Authentication in ASP.NET Core 2

# Identity and Claims

From [Wikipedia](https://en.wikipedia.org/wiki/Claims-based_identity): A claim is a statement that a subject (which is a person or an organization) makes about itself or some other subject. Basically a statement of, “This is who I am,” or “this is who Xxx is.” The claims come as a set of name-value pairs, and can contain literally any information—usually a username, a first name and last name, and other basic user info. Claims can also contain groups, buying preferences, ethnicity, sets of privileges, associations, or any other capabilities.

Claims only provide authentication, i.e.: what the subject is or is not. They do not provide info about what a subject may or may not do. So a claim can say that a user is a “helpdesk admin”, but it is up to the application receiving the claims to decide what each claim means (or to ignore it, altogether), and what a “helpdesk admin” can or cannot do.

Definition of terms:

* **Claim**: a set of key-value pairs that describe who a subject is, or is not. Claims can be thought of as properties or attributes related to a subject’s identity.
* **Subject**: a person or an organization.
* **Provider**: a subject making one or more claims.
* **Security Token Service**: a third party who handles the actual authentication (sign-in), and issues claims about a subject, usually in the form of a token.

Because we’ve split authentication (what a subject is / is not) from authorization (what a subject can / cannot do), we can now hand off authentication to a third party.

# Security token service

From Wikipedia:

To better understand the concept of security token service, consider the analogy of a night club with a doorman. The doorman wants to prevent under-age patrons from entry. To facilitate this he requests a patron to present a driver's license, health insurance card or other identification (the token) that has been issued by a trusted third party (the security token service) such as the provincial or state vehicle license department, health department or insurance company. The nightclub is thus alleviated of the responsibility of determining the patron's age. It only has to trust the issuing authority (and of course make its own judgment of the authenticity of the token presented). With these two steps completed the nightclub has successfully authenticated the patron with regard to the claim that he or she is of legal drinking age.

Continuing the analogy, the nightclub may have a membership system, and certain members may be regular or VIP. The doorman might ask for another token, the membership card, which might make another claim; that the member is a VIP. In this case the trusted issuing authority of the token would probably be the club itself. If the membership card makes the claim that the patron is a VIP, then the club can react accordingly, translating the authenticated VIP membership claim to a permission such as the patron being permitted to sit in the exclusive lounge area and be served free drinks.

Questions / observations:

* I’m not sure how the second part of this works…
  + does the application look at the subject’s claims and then *modifies* the list of claims (adding / deleting, as appropriate)?
  + Or is there a secondary Security Token Service that is part of the application, itself, which issues a second token that contains claims which grant permissions…?
* There is this concept of “claims acquisition”, where a user is authenticated by a security token service, but (?) will receive further claims later on?

# So what? Why do it this way?

Again, Wikipedia says it best (I’ve broken it down):

Claims-based identity has the potential to simplify authentication logic for individual software applications, because

1. those applications don't have to provide mechanisms for account creation, password creation, reset, and so on.
2. claims-based identity enables applications to know certain things about the user, without having to interrogate the user to determine those facts.
3. The facts, or claims, are transported in an "envelope" called a secure token [which is standardized, and can be decoded using commonly-available libraires/algorithms].
4. Claims-based identity can greatly simplify the authentication process because the user doesn't have to sign in multiple times to multiple applications.
5. A single sign in creates the token which is then used to authenticate against multiple applications, or web sites.
6. In addition, because certain facts (claims) are packaged with the token, the user does not have to tell each individual application those facts repeatedly, for instance by answering similar questions or completing similar forms.

So, goodness all around!

Further reading:

<https://andrewlock.net/introduction-to-authentication-with-asp-net-core/>

# ASP Data Types

Ok, so let’s go over all the 40 some-odd types (JK…)

Here is some UML, since I speak UML:

Claim

ClaimsIdentity

ClaimsPrinciple

A ClaimsPrinciple is kind of a container for one or more ClaimsIdentity instancess. This is the object that you’ll be working with when dealing with a user. Think of it as a portfolio of all the claims and identities that a user can exercise.

Some significant members in this class:

* Identities: a collection of one or more ClaimsIdentity instances.
* Identity: the first member in the Identities collection, usable as the primary identity for the user.
* Claims: I assume this is a union of all claims (?)
* IsInRole(): roles are deprecated! This is more or less used for backwards compatibility.

A ClaimsIdentity as a piece of identification, like a driver’s license, or a passport, or a membership card.

Important members:

* AuthenticationType: describes in human-readable terms what kind of ID this is. It’s a name for the ID
* IsAuthenticated: indicates that this user has signed in and really is bona-fide. If this property is false, then AuthenticationType will be set to null. If this property is true then AuthenticationType must contain a name.
* Claims: a collection of claims