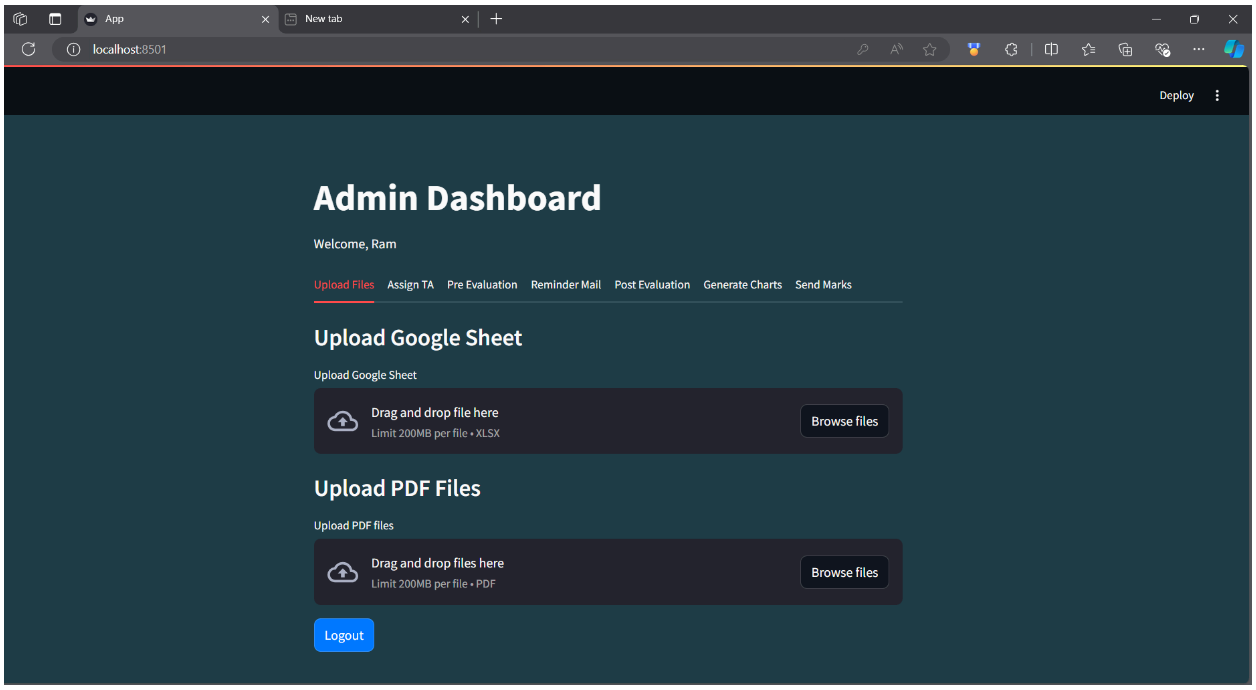
Peer Evaluation System UI/UX

Screenshots of the UI/UX design: -

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



Admin Dashboard

Welcome, Ram

Upload Files | Assign TA | Pre Evaluation | Reminder Mail | Post Evaluation | Generate Charts | Send Marks

Upload Google Sheet

Upload Google Sheet

Drag and drop file here
Limit 200MB per file • XLSX

Browse files

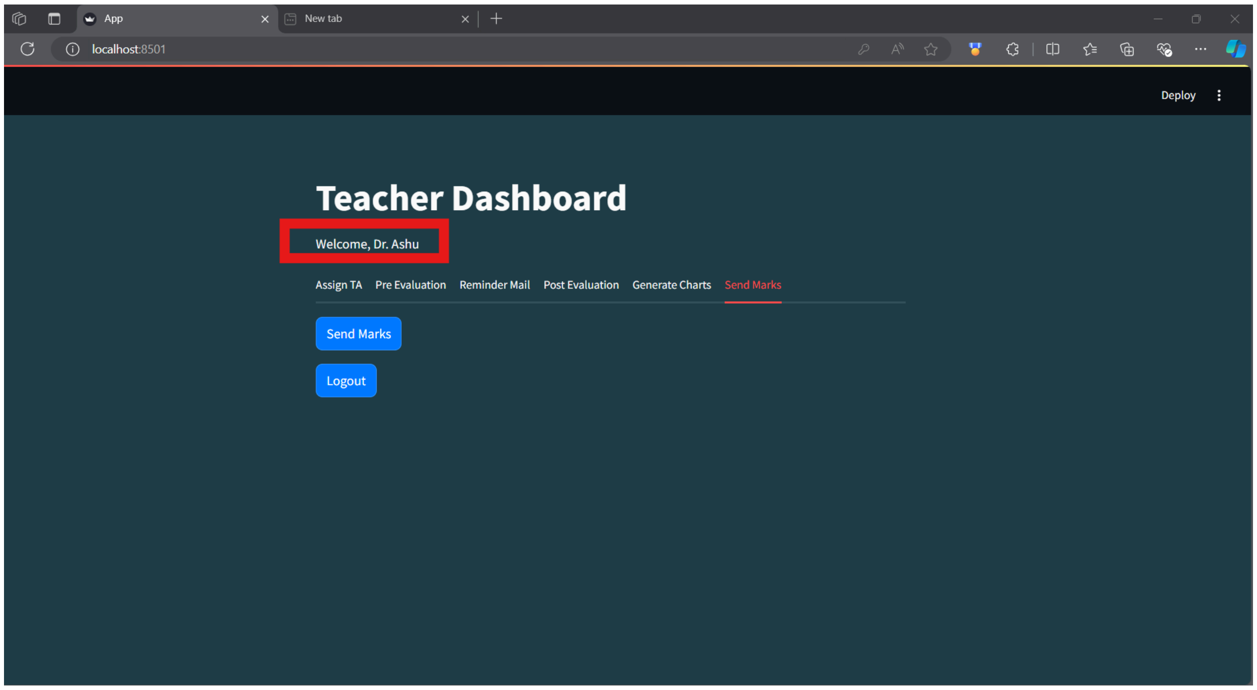
Upload PDF Files

Upload PDF files

Drag and drop files here
Limit 200MB per file • PDF

Browse files

Logout



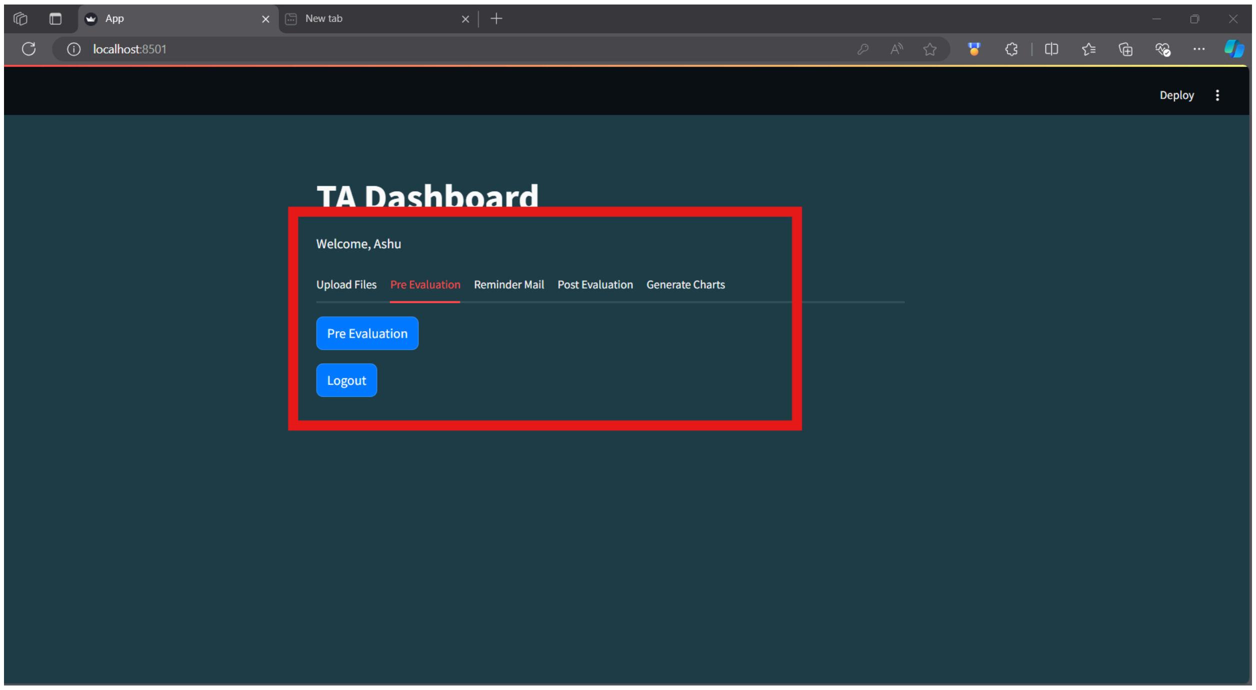
Teacher Dashboard

Welcome, Dr. Ashu

Assign TA | Pre Evaluation | Reminder Mail | Post Evaluation | Generate Charts | Send Marks

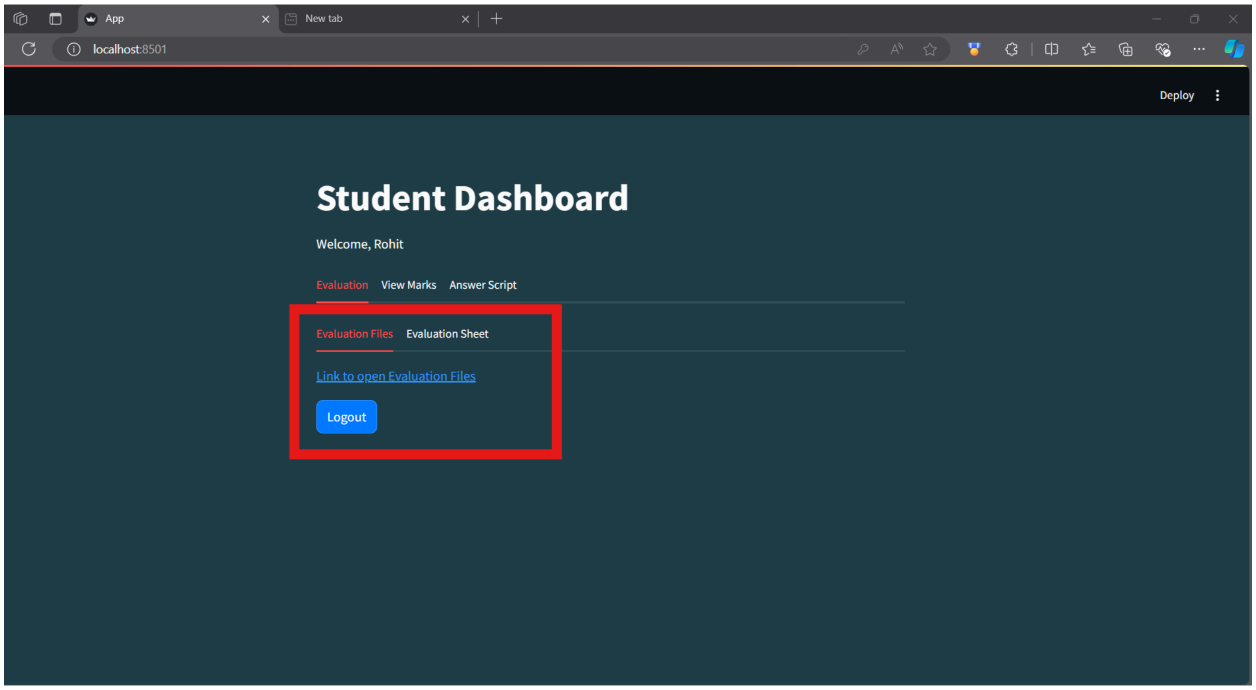
Send Marks

Logout



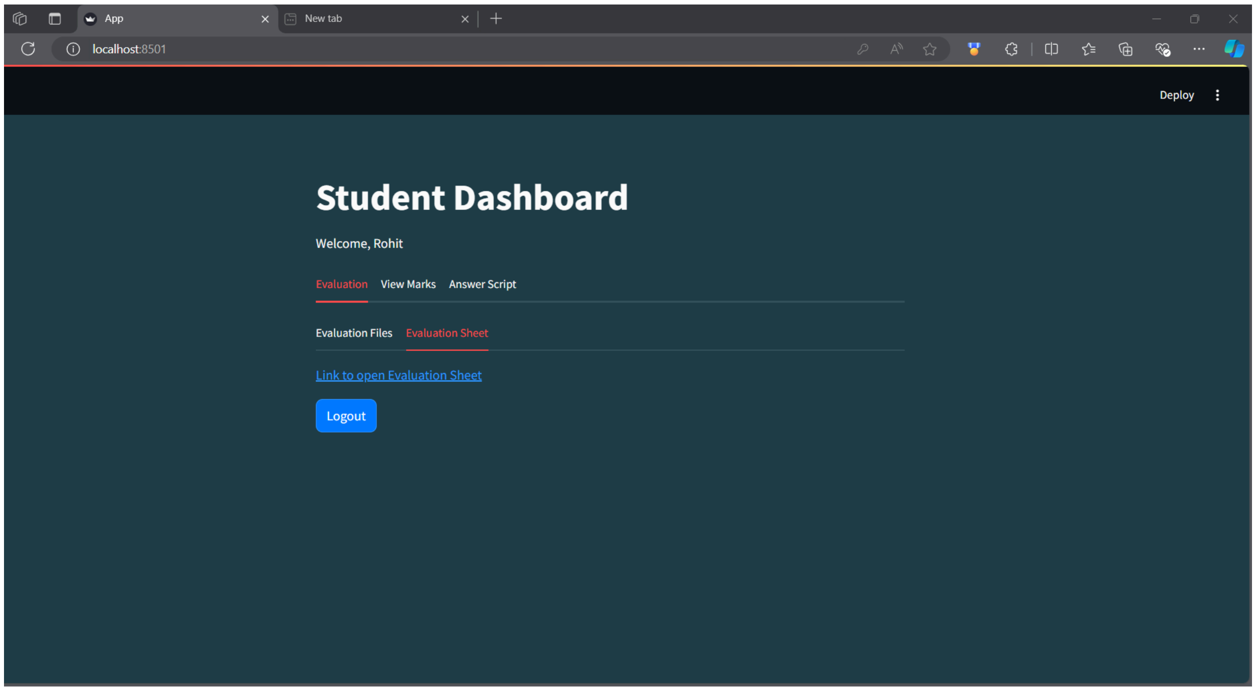
TA Dashboard

Welcome, Ashu



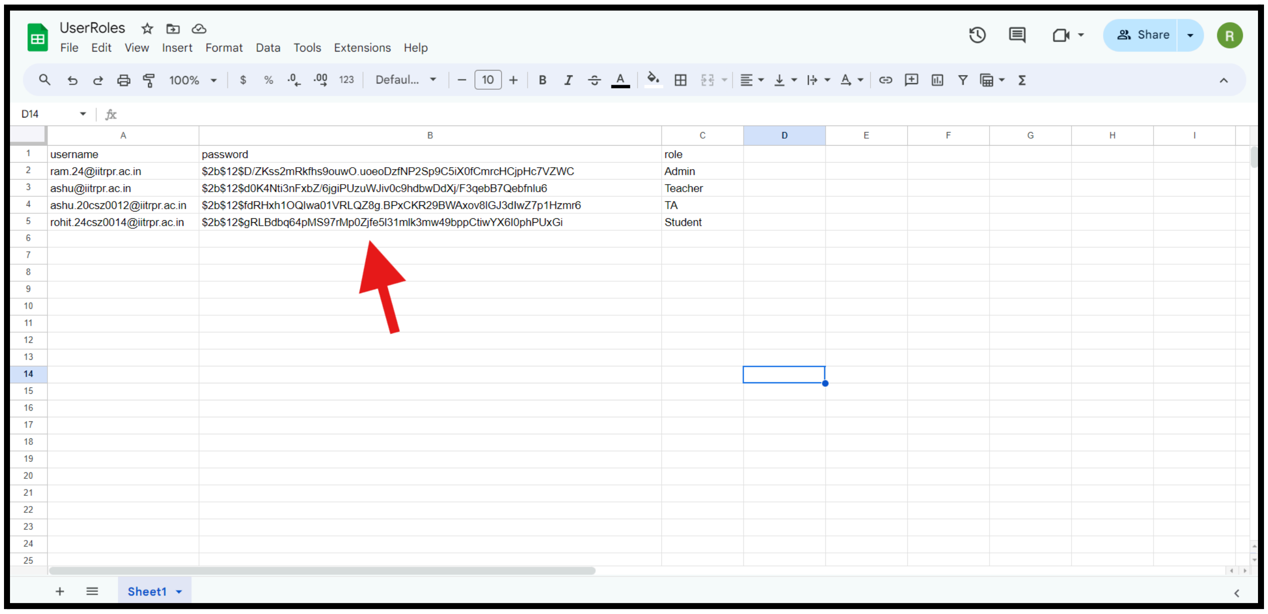
Student Dashboard

Welcome, Rohit



Student Dashboard

Welcome, Rohit



username	password	role
ram.24@iitrpr.ac.in	$2b$12$D/ZKss2mRkfhs9ouwO.uoeoDzfNP2Sp9C5iX0fCmrcHCjpHc7VZWC	Admin
ashu@iitrpr.ac.in	$2b$12$d0K4Nti3nFxbZ/6jgiPUzuWJiv0c9hdbwDdXj/F3qebB7Qebfnlu6	Teacher
ashu.20csz0012@iitrpr.ac.in	$2b$12$fdRHxh1OQlwa01VRLQZ8g.BPxCKR29BWAxov8lGJ3dlwZ7p1Hzmr6	TA
rohit.24csz0014@iitrpr.ac.in	$2b$12$gRLBdbq64pMS97rMp0Zjfe5I31mlk3mw49bppCtiwYX6I0phPUxGi	Student

Encrypted Passwords are reflected in the above image.

Code: -

1. **Python: -**

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

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


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


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


# Fetch users from Google Sheets
def get_users_from_sheets():
  sheet = connect_to_google_sheets()
  records = sheet.get_all_records()
  return records

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

  # Check if the email contains numeric values (assumed to be student)
  if re.search(r'\d', username):
     role = "Student"
  else:
     role = "Teacher"

  # Hash the password before saving
  hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

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

  #new_user = [username, password, role]
  sheet.append_row(new_user)
  return role


# Update role from Student to TA (only for Teachers)
def update_role_to_ta(username):
  sheet = connect_to_google_sheets()
  records = sheet.get_all_records()
  for i, user in enumerate(records, start=2):  # start=2 to account for 1-based index in
Google Sheets
     if user['username'] == username and user['role'] == 'Student':
        sheet.update_cell(i, 3, 'TA')  # Assuming role is in column 3
        return True
  return False
```

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


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


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 = st.tabs(
      ["Upload Files", "Assign TA", "Pre Evaluation", "Reminder Mail", "Post Evaluation",
"Generate Charts",
       "Send Marks"])

  # 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")


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 = st.tabs(["Assign TA", "Pre Evaluation", "Reminder
Mail", "Post Evaluation", "Generate Charts", "Send Marks"])

    # 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")




# 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 = st.tabs(
     ["Upload Files", "Pre Evaluation", "Reminder Mail", "Post Evaluation", "Generate
Charts"])

  # 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")



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']}")

    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!")

    # Creating tabs
    tab0, tab1, tab2 = st.tabs(["Evaluation", "View Marks", "Answer Script"])

    # Tab for opening the peer evaluation spreadsheet
    with tab0:
        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 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. **Appscript: -**

```
function createStudentWorkbooks(source_folder, target_folder, students_per_batch,
num_Questions) {
 /**
  * The code iterates through the Unique IDs and makes workbooks for them to
  *  evaluate the peers.
  *
  * Once it runs, it makes the workbooks in the target folder identified by
  *  @target_folder in the global variables.
  *
  * It puts the links of the workbooks for the respective workbook in the last
  *  column of the sheet identifies by the @sheetName variable
  *
  * It then gives access only to the respective student and nobody else.
  *
  * To be a part of the MAIN Driver Code
  */

 var mainWorkbook = SpreadsheetApp.getActiveSpreadsheet();
 var sheet = mainWorkbook.getSheetByName(sheetName);

 if (!sheet) {
  Logger.log('Sheet not found.');
  return;
 }
```

```
var data = sheet.getDataRange().getValues();
var folder = DriveApp.getFolderById(target_folder);

sheet.getRange(1, data[0].length + 1).setValue("Spreadsheet Link");


for (var i = 1; i < data.length; i++) { // Start from 1 to skip the header row
  var name = data[i][0]; // Column A
  var email = data[i][1]; // Column B
  var uniqueID = data[i][2]; // Column C
  var folderName = data[i][4]; // Column E

  Logger.log(uniqueID);
  // Create a new workbook for the student
  var studentWorkbook = SpreadsheetApp.create(name);
  var studentSheet = studentWorkbook.getActiveSheet();

  // Write the name and unique ID
  studentSheet.getRange('A1').setValue(name);
  studentSheet.getRange('A2').setValue(uniqueID);

  // Write the question numbers
  for (var q = 1; q <= num_Questions; q++) {
    studentSheet.getRange(2, q + 1).setValue('Question ' + q);
  }

  // Get all unique IDs in column D that have the specified folder name in column E
  var uniqueIDsWithFolder = [];
  for (var j = 1; j < data.length; j++) {
    if (data[j][3] == folderName) {
      uniqueIDsWithFolder.push(data[j][2]); // Column C
    }
  }

  // Write these unique IDs in column A of the student sheet
  for (var k = 0; k < uniqueIDsWithFolder.length; k++) {
    studentSheet.getRange(k + 3, 1).setValue(uniqueIDsWithFolder[k]);
  }

  // Protect the student sheet
```

```
    var protection = studentSheet.protect().setDescription('Protected Sheet');
    protection.removeEditors(protection.getEditors()); // Remove all editors
    protection.addEditor(email);

    // Protect specific ranges (Column A and rows 1 and 2)
    protectRange(studentSheet.getRange('A:A'));
    protectRange(studentSheet.getRange('1:1'));
    protectRange(studentSheet.getRange('2:2'));
    var lastRow = studentSheet.getLastRow();
    var nextRow = lastRow + 1;
    protectRange(studentSheet.getRange(nextRow + ':' + nextRow));

    var lastColumn = studentSheet.getLastColumn();
    var nextColumn = lastColumn + 1;
    protectRange(studentSheet.getRange(nextColumn + ':' + nextColumn));


    // Move the student workbook to the specified folder
    var studentFile = DriveApp.getFileById(studentWorkbook.getId());
    folder.addFile(studentFile);
    DriveApp.getRootFolder().removeFile(studentFile);

    // Generate the edit link for the student workbook
    var editLink = studentWorkbook.getUrl();

    // Add the edit link to the main sheet in the corresponding row and last column
    sheet.getRange(i + 1, data[0].length + 1).setValue(editLink); // Assuming data[0]
contains headers

    // Share the student workbook with the student email
    studentWorkbook.addEditor(email);
  }

  Logger.log('Student workbooks created and permissions assigned.');

}

function protectRange(range) {
  /**
   * This function is made to protect the cells
   * Used by createStudentWorkbooks() to protect
   *  the sheets before sending them to the students
   *
   * @param {string} range - The range of the cells that are to be protected
   * Prevents the editing of the cells.
   */

  var protection = range.protect().setDescription('Protected Range');
  protection.removeEditors(protection.getEditors()); // Remove all editors
  protection.setWarningOnly(false); // Prevent editing
}


/**
 * Send Mail to the students about the folder access and the spreadsheet access
 * This is for them to evaluate their peers and give them the marks
 */
function sendEmails(source_folder, target_folder, students_per_batch, num_Questions) {
  /**
   * This code iterates through to the last column in the sheet and checks the Email-IDs
   * Then it sends the links to them individually.
   *
   * Change the message before final implementation
   *
   * To be part of the MAIN Driver Code
   */

  var workbook = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = workbook.getSheetByName(sheetName);

  if (!sheet) {
    Logger.log('Sheet not found.');
    return;
  }

  var data = sheet.getDataRange().getValues();

  for (var i = 1; i < data.length; i++) { // Start from 1 to skip the header row
    var name = data[i][0];
    var email = data[i][1]; // Column B
    var editorLink1 = data[i][5]; // Column F (assuming it contains the first editor link)
    var lastColumn = data[0].length - 1; // Exclude the last column which may be a label

    // Find the last column with data in the row
    while (lastColumn > 0 && !data[i][lastColumn]) {
      lastColumn--;
    }

    var editorLink2 = data[i][lastColumn]; // Last column with data

    var subject = 'Peer Evaluation Link';
    var message = 'Dear ' + name + ',\n\n' +
      'Find below the link for the folder of answer scripts (view only) and the Evaluation
Sheet (edit access) for the same:\n\n' +
      'Folder Link: ' + editorLink1 + '\n' +
      'Evaluation Sheet Link: ' + editorLink2 + '\n\n' +
      'Kind Regards,\nStepWISE\n\nNOTE: This is a test mail, kindly check link activity
and report anomalies.';

    // Send email
    MailApp.sendEmail(email, subject, message);

    Logger.log(editorLink2);
    Logger.log('Email sent to ' + email);
  }

  Logger.log('All emails sent.');
}
```