

Metropolitan State University

ICS 432 - 01: Distributed and Cloud Computing

Fall 2021

Assignment 4: Using APIs to access a Firestore Collection

Out: Saturday, November 20, 2021

Due: 11:59 PM on Wednesday, December 8, 2021

Late submissions will NOT be accepted.

IMPORTANT

Late submissions are not accepted for this homework. Let me know ASAP if you have less than \$15 in your GCP Billing account because I may need to order more coupons for the class and the order can take up to three business days to be processed.

Objective

In this homework, you will create a web service and deploy it on GCP App Engine. The goal of the web service is to read data from a Firestore collection that include a data about Netflix shows. The application is implemented using the following seven steps:

- Step 1: Create a collection on Firestore to store the Netflix shows data set.
- Step 2: Create a Service Account with Firestore access role.
- Step 3: Write Python code to upload the data set to the Firestore collection.
- Step 4: Create a simple web service using Flask.
- Step 5: Add code to your Flask app to read data from the Firestore collection.
- Step 6: Deploy the application to GCP App Engine.
- Step 7: Clean up.

What to submit?

- Before you start, read the whole homework carefully.
- At various parts of the assignment, you are asked to take screen shots of your work. Open a word document and paste the screen shots in this document in the same order as mentioned in the lab. Make sure to highlight the screen shot number.
- After you include all the screenshots, upload the word document to the designated D2L folder by 11:59 PM on Wednesday, December 8, 2021.

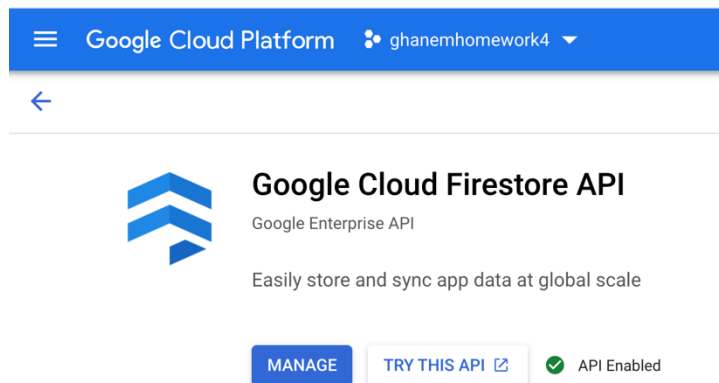
Step 1: Create a collection on Firestore to store the Netflix shows data set

1. Login to GPC console and create a project called `<your-last-name>homework4`.
2. For this homework, we will create a collection to store a data set about Netflix shows. The data set is given to you in a csv file where each line in the file includes the following 12 attributes (comma separated) about one netflix show: `show_id, type, title, director, cast, country, date_added, release_year, rating, duration, listed_in, description`.
3. Download the `netflix_titles.csv` file from D2L and examine the data.
4. Follow the steps from Lab 11 to create a collection on Firestore called `<your-last-name>netflixshows`. Do NOT insert any data in the collection.

Homework screenshot #1: take a screenshot of the collection dashboard showing that the Firestore collection is successfully created.

Step 2: Create a Service Account with Firestore access role

1. Enable Cloud Firestore API by from GCP main menu → APIs & Services → Library. Search for Firestore. Click on the output of the search and then and click on **Enable**. Note that if you are using the same GCP account as Lab 11, then you may find that this API is already enabled. Click on **Manage**.



2. From the left menu click on Credentials then click on **CREATE CREDENTIALS**. Click on Create a Service Account.
3. Enter account name as `<your-last-name>homework4serviceaccount`. Click on **CREATE AND CONTINUE**.
4. In the **Grant this service account access to project** page, click on the dropdown menu for 'Select a role' and select **Firebase Rules System** as the role. Click on **CONTINUE**. Click on **DONE**.

5. Create a json key in the same way as you did in Lab 11. The .json key file will be downloaded on your local computer. You will use this file in the following step.

Homework screenshot #2: take a screenshot of the service accounts dashboard showing your newly created service account.

Step 3: Write Python code to upload Netflix data to Firestore

1. Open Editor, create a folder called `<your-last-name>homework4` for the homework.
2. Upload the .json key file to that folder.
3. Upload `netflix_title.csv` to the folder.

Homework screenshot #3: Take as screenshot of your editor showing the directory and the two uploaded files.

4. Create a python file called `uploaddata.py`.
5. Write python code as shown in the following figure to do the following:
 - a. opens the `netflix_title.csv`
 - b. uses a for loop to read lines from the csv file and split each line (based on comma) into 12 attributes.
 - c. prints the first two attributes of each line.



```
uploaddata.py ×
CloudStorage > ghanemhomework4 > uploaddata.py > ...
1  import os
2  import sys
3  import csv
4
5  file_name = 'netflix_titles.csv'
6
7  #parse csv file
8  count = 0
9  with open(file_name, "r",encoding='latin-1') as csv_file:
10     csv_reader = csv.reader(csv_file, delimiter=',')
11     count = 0
12     for row in csv_reader:
13         print(count)
14         count = count + 1
15         print(row[0]+"\\t"+row[1])
16
```

6. In the editor's command window, use `pwd`, `ls`, and `cd` to go inside the directory that contains `uploaddata.py` and `netflix_title.csv` files.
7. Run your code and make sure movie ids and titles are printed correctly.

Homework screenshot #4: take a screenshot of your code.

Homework screenshot #5: take a screenshot showing the final few lines of the output of your code. How many rows are there in the csv file?

8. Add code in your `uploaddata.py` file to upload the Netflix shows data from `netflix_title.csv` to the Firestore collection created in Step 1. This code is similar to the code you wrote in Lab 11 to upload documents to your `Students` collection. For the Netflix shows collection, the data JSON object should include 12 attributes to include the 12 values you read from each line in the file (i.e., `row[0]` to `row[11]`). After you construct a data object for each line in the csv file, use the following to add the object to the collection. Make sure include the appropriate imports and variables as used in Lab 11.

```
doc_ref.add(data)
```

Homework screenshot #6: take a screenshot of your code.

Homework screenshot #7: go to Firestore dashboard and take two screenshots of two documents from the collection.

Step 4: Creating a Simple web service using Flask

Flask is web framework in Python that lets you develop web applications easily. In this step, you will use the Flask library to implement a very simple RESTful API to test the framework. In the next step, you will add code to your Flask app to retrieve data from the Firestore Netflix shows collection.

- 1- The code for the Flask app must be organized in a specific format. In the following steps, use the exact same file name and folder order.
 - a. Create a folder called `<your-last-name>netflixapp` inside your homework 4 folder.
 - b. Create the following three files inside your `netflixapp` folder. File names must be exactly as shown:
 - i. `requirements.txt`: includes python packages that are needed by the app.
 - ii. `app.yaml`: specification of the app runtime environment.
 - iii. `main.py`: app code.

2. Type the following in `requirements.txt`

```
Flask==1.1.2
firebase-admin
```

3. Type the following in `app.yaml`


```
runtime: python38
```

4. In the command window, use `cd netflixapp` to go inside the application folder then run the following command to install project dependencies.

```
pip install -r requirements.txt
```

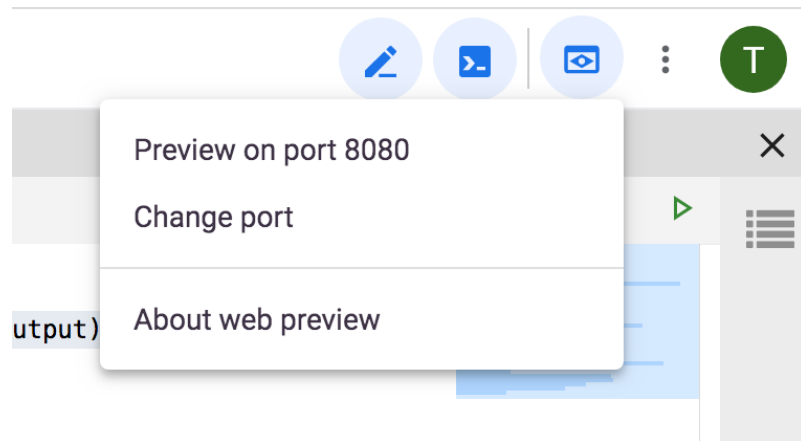
5. Type the following code in `main.py`. This code is just a simple Flask application that includes only one route `@app.route('/')` to just display a welcome message. You will use this route to call your application from the browser and make sure the application is set up correctly.

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def netflix():
7      return 'Welcome to Netflix Titles. You can list titles by Type or Release Year'
8
9  if __name__ == '__main__':
10     app.run(host='127.0.0.1', port=8080, debug=True)
11
```

6. Click run . If everything is correct, you will see the following message that tells you that your application is ready to use from the browser.

```
! Problems    >_ Python    >_ Python    >_ Python    >_ Python    >_ Python    >_ Python x
* Debug mode: on
* Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 841-084-802
□
Cloud Code - Minikube Stopped
```

7. From the Editor's top menu, click on the Web Preview symbol, and click on **Preview on Port 8080**.



Homework screenshot #8: take a screenshot of the browser window showing the Welcome message. Make sure to include URL bar in the screenshot.

Step 5: Adding code to your Flask App to interact with Firebase Collection

In this step, you will add three APIs (or three routes) to your Flask app where each route can be used to issue a query based on an attribute in the Netflix shows collection.

searchbyType API

The **searchbyType** API takes the `type` of the show (e.g., `TV Show`) as input and returns as output all entries in the collection that matches the input `type`. The route to searchAPI is `@app.route('/year/<year>')`. For example, to retrieve all Netflix shows with `type` equals to **TV Shows**, you write the following URL in your browser using your app's URL instead of the blue-highlighted part:

<https://8080-cs-62881455303-default.cs-us-central1-pits.cloudshell.dev/type/TV Show>

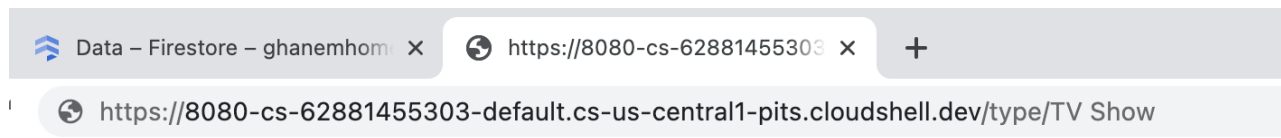
Add the following code to your `main.py` to implement `searchbyType` API. You need also to add the appropriate `import` to import libraries to interact with Firebase. You also need to define `cred`, `app`, and `store` as was done in Lab 11 and Step 3 of this homework. You also need to copy the `.json` key inside the `netflixapp` folder.

```

28     return 'list of movies produced of type {intype} is <br/> {list}'.format(intype=type,list=output)
29
30 @app.route('/year/<year>')
31 def searchbyYear(year):
32     doc_ref = store.collection(u'ghanemnetflixmovies').where(u'release_year',u'==',year)
33     docs = doc_ref.get()
34     output = ""
35     count = 1
36     for doc in docs:
37         output = output + str(count) + "\t" + format(doc.to_dict())+ "<br/>"
38         count = count + 1
39     return 'list of movies produced of type {intype} is <br/> {list}'.format(intype=type,list=output)
40
41 if __name__ == '__main__':
42     app.run(host='127.0.0.1', port=8080, debug=True)
43

```

Run the code and then view the app from the web browser. Add /type/TV Show to the URL as shown in the following figure and hit Enter. You will see JSON document that includes all show with type TV Show.



Homework screenshot #9: take a screenshot of your code.

Homework screenshot #10: take a screenshot of the browser output.

Homework screenshot #11: go to the Netflix collection on Firebase console and examine the documents and pick another value for show type . Call searchbyType API with the other show type and take screenshot of the output. Make sure to include URL in your screenshot.

searchbyYear API

Add another API, called searchbyYear, to your Flask application. The searchbyYear API takes year as input and prints in the output all shows with release_year equal to the input year.

Homework screenshot #12: take a screenshot of your code.

Homework screenshot #13: call the searchbyYear API using 2015 and take a screenshot of the output on the browser. Include URL bar in the screenshot.

Homework screenshot #14: call the searchbyYear API using another year value of your choice. Take a screenshot of the output. Make sure to include URL in your screenshot.

Implement an API of your choice

Choose another attribute from the Netflix shows data set and implement a third API to retrieve documents from the collection using this third attribute.

Homework screenshot #15: take a screenshot of your code.

Homework screenshot #16 and #17: call your API two times with two different values for your attribute and take two screenshots to show the output. Make sure to include the URLs in the screenshots

Step 6: Deploying the API

In Steps 4 and 5, you tested the application as it is hosted on your GCP Editor's account. In this step, you will deploy your application using Google App Engine so that other users can access your application.

App Engine uses YAML (Yet Another Markup Language) to specify a deployment's configuration where a file called `app.yaml` must be included in your application folder. In the application you implemented in this assignment, `app.yaml` just includes the runtime environment to specify that `python38` is needed to run this application.

To deploy your application:

- 1- open the Cloud Shell terminal, use `pwd`, `ls`, and `cd` to go inside your `netflixapp` folder.
- 2- Type the following command. If prompted, click on **Authorize**. Choose a region for your app. Note that, you may get an error saying that this project already has an app. If you get that error, then just proceed to the following step.

```
gcloud app create
```

- 3- Run the following command to get your current project id.

```
gcloud config list project
```

- 4- To deploy your app, run the following command and use your project id instead of the yellow-highlighted part. If prompted to 'Do you want to continue [y/n]?', enter Y.

```
gcloud app deploy app.yaml --project netflix-data-api
```

```
thanaa_ghanem@cloudshell:~/CloudStorage/ghanemhomework4/netflixapp (ghanemhomework4)$ gcloud config list project
[core]
project = ghanemhomework4

Your active configuration is: [cloudshell-21808]
thanaa_ghanem@cloudshell:~/CloudStorage/ghanemhomework4/netflixapp (ghanemhomework4)$ gcloud app deploy app.yaml --project ghanemhomework4
```


- 5- Wait until the deployment is completed, this may take few minutes. Once completed, you will be given a URL that you can use to access your app. The default URL of your app is a subdomain on `appspot.com` that starts with your project's ID (e.g., <https://netflix-data-api.uc.r.appspot.com>). Copy the URL and paste in a browser window to access your web service. You can test the APIs by adding `/type/TV Show` or `/year/2015` to the URL. For example,

`https://ghanemhomework4.uc.r.appspot.com/type/TV Show`
`https://ghanemhomework4.uc.r.appspot.com/year/2015`

Homework screenshot #18: take a screenshot of the command window after the deployment is complete to show your app's URL.

Homework screenshot #19, #20, and #21: take three screenshots to show the output of calling the three APIs implemented in Step 5. Make sure to include URLs in the screenshots.

Step 7: Clean Up

- 1- Disable your application to avoid incurring charges. From GCP main menu, go to **SERVERLESS** → **App Engine** → **Settings** page and click **Disable Application**. Enter your project id and click **DISABLE**.
- 2- Shut down your project to avoid wasting your credits. Click on project name, go to project settings, and click **SHUT DOWN**.