**ASP.Net Core Identity - Research**

This is research associated to ASP.Net Cores Identity Server 4. The purpose of this investigation is to understand how I can learn about and implement an accounts handling solution into our website. It will be used to manage accounts for administrators.

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.0&tabs=visual-studio>

<https://docs.microsoft.com/en-us/azure/active-directory/develop/>

Angular Samples:

<https://docs.microsoft.com/en-us/azure/active-directory/develop/sample-v2-code#single-page-applications>

Overview video:

<https://www.youtube.com/watch?v=y_fgJAatVhk&feature=youtu.be>

By using Identity we can create suitable administration accounts that can be used on both the website (to manage users accounts, scenarios and more) and in azure (to monitor resources in the event of an issue with the system).

There are tutorials and examples of how to implement this into an Angular website so the material is out there.

**Authentication** is the process of proving you are who you say you are. Authentication is sometimes shortened to AuthN.

**Authorization** is the act of granting an authenticated party permission to do something. It specifies what data you're allowed to access and what you can do with that data. Authorization is sometimes shortened to AuthZ.

### **Tenants**

A cloud identity provider serves many organizations. To keep users from different organizations separate, Azure AD is partitioned into tenants, with one tenant per organization.

Tenants keep track of users and their associated apps. The Microsoft identity platform also supports users that sign in with personal Microsoft accounts.

Azure AD also provides Azure Active Directory B2C so that organizations can sign in users, typically customers, using social identities like a Google account. For more information, see [Azure Active Directory B2C documentation](https://docs.microsoft.com/azure/active-directory-b2c) .

### **Security tokens**

Security tokens contain information about users and apps. Azure AD uses JSON based tokens (JWTs) that contain claims. A claim provides assertions about one entity to another. Applications can use claims for various tasks such as:

* Validating the token
* Identifying the subject's directory tenant
* Displaying user information
* Determining the subject's authorization

A claim consists of key-value pairs that provide information such as:

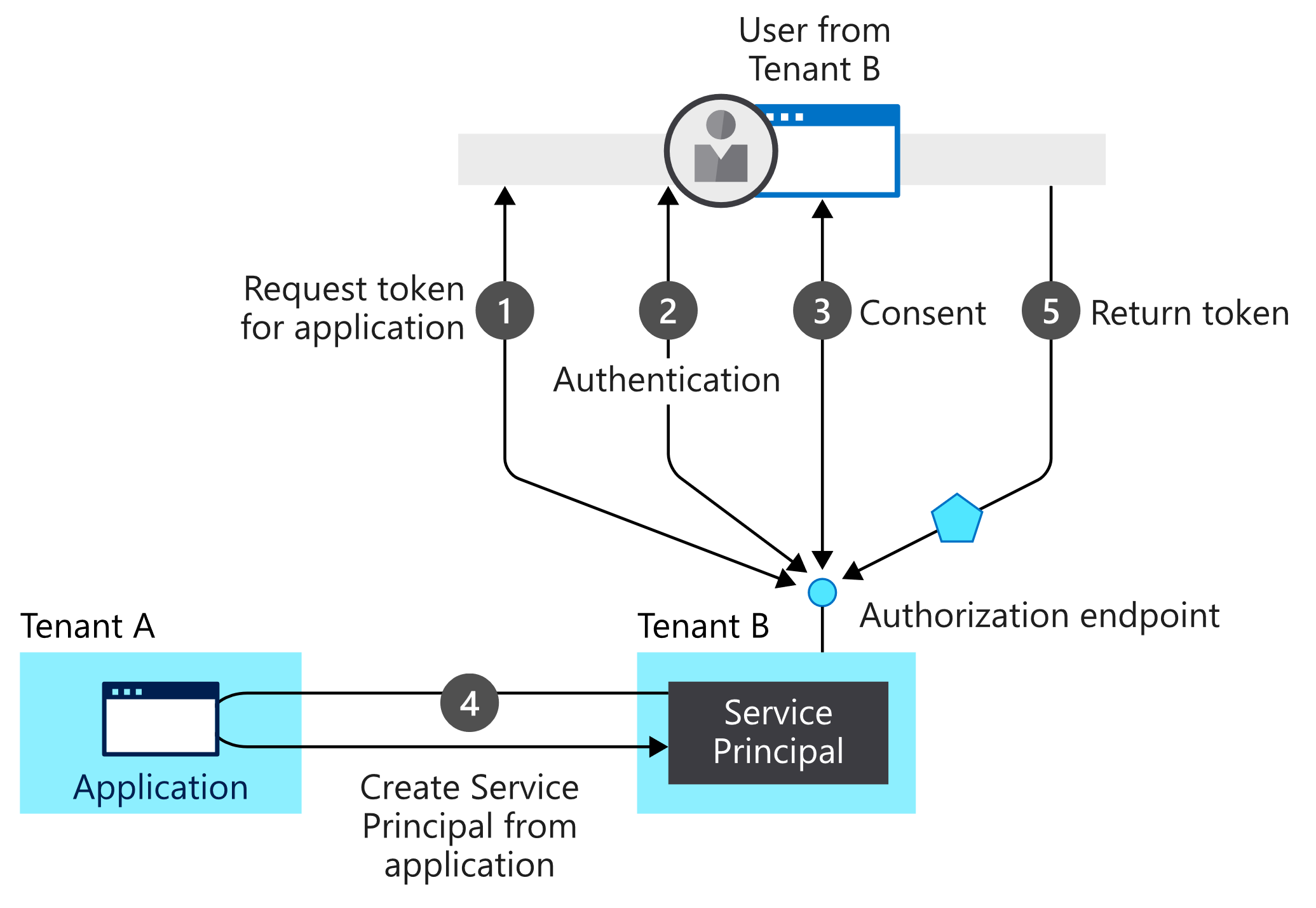
* the Security Token Server that generated the token.
* the date when the token was generated.
* the subject, such as the user (except for daemons).
* the audience, which is the app for which the token was generated.
* the app (the client) that asked for the token. In the case of web apps, this may be the same as the audience.

For more detailed claim information, see the [access tokens](https://docs.microsoft.com/en-us/azure/active-directory/develop/access-tokens) and [ID tokens](https://docs.microsoft.com/en-us/azure/active-directory/develop/id-tokens).

It's up to the app for which the token was generated, the web app that signed-in the user, or the Web API being called, to validate the token. The token is signed by the Security Token Server (STS) with a private key. The STS publishes the corresponding public key. To validate a token, the app verifies the signature by using the STS public key to validate that the signature was created using the private key.

Tokens are only valid for a limited amount of time. Usually the STS provides a pair of tokens: an access token to access the application or protected resource, and a refresh token used to refresh the access token when the access token is close to expiring.

Access tokens are passed to a Web API as the bearer token in the Authenticate header. An app can provide a refresh token to the STS, and if the user access to the app wasn't revoked, it will get back a new access token and a new refresh token. This is how the scenario of someone leaving the enterprise is handled. When the STS receives the refresh token, it won't issue another valid access token if the user is no longer authorized.



In this provisioning flow:

1. A user from tenant B attempts to sign in with the app, the authorization endpoint requests a token for the application.
2. The user credentials are acquired and verified for authentication.
3. The user is prompted to provide consent for the app to gain access to tenant B.
4. The Microsoft identity platform uses the application object in tenant A as a blueprint for creating a service principal in tenant B.
5. The user receives the requested token.