Experiment No. 8

Aim: To implement OAuth 2.0 and JWT-based API authentication in a sample web application.

Tools:

- **Python 3**
- **Flask** (pip install Flask)
- **Flask-Dance** (pip install Flask-Dance)
- **Flask-JWT-Extended** (pip install Flask-JWT-Extended)
- **Google Cloud Platform** (for OAuth 2.0 credentials)
- **Web Browser**
- **API Client** (like Postman or curl)

Learning Objective: To understand and implement a modern authentication flow combining OAuth 2.0 for third-party login (e.g., Google) and JWT for securing internal API endpoints.

Theory:

Modern web applications often separate user authentication from resource authorization. This experiment combines two standards to achieve this securely:

- **OAuth 2.0:** An authorization framework that allows applications to obtain limited access to user accounts on an HTTP service, such as Google, GitHub, or Facebook. It works by delegating user authentication to the service that hosts the user account and authorizing the third-party application to access the user's data. It provides consent-based access without sharing the user's actual credentials (like their password) with the application.
- **JSON Web Token (JWT):** A compact, URL-safe standard for creating access tokens that assert a number of claims. Once a user is authenticated (in our case, via OAuth 2.0), the application can generate a JWT. This token is signed by the application's server and given to the client. The client then includes this JWT in the Authorization header of subsequent requests to access protected API endpoints. The server can verify the JWT's signature to ensure the request is from an authenticated user, creating a stateless authentication system.

Procedure / Implementation:

This procedure is divided into configuration, application development, and testing.

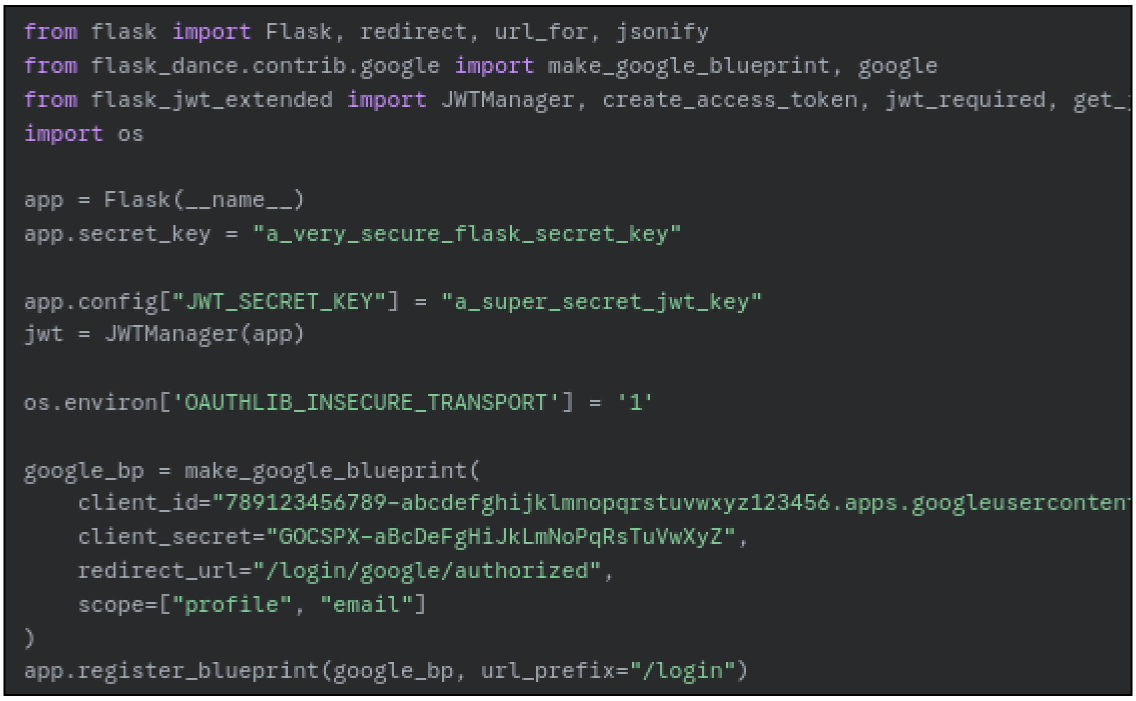
Part 1: Setup and Configuration

1. Install Required Libraries: **pip install flask flask-dance flask-jwt-extended**
2. Configure Google OAuth 2.0 Credentials:

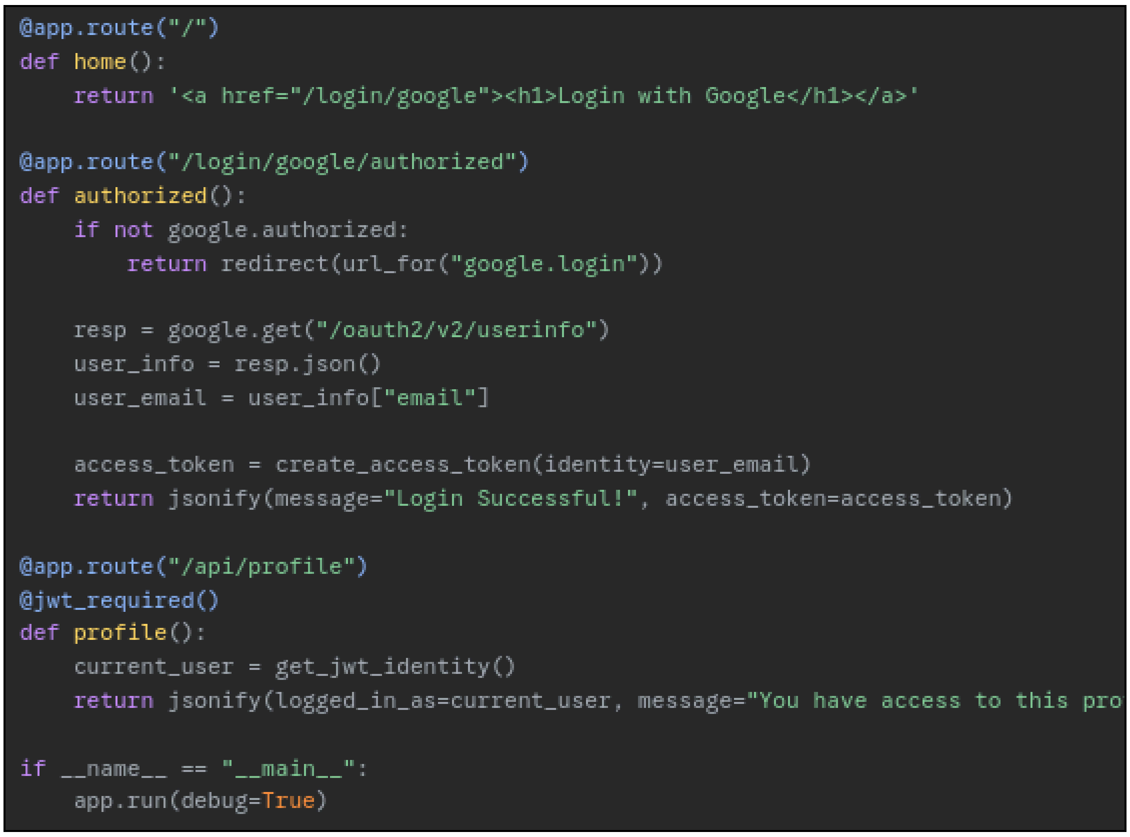
- Go to the Google Cloud Console.
- Create a new project or select an existing one.
- Navigate to **APIs & Services -> Credentials**.
- Click **Create Credentials -> OAuth client ID**.
- Select **Web application** as the application type.
- Under **Authorized redirect URIs**, add http://127.0.0.1:5000/login/google/authorized.
- Click **Create** and copy the **Client ID** and **Client Secret**. You will need these for the application.

Part 2: Application Development

1. Create the Flask Application named oauth_jwt_app.py.
2. Add the following code, replacing the placeholder values with your Google Client ID and Client Secret.



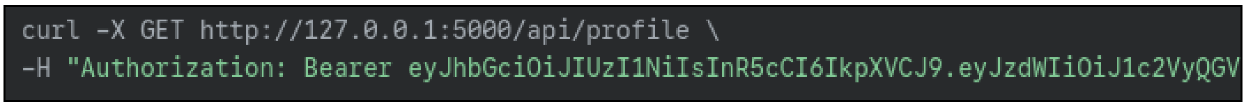
```
from flask import Flask, redirect, url_for, jsonify
from flask_dance.contrib.google import make_google_blueprint, google
from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_
import os

app = Flask(__name__)
app.secret_key = "a_very_secure_flask_secret_key"

app.config["JWT_SECRET_KEY"] = "a_super_secret_jwt_key"
jwt = JWTManager(app)

os.environ['OAUTHLIB_INSECURE_TRANSPORT'] = '1'

google_bp = make_google_blueprint(
    client_id="789123456789-abcdefghijklmnopqrstuvwxyz123456.apps.googleuserconten
    client_secret="GOCSPX-aBcDeFgHiJkLmNoPqRsTuVwXyZ",
    redirect_url="/login/google/authorized",
    scope=["profile", "email"]
)
app.register_blueprint(google_bp, url_prefix="/login")
```



```
@app.route("/")
def home():
    return '<a href="/login/google"><h1>Login with Google</h1></a>'

@app.route("/login/google/authorized")
def authorized():
    if not google.authorized:
        return redirect(url_for("google.login"))

    resp = google.get("/oauth2/v2/userinfo")
    user_info = resp.json()
    user_email = user_info["email"]

    access_token = create_access_token(identity=user_email)
    return jsonify(message="Login Successful!", access_token=access_token)

@app.route("/api/profile")
@jwt_required()
def profile():
    current_user = get_jwt_identity()
    return jsonify(logged_in_as=current_user, message="You have access to this pro

if __name__ == "__main__":
    app.run(debug=True)
```

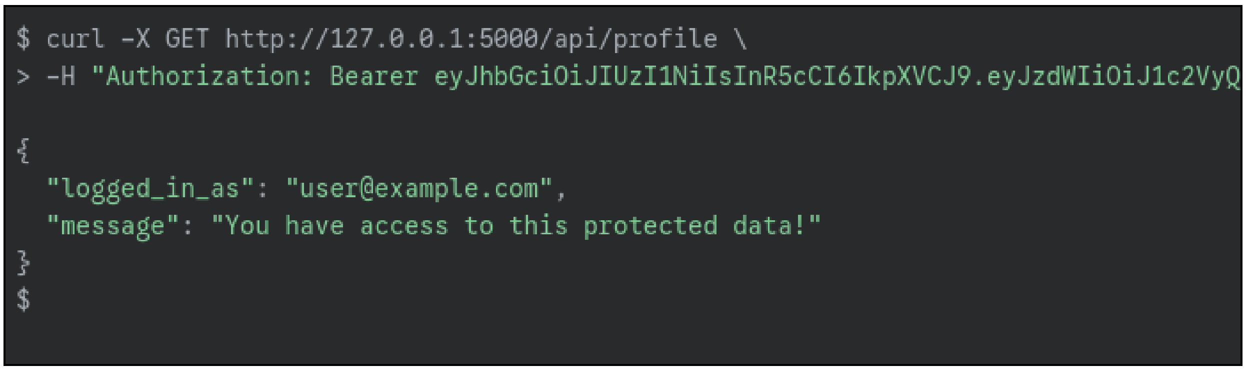
Part 3: Testing the Flow

1. Run the application: python oauth_jwt_app.py.
2. Initiate Login: Open a browser and go to http://127.0.0.1:5000/. Click the "Login with Google" link.
3. Authenticate with Google: You will be redirected to Google's login page. Sign in and grant the requested permissions.
4. Receive JWT: After authorization, Google will redirect you back to the app, which will display a JSON response containing your access_token. Copy this token.
5. Access the Protected API: Use an API client like curl to make a request to the protected endpoint, including the JWT in the header.

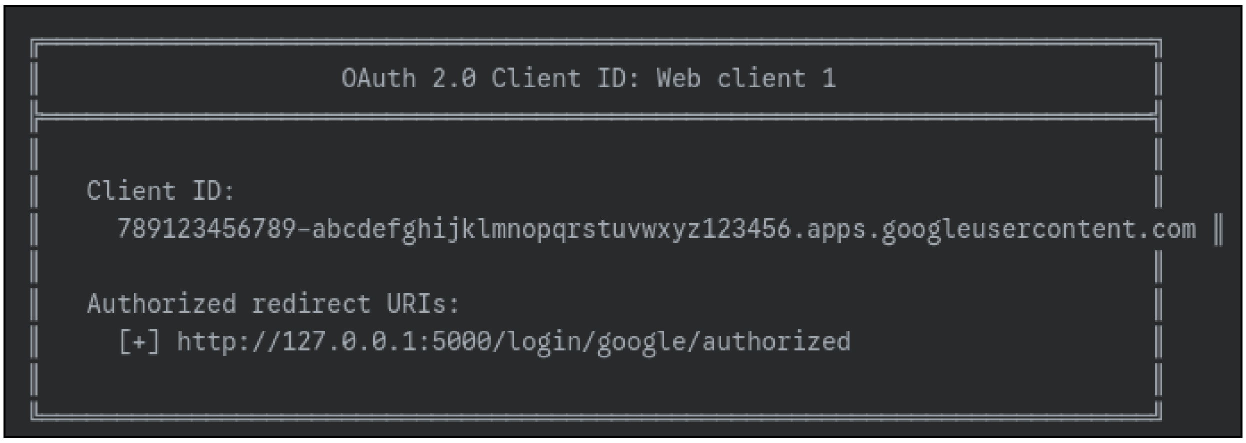


```
curl -X GET http://127.0.0.1:5000/api/profile \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyQGV
```

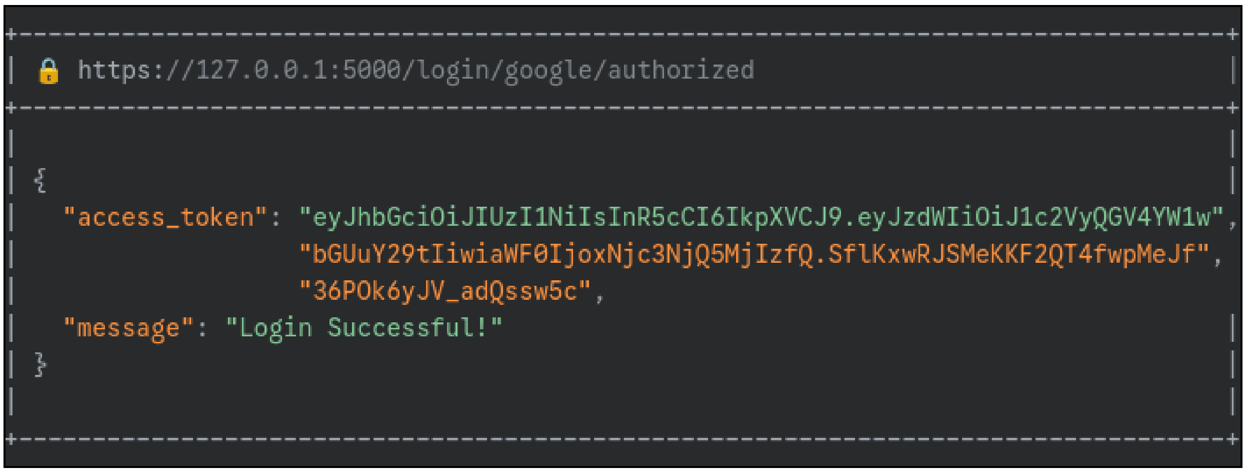
Output:



```
$ curl -X GET http://127.0.0.1:5000/api/profile \
> -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyQ

{
  "logged_in_as": "user@example.com",
  "message": "You have access to this protected data!"
}
$
```



```
╔══════════════════════════════════════════════════════════════════════════╗
║                     OAuth 2.0 Client ID: Web client 1                    ║
╠══════════════════════════════════════════════════════════════════════════╣
║                                                                          ║
║   Client ID:                                                             ║
║     789123456789-abcdefghijklmnopqrstuvwxyz123456.apps.googleusercontent.com ║
║                                                                          ║
║   Authorized redirect URIs:                                              ║
║     [+] http://127.0.0.1:5000/login/google/authorized                    ║
║                                                                          ║
╚══════════════════════════════════════════════════════════════════════════╝
```



```
+--------------------------------------------------------------------------------+
| 🔒 https://127.0.0.1:5000/login/google/authorized                              |
+--------------------------------------------------------------------------------+
|                                                                                |
| {                                                                              |
|   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyQGV4YW1w",
|                   "bGUuY29tIiwiaWF0IjoxNjc3NjQ5MjIzfQ.SflKxwRJSMeKKF2QT4fwpMeJf",
|                   "36POk6yJV_adQssw5c",
|   "message": "Login Successful!"                                               |
| }                                                                              |
|                                                                                |
+--------------------------------------------------------------------------------+
```

Learning Outcome: Performed an experiment to integrate third-party authentication using the OAuth 2.0 framework with Google. Successfully generated a JSON Web Token (JWT) upon user verification and used it to implement secure, stateless authentication for a protected API endpoint, effectively separating service authentication from application authorization.

Conclusion: This experiment successfully demonstrates a modern, robust authentication pattern that leverages the strengths of both OAuth 2.0 and JWT. By delegating user authentication to a trusted provider like Google, applications enhance security and improve user experience. The subsequent use of JWTs enables a scalable, stateless architecture for securing internal APIs. This hybrid approach is a standard and highly effective practice for building secure and modern web services.

Name:

Class: BE-CSE

Roll No.:

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				