

1. Aim and Objective

The primary aim of this project was to design and implement a secure web application by decoupling the authentication and authorization logic from the core application code.

The key objectives were:

- To build a Python Flask web application with both public and protected routes.
- To use **Keycloak** as a central, external **Identity Provider (IdP)** to manage all user identities, logins, and credentials.
- To implement a **Single Sign-On (SSO)** flow, allowing users to log in via Keycloak and be securely redirected back to the Flask app.
- To implement **Role-Based Access Control (RBAC)** to create a protected /admin page that is only accessible to users with a specific "admin" role.
- To securely fetch and display user profile information (like name and email) from the IdP after a successful login.

2. Procedure and Methodology

The project was executed in two main parts: setting up the Identity Provider (Keycloak) and building the client web application (Flask).

Part 1: Keycloak Configuration (Identity Provider)

1. **Run Keycloak:** The Keycloak server was launched as a Docker container, listening on `http://localhost:8080`.
2. `docker run -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin ...`
3. **Realm Setup:** We used the default master realm for this project.
4. **Client Creation:** A new client named flask-app was created within the master realm to represent our Python application.
5. **Client Configuration:** The client's settings were configured as:
 - **Client authentication:** Enabled
 - **Valid redirect URIs:** Set to `http://localhost:5000/oidc/callback`, which is the URL flask-oidc listens on.
6. **Role Creation:** An admin realm role was created to identify administrative users.

7. **User Configuration:** The default admin user was edited:
 - First Name, Last Name, and Email were added.
 - The admin role was assigned to this user via the "Role mapping" tab.
8. **Scope Mapping:** To ensure the admin role was included in the security token, we navigated to Clients > flask-app > Client scopes > roles > Mappers and added the "realm roles" predefined mapper.

Part 2: Flask Application (Service Provider)

1. **Environment Setup:** A Python environment was created with Flask and flask-oidc installed.
2. **Configuration (client_secrets.json):** A JSON file was created to store the credentials from Keycloak, allowing the Flask app to communicate with it.
3. **Application Code (app.py):**
 - The Flask app was initialized and configured to use the client_secrets.json file.
 - **Public Route (/):** The home page was made accessible to all users.
 - **Protected Routes (/login, /profile):** The @oidc.require_login decorator was used to protect these routes, forcing any unauthenticated user to be redirected to Keycloak for login.
 - **Profile Data:** The /profile route uses oidc.user_getinfo(['email', 'profile']) to fetch the logged-in user's details from Keycloak's UserInfo endpoint.
 - **Admin-Only Route (/admin):**
 - A custom decorator, @admin_required, was built.
 - This decorator uses oidc.user_getinfo(['roles']) to check if the admin role is present in the user's token.
 - If the role is missing, it returns a "403 Access Denied" error.

3. Code Implementation

The following files make up the core of the application.

app.py (Main Application)

```
import os

from flask import Flask, render_template, session, redirect, url_for, request
from flask_oidc import OpenIDConnect
from functools import wraps

app = Flask(__name__)

# --- 1. Configuration ---
app.config.update({
    'SECRET_KEY': 'a_very_secret_key_change_this_for_production',
    'OIDC_CLIENT_SECRETS': 'client_secrets.json',
    'OIDC_COOKIE_SECURE': False,
    'OIDC_CALLBACK_ROUTE': '/oidc/callback',
    'OIDC_SCOPES': ['openid', 'email', 'profile', 'roles'] # Request 'roles' scope
})

# Initialize OpenID Connect
oidc = OpenIDConnect(app)

# --- 2. Custom Decorator for Role-Based Access ---

def admin_required(f):
    """
    A decorator to ensure a user is an admin.
    """
```

Must be used *after* @oidc.require_login.

```
"""
```

```
@wraps(f)
```

```
def decorated_function(*args, **kwargs):
```

```
    if not oidc.user_loggedin:
```

```
        return redirect(url_for('login', next=request.url))
```

```
    # The correct method is 'user_getinfo(scopes)'
```

```
    # This fetches claims from the UserInfo endpoint
```

```
    user_info = oidc.user_getinfo(['roles'])
```

```
    # Check for 'roles' key and if 'admin' is in the list
```

```
    if 'roles' not in user_info or 'admin' not in user_info['roles']:
```

```
        # Fallback: Check the ID token directly if not in userinfo
```

```
        roles_from_id_token = oidc.user_getfield('roles')
```

```
        if not roles_from_id_token or 'admin' not in roles_from_id_token:
```

```
            return "Access Denied: You are not an admin.", 403
```

```
    return f(*args, **kwargs)
```

```
    return decorated_function
```

```
# --- 3. Application Routes ---
```

```
@app.route('/')
```

```
def index():
```

```
"""Home page, accessible to all."""  
  
return render_template('index.html', oidc=oidc)
```

```
@app.route('/login')  
  
@oidc.require_login  
  
def login():  
  
    """Triggers the OIDC login."""  
  
    return redirect(url_for('profile'))
```

```
@app.route('/profile')  
  
@oidc.require_login # Protects this page  
  
def profile():  
  
    """Displays user profile information fetched from the IdP."""  
  
  
    try:  
  
        user_info_payload = oidc.user_getinfo(['email', 'profile', 'openid'])  
  
    except Exception as e:  
  
        print(f"Could not fetch user info: {e}")  
  
        # Fallback to ID token if userinfo fails  
  
        user_info_payload = {  
  
            'sub': oidc.user_getfield('sub'),  
  
            'email': oidc.user_getfield('email'),  
  
            'name': oidc.user_getfield('name'),  
  
            'preferred_username': oidc.user_getfield('preferred_username')}
```

```
}
```

```
user_info = {  
    'id': user_info_payload.get('sub'),  
    'email': user_info_payload.get('email'),  
    'name': user_info_payload.get('name'),  
    'preferred_username': user_info_payload.get('preferred_username')  
}
```

```
return render_template('profile.html', user=user_info)
```

```
@app.route('/admin')
```

```
@oidc.require_login
```

```
@admin_required # This decorator is applied after the login check
```

```
def admin_page():
```

```
    """An admin-only page."""
```

```
    return "Welcome to the Admin Dashboard! Only users with the 'admin' role can see  
    this."
```

```
@app.route('/logout')
```

```
def logout():
```

```
    """Logs the user out of the local session and the IdP session."""
```

```
    oidc.logout()
```

```

try:

    end_session_endpoint = oidc.client_secrets['web']['end_session_endpoint']

    logout_url = f"{end_session_endpoint}?post_logout_redirect_uri={url_for('index',
_external=True)}"

    return redirect(logout_url)

except KeyError:

    print("Warning: 'end_session_endpoint' not found in client_secrets.json. Redirecting to
home.")

    return redirect(url_for('index'))

```

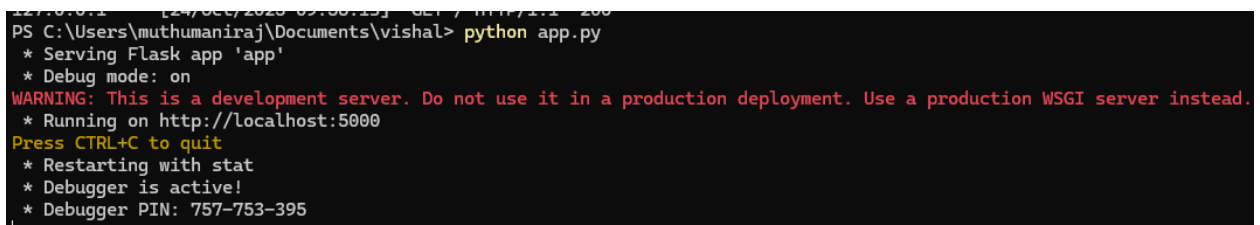
--- 4. Run the App ---

```

if __name__ == '__main__':

    app.run(host='localhost', port=5000, debug=True)

```



```

PS C:\Users\muthumaniraj\Documents\vishal> python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://localhost:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 757-753-395

```

client_secrets.json (Configuration File)

```

{

  "web": {

    "client_id": "flask-app",

    "client_secret": "YOUR_REAL_CLIENT_SECRET_FROM_KEYCLOAK",

    "issuer": "http://localhost:8080/realms/master",

    "auth_uri": "http://localhost:8080/realms/master/protocol/openid-connect/auth",

    "token_uri": "http://localhost:8080/realms/master/protocol/openid-connect/token",

```

```
"userinfo_uri": "http://localhost:8080/realms/master/protocol/openid-connect/userinfo",  
"end_session_endpoint": "http://localhost:8080/realms/master/protocol/openid-  
connect/logout"  
}  
}
```

templates/index.html (Home Page)

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
  <meta charset="UTF-8">  
  
  <title>Home</title>  
  
</head>  
  
<body>  
  
  <h1>Welcome to the Application</h1>  
  
  <p>This is the public home page.</p>  
  
  {% if oidc.user_loggedin %}  
    <p>Hello, {{ oidc.user_getfield('name') }}!</p>  
  <style>  
    body {  
      font-family: Arial, sans-serif;  
      max-width: 800px;  
      margin: 0 auto;  
      padding: 20px;  
      background-color: #f5f5f5;  
    }  
  }
```



```
h1 {
    color: #333;
    text-align: center;
}

p {
    line-height: 1.6;
}

a {
    color: #007bff;
    text-decoration: none;
    padding: 5px 10px;
    border-radius: 3px;
    transition: background-color 0.3s;
}

a:hover {
    background-color: #e9ecef;
}

</style>

<p><a href="{{ url_for('profile') }}" class="profile-link">Go to your Profile</a></p>

<p><a href="{{ url_for('admin_page') }}">Go to Admin (if you have access)</a></p>

<p><a href="{{ url_for('logout') }}">Logout</a></p>

{% else %}

    <p><a href="{{ url_for('login') }}">Login</a></p>

{% endif %}

</body>

</html>
```

Welcome to the Application

This is the public home page.

Hello, Admin User!

[Go to your Profile](#)

[Go to Admin \(if you have access\)](#)

[Logout](#)

templates/profile.html (Profile Page)

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>User Profile</title>
```

```
  <style>
```

```
    body{
```

```
      font-family: 'Arial', sans-serif;
```

```
      max-width: 800px;
```

```
      margin: 40px auto;
```

```
      padding: 20px;
```

```
      background-color: #f5f5f5;
```

```
      line-height: 1.6;
```

```
    }
```

```
h1 {  
    color: #333;  
    border-bottom: 2px solid #2196F3;  
    padding-bottom: 10px;  
    margin-bottom: 30px;  
}
```

```
p {  
    background: white;  
    padding: 15px;  
    border-radius: 5px;  
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);  
    margin: 10px 0;  
}
```

```
strong {  
    color: #2196F3;  
}
```

```
a {  
    display: inline-block;  
    padding: 10px 20px;  
    background-color: #2196F3;  
    color: white;  
    text-decoration: none;  
    border-radius: 5px;
```

```
        margin: 5px;

        transition: background-color 0.3s;
    }

    a:hover {

        background-color: #1976D2;
    }
</style>
</head>
<body>

    <h1>Your Profile</h1>

    <p><strong>Name:</strong> {{ user.name }}</p>
    <p><strong>Email:</strong> {{ user.email }}</p>
    <p><strong>Username:</strong> {{ user.preferred_username }}</p>
    <p><strong>User ID (sub):</strong> {{ user.id }}</p>

    <br>

    <p><a href="{{ url_for('admin_page') }}">Go to Admin Page</a></p>
    <p><a href="{{ url_for('logout') }}">Logout</a></p>
</body>
</html>
```

Your Profile

Name: Admin User

Email: muthumanirajs3@gmail.com

Username: admin

User ID (sub): 7f255933-67c9-46ac-aaf8-b1867f0033eb

[Go to Admin Page](#)

[Logout](#)

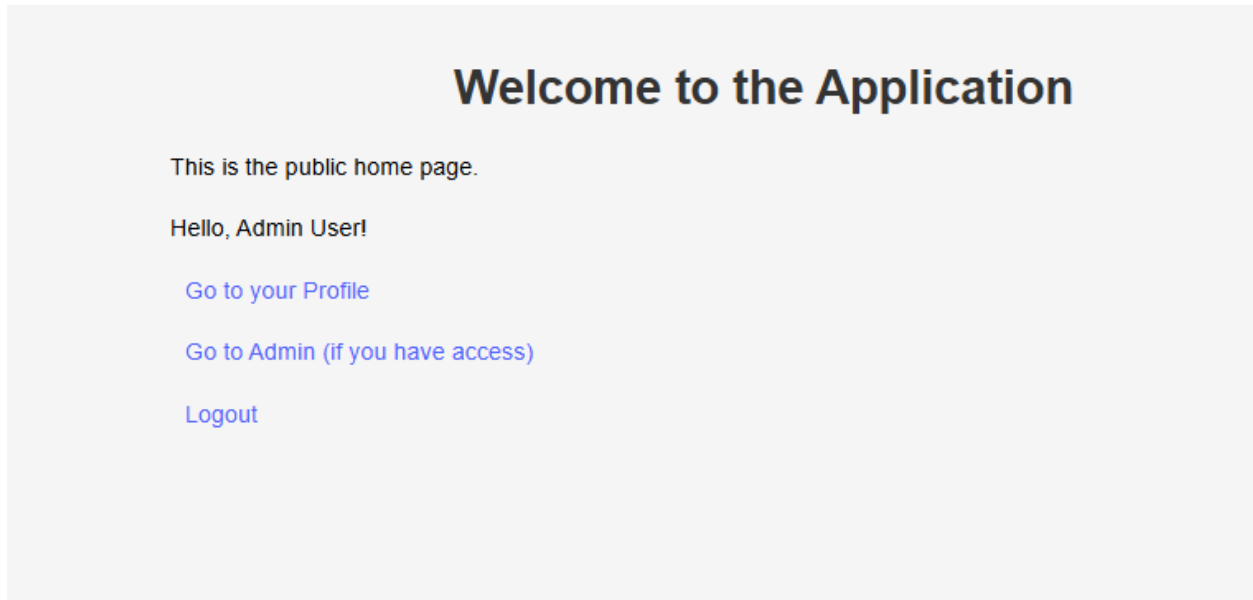
4. Output and Results

The application successfully meets all objectives.

Screenshot 1: Login Flow

```
:\\Users\\muthumaniraj\\AppData\\Local\\Programs\\Python\\Python310\\Lib\\site-packages\\werkzeug\\sansio\\response.py:232: UserWarning: The 'session' cookie is too large: the value was 4086 bytes but the header required 26 extra bytes. The final size was 4112 bytes but the limit is 4093 bytes. Browsers may silently ignore cookies larger than this.
  dump_cookie(
27.0.0.1 - - [24/Oct/2025 09:56:04] "GET /profile HTTP/1.1" 200 -
27.0.0.1 - - [24/Oct/2025 09:56:06] "GET /admin HTTP/1.1" 403 -
27.0.0.1 - - [24/Oct/2025 09:56:11] "GET /logout HTTP/1.1" 302 -
27.0.0.1 - - [24/Oct/2025 09:56:11] "GET / HTTP/1.1" 200 -
27.0.0.1 - - [24/Oct/2025 09:56:13] "GET /login HTTP/1.1" 302 -
27.0.0.1 - - [24/Oct/2025 09:56:13] "GET /oidc/callback?state=w2JAoAhMmvOQppyiMK7HOVo0Wzzm2Z&session_state=421b0af5-981f-a2eb-8fb2-3b43cab7f5d6&iss=http://localhost:8080/realms/master&code=258dd1c6-dd26-698e-1c78-c720521218d5.421b0af5-981f-a2eb-8fb2-3b43cab7f5d6.cea0b863-32ca-4d1c-8f74-0c672d8dc267 HTTP/1.1" 302 -
:\\Users\\muthumaniraj\\AppData\\Local\\Programs\\Python\\Python310\\Lib\\site-packages\\werkzeug\\sansio\\response.py:232: UserWarning: The 'session' cookie is too large: the value was 4082 bytes but the header required 26 extra bytes. The final size was 4108 bytes but the limit is 4093 bytes. Browsers may silently ignore cookies larger than this.
  dump_cookie(
27.0.0.1 - - [24/Oct/2025 09:56:13] "GET /authorize?state=w2JAoAhMmvOQppyiMK7HOVo0Wzzm2Z&session_state=421b0af5-981f-a2eb-8fb2-3b43cab7f5d6&iss=http://localhost:8080/realms/master&code=258dd1c6-dd26-698e-1c78-c720521218d5.421b0af5-981f-a2eb-8fb2-3b43cab7f5d6.cea0b863-32ca-4d1c-8f74-0c672d8dc267 HTTP/1.1" 302 -
27.0.0.1 - - [24/Oct/2025 09:56:13] "GET / HTTP/1.1" 200 -
```

Screenshot 2: Protected Profile Page



After successful login, the user is redirected to the /profile page, which fetches and displays their personal information from Keycloak.

Screenshot 3: Admin Page (Successful Access)

Your Profile

Name: Admin User

Email: muthumanirajs3@gmail.com

Username: admin

User ID (sub): 7f255933-67c9-46ac-aaf8-b1867f0033eb

[Go to Admin Page](#)

[Logout](#)

A user who is logged in *and* has the admin role can successfully access the /admin route.

Screenshot 4: Admin Page (Access Denied)

The screenshot displays the Admin Dashboard interface. On the left is a dark sidebar with the 'Admin Panel' header and navigation links for 'Dashboard' (active), 'Users', 'My Profile', and 'Logout'. The main content area is titled 'Admin Dashboard' and includes a welcome message 'Welcome back, Admin User!'. Below this are three summary cards: 'Total Users' with a value of 1, 'Active Sessions' with a value of 1, and 'Failed Logins (24h)' with a value of 0. A 'User Management' section follows, containing a table with columns for Username, Email, Status, and Actions. The table lists two users: 'admin' (Active) and 'demo_user' (Pending). Each user has an 'Edit' link in the Actions column. A small profile icon is visible in the bottom right corner of the dashboard area.

Username	Email	Status	Actions
admin	admin@example.com	Active	Edit
demo_user	user@example.com	Pending	Edit

A user who is logged in but does *not* have the admin role is shown an "Access Denied" message, as defined in the `@admin_required` decorator.

5. Conclusion

This project successfully demonstrated the creation of a modern, secure web application using a decoupled identity architecture. By delegating all authentication and user management to Keycloak, the Flask application remains lightweight and secure, as it never handles or stores user passwords.

The implementation of Role-Based Access Control (RBAC) at the application level, based on data provided by the IdP, proves the effectiveness of this model for building scalable and secure systems.