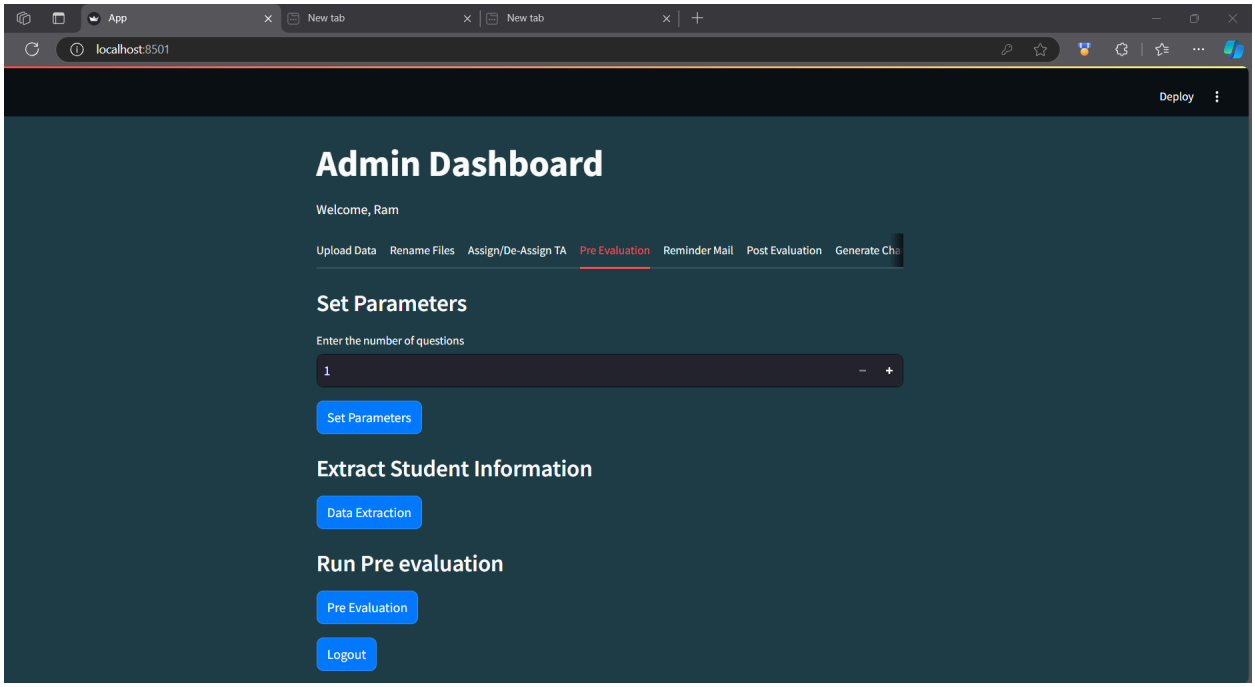
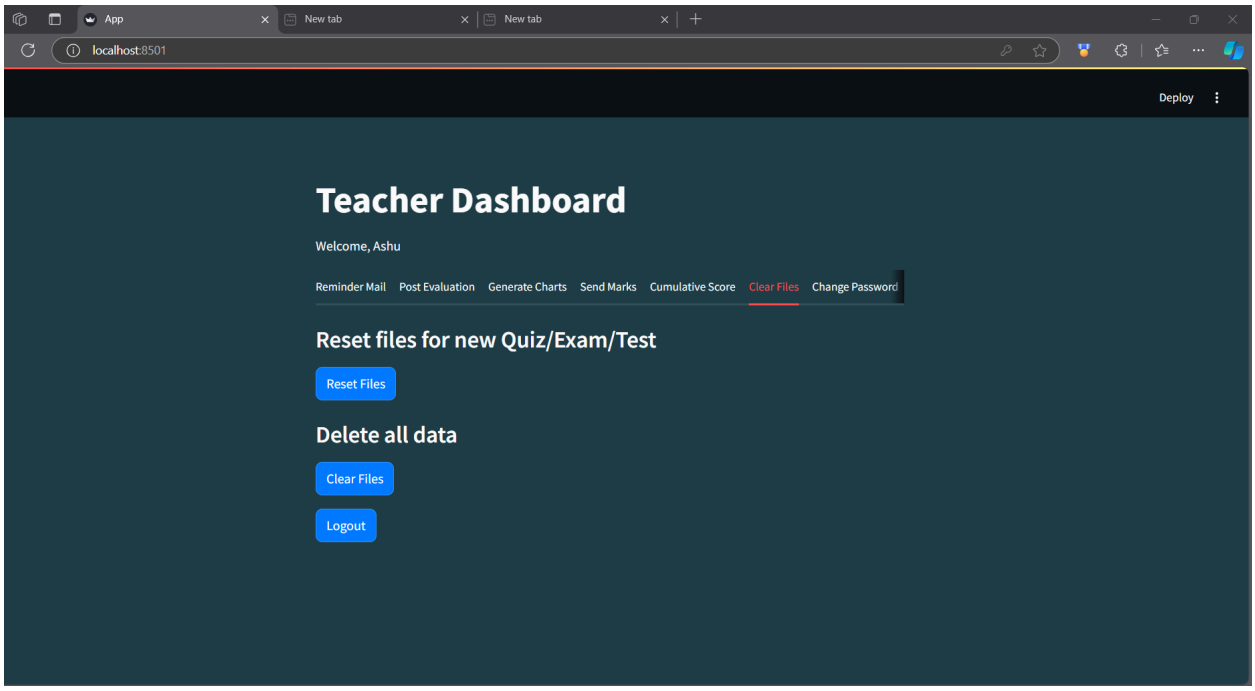
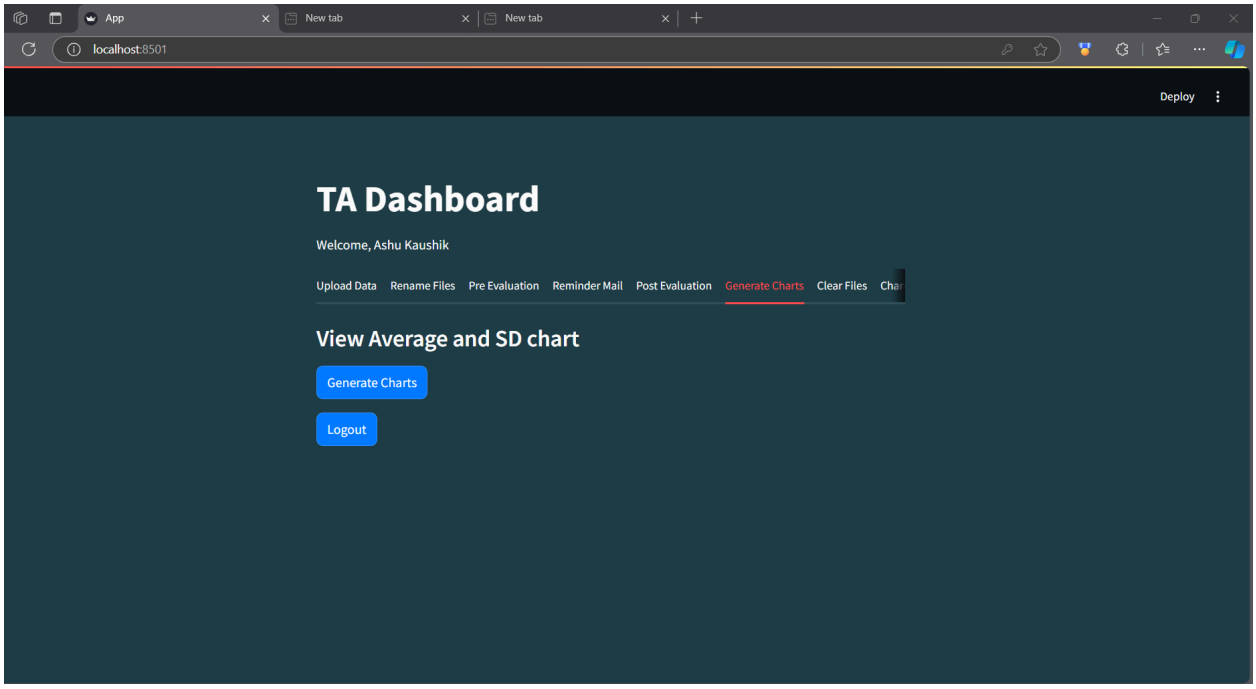
Peer Evaluation System UI/UX

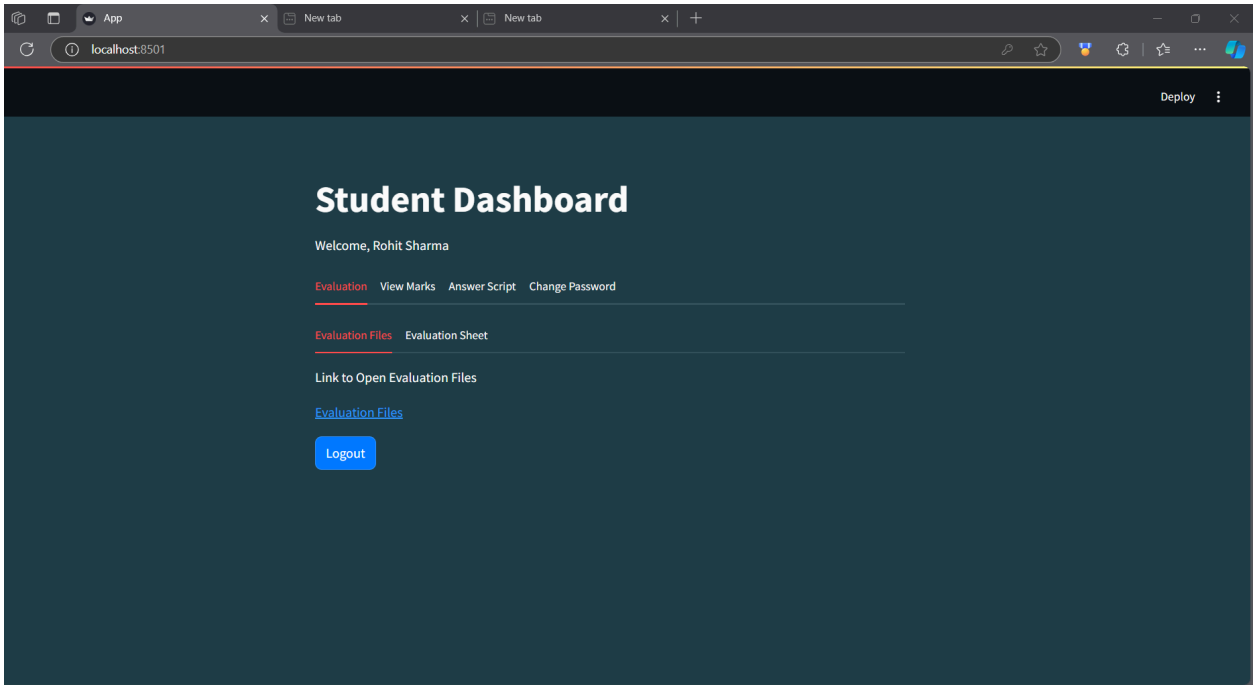
Screenshots of the UI/UX design: -

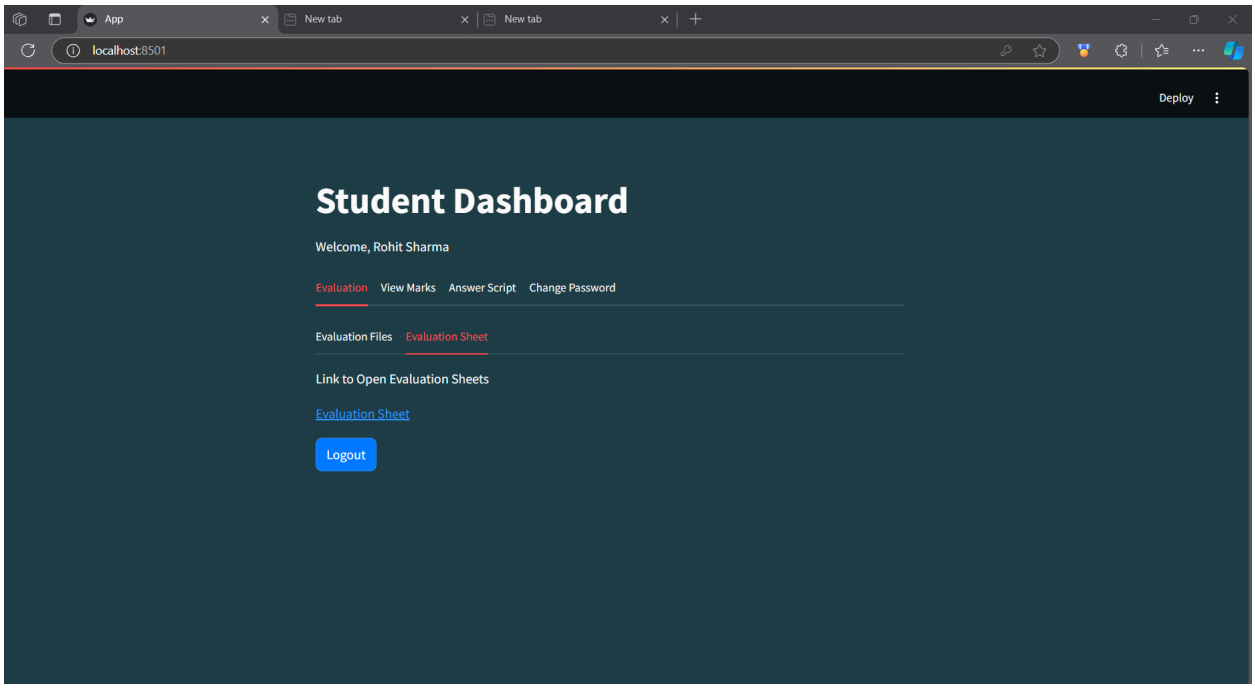
- The changes from the today's code are reflected below: -

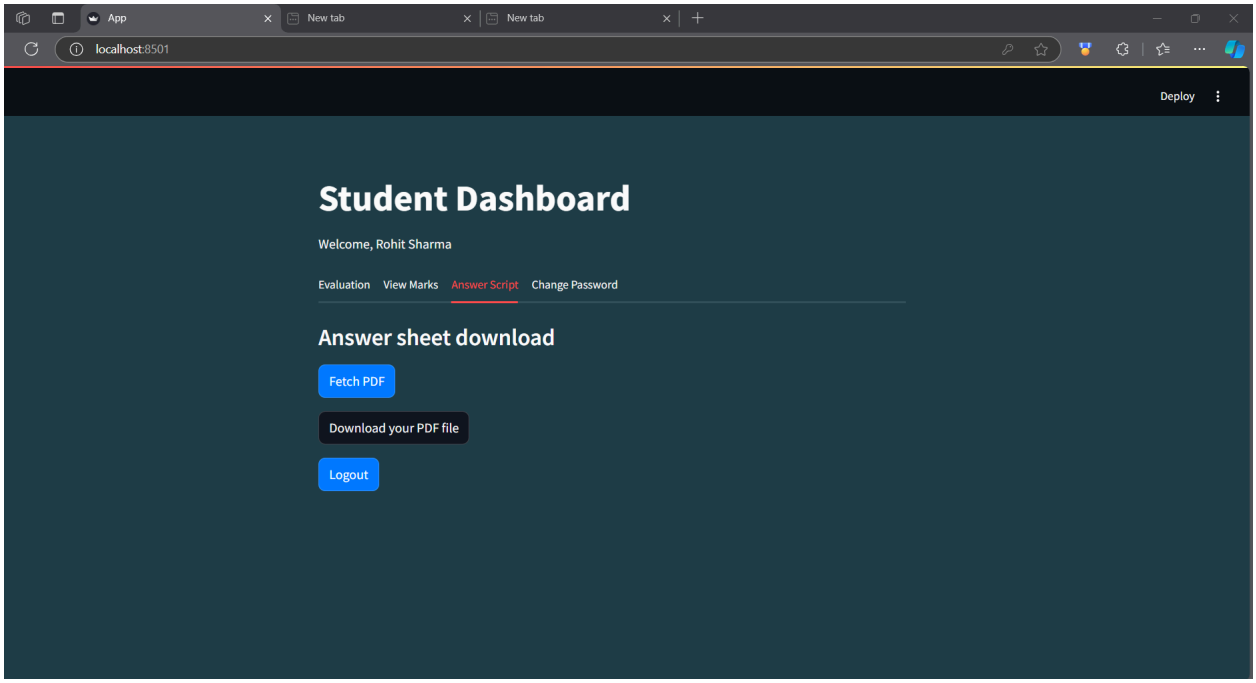





App
New tab
New tab
localhost:8501
Deploy
TA Dashboard
Welcome, Ashu Kaushik
Upload Data
Rename Files
Pre Evaluation
Reminder Mail
Post Evaluation
Generate Charts
Clear Files
View Average and SD chart
Generate Charts
Logout


App
New tab
New tab
localhost:8501
Deploy
Student Dashboard
Welcome, Rohit Sharma
Evaluation
View Marks
Answer Script
Change Password
Evaluation Files
Evaluation Sheet
Link to Open Evaluation Files
Evaluation Files
Logout


App
New tab
New tab
localhost:8501
Deploy
Student Dashboard
Welcome, Rohit Sharma
Evaluation
View Marks
Answer Script
Change Password
Evaluation Files
Evaluation Sheet
Link to Open Evaluation Sheets
Evaluation Sheet
Logout


App
New tab
New tab
localhost:8501
Deploy
Student Dashboard
Welcome, Rohit Sharma
Evaluation
View Marks
Answer Script
Change Password
Answer sheet download
Fetch PDF
Download your PDF file
Logout

Python Code: -

1. **App.py -**

```
import io
import re
import random
import smtplib
import time
import json
import bcrypt
import gspread
import requests
import streamlit as st
import plotly.express as px
import pandas as pd
from Rename_File import process_pdfs_in_folder  # process_signatures,
load_stored_signatures
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseUpload
from googleapiclient.http import MediaIoBaseDownload
from oauth2client.service_account import ServiceAccountCredentials
from email.mime.text import MIMEText

# Google Sheets and Google Drive setup
SCOPE = [
  "https://spreadsheets.google.com/feeds",
  "https://www.googleapis.com/auth/drive"
]
CREDENTIALS_FILE = "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


# Function to generate a random 6-digit OTP
def generate_otp():
  return str(random.randint(100000, 999999))


# Function to send OTP to email
def send_otp_email(otp, recipient_email, name):
  sender_email = "rohit.24csz0014@iitrpr.ac.in"  # Replace with your email
  sender_password = "azta drev diys dahu"  # Replace with your email password or
app-specific password

  # Create the email content
  message = MIMEText(f"Hello {name},\n\n Your Registration OTP is: {otp}.\n\n
Thanks & Regards,\nCSE, IIT Ropar")
  message["Subject"] = "OTP for Registration"
  message["From"] = sender_email
  message["To"] = recipient_email

  try:
    # Connect to SMTP server and send email
    server = smtplib.SMTP_SSL("smtp.gmail.com", 465)
    server.login(sender_email, sender_password)
    response = server.sendmail(sender_email, recipient_email, message.as_string())
```

```
        server.quit()
        if response:  # If response contains something, it means there was an issue
            print(f"Error: {response}")
            return False
        else:
            print("Email sent successfully!")
            return True  # Email sent successfully
    except smtplib.SMTPRecipientsRefused:
        # Handle case where recipient email is invalid or not found
        print(f"Error: Address not found for recipient: {recipient_email}")
        return -1  # Return -1 if the recipient email is invalid
    except Exception as e:
        print(f"Error sending OTP email: {e}")
        return False


def validate_password(password):
    pattern =
re.compile(r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}
$')
    return pattern.match(password)


# Add new user to Google Sheets with role auto-assignment
def register_user(username, password, name):
    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, name]

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


# Update role from TA to Student (only for Teachers)
def update_role_to_Student(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'] == 'TA':
      sheet.update_cell(i, 3, 'Student')  # 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
        st.session_state["name"] = user["name"]
        return
```

```
            else:
                st.error("Incorrect Password!")
                time.sleep(2)
                st.rerun()
                return
    st.error("Incorrect Username or Password!")
    time.sleep(2)
    st.rerun()


# Logout function
def logout():
    st.session_state["login_status"] = False
    st.session_state["role"] = None
    st.session_state["username"] = None
    st.session_state["name"] = None
    st.session_state["page"] = "login"
    st.success("Logging out!")
    time.sleep(0.5)
    # 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!")
        elif not validate_password(new_password):
            st.error(
                "Password must include at least: - \n1. One uppercase letter. \n2. One lowercase
letter. \n3. One special character. \n4. One numerical digit. \n5. Must be at least 8
characters long.")
        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/AKfycbwlBil062YhNYcbIqmP9obfLBKgoeIdTdRD
Q_BOB4rF1S6JhTxvVFH8MhW2x84bgyAVag/exec"
    # web_app_url =
"https://script.google.com/macros/s/AKfycbwyD-ImNeqaa7NzPCU5AZ6978PSRvQuuy
EreskmCnrzoM0P30EGHuR-sIoqklHBHlNlxQ/exec"
```

```
  #web_app_url =
"https://script.google.com/macros/s/AKfycbyyeKLZfm5VihTXPqu5qloIl2CEs8mRU3iJu
epZJ4AvWm3R0w3yWBG8INj9v2-TUeGk3w/exec"
  # web_app_url =
"https://script.google.com/macros/s/AKfycbwBrsqBz4_d-muszL1oclHczAqQtpddtmzRc
ZPtV9NVXZuS0t-18kWYfUT-F3W728A-AA/exec"
  # web_app_url =
"https://script.google.com/macros/s/AKfycbwBrsqBz4_d-muszL1oclHczAqQtpddtmzRc
ZPtV9NVXZuS0t-18kWYfUT-F3W728A-AA/exec"
  web_app_url =
"https://script.google.com/macros/s/AKfycbxIS5UFnp4xLX9oL0Zne8xnQ-pU1YNZbJO
LcEO_k_hF6eMAIVFChspBu36WHQ9GQLUKfg/exec"


  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_file, folder_id):
  drive_service = authenticate_drive()

  if file_exists(drive_service, folder_id, uploaded_file.name):
    st.warning(f"Google Sheet file '{uploaded_file.name}' already exists in the folder.")
    return

  file_metadata = {
    'name': uploaded_file.name,
    'parents': [folder_id]
  }

  # Upload the file to Google Drive
  media = MediaIoBaseUpload(uploaded_file, mimetype=uploaded_file.type,
resumable=True)
  uploaded = 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 connect_to_google_sheets_with_id(file_id):
  creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
  client = gspread.authorize(creds)
  sheet = client.open_by_key(file_id)
  return sheet


def get_student_details(username):
  # Connect to the specific Google Sheet containing details
  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()

  # Check if data exists
  if not records or 'Assigned Folder Link' not in records[0] or 'Spreadsheet Link' not in
records[0]:  # Checking the first row (header) for the column
    return -1, -1, -1  # Return None if the column does not exist

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

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


def get_student_marks(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()

    # Check if 'Average Marks' column exists
    if not records or 'Average Marks' not in records[0]:  # Checking the first row (header) for the column
        # print("Average Marks column not found")
        return -1  # Return None if the column does not exist

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

    return None  # If no marks 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 renaming_files():
    folder_id = '1fT-inciLQut85BGEQrjMSWbVRcTsdWfQ'  # Replace with your Google Drive folder ID
    process_pdfs_in_folder(folder_id)
    # Authenticate Google Drive
    # service = authenticate_drive()

    # Google Drive folder IDs
    # stored_signatures_folder_id = '14QLNPdIRUZ3ici-GePoEewUCmxemjhUD' #The folder where we want to keep the Stored signature
    # uploaded_signatures_folder_id = '1ORVrU-UoXyDS-1ovyuk7FAjb_p94gnsx' #This will be the folder where our pdf files are kept
    # destination_folder_id = '1bPhLMZONpsPDxM9z_vQD2J9jAjtwG3FG'  # Folder where renamed files will be moved

    # Load stored signatures directly from Google Drive
    # stored_signatures, stored_filenames = load_stored_signatures(service, stored_signatures_folder_id)

    # Process uploaded signatures and copy renamed files to the destination folder
    # matched_files = process_signatures(service, uploaded_signatures_folder_id, stored_signatures, stored_filenames, destination_folder_id)

    # print(f"Matching process completed. Total matched files: {len(matched_files)}")
```

```
# Connect to Google Sheets
def fetch_sheet_data(sheet_name, worksheet_name):
  # gc = gspread.service_account(filename=CREDENTIALS_FILE)
  # spreadsheet = gc.open(sheet_name)
  # worksheet = spreadsheet.worksheet(worksheet_name)
  # return worksheet.get_all_values()
  creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
                                                           SCOPE)  # Replace with your credentials file
  client = gspread.authorize(creds)
  sheet = client.open(sheet_name).worksheet(worksheet_name)  # Open the specific
worksheet
  return sheet


# Fetch Peer Average and Standard Deviation data
def get_peer_average_data(sheet):
  data = sheet.get_all_values()

  for index, row in enumerate(data):
    if row[0] == "Peer":  # Check the first column of each row
      st_idx = index
    if row[0] == "SD":  # Check the first column of each row
      std_idx = index
    if row[0] == "Peer Average":  # Check the first column of each row
      pavg_idx = index

  # Extract student IDs, Standard Deviations, and Peer Averages
  student_ids = data[st_idx][1:]  # First row has student IDs starting from B1
  std_devs = list(map(float, data[std_idx][1:]))  # Row 6 has Standard Deviations starting
from B6
  peer_averages = list(map(float, data[pavg_idx][1:]))  # Row 7 has Peer Averages
starting from B7

  # Create a DataFrame with relevant data
  df = pd.DataFrame({
    'Student': student_ids,
    'Standard Deviation': std_devs,
    'Average Score': peer_averages
  })
```

```
  return df


# Generate Standard Deviation Chart
def create_standard_deviation_chart(df):
  fig = px.bar(df, x='Student', y='Standard Deviation', title='Standard Deviation of Scores
for Each Student',
          color_discrete_sequence=['#4682B4'])
  fig.update_layout(yaxis_title='Standard Deviation', xaxis_title='Student',
              plot_bgcolor='white', paper_bgcolor='white',  # Set background colors to
white
              font_color='black', title_font_color='black',
              xaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black')),
              yaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black'))
              )
  return fig


# Generate Peer Average Scores Chart
def create_average_scores_chart(df):
  fig = px.bar(df, x='Student', y='Average Score', title='Peer Average Scores for Each
Student',
          color_discrete_sequence=['#FF6347'])
  fig.update_layout(yaxis_title='Peer Average', xaxis_title='Student',
              plot_bgcolor='white', paper_bgcolor='white',  # Set background colors to
white
              font_color='black', title_font_color='black',
              xaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black')),
              yaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black'))
              )
  return fig


# Trigger when "Generate Charts" button is clicked
def generate_charts():
  sheet = fetch_sheet_data('UI/UX Copy of Peer Evaluation2', 'Evaluation Results')
  df = get_peer_average_data(sheet)  # Get the data
  # Display two charts
  st.plotly_chart(create_standard_deviation_chart(df))
  st.plotly_chart(create_average_scores_chart(df))
```

```
# Function to convert Excel file to Google Sheets
def convert_excel_to_google_sheet(file_id):
  drive_service = authenticate_drive()

  # File metadata for conversion
  file_metadata = {
    'mimeType': 'application/vnd.google-apps.spreadsheet'
  }

  # Convert the file and get the new file ID
  converted_file = drive_service.files().copy(fileId=file_id,
body=file_metadata).execute()
  return converted_file['id']


# Function to list all files in the given folder (Google Drive)
def list_spreadsheets_in_folder(folder_id):
  service = authenticate_drive()  # Authenticate Drive using the new function
  query = f"'{folder_id}' in parents and
(mimeType='application/vnd.google-apps.spreadsheet' OR
mimeType='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')"
  results = service.files().list(q=query).execute()
  files = results.get('files', [])

  if not files:
    print('No spreadsheets found in the folder.')
    return []

  return files


# Function to delete only spreadsheet files in the specified folder
def delete_all_spreadsheets_in_folder(folder_id):
  drive_service = authenticate_drive()

  # List all files in the folder
  query = f"'{folder_id}' in parents and trashed = false"
  results = drive_service.files().list(q=query).execute()
```

```
    files = results.get('files', [])

    # Iterate through files and delete only those with the MIME type for Google Sheets
    for file in files:
        if file['mimeType'] == 'application/vnd.google-apps.spreadsheet' or file['mimeType']
== 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet':
            try:
                drive_service.files().delete(fileId=file['id']).execute()
                print(f"Deleted spreadsheet: {file['name']}")
            except Exception as e:
                print(f"Error deleting spreadsheet {file['name']}: {e}")
        else:
            print(f"Skipped non-spreadsheet file: {file['name']}")


# Function to copy data from each spreadsheet to the target sheet
def copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name):
    # Get the target spreadsheet using the new function
    target_spreadsheet = connect_to_google_sheets_with_name(target_spreadsheet_name)

    try:
        target_worksheet = target_spreadsheet.worksheet(target_worksheet_name)
    except gspread.WorksheetNotFound:
        print(f"Worksheet '{target_worksheet_name}' not found in the target spreadsheet.")
        return

    target_worksheet.clear()
    print("Target worksheet cleared successfully.")

    # List all spreadsheets in the folder
    files = list_spreadsheets_in_folder(folder_id)

    for file in files:
        file_id = file['id']
        file_name = file['name']
        mime_type = file['mimeType']

        print(f"Processing file: {file_name}")
```

```
        # If it's an Excel file, convert it to a Google Sheet
        if mime_type ==
'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet':
            file_id = convert_excel_to_google_sheet(file_id)

        # Now open the file as a Google Sheet

        source_spreadsheet = connect_to_google_sheets_with_id(file_id)
        source_worksheet = source_spreadsheet.get_worksheet(0)  # Assuming data is in the
first worksheet
        source_data = source_worksheet.get_all_values()

        # Append data to the target worksheet
        if source_data:
            target_worksheet.append_rows(source_data, value_input_option="RAW")
            return

    print("Data copied to PeerEval worksheet successfully.")
    #delete_all_spreadsheets_in_folder(folder_id)
    print("Spreadsheet is removed successfully")


# Function to clear all worksheet data except "Cumulative Marks"
def reset_worksheets(spreadsheet_name, keep_sheet_name):
    # Open the target spreadsheet
    spreadsheet = connect_to_google_sheets_with_name(spreadsheet_name)

    for worksheet in spreadsheet.worksheets():
        # Check if worksheet name is not "Cumulative Marks"
        if worksheet.title != keep_sheet_name:
            # Clear the content of the worksheet
            worksheet.clear()

    print(f"All worksheets except {keep_sheet_name} have been cleared.")


# Function to delete all worksheets except the specified one
def delete_other_worksheets(target_spreadsheet_name, keep_sheet_name):
    spreadsheet = connect_to_google_sheets_with_name(target_spreadsheet_name)
    try:
```

```
        # List all worksheets in the spreadsheet
        all_worksheets = spreadsheet.worksheets()

        # Loop through each worksheet
        for sheet in all_worksheets:
            # Check if the worksheet is not the one to keep
            if sheet.title != keep_sheet_name:
                # Delete the worksheet
                spreadsheet.del_worksheet(sheet)
                print(f"Deleted worksheet: {sheet.title}")

        print(f"All worksheets except '{keep_sheet_name}' have been deleted.")
    except Exception as e:
        print(f"Error while deleting worksheets: {e}")


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

    # Create tabs for each action
    tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6, tab7, tab8, tab9 = st.tabs(
        ["Upload Data", "Rename Files", "Assign/De-Assign TA", "Pre Evaluation",
"Reminder Mail", "Post Evaluation",
         "Generate Charts", "Send Marks", "Cumulative Score", "Clear Files", "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 Student Information")
        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)
            time.sleep(2)
            logout()
            st.rerun()

    with tab0:
        st.subheader("Rename PDF Files")
        if st.button("Rename Files"):
            renaming_files()

    # Tab for TA update
    with tab1:
        student_username = st.text_input("Enter Student's Username")
        st.subheader("Assign TA")
        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.")
        st.subheader("De-assign TA")
        if st.button("Update Role to Student"):
            if update_role_to_Student(student_username):
                st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to
Student.")
            else:
                st.error("Failed to update the role. Check if the username exists and belongs to a
TA.")

    # Tab for Pre Evaluation
    with tab2:
        st.subheader("Set Parameters")
```

```
    num_Questions = st.number_input("Enter the number of questions", min_value=1,
max_value=100)
    # Button to submit the form
    if st.button("Set Parameters"):
        # Prepare the data to send
        data = {
            "num_Questions": num_Questions  # Send `num_Questions` parameter
        }

        # Google Apps Script Web App URL (replace with your actual Web App URL)
        url =
"https://script.google.com/macros/s/AKfycbxIS5UFnp4xLX9oL0Zne8xnQ-pU1YNZbJO
LcEO_k_hF6eMAIVFChspBu36WHQ9GQLUKfg/exec"
        headers = {'Content-Type': 'application/json'}

        try:
            # Send a POST request to the Google Apps Script Web App
            response = requests.post(url, data=json.dumps(data), headers=headers)
            if response.status_code == 200:
                st.success(f"Data sent successfully! Response: {response.text}")
            else:
                st.error(f"Failed to send data. Status code: {response.status_code}")
        except Exception as e:
            st.error(f"Error occurred: {e}")

    target_spreadsheet_name = "UI/UX Copy of Peer Evaluation2"
    # target_spreadsheet_name = "Sample_Run"
    target_worksheet_name = "PeerEval"
    st.subheader("Extract Student Information")
    if st.button("Data Extraction"):
        copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        delete_all_spreadsheets_in_folder(folder_id)
    st.subheader("Run Pre evaluation")
    if st.button("Pre Evaluation"):
        trigger_google_apps_script("PreEval")

  # Tab for Checking Pending Evaluations
  with tab3:
    st.subheader("Send Reminder Mail")
```

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

    # Tab for Post Evaluation
    with tab4:
        st.subheader("Run Post evaluation")
        if st.button("Post Evaluation"):
            trigger_google_apps_script("PostEval")

    # Tab for Generating Charts
    with tab5:
        st.subheader("View Average and SD chart")
        if st.button("Generate Charts"):
            generate_charts()
            # trigger_google_apps_script("GenChart")

    # Tab for Sending Marks
    with tab6:
        st.subheader("Share Marks with Students")
        if st.button("Send Marks"):
            trigger_google_apps_script("SendMail")

    with tab7:
        st.subheader("Total Score")
        if st.button("Send Cumulative Score"):
            trigger_google_apps_script("SendFinalM")

    with tab8:
        st.subheader("Reset files for new Quiz/Exam/Test")
        if st.button("Reset Files"):
            reset_worksheets(target_spreadsheet_name, "Cumulative Marks")
        st.subheader("Delete all data")
        if st.button("Clear Files"):
            delete_other_worksheets(target_spreadsheet_name, target_worksheet_name)

    with tab9:
        change_password_dashboard()


def teacher_dashboard():
```

```
  st.title("Teacher Dashboard")
  # 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()}")
  st.write(f"Welcome, {st.session_state['name']}")

  # Create tabs for each action
  tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6, tab7, tab8 = st.tabs(
     ["Rename Files", "Assign/De-Assign TA", "Pre Evaluation", "Reminder Mail", "Post
Evaluation",
      "Generate Charts", "Send Marks", "Cumulative Score", "Clear Files", "Change
Password"])

  # Tab for TA update
  with tab:
     st.subheader("Rename PDF Files")
     if st.button("Rename Files"):
        renaming_files()
  with tab0:
     student_username = st.text_input("Enter Student's Username")
     st.subheader("Assign TA")
     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.")
     st.subheader("De-assign TA")
     if st.button("Update Role to Student"):
        if update_role_to_Student(student_username):
           st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to
Student.")
        else:
           st.error("Failed to update the role. Check if the username exists and belongs to a
TA.")

  # Tab for Pre Evaluation
  with tab1:
```

```
    st.subheader("Set Parameters")
    num_Questions = st.number_input("Enter the number of questions", min_value=1,
max_value=100)
    # Button to submit the form
    if st.button("Set Parameters"):
        # Prepare the data to send
        data = {
            "num_Questions": num_Questions  # Send `num_Questions` parameter
        }

        # Google Apps Script Web App URL (replace with your actual Web App URL)
        url =
"https://script.google.com/macros/s/AKfycbxIS5UFnp4xLX9oL0Zne8xnQ-pU1YNZbJO
LcEO_k_hF6eMAIVFChspBu36WHQ9GQLUKfg/exec"
        headers = {'Content-Type': 'application/json'}

        try:
            # Send a POST request to the Google Apps Script Web App
            response = requests.post(url, data=json.dumps(data), headers=headers)
            if response.status_code == 200:
                st.success(f"Data sent successfully! Response: {response.text}")
            else:
                st.error(f"Failed to send data. Status code: {response.status_code}")
        except Exception as e:
            st.error(f"Error occurred: {e}")

    folder_id = "1fT-inciLQut85BGEQrjMSWbVRcTsdWfQ"
    target_spreadsheet_name = "UI/UX Copy of Peer Evaluation2"
    # target_spreadsheet_name = "Sample_Run"
    target_worksheet_name = "PeerEval"
    st.subheader("Extract Student Information")
    if st.button("Data Extraction"):
        copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        delete_all_spreadsheets_in_folder(folder_id)
    st.subheader("Run Pre evaluation")
    if st.button("Pre Evaluation"):
        # copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        trigger_google_apps_script("PreEval")
```

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

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

# Tab for Generating Charts
with tab4:
    st.subheader("View Average and SD chart")
    if st.button("Generate Charts"):
        generate_charts()
        # trigger_google_apps_script("GenChart")
with tab5:
    st.subheader("Share Marks with Students")
    if st.button("Send Marks"):
        trigger_google_apps_script("SendMail")

with tab6:
    st.subheader("Total Score")
    if st.button("Send Cumulative Score"):
        trigger_google_apps_script("SendFinalM")

with tab7:
    st.subheader("Reset files for new Quiz/Exam/Test")
    if st.button("Reset Files"):
        reset_worksheets(target_spreadsheet_name, "Cumulative Marks")
    st.subheader("Delete all data")
    if st.button("Clear Files"):
        delete_other_worksheets(target_spreadsheet_name, target_worksheet_name)

with tab8:
    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['name']}")

  # Create tabs for each action
  tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6 = st.tabs(
     ["Upload Data", "Rename Files", "Pre Evaluation", "Reminder Mail", "Post
Evaluation", "Generate Charts",
      "Clear Files", "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 Student Information")
     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)
        time.sleep(2)
        logout()
        st.rerun()

  with tab0:
```

```
    st.subheader("Rename PDF Files")
    if st.button("Rename Files"):
        renaming_files()

  # Tab for Pre Evaluation
  with tab1:
    st.subheader("Set Parameters")
    num_Questions = st.number_input("Enter the number of questions", min_value=1,
max_value=100)
    # Button to submit the form
    if st.button("Set Parameters"):
        # Prepare the data to send
        data = {
            "num_Questions": num_Questions  # Send `num_Questions` parameter
        }

        # Google Apps Script Web App URL (replace with your actual Web App URL)
        url =
"https://script.google.com/macros/s/AKfycbxIS5UFnp4xLX9oL0Zne8xnQ-pU1YNZbJO
LcEO_k_hF6eMAIVFChspBu36WHQ9GQLUKfg/exec"
        headers = {'Content-Type': 'application/json'}

        try:
            # Send a POST request to the Google Apps Script Web App
            response = requests.post(url, data=json.dumps(data), headers=headers)
            if response.status_code == 200:
                st.success(f"Data sent successfully! Response: {response.text}")
            else:
                st.error(f"Failed to send data. Status code: {response.status_code}")
        except Exception as e:
            st.error(f"Error occurred: {e}")


    folder_id = "1fT-inciLQut85BGEQrjMSWbVRcTsdWfQ"
    target_spreadsheet_name = "UI/UX Copy of Peer Evaluation2"
    # target_spreadsheet_name = "Sample_Run"
    target_worksheet_name = "PeerEval"
    st.subheader("Extract Student Information")
    if st.button("Data Extraction"):
```

```
            copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
            delete_all_spreadsheets_in_folder(folder_id)
        st.subheader("Run Pre evaluation")
        if st.button("Pre Evaluation"):
            # copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
            trigger_google_apps_script("PreEval")

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

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

    # Tab for Generating Charts
    with tab4:
        st.subheader("View Average and SD chart")
        if st.button("Generate Charts"):
            generate_charts()
            # trigger_google_apps_script("GenChart")

    with tab5:
        st.subheader("Reset files for new Quiz/Exam/Test")
        if st.button("Reset Files"):
            reset_worksheets(target_spreadsheet_name, "Cumulative Marks")
        st.subheader("Delete all data")
        if st.button("Clear Files"):
            delete_other_worksheets(target_spreadsheet_name, target_worksheet_name)

    with tab6:
        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['name']}")

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

  # Tab for opening the peer evaluation spreadsheet
  with tab1:
    if st.session_state["username"]:
      # Fetch unique ID, and spreadsheet link using the session's username
      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:
      st.write("Link to Open Evaluation Files")
      if folder_link == -1:
        st.success("Hurray! Nothing to Evaluate.")
      elif folder_link:
        st.markdown(f"[Evaluation Files]({folder_link})", unsafe_allow_html=True)
      else:
        st.error("Folder link not found.")
    with t2:
      st.write("Link to Open Evaluation Sheets")
      if sheet_link == -1:
        st.success("Hurray! Nothing to Evaluate.")
      elif sheet_link:
        st.markdown(f"[Evaluation Sheet]({sheet_link})", unsafe_allow_html=True)
      else:
        st.error("Spreadsheet link not found.")

  # Tab for viewing marks
  with tab2:
    st.subheader("View Score")
    if st.button("Fetch Marks"):
```

```
            # Fetch Marks
            marks = get_student_marks(st.session_state["username"])
            if marks and unique_id and marks != -1:
                st.write(f"Your evaluation marks are =  {marks}")
            elif marks == -1:
                st.success("Marks are not released yet!")
            else:
                st.error("No marks are available!\nCheck with Teacher/TA.")

        # Tab for downloading PDF
        with tab3:
            st.subheader("Answer sheet download")
            if unique_id == -1:
                st.success("PDF not released yet!")
            else:
                if st.button("Fetch PDF"):
                    pdf_file, file_name = get_student_pdf(unique_id)
                    if pdf_file:
                        st.download_button(
                            label="Download your PDF file",
                            data=pdf_file,
                            file_name=file_name,
                            mime='application/pdf'
                        )
                    else:
                        st.error("PDF not found! Contact Teacher/TA.")

        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
if "name" not in st.session_state:
    st.session_state["name"] = 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")

        if "otp_sent" not in st.session_state:
            st.session_state["otp_sent"] = False  # Flag to track if OTP is sent
        if "otp_verified" not in st.session_state:
            st.session_state["otp_verified"] = False  # Flag to track if OTP is verified

        with st.form(key='register_form'):
            reg_name = st.text_input("Name", key='reg_name')
            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 be of @iitrpr.ac.in domain.")
                elif not validate_password(reg_password):
                    st.error(
                        "Password must include at least One: - \n1. Uppercase letter. \n2.
Lowercase letter. \n3. Special character. \n4. Numerical digit. \n5. Must be at least 8
characters long.")
                else:
                    var_otp = generate_otp()
                    if send_otp_email(var_otp, reg_username, reg_name):
                        st.session_state["otp_sent"] = True
                        st.session_state["otp"] = var_otp
                        st.session_state["username"] = reg_username
                        st.session_state["password"] = reg_password
```

```
                    st.session_state["name"] = reg_name
                    st.success(f"OTP has been sent to the {reg_username}")
                elif send_otp_email(var_otp, reg_username, reg_name) == -1:
                    st.error("Error: Email Address not found!")
                else:
                    st.error("Error! OTP not sent.")
        if st.session_state.get("otp_sent", False):
            with st.form(key='verification_form'):
                reg_otp = st.text_input("OTP", key='reg_otp')
                verify_button = st.form_submit_button("Verify")
                if verify_button:
                    if reg_otp == st.session_state["otp"]:
                        users = get_users_from_sheets()
                        if any(user['username'] == st.session_state["username"] for user in
users):
                            st.error("Username already exists")
                        else:
                            role = register_user(st.session_state["reg_username"],
st.session_state["password"], st.session_state["name"])
                            st.success(f"User registered successfully with role: {role}")
                            time.sleep(2)
                            # Redirect to the login page
                            st.session_state["page"] = "login"
                            st.rerun()
                    else:
                        st.error("Incorrect OTP!")

    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. **Rename_File.py -**

```
from pdf2image import convert_from_bytes
from PIL import Image
import pytesseract
import re
import io
from googleapiclient.discovery import build
from google.oauth2.service_account import Credentials
from googleapiclient.http import MediaIoBaseDownload, MediaIoBaseUpload

SCOPE = [
  "https://www.googleapis.com/auth/drive",
  "https://www.googleapis.com/auth/spreadsheets",
]
CREDENTIALS_FILE = "peer-evaluation-sem1-e2fcf8b5fc27.json"

# Authenticate Google Drive
def authenticate_drive():
  creds = Credentials.from_service_account_file(CREDENTIALS_FILE,
scopes=SCOPE)
  service = build('drive', 'v3', credentials=creds)
  return service

# Crop the top left corner of the image
def crop_top_left(image, crop_width, crop_height):
  left = 0
  top = 0
  right = crop_width
  bottom = crop_height
  return image.crop((left, top, right, bottom))

# Get PDF content from Google Drive without downloading
def get_pdf_from_drive(service, file_id):
  request = service.files().get_media(fileId=file_id)
  pdf_content = io.BytesIO()
  downloader = MediaIoBaseDownload(pdf_content, request)

  done = False
  while not done:
    status, done = downloader.next_chunk()
```

```
        print(f"Download {int(status.progress() * 100)}%.")

    pdf_content.seek(0)  # Move to the beginning of the stream
    return pdf_content

# Extract name from the top-left corner of the first page
def name_extraction(service, folder_id, file_id, pdf_filename):
    try:
        pdf_content = get_pdf_from_drive(service, file_id)
        images = convert_from_bytes(pdf_content.read())
        print(f"Successfully converted PDF to images. Number of pages: {len(images)}")
    except Exception as e:
        print(f"Error converting PDF to images: {e}")
        return

    if not images:
        raise Exception("Failed to convert PDF to images")

    image = images[0]  # Only the first page is processed
    crop_width = int(image.width * 0.2)
    crop_height = int(image.height * 0.1)

    cropped_image = crop_top_left(image, crop_width, crop_height)

    recognised_text = pytesseract.image_to_string(cropped_image, config='--psm 6')

    extracted_name = re.findall(r'\b\d{3}\b', recognised_text)
    text = "".join(extracted_name)

    if text:
        new_pdf_filename = f"{text}.pdf"
        print("Extracted name for renaming:", new_pdf_filename)
    else:
        print("No valid name extracted, keeping the original filename.")
        new_pdf_filename = pdf_filename

    # Re-upload the renamed PDF to Google Drive
    upload_pdf(service, folder_id, pdf_content, new_pdf_filename)

    # Delete the original PDF file
```

```
    delete_pdf(service, file_id)

# Upload the renamed file back to Google Drive without saving it locally
def upload_pdf(service, folder_id, pdf_content, new_filename):
    pdf_content.seek(0)  # Reset the stream position before uploading
    file_metadata = {
        'name': new_filename,
        'parents': [folder_id]
    }
    media = MediaIoBaseUpload(pdf_content, mimetype='application/pdf')
    file = service.files().create(body=file_metadata, media_body=media,
fields='id').execute()
    print(f"Uploaded renamed file as {new_filename}. File ID: {file.get('id')}")

# Function to delete the original PDF file
def delete_pdf(service, file_id):
    try:
        service.files().delete(fileId=file_id).execute()
        print(f"Deleted original PDF file with ID: {file_id}")
    except Exception as e:
        print(f"Error deleting file: {e}")

# Main function to process PDFs in the Google Drive folder
def process_pdfs_in_folder(folder_id):
    service = authenticate_drive()

    # List PDF files in the folder
    query = f"'{folder_id}' in parents and mimeType='application/pdf'"
    results = service.files().list(q=query, fields="files(id, name)").execute()
    files = results.get('files', [])

    if not files:
        raise Exception("No PDF files found in the specified folder")

    # Process each PDF file
    for file in files:
        file_id = file['id']
        pdf_filename = file['name']
        print(f"Processing file: {pdf_filename}")
        name_extraction(service, folder_id, file_id, pdf_filename)
```