```python
import streamlit as st
from streamlit_option_menu import option_menu
import pymongo
from pymongo import MongoClient  # Added this import
import psycopg2
import pandas as pd
from googleapiclient.discovery import build




#API key connection
def Api_connect():
    Api_Id="AIzaSyCRlE3-Ly-iitIQrSWM_M3zF-E6BNR9IXQ"

    api_service_name = "youtube"
    api_version = "v3"
    youtube = build(api_service_name,api_version,developerKey=Api_Id)
    return youtube

youtube=Api_connect()


#CHANNEL INFORMATION
def get_channel_info(channel_id):

    request = youtube.channels().list(
                part = "snippet,contentDetails,Statistics",
                id = channel_id)

    response1=request.execute()

    for i in range(0,len(response1["items"])):
        data = dict(
                    Channel_Name = response1["items"][i]["snippet"]["title"],
                    Channel_Id = response1["items"][i]["id"],
                    Subscription_Count= response1["items"][i]["statistics"]
["subscriberCount"],
                    Views = response1["items"][i]["statistics"]["viewCount"],
                    Total_Videos = response1["items"][i]["statistics"]
["videoCount"],
                    Channel_Description = response1["items"][i]["snippet"]
["description"],
                    Playlist_Id = response1["items"][i]["contentDetails"]
["relatedPlaylists"]["uploads"],
                    )
        return data


#PLAYLISTS INFORMATION
def get_playlist_info(channel_id):
    All_data = []
    next_page_token = None
    next_page = True
    while next_page:

        request = youtube.playlists().list(
            part="snippet,contentDetails",
            channelId=channel_id,
            maxResults=50,
```

```
            pageToken=next_page_token
            )
        response = request.execute()

        for item in response['items']:
            data={'PlaylistId':item['id'],
                    'Title':item['snippet']['title'],
                    'ChannelId':item['snippet']['channelId'],
                    'ChannelName':item['snippet']['channelTitle'],
                    'PublishedAt':item['snippet']['publishedAt'],
                    'VideoCount':item['contentDetails']['itemCount']}
            All_data.append(data)
        next_page_token = response.get('nextPageToken')
        if next_page_token is None:
            next_page=False
    return All_data

#CHANNEL VIDEO INFORMATION
def get_channel_videos(channel_id):
    video_ids = []
    # get Uploads playlist id
    res = youtube.channels().list(id=channel_id,
                                  part='contentDetails').execute()
    playlist_id = res['items'][0]['contentDetails']['relatedPlaylists']['uploads']
    next_page_token = None

    while True:
        res = youtube.playlistItems().list(
                                           part = 'snippet',
                                           playlistId = playlist_id,
                                           maxResults = 50,
                                           pageToken = next_page_token).execute()

        for i in range(len(res['items'])):
            video_ids.append(res['items'][i]['snippet']['resourceId']['videoId'])
        next_page_token = res.get('nextPageToken')

        if next_page_token is None:
            break
    return video_ids

#VIDEO_INFORMATION
def get_video_info(video_ids):

    video_data = []

    for video_id in video_ids:
        request = youtube.videos().list(
                    part="snippet,contentDetails,statistics",
                    id= video_id)
        response = request.execute()

        for item in response["items"]:
            data = dict(Channel_Name = item['snippet']['channelTitle'],
                        Channel_Id = item['snippet']['channelId'],
                        Video_Id = item['id'],
                        Title = item['snippet']['title'],
                        Tags = item['snippet'].get('tags'),
                        Thumbnail = item['snippet']['thumbnails']['default']['url'],
                        Description = item['snippet']['description'],
```

```
                        Published_Date = item['snippet']['publishedAt'],
                        Duration = item['contentDetails']['duration'],
                        Views = item['statistics']['viewCount'],
                        Likes = item['statistics'].get('likeCount'),
                        Comments = item['statistics'].get('commentCount'),
                        Favorite_Count = item['statistics']['favoriteCount'],
                        Definition = item['contentDetails']['definition'],
                        Caption_Status = item['contentDetails']['caption']
                        )
            video_data.append(data)
    return video_data
#COMMENT INFORMATION
def get_comment_info(video_ids):
    Comment_Information = []
    try:
            for video_id in video_ids:

                    request = youtube.commentThreads().list(
                            part = "snippet",
                            videoId = video_id,
                            maxResults = 50
                            )
                    response5 = request.execute()

                    for item in response5["items"]:
                            comment_information = dict(
                                    Comment_Id = item["snippet"]["topLevelComment"]
["id"],
                                    Video_Id = item["snippet"]["videoId"],
                                    Comment_Text = item["snippet"]
["topLevelComment"]["snippet"]["textOriginal"],
                                    Comment_Author = item["snippet"]
["topLevelComment"]["snippet"]["authorDisplayName"],
                                    Comment_Published = item["snippet"]
["topLevelComment"]["snippet"]["publishedAt"])

                            Comment_Information.append(comment_information)
    except:
            pass

            return Comment_Information


#MongoDB Connection
client =
pymongo.MongoClient("mongodb+srv://sabarishraja:youtube@youtube.u3ogmrx.mongodb.net/
?retryWrites=true&w=majority")
db = client["Youtube_data"]

# upload to MongoDB

def channel_details(channel_id):
    ch_details = get_channel_info(channel_id)
    pl_details = get_playlist_info(channel_id)
    vi_ids = get_channel_videos(channel_id)
    vi_details = get_video_info(vi_ids)
    com_details = get_comment_info(vi_ids)

    coll1 = db["channel_details"]
```

```
 coll1.insert_one({"channel_information":ch_details,"playlist_information":pl_details,"video_information":vi_details,
                     "comment_information":com_details})

    return "upload completed successfully"

#Table creation for channels,playlists, videos, comments
def channels_table():
    mydb = psycopg2.connect(host="localhost",
            user="postgres",
            password="postgresql",
            database= "Youtube_data",
            port = "5432"
            )
    cursor = mydb.cursor()

    drop_query = "drop table if exists channels"
    cursor.execute(drop_query)
    mydb.commit()

    try:
        create_query = '''CREATE TABLE IF NOT EXISTS channels(
                    Channel_Name VARCHAR(100),
                    Channel_Id VARCHAR(80) PRIMARY KEY,
                    Subscription_Count BIGINT,
                    Views BIGINT,
                    Total_Videos INT,
                    Channel_Description TEXT,
                    Playlist_Id VARCHAR(50))'''

        cursor.execute(create_query)
        mydb.commit()
    except:
        st.write("Channels Table alredy created")


    ch_list = []
    db = client["Youtube_data"]
    coll1 = db["channel_details"]

    for ch_data in coll1.find({}, {"_id": 0, "channel_information": 1}):
        ch_list.append(ch_data["channel_information"])

    df = pd.DataFrame(ch_list)

    for index, row in df.iterrows():
        insert_query = '''INSERT INTO channels(Channel_Name,
                                               Channel_Id,
                                               Subscription_Count,
                                               Views,
                                               Total_Videos,
                                               Channel_Description,
                                               Playlist_Id)
                          VALUES (%s, %s, %s, %s, %s, %s, %s)'''

        values = (
            row['Channel_Name'],
            row['Channel_Id'],
            row['Subscription_Count'],
```

```
            row['Views'],
            row['Total_Videos'],
            row['Channel_Description'],
            row['Playlist_Id']
        )
        try:
            cursor.execute(insert_query,values)
            mydb.commit()
        except:
            st.write("Channels values are already inserted")


def playlists_table():
    mydb = psycopg2.connect(host="localhost",
            user="postgres",
            password="postgresql",
            database= "Youtube_data",
            port = "5432"
            )
    cursor = mydb.cursor()
    drop_query = "drop table if exists playlists"
    cursor.execute(drop_query)
    mydb.commit()

    try:
        create_query = '''create table if not exists playlists(PlaylistId varchar(100) primary key,
                                                               Title varchar(80),
                                                               ChannelId varchar(100),
                                                               ChannelName varchar(100),
                                                               PublishedAt timestamp,
                                                               VideoCount int
                                                               )'''
        cursor.execute(create_query)
        mydb.commit()

    except:
        st.write("Playlists Table already created")

    db = client["Youtube_data"]
    coll1 =db["channel_details"]
    pl_list = []
    for pl_data in coll1.find({},{"_id":0,"playlist_information":1}):
        for i in range(len(pl_data["playlist_information"])):
                pl_list.append(pl_data["playlist_information"][i])
    df2 = pd.DataFrame(pl_list)



    for index,row in df2.iterrows():
        insert_query = '''INSERT into playlists(PlaylistId,
                                                Title,
                                                ChannelId,
                                                ChannelName,
                                                PublishedAt,
                                                VideoCount)
                                        VALUES(%s,%s,%s,%s,%s,%s)'''
```

```
        values =(
                row['PlaylistId'],
                row['Title'],
                row['ChannelId'],
                row['ChannelName'],
                row['PublishedAt'],
                row['VideoCount'])

        try:
            cursor.execute(insert_query,values)
            mydb.commit()
        except:
            st.write("Playlists values are already inserted")

def videos_table():

    mydb = psycopg2.connect(host="localhost",
            user="postgres",
            password="postgresql",
            database= "Youtube_data",
            port = "5432"
            )
    cursor = mydb.cursor()

    drop_query = "drop table if exists videos"
    cursor.execute(drop_query)
    mydb.commit()



    try:
        create_query = '''create table if not exists videos(
                        Channel_Name varchar(150),
                        Channel_Id varchar(100),
                        Video_Id varchar(50) primary key,
                        Title varchar(150),
                        Tags text,
                        Thumbnail varchar(225),
                        Description text,
                        Published_Date timestamp,
                        Duration interval,
                        Views bigint,
                        Likes bigint,
                        Comments int,
                        Favorite_Count int,
                        Definition varchar(10),
                        Caption_Status varchar(50)
                        )'''

        cursor.execute(create_query)
        mydb.commit()
    except:
        st.write("Videos Table already created")

    vi_list = []
    db = client["Youtube_data"]
    coll1 = db["channel_details"]
    for vi_data in coll1.find({},{"_id":0,"video_information":1}):
        for i in range(len(vi_data["video_information"])):
            vi_list.append(vi_data["video_information"][i])
```

```
    df3 = pd.DataFrame(vi_list)


    for index, row in df3.iterrows():

        insert_query = '''INSERT INTO videos(Channel_Name,
                                             Channel_Id,
                                             Video_Id,
                                             Title,
                                             Tags,
                                             Thumbnail,
                                             Description,
                                             Published_Date,
                                             Duration,
                                             Views,
                                             Likes,
                                             Comments,
                                             Favorite_Count,
                                             Definition,
                                             Caption_Status)
                        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s)'''
        values = (
            row['Channel_Name'],
            row['Channel_Id'],
            row['Video_Id'],
            row['Title'],
            row['Tags'],
            row['Thumbnail'],
            row['Description'],
            row['Published_Date'],
            row['Duration'],
            row['Views'],
            row['Likes'],
            row['Comments'],
            row['Favorite_Count'],
            row['Definition'],
            row['Caption_Status']
        )

        try:
            cursor.execute(insert_query,values)
            mydb.commit()
        except:
            st.write("videos values already inserted in the table")


def comments_table():

    mydb = psycopg2.connect(host="localhost",
            user="postgres",
            password="postgresql",
            database= "Youtube_data",
            port = "5432"
            )
    cursor = mydb.cursor()

    drop_query = "drop table if exists comments"
    cursor.execute(drop_query)
    mydb.commit()
```

```
403
404     try:
405         create_query = '''CREATE TABLE if not exists comments(Comment_Id
    varchar(100) primary key,
406                         Video_Id varchar(80),
407                         Comment_Text text,
408                         Comment_Author varchar(150),
409                         Comment_Published timestamp)'''
410         cursor.execute(create_query)
411         mydb.commit()
412
413     except:
414         st.write("Comments Table already created")
415
416     com_list = []
417     db = client["Youtube_data"]
418     coll1 = db["channel_details"]
419     for com_data in coll1.find({},{"_id":0,"comment_information":1}):
420         for i in range(len(com_data["comment_information"])):
421             com_list.append(com_data["comment_information"][i])
422     df4 = pd.DataFrame(com_list)
423
424     for index, row in df4.iterrows():
425         insert_query = '''
426             INSERT INTO comments (Comment_Id,
427                                   Video_Id ,
428                                   Comment_Text,
429                                   Comment_Author,
430                                   Comment_Published)
431             VALUES (%s, %s, %s, %s, %s)
432
433         '''
434         values = (
435             row['Comment_Id'],
436             row['Video_Id'],
437             row['Comment_Text'],
438             row['Comment_Author'],
439             row['Comment_Published']
440         )
441         cursor.execute(insert_query,values)
442         mydb.commit()
443         try:
444             cursor.execute(insert_query,values)
445             mydb.commit()
446         except:
447             st.write("This comments are already exist in comments table")
448
449 def tables():
450     channels_table()
451     playlists_table()
452     videos_table()
453     comments_table()
454
455     return "Tables created successfully"
456
457 # ... (previous code)
458
459 def show_channels_table():
460     ch_list = []
461     db = client["Youtube_data"]
```

```
    coll1 = db["channel_details"]
    for ch_data in coll1.find({}, {"_id": 0, "channel_information": 1}):
        ch_list.append(ch_data["channel_information"])
    channels_table=st.dataframe(ch_list)

    return channels_table

def show_playlists_table():
    pl_list = []
    db = client["Youtube_data"]
    coll1 = db["channel_details"]
    for pl_data in coll1.find({}, {"_id": 0, "playlist_information": 1}):
        for i in range(len(pl_data["playlist_information"])):
            pl_list.append(pl_data["playlist_information"][i])
    playlists_table=st.dataframe(pl_list)
    return playlists_table

def show_videos_table():
    vi_list = []
    db = client["Youtube_data"]
    coll1 = db["channel_details"]
    for vi_data in coll1.find({}, {"_id": 0, "video_information": 1}):
        for i in range(len(vi_data["video_information"])):
            vi_list.append(vi_data["video_information"][i])
    videos_table=st.dataframe(vi_list)
    return videos_table

def show_comments_table():
    com_list = []
    db = client["Youtube_data"]
    coll1 = db["channel_details"]
    for com_data in coll1.find({}, {"_id": 0, "comment_information": 1}):
        for i in range(len(com_data["comment_information"])):
            com_list.append(com_data["comment_information"][i])
    comments_table=st.dataframe(com_list)
    return comments_table

class HomePage:
    def show(self):
        st.title(":blue[YOUTUBE DATA HARVESTING] ")
        if st.button(":white[welcome]"):
            st.balloons()
            st.write({"Name": "Sabarish"}, {"Batch No": "D104"}, {"Project":
"Youtube Data Harvesting"})

# ... (previous code)

class ProjectPage:
    def show(self):
        st.title("Welcome to :red[Project Section] ")
        channel_id = st.text_input("Enter the Channel id")
        channels = channel_id.split(',')
        channels = [ch.strip() for ch in channels if ch]

        if st.button("Collect and Store data"):
            for channel in channels:
                ch_ids = []
                db = client["Youtube_data"]
                coll1 = db["channel_details"]
                for ch_data in coll1.find({}, {"_id": 0, "channel_information": 1}):
```

```
521                     ch_ids.append(ch_data["channel_information"]["Channel_Id"])
522                 if channel in ch_ids:
523                     st.success("Channel details of the given channel id: " + channel
    + " already exist")
524                 else:
525                     output = channel_details(channel)
526                     st.success(output)
527 
528         if st.button("Migrate to SQL"):
529             Tables = tables()
530             st.success(Tables)
531 
532         show_table = st.radio("SELECT THE TABLE FOR VIEW",
533                               (":green[channels]", ":orange[playlists]",
    ":red[videos]", ":blue[comments]"))
534 
535         if show_table == ":green[channels]":
536             show_channels_table()
537         elif show_table == ":orange[playlists]":
538             show_playlists_table()
539         elif show_table == ":red[videos]":
540             show_videos_table()
541         elif show_table == ":blue[comments]":
542             show_comments_table()
543 
544         # SQL connection
545         mydb = psycopg2.connect(host="localhost",
546                                 user="postgres",
547                                 password="postgresql",
548                                 database="Youtube_data",
549                                 port="5432")
550         cursor = mydb.cursor()
551 
552         question = st.selectbox(
553             'please select your queries',
554             (
555                 "1. What are the names of all the videos and their corresponding
    channels?",
556                 "2. Which channels have the most number of videos, and how many
    videos do they have?",
557                 "3. What are the top 10 most viewed videos and their respective
    channels?",
558                 "4. How many comments were made on each video, and what are their
    corresponding video names?",
559                 "5. Which videos have the highest number of likes, and what are
    their corresponding channel names?",
560                 "6. What is the total number of likes and dislikes for each video,
    and what are their corresponding video names?",
561                 "7. What is the total number of views for each channel, and what are
    their corresponding channel names?",
562                 "8. What are the names of all the channels that have published
    videos in the year 2022?",
563                 "9. What is the average duration of all videos in each channel, and
    what are their corresponding channel names?",
564                 "10. Which videos have the highest number of comments, and what are
    their corresponding channel names?"
565 
566             )
567         )
```

```
        if question == '1. What are the names of all the videos and their corresponding channels?':
            query1 = "select Title as videos, Channel_Name as ChannelName from videos;"
            cursor.execute(query1)
            mydb.commit()
            t1=cursor.fetchall()
            st.write(pd.DataFrame(t1, columns=["Video Title","Channel Name"]))

        elif question == '2. Which channels have the most number of videos, and how many videos do they have?':
            query2 = "select Channel_Name as ChannelName,Total_Videos as NO_Videos from channels order by Total_Videos desc;"
            cursor.execute(query2)
            mydb.commit()
            t2=cursor.fetchall()
            st.write(pd.DataFrame(t2, columns=["Channel Name","No Of Videos"]))

        elif question == '3. What are the top 10 most viewed videos and their respective channels?':
            query3 = '''select Views as views , Channel_Name as ChannelName,Title as VideoTitle from videos
                                where Views is not null order by Views desc limit 10;'''
            cursor.execute(query3)
            mydb.commit()
            t3 = cursor.fetchall()
            st.write(pd.DataFrame(t3, columns = ["views","channel Name","video title"]))
        elif question == '4. Comments in each video':
            query4 = "select Comments as No_comments ,Title as VideoTitle from videos where Comments is not null;"
            cursor.execute(query4)
            mydb.commit()
            t4=cursor.fetchall()
            st.write(pd.DataFrame(t4, columns=["No Of Comments", "Video Title"]))

        elif question == '5. Videos with highest likes':
            query5 = '''select Title as VideoTitle, Channel_Name as ChannelName, Likes as LikesCount from videos
                            where Likes is not null order by Likes desc;'''
            cursor.execute(query5)
            mydb.commit()
            t5 = cursor.fetchall()
            st.write(pd.DataFrame(t5, columns=["video Title","channel Name","like count"]))

        elif question == '6. likes of all videos':
            query6 = '''select Likes as likeCount,Title as VideoTitle from videos;'''
            cursor.execute(query6)
            mydb.commit()
            t6 = cursor.fetchall()
            st.write(pd.DataFrame(t6, columns=["like count","video title"]))

        elif question == '7. views of each channel':
            query7 = "select Channel_Name as ChannelName, Views as Channelviews from channels;"
            cursor.execute(query7)
            mydb.commit()
```

```
            t7=cursor.fetchall()
            st.write(pd.DataFrame(t7, columns=["channel name","total views"]))

        elif question == '8. videos published in the year 2022':
            query8 = '''select Title as Video_Title, Published_Date as VideoRelease, Channel_Name as ChannelName from videos
                        where extract(year from Published_Date) = 2022;'''
            cursor.execute(query8)
            mydb.commit()
            t8=cursor.fetchall()
            st.write(pd.DataFrame(t8,columns=["Name", "Video Published On", "ChannelName"]))

        elif question == '9. average duration of all videos in each channel':
            query9 =  "SELECT Channel_Name as ChannelName, AVG(Duration) AS average_duration FROM videos GROUP BY Channel_Name;"
            cursor.execute(query9)
            mydb.commit()
            t9=cursor.fetchall()
            t9 = pd.DataFrame(t9, columns=['ChannelTitle', 'Average Duration'])
            T9=[]
            for index, row in t9.iterrows():
                channel_title = row['ChannelTitle']
                average_duration = row['Average Duration']
                average_duration_str = str(average_duration)
                T9.append({"Channel Title": channel_title ,  "Average Duration": average_duration_str})
            st.write(pd.DataFrame(T9))

        elif question == '10. videos with highest number of comments':
            query10 = '''select Title as VideoTitle, Channel_Name as ChannelName, Comments as Comments from videos
                            where Comments is not null order by Comments desc;'''
            cursor.execute(query10)
            mydb.commit()
            t10=cursor.fetchall()
            st.write(pd.DataFrame(t10, columns=['Video Title', 'Channel Name', 'NO Of Comments']))


class AccountPage:
    def show(self):
        st.title("Welcome to :violet[Profile]😎")

        choice = st.selectbox('Login/Signup', ['Login', 'Signup'])
        if choice == 'Login':
            email = st.text_input("Email Address")
            password = st.text_input("Password", type='password')

            st.button('Login')

        else:
            email = st.text_input("Email Address")
            password = st.text_input("Password", type="password")

            username = st.text_input("Enter your username")
            st.button("Create my account")

class MultiPageApp:
    def __init__(self):
```

```
        self.pages = {
            "Home": HomePage(),
            "Project": ProjectPage(),
            "Account": AccountPage()
        }


    def run(self):
        st.title("Project 1")
        options = st.sidebar.radio("Main Menu", ["Home", "Project", "Account"])
        # Display the selected page
        selected_page = self.pages.get(options, HomePage())
        selected_page.show()

if __name__ == "__main__":
    st.set_page_config(page_title="Streamlit MultiPage Example")

    app = MultiPageApp()
    app.run()
```