Sunday, November 12, 2023

Integrating Keycloak with Django

> *"1-Introduction keycloak application*
> *2-Django authentication*
> *3- Keycloak with Django Rest Framework (DRF)*
> *4- Django Authentication and Authorization*
> *5- Middleware OpenID connect"*

Keycloak Introduction:

Keycloak is an open-source identity and access management solution developed by Red Hat. It provides a powerful and flexible set of features for securing applications and services, including single sign-on (SSO), user authentication, authorization, and identity brokering. Keycloak is designed to simplify the implementation of identity management and security mechanisms in modern applications.

Key Features:

Keycloak enables users to log in once and access multiple applications without the need to log in again. This enhances the user experience and simplifies authentication.

Identity Federation:

Supports identity federation, allowing you to integrate Keycloak with external identity providers (e.g., Google, Facebook, LDAP) for a unified authentication experience.

User Authentication and Management:

Provides a robust user authentication system with support for multi-factor authentication. Allows administrators to manage users, define user attributes, and enforce password policies.

Authorization and Permissions:

Implements a flexible and fine-grained authorization system. Administrators can define roles and permissions, and applications can enforce access control based on these roles and permissions.

Social Login:

Supports social login, allowing users to log in using their existing accounts on platforms like Google, Facebook, or Twitter.

OpenID Connect and OAuth 2.0:

Keycloak is built on OpenID Connect and OAuth 2.0 standards, providing secure and industry-standard protocols for authentication and authorization.

Security and Compliance:

Offers features to enhance the security of applications, including secure token handling, SSL/TLS support, and compliance with security standards.

User Self-Registration and Account Management:

Enables users to register themselves and manage their accounts, reducing administrative overhead.

Customizable Themes:

Allows customization of the Keycloak login page and other UI elements to match the branding of your applications.

Cross-Platform Support:

Keycloak is platform-agnostic and can be integrated with applications developed in various programming languages and frameworks.

Extensibility:

Use Cases:

Enterprise Authentication:

Centralized authentication and authorization for enterprise applications.

Microservices Security:

Secure access to microservices architectures, ensuring that only authorized users and services can interact.

API Security:

Protects APIs by validating and enforcing OAuth 2.0 access tokens.

Cloud-Native Applications:

Ideal for securing modern, cloud-native applications and services.

Internet of Things (IoT):

Provides identity and access management for IoT devices and applications.

Keycloak is a versatile identity and access management solution suitable for a wide range of applications and industries. Its open-source nature and comprehensive feature set make it a popular choice for developers and organizations seeking a robust and scalable identity solution.

Django authentication

Integrating Django with Keycloak involves several steps, from setting up Keycloak and configuring Django to implementing in your application. Below is a detailed flow for integrating Django with Keycloak:

1. Install Required Packages

Install Django and the python-keycloak library:

```
# pip install Django python-keycloak
```

2. Create a Django Project and App

Create a new Django project and app:

```
# pip install Django python -keycloak
```

3. Create a Django Project and App

x

```
# python manage.py startapp myapp
```

3. Configure Database

Configure your database settings in settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
```

Apply migration:

```
# python manage.py migrate
```

4. Install and Set Up Keycloak

Install and set up Keycloak by following the Keycloak Installation Guide.

https://www.keycloak.org/

5. Create a Realm in Keycloak

Log in to the Keycloak admin console.

Create a new realm for your Django application.

6. Configure Keycloak Client

Create a **new client** in Keycloak for your Django application.

Note down the **client ID** and secret.

7. Integrate Keycloak with Django

Install the **python-keycloak library**:

```
# pip install python-keycloak
```

Create a keycloak.json file with your client configuration:

```
{
    "realm": "your-realm",
    "auth-server-url": "http://keycloak-server/auth"
```

x

```
        "secret": "your-client-secret"
    },
    "confidential-port": 0
}
```

8. Django Middleware Configuration

Configure Django middleware to handle Keycloak authentication. Add the following to your MIDDLEWARE setting in settings.py:

```
# MIDDLEWARE = [
    # ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'keycloak_oidc.middleware.OIDCMiddleware',
    # ...
]
```

9. Protected Views with Keycloak Use the keycloak protect decorator to protect specific views in your Django app:

```
from keycloak_oidc.decorators import keycloak_protect
@keycloak_protect()
def protected_view(request):
    # Your view logic here
```

10. Enable Social Authentication

Install the social-auth-app-django library:

```
# pip install social-auth-app-django
```

Configure settings for social authentication in settings.py.

11. Testing

Test your Django application by running the development server and accessing protected views:

```
# python manage.py runserver
```

12. Additional Features

Explore additional features such as role-based access control, user profile synchronization, etc.

This flow provides a comprehensive guide for integrating Django with Keycloak. Keep in mind that each step may require additional configuration and adjustments based on your specific project requirements and Keycloak setup. Refer to the official documentation for Django, python-keycloak, and social-auth-app-django for more

x

PART 2 — Django Rest Framework (DRF)

Integrating Keycloak with Django Rest Framework (DRF) for custom authentication involves creating a custom authentication class and configuring your Django project to use it. Below is a step-by-step guide on how to achieve this:

1. Install Required Packages

Make sure you have Django, Django Rest Framework, and python-keycloak installed:

```
# pip install django djangorestframework python-keycloak
```

2. Set Up a Django Project and App

```
# django-admin startproject myproject
# cd myproject
# python manage.py startapp myapp
```

3. Configure Django Settings

Update your INSTALLED_APPS in settings.py to include rest_framework and your custom app:

```
INSTALLED_APPS = [
    # ...
    'rest_framework',
    'myapp',
]
```

4. Create a Custom Authentication Class

Create a custom authentication class in your myapp/authentication.py:

```
from rest_framework.authentication import BaseAuthentication
from keycloak_oidc import KeycloakOIDC

class KeycloakAuthentication(BaseAuthentication):
    def authenticate(self, request):
        # Your authentication logic here using python-keycloak
        # Example:
        token = request.META.get('HTTP_AUTHORIZATION', '').split('Bearer ')[-1]
        keycloak = KeycloakOIDC(server_url='http://keycloak-server/auth',
                                client_id='your-client-id',
                                realm_name='your-realm')

        user_info = keycloak.decode_token(token)
```

x

```
        return user, None
```

5. Configure Django Rest Framework

In your myapp/settings.py, configure Django Rest Framework to use your custom authentication class:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'myapp.authentication.KeycloakAuthentication',
    ],
}
```

6. Use the Custom Authentication in Views In your DRF views, use the Keycloak Authentication class as the authentication class:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated

class MyAPIView(APIView):
    authentication_classes = [KeycloakAuthentication]
    permission_classes = [IsAuthenticated]

    def get(self, request):
        return Response({'message': 'Authenticated successfully!'})
```

7. Test Your Authentication

Run your Django development server:

```
# python manage.py runserver
```

Access your protected endpoint with a request containing a valid Keycloak token.

> *"Note: This is a basic example, and you might need to adapt it to your specific Keycloak configuration and Django project structure. Ensure you handle token validation, user creation or retrieval, and other aspects according to your application requirements. This guide provides a foundation for integrating Keycloak with Django Rest Framework using a custom authentication class. Adjustments may be needed based on your specific use case and requirements."*

PART3-Django Authentication and Authorization

x

1. Install Required Packages

Ensure you have Django, python-keycloak, and any other necessary packages installed:

```
# pip install django python-keycloak
```

2. Set Up a Django Project and App

Create a new Django project and app:

```
# django-admin startproject myproject
# cd myproject
# python manage.py startapp myapp
```

3. Configure Django Settings

Update your INSTALLED_APPS in settings.py to include myapp and configure the database:

```
INSTALLED_APPS = [
    # ...
    'myapp',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # ...
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
```

4. Install and Set Up Keycloak

> *“Install and set up Keycloak by following the official documentation.”*

5. Create a Keycloak Realm and Client

Log in to the Keycloak **admin console**.

Create a new realm for your Django application.

x

Add Keycloak settings to your Django settings.py:

```
# settings.py

KEYCLOAK_SERVER_URL = 'http://keycloak-server/auth'
KEYCLOAK_REALM = 'your-realm'
KEYCLOAK_CLIENT_ID = 'your-client-id'
KEYCLOAK_CLIENT_SECRET = 'your-client-secret'
```

7. Keycloak Authentication Backend Create a file myapp/keycloak backend.pay for the authentication backend:

```
# keycloak_backend.py

from keycloak_oidc.backends import OIDCAuthenticationBackend

class KeycloakBackend(OIDCAuthenticationBackend):
    pass
```

8. Django Authentication Backend Configuration

Add your custom authentication backend to the AUTHENTICATION_BACKENDS setting in settings.py:

```
# settings.py
AUTHENTICATION_BACKENDS = [
    'myapp.keycloak_backend.KeycloakBackend',
    'django.contrib.auth.backends.ModelBackend',
]
```

9. Middleware Configuration Add the Keycloak Middleware to your middleware in settings.py:

```
# settings.py

MIDDLEWARE = [
    # ...
    'keycloak_oidc.middleware.OIDCMiddleware',
    # ...
]
```

10. Create a Superuser

Create a superuser for the Django admin:

```
# python manage.py createsuperuser
```

x

```
# python manage.py runserver
```

Access the Django admin at http://localhost:8000/admin and log in with the superuser credentials.

12. Protect Views with Keycloak

Use the keycloak_protect decorator to protect specific views:

```
# views.py

from keycloak_oidc.decorators import keycloak_protect
from django.http import HttpResponse

@keycloak_protect()
def protected_view(request):
    return HttpResponse('This is a protected view!')
```

PART 4- OAuth 2.0

Securing a Django app with OAuth 2.0 and Keycloak involves configuring your Django project to use Keycloak as an identity provider. Here are the general steps to achieve this:

1. Install Required Packages

Install the necessary Django packages for OAuth 2.0 integration:

```
# pip install social-auth-app-django python-keycloak
```

2. Set Up Keycloak

Install and configure Keycloak as per the official documentation or below link

https://medium.com/@ozbillwang/run-keycloak-locally-with-docker-compose-db9a9f2fb437

Create a Keycloak realm and client for your Django app.

3. Configure Django Settings

In your Django project's settings.py, configure the authentication backends and social authentication settings:

```
# settings.py

INSTALLED_APPS = [
    # ...
    'social_django',
    # ...
```

x

```
    'social_core.backends.keycloak.KeycloakOAuth2',
    'django.contrib.auth.backends.ModelBackend',
)

SOCIAL_AUTH_KEYCLOAK_KEY = 'your-client-id'
SOCIAL_AUTH_KEYCLOAK_SECRET = 'your-client-secret'
SOCIAL_AUTH_KEYCLOAK_BASE_URL = 'http://keycloak-server/auth/realms/your-realm'
LOGIN_URL = 'login'
LOGOUT_URL = 'logout'
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
```

Replace 'your-client-id', 'your-client-secret', and 'your-realm' with your Keycloak client information.

4. Configure Django URLs

Include the social-auth URLs in your Django project's urls.py:

```
# urls.py

from django.urls import path, include

urlpatterns = [
    # ...
    path('auth/', include('social_django.urls', namespace='social')),
    # ...
]
```

5. Configure Keycloak Client

In the Keycloak admin console, configure the client settings for your Django app.

Set the "Valid Redirect URIs" to include your Django development server URL (e.g., http://localhost:8000/auth/complete/keycloak/)..)

6. Test the Integration

Run your Django development server:

```
# python manage.py runserver
```

Access your Django app, and you should see a "Login with Keycloak" option. Clicking on it should redirect you to the Keycloak login page.

7. Further Customization

You can customize the login page, handle user data synchronization, and manage permissions based on your

x

Remember to adapt the configurations based on your specific Django project structure and Keycloak setup. Always follow best practices for securing OAuth 2.0 integrations and keep your Django and Keycloak instances updated with the latest security patches.

PART5- Middleware OIDC

Adding Keycloak **middleware** to Django involves using the Keycloak Middleware provided by the keycloak_OIDC package. This middleware helps in handling aspects of the authentication process, such as obtaining user information from the ID token. Here are the steps to add Keycloak middleware to your Django project:

1. Install keycloak_oidc

Ensure you have the keycloak_oidc package installed. You can install it using:

```
# pip install keycloak-oidc
```

2. Configure Django Settings

In your Django project's settings.py, configure the Keycloak settings under the KEYCLOAK_AUTH dictionary:

```
# settings.py

INSTALLED_APPS = [
    # ...
    'keycloak_oidc',
    'yourapp',
    # ...
]

KEYCLOAK_AUTH = {
    'REALM': 'your-realm',
    'CLIENT_ID': 'your-client-id',
    'CLIENT_SECRET': 'your-client-secret',
    'OIDC_ENDPOINT': 'http://keycloak-server/auth/realms/your-realm',
}
```

Make sure to replace 'your-realm', 'your-client-id', and 'your-client-secret' with your Keycloak realm, client ID, and client secret.

3. Add Keycloak Middleware

Include the KeycloakMiddleware in your Django project's middleware configuration:

x

```
    'keycloak_oidc.middleware.OIDCMiddleware',
    # ...
]
```

This middleware is responsible for handling authentication-related tasks during the request-response cycle.

4. (Optional) Configure Additional Middleware

Depending on your specific requirements, you might need to configure additional middleware. For example, if you want to enable Django's authentication middleware alongside Keycloak, you can include it in the MIDDLEWARE setting:

```
# settings.py

MIDDLEWARE = [
    # ...
    'keycloak_oidc.middleware.OIDCMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    # ...
]
```

5. Protect Views with Keycloak

Use the keycloak_protect decorator to protect specific views in your Django app:

```
# views.py

from keycloak_oidc.decorators import keycloak_protect
from django.http import HttpResponse

@keycloak_protect()
def protected_view(request):
    return HttpResponse('This is a protected view!')
```

This decorator ensures that the user is authenticated with Keycloak before accessing the view.

6. Test Your Setup

Run your Django development server:

```
# python manage.py runserver
```

Access the protected view, and Keycloak will handle the authentication process.

Remember to customize the configuration based on your Keycloak setup and Django project requirements. The provided steps give you a basic integration of Keycloak middleware with Django, but you might need to adjust

x

Discuss on Twitter • View on GitHub

Load Comments

TAGS

KEYCLOAK DJANGO SEURITY DEVOPS CODING PYTHON PROGRAMMING

PREVIOUS ARTICLE
Git Workflow: Best Practices for a Smooth and Efficient Development Process

NEXT ARTICLE
The Art of State Management in Swift for iOS

← Back to the blog

x