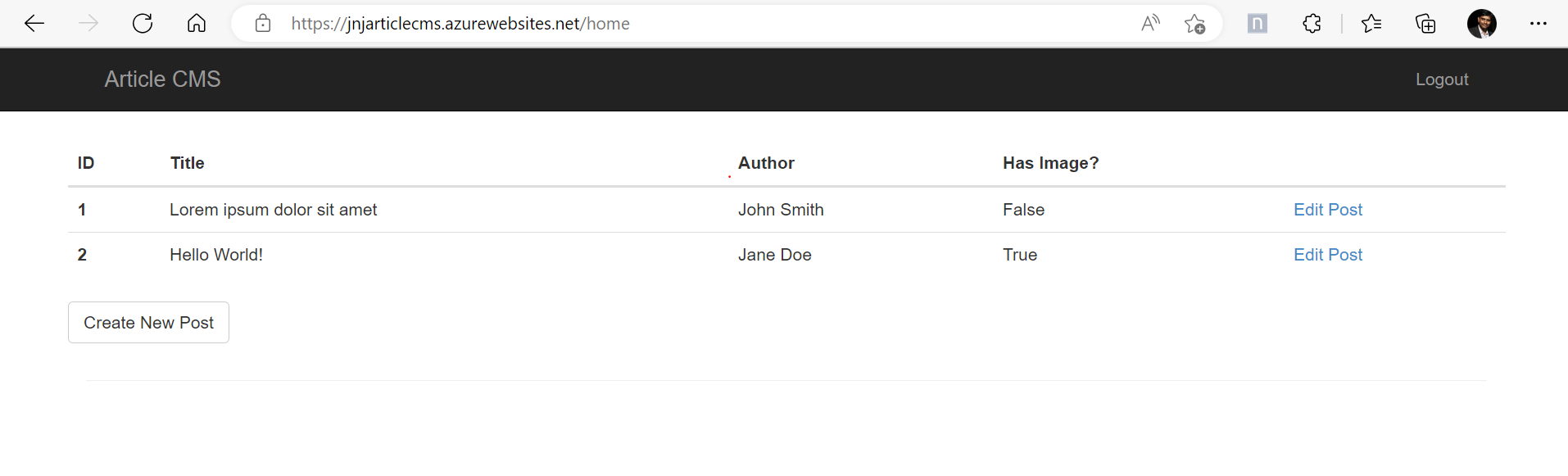
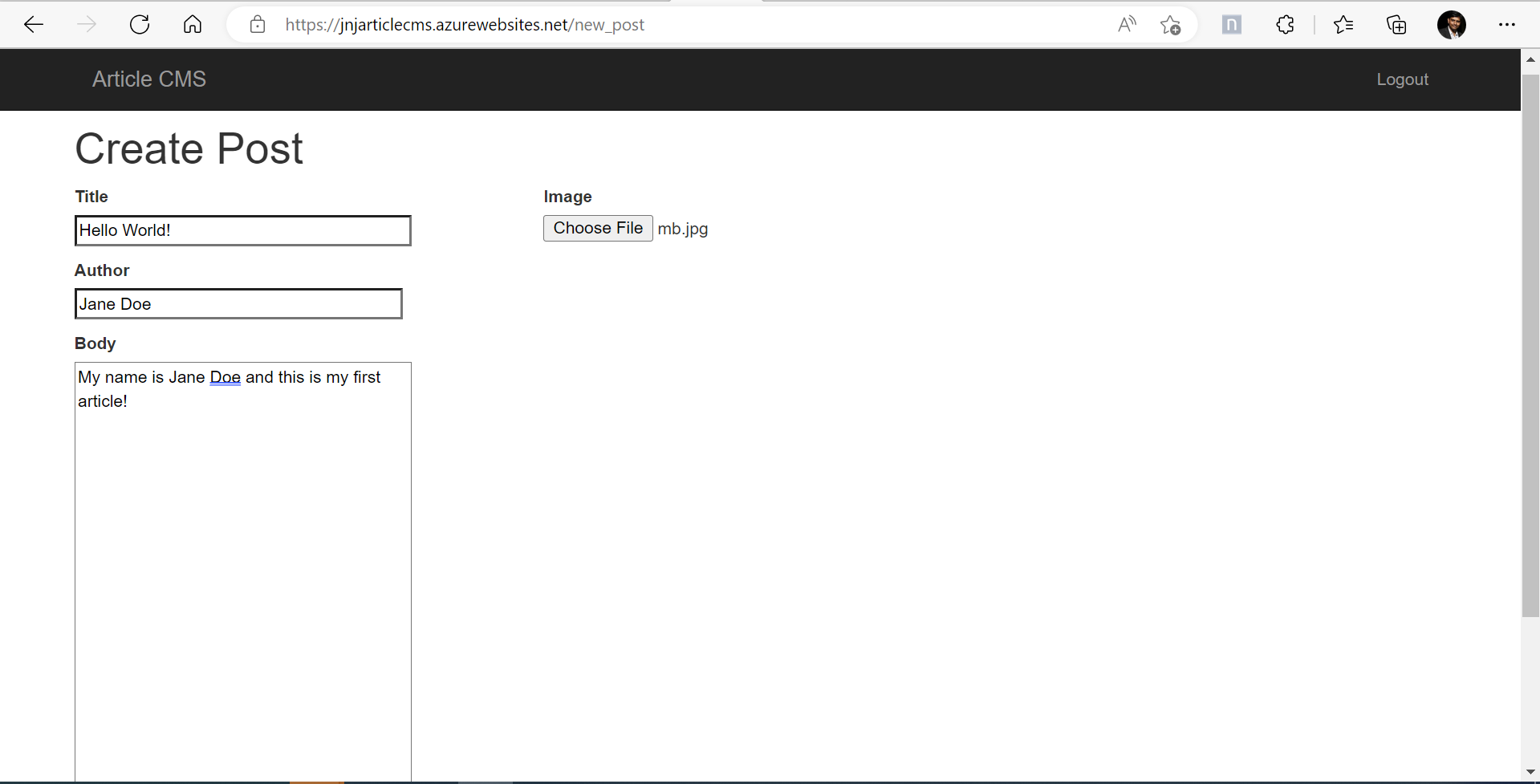
Article CMS Project

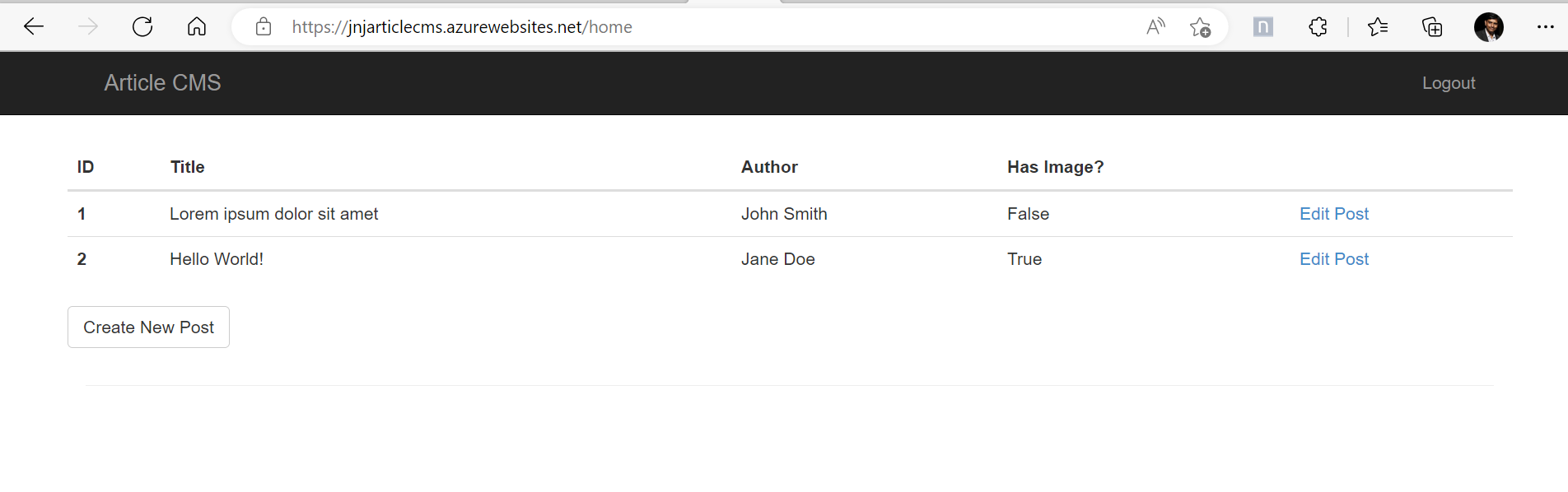
Home Page



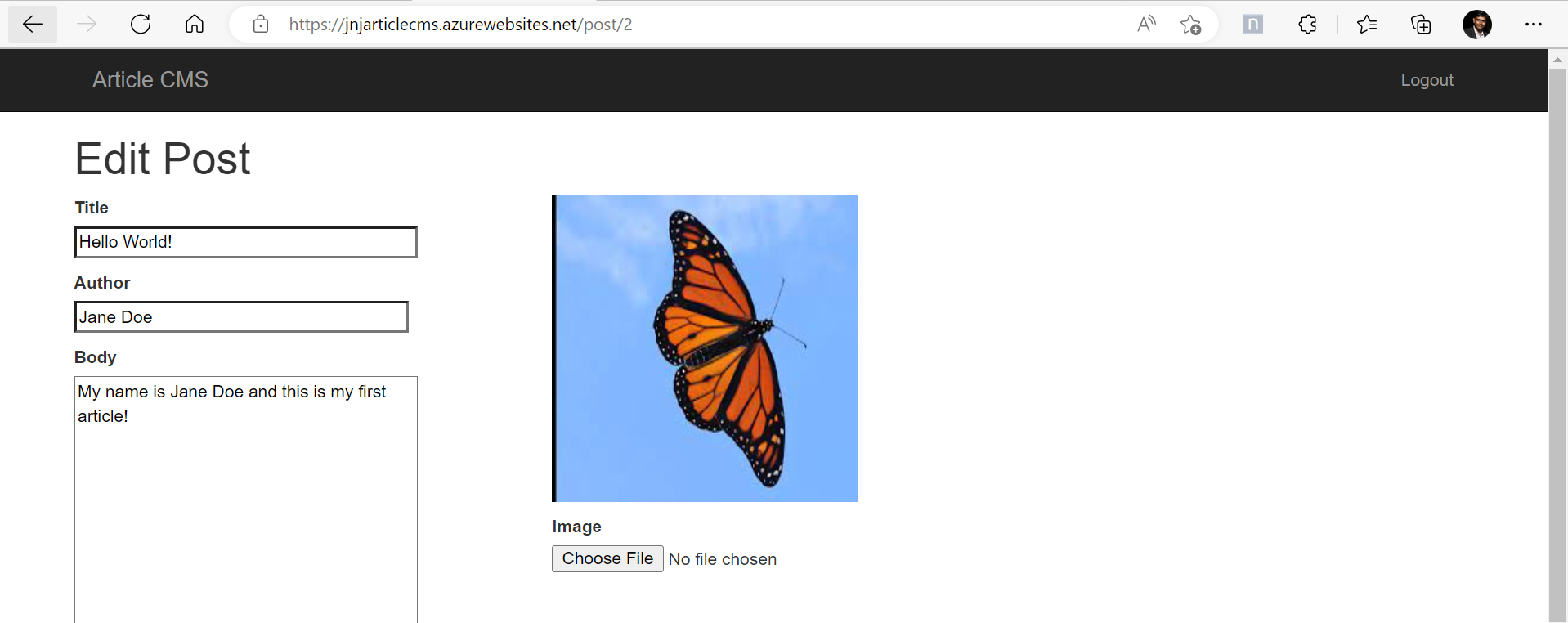
New post Screenshot:



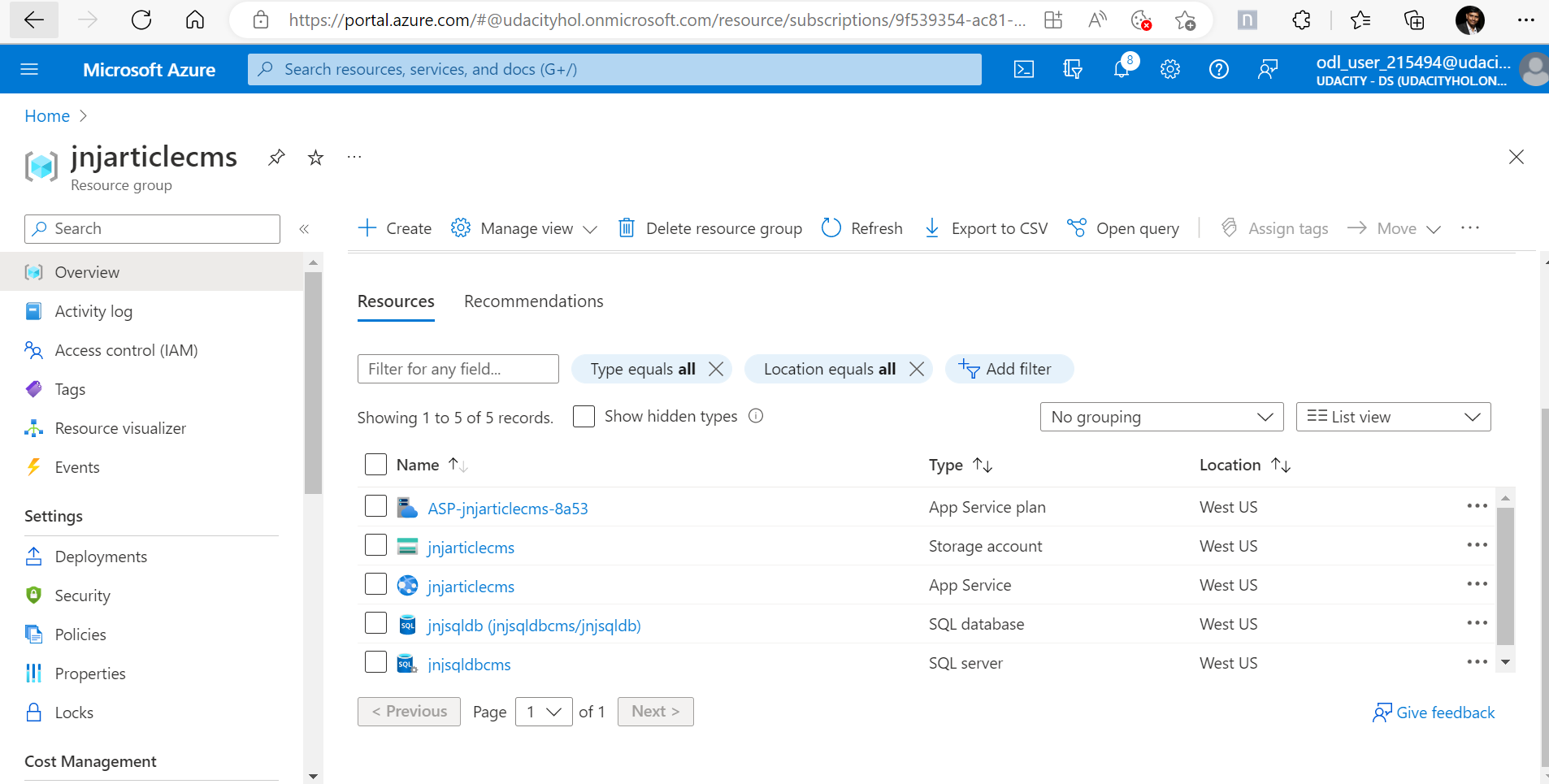
Once Submitted the new post



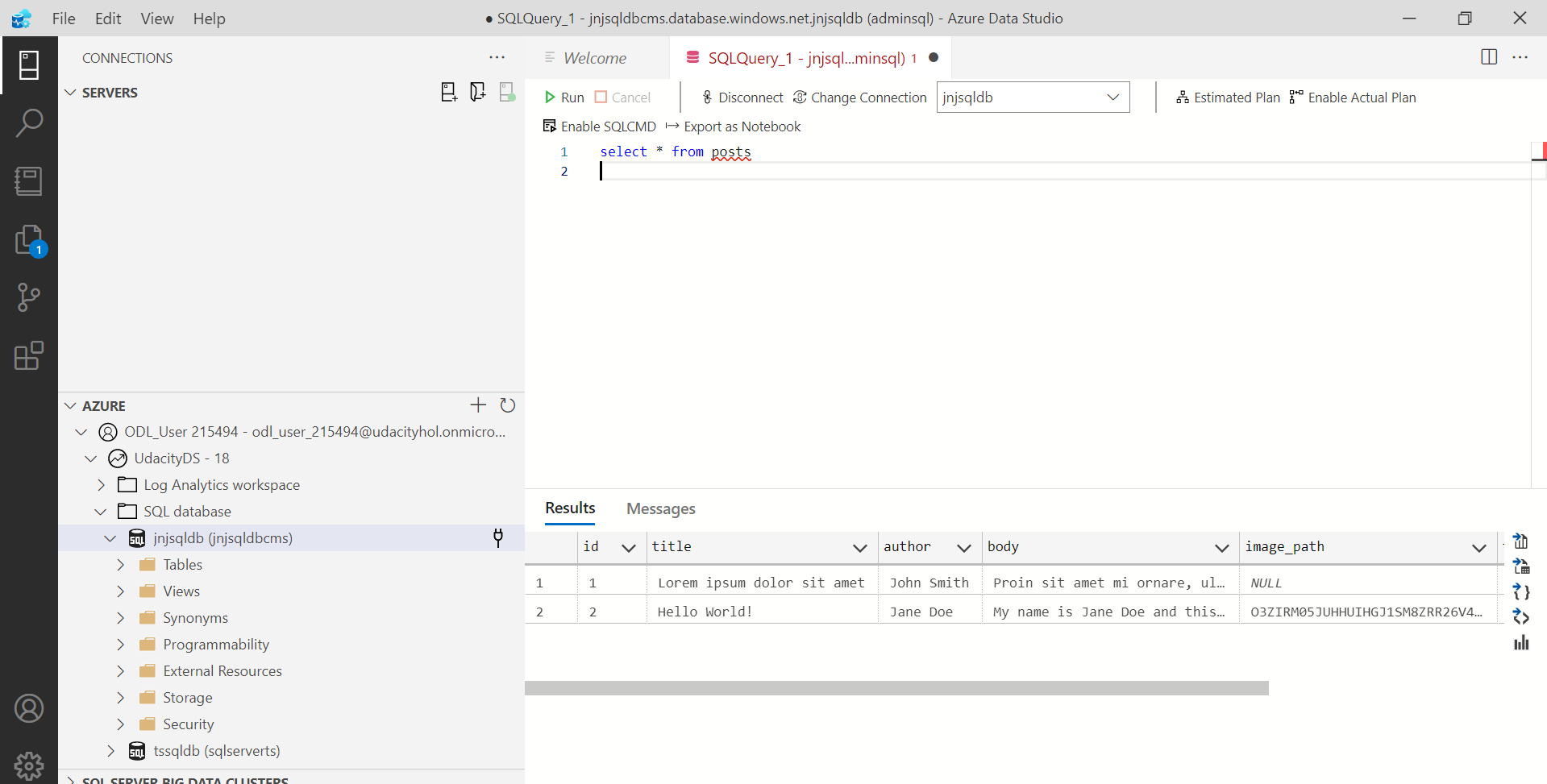
Edit Post Snapshot:



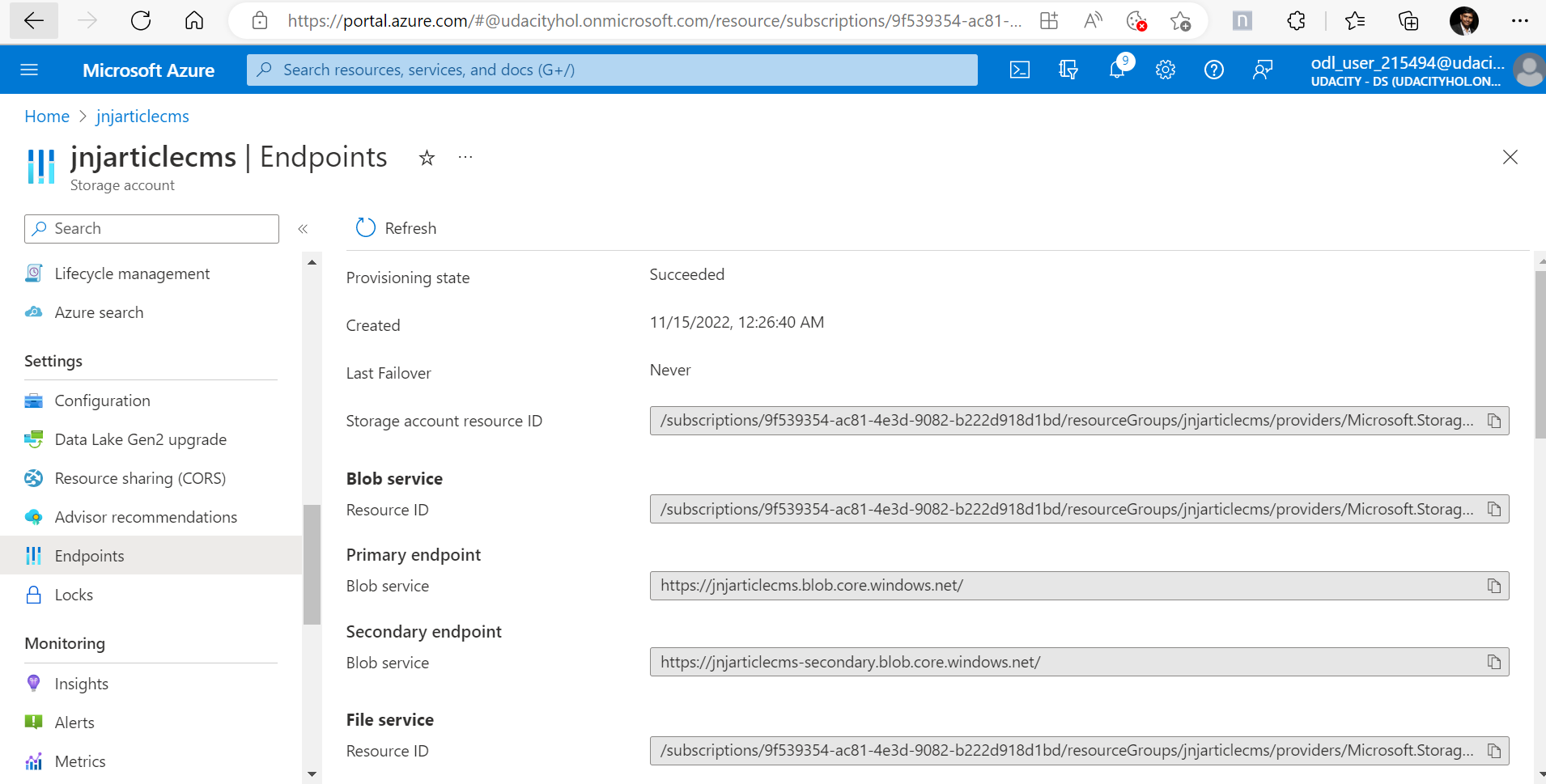
Resource group along with the resources:



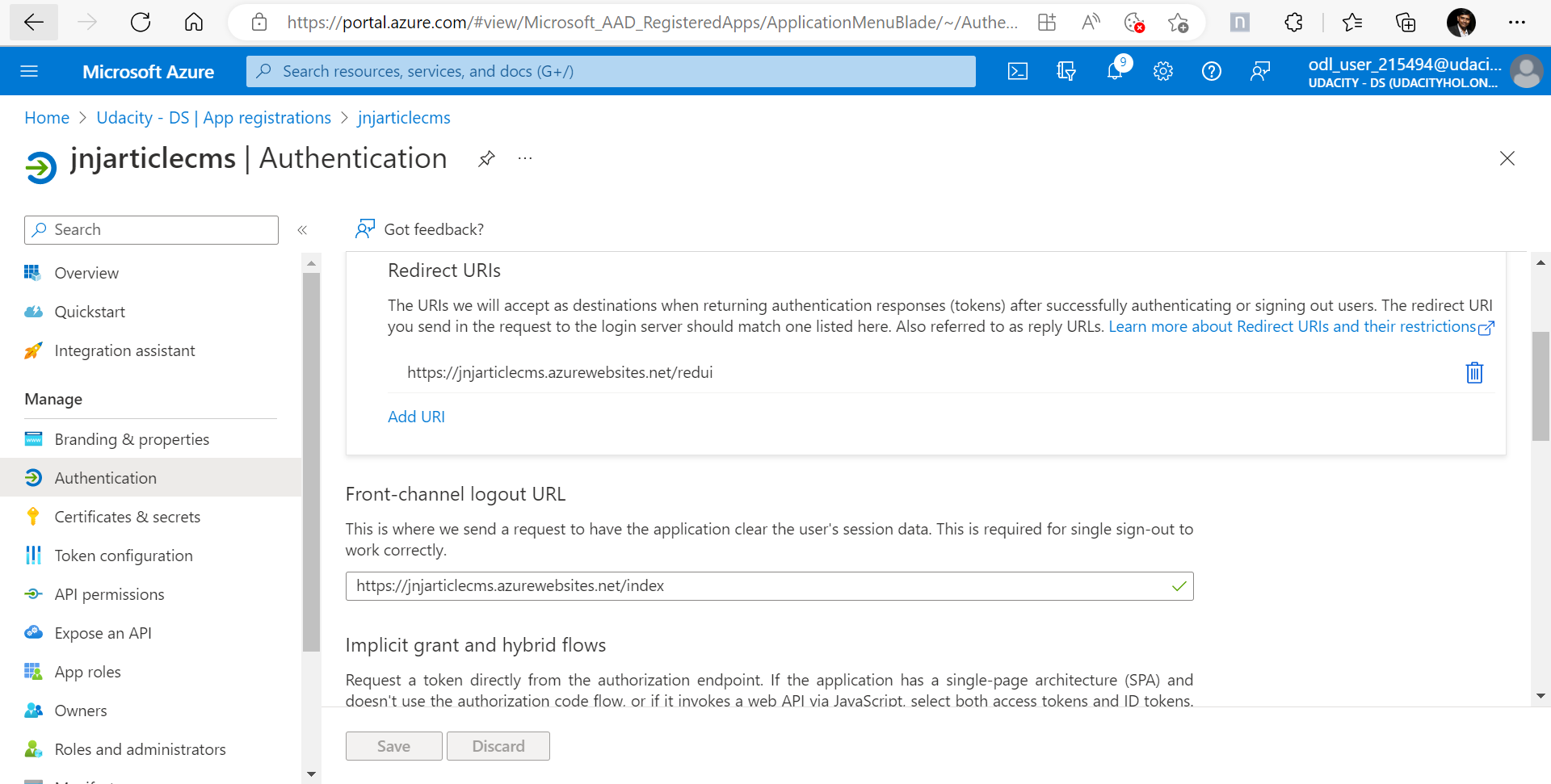
SQL Database screenshot:



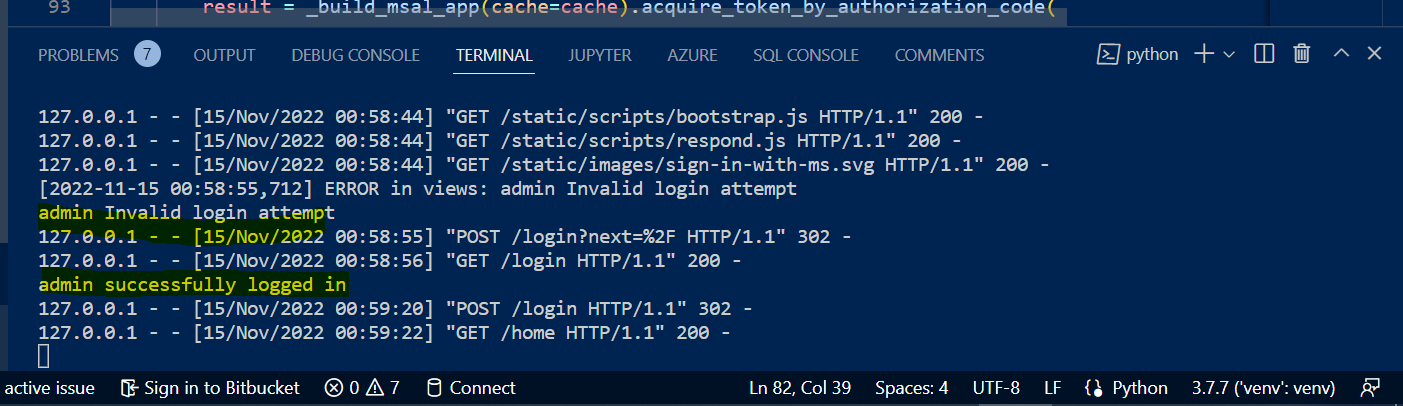
Blob screenshot:



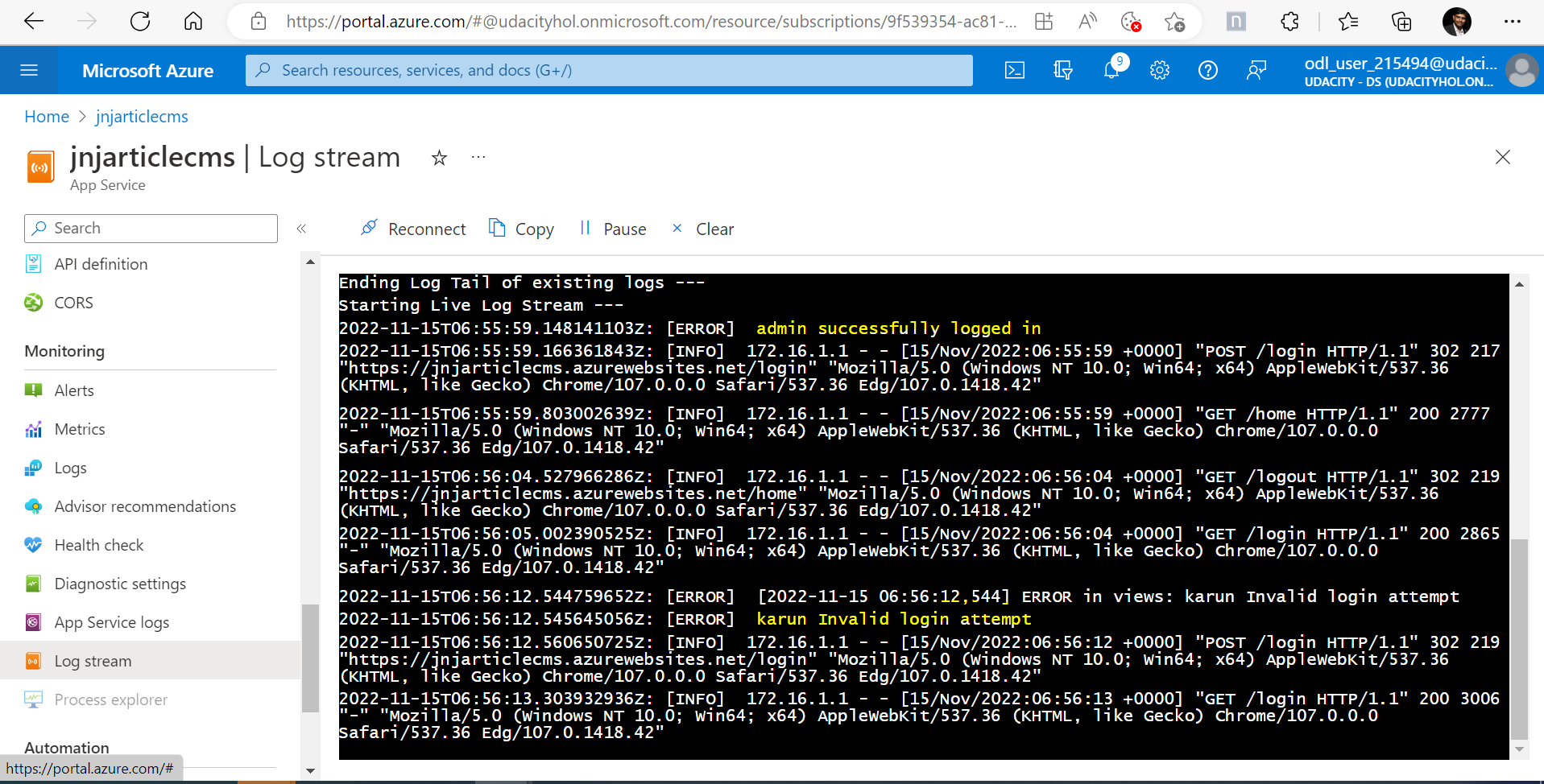
Redirect URI:



Log Screenshot:



Log Stream Output:



Main.py code:

"""

The flask application package.

"""

import logging

from flask import Flask

from config import Config

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager

from flask\_session import Session

app = Flask(\_\_name\_\_)

app.config.from\_object(Config)

# TODO: Add any logging levels and handlers with app.logger

app.logger.setLevel(logging.WARNING)

streamHandler = logging.StreamHandler()

streamHandler.setLevel(logging.WARNING)

app.logger.addHandler(streamHandler)

Session(app)

db = SQLAlchemy(app)

login = LoginManager(app)

login.login\_view = 'login'

import FlaskWebProject.views

Views.py code:

"""

Routes and views for the flask application.

"""

from datetime import datetime

from flask import render\_template, flash, redirect, request, session, url\_for

from werkzeug.urls import url\_parse

from config import Config

from FlaskWebProject import app, db

from FlaskWebProject.forms import LoginForm, PostForm

from flask\_login import current\_user, login\_user, logout\_user, login\_required

from FlaskWebProject.models import User, Post

import msal

import uuid

import logging

imageSourceUrl = 'https://'+ app.config['BLOB\_ACCOUNT']  + '.blob.core.windows.net/' + app.config['BLOB\_CONTAINER']  + '/'

@app.route('/')

@app.route('/home')

@login\_required

def home():

    user = User.query.filter\_by(username=current\_user.username).first\_or\_404()

    posts = Post.query.all()

    return render\_template(

        'index.html',

        title='Home Page',

        posts=posts

    )

@app.route('/new\_post', methods=['GET', 'POST'])

@login\_required

def new\_post():

    form = PostForm(request.form)

    if form.validate\_on\_submit():

        post = Post()

        post.save\_changes(form, request.files['image\_path'], current\_user.id, new=True)

        return redirect(url\_for('home'))

    return render\_template(

        'post.html',

        title='Create Post',

        imageSource=imageSourceUrl,

        form=form

    )

@app.route('/post/<int:id>', methods=['GET', 'POST'])

@login\_required

def post(id):

    post = Post.query.get(int(id))

    form = PostForm(formdata=request.form, obj=post)

    if form.validate\_on\_submit():

        post.save\_changes(form, request.files['image\_path'], current\_user.id)

        return redirect(url\_for('home'))

    return render\_template(

        'post.html',

        title='Edit Post',

        imageSource=imageSourceUrl,

        form=form

    )

@app.route('/login', methods=['GET', 'POST'])

def login():

    if current\_user.is\_authenticated:

        return redirect(url\_for('home'))

    form = LoginForm()

    if form.validate\_on\_submit():

        user = User.query.filter\_by(username=form.username.data).first()

        if user is None or not user.check\_password(form.password.data):

            flash('Invalid username or password')

            #flash(form.username.data)

            app.logger.error('%s Invalid login attempt',form.username.data)

            return redirect(url\_for('login'))

        login\_user(user, remember=form.remember\_me.data)

        next\_page = request.args.get('next')

        app.logger.error('%s successfully logged in',form.username.data)

        if not next\_page or url\_parse(next\_page).netloc != '':

            next\_page = url\_for('home')

        return redirect(next\_page)

    session["state"] = str(uuid.uuid4())

    auth\_url = \_build\_auth\_url(scopes=Config.SCOPE, state=session["state"])

    return render\_template('login.html', title='Sign In', form=form, auth\_url=auth\_url)

@app.route(Config.REDIRECT\_PATH)  # Its absolute URL must match your app's redirect\_uri set in AAD

def authorized():

    if request.args.get('state') != session.get("state"):

        return redirect(url\_for("home"))  # No-OP. Goes back to Index page

    if "error" in request.args:  # Authentication/Authorization failure

        return render\_template("auth\_error.html", result=request.args)

    if request.args.get('code'):

        cache = \_load\_cache()

        # TODO: Acquire a token from a built msal app, along with the appropriate redirect URI

        result = \_build\_msal\_app(cache=cache).acquire\_token\_by\_authorization\_code(

        request.args['code'],scopes=Config.SCOPE, redirect\_uri=url\_for('authorized', \_external=True, \_scheme='https'))

        if "error" in result:

            return render\_template("auth\_error.html", result=result)

        session["user"] = result.get("id\_token\_claims")

        # Note: In a real app, we'd use the 'name' property from session["user"] below

        # Here, we'll use the admin username for anyone who is authenticated by MS

        user = User.query.filter\_by(username="admin").first()

        login\_user(user)

        \_save\_cache(cache)

    return redirect(url\_for('home'))

@app.route('/logout')

def logout():

    logout\_user()

    if session.get("user"): # Used MS Login

        # Wipe out user and its token cache from session

        session.clear()

        # Also logout from your tenant's web session

        return redirect(

            Config.AUTHORITY + "/oauth2/v2.0/logout" +

            "?post\_logout\_redirect\_uri=" + url\_for("login", \_external=True))

    return redirect(url\_for('login'))

def \_load\_cache():

    # TODO: Load the cache from `msal`, if it exists

    cache = msal.SerializableTokenCache()

    if session.get('token\_cache'):

        cache.deserialize(session['token\_cache'])

    return cache

def \_save\_cache(cache):

    # TODO: Save the cache, if it has changed

    if cache.has\_state\_changed:

        session['token\_cache'] = cache.serialize()

def \_build\_msal\_app(cache=None, authority=None):

    # TODO: Return a ConfidentialClientApplication

    return msal.ConfidentialClientApplication(

       Config.CLIENT\_ID, authority=authority or Config.AUTHORITY,

       client\_credential=Config.CLIENT\_SECRET, token\_cache=cache)

def \_build\_auth\_url(authority=None, scopes=None, state=None):

    # TODO: Return the full Auth Request URL with appropriate Redirect URI

    return \_build\_msal\_app(authority=authority).get\_authorization\_request\_url(

       scopes or [],

       state=state or str(uuid.uuid4()),

       redirect\_uri=url\_for('authorized', \_external=True, \_scheme='https'))

Writeup

**rite-up Template**

**Analyze, choose, and justify the appropriate resource option for deploying the app.**

*For****both****a VM or App Service solution for the CMS app:*

* *Analyze costs, scalability, availability, and workflow*
* *Choose the appropriate solution (VM or App Service) for deploying the app*
* *Justify your choice*

Article CMS project is a light weighted Web project combination of

a) SQL server - To store the users and post tables

b) Storage account - To Store the pictures

c) Flask Web application - Small python based web applications

d) MSAL & Active Directory - For the standard authentication and logging activities

**My preference is to go with the Web Applications over VM**

**VM (Vurtual Machine):** Generally, VM cost more(compared to Web Apps) and is specially used when custom software applications are to be installed. This application is not required to have full control of underlying operating systems.

**WebApps:** *Web Apps support a maximum of 14 GB of memory and 4vCPU cores. That is enough for our application. Cost is cheaper compared to VM. Our app is a nonproduction use and doesn't have any sensitive information.*

**Assess app changes that would change your decision.**

*Detail how the app and any other needs would have to change for you to change your decision in the last section.* If we have highly secured data(Government or confidential) to deal with recommended to go with VM as compared to web apps. If we need a production app with a higher user base across regions I would go with VM.