**Problem Statement:**

This code challenge is designed to test your ability to problem solve **with limited information**, analyze an endpoint, demonstrate your ability to manage a complex Oauth2 session, and use that session to test an API endpoint.

Action to do in Browser

1. Go to https://clarity.dexcom.com/
2. Click "Dexcom CLARITY for Home Users"
3. Give the username/password: nilepatest001/Password@1 in the login window
4. Click Login to Clarity home page.

Analyze network traffic when doing above manual steps in browser. Suggest to use Network console in Chrome. Write a test that only interacts with api http request calls.

**Test Steps:**

1. Login with username and password nilepatest001/Password@1
2. Make HTTP POST request call to "/api/subject/1594950620847472640/analysis\_session"
3. Assert analysisSessionId should not be None

Some Requirements and hints:

* **DO NOT** use selenium and webdriver for this test. Create a test for API, not WEB.
* Use sessions obj from python requests library to easily maintain a "session"
* (Optional) Use a python automation framework. e.g. robot framework (Keyword Driven), unittest (like JUnit), cucumber (BDD) or any other test framework.
* (Optional) Centralize configuration for host, username and password (Do not hard code) – dynamically entering the username/password:
* Upload code in GitHub is highly recommended.
* **Code completion in python is not mandatory. Explain the understanding of login process in detail or complete code in a different language as an alternative.**

**Out of Scope:**

For the purpose of this problem, the below scenarios are not in scope:

* Error handling scenarios
* Server outages
* Use of Selenium webdriver to automate the UI.

**My Understanding of how OAuth2 works in Clarity application within the Dexcom domain:**

OAuth2 protocol is a means of providing authentication and authorization to secure the access to an api endpoint which is widely used. When a 3rd party application uses the API, it redirects the user to the login page of the Dexcom’s Clarity page where the user can enter his credentials and upon the user’s authorization, the 3rd party gets access to the user information that is specified in the ‘Scope’. In this problem statement, the ‘scope’ is limited to openid profile offline\_access.

**Assumption**:

The application is already registered and has the client ID. I was able to retrieve the client ID (DAEC20AC-9626-4B0E-94B5-B674E298F51E) by analyzing the network traffic.

**Steps identified from Chrome Inspect - Network trace**:

Given the problem statement, the user clicks on the <https://clarity.dexcom.com> and upon clicking “*Dexcom for home users*” we are redirected to the login page which provides us with an option to enter the username and password.

After I enter the username and password, (the call is a made to the authorization server by using the URL : <https://uam1.dexcom.com/identity/connect/authorize?client_id=DAEC20AC-9626-4B0E-94B5-B674E298F51E> .This https requests informs the server about user’s identity and requests for the authorization code by passing the client Id, response type as code in the parameters to fetch the auth\_code.

Upon successful authentication and authorization by the user, a HTTP 302 will be returned and the authorization code gets appended to the ‘code’ parameter in the Redirect Uri (<https://clarity.dexcom.com/users/auth/dexcom_sts/callback>).

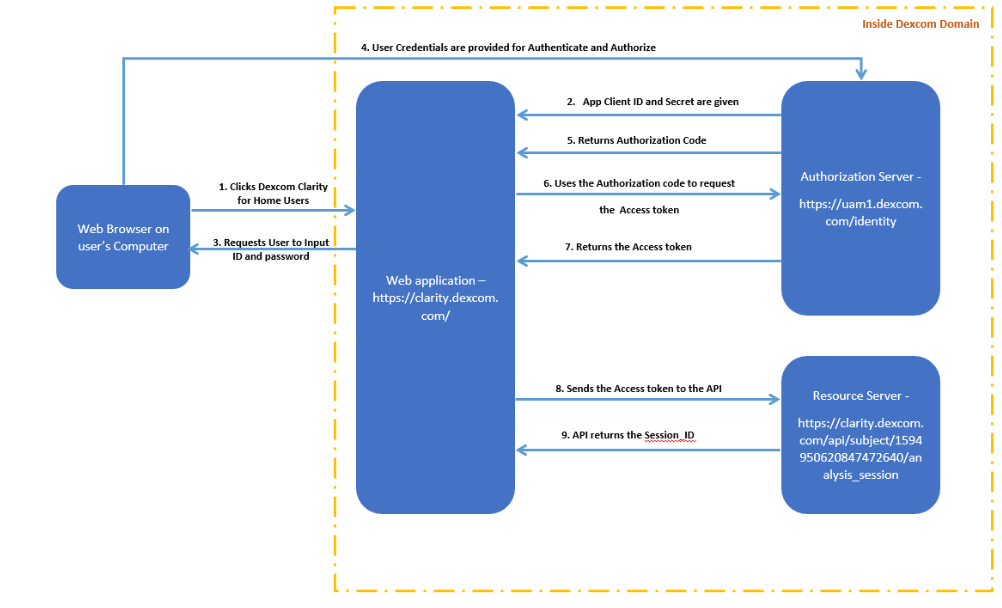
The client then sends back this authorization code to the authorization server by making a request to the URL:  [https://uam1.dexcom.com/identity/connect/token](%20https://uam1.dexcom.com/identity/connect/token) to fetch the access token.

Once the access token is received, a POST request is made on the endpoint API (<https://clarity.dexcom.com/api/subject/1594950620847472640/analysis_session>) and the response received is the value of the AnalysisSessionID along with other parameters

We can then assert on the AnalysisSessionID to check if it not NULL and pass the test case, otherwise fail it.

**Flow Diagram:**

The below flow diagram depicts my understanding of the login process as well as the endpoints involved.



**Code Snippet (I have attached the .py files for both the approaches)**

**Approach 1(Approach1.py):** If the access token is generated by the service test account and if I can have access to it directly, I could use the access token and pass it in the post method for the API endpoint like below. I could save the associated refresh token and use that to continually renew my authenticated session.

import requests

base\_url = "https://clarity.dexcom.com/api/subject/1594950620847472640/analysis\_session"

headers = {

'access-token': "<Provide the Access Token>",

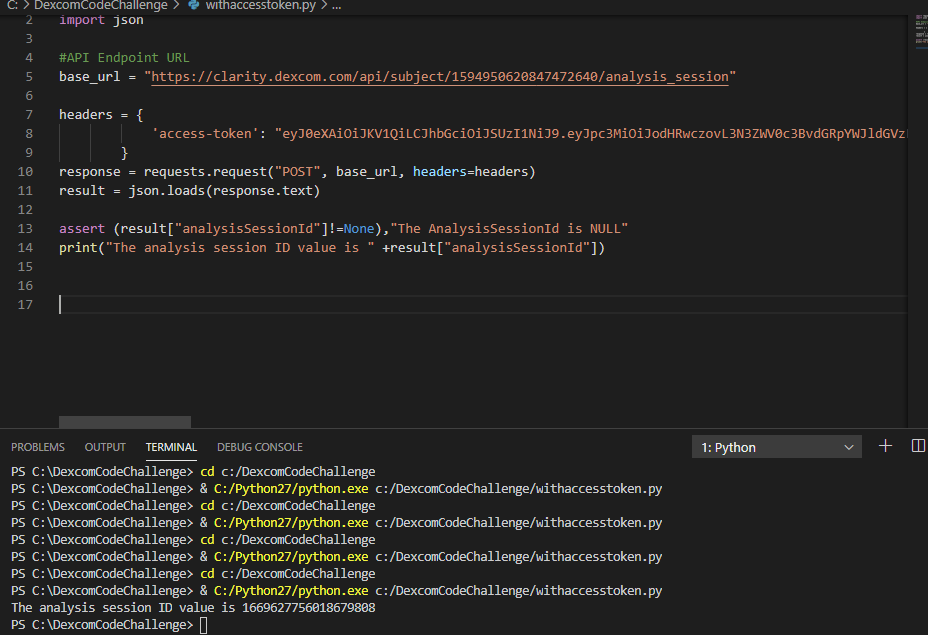
}

response = requests.request("POST", base\_url, headers=headers)

result = json.loads(response.text)

assert (result["analysisSessionId"]!=None),"The AnalysisSessionId is NULL"

print("The analysis session ID value is " +result["analysisSessionId"])

Here is the screenshot of the output. I have passed the access token directly by fetching it from the network tab. ****

**Approach2(Approach2.py):** In the below code I have tried to fetch the auth\_code from the response header location value and then pass that auth\_code to the get\_access\_token function to fetch the access token, however with the below code I am getting the 404 error and that is the challenge I am facing after hours of debugging. The point that I am stuck is at when we pass the user credentials, it is a POST request (https://uam1.dexcom.com/identity/connect/login?) followed by a get request with the URL : https://uam1.dexcom.com/identity/connect/authorize?client\_id=DAEC20AC-9626-4B0E-94B5-B674E298F51E&prompt&redirect\_uri=https%3A%2F%2Fclarity.dexcom.com%2Fusers%2Fauth%2Fdexcom\_sts%2Fcallback&response\_type=code&scope=openid%20profile%20offline\_access&state=d3e5615a6b210ecee17368879b4a02e49ffc43564bb086bf&ui\_locales=en-US

I think we need to create a user session and use that session to authorize and fetch the token.

The response header is not returning the location value in the below code.

import urllib, urllib2, urlparse

# This sample shows how to obtain an OAuth access token

# without the user/browser flow

# Dexcom OAuth client ID

client\_id = "DAEC20AC-9626-4B0E-94B5-B674E298F51E"

#client\_secret = 'client\_secret\_here'

#base\_url ="https://api.dexcom.com/v2/oauth2"

base\_url = 'https://uam1.dexcom.com/identity/connect'

api\_url = "https://clarity.dexcom.com/api/subject/1594950620847472640/analysis\_session"

# put login credentials here

username = "nilepatest001"

password = "Password@1"

class AuthCodeRedirectHandler(urllib2.HTTPRedirectHandler):

"""

redirect handler that pulls the auth code sent back

by OAuth off the query string of the redirect URI given in the

Location header. Does no checking for other errors or bad/missing

information.

"""

def http\_error\_302(self, req, fp, code, msg, headers):

"""handler for 302 responses that assumes a properly constructed

OAuth 302 response and pulls the auth code out of the header"""

qs = urlparse.urlparse(headers["location"]).query

auth\_code = urlparse.parse\_qs(qs)['code'][0]

return auth\_code

def build\_auth\_code\_request(username, password):

"""This method builds the URL request with the below necessary parameters"""

auth\_data = urllib.urlencode({

"client\_id": client\_id,

"response\_type": "code",

"username": username,

"password": password,

"action": "Login",

"scope":"openid profile offline\_access",

"redirect\_uri" :"https://clarity.dexcom.com/users/auth/dexcom\_sts/callback",

})

req = urllib2.Request(url=base\_url + "/authorize"+auth\_data)

return req

def get\_access\_token(code):

"""

Gets an OAuth access token given an OAuth authorization code

"""

access\_token\_params = urllib.urlencode({

'grant\_type': 'authorization\_code',

'client\_id': client\_id,

'code': code

})

req = urllib2.Request(base\_url + '/token', access\_token\_params)

f = urllib2.urlopen(req)

return f.read()

if \_\_name\_\_ == "\_\_main\_\_":

req = build\_auth\_code\_request(username, password)

opener = urllib2.build\_opener(AuthCodeRedirectHandler)

auth\_code = opener.open(req)

print(auth\_code) #print the auth code that is retrieved from the redirect URI

access\_token = get\_access\_token(auth\_code)

print(access\_token)

#Do a post request on the API endpoint URL by passing the access token in the header.

response = requests.request("POST", api\_url, headers=access\_token)

#Parsing the Json String into python dictionary

result = json.loads(response.text)

#Assert that the analysisSessionId is not NULL and if it is NUll, display the Assert Error

assert (result["analysisSessionId"]!=None),"The AnalysisSessionId is NULL"

print("The analysis session ID value is " +result["analysisSessionId"])