    else:
        return False # Current password is incorrect
return False # User not found

```

```

def change_password_dashboard():

```

```

    st.header("Change Password")

```

```

    current_password = st.text_input("Current Password", type="password")
    new_password = st.text_input("New Password", type="password")
    confirm_password = st.text_input("Confirm New Password", type="password")

```

```

    if st.button("Change Password"):

```

```

        if new_password != confirm_password:

```

```

            st.error("New password and confirm password do not match!")

```

```

        else:

```

```

            success = change_password(st.session_state['username'], current_password,
new_password)

```

```

            if success:

```

```

                st.success("Password changed successfully!")

```

```

                time.sleep(2)

```

```

                logout()

```

```

                st.rerun()

```

```

            else:

```

```

                st.error("Failed to change password. Incorrect current password.")

```

```

def trigger_google_apps_script(function_name):

```

```

    web_app_url =

```

```

    "https://script.google.com/macros/s/AKfycbwIbIl062YhNYcbIqmP9obfLBKgoeIdTdRD
Q_BOB4rF1S6JhTxvVFH8MhW2x84bgyAVag/exec" # Replace with your web app
URL

```

```

url = f'{web_app_url}?action={function_name}' # Append the function name as the
'action' parameter
try:
    response = requests.get(url)
    if response.status_code == 200:
        st.success(f'{function_name} executed successfully!')
    else:
        st.error(f'Failed to execute {function_name}. Status code:
{response.status_code}')
except Exception as e:
    st.error(f'An error occurred: {str(e)}')

```

```

# Function to check if a file already exists in Google Drive folder
def file_exists(drive_service, folder_id, file_name):
    query = f'{folder_id}' in parents and name='{file_name}'
    results = drive_service.files().list(q=query, spaces='drive', fields='files(id,
name)').execute()
    files = results.get('files', [])
    return any(file['name'] == file_name for file in files)

```

```

# Function to upload PDF files to Google Drive
def upload_pdfs(uploaded_files, folder_id):
    drive_service = authenticate_drive()
    count = 0

    for uploaded_file in uploaded_files:
        if file_exists(drive_service, folder_id, uploaded_file.name):
            #st.warning(f'PDF file '{uploaded_file.name}' already exists in the folder.')
            continue

        file_metadata = {
            'name': uploaded_file.name,
            'parents': [folder_id]
        }
        media = MediaIoBaseUpload(uploaded_file, mimetype='application/pdf')
        drive_service.files().create(body=file_metadata, media_body=media,
fields='id').execute()
        count = count + 1

```



```
    #st.session_state["success_message"] = f"Uploaded PDF file '{uploaded_file.name}'  
    to Google Drive"
```

```
    st.success(f" The {count} files are uploaded to the Google Drive.")
```

```
# Function to upload Google Sheets files to Google Drive
```

```
def upload_sheets(uploaded_files, folder_id):
```

```
    drive_service = authenticate_drive()
```

```
    for uploaded_file in uploaded_files:
```

```
        if file_exists(drive_service, folder_id, uploaded_file.name):
```

```
            #st.warning(f"Google Sheet file '{uploaded_file.name}' already exists in the  
            folder.")
```

```
            continue
```

```
        file_metadata = {
```

```
            'name': uploaded_file.name,
```

```
            'parents': [folder_id],
```

```
            'mimeType': 'application/vnd.google-apps.spreadsheet'
```

```
        }
```

```
        media = MediaIoBaseUpload(uploaded_file, mimetype='application/vnd.ms-excel')
```

```
        drive_service.files().create(body=file_metadata, media_body=media,  
        fields='id').execute()
```

```
    st.success("The Excel sheet has been uploaded to the Google Drive.")
```

```
# Helper function to connect to a specific Google Sheet
```

```
def connect_to_google_sheets_with_name(sheet_name):
```

```
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,  
    SCOPE)
```

```
    client = gspread.authorize(creds)
```

```
    sheet = client.open(sheet_name)
```

```
    return sheet
```

```
def get_student_details(username):
```

```
    # Connect to the specific Google Sheet containing marks
```

```
    sheet_name = "UI/UX Copy of Peer Evaluation2"
```

```

    sheet = connect_to_google_sheets_with_name(sheet_name) # Modify to accept a sheet
name
    peer_eval_sheet = sheet.worksheet('PeerEval') # Open the "PeerEval" sheet

    # Fetch all the data from the "PeerEval" sheet
    records = peer_eval_sheet.get_all_records()

    # Find marks for the current user
    for record in records:
        if record['EMail ID'] == username: # Ensure this matches your column name
            return record['Average Marks'], record['Unique ID'], record['Assigned Folder
Link'], record['Spreadsheet Link'] # Returning the Average Mark's and Unique id

    return None, None, None, None # If no details found for the user

# Fetch the student's PDF from Google Drive using unique ID
def get_student_pdf(unique_id):
    drive_service = authenticate_drive()
    folder_id = "1fT-incILQut85BGEQrjMSWbVRcTsdWfQ"
    query = f"'{folder_id}' in parents and name contains '{unique_id}'"
    results = drive_service.files().list(q=query, fields="files(id, name)").execute()
    files = results.get('files', [])

    if files:
        file_id = files[0]['id']
        file_name = files[0]['name']

        # Download the PDF
        request = drive_service.files().get_media(fileId=file_id)
        fh = io.BytesIO()
        downloader = MediaIoBaseDownload(fh, request)
        done = False
        while not done:
            status, done = downloader.next_chunk()

        fh.seek(0)
        return fh, file_name

    return None, None

```

```

def admin_dashboard():
    st.title("Admin Dashboard")
    st.write(f'Welcome, {st.session_state['username'].split('.')[0].capitalize()}')

    # Create tabs for each action
    tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6 = st.tabs(
        ["Upload Files", "Assign TA", "Pre Evaluation", "Reminder Mail", "Post Evaluation",
        "Generate Charts",
        "Send Marks", "Change Password"])

    # Tab for File upload option
    with tab:
        # Folder ID for the Google Drive folder where the files will be saved
        folder_id = "1fT-incilQut85BGEQrjMSWbVRcTsdWfQ" # Replace this with your
        folder ID

        # Allow file upload for multiple Google Sheets
        st.subheader("Upload Google Sheet")
        sheet_files = st.file_uploader("Upload Google Sheet", type=["xlsx"],
        accept_multiple_files=False,
        key="sheet_uploader")

        if sheet_files:
            upload_sheets(sheet_files, folder_id)

        # Allow file upload for multiple PDFs
        st.subheader("Upload PDF Files")
        pdf_files = st.file_uploader("Upload PDF files", type=["pdf"],
        accept_multiple_files=True, key="pdf_uploader")

        if pdf_files:
            upload_pdfs(pdf_files, folder_id)

    # Tab for TA update
    with tab0:
        student_username = st.text_input("Enter Student's Username")
        if st.button("Update Role to TA"):

```

```
        if update_role_to_ta(student_username):
            st.success(f'{student_username.split('.')[0].capitalize()}'s role updated to TA.")
        else:
            st.error("Failed to update the role. Check if the username exists and belongs to a student.")
```

```
# Tab for Pre Evaluation
```

```
with tab1:
```

```
    if st.button("Pre Evaluation"):
        trigger_google_apps_script("PreEval")
```

```
# Tab for Checking Pending Evaluations
```

```
with tab2:
```

```
    if st.button("Reminder Mail"):
        trigger_google_apps_script("CheckEval")
```

```
# Tab for Post Evaluation
```

```
with tab3:
```

```
    if st.button("Post Evaluation"):
        trigger_google_apps_script("PostEval")
```

```
# Tab for Generating Charts
```

```
with tab4:
```

```
    if st.button("Generate Charts"):
        trigger_google_apps_script("GenChart")
```

```
# Tab for Sending Marks
```

```
with tab5:
```

```
    if st.button("Send Marks"):
        trigger_google_apps_script("SendMail")
```

```
with tab6:
```

```
    change_password_dashboard()
```

```
def teacher_dashboard():
```

```
    st.title("Teacher Dashboard")
```

```
    #st.write(f'Welcome, {st.session_state['username']}')"
```

```
    var_user = st.session_state['username'].split('@')[0]
```

```
    if '.' in var_user:
```

```

        st.write(f"Welcome, Dr. {var_user.split('.')[0].capitalize()}")
    else:
        st.write(f"Welcome, Dr. {var_user.capitalize()}")

# Create tabs for each action
tab0, tab1, tab2, tab3, tab4, tab5, tab6 = st.tabs(["Assign TA", "Pre Evaluation",
"Reminder Mail", "Post Evaluation", "Generate Charts", "Send Marks", "Change
Password"])

# Tab for TA update
with tab0:
    student_username = st.text_input("Enter Student's Username")
    if st.button("Update Role to TA"):
        if update_role_to_ta(student_username):
            st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to TA.")
        else:
            st.error("Failed to update the role. Check if the username exists and belongs to a
student.")

# Tab for Pre Evaluation
with tab1:
    if st.button("Pre Evaluation"):
        trigger_google_apps_script("PreEval")

# Tab for Checking Pending Evaluations
with tab2:
    if st.button("Reminder Mail"):
        trigger_google_apps_script("CheckEval")

# Tab for Post Evaluation
with tab3:
    if st.button("Post Evaluation"):
        trigger_google_apps_script("PostEval")

# Tab for Generating Charts
with tab4:
    if st.button("Generate Charts"):
        trigger_google_apps_script("GenChart")

```

```

# Tab for Sending Marks
with tab5:
    if st.button("Send Marks"):
        trigger_google_apps_script("SendMail")

with tab6:
    change_password_dashboard()


# Role-based content: Teacher Dashboard with multiple file uploads
def ta_dashboard():
    st.title("TA Dashboard")
    st.write(f'Welcome, {st.session_state['username'].split('.')[0].capitalize()}')
    #st.write(f'Welcome, {st.session_state['username']}')

    # Create tabs for each action
    tab, tab0, tab1, tab2, tab3, tab4 = st.tabs(
        ["Upload Files", "Pre Evaluation", "Reminder Mail", "Post Evaluation", "Generate
Charts", "Change Password"])

    # Tab for File upload option
    with tab:
        # Folder ID for the Google Drive folder where the files will be saved
        folder_id = "1fT-incilQut85BGEQrjMSWbVRcTsdWfQ" # Replace this with your
folder ID

        # Allow file upload for multiple Google Sheets
        st.subheader("Upload Google Sheet")
        sheet_files = st.file_uploader("Upload Google Sheet", type=["xlsx"],
accept_multiple_files=False,
key="sheet_uploader")

        if sheet_files:
            upload_sheets(sheet_files, folder_id)

        # Allow file upload for multiple PDFs
        st.subheader("Upload PDF Files")

```

```

pdf_files = st.file_uploader("Upload PDF files", type=["pdf"],
accept_multiple_files=True, key="pdf_uploader")

if pdf_files:
    upload_pdfs(pdf_files, folder_id)

# Tab for Pre Evaluation
with tab0:
    if st.button("Pre Evaluation"):
        trigger_google_apps_script("PreEval")

# Tab for Checking Pending Evaluations
with tab1:
    if st.button("Reminder Mail"):
        trigger_google_apps_script("CheckEval")

# Tab for Post Evaluation
with tab2:
    if st.button("Post Evaluation"):
        trigger_google_apps_script("PostEval")

# Tab for Generating Charts
with tab3:
    if st.button("Generate Charts"):
        trigger_google_apps_script("GenChart")

with tab4:
    change_password_dashboard()

def student_dashboard():
    st.title("Student Dashboard")
    st.write(f"Welcome, {st.session_state['username'].split('.')[0].capitalize()}")
    #st.write(f"Welcome, {st.session_state['username']}")

# Creating tabs

```

```
tab1, tab2, tab3, tab4 = st.tabs(["Evaluation", "View Marks", "Answer Script", "Change Password"])
```

```
# with tab0:
# change_password_dashboard()
# if st.session_state["username"]:
#     # Fetch marks, unique ID, and spreadsheet link using the session's username
#     marks, unique_id, folder_link, sheet_link =
get_student_details(st.session_state["username"])
# else:
#     st.error("Username is Incorrect!")
```

```
# Tab for opening the peer evaluation spreadsheet
with tab1:
    if st.session_state["username"]:
        # Fetch marks, unique ID, and spreadsheet link using the session's username
        marks, unique_id, folder_link, sheet_link =
get_student_details(st.session_state["username"])
    else:
        st.error("Username is Incorrect!")
```

```
t1, t2 = st.tabs(["Evaluation Files", "Evaluation Sheet"])
with t1:
    if folder_link:
        st.markdown(f"[Link to open Evaluation Files]({folder_link})",
unsafe_allow_html=True)
    else:
        st.error("Folder link not found,")
with t2:
    if sheet_link:
        st.markdown(f"[Link to open Evaluation Sheet]({sheet_link})",
unsafe_allow_html=True)
    else:
        st.error("Spreadsheet link not found.")
```

```
# Tab for viewing marks
with tab2:
    if st.button("See Marks"):
        if marks and unique_id:
            st.write(f"Your evaluation marks are = {marks}")
```



```

        else:
            st.error("No marks are available.")

# Tab for downloading PDF
with tab3:
    pdf_file, file_name = get_student_pdf(unique_id)
    if pdf_file:
        st.download_button(
            label="Download your Evaluation PDF",
            data=pdf_file,
            file_name=file_name,
            mime='application/pdf'
        )
    else:
        st.error("PDF not found.")

with tab4:
    change_password_dashboard()

```

```

# Main Streamlit app
def main():
    # Initialize session state variables if not present
    if "login_status" not in st.session_state:
        st.session_state["login_status"] = False
    if "role" not in st.session_state:
        st.session_state["role"] = None
    if "username" not in st.session_state:
        st.session_state["username"] = None
    if "page" not in st.session_state:
        st.session_state["page"] = "login"
    if "message" not in st.session_state:
        st.session_state["message"] = None
    if "success_message" not in st.session_state:
        st.session_state["success_message"] = None

# Set background color and input field styling using HTML

```

```

st.markdown(
    """
    <style>
    .stApp {
        background-color: #1f3f49; /* Light blue background */
    }
    .stTextInput>div>input, .stPasswordInput>div>input {
        background-color: white; /* White background for text and password inputs */
        color: black; /* Text color for input fields */
    }
    .stButton>button {
        background-color: #007bff; /* Optional: Style buttons with a color */
        color: white;
    }
    </style>
    """,
    unsafe_allow_html=True
)

```

```

# Page routing based on session state
if st.session_state["page"] == "login":
    st.title("Peer Evaluation System")

```

```

# Tabs for Login and Registration
tab1, tab2 = st.tabs(["Login", "Register"])

```

```

with tab1:
    st.header("Login")

```

```

with st.form(key='login_form'):
    username = st.text_input("Email ID")
    password = st.text_input("Password", type="password")
    submit_button = st.form_submit_button("Login")

```

```

if submit_button:
    users = get_users_from_sheets()
    login(username, password, users)
    if st.session_state["login_status"]:
        st.rerun()

```

```

with tab2:
    st.header("Register")

    with st.form(key='register_form'):
        reg_username = st.text_input("Email ID", key='reg_username')
        reg_password = st.text_input("Password", type="password",
key='reg_password')
        register_button = st.form_submit_button("Register")

    if register_button:
        if not reg_username.endswith("@iitrpr.ac.in"):
            st.error("Email ID must end with @iitrpr.ac.in")
        else:
            users = get_users_from_sheets()
            if any(user['username'] == reg_username for user in users):
                st.error("Username already exists")
            else:
                role = register_user(reg_username, reg_password)
                st.success(f"User registered successfully with role: {role}")
                time.sleep(2)
                # Redirect to the login page
                st.session_state["page"] = "login"
                st.rerun()

elif st.session_state["page"] == "dashboard":
    if st.session_state["role"] == "Admin":
        admin_dashboard()
    elif st.session_state["role"] == "Teacher":
        teacher_dashboard()
    elif st.session_state["role"] == "TA":
        ta_dashboard()
    elif st.session_state["role"] == "Student":
        student_dashboard()

# Logout button
if st.button("Logout"):
    logout()
    st.rerun()

```

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

2. Appscript: -

```
function runAllChartFunctions() {  
    var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();  
    var sheetName = 'Evaluation Results'; // Replace with your sheet name  
    var sheet = spreadsheet.getSheetByName(sheetName);  
  
    if (!sheet) {  
        Logger.log("Sheet '" + sheetName + "' not found.");  
        return;  
    }  
  
    createStandardDeviationChart(sheet, 10, 1); // Example position  
    createAverageScoresChart(sheet, 30, 1); // Example position  
    createIndividualPeerScoresChart(sheet, 10, 7); // Example position  
    createScoresClusteredColumnChart(sheet, 30, 7); // Example position  
    createScatterPlotAvgVsStdDev(sheet, 50, 4); // Example position  
}  
  
function createStandardDeviationChart(sheet, row, column) {  
    if (!sheet) {  
        Logger.log("Sheet is undefined in createStandardDeviationChart.");  
        return;  
    }  
  
    var stdRowData = sheet.getRange('B5:I5').getValues()[0]; // Adjust range as needed  
  
    var chartData = [['Student', 'Standard Deviation']];  
    for (var i = 0; i < stdRowData.length; i++) {  
        chartData.push(['Student ' + (400 + i), stdRowData[i]]);  
    }  
}
```

```

var chart = sheet.newChart()
    .setChartType(Charts.ChartType.COLUMN)
    .addRange(sheet.getRange('A5:A5')) // Range for single cell A5 (Student ID label)
    .addRange(sheet.getRange('B5:I5')) // Range for standard deviation data
    .setOption('title', 'Standard Deviation of Scores for Each Student')
    .setOption('hAxis', { title: 'Student', slantedText: false, slantedTextAngle: 0 }) // Set
title for horizontal axis (x-axis) and disable slanted text
    .setOption('vAxis', { title: 'Standard Deviation' }) // Set title for vertical axis (y-axis)
    .setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top
    .setPosition(row, column, 0, 0) // Adjust position as needed
    .build();

sheet.insertChart(chart);
}

```

```

function createAverageScoresChart(sheet, row, column) {
    var avgRowData = sheet.getRange('B6:I6').getValues()[0]; // Adjust range as needed

    var chartData = [['Student', 'Average Score']];
    for (var i = 0; i < avgRowData.length; i++) {
        chartData.push(['Student ' + (400 + i), avgRowData[i]]);
    }
}

```

```

var chart = sheet.newChart()
    .setChartType(Charts.ChartType.COLUMN)
    .addRange(sheet.getRange('A6:A6'))
    .addRange(sheet.getRange('B6:I6'))
    .setOption('title', 'Average Scores for Each Student')
    .setOption('hAxis', { title: 'Student' })
    .setOption('vAxis', { title: 'Average Score' })
    .setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top
    .setPosition(row, column, 0, 0) // Adjust position as needed
    .build();

sheet.insertChart(chart);
}

```

```

function createIndividualPeerScoresChart(sheet, row, column) {
    var students = ['400', '401', '402', '403', '404', '405', '406', '407'];
    var chartData = [['Student', 'Peer 1', 'Peer 2', 'TA']];
}

```

```

for (var i = 0; i < students.length; i++) {
    var peer1Score = sheet.getRange('B2').offset(0, i).getValue();
    var peer2Score = sheet.getRange('B2').offset(1, i).getValue();
    var taScore = sheet.getRange('B2').offset(2, i).getValue();
    chartData.push([students[i], peer1Score, peer2Score, taScore]);
}

// Clear a temporary range to hold the chart data
var tempRange = sheet.getRange('A10:D18');
tempRange.clearContent();
tempRange.setValues(chartData);

var chart = sheet.newChart()
    .setChartType(Charts.ChartType.COLUMN)
    .addRange(tempRange)
    .setOption('title', 'Scores Given by Each Peer and TA')
    .setOption('hAxis', { title: 'Student' })
    .setOption('vAxis', { title: 'Score' })
    .setOption('series', {
        0: { color: 'blue', labelInLegend: 'Peer 1' },
        1: { color: 'red', labelInLegend: 'Peer 2' },
        2: { color: 'green', labelInLegend: 'TA' }
    })
    .setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top
    .setOption('isStacked', false)
    .setPosition(row, column, 0, 0) // Adjust position as needed
    .build();

sheet.insertChart(chart);
}

function createScoresClusteredColumnChart(sheet, row, column) {
    // Example data range:
    // Column A: Assessment/Time Point
    // Columns B-H: Scores for Students 400-407
    // Adjust these ranges based on your actual data layout

    var dataRange = sheet.getRange('A2:I6'); // Adjust this range as needed

```

```

var chart = sheet.newChart()
    .setChartType(Charts.ChartType.COLUMN)
    .addRange(dataRange)
    .setOption('title', 'Clustered Scores Across Students')
    .setOption('hAxis', { title: 'Assessment/Time Point' })
    .setOption('vAxis', { title: 'Score' })
    .setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top
    .setOption('series', {
        0: { color: 'blue', labelInLegend: 'Student 400' },
        1: { color: 'red', labelInLegend: 'Student 401' },
        2: { color: 'green', labelInLegend: 'Student 402' },
        3: { color: 'orange', labelInLegend: 'Student 403' },
        4: { color: 'purple', labelInLegend: 'Student 404' },
        5: { color: 'cyan', labelInLegend: 'Student 405' },
        6: { color: 'magenta', labelInLegend: 'Student 406' },
        7: { color: 'gray', labelInLegend: 'Student 407' }
    })
    .setPosition(row, column, 0, 0) // Adjust position as needed
    .build();

sheet.insertChart(chart);
}

function createScatterPlotAvgVsStdDev(sheet, row, column) {
    var avgRange = sheet.getRange('B6:I6'); // Average scores range
    var stdDevRange = sheet.getRange('B5:I5'); // Standard deviation range

    var avgData = avgRange.getValues()[0];
    var stdDevData = stdDevRange.getValues()[0];

    var scatterData = [['Average Score', 'Standard Deviation']];
    for (var i = 0; i < avgData.length; i++) {
        scatterData.push([avgData[i], stdDevData[i]]);
    }

    // Clear a temporary range to hold the chart data
    var tempRange = sheet.getRange('A30:B38');
    tempRange.clearContent();
    tempRange.setValues(scatterData);
}

```

```
var chart = sheet.newChart()
  .setChartType(Charts.ChartType.SCATTER)
  .addRange(tempRange)
  .setOption('title', 'Average Score vs. Standard Deviation')
  .setOption('hAxis', { title: 'Average Score' })
  .setOption('vAxis', { title: 'Standard Deviation' })
  .setOption('legend', { position: 'none' }) // Hide legend for scatter plot
  .setPosition(row, column, 0, 0) // Adjust position as needed
  .build();

sheet.insertChart(chart);
}
```