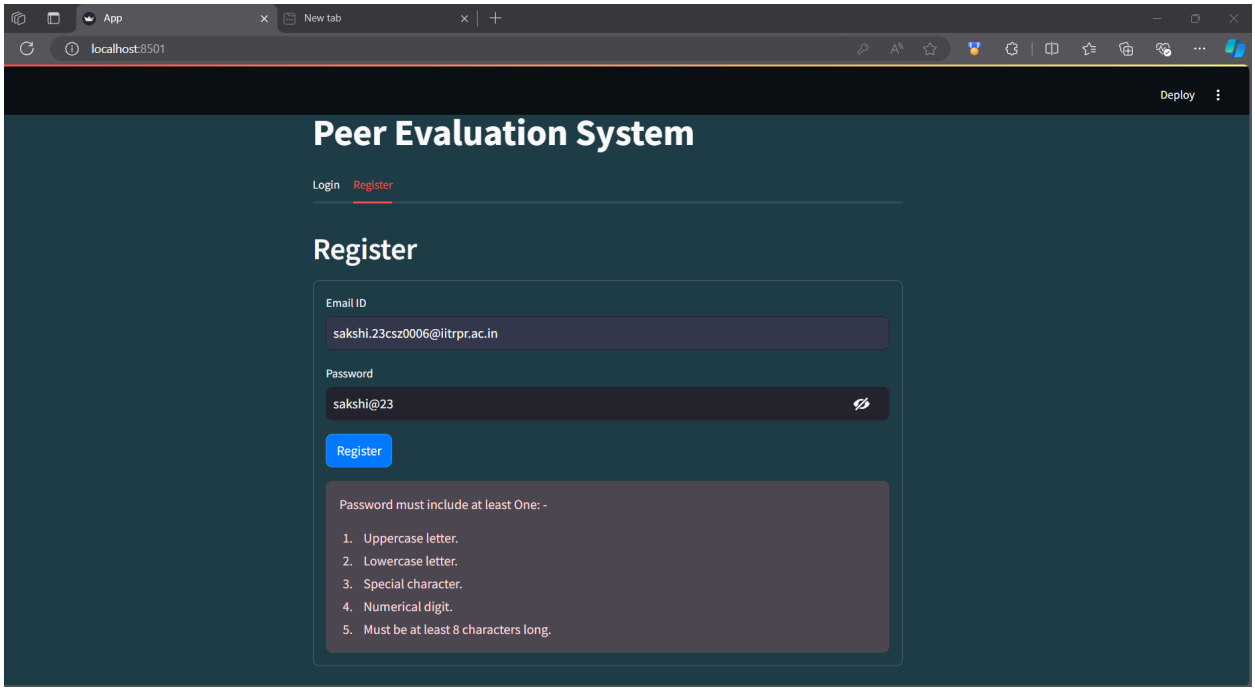
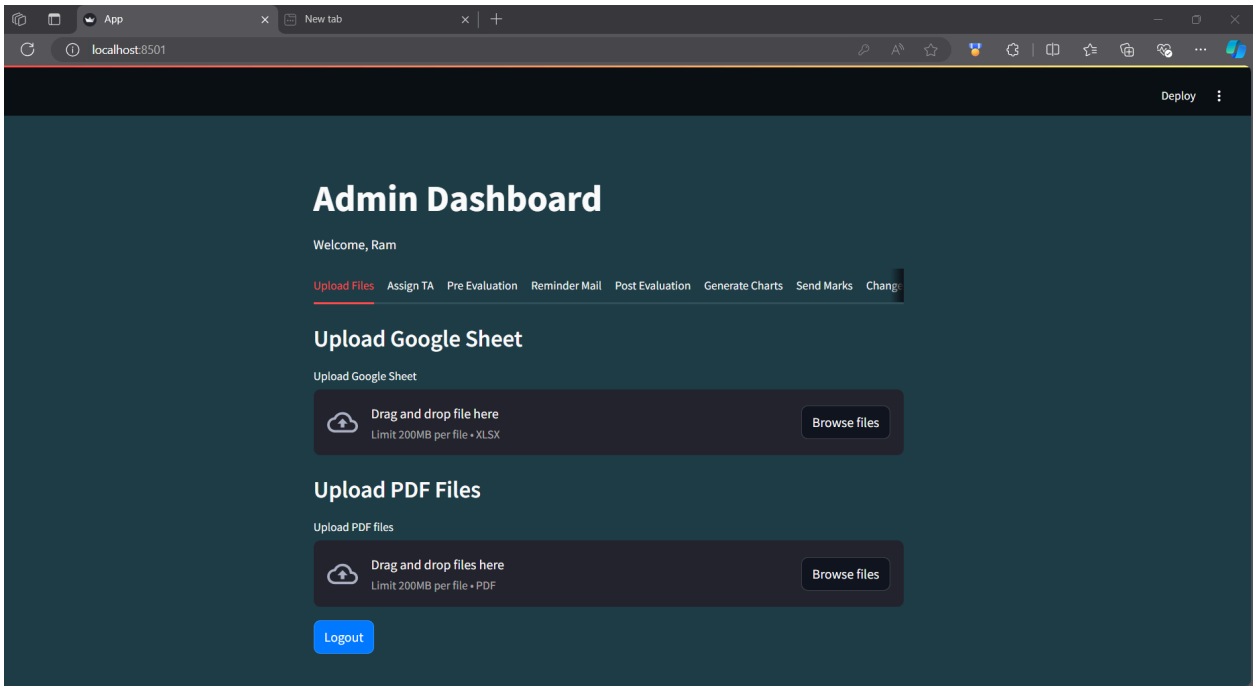
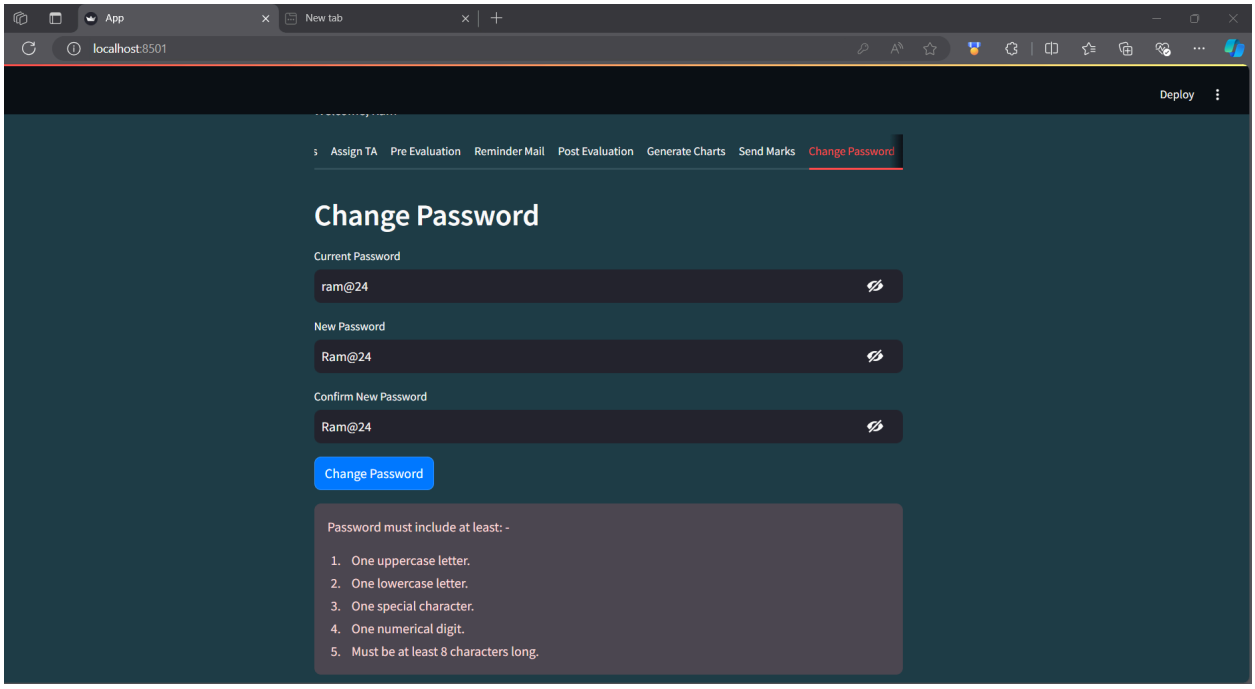
Peer Evaluation System UI/UX

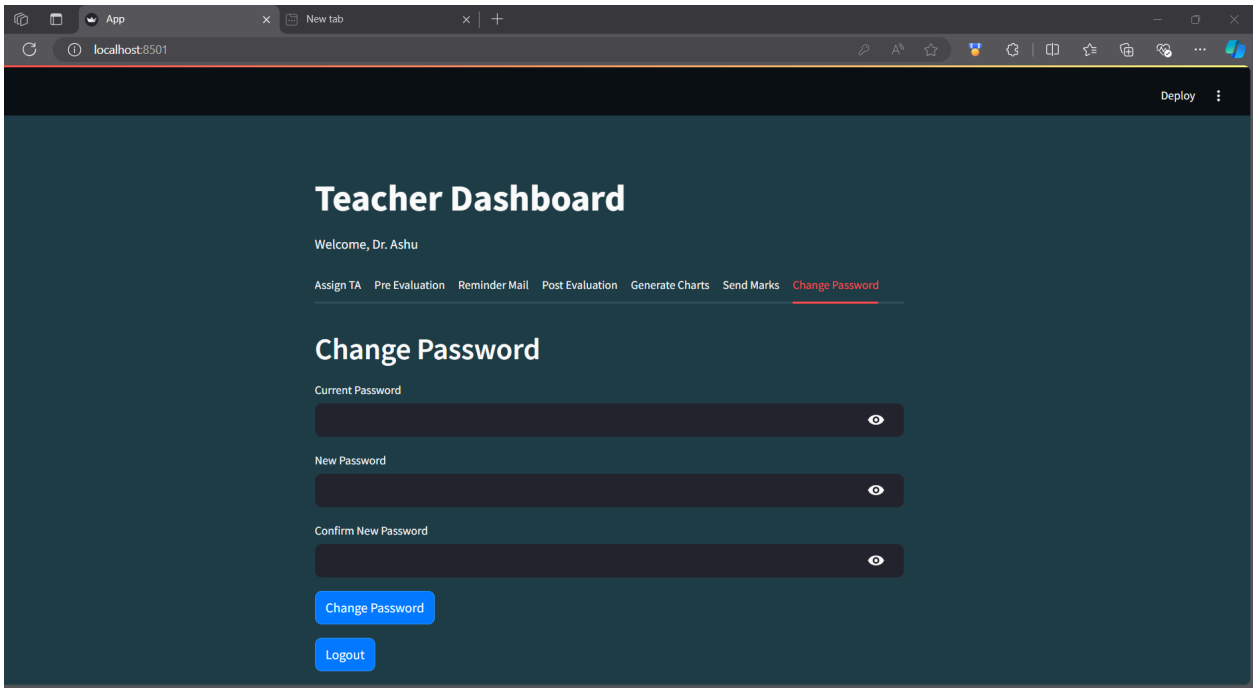
Screenshots of the UI/UX design: -

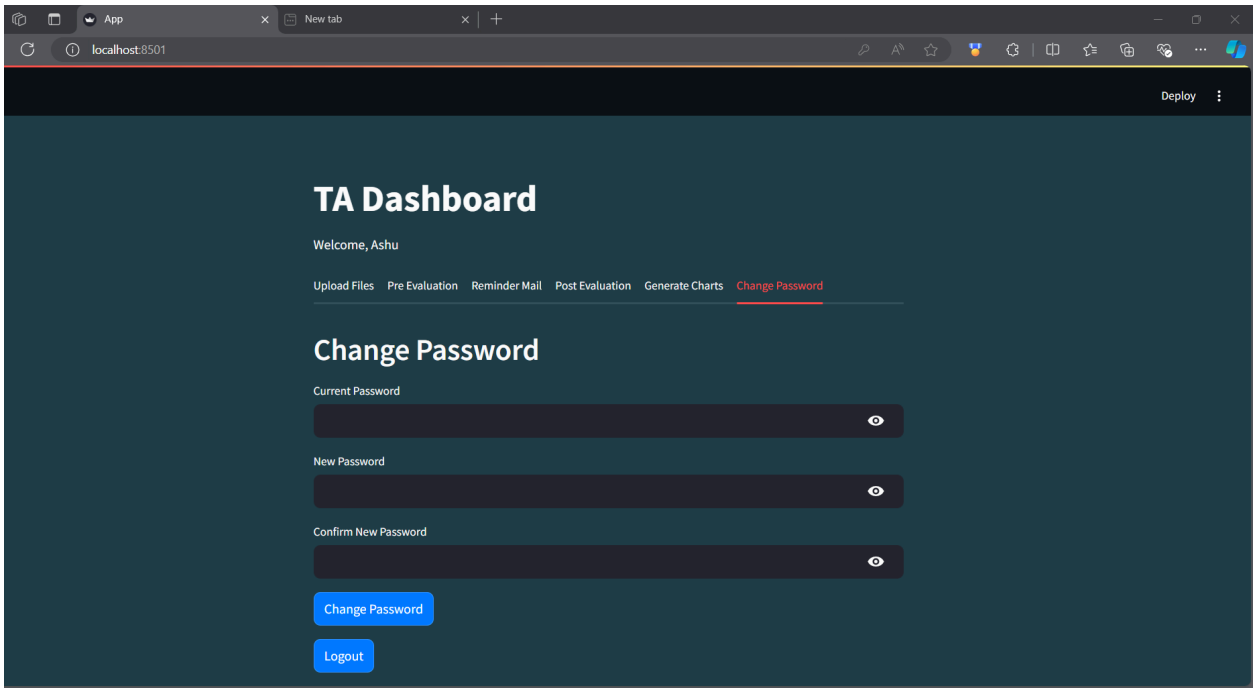
- The changes from the today's code are reflected below: -

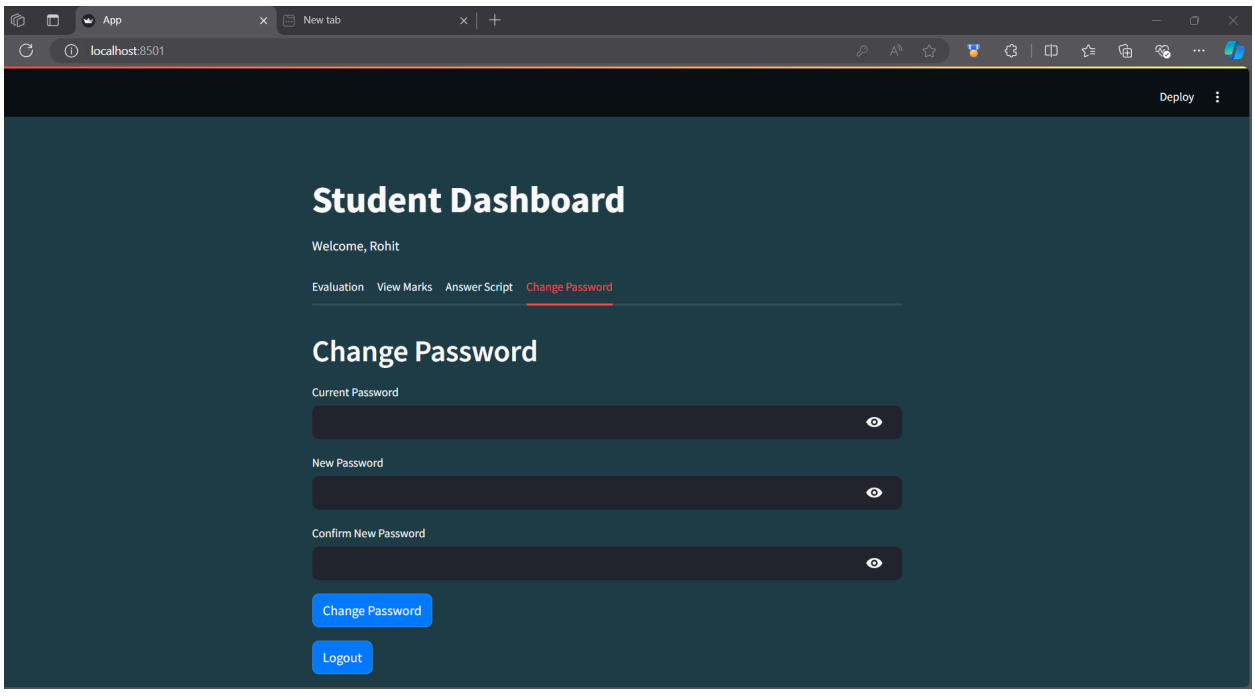





App
New tab
localhost:8501
Deploy
Assign TA
Pre Evaluation
Reminder Mail
Post Evaluation
Generate Charts
Send Marks
Change Password
Change Password
Current Password
ram@24
New Password
Ram@24
Confirm New Password
Ram@24
Change Password
Password must include at least: -
1. One uppercase letter.
2. One lowercase letter.
3. One special character.
4. One numerical digit.
5. Must be at least 8 characters long.


App
New tab
localhost:8501
Deploy
Teacher Dashboard
Welcome, Dr. Ashu
Assign TA
Pre Evaluation
Reminder Mail
Post Evaluation
Generate Charts
Send Marks
Change Password
Change Password
Current Password
New Password
Confirm New Password
Change Password
Logout


App
New tab
localhost:8501
Deploy
TA Dashboard
Welcome, Ashu
Upload Files
Pre Evaluation
Reminder Mail
Post Evaluation
Generate Charts
Change Password
Change Password
Current Password
New Password
Confirm New Password
Change Password
Logout


App
New tab
localhost:8501
Deploy
Student Dashboard
Welcome, Rohit
Evaluation
View Marks
Answer Script
Change Password
Change Password
Current Password
New Password
Confirm New Password
Change Password
Logout

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 = "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

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):
    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.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["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"  # 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")
                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:
                    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. **Google Colab: -**

```
import os
import cv2
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

# Define image dimensions
IMG_WIDTH = 128
```

```
IMG_HEIGHT = 64

# Load images from the dataset directory
def load_images(folder):
    images = []
    labels = []

    for label, subfolder in enumerate(['real', 'forged']):
        folder_path = os.path.join(folder, subfolder)
        for filename in os.listdir(folder_path):
            img_path = os.path.join(folder_path, filename)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
            img = img / 255.0  # Normalize pixel values to [0,1]
            images.append(img)
            labels.append(label)

    return np.array(images), np.array(labels)

# Load dataset
dataset_folder = 'path_to_dataset'
X, y = load_images(dataset_folder)
X = X.reshape(-1, IMG_WIDTH, IMG_HEIGHT, 1)  # Reshape for CNN input

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# CNN model
model = Sequential()

# Convolutional layer 1
model.add(Conv2D(32, (3, 3), input_shape=(IMG_WIDTH, IMG_HEIGHT, 1),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Convolutional layer 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten layer
```

```
model.add(Flatten())

# Dense layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

# Output layer
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False
)

# Fit the model
model.fit(datagen.flow(X_train, y_train, batch_size=32), validation_data=(X_test,
y_test), epochs=20)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')

# Save the model
model.save('signature_recognition_model.h5')

from keras.models import load_model

# Load the pre-trained model
model = load_model('signature_recognition_model.h5')

def predict_signature(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
    img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))
    img = img / 255.0  # Normalize
    img = img.reshape(1, IMG_WIDTH, IMG_HEIGHT, 1)  # Reshape for model input

    prediction = model.predict(img)[0][0]

    if prediction > 0.5:
        print("Signature is real.")
    else:
        print("Signature is forged.")

# Predict on a new signature
predict_signature('path_to_new_signature_image.jpg')
```