Human Brain Project

EBRAINS

Collaboratory OIDC clients

Presentation and Q&A

Simulate with EBRAINS

7-10 November 2022, online

Paul Chaney

Co-funded by the European Union

What is a Collaboratory OIDC Client?

- As a service developer, if your service needs to authenticate its users, that authentication is done centrally by the Collaboratory IAM service. IAM is based on the Keycloak software.
- Communication between your service and IAM uses the OpenID Connect (OIDC) protocol.
- Services need to create an OIDC client to communicate with IAM.
- Process:
  - Your service requests from IAM whether a user is authenticated.
  - If the user is not authenticated, IAM will request that the user logs in.
  - IAM then responds indicating whether the user is authenticated, and if so, whether they are authorized to used your service.
  - Additionally, your service can request access to specific scopes of information about the user.

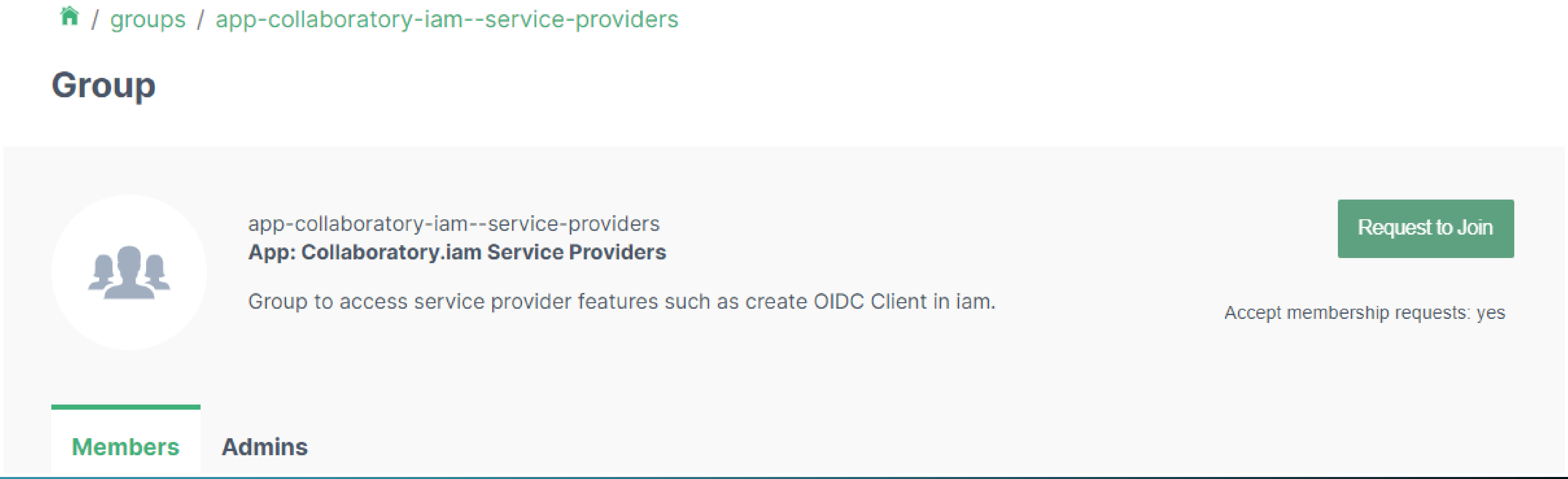
Manage your OIDC clients

- Documentation: How to register an OIDC Client

- Lab: Notebook to create and manage OIDC client

Requirement

- Service developers need permission to create an OIDC client. Permission is only granted to users in the following Group:
  - https://wiki.ebrains.eu/bin/view/Identity/#/groups/app-collaboratory-iam--service-providers
- Developers can browse to this page and click "Request to Join" to get this permission. Support may request additional information and/or motivation



app-collaboratory-iam--service-providers
App: Collaboratory.iam Service Providers

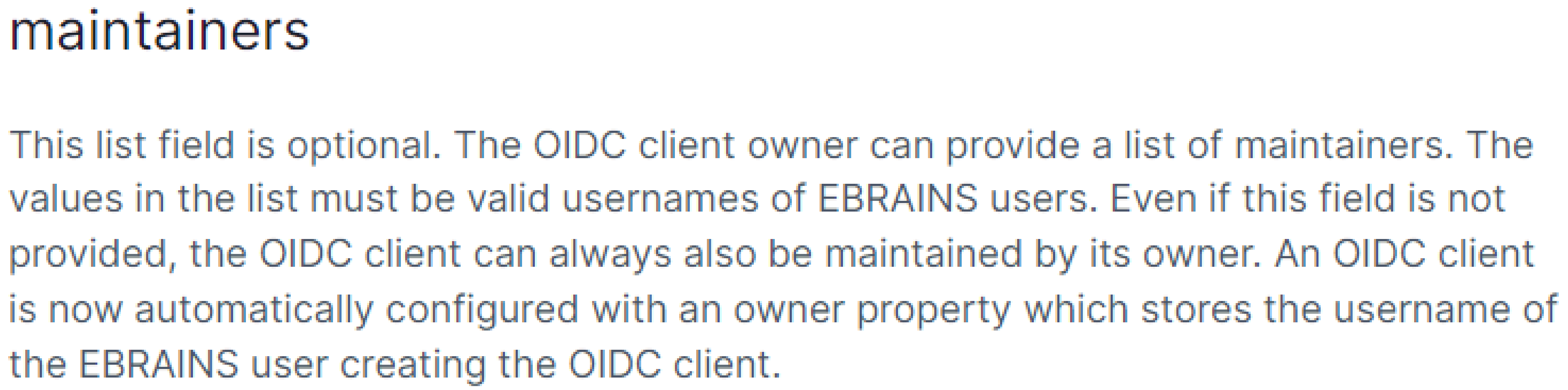
Group to access service provider features such as create OIDC Client in iam.

Accept membership requests: yes

- Documentation in the collab "The Collaboratory"
- How to register an OIDC client

```
response = requests.post(
    API,
    headers={'Authorization': 'Bearer %s' % bearer_token},
    json={
        "client": {
            "clientId": clientId,
            "name": "Tutorial OIDC API",
            "description": "A sample client demo for the OIDC API documentation",
            "rootUrl": "https://example.org",
            "baseUrl": "https://example.org",
            "redirectUris": [
                "https://example.org/login/*"
            ],
            "bearerOnly": False,
            "consentRequired": True,
            "standardFlowEnabled": True,
            "implicitFlowEnabled": False,
            "directAccessGrantsEnabled": False,
            "attributes": {
                "contacts": "first.contact@example.com; second.contact@example.com"
            },
            "defaultClientScopes": ["openid", "email"],
            "optionalClientScopes": ["profile","team","group"]
        },
        "maintainers": ["messines","bougault"],
        "featureAuthenticate" : False,
        "accessDeniedToGuests" : True
    })
```

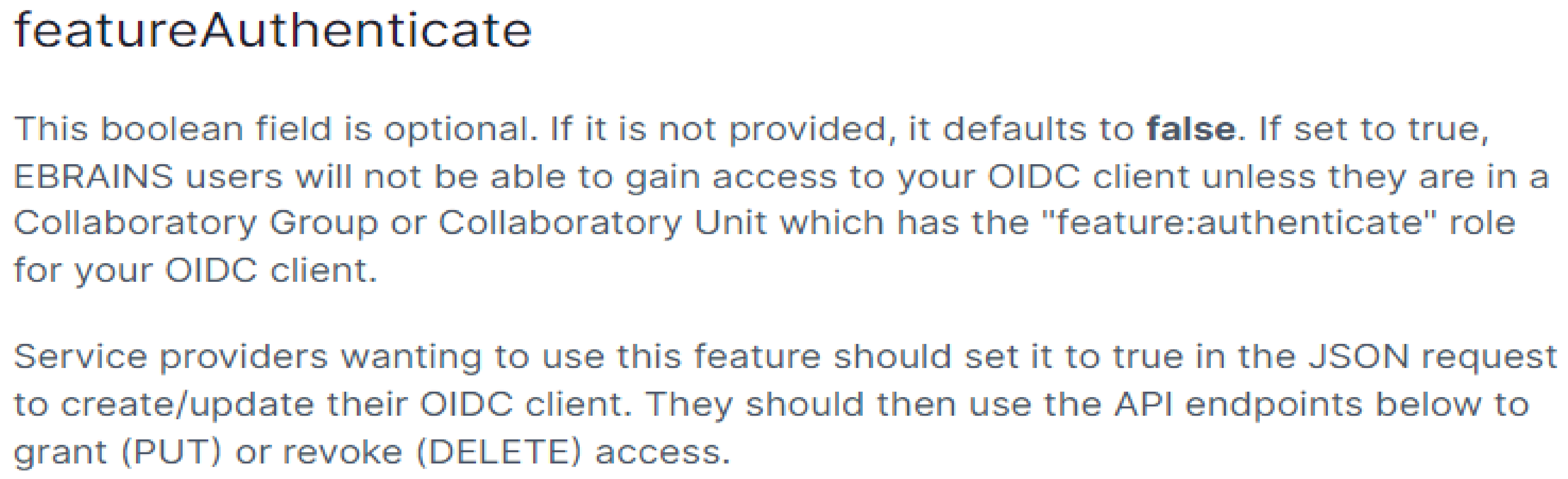
- You can now add "Maintainers" to your OIDC clients.
  - Allows other users to edit your OIDC clients.



maintainers

This list field is optional. The OIDC client owner can provide a list of maintainers. The values in the list must be valid usernames of EBRAINS users. Even if this field is not provided, the OIDC client can always also be maintained by its owner. An OIDC client is now automatically configured with an owner property which stores the username of the EBRAINS user creating the OIDC client.

- Manage who can access your OIDC clients.
  - feature:authenticate is an optional value to grant access to your OIDC client
  - By default, your client will be accessible to EBRAINS users.
  - If feature:authenticate is true, then you can specify which Groups/Units of users are authorized to access your OIDC client

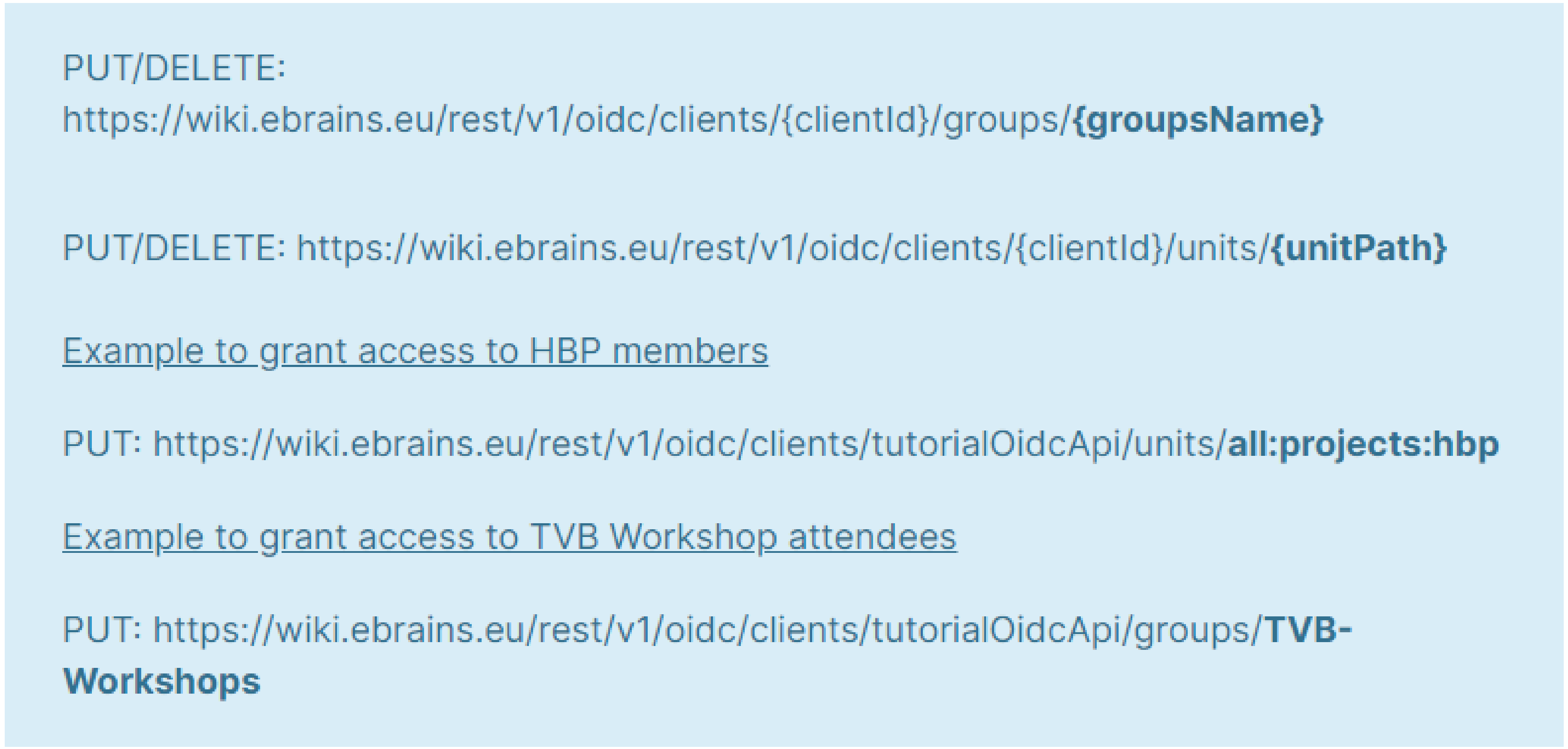


featureAuthenticate

This boolean field is optional. If it is not provided, it defaults to **false**. If set to true, EBRAINS users will not be able to gain access to your OIDC client unless they are in a Collaboratory Group or Collaboratory Unit which has the "feature:authenticate" role for your OIDC client.

Service providers wanting to use this feature should set it to true in the JSON request to create/update their OIDC client. They should then use the API endpoints below to grant (PUT) or revoke (DELETE) access.

- Allow a Group or Unit to access your client via the Collaboratory Wiki API



PUT/DELETE:
https://wiki.ebrains.eu/rest/v1/oidc/clients/{clientId}/groups/**{groupsName}**

PUT/DELETE: https://wiki.ebrains.eu/rest/v1/oidc/clients/{clientId}/units/**{unitPath}**

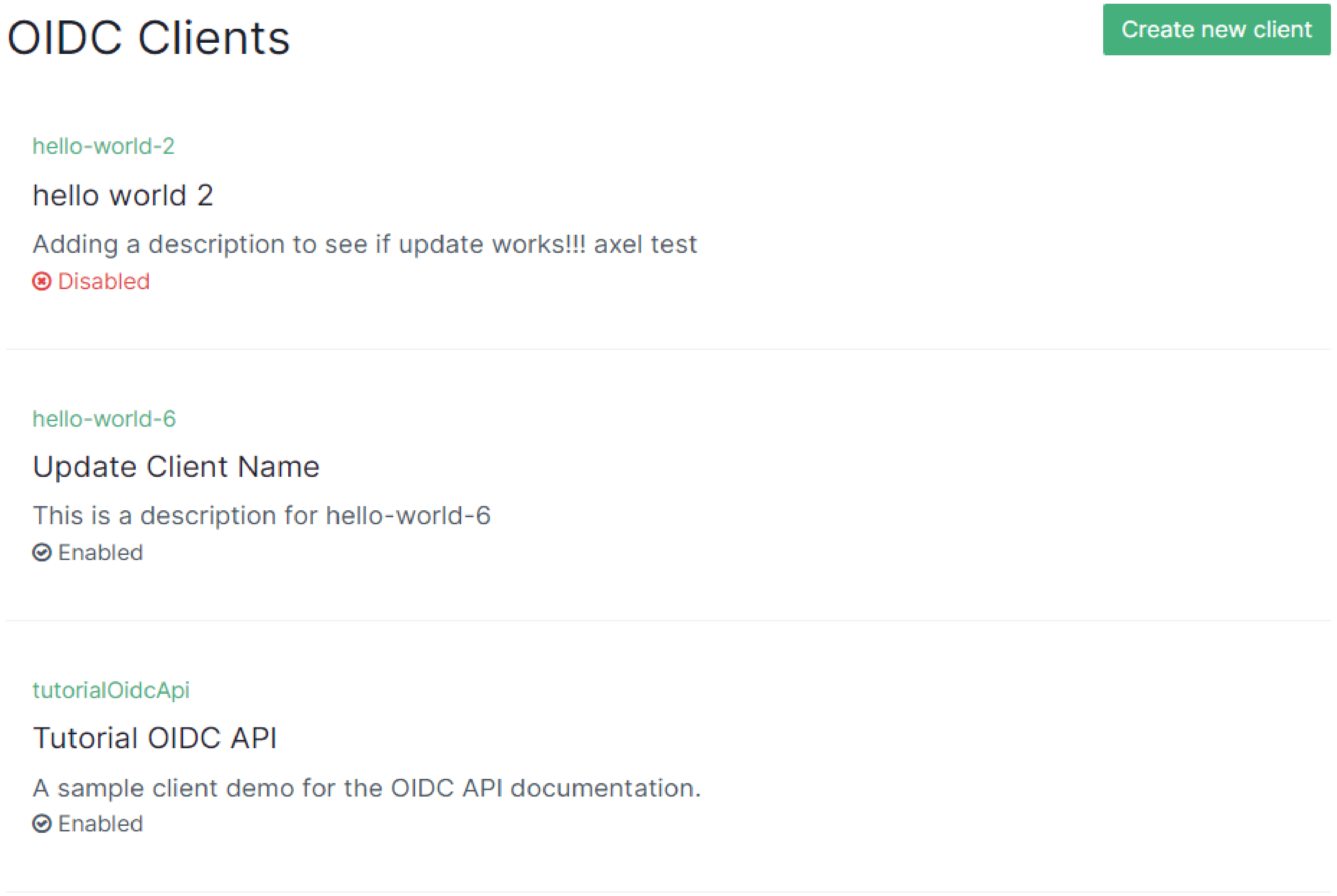
Example to grant access to HBP members

PUT: https://wiki.ebrains.eu/rest/v1/oidc/clients/tutorialOidcApi/units/**all:projects:hbp**

Example to grant access to TVB Workshop attendees

PUT: https://wiki.ebrains.eu/rest/v1/oidc/clients/tutorialOidcApi/groups/**TVB-Workshops**

- Much easier way to manage your OIDC clients



OIDC Clients

hello-world-2

hello world 2

Adding a description to see if update works!!! axel test

Disabled

hello-world-6

Update Client Name

This is a description for hello-world-6

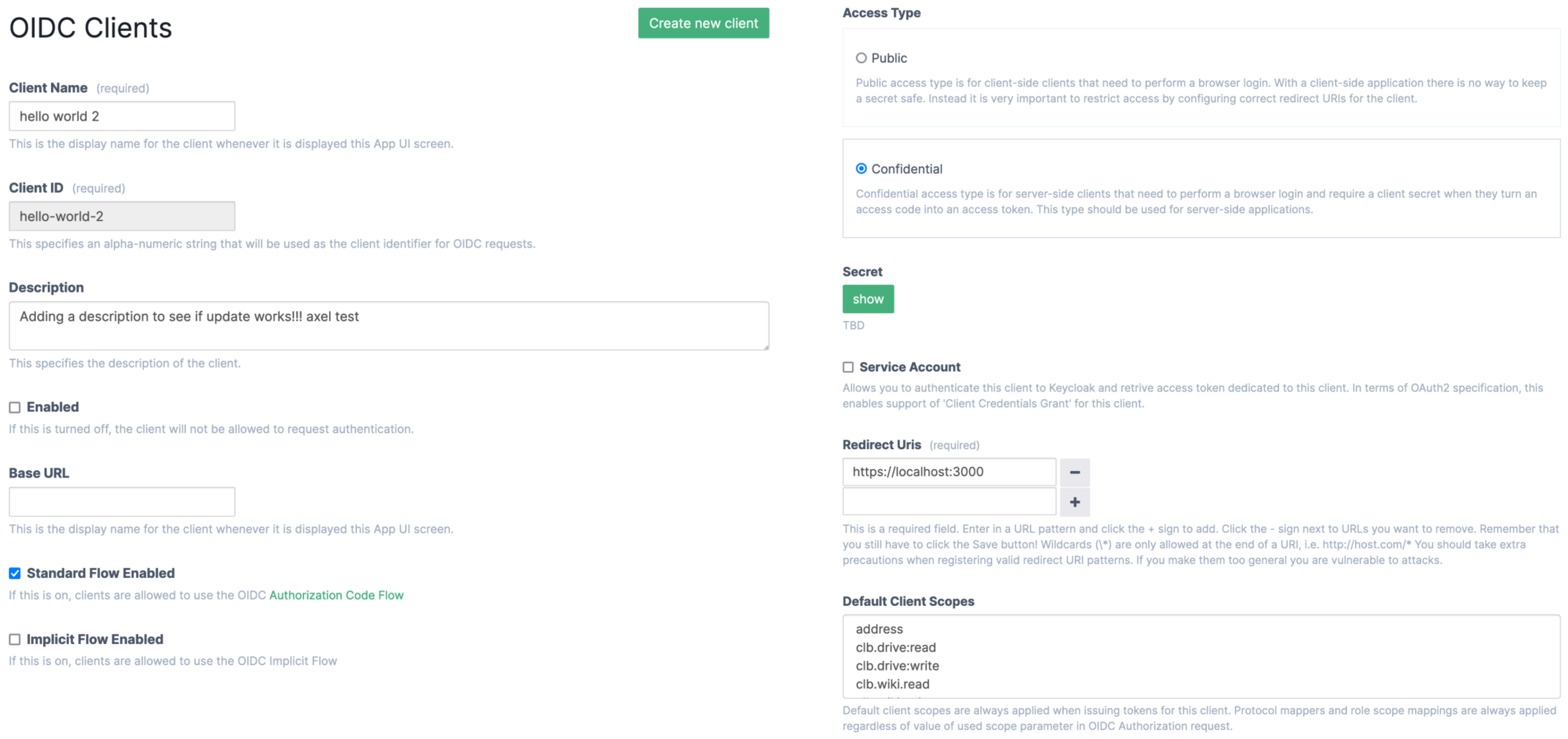
Enabled

tutorialOidcApi

Tutorial OIDC API

A sample client demo for the OIDC API documentation.

Enabled



OIDC Clients

Create new client

Client Name (required)

hello world 2

This is the display name for the client whenever it is displayed this App UI screen.

Client ID (required)

hello-world-2

This specifies an alpha-numeric string that will be used as the client identifier for OIDC requests.

Description

Adding a description to see if update works!!! axel test

This specifies the description of the client.

☐ **Enabled**

If this is turned off, the client will not be allowed to request authentication.

Base URL

This is the display name for the client whenever it is displayed this App UI screen.

☑ **Standard Flow Enabled**

If this is on, clients are allowed to use the OIDC Authorization Code Flow

☐ **Implicit Flow Enabled**

If this is on, clients are allowed to use the OIDC Implicit Flow

Access Type

○ Public

Public access type is for client-side clients that need to perform a browser login. With a client-side application there is no way to keep a secret safe. Instead it is very important to restrict access by configuring correct redirect URIs for the client.

● Confidential

Confidential access type is for server-side clients that need to perform a browser login and require a client secret when they turn an access code into an access token. This type should be used for server-side applications.

Secret

show

TBD

☐ **Service Account**

Allows you to authenticate this client to Keycloak and retrive access token dedicated to this client. In terms of OAuth2 specification, this enables support of 'Client Credentials Grant' for this client.

Redirect Uris (required)

https://localhost:3000 −

+

This is a required field. Enter in a URL pattern and click the + sign to add. Click the - sign next to URLs you want to remove. Remember that you still have to click the Save button! Wildcards (*) are only allowed at the end of a URI, i.e. http://host.com/* You should take extra precautions when registering valid redirect URI patterns. If you make them too general you are vulnerable to attacks.

Default Client Scopes

address
clb.drive:read
clb.drive:write
clb.wiki.read

Default client scopes are always applied when issuing tokens for this client. Protocol mappers and role scope mappings are always applied regardless of value of used scope parameter in OIDC Authorization request.

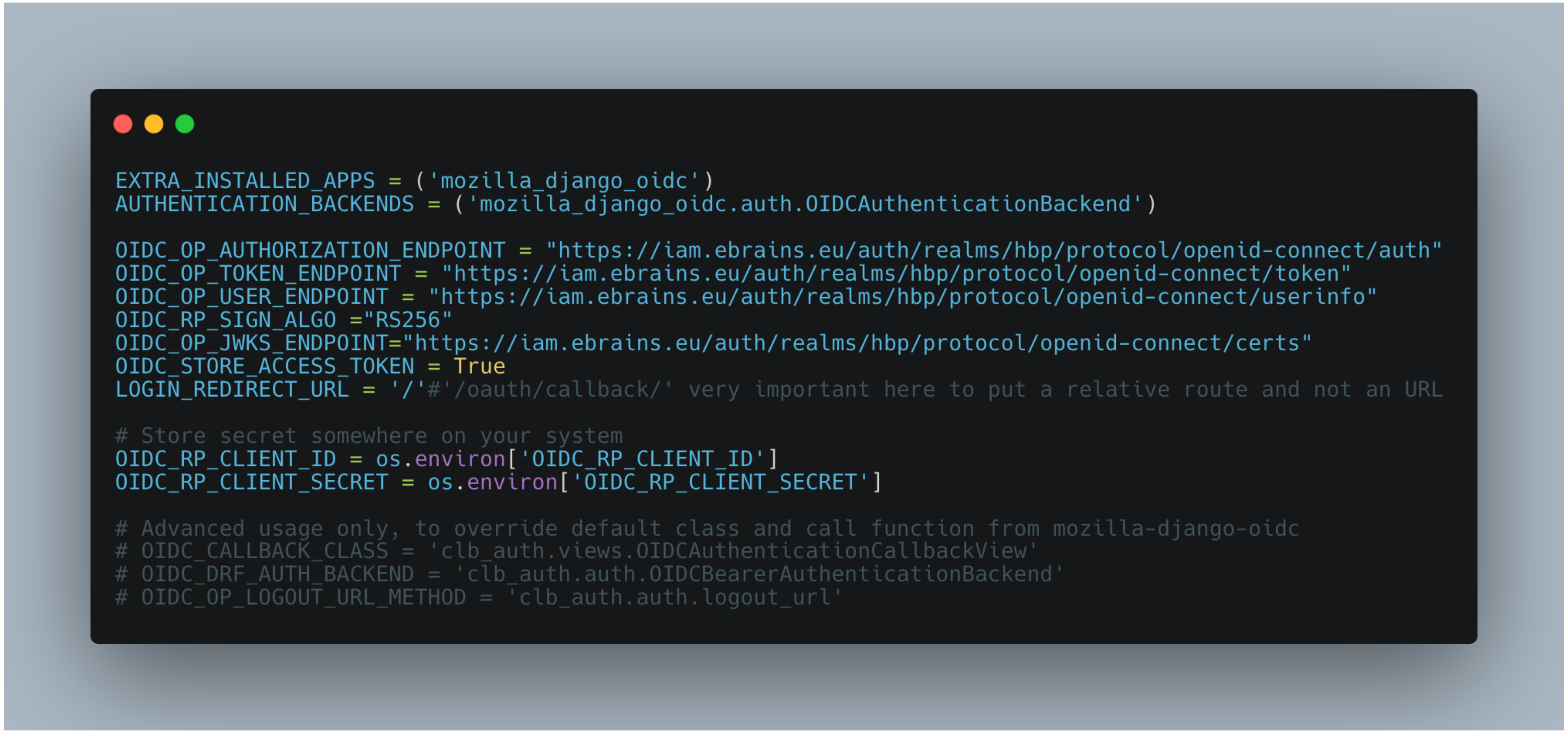
- Instructions on how to authenticate with your OIDC client is detailed in our [documentation]

- Every OIDC endpoint available to users is available to view here in JSON format:
  - https://iam.ebrains.eu/auth/realms/hbp/.well-known/openid-configuration

- For Django-based applications, please see mozilla-django-OIDC documentation:
  - https://mozilla-django-oidc.readthedocs.io/en/stable/installation.html

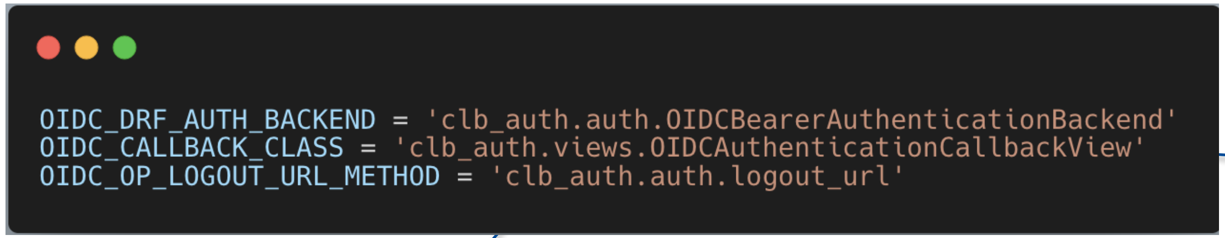
- For Flask-based applications, please see flask-OIDC:
  - https://flask-oidc.readthedocs.io/en/latest/

- In settings.py, configure the OIDC plugin

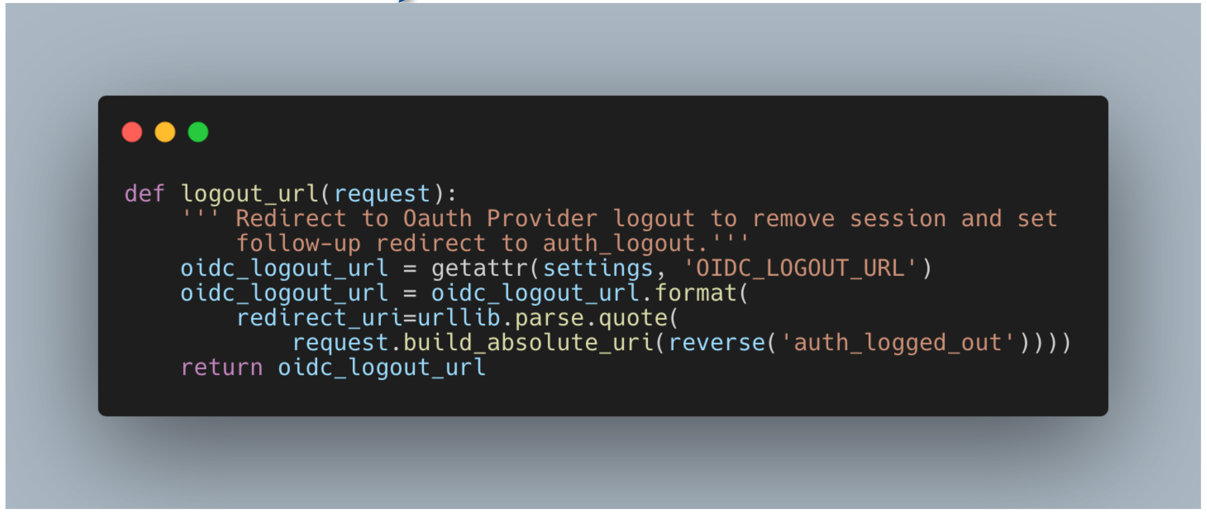


```
EXTRA_INSTALLED_APPS = ('mozilla_django_oidc')
AUTHENTICATION_BACKENDS = ('mozilla_django_oidc.auth.OIDCAuthenticationBackend')

OIDC_OP_AUTHORIZATION_ENDPOINT = "https://iam.ebrains.eu/auth/realms/hbp/protocol/openid-connect/auth"
OIDC_OP_TOKEN_ENDPOINT = "https://iam.ebrains.eu/auth/realms/hbp/protocol/openid-connect/token"
OIDC_OP_USER_ENDPOINT = "https://iam.ebrains.eu/auth/realms/hbp/protocol/openid-connect/userinfo"
OIDC_RP_SIGN_ALGO ="RS256"
OIDC_OP_JWKS_ENDPOINT="https://iam.ebrains.eu/auth/realms/hbp/protocol/openid-connect/certs"
OIDC_STORE_ACCESS_TOKEN = True
LOGIN_REDIRECT_URL = '/'#'/oauth/callback/' very important here to put a relative route and not an URL

# Store secret somewhere on your system
OIDC_RP_CLIENT_ID = os.environ['OIDC_RP_CLIENT_ID']
OIDC_RP_CLIENT_SECRET = os.environ['OIDC_RP_CLIENT_SECRET']

# Advanced usage only, to override default class and call function from mozilla-django-oidc
# OIDC_CALLBACK_CLASS = 'clb_auth.views.OIDCAuthenticationCallbackView'
# OIDC_DRF_AUTH_BACKEND = 'clb_auth.auth.OIDCBearerAuthenticationBackend'
# OIDC_OP_LOGOUT_URL_METHOD = 'clb_auth.auth.logout_url'
```

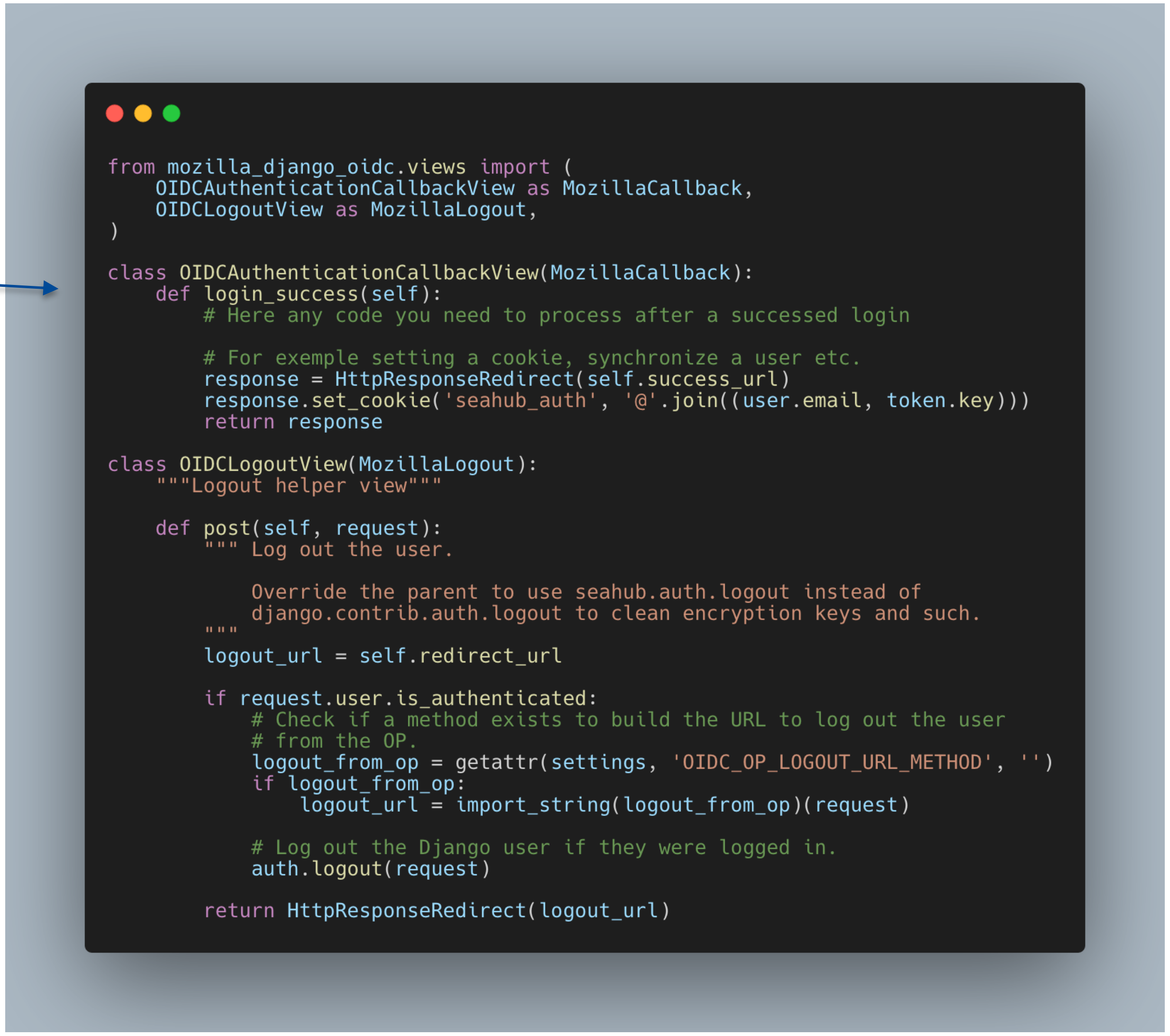
More advanced usage, override Mozilla library



```python
OIDC_DRF_AUTH_BACKEND = 'clb_auth.auth.OIDCBearerAuthenticationBackend'
OIDC_CALLBACK_CLASS = 'clb_auth.views.OIDCAuthenticationCallbackView'
OIDC_OP_LOGOUT_URL_METHOD = 'clb_auth.auth.logout_url'
```



```python
def logout_url(request):
    ''' Redirect to Oauth Provider logout to remove session and set
        follow-up redirect to auth_logout.'''
    oidc_logout_url = getattr(settings, 'OIDC_LOGOUT_URL')
    oidc_logout_url = oidc_logout_url.format(
        redirect_uri=urllib.parse.quote(
            request.build_absolute_uri(reverse('auth_logged_out'))))
    return oidc_logout_url
```



```python
from mozilla_django_oidc.views import (
    OIDCAuthenticationCallbackView as MozillaCallback,
    OIDCLogoutView as MozillaLogout,
)

class OIDCAuthenticationCallbackView(MozillaCallback):
    def login_success(self):
        # Here any code you need to process after a successed login

        # For exemple setting a cookie, synchronize a user etc.
        response = HttpResponseRedirect(self.success_url)
        response.set_cookie('seahub_auth', '@'.join((user.email, token.key)))
        return response

class OIDCLogoutView(MozillaLogout):
    """Logout helper view"""

    def post(self, request):
        """ Log out the user.

            Override the parent to use seahub.auth.logout instead of
            django.contrib.auth.logout to clean encryption keys and such.
        """
        logout_url = self.redirect_url

        if request.user.is_authenticated:
            # Check if a method exists to build the URL to log out the user
            # from the OP.
            logout_from_op = getattr(settings, 'OIDC_OP_LOGOUT_URL_METHOD', '')
            if logout_from_op:
                logout_url = import_string(logout_from_op)(request)

            # Log out the Django user if they were logged in.
            auth.logout(request)

        return HttpResponseRedirect(logout_url)
```

- https://github.com/HumanBrainProject/mozilla-django-oidc-demo

- Service developers are asked to develop their OIDC clients in the integration environment first, and then deploying the same to the production environment.
  - Ensure you have access to the integration environment
    - Instructions on how to register your user account in the integration environment can be provided by support
  - Follow the steps outlined in this presentation to create an OIDC client on the integration environment

Human Brain Project

EBRAINS

Quota manager

Presentation and Q&A

Simulate with EBRAINS
7-10 November 2022, online

Paul Chaney

Co-funded by the European Union

- EBRAINS services run on service accounts that receive Fenix resources and redistribute them to end users. Such services should use the central quota manager to help manage their users' quotas:
  - https://wiki.ebrains.eu/bin/view/Collabs/quota-management
  - https://wiki.ebrains.eu/bin/view/Collabs/quota-management/API
- The quota manager has several main purposes
  - It defines resources that are managed by EBRAINS services on behalf of end users
  - It maps quota level names to quota values for each resource
  - It keeps track of the resources that each user has consumed
- Main use case
  - User Alice wants to perform a command in service X that will consume more resources
  - Alice and or X estimate how many resources will be consumed by the command.
  - X calls the Quota Manager to check if Alice has enough quota for this command.
  - The Quota Manager responds true or false.
  - The service X uses the response to launch the command or to block it

- Each service is responsible for informing the quota manager about consumption
  - The quota manager does not know what a user is consuming without information from the service itself
- A service can end up querying multiple quotas:
  - For example, the image service consumes its own HPC resources and also consumes resources of the Bucket service for its output.
- The quota manager does not necessarily keep track of resources synchronously
  - This means the quota manager may not always be 100% up to date
    - E.g If the quota manager is down for any reason, new public collabs created during the down period will not be recorded immediately but will be updated once the manager is back up
    - It is the services' responsibility to ensure the quota manager has up to date values to prevent excessive over consumption.
- Although not yet implemented, a dashboard to view users' quotas and quota consumption will most likely be developed.

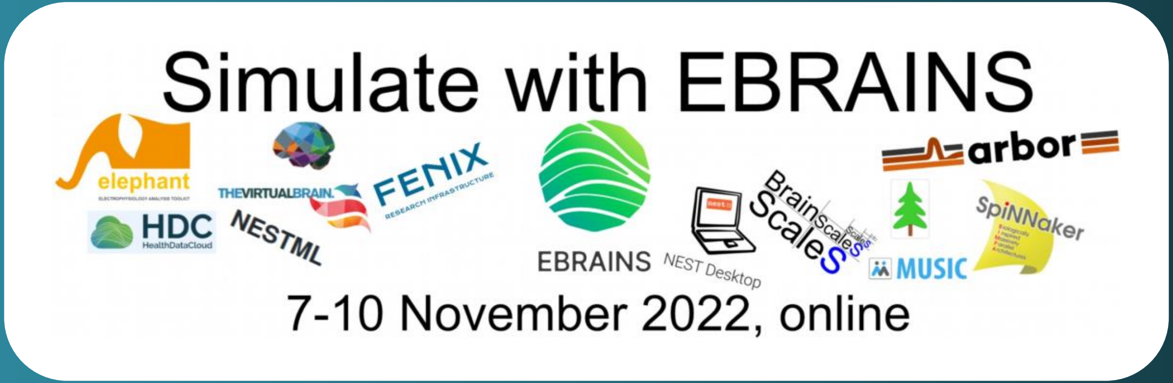
- There are currently several resources being tracked in the quota manager
  - Drive service's storage
  - Bucket service's storage
  - Image service's Supercomputing node hours
  - Wiki service's number of public collabs
- More resources will be tracked as needed, with some soon to be implemented:
  - Lab service's RAM hours
- While utilizing the quota manager, services should ensure that they can continue functioning if the quota manager cannot be contacted.
  - Services can update the resource consumption at a later time.
- It is important to note that collabs are accessible by multiple users. Their resource usage (Drive, Bucket, number of collabs) are assigned to a single owner per collab.
  - The collab creator is the owner. The ability to transfer ownership of collabs will be added soon.

- In order to integrate your service with the Quota manager:
  - Draw up a proposal
  - Contact support with your proposal

Human Brain Project

EBRAINS

Thank you


Simulate with EBRAINS

7-10 November 2022, online

www.humanbrainproject.eu www.ebrains.eu

Co-funded by the European Union