Azure configuration

Azure configuration

We'll need to create two application registrations for Azure AD authentication to cover both direct API use and usage from the OpenAPI (swagger) documentation.

We'll start with the API.

Backend API

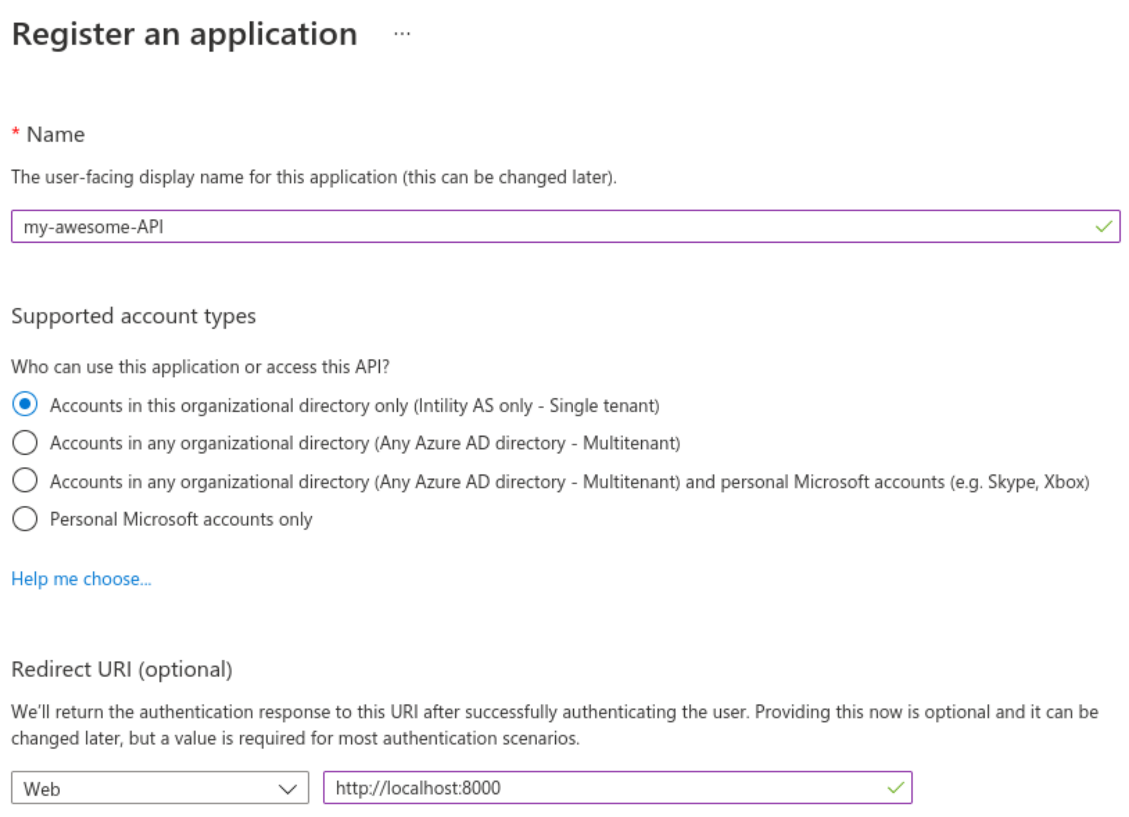
Step 1 - Create app registration

Head over to Azure -> Azure Active Directory -> App registrations, and create a new registration.

Select a fitting name for your project; Azure will present the name to the user during consent.

- `Supported account types`: `Single tenant`
- `Redirect URI`: Choose `Web` and `http://localhost:8000` as a value

Press **Register**


Register an application ···

* Name

The user-facing display name for this application (this can be changed later).

my-awesome-API ✓

Supported account types

Who can use this application or access this API?

- [✓ Accounts in this organizational directory only (Intility AS only - Single tenant)]
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- Personal Microsoft accounts only

Help me choose...

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

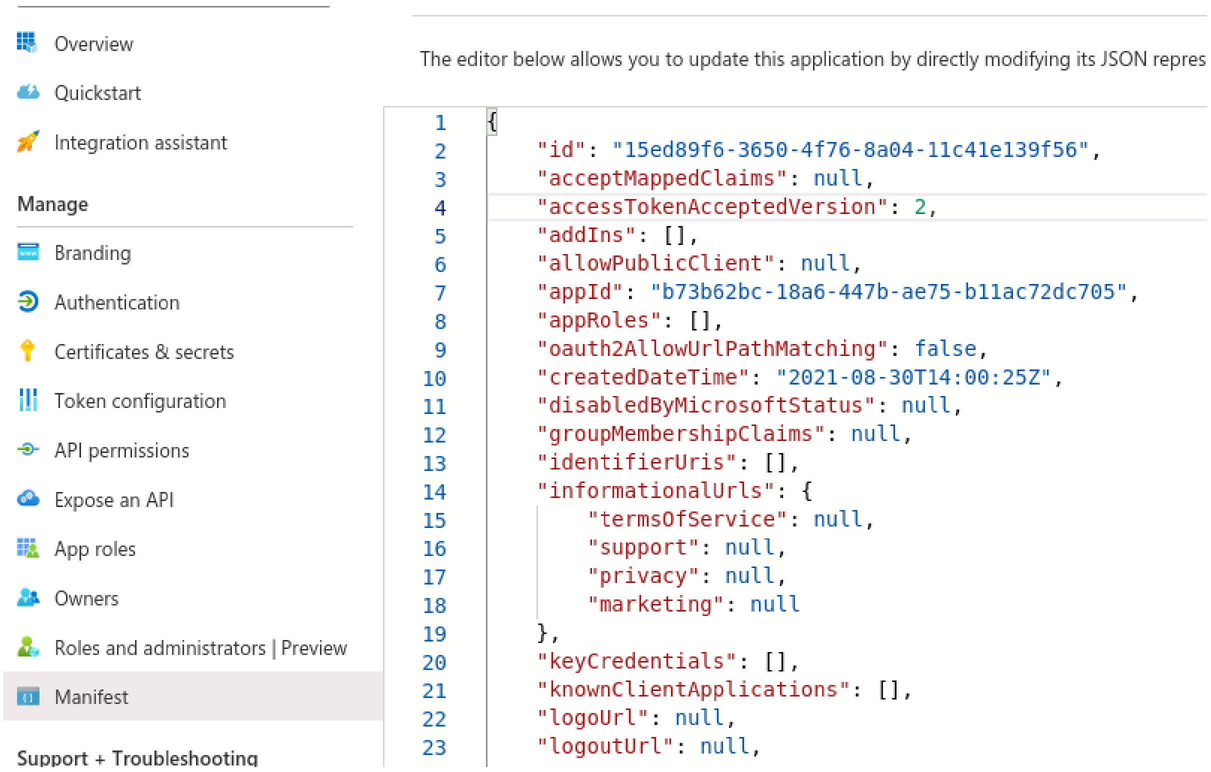
Web | http://localhost:8000 ✓

Step 2 - Change token version to `v2`

First we'll change the token version to version 2. In the left menu bar, click `Manifest` and find the line that says `accessTokenAcceptedVersion`. Change its value from `null` to `2`.

Press **Save**

(This change can take some time to happen, which is why we do this first.)



The editor below allows you to update this application by directly modifying its JSON repres

```
{
    "id": "15ed89f6-3650-4f76-8a04-11c41e139f56",
    "acceptMappedClaims": null,
    "accessTokenAcceptedVersion": 2,
    "addIns": [],
    "allowPublicClient": null,
    "appId": "b73b62bc-18a6-447b-ae75-b11ac72dc705",
    "appRoles": [],
    "oauth2AllowUrlPathMatching": false,
    "createdDateTime": "2021-08-30T14:00:25Z",
    "disabledByMicrosoftStatus": null,
    "groupMembershipClaims": null,
    "identifierUris": [],
    "informationalUrls": {
        "termsOfService": null,
        "support": null,
        "privacy": null,
        "marketing": null
    },
    "keyCredentials": [],
    "knownClientApplications": [],
    "logoUrl": null,
    "logoutUrl": null,
```

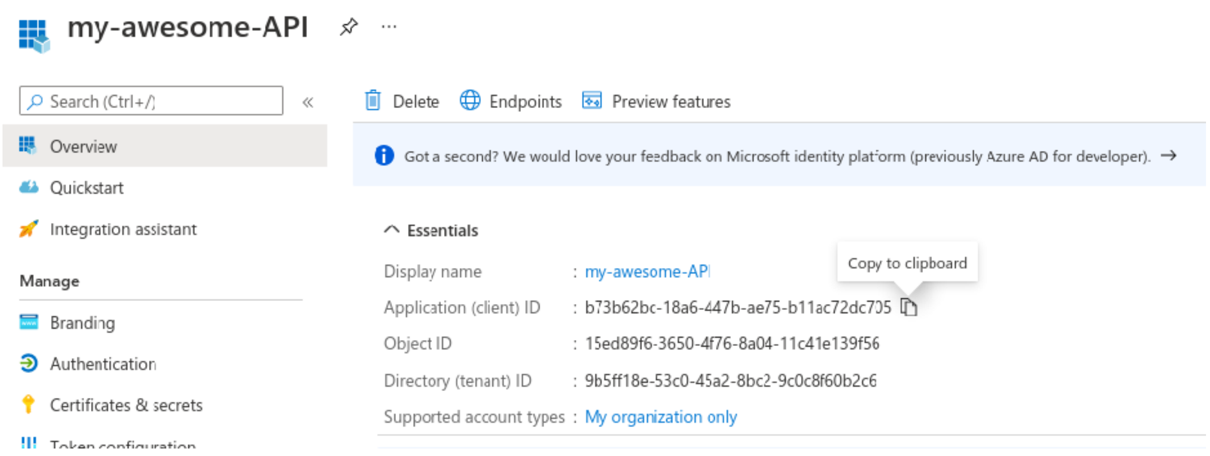
Step 3 - Note down your application IDs

Go back to the `Overview`, found in the left menu.

Copy the `Application (Client) ID` and `Directory (tenant) ID`, we'll need these for later. I like to use `.env` files to store variables like these:

.env

```
TENANT_ID=
APP_CLIENT_ID=
OPENAPI_CLIENT_ID=
```


my-awesome-API

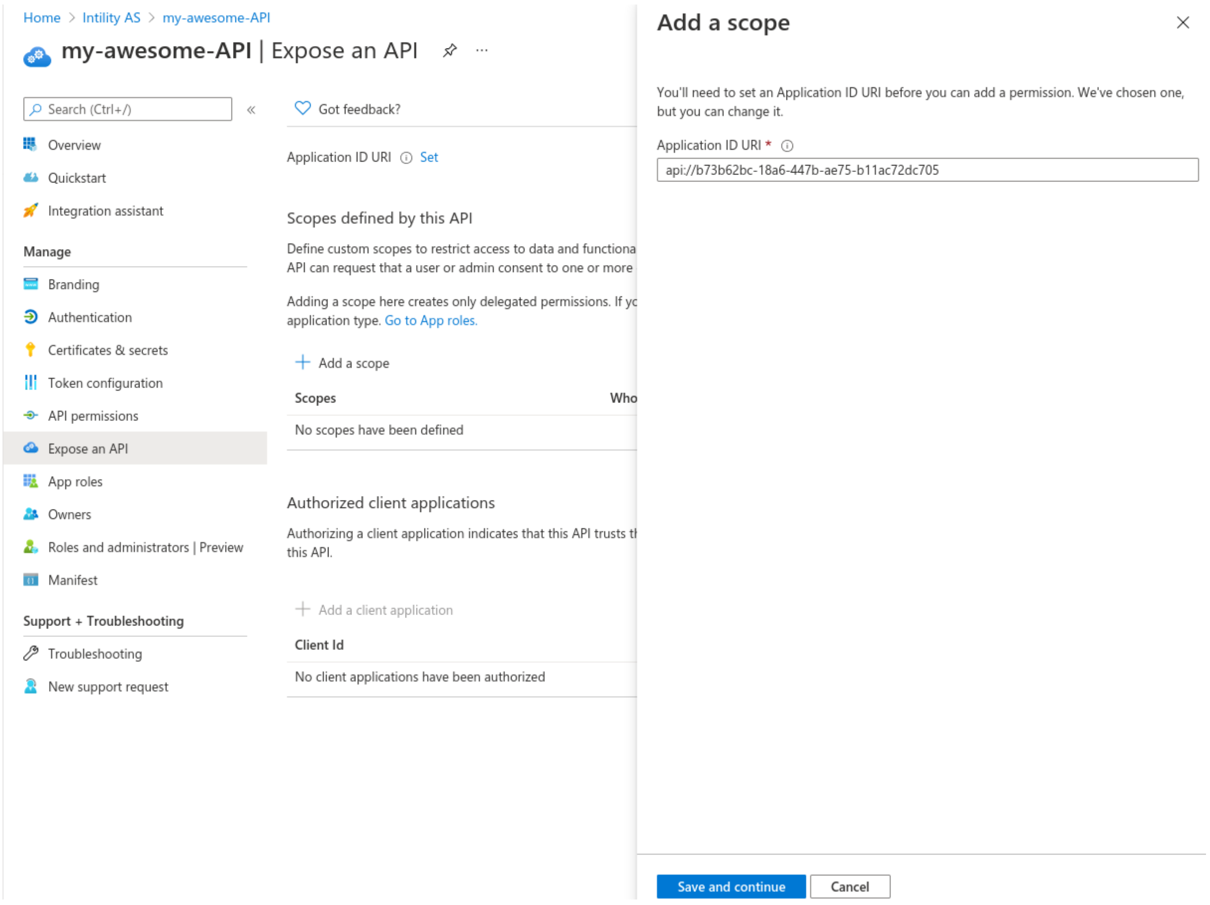
Essentials

Display name	: my-awesome-API
Application (client) ID	: b73b62bc-18a6-447b-ae75-b11ac72dc705
Object ID	: 15ed89f6-3650-4f76-8a04-11c41e139f56
Directory (tenant) ID	: 9b5ff18e-53c0-45a2-8bc2-9c0c8f60b2c6
Supported account types	: My organization only

Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Step 4 - Add an application scope

1. Go to **Expose an API** in the left menu bar under your app registration.
2. Press **+ Add a scope**
3. You'll be prompted to set an Application ID URI, leave the suggested one and press **Save and continue**


Home > Intility AS > my-awesome-API

my-awesome-API | Expose an API

Application ID URI ⓘ Set

Scopes defined by this API

Define custom scopes to restrict access to data and functiona API can request that a user or admin consent to one or more

Adding a scope here creates only delegated permissions. If yc application type. Go to App roles.

+ Add a scope

Scopes	Who
No scopes have been defined	

Authorized client applications

Authorizing a client application indicates that this API trusts tl this API.

+ Add a client application

Client Id
No client applications have been authorized

Add a scope

You'll need to set an Application ID URI before you can add a permission. We've chosen one, but you can change it.

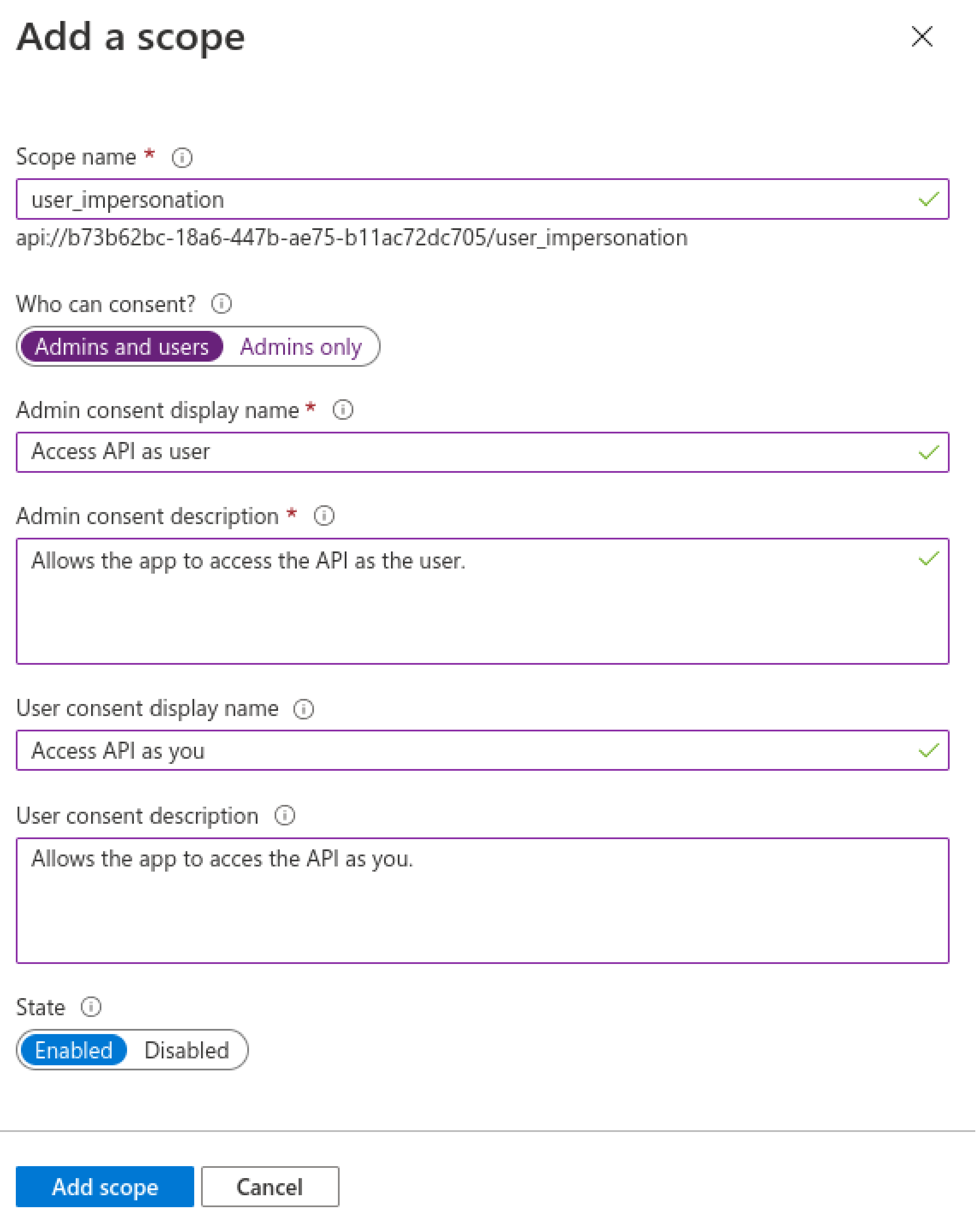
Application ID URI * ⓘ

api://b73b62bc-18a6-447b-ae75-b11ac72dc705

Add a scope named `user_impersonation` that can be consented by `Admins and users`.

You can use the following descriptions:

```
Access API as userAllows the app to access the API as the user.
Access API as youAllows the app to access the API as you.
```



Add a scope

Scope name * user_impersonation
api://b73b62bc-18a6-447b-ae75-b11ac72dc705/user_impersonation

Who can consent? [✓ Admins and users] Admins only

Admin consent display name * Access API as user

Admin consent description * Allows the app to access the API as the user.

User consent display name Access API as you

User consent description Allows the app to acces the API as you.

State [✓ Enabled] Disabled

OpenAPI Documentation

Our OpenAPI documentation will use the `Authorization Code Grant Flow, with Proof Key for Code Exchange` flow. It's a flow that enables a user of a Single-Page Application to safely log in, consent to permissions and fetch an `access_token` in the `JWT` format. When the user clicks `Try out` on the APIs, the `access_token` is attached to the header as a `Bearer` token. This is the token the backend will validate.

So, let's set it up!

Step 1 - Create app registration

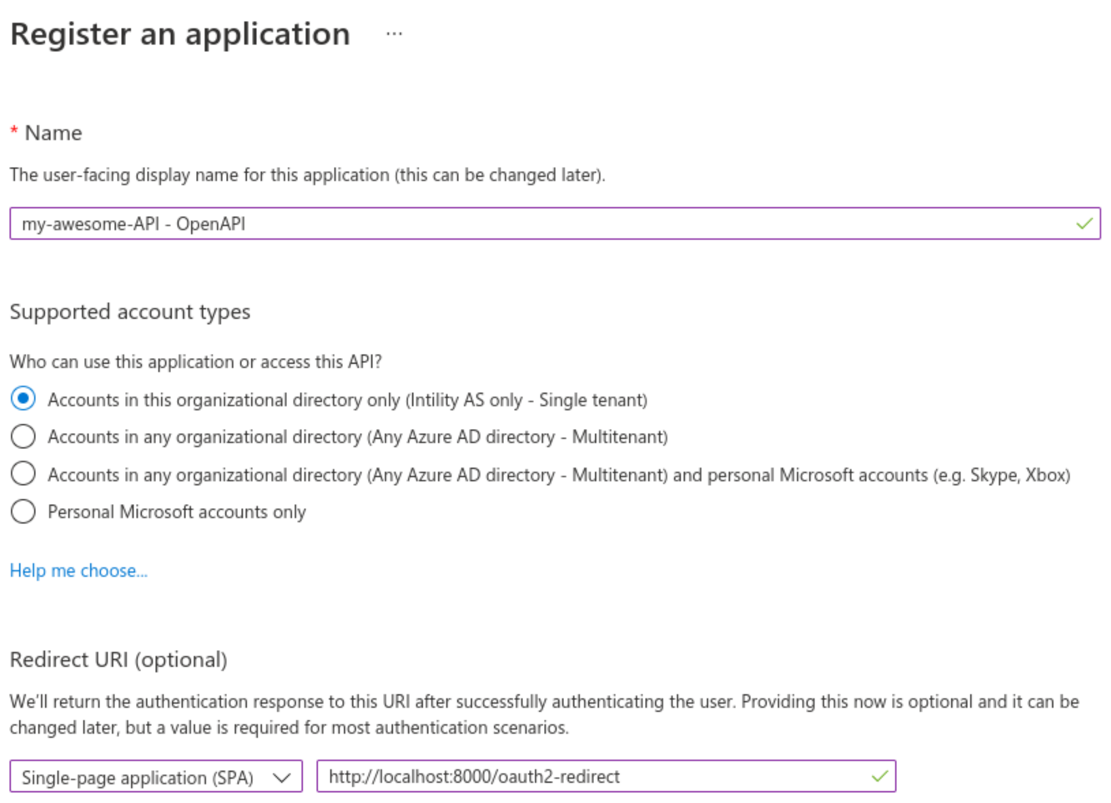
Just like in the previous chapter, we have to create an application registration for our OpenAPI.

Head over to Azure -> Azure Active Directory -> App registrations, and create a new registration.

Use the same name, but with `- OpenAPI` appended to it.

- `Supported account types`: `Single tenant`
- `Redirect URI`: Choose `Single-Page Application (SPA)` and `http://localhost:8000/oauth2-redirect` as a value

Press **Register**


Register an application ...

* Name

The user-facing display name for this application (this can be changed later).

my-awesome-API - OpenAPI ✓

Supported account types

Who can use this application or access this API?

[✓ Accounts in this organizational directory only (Intility AS only - Single tenant)]

Accounts in any organizational directory (Any Azure AD directory - Multitenant)

Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

Personal Microsoft accounts only

Help me choose...

Redirect URI (optional)

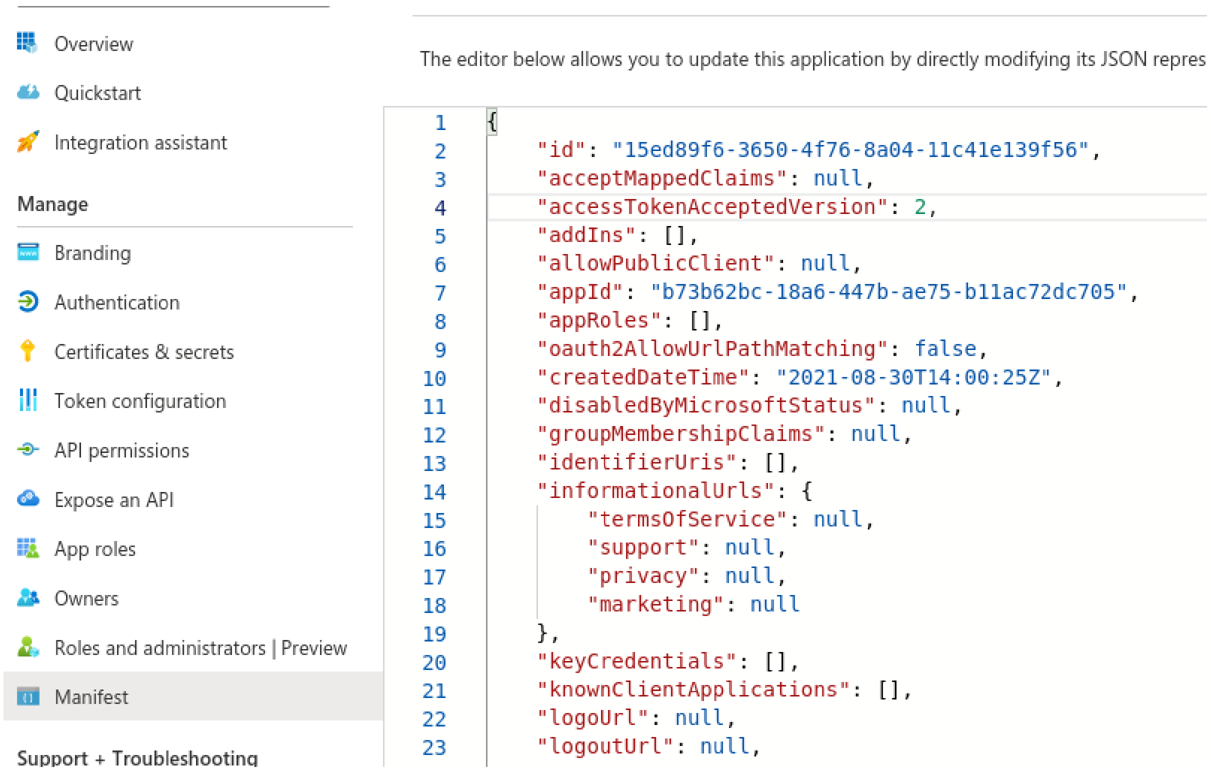
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Single-page application (SPA) | http://localhost:8000/oauth2-redirect ✓

Step 2 - Change token version to `v2`

Like last time, we'll change the token version to version 2. In the left menu bar, click `Manifest` and find the line that says `accessTokenAcceptedVersion`. Change its value from `null` to `2`.

Press **Save**



The editor below allows you to update this application by directly modifying its JSON repres

```
{
    "id": "15ed89f6-3650-4f76-8a04-11c41e139f56",
    "acceptMappedClaims": null,
    "accessTokenAcceptedVersion": 2,
    "addIns": [],
    "allowPublicClient": null,
    "appId": "b73b62bc-18a6-447b-ae75-b11ac72dc705",
    "appRoles": [],
    "oauth2AllowUrlPathMatching": false,
    "createdDateTime": "2021-08-30T14:00:25Z",
    "disabledByMicrosoftStatus": null,
    "groupMembershipClaims": null,
    "identifierUris": [],
    "informationalUrls": {
        "termsOfService": null,
        "support": null,
        "privacy": null,
        "marketing": null
    },
    "keyCredentials": [],
    "knownClientApplications": [],
    "logoUrl": null,
    "logoutUrl": null,
```

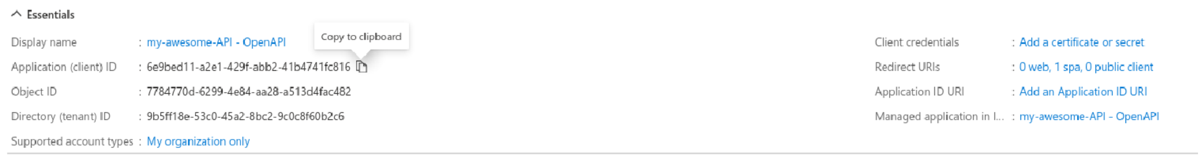
Step 3 - Note down your application IDs

Go back to the `Overview`, found in the left menu.

Copy the `Application (Client) ID` and save it as your `OPENAPI_CLIENT_ID`:

.env

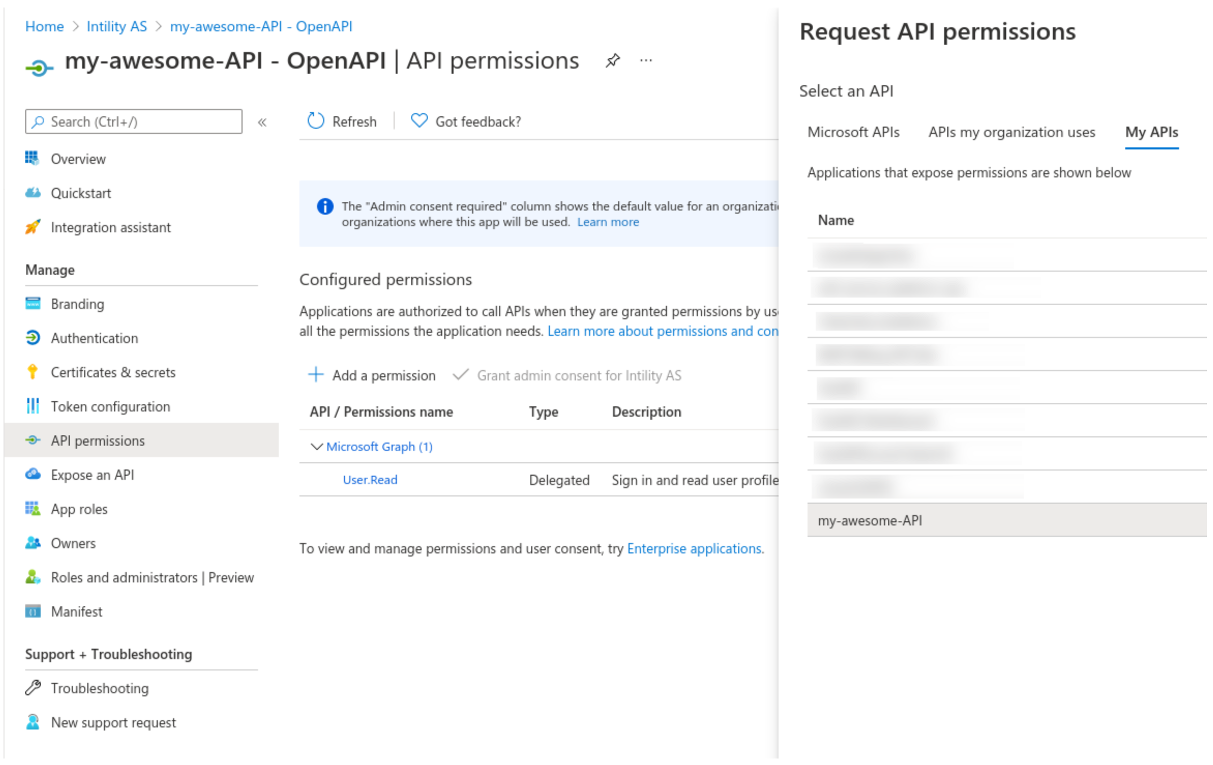
```
TENANT_ID=
APP_CLIENT_ID=
OPENAPI_CLIENT_ID=
```



Display name	: my-awesome-API - OpenAPI
Application (client) ID	: 6e9bed11-a2e1-429f-abb2-41b4741fc816
Object ID	: 7784770d-6299-4e84-aa28-a513d4fac482
Directory (tenant) ID	: 9b5ff18e-53c0-45a2-8bc2-9c0c8f60b2c6
Supported account types	: My organization only
Client credentials	: Add a certificate or secret
Redirect URIs	: 0 web, 1 spa, 0 public client
Application ID URI	: Add an Application ID URI
Managed application in l...	: my-awesome-API - OpenAPI

Step 4 - Allow OpenAPI to talk to the backend

To allow OpenAPI to talk to the backend API, you must add API permissions to the OpenAPI app registration. In the left menu, go to **API Permissions** and **Add a permission**.


Home > Intility AS > my-awesome-API - OpenAPI

my-awesome-API - OpenAPI | API permissions

Search (Ctrl+/)

- Overview
- Quickstart
- Integration assistant

Manage

- Branding
- Authentication
- Certificates & secrets
- Token configuration
- API permissions
- Expose an API
- App roles
- Owners
- Roles and administrators | Preview
- Manifest

Support + Troubleshooting

- Troubleshooting
- New support request

Refresh | Got feedback?

The "Admin consent required" column shows the default value for an organizati organizations where this app will be used. Learn more

Configured permissions

Applications are authorized to call APIs when they are granted permissions by us all the permissions the application needs. Learn more about permissions and con

+ Add a permission ✓ Grant admin consent for Intility AS

API / Permissions name	Type	Description
Microsoft Graph (1)		
User.Read	Delegated	Sign in and read user profile

To view and manage permissions and user consent, try Enterprise applications.

Request API permissions

Select an API

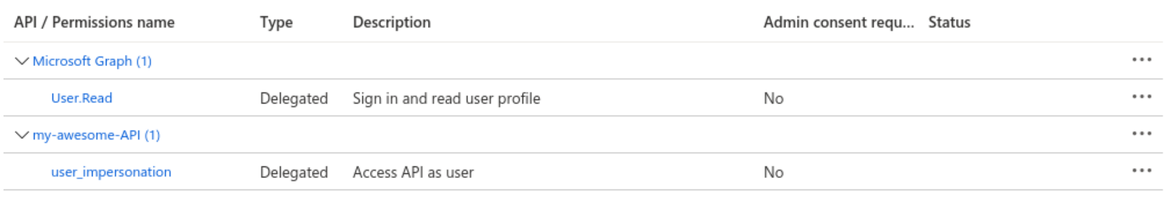
Microsoft APIs | APIs my organization uses | My APIs

Applications that expose permissions are shown below

Name
[redacted]
my-awesome-API

Select the `user_impersonation` scope, and press **Add a permission**.

Your view should now look something like this:



API / Permissions name	Type	Description	Admin consent requ...	Status
Microsoft Graph (1)				
User.Read	Delegated	Sign in and read user profile	No	
my-awesome-API (1)				
user_impersonation	Delegated	Access API as user	No	

That's it! Next step is to configure the FastAPI application.

FastAPI configuration

We'll do the **simplest setup possible** in these docs, through a one-file `main.py`.

We assume you've done the FastAPI tutorial and have dependencies installed, such as `FastAPI` and `Gunicorn`.

For a more "real life" project example, look at the project demo https://xsystem.espressif.cn/myehrproject/ehr_auth

Getting started

First, either create your `.env` file and fill out your variables or insert them directly in your settings later.

.env

```
APP_CLIENT_ID=
TENANT_ID=
OPENAPI_CLIENT_ID=
```

Create your `main.py` file:

main.py

```
from fastapi import FastAPI
import uvicorn

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}

if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

Run your application and ensure that everything works on http://localhost:8000/docs

INFO

You need to run the application on the configured port in Azure AD for the next steps to work!

Add your settings

First, add your settings to the application. We'll need these later. The way I've set it up will look for a `.env`-file to populate your settings, but you can also just set a `default` value directly.

main.py

```
from typing import Union

import uvicorn
from fastapi import FastAPI
from pydantic import AnyHttpUrl, BaseSettings, Field


class Settings(BaseSettings):
    SECRET_KEY: str = Field('my super secret key', env='SECRET_KEY')
    BACKEND_CORS_ORIGINS: list[Union[str, AnyHttpUrl]] = ['http://localhost:8000']
    OPENAPI_CLIENT_ID: str = Field(default='', env='OPENAPI_CLIENT_ID')
    APP_CLIENT_ID: str = Field(default='', env='APP_CLIENT_ID')
    TENANT_ID: str = Field(default='', env='TENANT_ID')

    class Config:
        env_file = '.env'
        env_file_encoding = 'utf-8'
        case_sensitive = True

settings = Settings()

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}


if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

Configure `CORS`

Now, let's configure our `CORS`. Without `CORS` your OpenAPI docs won't work as expected:

main.py

```
from typing import Union

import uvicorn
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import AnyHttpUrl, BaseSettings, Field


class Settings(BaseSettings):
    SECRET_KEY: str = Field('my super secret key', env='SECRET_KEY')
    BACKEND_CORS_ORIGINS: list[Union[str, AnyHttpUrl]] = ['http://localhost:8000']
    OPENAPI_CLIENT_ID: str = Field(default='', env='OPENAPI_CLIENT_ID')
    APP_CLIENT_ID: str = Field(default='', env='APP_CLIENT_ID')
    TENANT_ID: str = Field(default='', env='TENANT_ID')

    class Config:
        env_file = '.env'
        env_file_encoding = 'utf-8'
        case_sensitive = True

settings = Settings()

app = FastAPI()

if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[str(origin) for origin in settings.BACKEND_CORS_ORIGINS],
        allow_credentials=True,
        allow_methods=['*'],
        allow_headers=['*'],
    )


@app.get("/")
async def root():
    return {"message": "Hello World"}


if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

Configure OpenAPI Documentation

In order for our OpenAPI documentation to work, we have to configure a few settings directly in the `FastAPI` application.

main.py

```
from typing import Union

import uvicorn
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import AnyHttpUrl, BaseSettings, Field


class Settings(BaseSettings):
    SECRET_KEY: str = Field('my super secret key', env='SECRET_KEY')
    BACKEND_CORS_ORIGINS: list[Union[str, AnyHttpUrl]] = ['http://localhost:8000']
    OPENAPI_CLIENT_ID: str = Field(default='', env='OPENAPI_CLIENT_ID')
    APP_CLIENT_ID: str = Field(default='', env='APP_CLIENT_ID')
    TENANT_ID: str = Field(default='', env='TENANT_ID')

    class Config:
        env_file = '.env'
        env_file_encoding = 'utf-8'
        case_sensitive = True

settings = Settings()

app = FastAPI(
    swagger_ui_oauth2_redirect_url='/oauth2-redirect',
    swagger_ui_init_oauth={
        'usePkceWithAuthorizationCodeGrant': True,
        'clientId': settings.OPENAPI_CLIENT_ID,
    },
)

if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[str(origin) for origin in settings.BACKEND_CORS_ORIGINS],
        allow_credentials=True,
        allow_methods=['*'],
        allow_headers=['*'],
    )


@app.get("/")
async def root():
    return {"message": "Hello World"}


if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

The `swagger_ui_oauth2_redirect_url` setting for redirect should be as configured in Azure AD. The `swagger_ui_init_oauth` are standard mapped OpenAPI properties. You can find documentation about them here

We've used two flags: `usePkceWithAuthorizationCodeGrant`, which is the authentication flow. `clientId` is our application Client ID, which will autofill a field for the end users later.

Implementing ESP-Auth

Now, the fun part begins! 🚀

Import the `SingleTenantAzureAuthorizationCodeBearer` from `ESP-Auth` and configure it:

main.py

```
from typing import Union

import uvicorn
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import AnyHttpUrl, BaseSettings, Field
from ehr_auth.fastapi.auth import SingleTenantAzureAuthorizationCodeBearer



class Settings(BaseSettings):
    SECRET_KEY: str = Field('my super secret key', env='SECRET_KEY')
    BACKEND_CORS_ORIGINS: list[Union[str, AnyHttpUrl]] = ['http://localhost:8000']
    OPENAPI_CLIENT_ID: str = Field(default='', env='OPENAPI_CLIENT_ID')
    APP_CLIENT_ID: str = Field(default='', env='APP_CLIENT_ID')
    TENANT_ID: str = Field(default='', env='TENANT_ID')

    class Config:
        env_file = '.env'
        env_file_encoding = 'utf-8'
        case_sensitive = True

settings = Settings()

app = FastAPI(
    swagger_ui_oauth2_redirect_url='/oauth2-redirect',
    swagger_ui_init_oauth={
        'usePkceWithAuthorizationCodeGrant': True,
        'clientId': settings.OPENAPI_CLIENT_ID,
    },
)

if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[str(origin) for origin in settings.BACKEND_CORS_ORIGINS],
        allow_credentials=True,
        allow_methods=['*'],
        allow_headers=['*'],
    )

azure_scheme = SingleTenantAzureAuthorizationCodeBearer(
    app_client_id=settings.APP_CLIENT_ID,
    tenant_id=settings.TENANT_ID,
    scopes={
        f'api://{settings.APP_CLIENT_ID}/user_impersonation': 'user_impersonation',
    }
)


@app.get("/")
async def root():
    return {"message": "Hello World"}


if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

We pass the `app_client_id=` to be our Backend application ID, our `tenant_id` to be our Tenant ID, and then lastly our scopes. We'll get back to the scopes later.

Add loading of OpenID Configuration on startup

By adding `on_event('startup')` we're able to load the OpenID configuration immediately, instead of doing it when the first user authenticates. This isn't required, but makes things a bit quicker. When 24 hours has passed, the configuration will be considered out of date, and update when a user does a request. You can use background tasks to refresh it before that happens if you'd like.

main.py

```
from typing import Union

import uvicorn
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import AnyHttpUrl, BaseSettings, Field
from ehr_auth.fastapi.auth import SingleTenantAzureAuthorizationCodeBearer



class Settings(BaseSettings):
    SECRET_KEY: str = Field('my super secret key', env='SECRET_KEY')
    BACKEND_CORS_ORIGINS: list[Union[str, AnyHttpUrl]] = ['http://localhost:8000']
    OPENAPI_CLIENT_ID: str = Field(default='', env='OPENAPI_CLIENT_ID')
    APP_CLIENT_ID: str = Field(default='', env='APP_CLIENT_ID')
    TENANT_ID: str = Field(default='', env='TENANT_ID')

    class Config:
        env_file = '.env'
        env_file_encoding = 'utf-8'
        case_sensitive = True

settings = Settings()

app = FastAPI(
    swagger_ui_oauth2_redirect_url='/oauth2-redirect',
    swagger_ui_init_oauth={
        'usePkceWithAuthorizationCodeGrant': True,
        'clientId': settings.OPENAPI_CLIENT_ID,
    },
)

if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[str(origin) for origin in settings.BACKEND_CORS_ORIGINS],
        allow_credentials=True,
        allow_methods=['*'],
        allow_headers=['*'],
    )

azure_scheme = SingleTenantAzureAuthorizationCodeBearer(
    app_client_id=settings.APP_CLIENT_ID,
    tenant_id=settings.TENANT_ID,
    scopes={
        f'api://{settings.APP_CLIENT_ID}/user_impersonation': 'user_impersonation',
    }
)


@app.on_event('startup')
async def load_config() -> None:
    """
    Load OpenID config on startup.
    """
    await azure_scheme.openid_config.load_config()


@app.get("/")
async def root():
    return {"message": "Hello World"}


if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

Adding authentication to our vie

There's two ways of adding dependencies in FastAPI. You can use `Depends()` or `Security()`. `Security()` has an extra property called `scopes`. `FastAPI-Azure-Auth` support both, but if you use `Security()` you can also lock down your API views based on the scope.

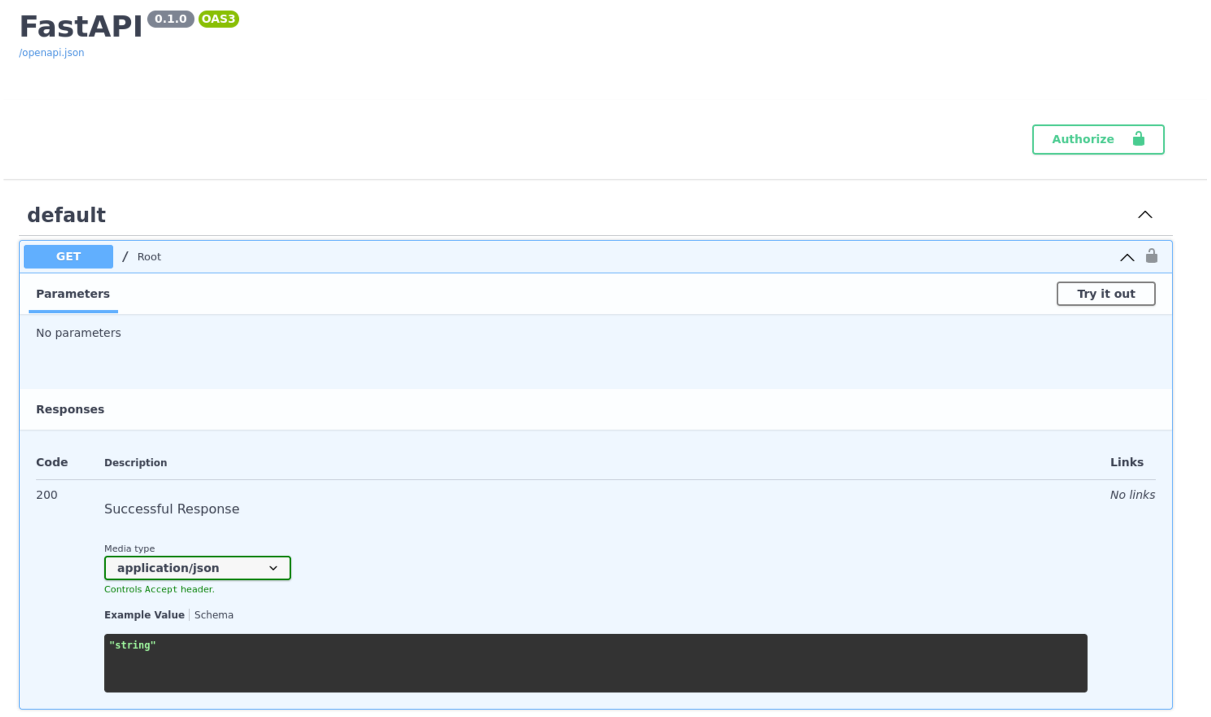
Let's do that:

main.py

```
from typing import Union

import uvicorn
from fastapi import FastAPI, Security
from fastapi.middleware.cors import CORSMiddleware
from pydantic import AnyHttpUrl, BaseSettings, Field
from ehr_auth.fastapi.auth import SingleTenantAzureAuthorizationCodeBearer



class Settings(BaseSettings):
    SECRET_KEY: str = Field('my super secret key', env='SECRET_KEY')
    BACKEND_CORS_ORIGINS: list[Union[str, AnyHttpUrl]] = ['http://localhost:8000']
    OPENAPI_CLIENT_ID: str = Field(default='', env='OPENAPI_CLIENT_ID')
    APP_CLIENT_ID: str = Field(default='', env='APP_CLIENT_ID')
    TENANT_ID: str = Field(default='', env='TENANT_ID')

    class Config:
        env_file = '.env'
        env_file_encoding = 'utf-8'
        case_sensitive = True

settings = Settings()

app = FastAPI(
    swagger_ui_oauth2_redirect_url='/oauth2-redirect',
    swagger_ui_init_oauth={
        'usePkceWithAuthorizationCodeGrant': True,
        'clientId': settings.OPENAPI_CLIENT_ID,
    },
)

if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=[str(origin) for origin in settings.BACKEND_CORS_ORIGINS],
        allow_credentials=True,
        allow_methods=['*'],
        allow_headers=['*'],
    )

azure_scheme = SingleTenantAzureAuthorizationCodeBearer(
    app_client_id=settings.APP_CLIENT_ID,
    tenant_id=settings.TENANT_ID,
    scopes={
        f'api://{settings.APP_CLIENT_ID}/user_impersonation': 'user_impersonation',
    }
)


@app.on_event('startup')
async def load_config() -> None:
    """
    Load OpenID config on startup.
    """
    await azure_scheme.openid_config.load_config()


@app.get("/", dependencies=[Security(azure_scheme)])
async def root():
    return {"message": "Hello World"}


if __name__ == '__main__':
    uvicorn.run('main:app', reload=True)
```

Testing it out

Head over to your OpenAPI documentation at http://localhost:8000/docs and check out your API documentation. You'll see a new button called `Authorize`. Before clicking it, try out your API to see that you're unauthorized.


FastAPI 0.1.0 OAS3
/openapi.json

Authorize

default

GET / Root

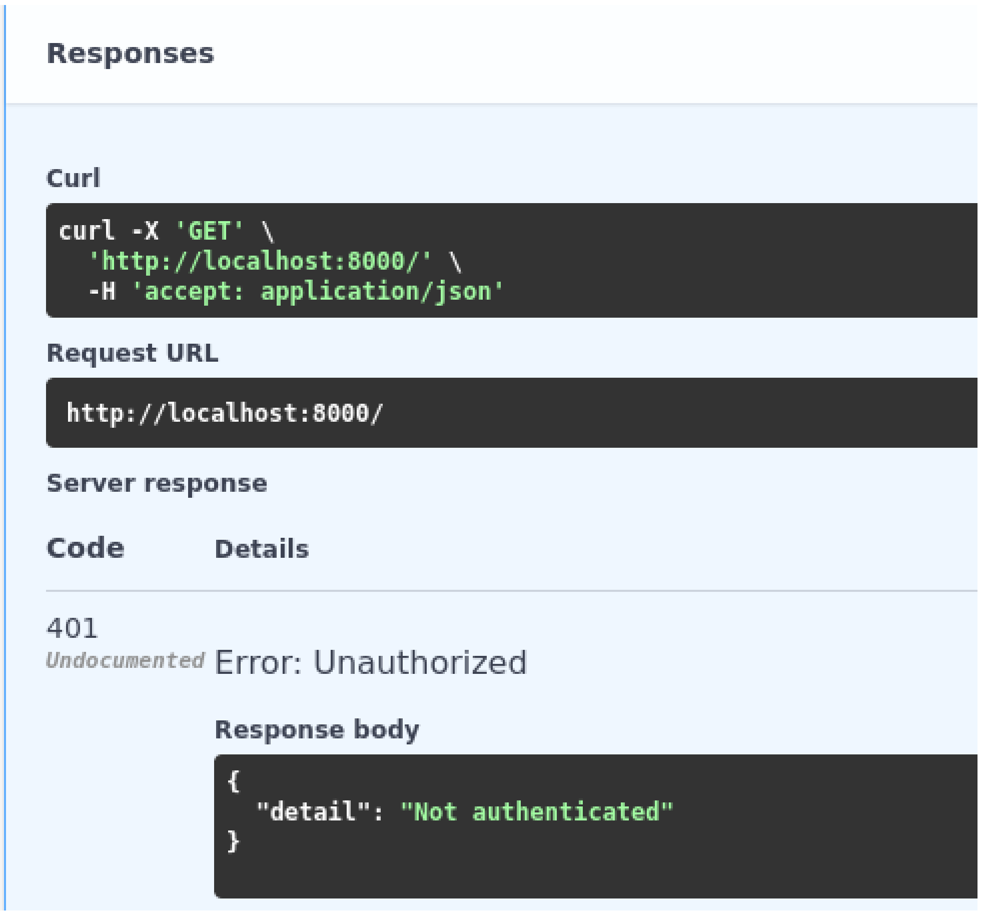
Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response Media type: application/json Controls Accept header. Example Value | Schema "string"	No links


Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:8000/
```

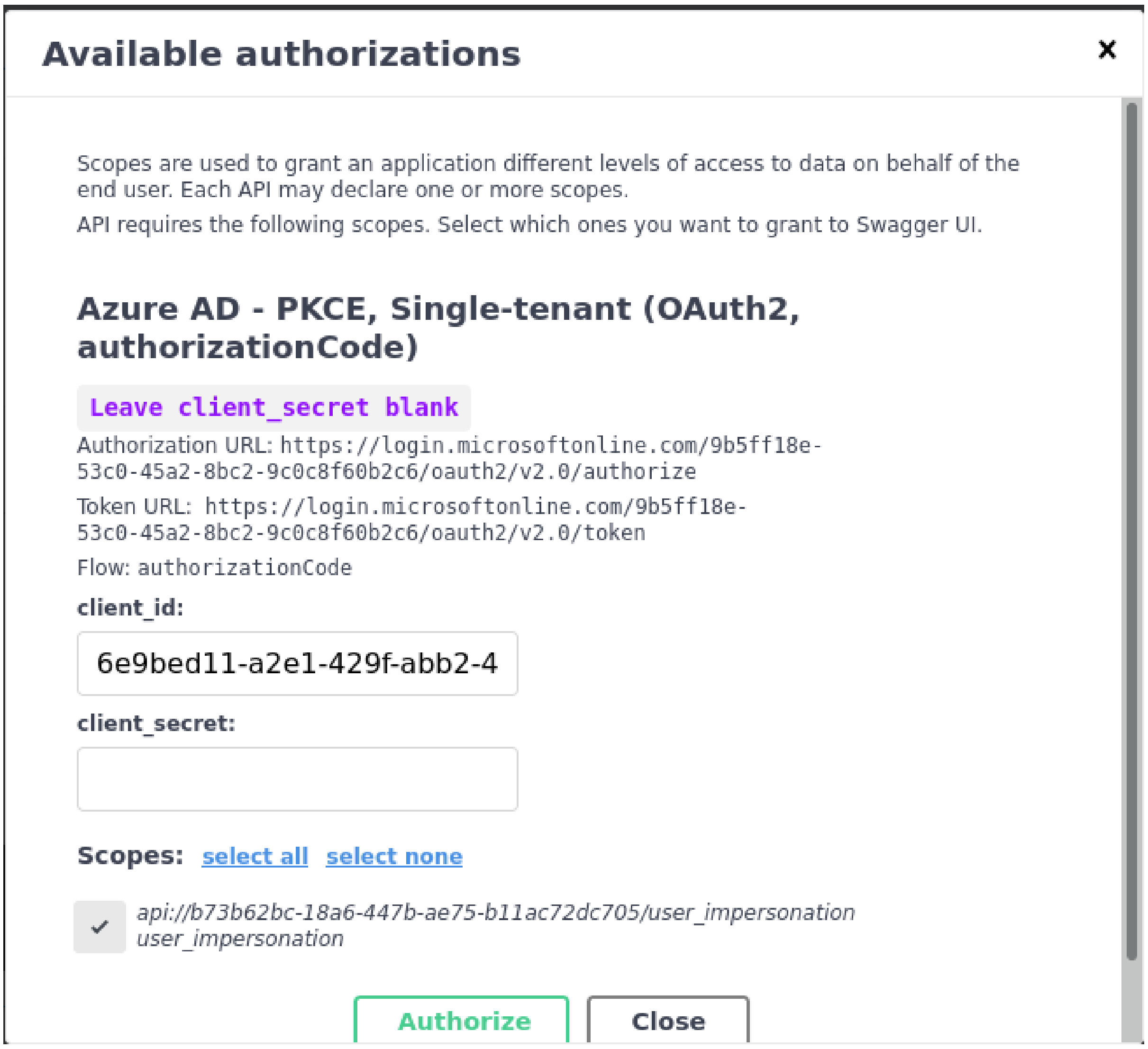
Server response

Code	Details
401 *Undocumented*	Error: Unauthorized

Response body

```
{
  "detail": "Not authenticated"
}
```

Now, let's authenticate. Click the **Authorize** button. Check your scope, and leave `Client secret` blank. You do not need that with the PKCE flow.



Available authorizations

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.

API requires the following scopes. Select which ones you want to grant to Swagger UI.

Azure AD - PKCE, Single-tenant (OAuth2, authorizationCode)

`Leave client_secret blank`

Authorization URL: `https://login.microsoftonline.com/9b5ff18e-53c0-45a2-8bc2-9c0c8f60b2c6/oauth2/v2.0/authorize`

Token URL: `https://login.microsoftonline.com/9b5ff18e-53c0-45a2-8bc2-9c0c8f60b2c6/oauth2/v2.0/token`

Flow: `authorizationCode`

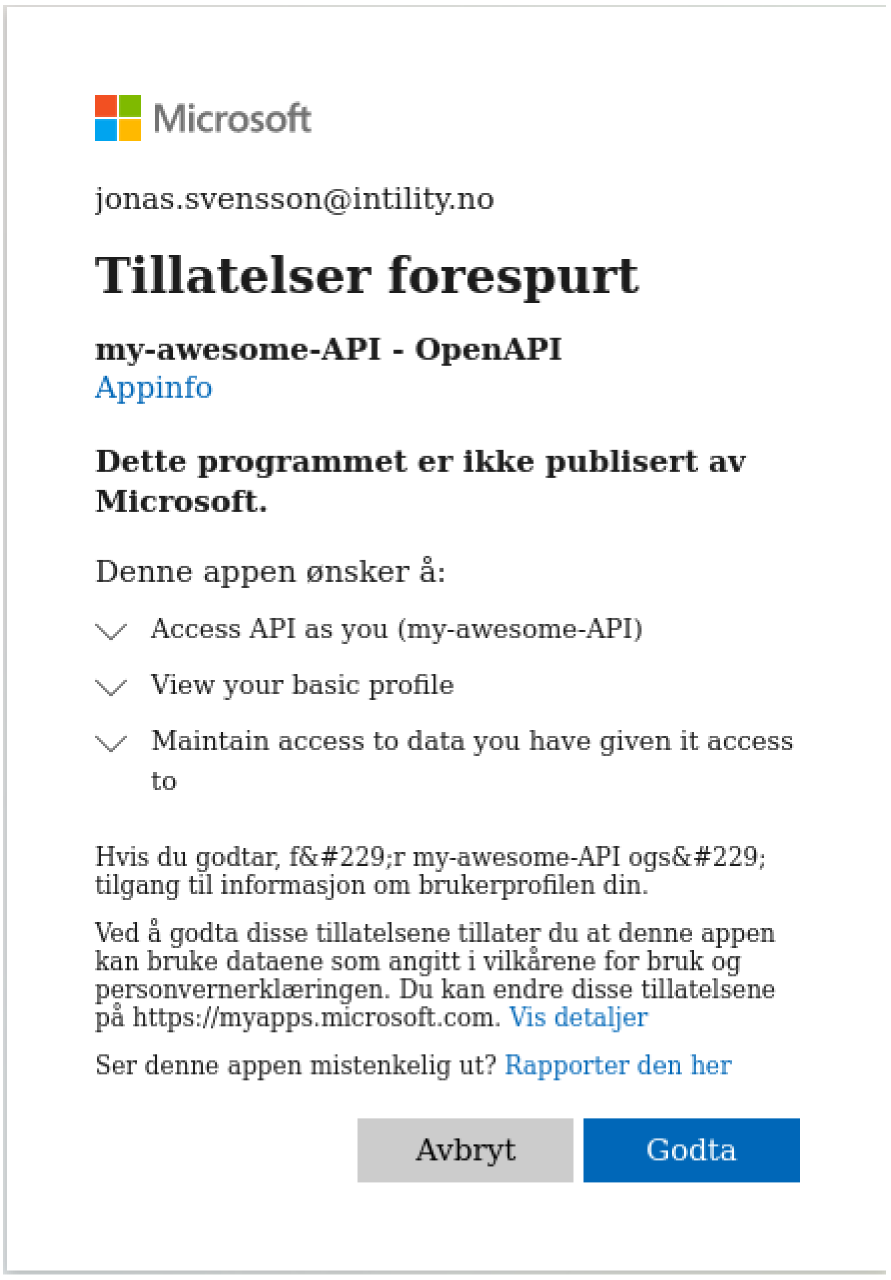
client_id: 6e9bed11-a2e1-429f-abb2-4

client_secret:

Scopes: select all select none

✓ *api://b73b62bc-18a6-447b-ae75-b11ac72dc705/user_impersonation*
user_impersonation

Consent to the permissions requested:



jonas.svensson@intility.no

Tillatelser forespurt

my-awesome-API - OpenAPI
Appinfo

Dette programmet er ikke publisert av Microsoft.

Denne appen ønsker å:

- Access API as you (my-awesome-API)
- View your basic profile
- Maintain access to data you have given it access to

Hvis du godtar, f&#229;r my-awesome-API ogs&#229; tilgang til informasjon om brukerprofilen din.

Ved å godta disse tillatelsene tillater du at denne appen kan bruke dataene som angitt i vilkårene for bruk og personvernerklæringen. Du kan endre disse tillatelsene på https://myapps.microsoft.com. Vis detaljer

Ser denne appen mistenkelig ut? Rapporter den her

INFO

If you get a warning that your redirect URL is wrong, you're probably using `127.0.0.1` instead of `localhost`

Try out your API again to see that it works!

Last thing..

As discussed earlier, there is a `scope` parameter to the `Security()` version of `Depends()`. If you'd want to lock down your API to only be accessible by those with certain scopes, you can simply pass it into the dependency.

```
@app.get("/", dependencies=[Security(azure_scheme, scopes=['wrong_scope'])])
```
Copy

If you do this and try out your API again, you'll see that you're denied.

`WebContext ContextVar`

We added context support, In the process of user requesting resources, you can store data and use it at will

```
@router.get(
    '/hello',
    response_model=HelloWorldResponse,
    summary='Say hello',
    name='hello_world',
    operation_id='helloWorld',
    dependencies=[Depends(validate_is_admin_user)],
)
async def world(request: Request) -> dict[str, Union[str, UserModel]]:
    """
    Wonder who we say hello to?
    """
    user: UserModel = IdentityWebContext.get_context().token_info
    return {'hello': 'world', 'user': user}
```

You're now safe and secure! Good luck! 🔒🚀