DAY-10

Task 1: Create a Python script that uses OAuth authentication to connect to a RESTful service.

SOLUTION:

```
import os

import secrets

from urllib.parse import urlencode


from env import load_env

from flask import Flask, redirect, url_for, render_template, flash, session, \

    current_app, request, abort

from flask_sqlalchemy import SQLAlchemy

from flask_login import LoginManager, UserMixin, login_user, logout_user,\

    current_user

import requests
```

```
load_dotenv()


app = Flask(__name__)
app.config['SECRET_KEY'] = 'top secret!'
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://root:admin_12345@localhost/demodb_4'
app.config['OAUTH2_PROVIDERS'] = {
    # Google OAuth 2.0 documentation:
    #
https://developers.google.com/identity/protocols/oauth2/web-server#httprest



    # GitHub OAuth 2.0 documentation:
    #
https://docs.github.com/en/apps/oauth-apps/building-o
```

auth-apps/authorizing-oauth-apps

```
        'github': {
                'client_id': os.environ.get('GITHUB_CLIENT_ID'),
                'client_secret': os.environ.get('GITHUB_CLIENT_SECRET'),
                'authorize_url': 'https://github.com/login/oauth/authorize',
                'token_url': 'https://github.com/login/oauth/access_token',
                'userinfo': {
                        'url': 'https://api.github.com/user/emails',
                        'email': lambda json: json[0]['email'],
                },
                'scopes': ['user:email'],
        },
}
```

```
db = SQLAlchemy(app)
login = LoginManager(app)
login.login_view = 'index'



class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64),
nullable=False)
    email = db.Column(db.String(64), nullable=True)



@login.user_loader
def load_user(id):
    return db.session.get(User, int(id))
```

```
@app.route('/')
def index():
    return render_template('index.html')


@app.route('/logout')
def logout():
    logout_user()
    flash('You have been logged out.')
    return redirect(url_for('index'))


@app.route('/authorize/<provider>')
def oauth2_authorize(provider):
    if not current_user.is_anonymous:
        return redirect(url_for('index'))
```

```
    provider_data =
current_app.config['OAUTH2_PROVIDERS'].get(provider)
    if provider_data is None:
        abort(404)

    # generate a random string for the state parameter
    session['oauth2_state'] = secrets.token_urlsafe(16)

    # create a query string with all the OAuth2
parameters
    qs = urlencode({
        'client_id': provider_data['client_id'],
        'redirect_uri': url_for('oauth2_callback',
provider=provider,

_external=True),
        'response_type': 'code',
```

```
            'scope': ' '.join(provider_data['scopes']),
            'state': session['oauth2_state'],
        })

        # redirect the user to the OAuth2 provider
authorization URL
        return redirect(provider_data['authorize_url'] + '?' +
qs)


@app.route('/callback/<provider>')
def oauth2_callback(provider):
        if not current_user.is_anonymous:
            return redirect(url_for('index'))

        provider_data =
current_app.config['OAUTH2_PROVIDERS'].get(provider)
        if provider_data is None:
```

```
            abort(404)

        # if there was an authentication error, flash the
error messages and exit
        if 'error' in request.args:
            for k, v in request.args.items():
                if k.startswith('error'):
                    flash(f'{k}: {v}')
            return redirect(url_for('index'))

        # make sure that the state parameter matches the
one we created in the
        # authorization request
        if request.args['state'] !=
session.get('oauth2_state'):
            abort(401)

        # make sure that the authorization code is present
```

```
    if 'code' not in request.args:
        abort(401)

    # exchange the authorization code for an access token
    response = requests.post(provider_data['token_url'], data={
        'client_id': provider_data['client_id'],
        'client_secret': provider_data['client_secret'],
        'code': request.args['code'],
        'grant_type': 'authorization_code',
        'redirect_uri': url_for('oauth2_callback', provider=provider,

_external=True),
    }, headers={'Accept': 'application/json'})
    if response.status_code != 200:
        abort(401)
```

```
        oauth2_token = response.json().get('access_token')
        if not oauth2_token:
            abort(401)

        # use the access token to get the user's email address
        response = requests.get(provider_data['userinfo']['url'], headers={
            'Authorization': 'Bearer ' + oauth2_token,
            'Accept': 'application/json',
        })
        if response.status_code != 200:
            abort(401)
        email = provider_data['userinfo']['email'](response.json())

        # find or create the user in the database
        user =
```

```
db.session.scalar(db.select(User).where(User.email ==
email))
        if user is None:
                user = User(email=email,
username=email.split('@')[0])
                db.session.add(user)
                db.session.commit()

        # log the user in
        login_user(user)
        return redirect(url_for('index'))



with app.app_context():
        db.create_all()

if __name__ == '__main__':
```

```
app.run(debug=True)
```