Azure AD App Registration Guide for AD Authentication & Authorization

Use Case: Business Users + Service Principals (OAuth2: Authorization Code + Client Credentials Flow)

This guide walks through setting up Azure AD app registrations to support:

- **Authorization Code Flow** for human users (frontend → backend)

Use case: Human users accessing a backend API via a frontend app (like React SPA).

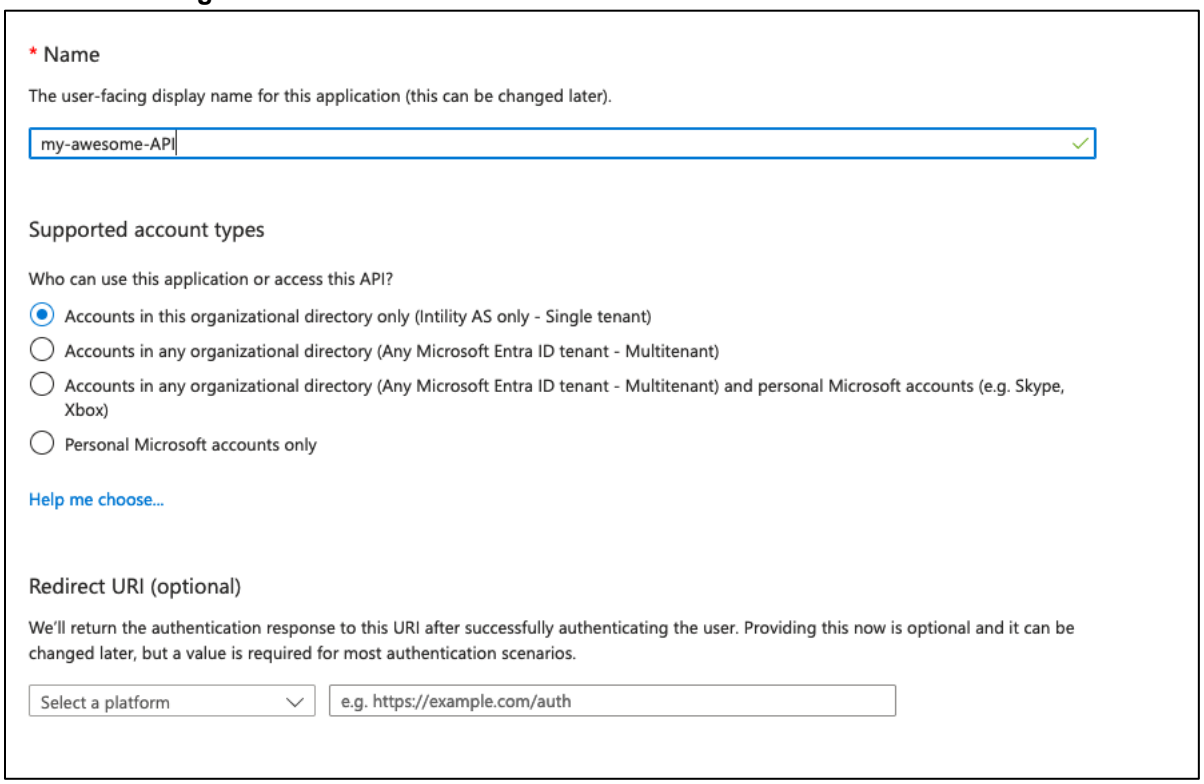
- **Client Credentials Flow** for service/service users (backend → backend)
- Centralized **roles/scopes/groups-based access control**

Part 1: App Registration for Users Authorization Code Flow (Frontend → Backend)

React → MSAL → Azure AD → Token → Backend with Authorization: Bearer

🔷 Step 1: Register the Backend API

1. Go to **Azure Portal → Azure Active Directory → App registrations → New registration**
2. Name: Backend API App
3. Supported account types:
   Accounts in this organizational directory only
4. Redirect URI:
   (Leave blank — backend doesn't require redirect)
5. Click **Register**



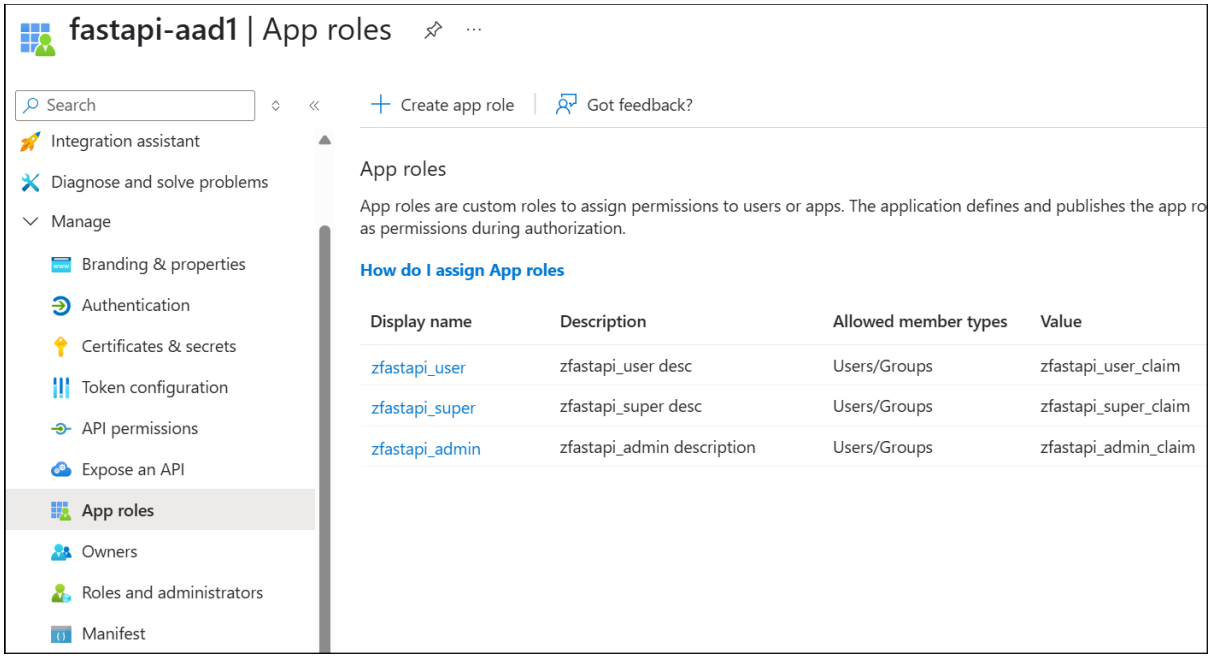
✅ Save:

- Application (client) ID
- Directory (tenant) ID

🔷 Step 2: Define App Roles (RBAC)

1. Go to the **"App roles"** section of the registered app
2. Click **+ Create app role**

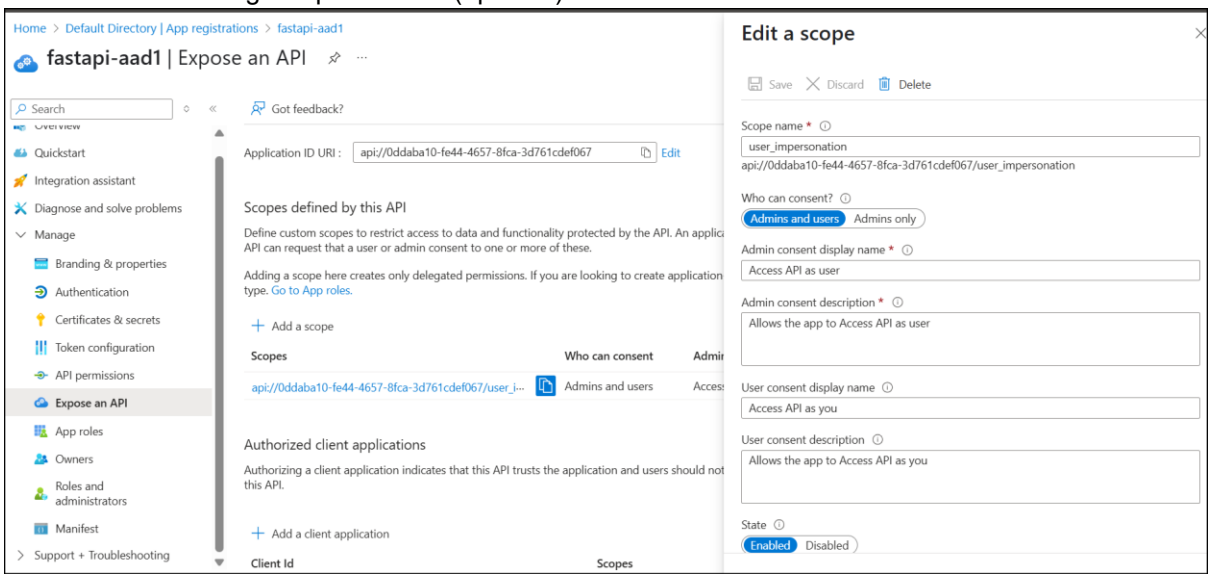
3. Add roles like:



4. Repeat for Manager, User, etc.
5. Click **Save and expose roles**

🔷 Step 3: Expose API Scopes

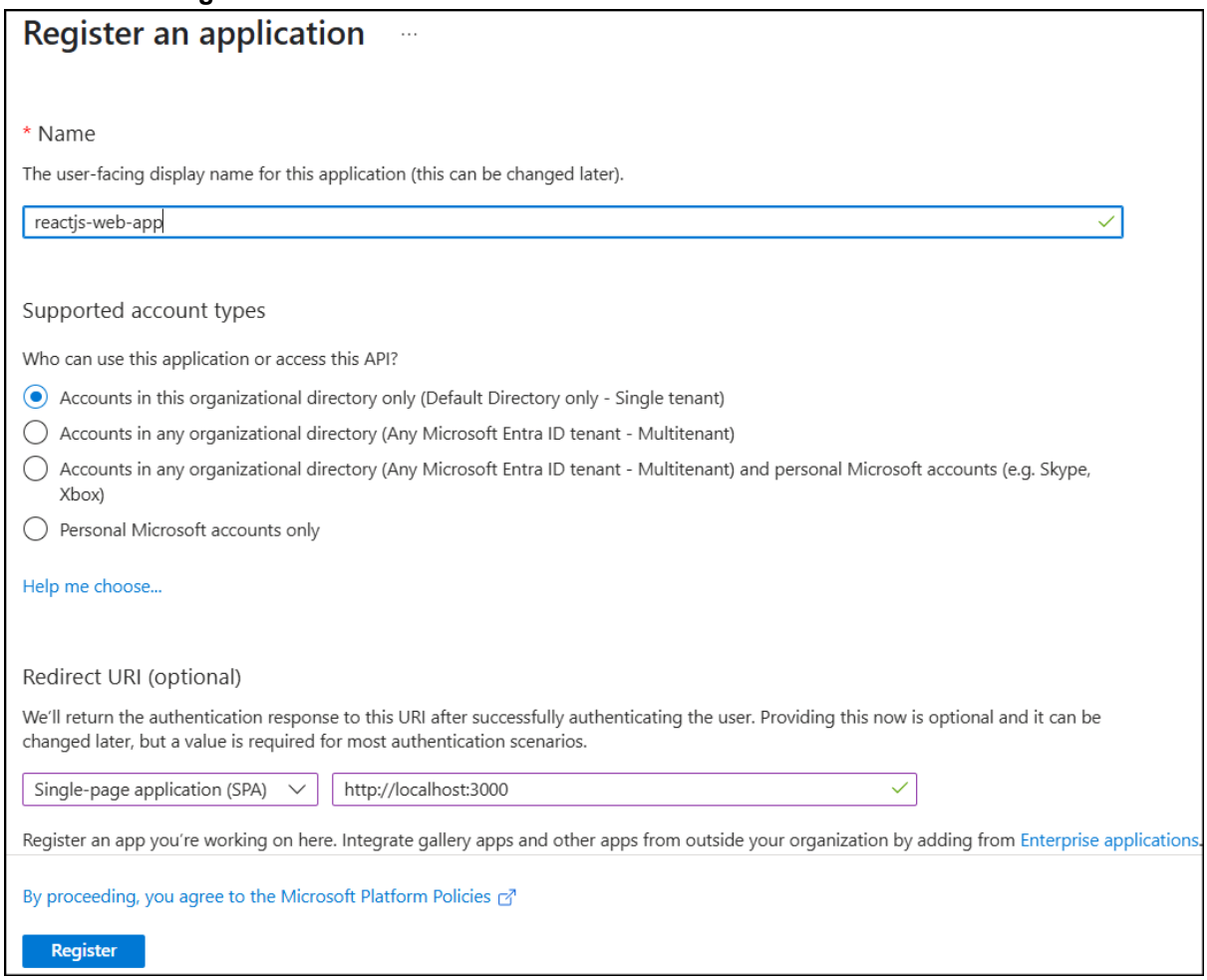
1. Go to **Expose an API**
2. Click **Set** for Application ID URI → api://<client-id>
3. Add a scope:
    - Scope name: access_as_user
    - Admin consent display name: Access API as user
    - Who can consent: Admins only
    - Assign required roles (optional)



Frontend App Registration

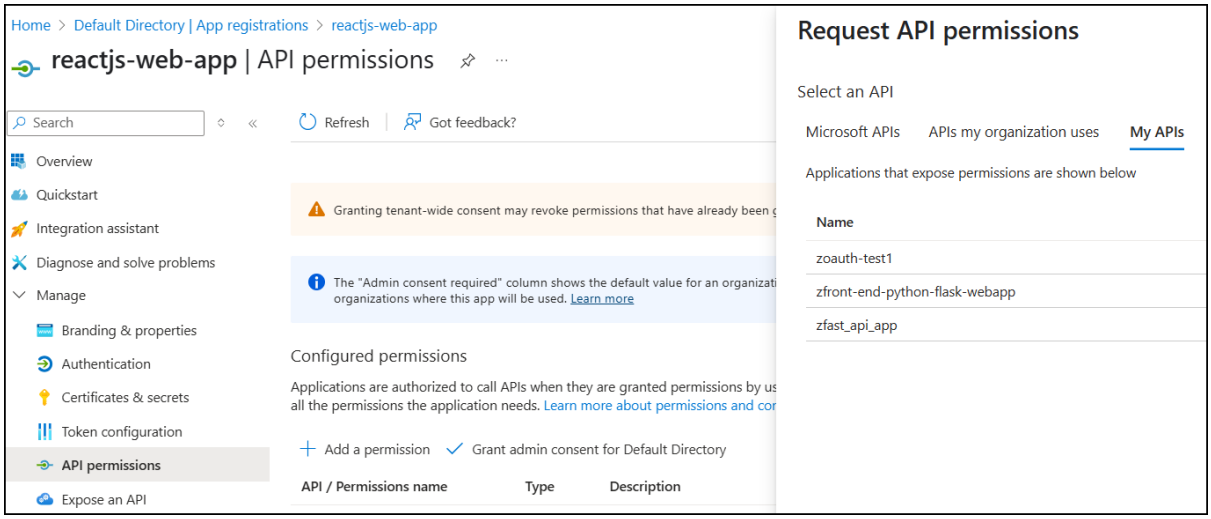
Step 1: Register Frontend App (React / SPA)

1. Go to **App registrations → New registration**
2. Name: Frontend React App
3. Supported account types:
   Accounts in this organizational directory only
4. Redirect URI: http://localhost:3000 (or your deployed URL)
5. Click **Register**



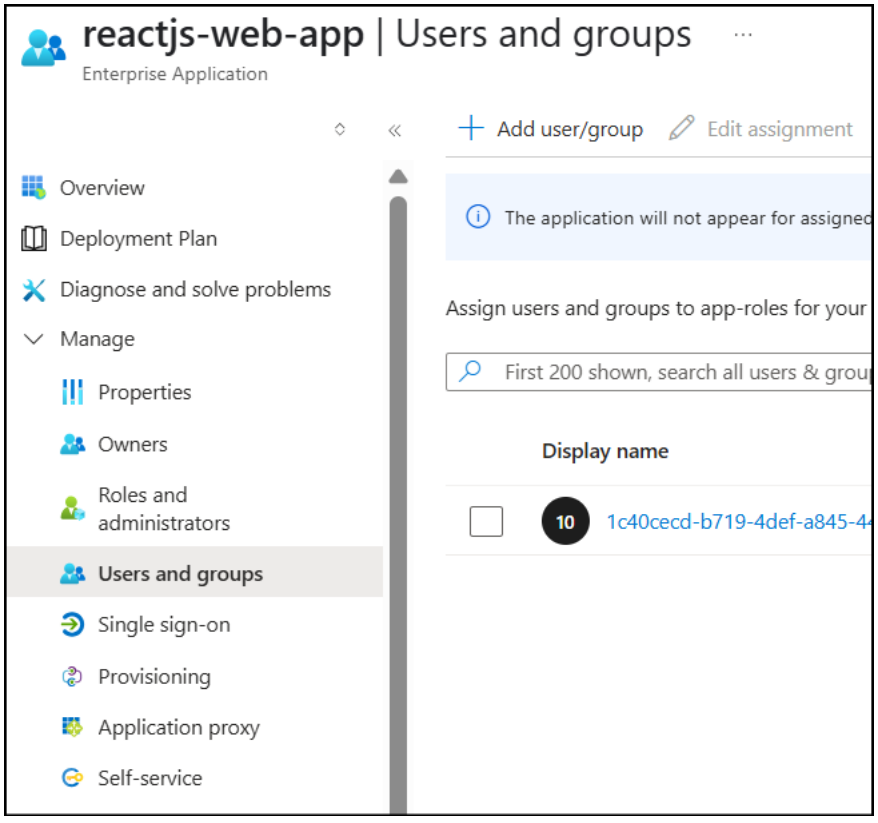
Step 2: Grant API Permissions

1. Go to **Frontend App → API permissions**
2. Click **Add a permission → My APIs**
3. Select the Backend API App
4. Choose scope: access_as_user
5. Click **Grant admin consent**



🔷 Step 3 Optional — Assign Roles to Users or Groups

- Go to **Enterprise Applications** → Find your backend app
- Open **Users and Groups**
- Assign Azure AD users or groups to the roles created earlier



✅ Flow Summary (Authorization Code Flow)

Actor	Action
Human User	Logs in through React SPA using MSAL
React App	Requests token from Azure AD with access_as_user scope
Azure AD	Returns access token to frontend
React App	Sends token to FastAPI via Authorization: Bearer
FastAPI	Validates token, checks roles, serves API

Backend → Backend / Service-to-Service API - App Registration - Client Credentials Flow

service fastapi-aad1 api calls to zfast_api_app api

Part 2: App Registration for Service Principals (Client Credentials Flow)

🔶 Step 1: Register a Service Client App

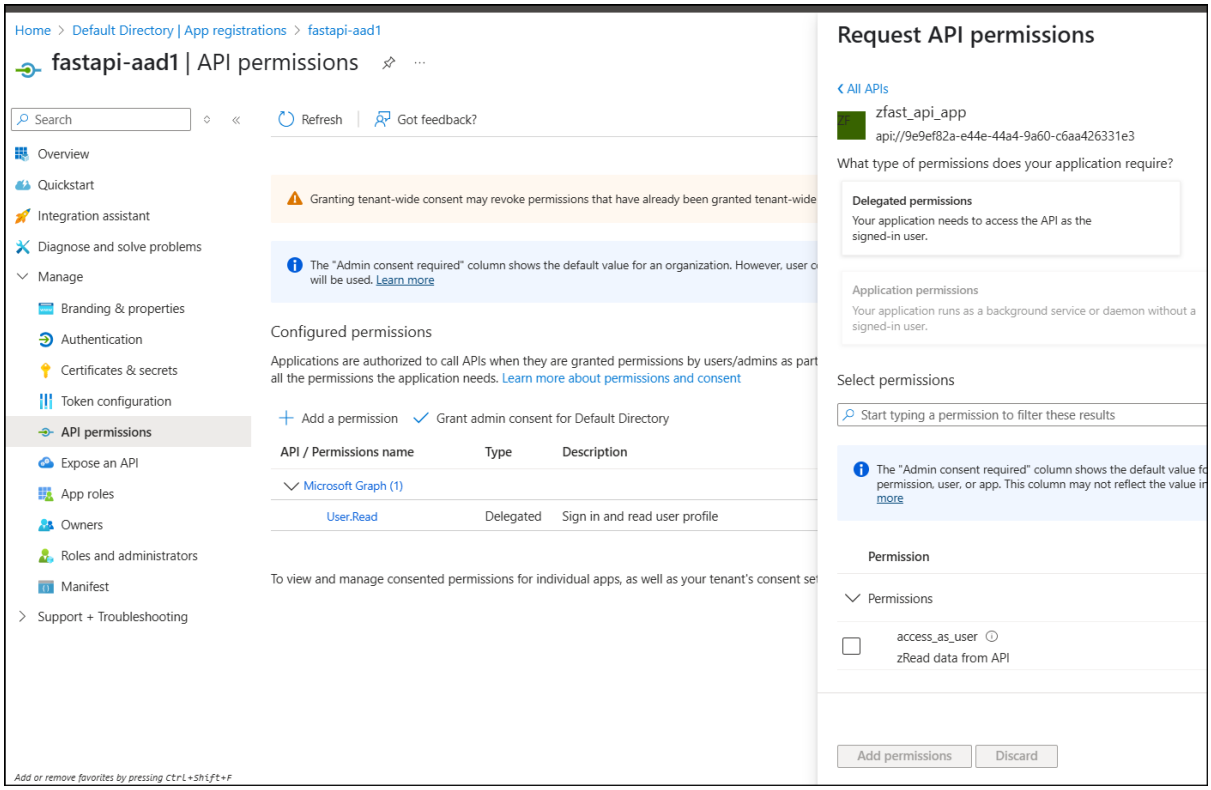
1. Go to **App registrations** → **New registration**
2. Name: Backend Service Client
3. Supported account types:
   Accounts in this organizational directory only
4. Redirect URI: *(leave blank)*
5. Register the app

🔶 Step 2: Create Client Secret

1. In the new app, go to **Certificates & Secrets**
2. Click **+ New client secret**
3. Set expiration (e.g., 6 months, 12 months)
4. Save the **value** immediately (you won't see it again)

🔶 Step 3: Grant API Access to Backend API (fastapi-aad1 api calls to zfast_api_app api)

1. Go to **API Permissions** → **Add a permission**
2. Select **My APIs**
3. Choose Backend API App
4. Select scope: access_as_user OR add a custom **Application-only** scope (optional)
5. Click **Grant admin consent**



🔶 Step 4: (Optional) Expose Application-only Scopes

In **Backend API App** → Expose an API:

- Define new scope:

- Name: access_as_app
- Who can consent: Admins only
- Assign to **Application** only (vs. User)

🔶 **Step 5: Use service principle client and secret to generate token**

✅ **Flow Summary (Client Credentials Flow)**

Actor	Action
Service A	Calls Azure AD for token using secret
Azure AD	Returns access token with access_as_user or .default scope
Service A	Sends token to Backend API in Authorization header
Backend API	Validates token, authorizes request

🔐 **Security Best Practices**

- Rotate client secrets frequently
- Use **Key Vault** for secret storage
- Enforce **least privilege** via scopes & roles
- Use **RBAC** to restrict endpoint access
- Validate:
  - aud (audience matches backend)
  - iss (token issued by Azure)
  - exp (not expired)
  - roles (if applicable)

✅ **1. Authorization Code Flow (Frontend → Backend)**

Use case: Human users accessing a backend API via a frontend app (like React SPA).

- **Backend API app registration**
  - Registers a resource (API) in Azure AD
  - Defines **roles (RBAC)** and **API scopes** (access_as_user)
  - Uses api://<client-id> format for Application ID URI
- **Frontend app registration**
  - Uses MSAL to request tokens
  - Adds API permissions to call the backend
  - Handles **admin consent** properly
- **Flow summary is accurate**:
  - React → MSAL → Azure AD → Token → Backend with Authorization: Bearer
  - Backend validates token and applies role checks

✅ **2. Client Credentials Flow (Backend → Backend / Service-to-Service)**

Use case: Machine-to-machine communication between services (e.g., microservices or backend jobs).

- **Service client app registration**
  - No redirect URI needed
  - Uses client secret
- **Backend API app registration**
  - Exposes API scopes (e.g., access_as_service)
  - Permissions granted via API permissions
- client authenticating using **client ID + secret** to get a token and call another API