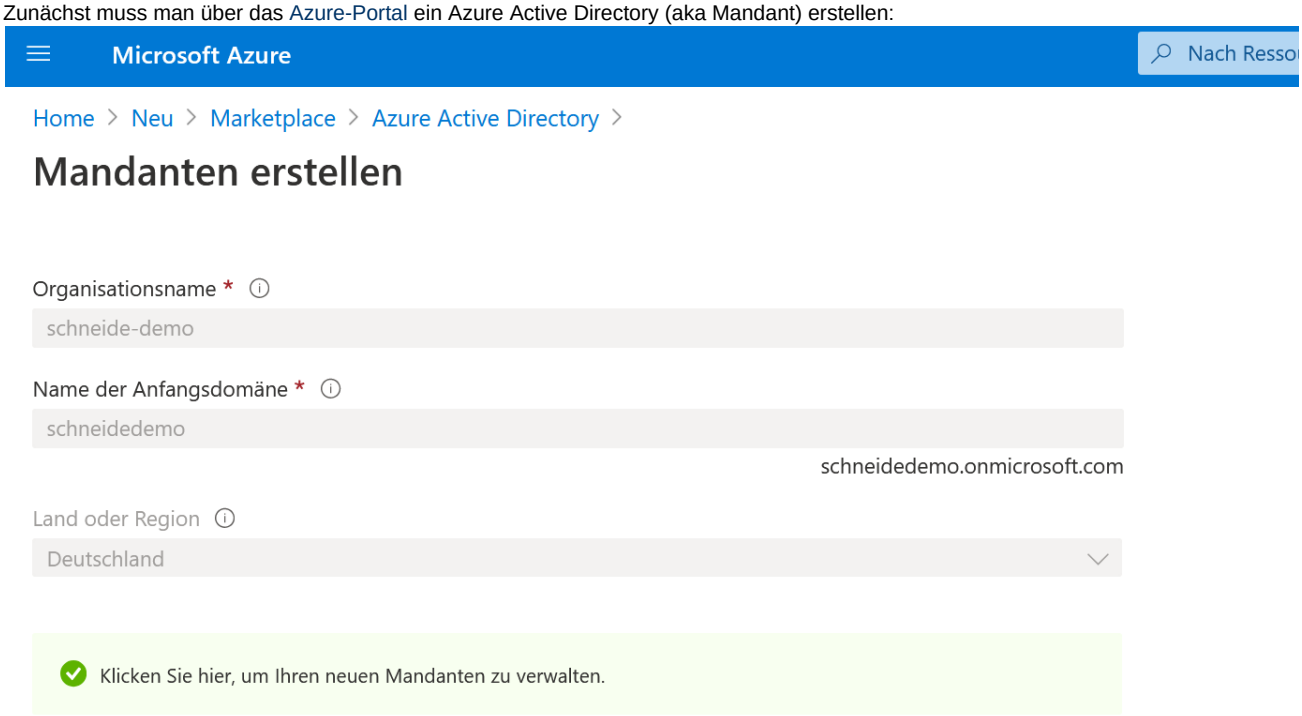
Entwicklerdokumentation

Damit man die Authentifizierung einer Webanwendung über Microsoft's Azure Active Directory abwickeln lassen kann, muss sowohl auf der Azure-Seite als auch anwendungseitig einiges konfiguriert werden. Dabei soll dieses Dokument helfen.

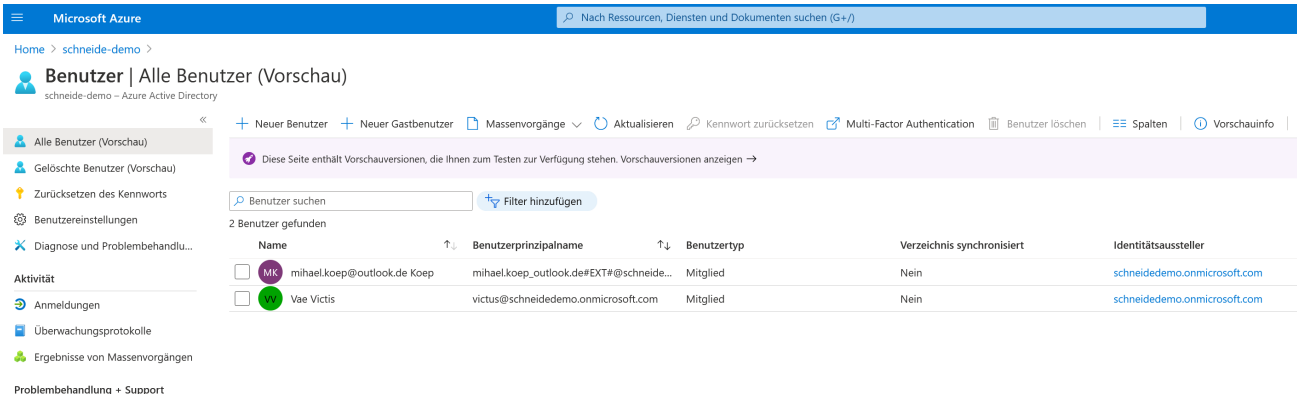
Azure Active Directory

Man benötigt (natürlich) einen Microsoft Account. Ein Microsoft Live/Outlook-Account funktioniert auch, so dass man auch als Entwickler gut entwickeln und testen kann, ohne Produktiv-Instanzen zu beeinflussen oder administrieren zu müssen.

1. Zunächst muss man über das Azure-Portal ein Azure Active Directory (aka Mandant) erstellen:

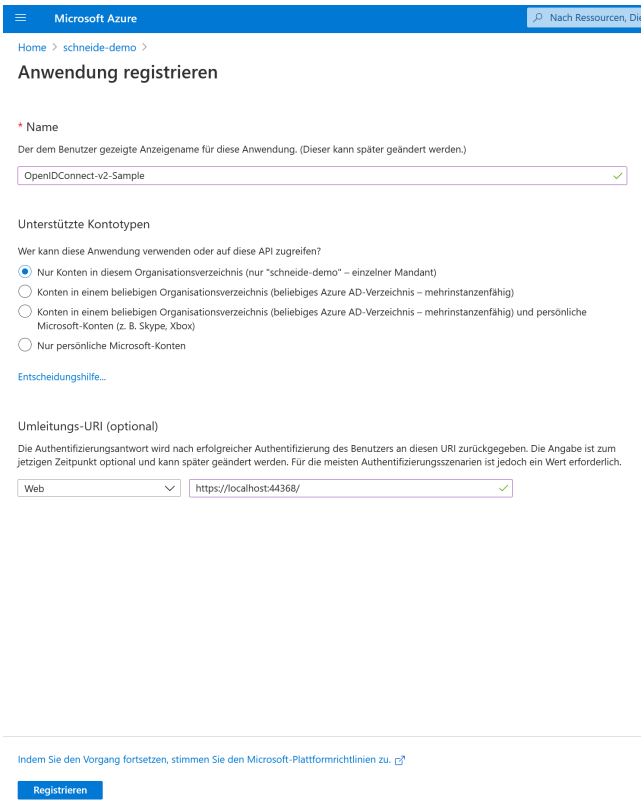


2. Dann zumindest einen Testnutzer erstellen:



Die Oberfläche ist an dieser Stelle recht intuitiv und spricht für sich selbst.

3. Nun sollte man eine **App-Registrierung** anlegen. Dabei muss man sich entscheiden, ob die Applikation nur authentifizierte Benutzer eines einzigen Azure Active Directories aktzeptiert ("Single-Tenant", einzelner Mandant), oder Benutzer beliebiger AADs ("Multi-Tenant", mehrinstanzenfähig). Damit nicht jeder Benutzer mit einem Microsoft-Konto auf die Anwendung zugreifen kann, hinterlegt man in der Anwendung die akzeptierten ADs. Benutzer aller anderen werden abgewiesen. Die Konfiguration für diesen Fall ist nachfolgend beschrieben.



Der Name kann frei gewählt werden, sollte aber leicht in Bezug mit der entsprechenden Anwendung gebracht werden können. Die Umleitungs-URI ist dann die URL auf die der Nutzer nach erfolgreichem Login weitergeleitet wird und damit typischerweise die Adresse der eigenen Anwendung.

> **In der Entwicklung**
>
> In der Entwicklung ist die Umleitungs-URI normalerweise eine Adresse auf `localhost`. Die URL sollte das HTTPS-Protokoll unterstützen, verwendet man IIS Express muss der **Port mit 443 beginnen**!

Bei der App-Registrierung muss man sicherstellen, dass man den "Implicit Grant" ("Implizite Gewährung") für das ID-Token erlaubt.

Anbindung von AAD/SSO für eine normale ASP.NET Anwendung

Die Azure Active Directory Anbindung erfordert eine OWIN-Kompatible ASP.NET-Anwendung, da nur hier die Konfiguration bequem möglich ist. Diese Konfiguration erfolgt in der `Configuration()`-Methode `Startup`-Klasse mit den gewohnten Mitteln. Durch unsere AzureActiveDirectoryAuth-Bibliothek ist sehr wenig Code notwendig, um die Anbindung durchzuführen und die Authentifizierungs-Informationen abzufragen.

Authentifizierung

1. Erzeugung einer OWIN-Kompatiblen Startup-Klasse damit man die Authentifizierung konfigurieren kann. Dazu muss man OWIN bzw. Microsoft.Owin als Abhängigkeit hinzufügen.
2. Dann muss die ganze Authentifizierung konfiguriert werden. Dazu werden einige Konfigurationswerte benötigt. Da diese Ids und Secrets für das Vertrauensverhältnis zwischen der Anwendung (aka Service Provider, SP) und dem Authentifizierungsdienst (aka Tenant, auch Identity Provider, IDP) wichtig sind, sollten sie nicht kompromittiert werden und auch nicht in der Versionskontrolle landen. Um die Einstellungen von der Anwendungskonfiguration in Web.config zu trennen, kann man folgende Anpassung machen:

web.config

```
<configuration>
  <appSettings file="PrivateSettings.config">
    <add key="webpages:Version" value="3.0.0.0"/>
    <add key="webpages:Enabled" value="false"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
  </appSettings>
...
</configuration>
```

Damit verweist man auf eine weitere Datei die "appSettings" enthält. Die Datei sieht dann folgendermaßen aus:

PrivateSettings.config (Multi-Tenant)

```
<?xml version="1.0" encoding="utf-8"?>
<appSettings>
    <add key="ClientId" value="aaaaaaaa-1111-2222-3333-bbbbbbbbbbbb"/>
    <add key="TenantId"   value="cccccccc-4444-5555-6666-dddddddddddd"
/>
<!-- This is an example for a multi-tenant application, that defines
another allowed tenant -->
    <add key="SecondAllowedTenantId" value="cccccccc-9999-8888-7777-
ddddddddddd" />
<!-- For multi-tenant applications you can replace organizations with
"common" to allow personal accounts, too. -->
    <add key="Authority" value="https://login.microsoftonline.com
/organizations/v2.0" />
    <add key="redirectUri" value="https://localhost:44355/"/>
    <add key="ClientSecret" value="_7yQ_F2v-oPKP9.l9DpzYI.GS2280kc9W9"
/>
</appSettings>
```

Die `ClientId` entnimmt man der **App-Registrierung** und den `Tenant` der Mandanten-ID des entsprechenden Azure Active-Directory. Beides kann man auch im Nachhinein im Azure-Portal einsehen. Das ClientSecret kann man auch in der App-Registrierung erstellen, mit einem Ablaufdatum versehen - sofern man möchte - und in diese private Konfiguration übertragen.
Benötigt man nur die Authentifizierung gegen **ein** Azure AD (aka "Single-Tenant"-Anwendung) sie die Konfiguration der `Authority` etwas anders aus:

PrivateSettings.config (Single-Tenant)

```
<?xml version="1.0" encoding="utf-8"?>
<appSettings>
    <add key="ClientId" value="aaaaaaaa-1111-2222-3333-bbbbbbbbbbbb"
/>
    <add key="TenantId" value="cccccccc-4444-5555-6666-dddddddddddd"
/>
<!-- Use urls like below for single tenant applications -->
    <add key="Authority" value="https://login.microsoftonline.com/{0}
/v2.0" />
    <add key="redirectUri" value="https://localhost:44355/" />
    <add key="ClientSecret" value="_7yQ_F2v-oPKP9.l9DpzYI.GS2280kc9W9"
/>
</appSettings>
```

3. Durch unsere AzureActiveDirectoryAuth-Bibliothek, die einfach in den Projektabhängigkeiten referenziert werden kann, ist der notwendige Konfigurations-Code minimal:

```
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
                // For multi-tenant applications populate this list
with all the tenants where authenticated users are
                // allowed to access this application. This
information can come from any source like configuration,
                // database, some other webservices etc.
                // For single-tenant applications leave the list
empty or do not set the ValidTenants property of
                // the OpenIdSettings.
                var allowedTenants = new List<string>
                {
                    System.Configuration.ConfigurationManager.
AppSettings["SecondAllowedTenantId"],
                    System.Configuration.ConfigurationManager.
AppSettings["TenantId"],
                };
        app.ConfigureOpenIdAuthentication(new OpenIdSettings
        {
            ClientId = System.Configuration.ConfigurationManager.
AppSettings["ClientId"],
            TenantId = System.Configuration.ConfigurationManager.
AppSettings["TenantId"],
            Authority = System.Configuration.ConfigurationManager.
AppSettings["Authority"],
            RedirectUri = System.Configuration.ConfigurationManager.
AppSettings["redirectUri"],
                        ClientSecret = System.Configuration.
ConfigurationManager.AppSettings["ClientSecret"],
        });
    }
}
```

Gewährung der Zugriffsrechte auf die Gruppeninformationen im AAD

Damit die Anwendung die Gruppen und deren Namen eines Benutzers über die Graph API von Microsoft erfragen kann, benötigt sie den **Scope** `Group.Read.All`. Für diesen muss **einmalig** ein Administrator des AADs auf das die Anwendung Zugriff erhalten soll, diesen Zugriff erlauben. Versucht man sich einzuloggen, ohne diesen "Admin Consent", so bekommt eine entsprechende Aufforderung:

Wenn man nun zu einem AAD-Administrator-Account wechselt, kann man die Applikation freischalten:

> Ob/wie das ohne Interaktion mit der Anwendung geht, ist aktuell unbekannt.

Verwendung in Controllern/ASP-Seiten

Nachdem die Anbindung an AAD erfolgt ist, kann man mit einigen wenigen Mitteln flexibel auf den Authentifizierungsstatus des Benutzers reagieren.

Abfrage des Authentifizierungsstatus

Überall, wo es sinnvoll/notwendig ist, zu prüfen, ob der Request von einem authentifizierten Nutzer kommt, kann man das direkt über ein Property am `Request`-Objekt abfragen:

```
if (Request.IsAuthenticated)
{
  ...
}
```

Wenn ein Benutzer authentifiziert ist, bekommt man seine, im AAD hinterlegten Informationen über seine Identität, die auch in Controllern und ASP-Seiten automatisch verfügbar ist:

```
var userClaims = new AadInformation(User.Identity as ClaimsIdentity);

ViewBag.Name = userClaims.Name;
ViewBag.Username = userClaims.Username;
ViewBag.TenantId = userClaims.TenantId;
ViewBag.Groups = string.Join(", ", userClaims.Groups);
```

Authorisierung über AAD-Gruppen

Damit man ein einfaches Authorisierungskonzept umsetzen kann, ist ein gängiger Weg Benutzergruppen zu definieren, die unterschiedliche Berechtigungen haben. Auch hier sind Einstellungen im Azure Portal und der Webanwendung vorzunehmen:

Gruppen im Azure Portal erzeugen

Man navigiert zum entsprechenden Mandanten und wählt das "Blade" Gruppen. Dort können die Gruppen und ihre Mitglieder verwaltet werden. Gruppen sind dabei hierarchisch und können also auch Untergruppen beinhalten.

Nicht mehr notwendig

Normalerweise müsste man jetzt bei der App-Registrierung in der Tokenkonfiguration einen "Gruppenanspruch" (GroupClaim) hinzufügen, damit die Gruppen des authentifizierten Nutzers auch an die Webapplikation übertragen werden. Da dadurch jedoch nur die GUIDs der Gruppen im Token mitübertragen werden, können wir uns diesen Schritt sparen. Wir fragen in jedem Fall die MS Graph API an, um

- Alle Gruppen des Nutzers zu bekommen, nicht nur die ersten 150.
- Die Namen der Gruppen zu kommen.

Authorisierung in der Webanwendung über Gruppen realisieren

Nicht mehr notwendig

Hat man nur die Gruppen-GUIDs zur verfügung, weil man die Graph-API nicht nutzen kann oder will, muss man die Bedeutung der Gruppen-GUIDs in der Konfiguration der Anwendung hinterlegen:

```
<appSettings>
  <add key="GroupAdmin" value="74b69a88-e7da-4061-9533-7fb22b529d16"
/>
  <add key="GroupUser" value="5b135fdd-86ee-4549-a008-5f11918c9cbd"
/>
</appSettings>
```

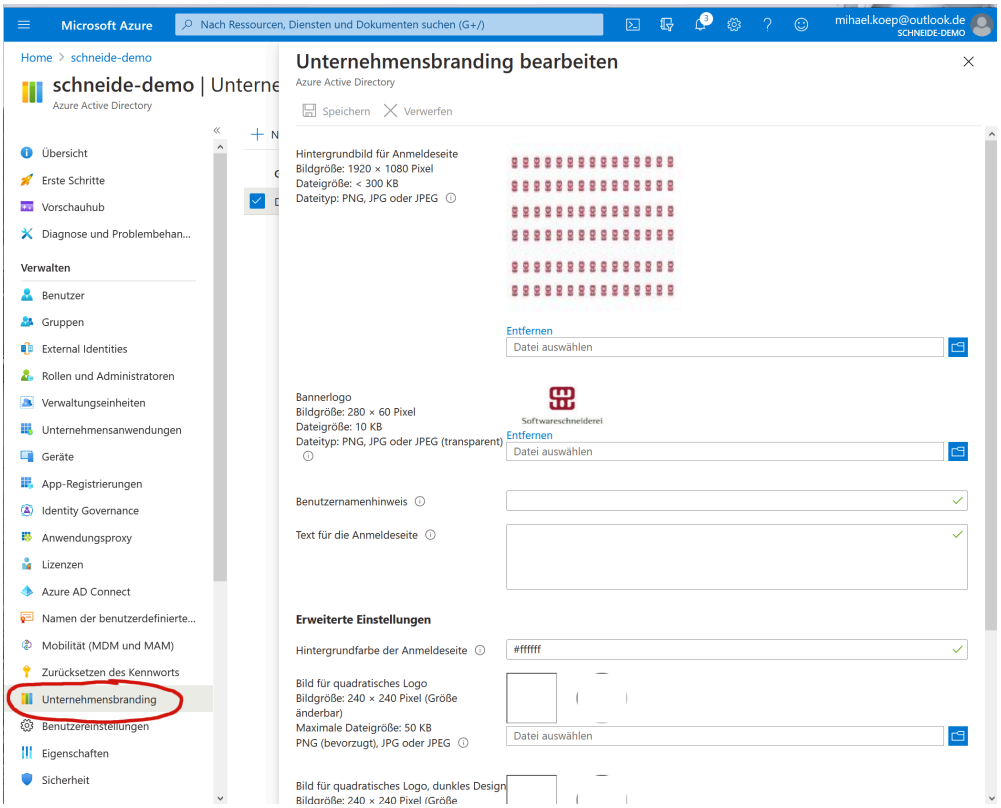
Da wir die für die Authorisierung die Gruppennamen verwenden, ist dies **nicht** mehr notwendig!

Nach erfolgreicher Authentifizierung hat man zugriff auf den eingeloggten Benutzer und kann an den entsprechenden Stellen erfragen, ob er Mitglied der entsprechenden Gruppe ist:

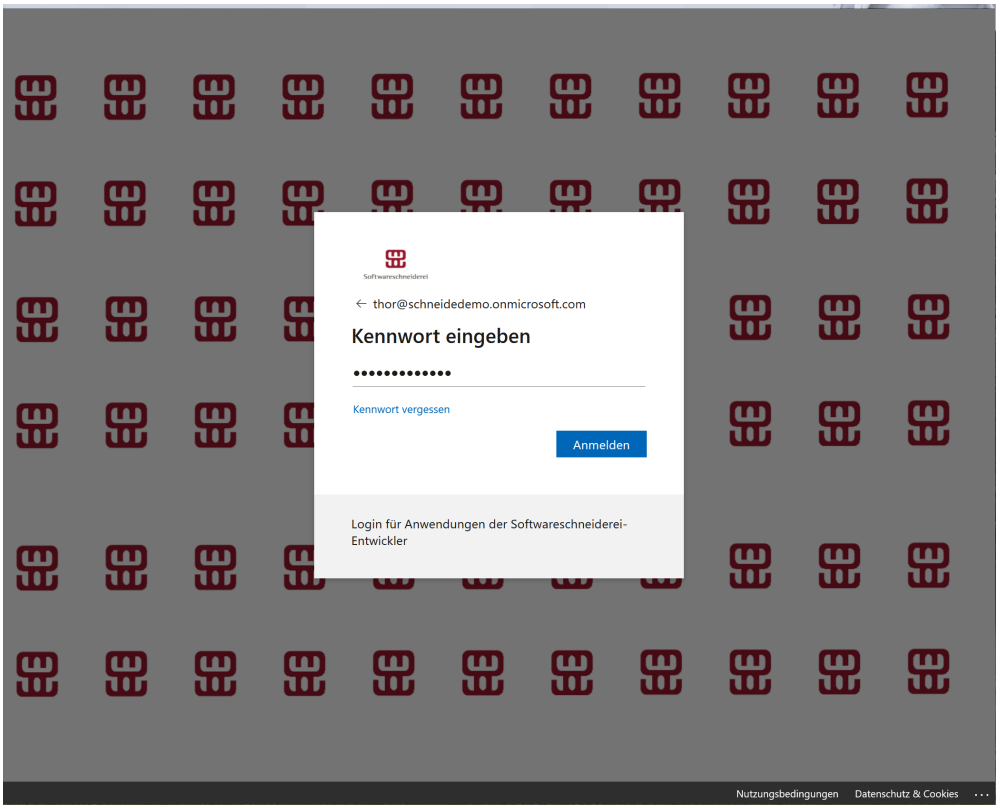
```
if (!Request.IsAuthenticated) return View();

var user = new AadInformation(User.Identity as ClaimsIdentity);
// Gruppen des aktuellen Nutzer anzeigen
ViewBag.Groups = string.Join(", ", user.Groups);
if (user.IsMemberOf(GroupConfiguration.Admin))
{
    // Dinge, die nur Admins sehen dürfen definieren
}

return View();
```

Anpassung der Login-Seiten

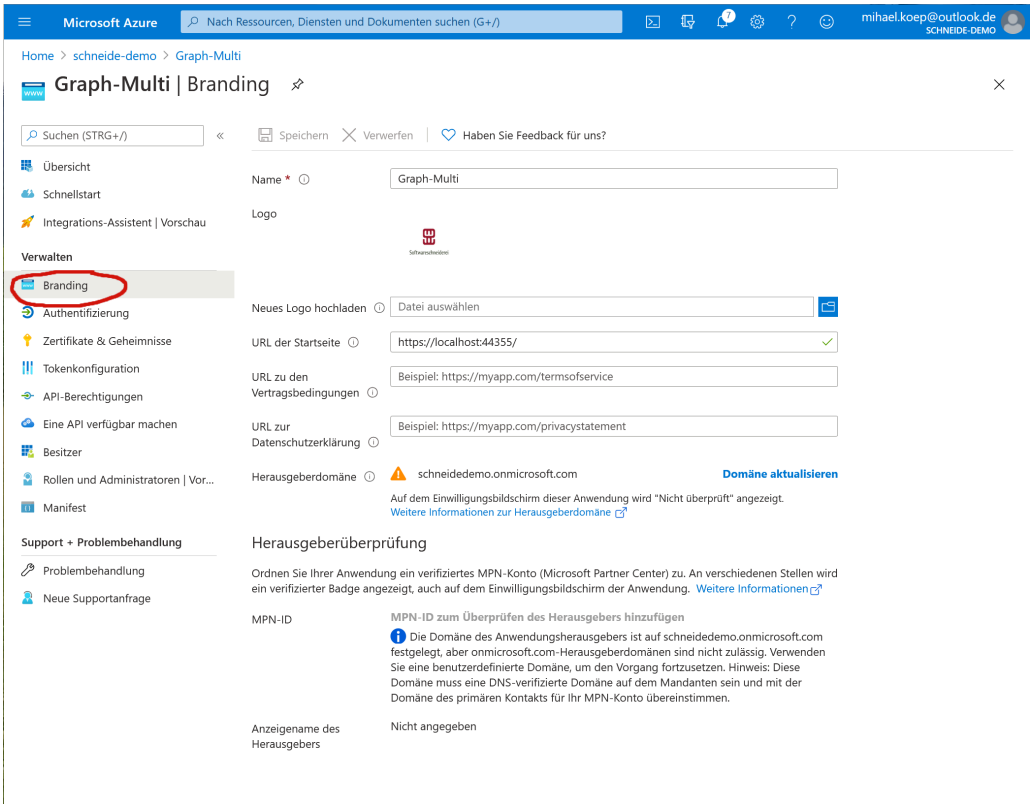
Es passiert leicht, dass Anwender verunsichert sind, wenn sie von der Anwendungsseite auf Anmeldungsseiten von Microsoft umgeleitet werden, ohne dass diese einen Hinweis bieten, dass es beim Login immernoch um die gewünschte Anwendung geht. Microsofts Azure bietet deswegen Möglichkeiten, die Gestaltung der Masken mit dem Anwendungs/Unternehmensbranding zu versehen. Im Azure-Portal kann man das für jedes AD gesondert konfigurieren:



Das führt dann zu entsprechend angepassten Anmeldeseiten, sobald die Domäne/der Tenant des Nutzers (also das AD, gegen das er sich authentifiziert) bekannt ist:



Des Weiteren ist es möglich, für jede App-Registrierung ein Branding zu hinterlegen:



Wo überall die hier hinterlegten Eigenschaften angezeigt werden, ist noch ungeklärt, höchst wahrscheinlich jedoch mindestens beim gewähren der Zugriffsrechte auf Kontoinformationen für die Applikation.