Security back in the day for applications:

1. **Thick client applications** used Windows Authentication
2. **Server-sided applications** used Windows or Form Authentication

* With the advent of RESTful APIs, we now have predominantly client-side applications
* A lot of these APIs are public, therefore Windows and Form authentication is no longer a safe way of securing these apps.
* To solve this problem, **token based security** was created
* Tokens represent **consent** given by the user to the client application to access the API
* The token can then be used by the API to grant access
* Then the question arose: **how do we create these tokens and safely deliver them to the applications that require them?**
* Initially, there were home-grown token services which were calls to a self-created login endpoint that would take username + password and return a JSON web token and then distribute that token to APIs, but that still required us to share username + password with that API
* Once we start creating endpoints ourselves, we start reinventing the wheel, which leaves room for much error
* That is why a **Central Identity Provider** was created
* It’s the responsibility of an IDP (Identity Provider) **(not the application itself)** to authenticate the user and, if needed, safely provide identity to an application
* **Tasks of the IDP:**

1) User registration and management

2) Locking out users

3) Password policies, strengths and resets

**These are all tedious tasks, prone to change which leaves room for a lot of error,** that’s why we rely on IDP to handle all these tasks for us

- Handle all of the above tasks at IDP level and **re-use them across applications**

**-** User accounts can be reused across applications, and the tasks related to this accounts can be common for all applications and **not** application specific (like changing a password)

- Safely storing account-related info (by all kinds of encryption) is prone to change

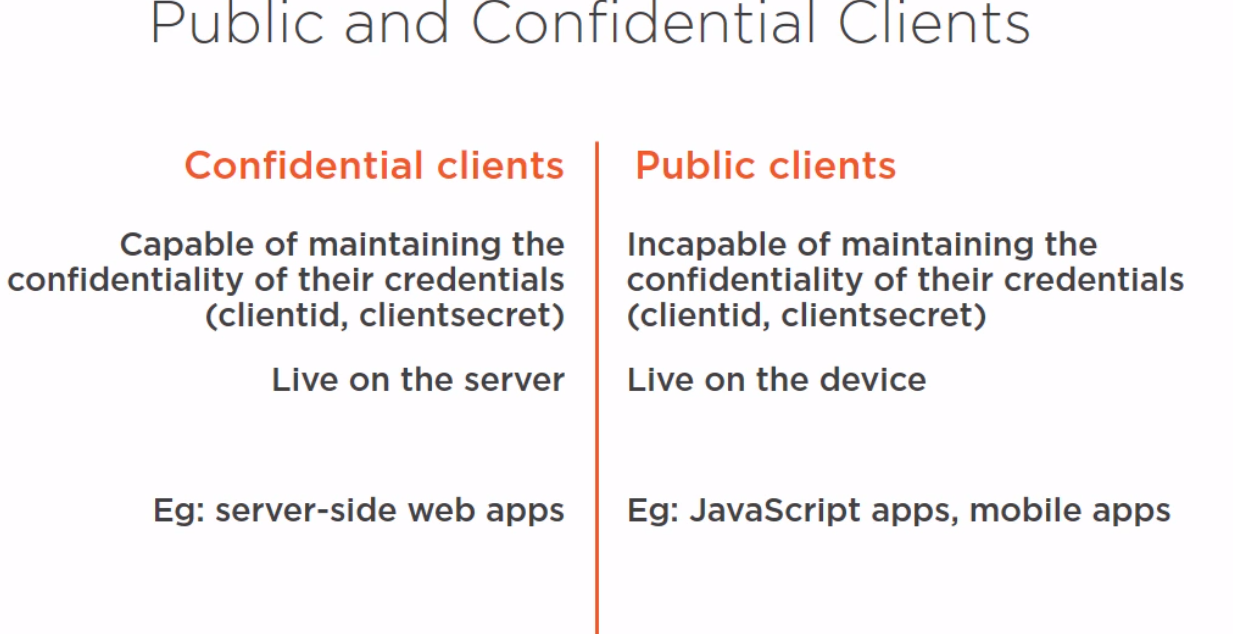
- Means of authentication are prone to change

- Because of all these things mentioned above, it is very essential to offload these tasks to a **central identity provider**

**OAuth2** - an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications

* OAuth2 allows a client application to gain access to an API by using an **access token**
* OAuth2 also defines how a client application can get an access token from a security service
* Home-grown endpoints are being replaced by the endpoints from the OAuth2 standard
* OAuth2 also defines how to use these endpoints for different types of client application types
* **IdentityServer** implement the OAuth2 standard (this is what you’d also call MIDDLEWARE)
* OAuth2 only fetches access tokens for the client application to gain access to an API - this is **authorization**, so it’s only half of the puzzle
* **Authentication** must be handled by **Open ID Connect (OIDC) -** this is also what you’d call MIDDLEWARE
* **OIDC** is a simple identity layer on top of the OAuth2 protocol (extends OAuth2)
* With OIDC, a client application can request an **identity token** which can be used to sign in to the client application (**authentication**) while the same application can use the access token from OAuth2 to access an API

**How OpenID Connect Works**

* Client application requires a user’s identity
* Authentication request is created by the client application which redirects the user to the IDP
* The user proves who they are by providing a username + password combo. The IDP creates an **identity token** and signs it. The ID token contains the user’s verifiable identity.
* **Redirects** user to the client application, passing through the identity token
* Client application gets the token from the redirect URI. The client app now has proof of identity. This ID token is then used to create claims identity -> stored as an ID ticket in an encrypted cookie. The browser sends that cookie on each request to the client application.
* Not all client applications are created equal, which is where **flows** come in
* **Two** client types: confidential and public
* ****
* The **flow** determines how the code and/or token(s) are returned to the client

**OpenID Connect Endpoints**

* **Authorization endpoint (IDP level)** 
  + Used by the client application to obtain authentication and/or authorization, via redirection
  + SSL is a **requirement**
* **Redirection endpoint (client level)** 
  + Used by the IDP to return code & token(s) to the client application
* **Token Endpoint (IDP level)** 
  + Used by the client application to request tokens (**without** redirection, so it is done via an HTTPost) from the IDP

**Types of Flows**

1. **Authorization Code**

* Returns an authorization code from the authorization endpoint and tokens from the token endpoint
* Suitable for confidential clients and long-lived access

2) **Implicit flow**

* Returns tokens from authorization endpoint and there’s no authorization code
* Suitable for public clients (there’s no public authentication)
* No long-lived access through refresh tokens

3) **Hybrid flow**

* Returns some tokens from authorization endpoint and token endpoint
* Confidential clients
* Long-lived access

**Role-based Authorization**

**Authentication** - process of determining who you are

**Authorization -** the process of determining what you’re allowed to do

1. Role-based access control
2. Attribute-based access control

**JWT -** JSON web token