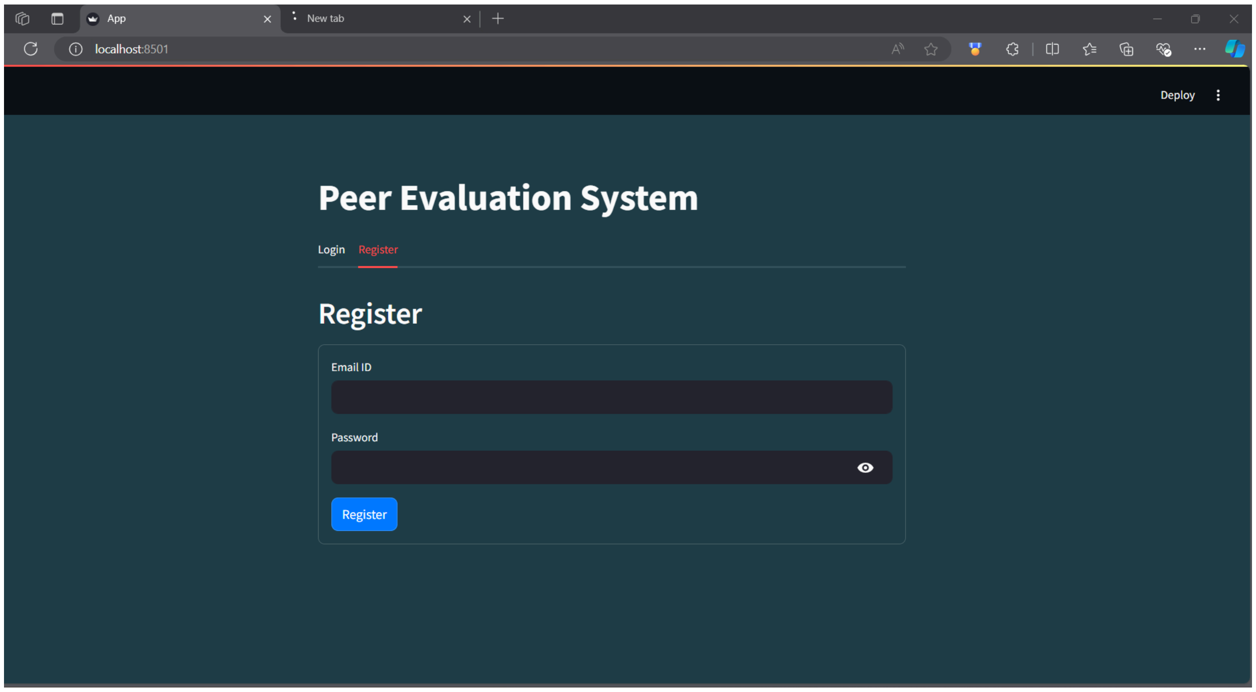
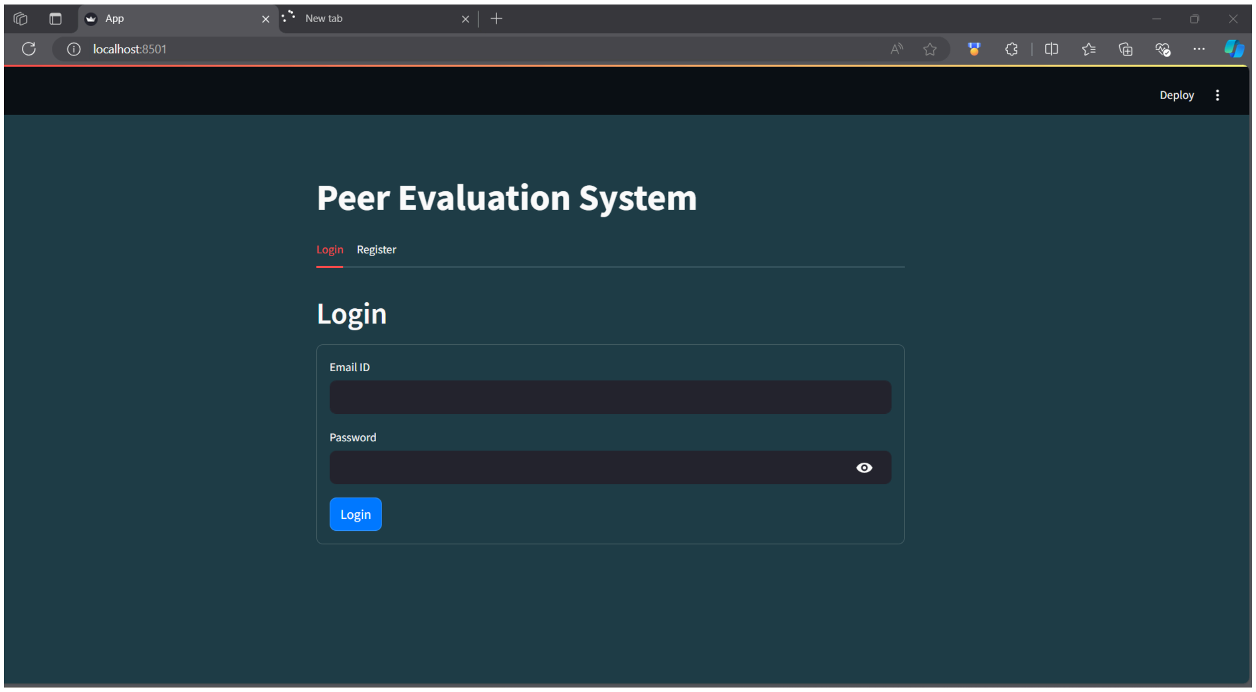
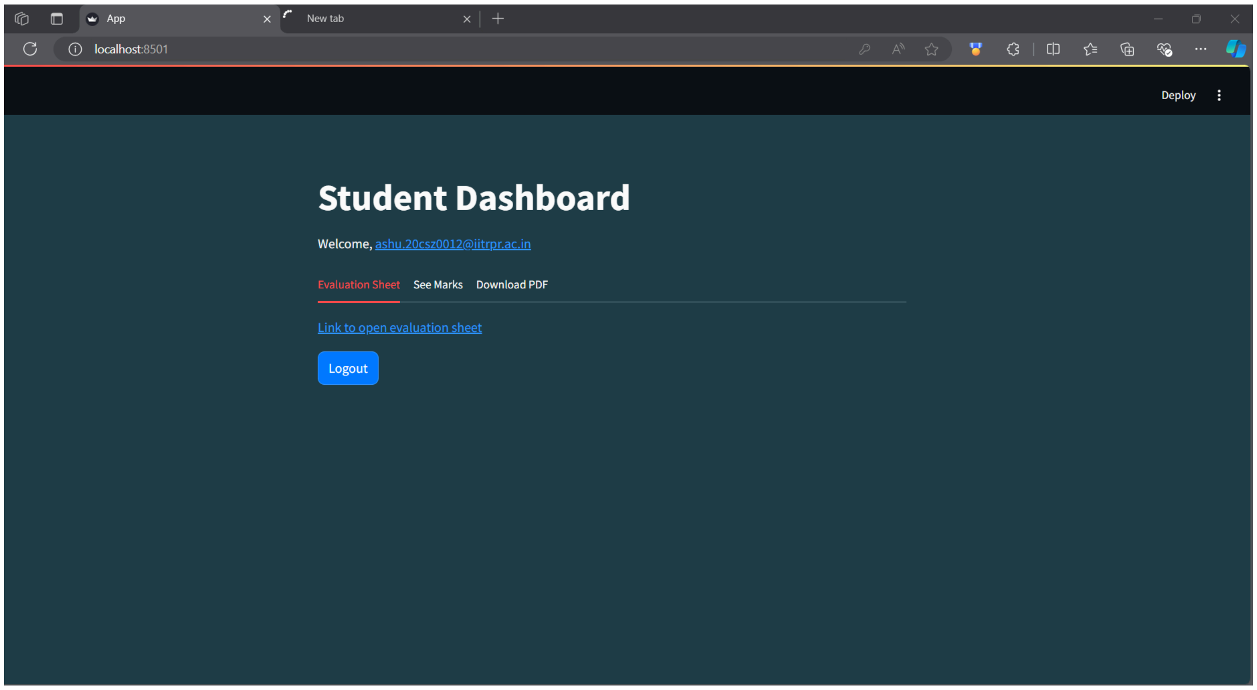
Peer Evaluation System UI/UX

Sample Screenshots of the UI/UX design: -

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

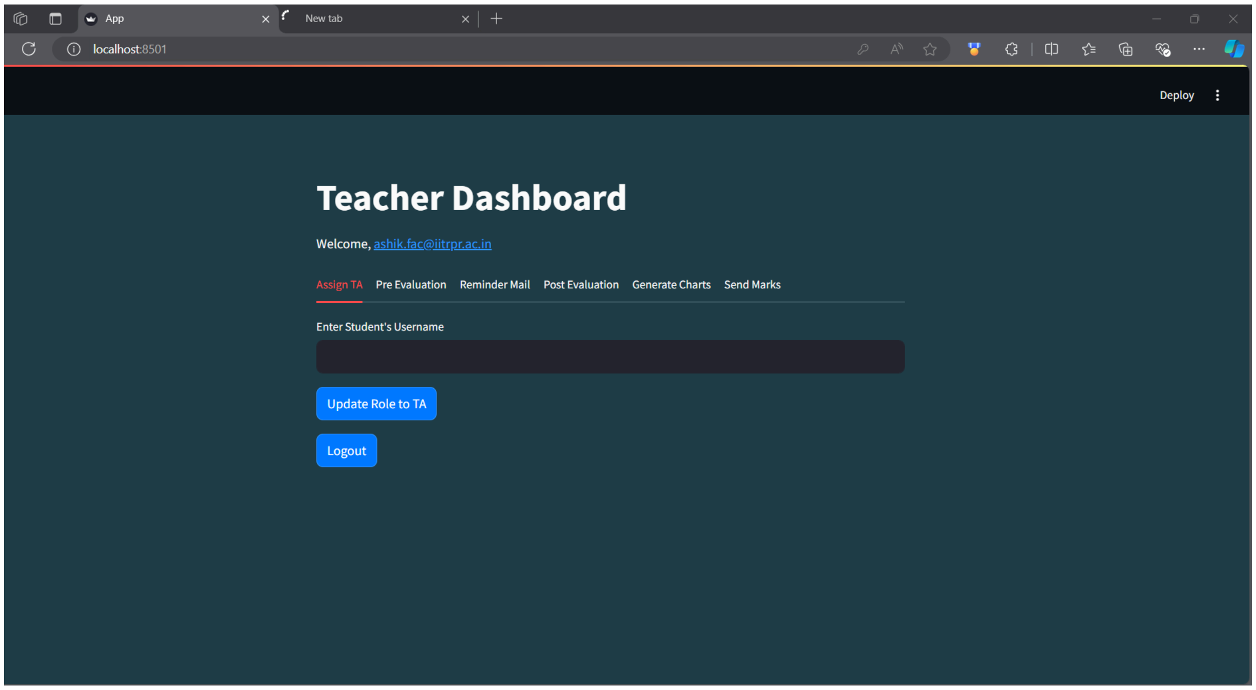






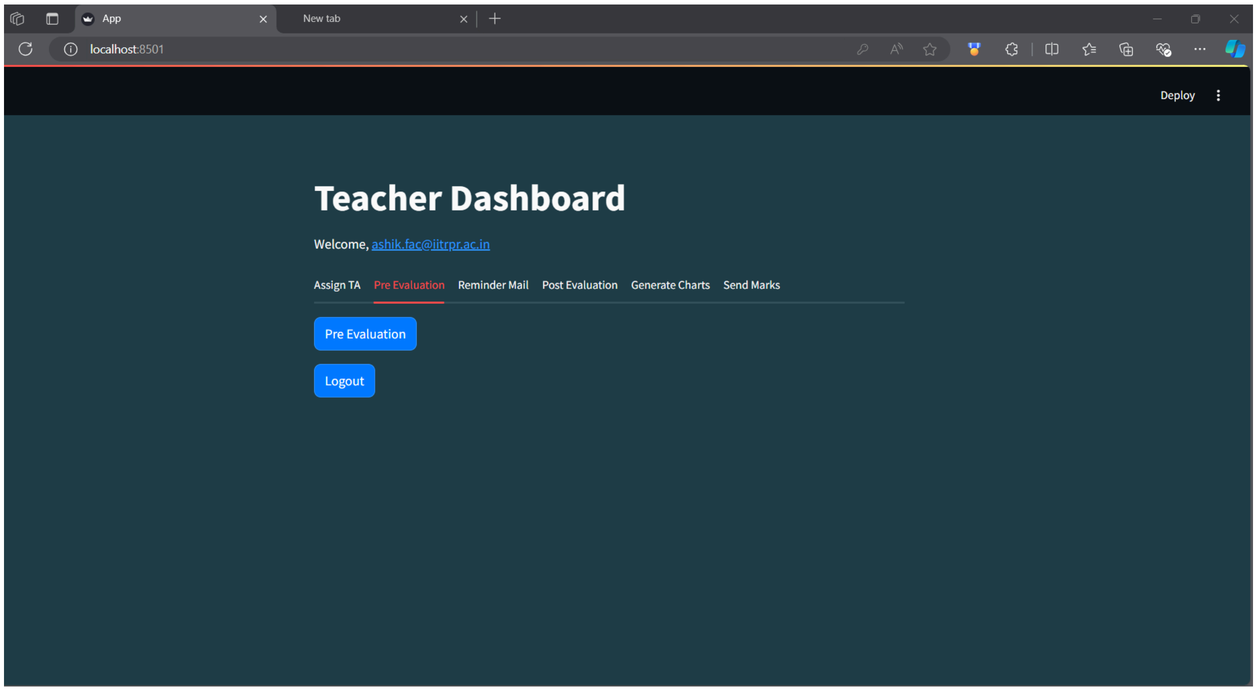
Student Dashboard

Welcome, ashu.20csz0012@iitrpr.ac.in



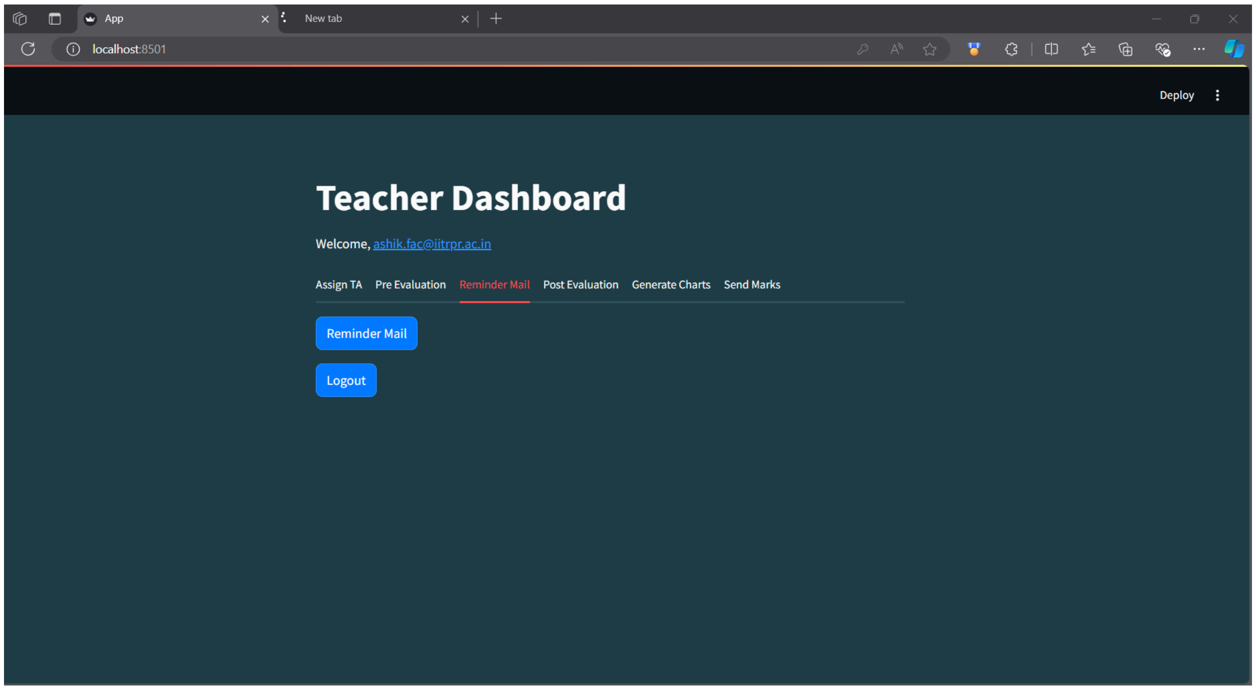
Teacher Dashboard

Welcome, ashik.fac@iitrpr.ac.in



Teacher Dashboard

Welcome, ashik.fac@iitrpr.ac.in



Teacher Dashboard

Welcome, ashik.fac@iitrpr.ac.in

Code: -

1. **Python: -**

```
import io
import re
import time
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"

  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 and user['password'] == password:
        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
    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"


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)}")


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

def teacher_dashboard():
    st.title("Teacher Dashboard")
    st.write(f"Welcome, {st.session_state['username']}")

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

    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}'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")
```

```
# 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.")


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

  # 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 Sheets")
  sheet_files = st.file_uploader("Upload Google Sheets", type=["xlsx"],
accept_multiple_files=True,
                               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)


# 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['Spreadsheet Link']  #
Returning the Average Mark's and Unique id

  return None, None, 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 student_dashboard():
    st.title("Student Dashboard")
    st.write(f"Welcome, {st.session_state['username']}")

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

    # Creating tabs
    tab0, tab1, tab2 = st.tabs(["Evaluation Sheet", "See Marks", "Download PDF"])

    # Tab for opening the peer evaluation spreadsheet
    with tab0:
        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 tab1:
        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 tab2:
        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.")




# 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. **Python Colab Code: -**

```
!apt-get install -y poppler-utils
!apt-get install -y poppler-utils tesseract-ocr
!pip install pdf2image pytesseract pillow


from google.colab import drive
from pdf2image import convert_from_path
from PIL import Image, ImageDraw
import pytesseract
import os
import re

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))

def name_extraction(folder_path, pdf_path):
  try:
    images = convert_from_path(pdf_path)
    print(f"Successfully converted PDF to images. Number of pages: {len(images)}")
  except Exception as e:
    print(f"Error converting PDF to images: {e}")
  if not images:
    raise Exception("Failed to convert PDF to images")

  image_path = 'page_1.jpg'
  images[0].save(image_path, 'JPEG')
  print(f"Saved first page as image: {image_path}")

  image = Image.open(image_path)

  crop_width = int(image.width * 0.2)
  crop_height = int(image.height * 0.1)

  cropped_image = crop_top_left(image, crop_width, crop_height)

  cropped_image_path = 'cropped_page_1.jpg'
  cropped_image.save(cropped_image_path)
  print(f"Saved cropped image: {cropped_image_path}")

  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 )

  print("Extracted text from top left corner:", text)
  new_pdf_filename = "{0}.pdf".format(text)
  print("Latest name", new_pdf_filename)
  new_pdf_path = os.path.join(folder_path, new_pdf_filename)

  try:
    os.rename(pdf_path, new_pdf_path)
    #print(f"Renamed file from '{pdf_filename}' to '{new_pdf_filename}'")
  except FileNotFoundError:
    print(f"File not found: {pdf_path}")
  except Exception as e:
    print(f"Error renaming file: {e}")


drive.mount('/content/drive')
```

```
folder_path = '/content/drive/My Drive/Exam Upload/Source Folder/'

pdf_files = [f for f in os.listdir(folder_path) if f.endswith('.pdf')]

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

for i in range(len(pdf_files)):
  pdf_path = os.path.join(folder_path, pdf_files[i])
  print(f"Selected PDF file: {pdf_path}")
  name_extraction(folder_path, pdf_path)
```

3. **Appscript: -**

```
function evalMarksInSheets() {

  var mainSheetName = "PeerEval";

  var mainSheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName(mainSheetName);

  var headers = mainSheet.getRange(1, 1, 1, mainSheet.getLastColumn()).getValues()[0];

  var linksColIndex = headers.indexOf("Spreadsheet Link");
  if (linksColIndex === -1) {
    Logger.log('Spreadsheet Link column not found.');
    return;
  }
  linksColIndex += 1;

  var evaluationColIndex = headers.indexOf("Evaluation");

  if (evaluationColIndex === -1) {
    evaluationColIndex = headers.length;
    mainSheet.getRange(1, evaluationColIndex + 1).setValue("Evaluation");
  } /*else {
    evaluationColIndex += 1;
  }*/
  var data = mainSheet.getRange(2, linksColIndex, mainSheet.getLastRow() - 1,
1).getValues();

  for (var i = 0; i < data.length; i++) {
    var sheetLink = data[i][0];

    if (sheetLink) {
      try {
        var spreadsheet = SpreadsheetApp.openByUrl(sheetLink);

        var sheetToCheck = spreadsheet.getActiveSheet();
        var columnToCheck = 2;
        var range = sheetToCheck.getRange(1, columnToCheck,
sheetToCheck.getLastRow()).getValues();

        var marksFound = false;
        for (var j = 0; j < range.length; j++) {
          if (typeof range[j][0] === 'number' && !isNaN(range[j][0])) {
            marksFound = true;
            break;
          }
        }

        if (marksFound) {
          mainSheet.getRange(i + 2, evaluationColIndex+1).setValue("Done");
        } else {
          mainSheet.getRange(i + 2, evaluationColIndex+1).setValue("Not done");
        }

      } catch (e) {
        Logger.log("Error opening sheet: " + sheetLink);
        mainSheet.getRange(i + 2, evaluationColIndex).setValue("Error Accessing Sheet");
      }
    } else {
      mainSheet.getRange(i + 2, evaluationColIndex).setValue("No Link Provided");
    }
  }
}
```

```
function emailPeerPendingEval() {

 var mainSheetName = "PeerEval";

 var mainSheet =
SpreadsheetApp.getActiveSpreadsheet().getSheetByName(mainSheetName);

 var headers = mainSheet.getRange(1, 1, 1, mainSheet.getLastColumn()).getValues()[0];

 var nameColIndex = headers.indexOf("Name");
 var emailColIndex = headers.indexOf("EMail ID");
 var evaluationColIndex = headers.indexOf("Evaluation");

 if (nameColIndex === -1 || emailColIndex === -1 || evaluationColIndex === -1) {
  Logger.log('Required columns not found.');
  return;
 }

 nameColIndex += 1;
 emailColIndex += 1;
 evaluationColIndex += 1;

 var data = mainSheet.getRange(2, 1, mainSheet.getLastRow() - 1,
mainSheet.getLastColumn()).getValues();

 for (var i = 0; i < data.length; i++) {
  var name = data[i][nameColIndex - 1];
  var email = data[i][emailColIndex - 1];
  var evaluationStatus = data[i][evaluationColIndex - 1];

  if (evaluationStatus === "Not done" && email) {

   var subject = "Gentle Reminder! For Pending Evaluation";

   var body = "Dear " + name + ",<br><br>" +
        "Our records indicate that you have not yet completed your evaluation. " +
        "Please complete it as soon as possible.<br><br>" +
        "Best regards,<br>CSE, IIT Ropar";
```

```
      MailApp.sendEmail({
        to: email,
        subject: subject,
        htmlBody: body
      });

      Logger.log('Email sent to: ' + email);
    }
  }
}
```