**Step1**

We need to set up our application in Microsoft Azure Portal:

After registration, note down these values:

* Application (client) ID
* Directory (tenant) ID
* Create a new client secret in "Certificates & secrets"

**Step2**

We need a secret key for session encryption.

**Step3**

Create a config file (basic)

'''

Author: Aditya Bhatt

Date: 15-01-2025 11:00 AM

NOTE:

1.Basic or bare minimum config for Microsoft OAuth

'''

from dotenv import load\_dotenv

import os

load\_dotenv()

MICROSOFT\_CONFIG = {

    'client\_id': os.getenv('MICROSOFT\_CLIENT\_ID'),

    'client\_secret': os.getenv('MICROSOFT\_CLIENT\_SECRET'),

    'tenant\_id': os.getenv('MICROSOFT\_TENANT\_ID'),

    'redirect\_uri': os.getenv('MICROSOFT\_REDIRECT\_URI'),

    #OAuth Endpoints

    'authorize\_endpoint':f'https://login.microsoftonline.com/{MICROSOFT\_CONFIG["tenant\_id"]}/oauth2/v2.0/authorize',

     "token\_url": f"https://login.microsoftonline.com/{os.getenv('MICROSOFT\_TENANT\_ID')}/oauth2/v2.0/token",

    # Basic scopes we need

    "scopes": ["openid", "profile", "email"]

    #Openid is like userid

    #Profile is like user name

    #Email is like user email

}

In OAuth flow, we need two main routes:

1. Login Route (/auth/login):
   * This is where users will be redirected to start the login process
   * It will create the Microsoft login URL with our:
     + client\_id
     + redirect\_uri
     + scopes
   * When users hit this route, they'll be redirected to Microsoft's login page
2. Callback Route (/auth/callback):
   * This is where Microsoft will redirect back to after successful login
   * It will:
     + Receive the authorization code from Microsoft
     + Exchange this code for access token
     + Get user information using the token
     + Create a session for the user

**Step 4**

Create a Login Route (/auth/login)

'''

Author: Aditya Bhatt

Date: 15-01-2025 11:00 AM

NOTE:

'''

from fastapi import APIRouter,Request

from fastapi.responses import RedirectResponse

from .config import MICROSOFT\_CONFIG

from urllib.parse import urlencode

router = APIRouter()

@router.get("/login")

async def login():

    query\_params = {

        "client\_id": MICROSOFT\_CONFIG["client\_id"],

        "response\_type": "code",

        "redirect\_uri": MICROSOFT\_CONFIG["redirect\_uri"],

        "scope": " ".join(MICROSOFT\_CONFIG["scopes"]),

    }

    auth\_url = f"{MICROSOFT\_CONFIG['authorize\_endpoint']}?{urlencode(query\_params)}"

    return RedirectResponse(url=auth\_url)

**Step 5**

**@router.get("/callback")**

**def auth\_callback(code: str):**

**"""**

**Handle the OAuth callback from Microsoft**

**"""**

**try:**

**# Exchange authorization code for access token**

**token\_response = requests.post(**

**MICROSOFT\_CONFIG['token\_url'],**

**data={**

**'client\_id': MICROSOFT\_CONFIG['client\_id'],**

**'client\_secret': MICROSOFT\_CONFIG['client\_secret'],**

**'code': code,**

**'redirect\_uri': MICROSOFT\_CONFIG['redirect\_uri'],**

**'grant\_type': 'authorization\_code'**

**}**

**)**

**token\_response.raise\_for\_status()**

**token\_info = token\_response.json()**

**# Get user info using the access token**

**user\_response = requests.get(**

**'https://graph.microsoft.com/v1.0/me',**

**headers={'Authorization': f'Bearer {token\_info["access\_token"]}'}**

**)**

**user\_response.raise\_for\_status()**

**user\_info = user\_response.json()**

**# Return user information**

**return {**

**"message": "Successfully authenticated",**

**"email": user\_info.get("mail") or user\_info.get("userPrincipalName"),**

**"name": user\_info.get("displayName")**

**}**

**except requests.RequestException as e:**

**raise HTTPException(**

**status\_code=400,**

**detail=f"Error during authentication: {str(e)}"**

**)**

@router.get("/callback")

def auth\_callback(code: str):

    """

    Handle the OAuth callback from Microsoft

    """

    try:

        # Exchange authorization code for access token

        token\_response = requests.post(

            MICROSOFT\_CONFIG['token\_url'],

            data={

                'client\_id': MICROSOFT\_CONFIG['client\_id'],

                'client\_secret': MICROSOFT\_CONFIG['client\_secret'],

                'code': code,

                'redirect\_uri': MICROSOFT\_CONFIG['redirect\_uri'],

                'grant\_type': 'authorization\_code'

            }

        )

        token\_response.raise\_for\_status()

        token\_info = token\_response.json()

        # Get user info using the access token

        user\_response = requests.get(

            'https://graph.microsoft.com/v1.0/me',

            headers={'Authorization': f'Bearer {token\_info["access\_token"]}'}

        )

        user\_response.raise\_for\_status()

        user\_info = user\_response.json()

        # Return user information

        return {

            "message": "Successfully authenticated",

            "email": user\_info.get("mail") or user\_info.get("userPrincipalName"),

            "name": user\_info.get("displayName")

        }

    except requests.RequestException as e:

        raise HTTPException(

            status\_code=400,

            detail=f"Error during authentication: {str(e)}"

        )

**Note –**

** requests is synchronous and simpler to use**

** httpx is asynchronous and more complex**

**Setp 6**

**STEP 1: Adding Session Management**

**First, let's understand what we need:**

1. **Store user info after successful login**
2. **Create a session token for the user**
3. **Check this token on protected routes**

With Sessions: - User logs in with Microsoft - App creates a session (like a ticket) - User can go anywhere in the app - App remembers who they are

**# When user logs in:**

**request.session['user'] = {**

**"email": "user@example.com",**

**"name": "John"**

**}**

**# Behind the scenes, SessionMiddleware:**

**1. Creates a secure cookie in user's browser**

**2. Stores session data on server**

**3. Links cookie to session data**

1. **Key Benefits of FastAPI's SessionMiddleware:**

* Built-in async support (perfect for FastAPI)
* Automatic cookie handling
* Built-in security features
* Works seamlessly with FastAPI's dependency system
* No extra database needed (stores in-memory)
* Production-ready with minimal setup

Update main.py to add session\_middleware

import os

from fastapi import FastAPI, Request, HTTPException

from fastapi.middleware.cors import CORSMiddleware

import uvicorn

from dotenv import load\_dotenv

from app.route import router

from app.auth.route import router as auth\_router

from starlette.middleware.sessions import SessionMiddleware

# Load environment variables

load\_dotenv()

# Initialize FastAPI app

app = FastAPI(title="ChatBot Service", version="1.0.0")

# Add session middleware

app.add\_middleware(

    SessionMiddleware,

    secret\_key=os.getenv("SECRET\_KEY"),

    session\_cookie="chatbot\_session",    # Name of the cookie

    max\_age=3600,                        # Session expires in 1 hour

    same\_site="lax",                     # Cookie security setting

    https\_only=False                     # Set to True in production with HTTPS

)

# Configure CORS

origins = ["\*"]  # Allow all origins for development

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=origins,

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

)

# Include routers

app.include\_router(router, prefix="/chat", tags=["chat"])

app.include\_router(auth\_router, prefix="/auth", tags=["auth"])

# Root endpoint

@app.get("/")

async def root(request: Request):

    user=request.session.get("user")

    if user:

        return {"message": f"Welcome back, {user['name']}!"}

    else:

        return {"message": "This is a backend service for chatbot, please go to /docs to see the API documentation"}

if \_\_name\_\_ == "\_\_main\_\_":

    uvicorn.run(app, host="0.0.0.0", port=8000, reload=True)

Now test and Debug

The beauty is that FastAPI's SessionMiddleware handles all the complex encryption/decryption and cookie management automatically, while we just work with the simple session dictionary interface.