IdentityServer in Production

T339

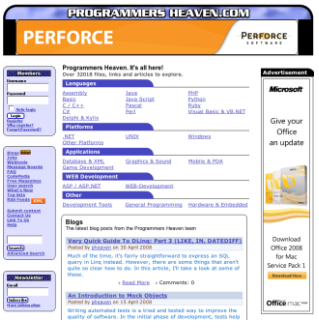
Presentation

Identity Server in Production

https://www.tn-data.se

About your teacher

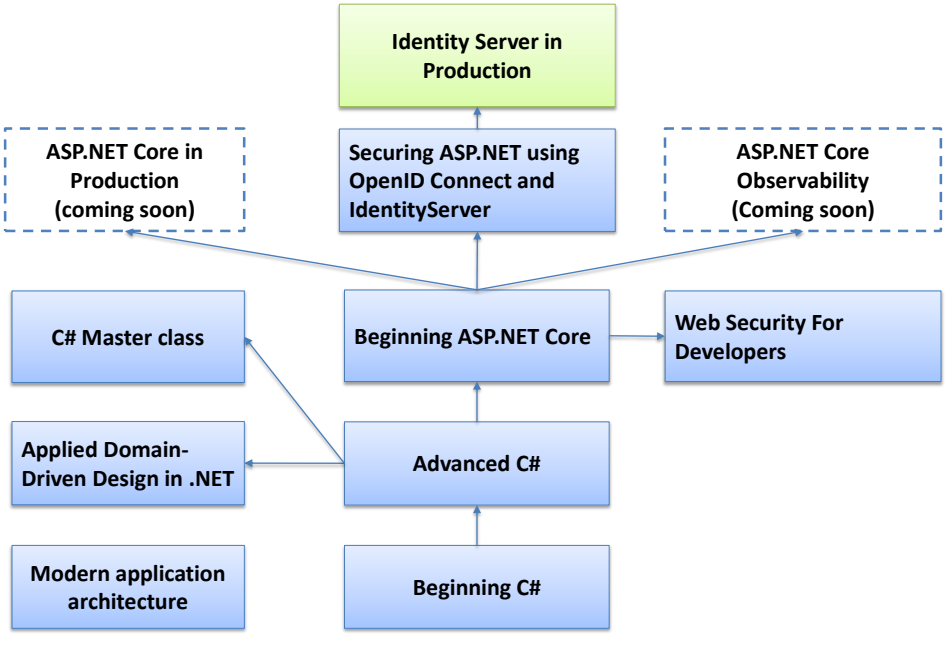
- Name: Tore Nestenius
- Been programming for over 40 years
- 1996 Created www.programmersheaven.com
  A popular website for programmers with over 750,000 monthly visitors. Left the site in 2007.
- 2010, co-founded Edument AB https://www.edument.se
- Spends time on Stack Overflow



About your teacher

- Today works at TN Datakonsult AB
- Teaches and coaches developers all over the world
- Focus on C#, .NET, Security, Architecture, Identity
- Helps companies to be more productive
- Web: https://www.tn-data.se/

.NET Course tree



Course prerequisites

This course assumes basic knowledge about:

- OAuth / OpenID Connect
- IdentityServer
- ASP.NET Core
  - Model, View, Controller...
  - Authentication and authorization
- HTTP(s)
  - Headers, verbs...
- Git and GitHub

Course target

This course is based on:

- Visual Studio 2019 / 2022
- ASP.NET core 5 / .NET 5
- Duende Identity Server 5

Replacement for IdentityServer4

The goals with this course:

- How to deploy to production
- Services
  - IdentityServer
  - Web client
  - API
- Security
  - HTTPS and certificates
  - Crypto Keys
  - Data Protection
- Logging
- …

How should we learn this? 

Goal

We will not:

- learn this by just configuring magic libraries
- Learn by just running it locally on your machine

We will:

- Understand the challenges, one step at the time
- Deploy live to production
- Running live on the internet in the cloud

We will learn by doing!

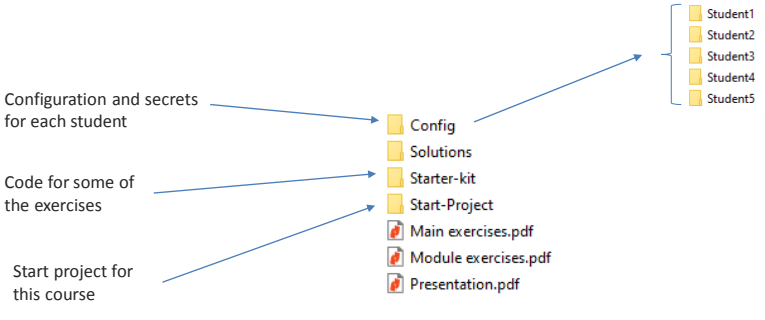
Agenda

The agenda is:

1 - Introduction
2 - Configuration
3 - HTTPS and Security
4 - Adding IdentityServer
5 - Logging
6 - Error Handling
7 - Securing the client
8 - Data Protection
9 - Private-Public crypto
10 - Keys, certificates & PKCS12
11 - Adding signing keys
12 - Database
13 - Users
14 - Tokens and claims
15 - Securing the API
16 - Securing the API – Adv.
17 - Consuming the API
18 - Refresh tokens
19 - Consuming the API – Adv.

Exercises

In the course material you will find:



Important

FEEDBACK!

We would like your feedback during and after the class:

- Suggestions for improvements?
- Missing things?
- What's good and bad?
- Coding style or code improvements

Exercises

We will be doing a **lot of exercises**

Feel free to collaborate!

Presentation

Please introduce yourself:

- Your name and company
- Role
- Background
- Experience in:
  - .NET, ASP.NET Core and web
  - OAuth/OpenID Connect / IdentityServer
  - Authentication and authorization
- Expectations on this course
  - What do you want to learn?

Exercise

Get to know each other in the breakout rooms



During the next 5 minutes, talk about:

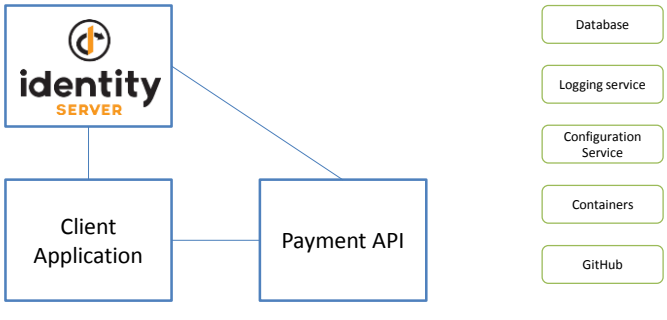
- Share a few nerd facts about yourself
- From the agenda, what do you look most forward to learn about?
- Do you have any hidden talents? What can you do?
- What's the weirdest food you've ever eaten?

Introduction

Module #1

Goal

The goal for this course is to create the following setup:

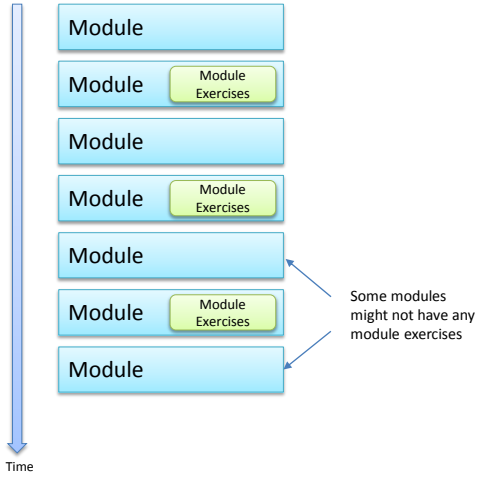


Using **best practices**, **security** and **production** in mind

How will we achieve this?

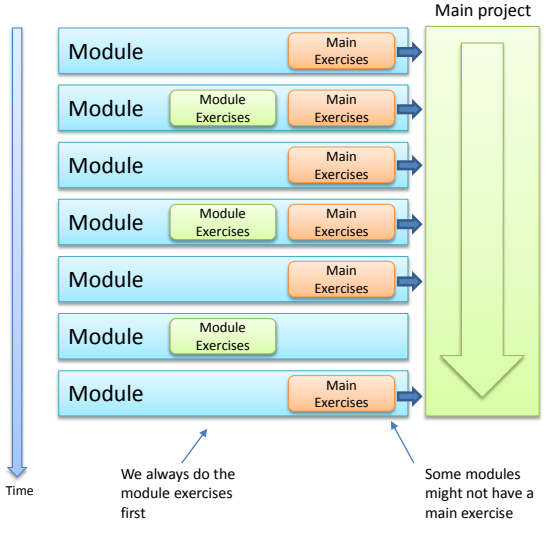
How will we achieve this?

During this course we will work through various modules



How will we achieve this?

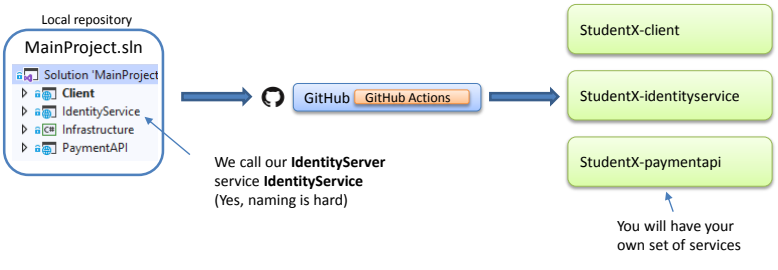
After each module exercise, we will apply what we have learned to the **main project**



Technical setup

Technical setup

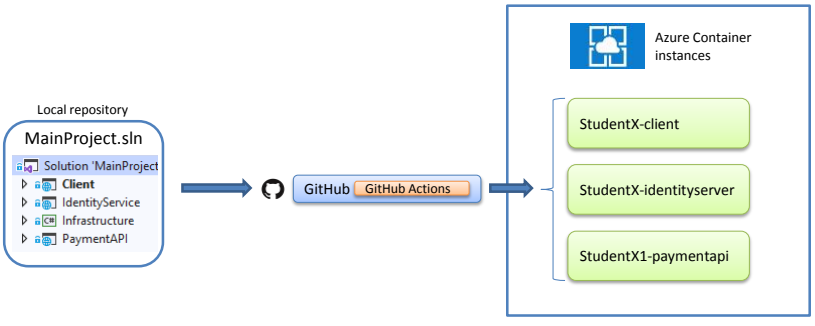
When we push our code, GitHub will deploy it to the cloud



Each service will be deployed as a separate instance

Technical setup

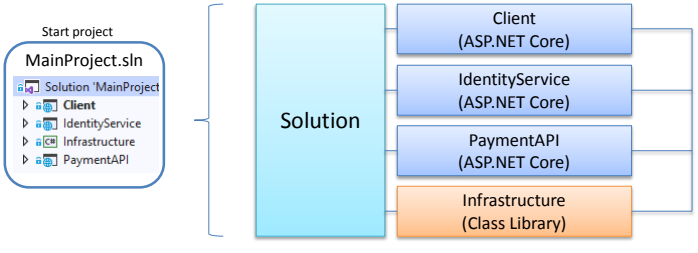
We deploy to Linux **container** instances in **Azure**



Containers and **Cloud** are not the focus in this course

Project structure

The start project source code structure:



Remote Desktop machine

For some exercises, we have prepared a **Virtual Machine**

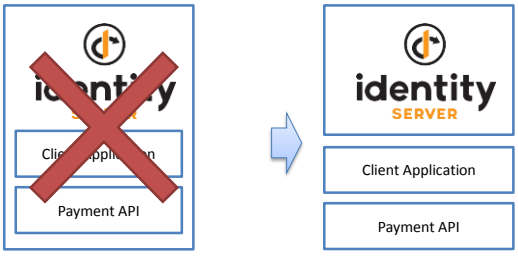
- Windows 10 Pro machine
- Hosted in Azure
- Prepared with various software

You will have one machine instance each

Common mistakes

Common mistakes

A common mistake is to place all in the same web-site:



Why?

It is really hard to troubleshoot and reason about it

Exercises

Let’s do main exercise #1.x

Discussing the exercises with your fellow students is encouraged!

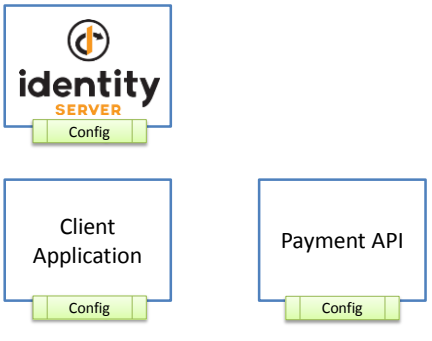
Please write in the chat when you are done

Configuration

Module #2

Configuration

Clearly our applications need configuration data



What key configuration requirements do we have?

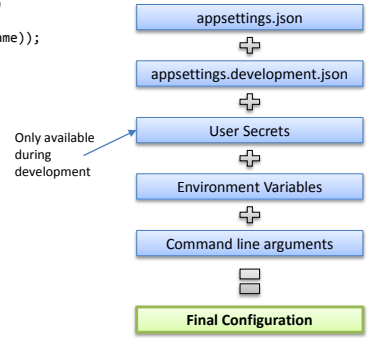
Configuration in ASP.NET Core

Configuration in ASP.NET Core

At startup the default configuration setup is:

Internally in ASP.NET Core

```
var env = hostingContext.HostingEnvironment;

config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
      .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true, reloadOnChange: true);

if (env.IsDevelopment() && !string.IsNullOrEmpty(env.ApplicationName))
{
    var appAssembly = Assembly.Load(new AssemblyName(env.ApplicationName));
    if (appAssembly != null)
    {
        config.AddUserSecrets(appAssembly, optional: true);
    }
}

config.AddEnvironmentVariables();

if (args != null)
{
    config.AddCommandLine(args);
}
```



By default, the settings in **appsettings** will reload on change

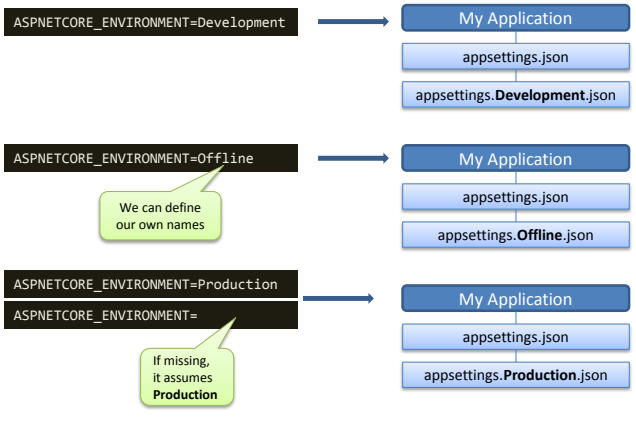
https://source.dot.net/#Microsoft.Extensions.Hosting/HostingHostBuilderExtensions.cs,5d86d5acb42aaed3

Environments

Environments

We want different config in **dev, offline** and **prod**

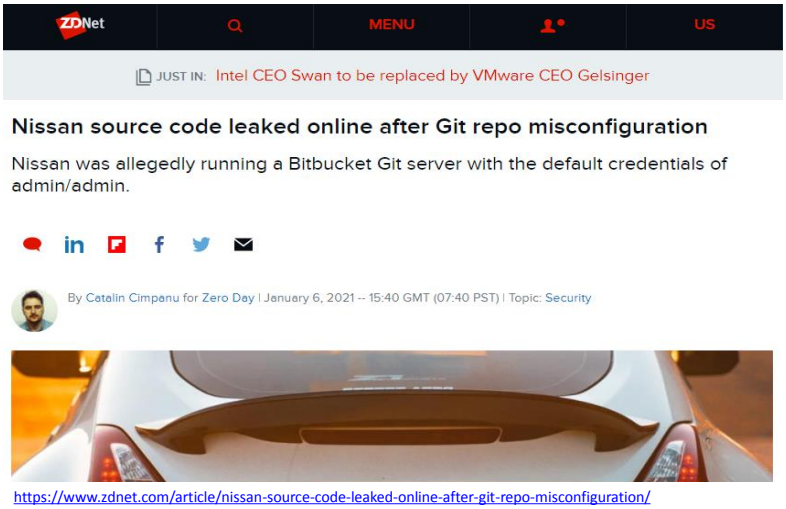
Using the **ASPNETCORE_ENVIRONMENT** variable we can define the current environment



What is the golden rule for configuration data?

Sensitive data

Never store sensitive data in the project or source files!



https://www.zdnet.com/article/nissan-source-code-leaked-online-after-git-repo-misconfiguration/

Where should we store sensitive data?

Sensitive data - locally

On our **development machine** we have several options:

- Environment variables
- Config file
  c:\TnData\AppConfig.json
- User Secrets
  %APPDATA%\Microsoft\UserSecrets\<user_secrets_id>\secrets.json

Located outside the project folder

We will explore user secrets in the exercises

Sensitive data - locally

In **production** we have slightly different options:

- Environment variables
- Inject config and settings during **build** or **start-up**
- Command line arguments
- Config file
  c:\TnData\AppConfig.json

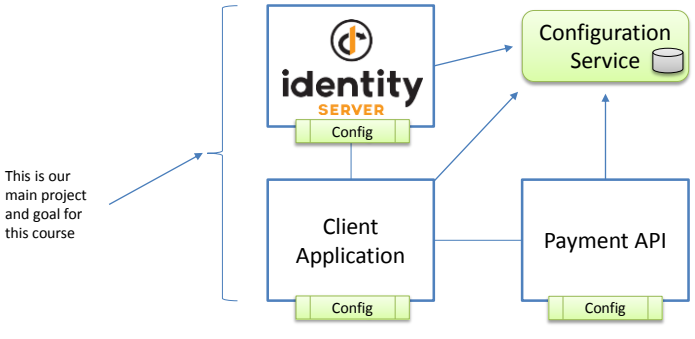
- Located outside the project folder, especially in production
- Restricted access using Operating System rights

What else can we do?

External secrets management

External secrets management

Using a **central configuration service** can simplify things

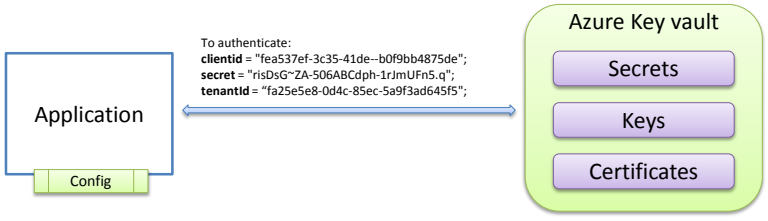


Now the amount of secrets we need to pass to our application is reduced.

External secrets management

In this course we will use **Azure Key Vault** as a config source

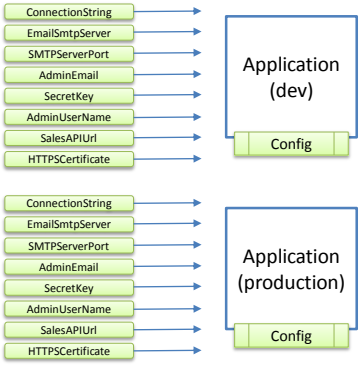
It's simple to integrate with ASP.NET Core



What is the advantage of centralized configuration?

External secrets management

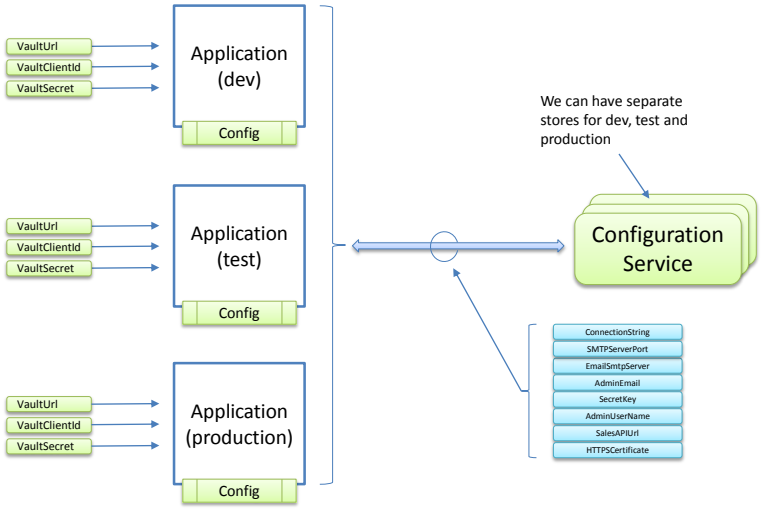
Without, we need to pass each secret, one by one:



This can be pretty tedious to maintain

External secrets management

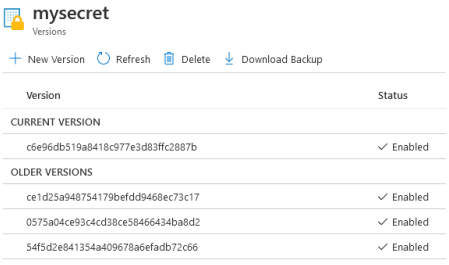
With centralised configuration management:

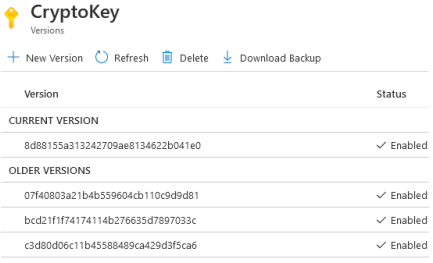


Versioning

Versioning

With AKV we get automatic versioning





We can retrieve any version we like from AKV

Azure Key Vault gotchas

Azure Key Vault gotchas

At startup, it will load each secret, one by one

With many secrets, this can slow down the startup time

Slow Startup for ASP.NET Core application using Azure Keyvault configuration provider
https://oliviervaillancourt.com/posts/AspNetCore-KeyvaultConfigurationProvider-StartupPerformance-Pitfall
Azure subscription and service limits, quotas, and constraints
https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/azure-subscription-service-limits

Exercise

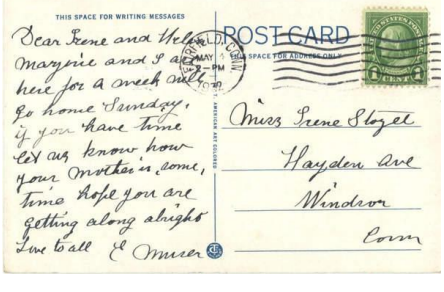
Let’s do module and main exercise #2.x

Please write in the chat when you are done

HTTPS & Security

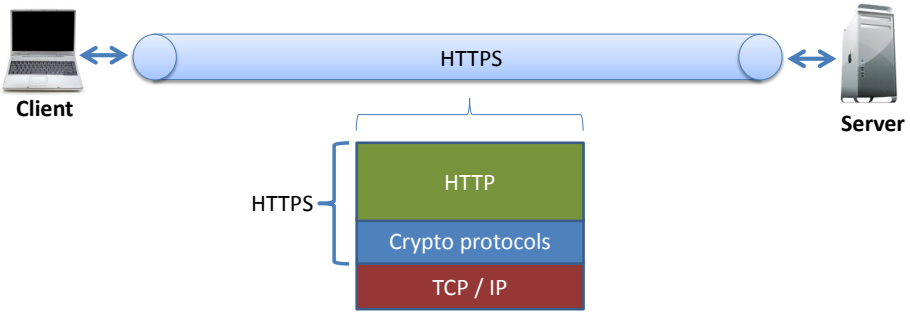
Module #3

What are the problems with HTTP?



HTTPS

HTTPS uses cryptographic protocols to create a secure channel between two machines



Technically, **HTTPS** is not a protocol in itself; rather, it is the result of simply layering **HTTP** on top of crypto protocols

HTTPS

HTTPS solves these key issues:

- **Authentication**
  Makes sure we are talking to the right server
- **Data Integrity**
  Makes sure our data can't be modified
- **Encryption**
  Prevents others from accessing my information

Cryptographic protocols

During the HTTPS connection handshake one of the following **cryptographic protocols** is selected:

Name	Comment
TLS 1.3	**Latest and most secure**
TLS 1.2	**Secure**
TLS 1.1	Deprecated in 2020
TLS 1.0	Deprecated in 2020
SSL 3	Deprecated in 2015
SSL 2	Deprecated in 2011

Brings faster, safer, and better security

To be deprecated From browsers in 2020

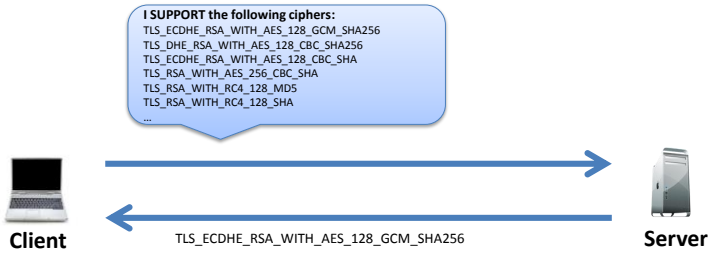
Deprecated by RFC 8996

SSL should be disabled on your server unless you really need to support older browsers

Firefox, Chrome, Safari and Edge Dropping TLS 1.0, 1.1 in 2020
https://www.tomshardware.com/news/major-browsers-deprecate-tls-1.0-1.1,37932.html

cryptographic algorithms

After the protocol is established, a **cryptographic cipher** is then selected:



The cipher suite defines:

- Key exchange algorithm
- Encryption algorithm
- Message authentication code (MAC) algorithm
- Pseudorandom function (PRF)

How can we fix the HTTP->HTTPS redirect problem?

```
GET http://www.edument.se/ HTTP/1.1
```

```
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Sun, 06 Aug 2017 16:49:56 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://edument.se/
```

HTTP Strict Transport Security

Using the **HSTS header**, we can tell the browser to always use HTTPS on this domain

This is enabled using this response header:

```
Strict-Transport-Security: max-age=15768000; includeSubDomains
```

This header is Ignored if sent over **http**

Optionally, tell the browser to remember to use HSTS

If the server forgets to include the HSTS header in a response, the browser will still use HSTS.

HTTP Strict Transport Security

Once enabled, it will:

Automatically rewrite all insecure HTTP requests to use the secure HTTPS protocol instead.

```
http://www.example.se/homepage
```

```
https://www.example.se/homepage
```

HTTP Strict Transport Security (HSTS) is specified in **RFC6797**

Certificates

Certificates

Digital certificates are used to identify people or servers over the internet.

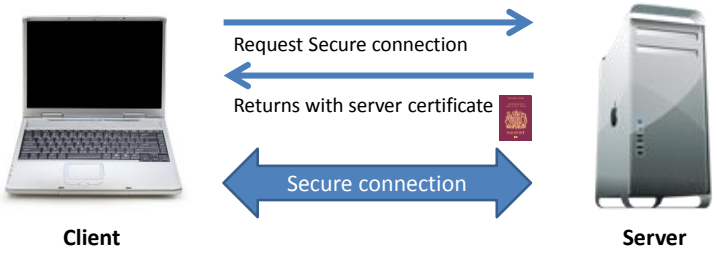
A certificate is like a **passport** that you use when you travel to another country.

A certificate is always issued by **someone we trust**, like a Certificate Authority (CA).

How does HTTPS work (simplified)

The client first requests the **server certificate**

If the client trusts the issued certificate, then a secure connection can be established



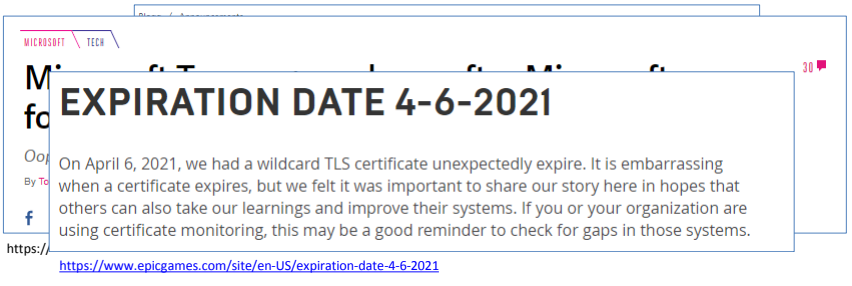
For how long should a certificate be valid?

Certificate lifetime

Most certificates that you buy have a **1-3** year expire time

Is this good?

Let's encrypt uses only **90** days! To short?



https://www.epicgames.com/site/en-US/expiration-date-4-6-2021

For how long should a certificate be valid? 

Certificate lifetime

Having short life-time, means:

- We do it often, so we don’t forget how to do it
- We will automate it
- The damage from a compromised certificate is reduced

When things are hard or scary,
we should do them more often!

What is the maximum lifetime? 

What is the maximum lifetime?

Certs valid for more than **398 days** will not be trusted!

Reducing TLS Certificate Lifespans to 398 Days

Ben Wilson | July 9, 2020

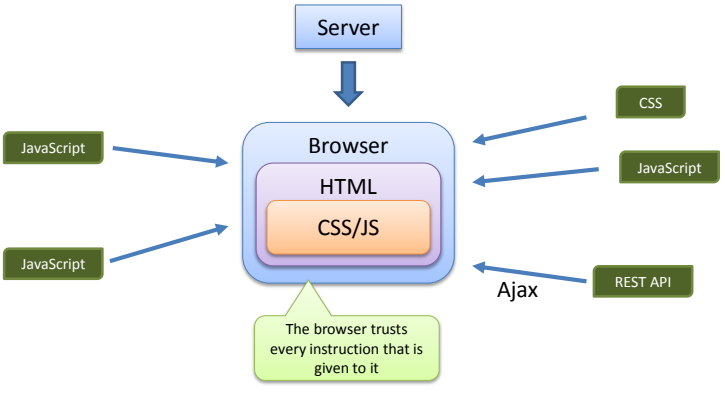
We intend to update Mozilla's Root Store Policy to reduce the maximum lifetime of TLS certificates from 825 days to 398 days, with the aim of protecting our user's HTTPS connections. Many reasons for reducing the lifetime of certificates have been provided and summarized in the CA/Browser Forum's Ballot SC22. Here are Mozilla's top three reasons for supporting this change.

https://blog.mozilla.org/security/2020/07/09/reducing-tls-certificate-lifespans-to-398-days

Content Security Policy

CSP - Content Security Policy

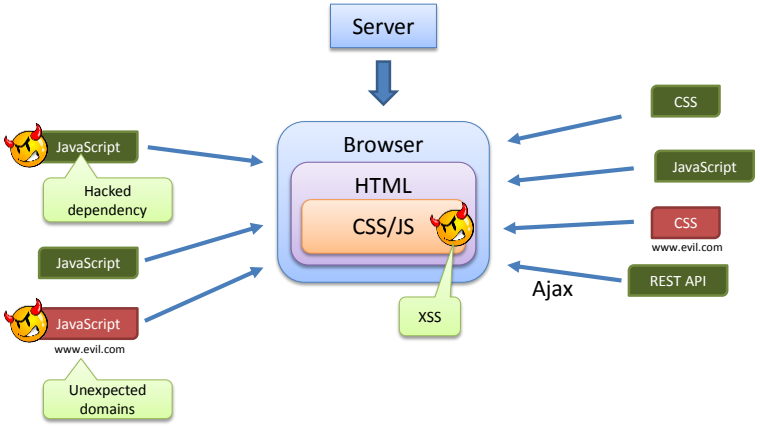
A typical web page:



What can go wrong here?

CSP - Content Security Policy

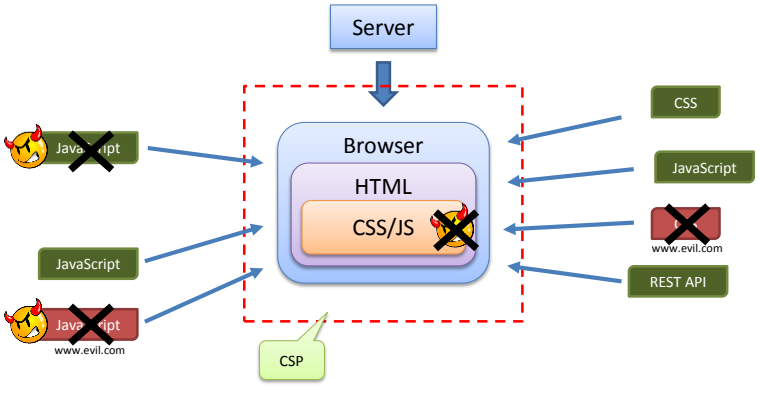
A lot of things can go wrong:



How can we secure this?

CSP - Content Security Policy

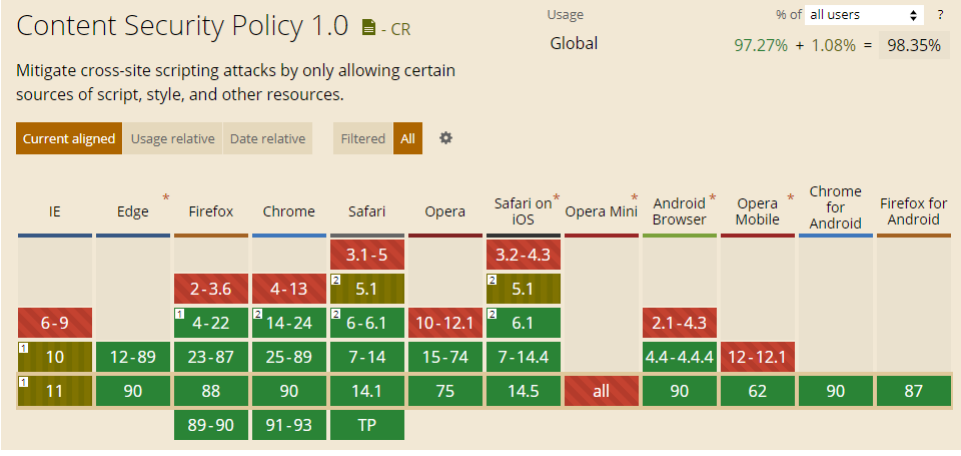
CSP provide a set of **security rules** to the browser



Request to other resources will be **blocked**

CSP - Content Security Policy

CSP 1.0 is today supported by most modern browsers:



http://caniuse.com/#feat=contentsecuritypolicy

CSP headers & directives

CSP - Header

Several HTTP non-standard headers exist to control CSP

- X-Content-Security-Policy
- X-WebKit-CSP

They have all been replaced with the standardized **Content-Security-Policy** header.

For example, **GitHub.com** uses this header:

```
Content-Security-Policy: default-src 'none'; base-uri 'self'; block-all-mixed-content; child-src
render.githubusercontent.com; connect-src 'self' uploads.github.com status.github.com
collector.githubapp.com api.github.com www.google-analytics.com github-cloud.s3.amazonaws.com github-
production-repository-file-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-
7fdce7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.com wss://live.github.com;
font-src assets-cdn.github.com; form-action 'self' github.com gist.github.com; frame-ancestors 'none';
img-src 'self' data: assets-cdn.github.com identicons.github.com collector.githubapp.com github-
cloud.s3.amazonaws.com *.githubusercontent.com; media-src 'none'; script-src assets-cdn.github.com;
style-src 'unsafe-inline' assets-cdn.github.com
```

CSP – GitHub header

The header consists of a list of **directives** separated by **;** as seen below from GitHub.com

```
default-src           'none';
base-uri              'self';

block-all-mixed-content;

child-src             render.githubusercontent.com; connect-src 'self' uploads.github.com
                      status.github.com collector.githubapp.com api.github.com www.google-analytics.com
                      github-cloud.s3.amazonaws.com
                      github-production-repository-file-5c1aeb.s3.amazonaws.com
                      github-production-upload-manifest-file-7fdce7.s3.amazonaws.com
                      github-production-user-asset-6210df.s3.amazonaws.com wss://live.github.com;

font-src              assets-cdn.github.com;

form-action           'self' github.com gist.github.com;
frame-ancestors       'none';
img-src               'self' data: assets-cdn.github.com identicons.github.com collector.githubapp.com
                      github-cloud.s3.amazonaws.com *.githubusercontent.com;
media-src             'none';
script-src            assets-cdn.github.com;
style-src             'unsafe-inline' assets-cdn.github.com
```

Directive name — Directive value

Exercise

Let’s do main exercise #3.x

Discussing the exercises with your fellow students is encouraged!

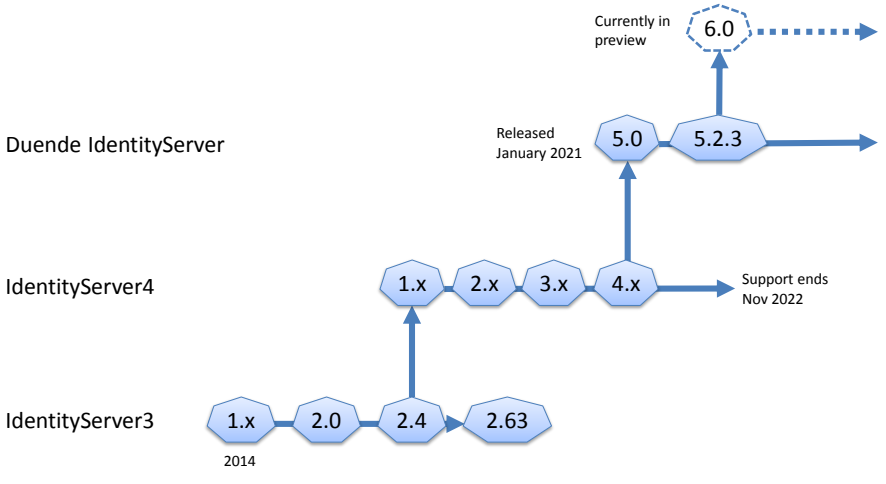
Please write in the chat when you are done

Adding IdentityServer

Module #4

Identity Server NuGet packages

We have several generations of IdentityServer:

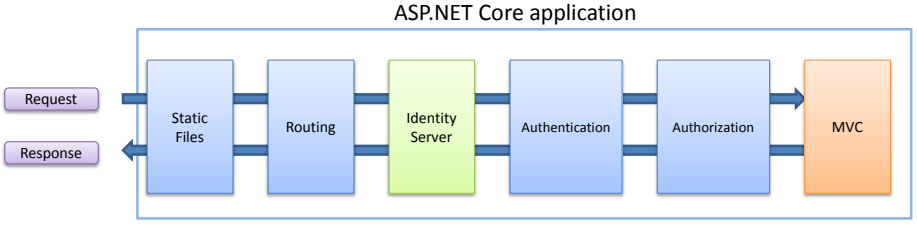


We will use **Duende IdentityServer v5** in this release

Getting concrete

Identity Server

The core of Identity Server is implemented as a **middleware**



In our Startup class

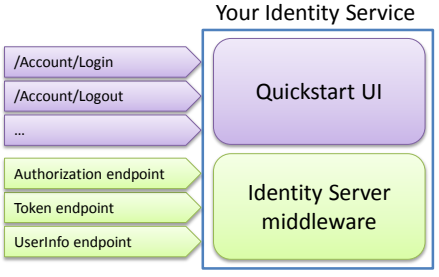
```
public void ConfigureServices(…)
{
    services.AddIdentityServer(options =>
        {
            /* options /*
        }));
}
```

Includes a call to UseAuthentication()

```
public void Configure(…)
{
    app.UseIdentityServer();

    app.UseAuthorization();
}
```

Identity Server

Your **Identity Server** service typically consists of these parts:

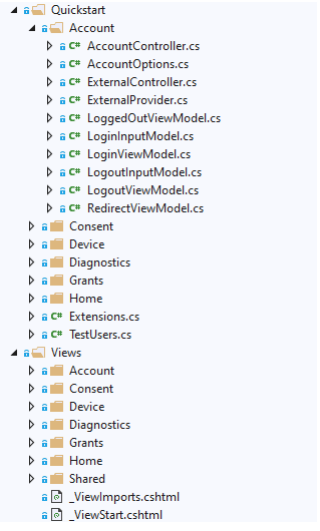


To reduce the attack surface, it should not include any other functionality -> **Single Responsibility**

Quickstart UI

Quickstart UI

Contains sample pages for login, logout, grant and consent



https://github.com/DuendeSoftware/IdentityServer.Quickstart.UI

Exercise

Let’s do main exercise #4.x

Discussing the exercises with your fellow students is encouraged!

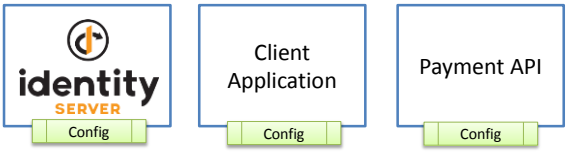
Please write in the chat when you are done

Logging

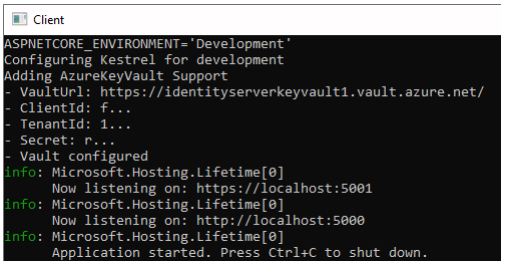
Module #5

Logging

We need to create a solid plan for our logging requirements

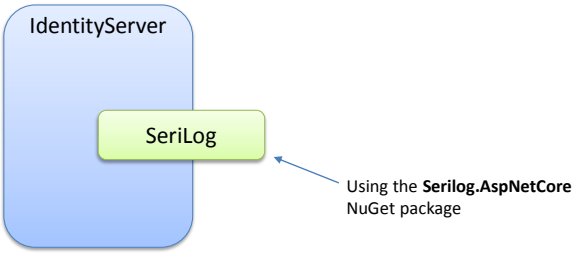


Just log to the console is not good enough



Logging IdentityServer

IdentityServer depends on **SeriLog** for logging



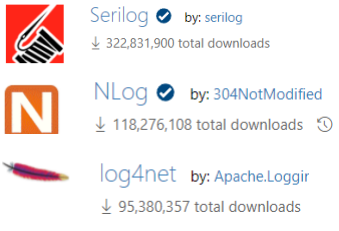
Why SeriLog?

Serilog
https://serilog.net/

Why SeriLog?

EDUMENT

- Most popular
- Open-source
- Flexible configuration
- Over 500 addons, enrichers and sinks
- Designed around **structured logging**



What is **structured logging**?

https://serilog.net/

Structured logging

Structured logging

Traditional logging usually looks something like this:

```
[09:49:24 INF] User Logged in 5434 Joe False 83.218.206.195 Sweden 23 2020-04-02 07:49:24
[09:52:02 INF] User Logged in 25312 Andy False 81.201.23.2 Sweden 66 2020-04-02 07:52:02
[09:55:37 INF] User Logged in 15319 Johan True 194.229.17.41 Netherlands 12 2020-04-02 07:55:37
[10:09:02 ERR] Application Crashed, Null Exception in module program.cs, line 236
[11:53:24 ERR] Timeout error when talking to payment API, Customer=23, OrderID=23212
...
```

What is the problem with this log? 

- How do we extract the data? Regex?
- Hard to parse
- Time zone?
- What does the various columns represent?

Structured logging

Compare it with structured logging


```
{"Timestamp":"2020-04-29T09:58:34.0392782+02:00","Level":"Information","MessageTemplate":"User
Logged in {UserID} {UserName} {IsAdmin} {IP} {Country} {NumberOfLogins}
{time}","Properties":{"UserID":5434,"UserName":"Joe","IsAdmin":false,"IP":"83.218.206.195","Countr
y":"Sweden","NumberOfLogins":23,"time":"2020-04-29T07:58:34.0258384Z"}
```


If we format the data:


```
{
   "Timestamp":"2020-04-29T09:58:34.0392782+02:00",
   "Level":"Information",
     "Properties":{
       "UserID":5434,
       "UserName":"Joe",
       "IsAdmin":false,
       "IP":"83.218.206.195",
       "Country":"Sweden",
       "NumberOfLogins":23,
       "time":"2020-04-29T07:58:34.0258384Z",
   }
}
```


Much easier to parse, process and query

Structured logging

The parsing of the logs can be improved even further by including the **message template**


```
{
   "Timestamp":"2020-04-29T09:58:34.0392782+02:00",
   "Level":"Information",
   "MessageTemplate":"User Logged in {UserID} {UserName} {IsAdmin} {IP} {Country} {NumberOfLogins}
{time}",
   "Properties":{
      "UserID":5434,
      "UserName":"Joe",
      "IsAdmin":false,
      "IP":"83.218.206.195",
      "Country":"Sweden",
      "NumberOfLogins":23,
      "time":"2020-04-29T07:58:34.0258384Z",
   }
}
```


So what is the message template? 

Message template

Message template

A message template is a human friendly format specifier with named holes for event data:

```
"User Logged in {UserID} {UserName} {IsAdmin} {IP} {Country} {NumberOfLogins} {time}"
```

```
Log.Information("User Logged in {UserID} {UserName} {IsAdmin} {IP} {Country} {NumberOfLogins} {time}",
                UserID,UserName,IsAdmin,IP,Country,NumberOfLogins,time);
```

Message template

If we have the data and the template, then we can recreate the original human readable text!

```
"User Logged in {UserID} {UserName} {IsAdmin} {IP} {Country} {NumberOfLogins} {time}"
```

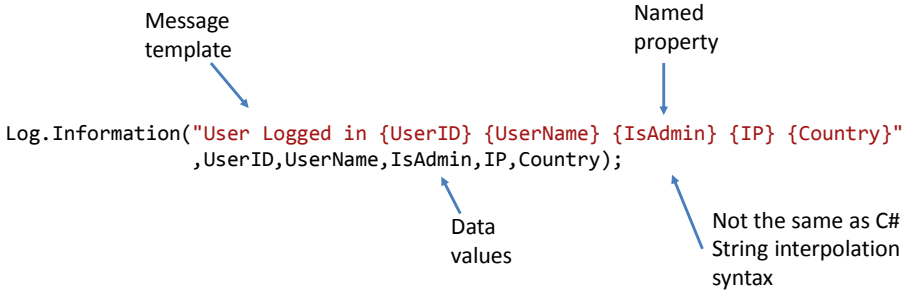
```
{
   "Timestamp":"2020-04-29T09:58:34.0392782+02:00",
   "Level":"Information",
   "MessageTemplate":"User Logged in {UserID} {UserName}
      {IsAdmin} {IP} {Country} {NumberOfLogins} {time}",
   "Properties":{
      "UserID":5434,
      "UserName":"Joe",
      "IsAdmin":false,
      "IP":"83.218.206.195",
      "Country":"Sweden",
      "NumberOfLogins":23,
      "time":"2020-04-29T07:58:34.0258384Z",
   }
}
```

```
[09:49:24 INF] User Logged in 5434 Joe False 83.218.206.195 Sweden 23 2020-04-02 07:49:24
```

Logging data

Message template

When we write to the log, we provide a message template:

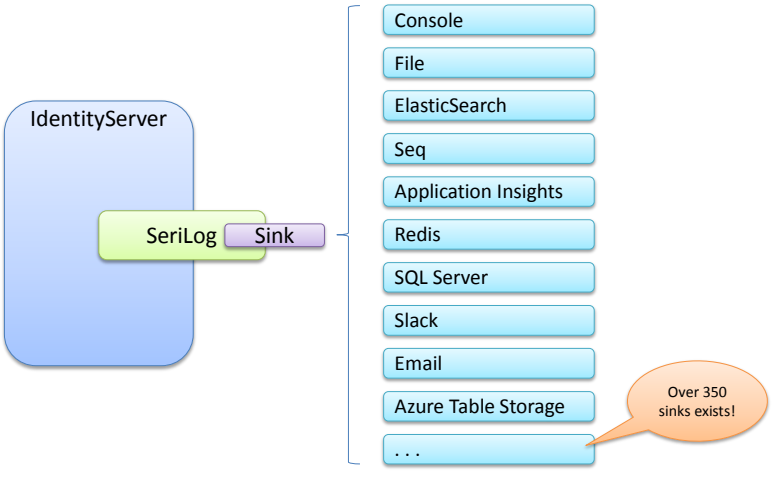


A standard have emerged around this template:
https://messagetemplates.org

Sinks

Serilog sinks

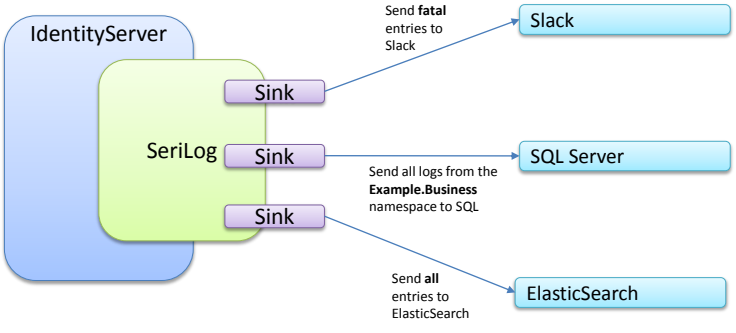
We use **sinks** as the destination for the log entries:



We can add one or multiple sinks to IdentityServer

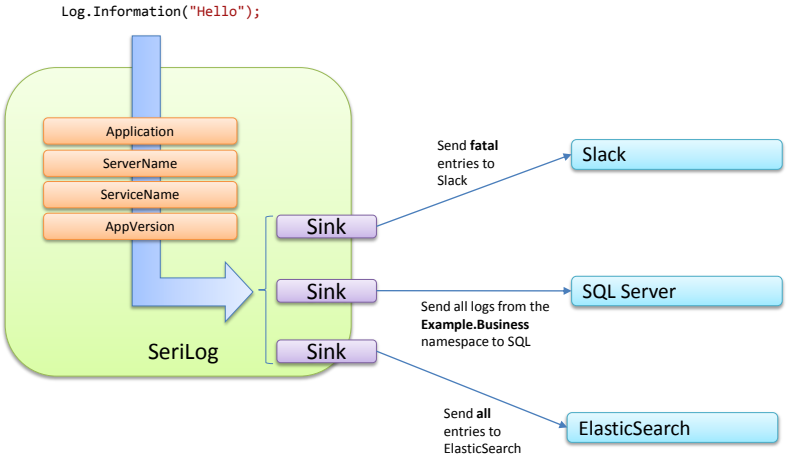
Multiple sinks

We can send different entries to different sinks



Enrichers

Enrichers can further add properties to our log data



Configuring logging

Configuring logging

We typically configure logging in three steps:

```
public class Program
{
    public static int Main(string[] args)
    {
        Log.Logger = new LoggerConfiguration()
            .MinimumLevel.Debug()
            .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
            .MinimumLevel.Override("Microsoft.Hosting.Lifetime", LogEventLevel.Information)
            .MinimumLevel.Override("System", LogEventLevel.Warning)
            .MinimumLevel.Override("Microsoft.AspNetCore.Authentication", LogEventLevel.Information)
            .Enrich.FromLogContext()
            .WriteTo.Console(outputTemplate: "[{Timestamp:HH:mm:ss} {Level}] {SourceContext}{NewLine}{Message:lj}{NewLine}{Exception}{NewLine}",
                             theme: AnsiConsoleTheme.Code)
            .CreateLogger();

        try
        {
            Log.Information("Starting host...");
            CreateHostBuilder(args).Build().Run();
            return 0;
        }
        catch (Exception ex)
        {
            Log.Fatal(ex, "Host terminated unexpectedly.");
            return 1;
        }
        finally
        {
            Log.CloseAndFlush();
        }
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .UseSerilog()
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

1
2
3

Let’s break down this code

Configuring logging

First we configure and create an instance of **ILogger**

```
Log.Logger = new LoggerConfiguration()

    .MinimumLevel.Debug()

    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .MinimumLevel.Override("Microsoft.Hosting.Lifetime", LogEventLevel.Information)
    .MinimumLevel.Override("System", LogEventLevel.Warning)
    .MinimumLevel.Override("Microsoft.AspNetCore.Authentication", LogEventLevel.Information)

    .Enrich.FromLogContext()

    .WriteTo.Console(outputTemplate: "[{Timestamp:HH:mm:ss} {Level}]
{SourceContext}{NewLine}{Message:lj}{NewLine}{Exception}{NewLine}",
                     theme: AnsiConsoleTheme.Code)

    .CreateLogger();
```

Configures the minimum level at which events will be passed to sinks.

Override the minimum level for events from a specific namespace or type name.

Add additional properties from the logging context to the logs

Where to send the logs to

Create the resulting instance of ILogger

Configuring logging

Then we create our host and run it

```
try
{
    Log.Information("Starting host...");
    CreateHostBuilder(args).Build().Run();
    return 0;
}
catch (Exception ex)
{
    Log.Fatal(ex, "Host terminated unexpectedly.");
    return 1;
}
finally
{
    Log.CloseAndFlush();
}
```

Make sure all buffered events are stored
And dispose the logger

Configuring logging

Finally, we add SeriLog to ASP.NET Core system:

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .UseSerilog()
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
```

Sets **Serilog** as the logging provider

How should we use the log levels? 

```
public enum LogLevel
{
  Trace = 0,
  Debug = 1,
  Information = 2,
  Warning = 3,
  Error = 4,
  Critical = 5,
  None = 6,
}
```

```
public enum LogEventLevel
{
    Verbose,
    Debug,
    Information,
    Warning,
    Error,
    Fatal
}
```

Serilog uses a slightly different names

Log levels

IdentityServer use these log level definitions

Level	Description
Trace / Verbose	For information that is valuable only to a developer troubleshooting an issue. Should not be enabled in a production environment.
Debug	For following the internal flow and understanding why certain decisions are made. Has short-term usefulness during development / debugging.
Information	For tracking the general flow of the application. These logs typically have some long-term value.
Warning	For abnormal or unexpected events in the application flow. These may include errors or other conditions that do not cause the application to stop, but which may need to be investigated.
Error	For errors and exceptions that cannot be handled. Examples: failed validation of a protocol request.
Fatal / Critical	For failures that require immediate attention. Examples: missing store implementation, invalid key material…

https://docs.duendesoftware.com/identityserver/v5/diagnostics/logging/

Seq

Seq

We will send all of our log entries to Seq

https://datalust.co/

Exercise

Let’s do main exercise #5.x

Discussing the exercises with your fellow students is encouraged!

Please write in the chat when you are done

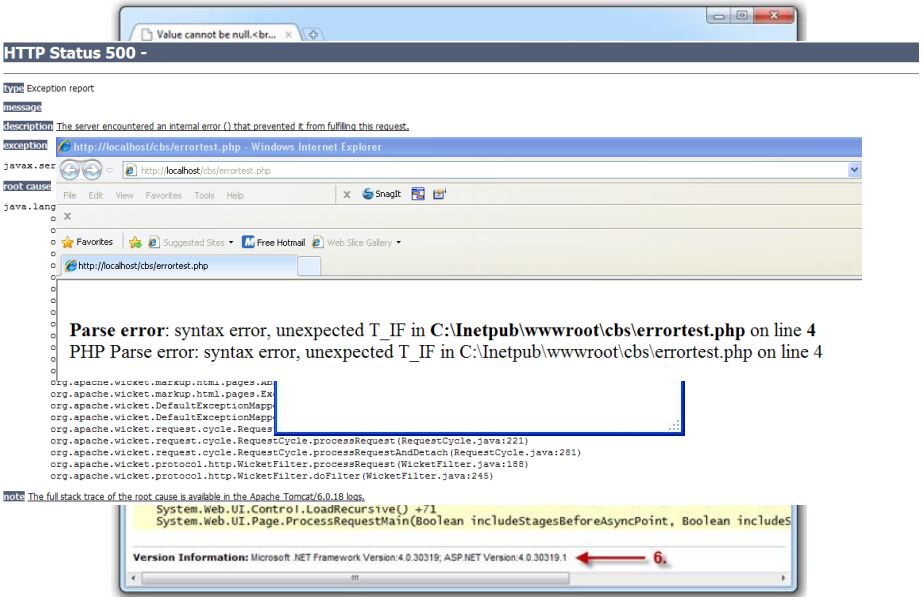
Error handling

Module #6

Why is proper error handling important? 

Error handling

How do we make sure we never show this?



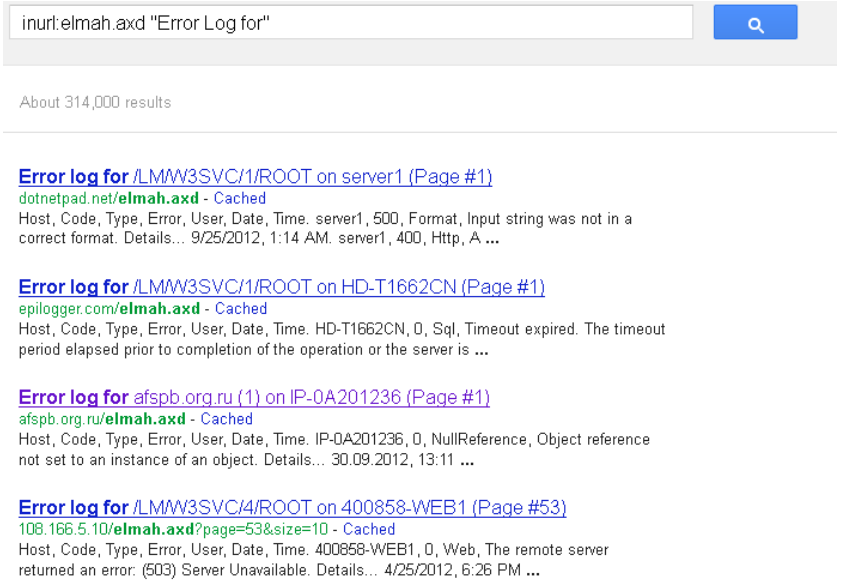
Securing ASP.NET

What happens if we search on google for

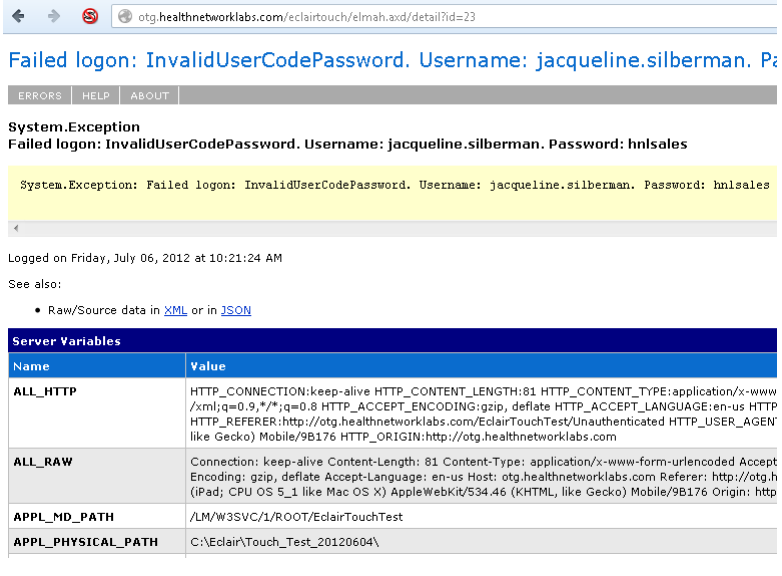
```
inurl:elmah.axd "Error Log for"
```

ELMAH (Error Logging Modules and Handlers) is an application-wide error logging framework. It can be dynamically added to a running ASP.NET web application without any need for re-compilation or re-deployment.

Securing ASP.NET



Securing ASP.NET



Securing ASP.NET

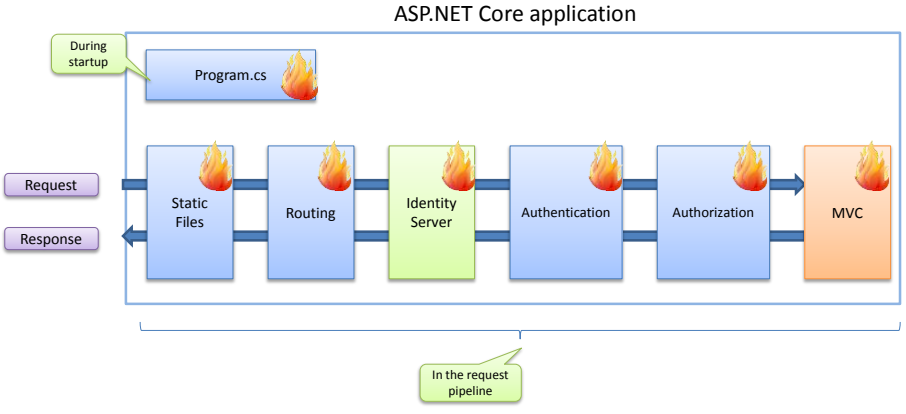
You can even find the session and if the user is authenticated!

GATEWAY_INTERFACE	CGI/1.1
HTTP_ACCEPT	text/css,*/*;q=0.1
HTTP_ACCEPT_CHARSET	windows-1251,utf-8;q=0.7,*;q=0.3
HTTP_ACCEPT_ENCODING	gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE	ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4
HTTP_CACHE_CONTROL	max-age=0
HTTP_CONNECTION	keep-alive
HTTP_COOKIE	ASP.NET_SessionId=bkcplxrrtfga0x41ztcuvei0; .ASPXAUTH=3EA6EDC3B6E65B3B4969F9153BDD0844A7A99B8CC8D9B148F4CDE7D5B8B6C89FD18622F34169FAD358BC6C017C907A1 __utma=168781539.293915852.1361646721.1361646721.1361646721.1; __utmb=168781539.8.10.1361646721; __utmc=168781539; _
HTTP_HOST	31.28.161.247
HTTP_IF_MODIFIED_SINCE	Sun, 17 Feb 2013 09:12:01 GMT
HTTP_IF_NONE_MATCH	"-901174619"
HTTP_REFERER	http://31.28.161.247/
HTTP_USER_AGENT	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.22 (KHTML, like Gecko) Chrome/25.0.1364.97 Safari/537.22
HTTPS	off
HTTPS_KEYSIZE	
HTTPS_SECRETKEYSIZE	
HTTPS_SERVER_ISSUER	
HTTPS_SERVER_SUBJECT	
INSTANCE_ID	2

Exception handling in ASP.NET Core

Exception handling

We need to handle exceptions in two places



Error handling in program.cs

In **Program.cs** we have this error handling:

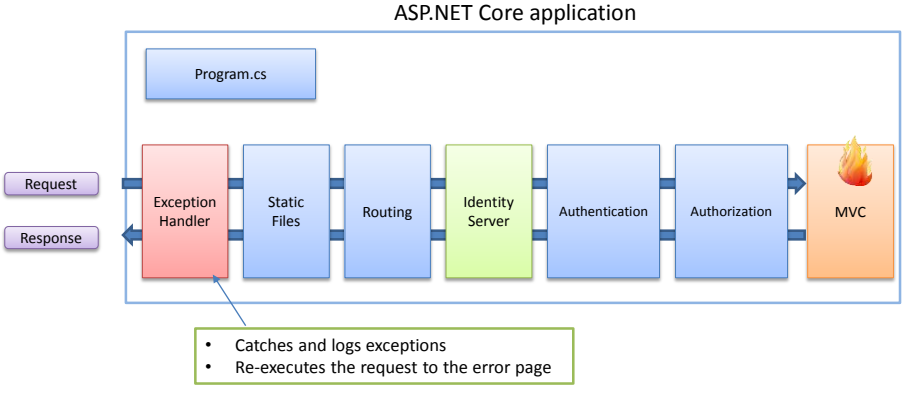
```
try
{
    Log.Information("Starting web host for " + _applicationName);
    CreateHostBuilder(args)
        .ConfigureSerilog(_applicationName, options =>
        {
            ConfigureLogLevels(options);
        })
        .UseSerilog()
        .Build()
        .Run();
    return 0;

}
catch (Exception ex)
{
    Console.WriteLine(ex);
    Log.Fatal(ex, "Host terminated unexpectedly");
    return 1;
}
finally
{
    Log.CloseAndFlush();
}
```

We writ to the console in case logging does not work

Returning a non-zero value means that an error occurred

Error handling in the pipeline

For exceptions during requests, we add an **Exception handler**




```
app.UseExceptionHandler("/Error");
```

Handing of application errors

Handing of application errors

By default MVC will return empty error responses

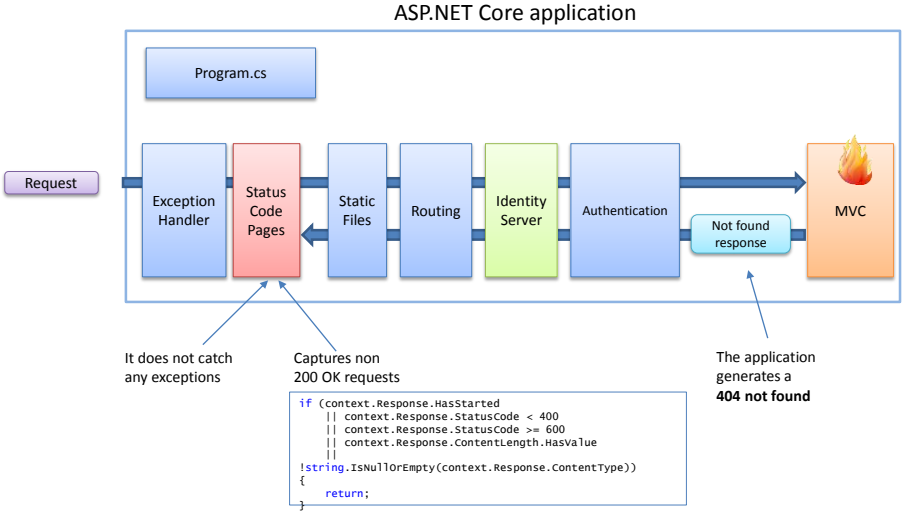
```
https://localhost/XXXXX
```

```
HTTP/1.1 404 Not Found
Date: Thu, 14 Jan 2021 14:51:59 GMT
Content-Length: 0
```

How can we improve this? 

UseStatusCodePages

We can add a status code **error handler** to the pipeline




```
if (context.Response.HasStarted
    || context.Response.StatusCode < 400
    || context.Response.StatusCode >= 600
    || context.Response.ContentLength.HasValue
    ||
!string.IsNullOrEmpty(context.Response.ContentType))
{
    return;
}
```

Adding Status code handler

Without any status handler, we get this response:

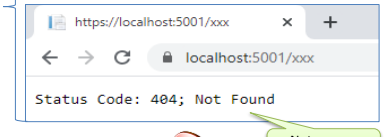
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // No handler (Default)
    ...
}
```

```
HTTP/1.1 404 Not Found
Date: Thu, 14 Jan 2021 14:51:59 GMT
Content-Length: 0
```

Adding **UseStatusCodePages**:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // With default options
    app.UseStatusCodePages();
    ...
}
```

```
HTTP/1.1 404 Not Found
Date: Thu, 14 Jan 2021 15:06:07 GMT
Content-Length: 527

Status Code: 404; Not Found
```



Not so user friendly

How can we improve this further?

A better option is to use one of these ones:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
  ...
  app.UseStatusCodePagesWithRedirects(locationFormat: "/error?error={0}");

  app.UseStatusCodePagesWithReExecute(pathFormat: "/error", queryFormat: "?error={0}");
  ...
}
```

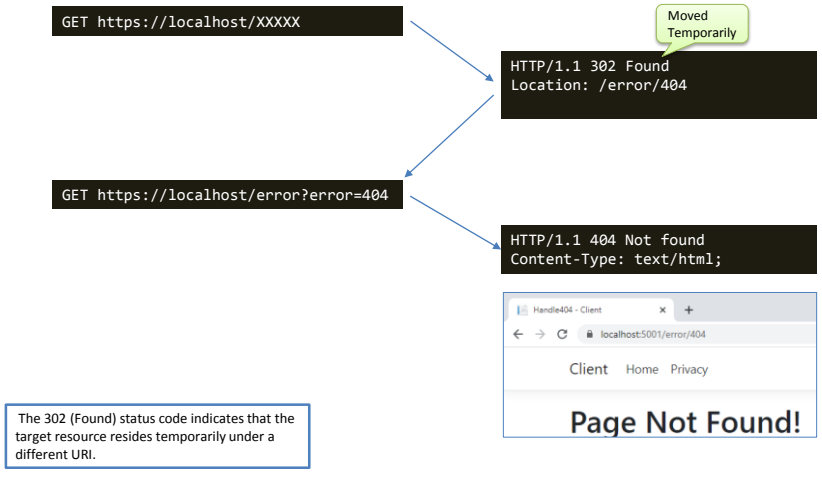
```
/error?error=400
/error?error=404
/error?error=500
/error?error=503
```

What is the difference?

With Redirects

With redirects:

```
app.UseStatusCodePagesWithRedirects(locationFormat: "/error?error={0}");
```

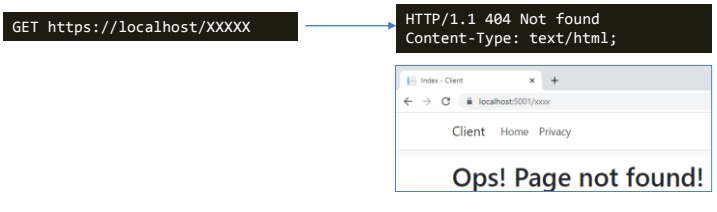


The 302 (Found) status code indicates that the target resource resides temporarily under a different URI.

With re-execute

This alternative will **re-execute** the request

```
app.UseStatusCodePagesWithReExecute(pathFormat: "/error",queryFormat: "?error={0}");
```

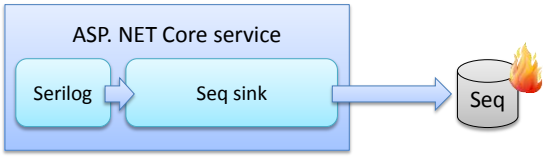


Demonstration

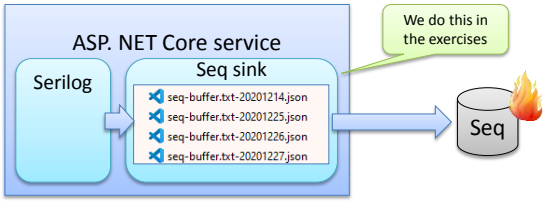
Hardening the logging to Seq

Hardening the logging to Seq

What will happen if **Seq** is not reachable?



We can configure to use **durable log shipping**



Exercise

Let’s do main exercise #6.x

Discussing the exercises with your fellow students is encouraged!

Please write in the chat when you are done

Securing the client with OIDC

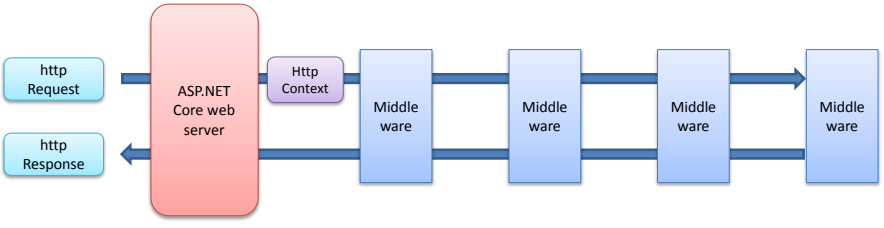
Module #7

Authentication fundamentals

HttpContext

For each incoming request, the ASP.NET Core will construct a **HttpContext** object

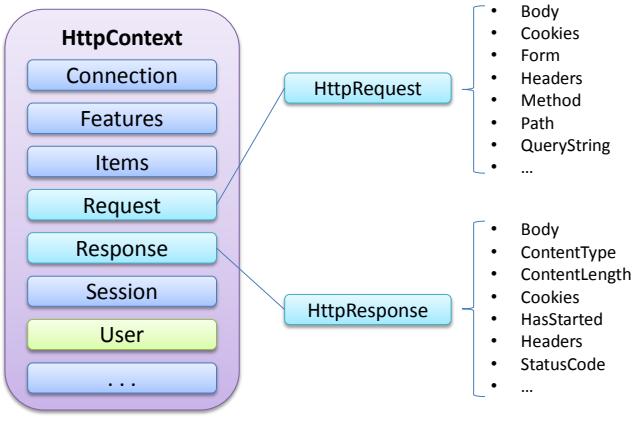
The **HttpContext** is then passed through the pipeline



It's up to each middleware to **inspect** and **modify** this object to build up a proper response

HttpContext

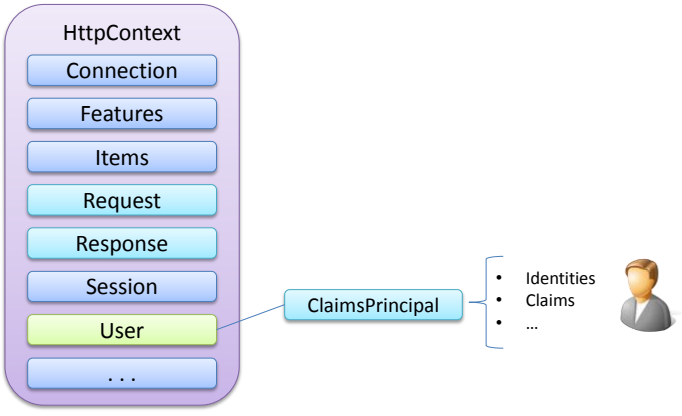
It contains everything about a specific request:



The HttpContext also contains a **User**

HttpContext.User

The User is represented as a **ClaimsPrincipal**

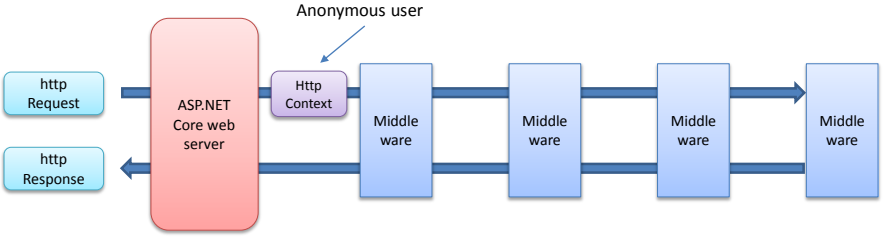


Authenticating the user

Authenticating the user

EDUMENT

Each incoming request is initially set to an **anonymous unauthenticated** user without any claims

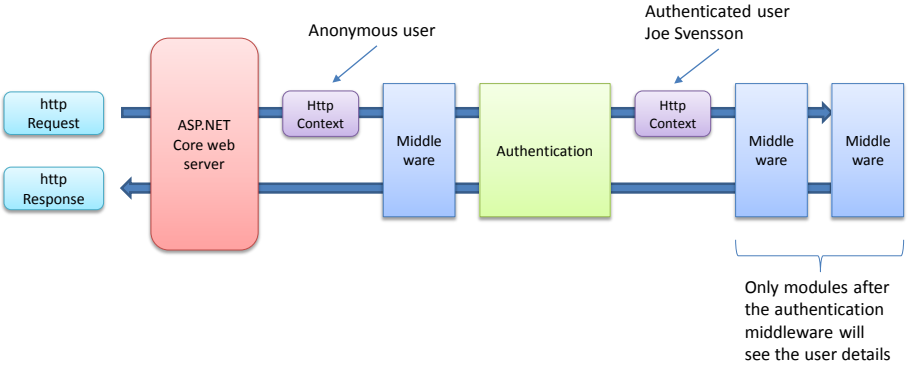


Who sets the user?

Setting the ClaimsPrincipal

EDUMENT

The **Authentication middleware** is in charge of creating and managing the **ClaimsPrincipal** User.

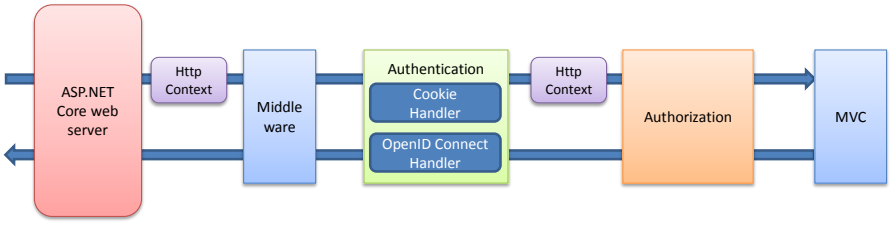


How do we add support for OpenID Connect?

ASP.NET Core and OpenID Connect

EDUMENT

We add the **Cookie** and the **OpenIDConnect** handler

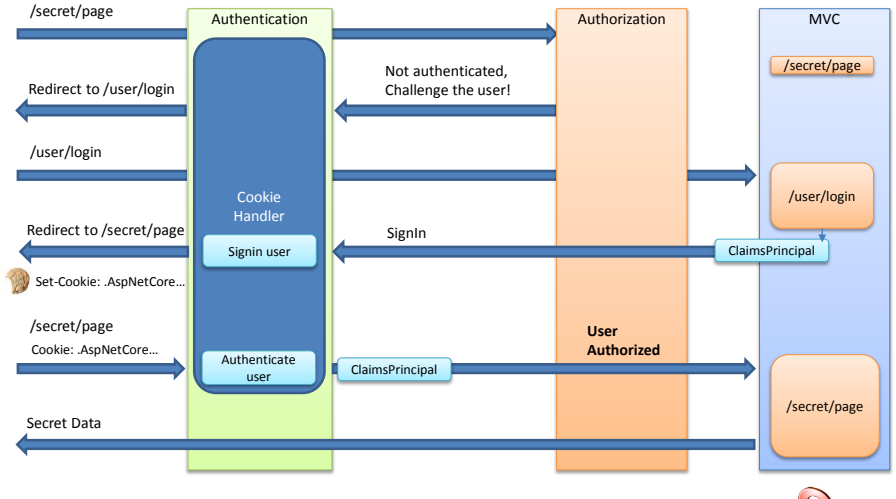


Why do we need two handlers?

Why do we need two handlers?

EDUMENT

When we use the **Cookie handler**, the flow is:

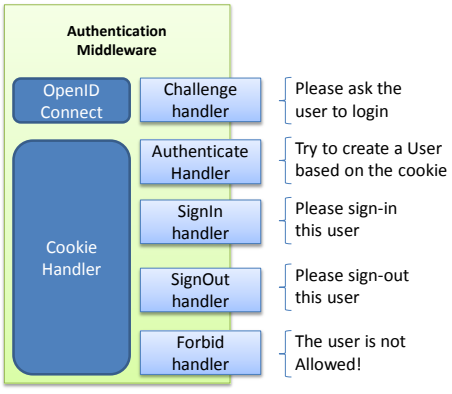


What do we want to achieve by adding OIDC?

Why do we need two handlers?

EDUMENT

The **Authentication** middleware have these core functions:



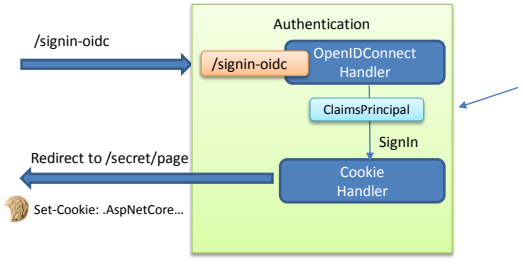
With OpenIDConnect we want the OpenIDConnect handler to only handle the **challenge** part

Why do we need two handlers?

EDUMENT

When challenged, it redirect the user to the STS

After the user login and give consent:



After it have a valid token, the handler asks the Cookie handler to SignIn and create the cookie

Let's add some code!

Securing the client

We need to add the following to add OIDC support:

Register and configure the handers

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddAuthentication(options =>
    {
    }).AddCookie(options =>
    {
    }).AddOpenIdConnect(options =>
    {
    });
}
```

Add the middlewares

```
public void Configure(IApplicationBuilder app)
{
    ...
    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        ...
    });
}
```

Let’s review this configuration

AddAuthentication

AddAuthentication

Here we define **who** should handle the various interactions:

```
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = "Cookies";
    options.DefaultSignInScheme = "Cookies";
    options.DefaultSignOutScheme = "Cookies";
    options.DefaultChallengeScheme = "OpenIdConnect";
    options.DefaultForbidScheme = "Cookies";
})
```

Using this is convenient, but we might run into typos!

This is the same as:

```
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultSignOutScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
    options.DefaultForbidScheme = CookieAuthenticationDefaults.AuthenticationScheme;
})
```

This can further be reduced to:

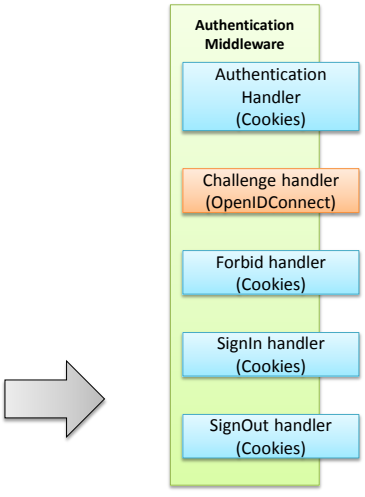
```
services.AddAuthentication(options =>
{
    options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
}
```

AddAuthentication

The corresponding handler is then called when we use these methods:

```
await HttpContext.AuthenticateAsync(...);
await HttpContext.SignInAsync(...);
await HttpContext.SignOutAsync()...;
await HttpContext.ChallengeAsync(...);
await HttpContext.ForbidAsync(...);
```

```
services.AddAuthentication(options =>
{
    options.DefaultScheme = "Cookies";
    options.DefaultChallengeScheme = "OpenIdConnect";
}
).AddCookie("Cookies", options =>
{
}
).AddOpenIdConnect("OpenIdConnect", options =>
{
}
);
```



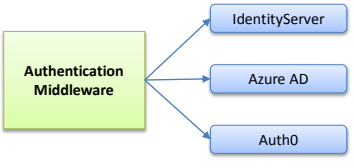
AddAuthentication

A more advanced example:

```
services.AddAuthentication(options =>
{
    options.DefaultScheme = "Cookies";
    options.DefaultChallengeScheme = "IdentityServer";
}
 ).AddCookie("Cookies", options =>
{
}).AddOpenIdConnect("IdentityServer", options =>
{
}).AddOpenIdConnect("AzureActiveDirectory", options =>
{
}).AddOpenIdConnect("Auth0", options =>
{
});
```

In this example the user or application can choose how to authenticate the user

```
// Challenge using IdentityServer
HttpContext.ChallengeAsync("IdentityServer");

// Challenge using AzureActiveDirectory
HttpContext.ChallengeAsync("AzureActiveDirectory");

// Challenge using Auth0
HttpContext.ChallengeAsync("Auth0");
```



AddOpenIdConnect

AddOpenIdConnect

To add **OpenIDConnect** support, the minimum is:

```
.AddOpenIdConnect(options =>
{
    options.Authority = "https://localhost:6001";
    options.ClientId = "myClientId";
    options.ClientSecret = "MyClientSecret";
    options.ResponseType = "code";

    options.Scope.Clear();
    options.Scope.Add("openid");
    options.Scope.Add("profile");
    options.Scope.Add("email");
    options.Scope.Add("offline_access");

    options.GetClaimsFromUserInfoEndpoint = true;
    options.SaveTokens = true;

    options.Prompt = "consent";
});
```

code represents the **authorization code flow**

If we don't set the scopes, then the **openid** and **profile** scopes will be requested by default

Save the token in the session cookie

Always show the consent screen.

The **prompt** parameter can be used by the Client to make sure that the End-User is still present for the current session or to bring attention to the request. (optional)

Will also set these defaults:

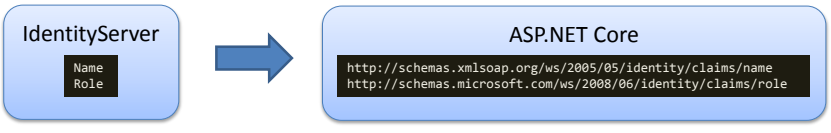
```
options.RequireHttpsMetadata = true;

options.CallbackPath = "/signin-oidc";
options.SignedOutCallbackPath = "/signout-callback-oidc";
options.RemoteSignOutPath = "/signout-oidc";
options.ResponseMode= OpenIdConnectResponseMode.FormPost;
```

AddOpenIdConnect

Typically, the **Name** and **Role** claim needs to be remapped as well:

```
.AddOpenIdConnect(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
       {
            NameClaimType = JwtClaimTypes.Name,
            RoleClaimType = JwtClaimTypes.Role
       };
});
```

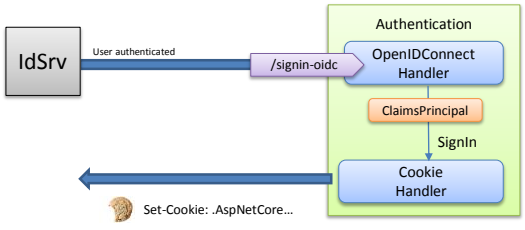
IdentityServer and ASP.NET Core don’t agree about the name of these claims.



AddCookie

AddCookie

After a successful authentication, the **OpenID Connect** handler will ask the **cookie handler** to sign-in the user



This is controlled by the **SignInScheme** property:

```
services.AddAuthentication(options =>
    { } )
.AddCookie("Cookies", options =>
    { } )
.AddOpenIdConnect("OpenIdConnect", options =>
    {
        //By default, set to the Default signin scheme
        options.SignInScheme = "Cookies";
    });
```

AddCookie

We use the **Cookie handler** for everything except sign-in

```
.AddCookie(options =>
  {
      options.LogoutPath = "/User/Logout";
      options.AccessDeniedPath = "/User/AccessDenied";
  })
```

This will also set these default values:

```
SameSite = SameSiteMode.Lax;
HttpOnly = true;
SecurePolicy = CookieSecurePolicy.SameAsRequest;
IsEssential = true;
ExpireTimeSpan = TimeSpan.FromDays(14);
SlidingExpiration = true;
```

What does the various paths do? 

LogoutPath

The **LogoutPath** controls if the redirect is allowed or not

```
.AddCookie(options => { opt.LogoutPath = "/User/Logout"; })
```

Security feature!

For example, in this case:

/User/Logout

```
public class UserController : Controller
{
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task Logout()
    {
        await HttpContext.SignOutAsync(new AuthenticationProperties()
          {
              RedirectUri = "/account/signedout"
          });
    }
}
```

We only allow the redirect if the page asking for the redirect matches the LogOutPath

If the **LogOutPath == current page**, then redirect will occur

AccessDeniedPath

When you are **denied access**, you will be redirected here:

```
.AddCookie(options => {options.AccessDeniedPath = "/User/AccessDenied"; })
```

/ForbiddenPage

```
await context.ForbidAsync();
```



```
HTTP/1.1 302 Found
Date: Tue, 06 Oct 2020 11:54:13 GMT
Server: Kestrel
Content-Length: 0
Location:
https://localhost:5001/User/AccessDenied?ReturnUrl=%2FForbidden
```

The ReturnURI can be customized

Exercise

Let’s do main exercise #7.x

Discussing the exercises with your fellow students is encouraged!

Please write in the chat when you are done

ASP.NET Core Data Protection

Module #8

The problem

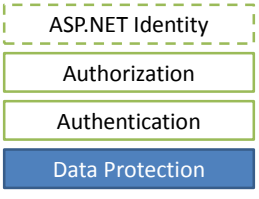
In our log files, we will eventually see errors like:

```
Microsoft.AspNetCore.Antiforgery.AntiforgeryValidationException:
The antiforgery token could not be decrypted.
System.Security.Cryptography.CryptographicException:
The key {dc3ff6b5-11a6-4d20-a472-c7f1ed71f74a} was not found in the key ring.
```

```
Microsoft.AspNetCore.Authentication.Cookies.CookieAuthenticationHandler
Cookies was not authenticated. Failure message: Unprotect ticket failed
```

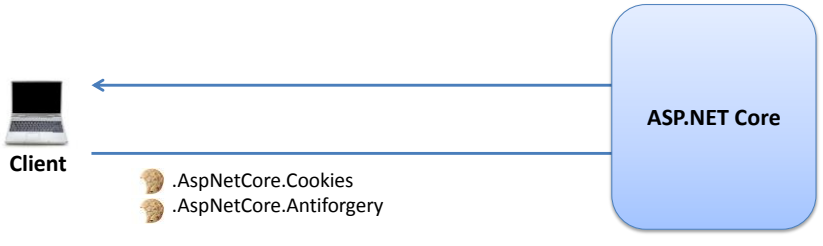
In this module we will learn how to fix these errors

Data Protection API



Data Protection API

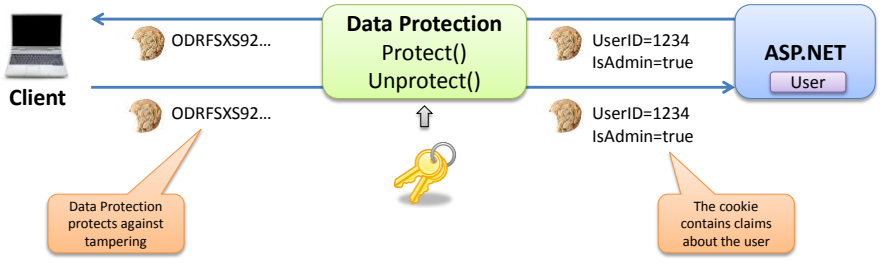
ASP.NET Core writes two types of sensitive cookies:



ASP.NET Core uses The **Data Protection API** to protect them

Data Protection API

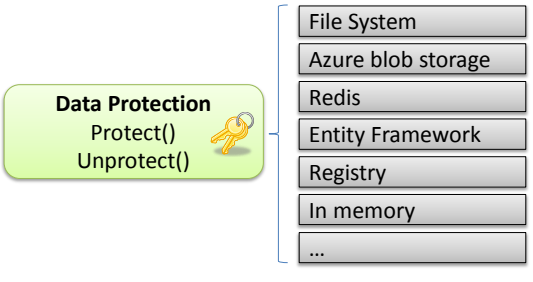
The **Data Protection API** provides an API to **encrypt / decrypt** data using an encryption key



Where should we store the secret crypto key?

Data Protection API

The key can be stored in the following places:

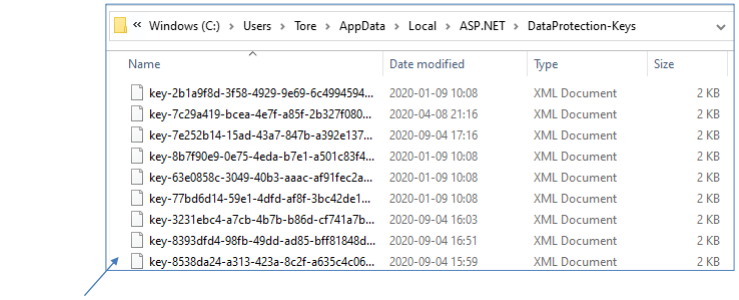


More providers exists on NuGet
https://www.nuget.org/packages?q=AspNetCore.DataProtection

Data Protection API

By default, all the keys are stored on disk:

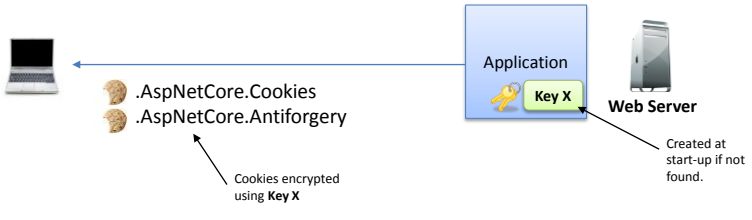
```
%LOCALAPPDATA%\ASP.NET\DataProtection-Keys
```



What can go wrong here?

Data Protection API

Our cookies are secured using an **encryption key**

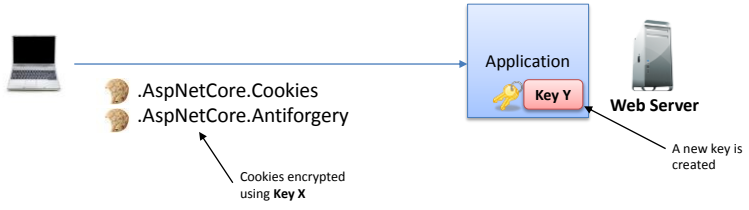


If the key is not found, then it will be created

Demonstration #1

Data Protection API

If we do a full **redeployment**, a new **key** is created

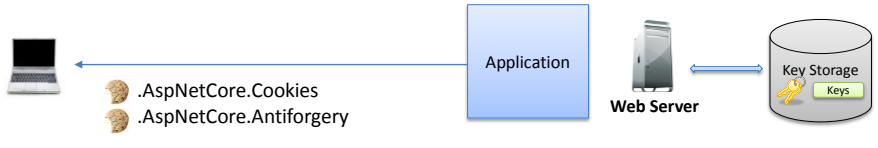


Now existing cookies can't be decrypted!

Demonstration #2

Data Protection API

To solve this, we should store the key in a persisted storage

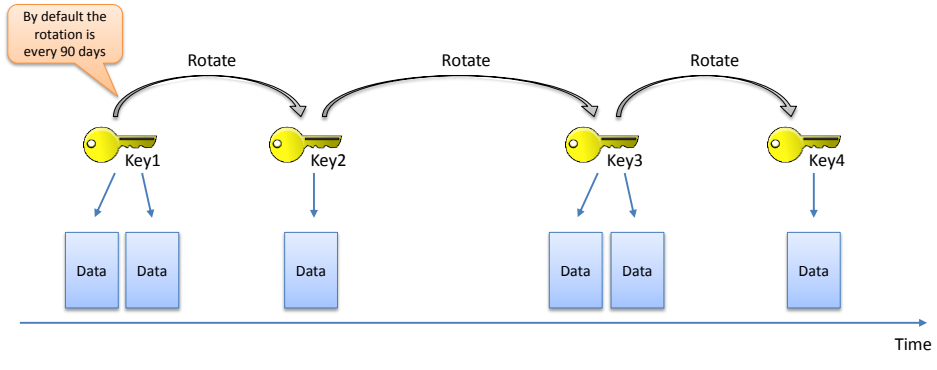


We want the key to be the same across deployments

Key rotation and the key ring

Key rotation

Periodically a new key is generated and the old key is retired



The latest key is always used to encrypt the data

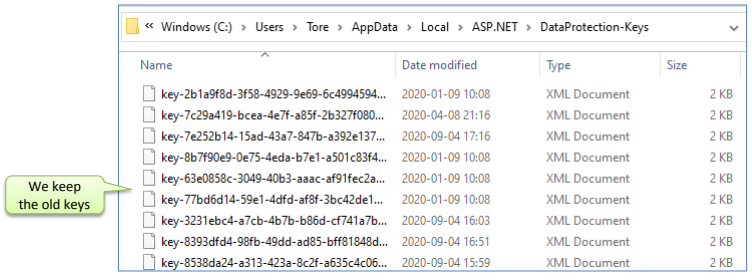
To decrypt data, the key used to encrypt it, must be used

Why Key Rotation
https://medium.com/@dieswaytoofast/why-key-rotation-f374c71b9c6f

Demonstration #3

Key rotation

We see the effect of key rotation here:



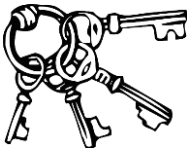
Only the latest and valid key-ring is used

Old keys can be used to decrypt existing data

Key ring

All keys are collected and stored in a **key ring**

The main interface for working with the key ring is defined as:



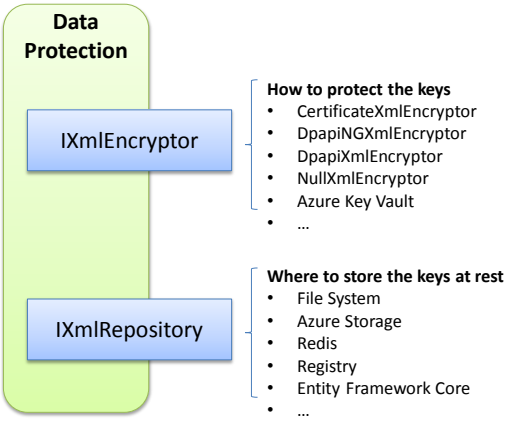
```
public interface IKeyManager
{
    IKey CreateNewKey(DateTimeOffset activationDate, DateTimeOffset expirationDate);
    IReadOnlyCollection<IKey> GetAllKeys();
    CancellationToken GetCacheExpirationToken();
    void RevokeAllKeys(DateTimeOffset revocationDate, string reason = null);
    void RevokeKey(Guid keyId, string reason = null);
}
```

We usually never delete keys!
Instead we revoke or expire them

Configuring Data Protection

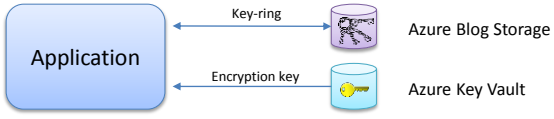
Data Protection API

To configure it, it needs two things:



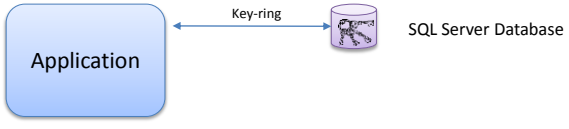
Sample configurations

```
services.AddDataProtection()
    .PersistKeysToAzureBlobStorage(new Uri("<blogstorageUri>"))
    .ProtectKeysWithAzureKeyVault("<keyIdentifier>",
                                  "<clientId>",
                                  "<clientSecret>");
```




```
// Add a DbContext to store your Database Keys
services.AddDbContext<MyKeysContext>(options =>
    options.UseSqlServer(
        _config.GetConnectionString("MyConnectionString")));

services.AddDataProtection()
    .PersistKeysToDbContext<MyKeysContext>();
```

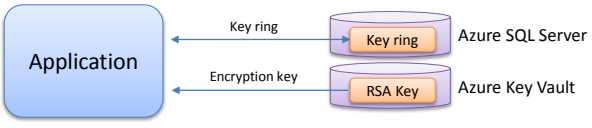
Keys will not be encrypted in the DB



Storing the key ring in a database

Storing the key ring in a database

In this course we will store the key ring in a **database**



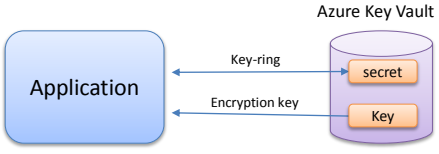
The keys will be protect by a key in **Azure Key Vault**

We will use this approach in the exercises

Storing the key ring in Azure Key Vault

Storing the key ring in Azure Key Vault

Why don't we store both items in Azure Key Vault?



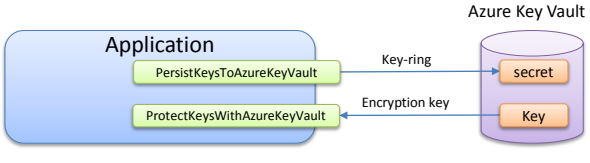
And configuring it using:

```
services.AddDataProtection()
    .PersistKeysToAzureKeyVault(...[Config]...)
    .ProtectKeysWithAzureKeyVault(...[Config]...);
```

But we have one problem…

Storing the key ring in Azure Key Vault

PersistKeysToAzureKeyVault does not exist, so we wrote one



We blogged about it here:

Storing the ASP.NET Core Data Protection Key Ring in Azure Key Vault
https://www.edument.se/en/blog/post/storing-the-asp-net-core-data-protection-key-ring-in-azure-key-vault

However, we will use the database in this course

Exercise

Let’s do module and main exercise #8.x

Discussing the exercises with your fellow students is encouraged!

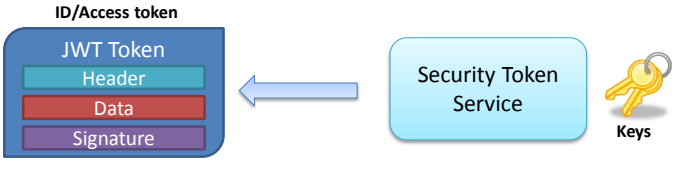
Please write in the chat when you are done

Private/Public-key cryptography

Module #9

Token signing keys

All issued **JWT-tokens** needs to be **digitally signed**:

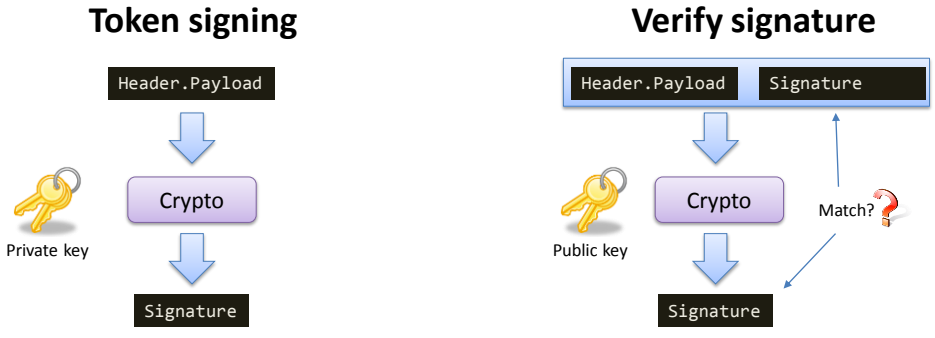


The signature is added to the end of the tokens

How do we sign our tokens?

Signing using private/public keys

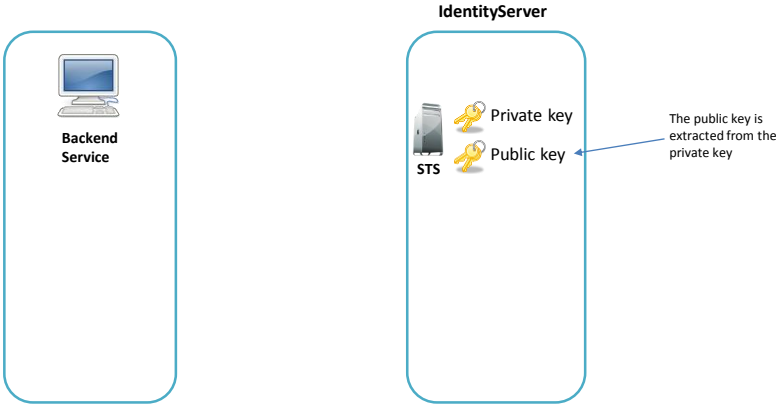
With IdentityServer we use **asymmetric cryptography**



How does this work in practice?

Signing using private/public keys

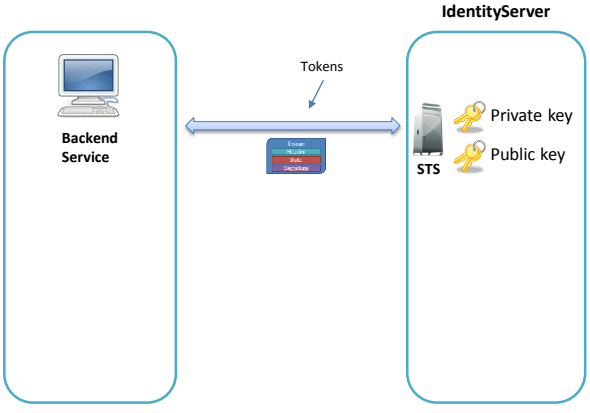
First, we add a **private** key to our IdentityServer



All issued tokens are signed using the **private key**

Signing using private/public keys

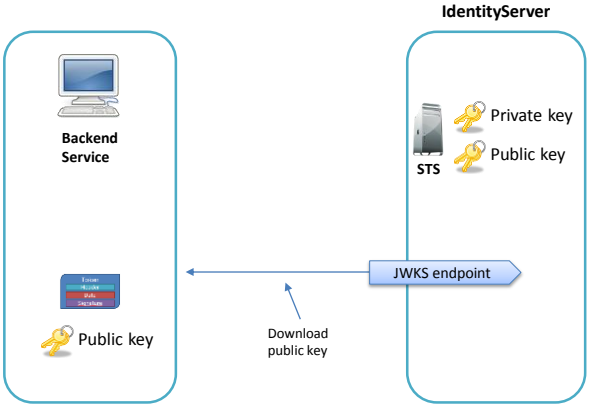
The STS issues **tokens** for the client



How can the client verify the token?

Signing using private/public keys

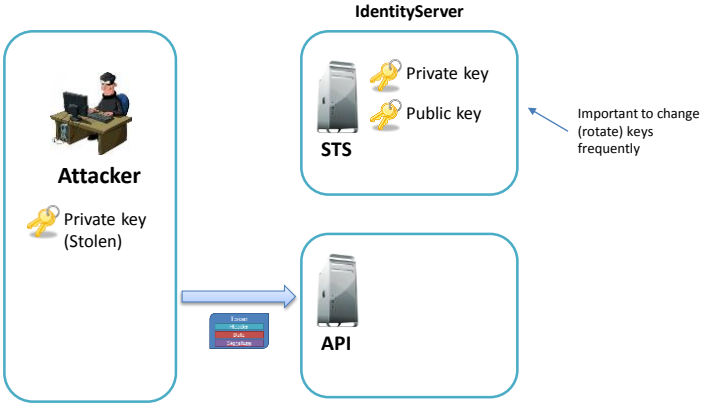
The client downloads the **public key** to validate the tokens



What happens if the **private key** is lost?

Losing the private key

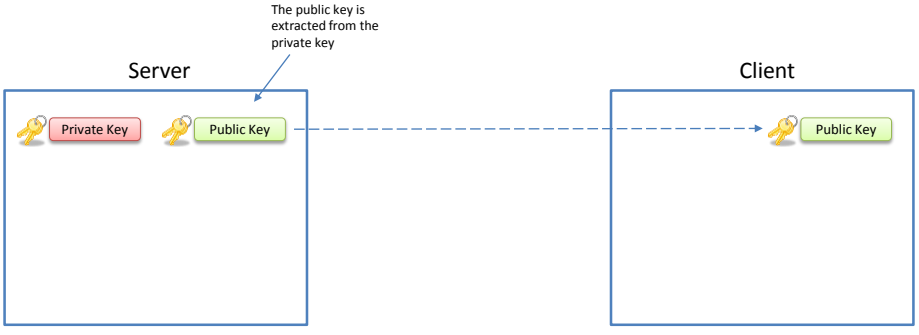
An attacker can issue their own tokens without the STS!



How does **public/private-key** signing work?

Public/private-key signing

First we generate a **public** and **private** key pair



Then we pass that **public** key to the receiver

The public key is **public**, does not need to be protected

Public/Private-key signing

A **signature** is then produced using a signature algorithm

The **data** and **signature** is then sent to the client

Public/Private-key signing

The data/signature is then **verified** using the **public key**

How can this be safe?

Private/Public-key algorithms

Private/Public-key algorithms

The two most common algorithms are:

- **RSA**
  Rivest–Shamir–Adleman
- **ECDSA**
  Elliptic Curve Digital Signature Algorithm

RSA

Rivest–Shamir–Adleman (RSA)

- Published in 1977
- Everyone supports RSA
- Today the minimum key-size is **2048** bits
- **3072** and **4096** bits also exists ,but this is much slower (4096 might be 8-10 times slower than 2048 to compute)
- **4096** bit crypto improves the security by 16% (not the double)
- Trusted and well established

ECDSA

Elliptic Curve Digital Signature Algorithm (ECDSA)

- Uses Elliptic-curve cryptography (ECC)
- Faster than RSA
  (Except during signature verification)
- Shorter keys compared to RSA

Effective strength	RSA	ECDSA
112	2048	224-255
128	3072	256-383
192	7680	384-511

- Used by Bitcoins
- Unclear if the standard NIST-curves has NSA-backdoors

- NIST (National Institute of Standards and Technology)
- NSA (National Security Agency)

Creating RSA private/public keys

Creating RSA private/public keys

To create our keys, we will use:

- **OpenSSL**
  An open-source toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols.

Supports both **RSA** and **ECDSA** keys

OpenSSL
Cryptography and SSL/TLS Toolkit
https://www.openssl.org/

Creating RSA key pair

First, we create the **private key**

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -aes256 -out rsa-private-key.pem
```

Parameter	Description
genpkey	Private key generating utility in OpenSSL
-algorithm	The algorithm to use, RSA or EC
-pkeyopt	Key-generation options rsa_keygen_bits:2048 – Create a 2048 bit key (Other key-sizes are for example 3072 and 4096)
-aes256	Encrypt the private key using a passphrase
-out	The name of the private-key file

https://www.openssl.org/docs/man1.1.0/man1/req.html

Creating RSA key pair

Optionally, if you want to extract the **public key**:

```
openssl rsa -in rsa-private-key.pem –pubout -out rsa-public-key.pem
```

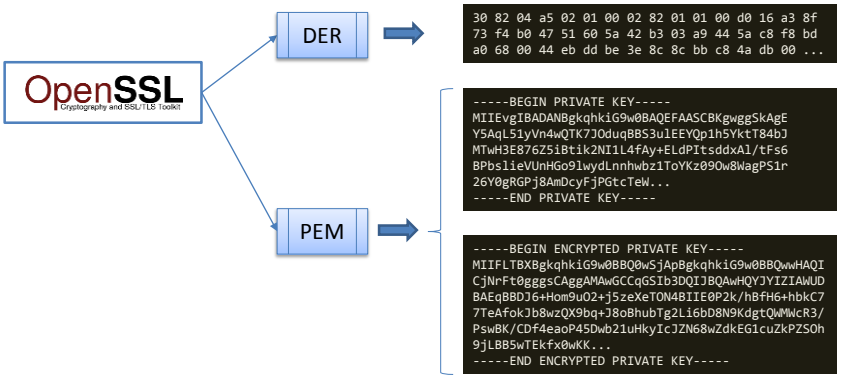
Parameter	Description
rsa	Use the OpenSSL RSA key processing tool
-in	The private key filename
-pubout	Export the public key
-out	The name of the public key to create

https://www.openssl.org/docs/man1.1.0/man1/req.html

Key file formats

Key file formats

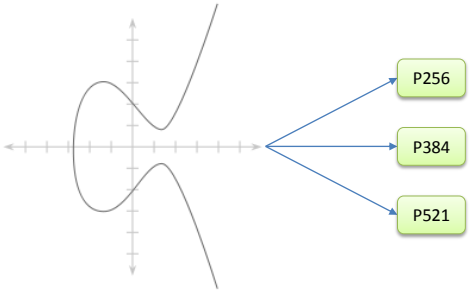
To store keys, we use either the **.DER** or **.PEM** format



The PEM file is a **base-64** encoded version of the DER file

PEM (Privacy-Enhanced Mail)
https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail
DER (Distinguished Encoding Rules)
https://en.wikipedia.org/wiki/X.690

Creating ECDSA private/public keys



Creating ECDSA signing key

First, we need to generate a private key

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -aes256 -out p256-private.pem
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-384 -aes256 -out p384-private.pem
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-521 -aes256 -out p521-private.pem
```

Parameter	Description
genpkey	Private key generating utility in OpenSSL
-algorithm	The algorithm to use, RSA or EC
-pkeyopt	Key-generation options ec_paramgen_curve:P-256 (Use the prime256v1 curve) ec_paramgen_curve:P-384 (Use the secp384r1 curve) ec_paramgen_curve:P-521 (Use the secp521r1 curve)
-aes256	Encrypt the public key using a passphrase
-out	The filename of the private key

https://www.openssl.org/docs/man1.1.1/man1/ecparam.html

Creating ECDSA signing key

Optionally, if you want to extract the **public key**:

```
openssl ec -in p256-private.pem -pubout -out p256-public.pem
openssl ec -in p384-private.pem -pubout -out p384-public.pem
openssl ec -in p521-private.pem -pubout -out p521-public.pem
```

Parameter	Description
ec	Use the OpenSSL EC key processing tool
-in	The private key filename
-pubout	Export the public key
-out	The name of the public key to create

https://www.openssl.org/docs/man1.1.1/man1/ecparam.html

Exercise

Let’s do module exercise #9

Discussing the exercises with your fellow students is encouraged!

Please write in the chat when you are done

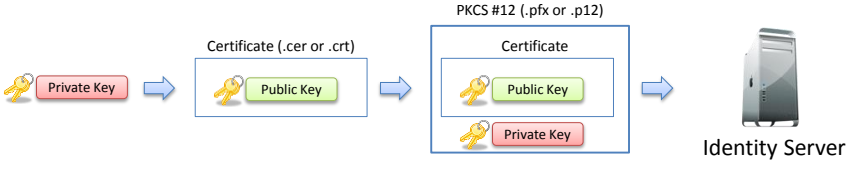
Keys, certificates and PKCS #12

Module #10

Adding keys to IdentityServer

Working directly with **key** files in .NET Core is not so easy

One option is to involve a **certificate** and a **PKCS #12** file



Let's go through this process

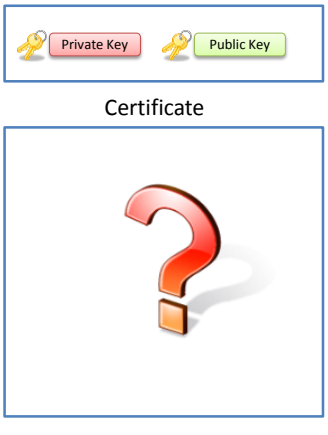
In .NET 5 we will have better support for .PEM files

Creating the certificate

What is the purpose of a certificate?

Creating the certificate

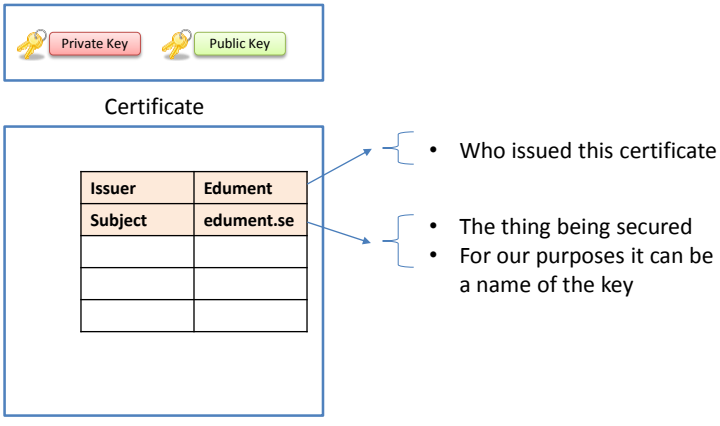
A **certificate** is used to prove the ownership of a **private key**



What does a certificate contain?

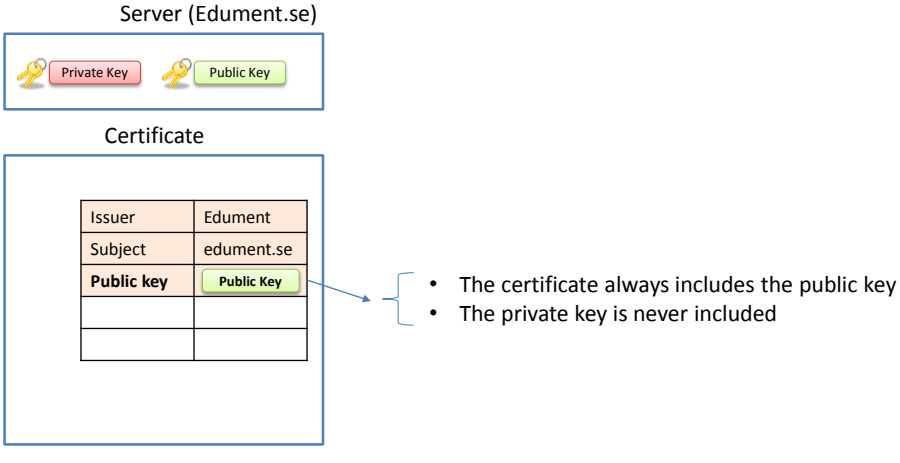
Creating the certificate

We first add the **issuer** and **subject**



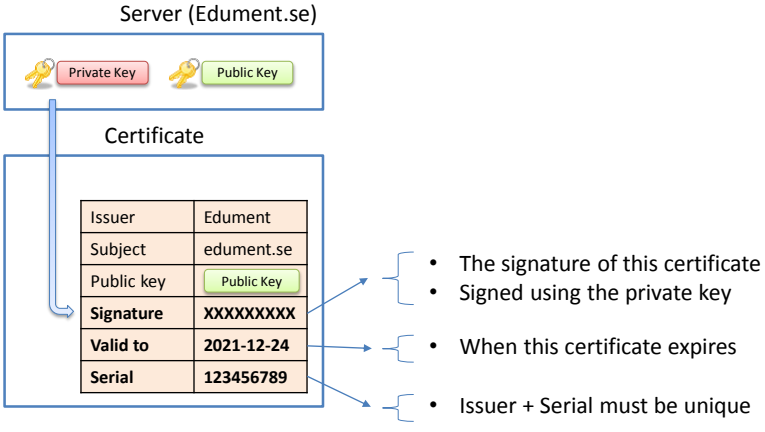
Creating a certificate

Then we add our public key



Creating a certificate

Then we add the **signature, expire date** and **serial number**



X.509 Certificate specification
https://tools.ietf.org/html/rfc5280

X.509

X.509 Certificates

X.509 is a standard for **public key certificates**

To load a certificate in .NET:

```
var cert = new X509Certificate2(fileName: "C:\\p256-cert.crt",
                                password: "edument");
```

New version of X509Certificate

What is the difference between X509Certificate2 and X509Certificate in .NET?
https://stackoverflow.com/questions/1182612

Demonstration

Creating a certificate

We can use OpenSSL to create an x509 certificate

RSA

```
openssl req -new -x509 -key rsa-private-key.pem -days 365 -subj "/CN=MyRSACert" -out rsa-cert.crt
```

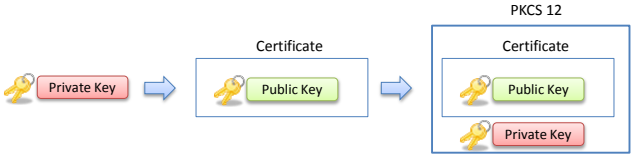
ECDSA

```
openssl req -new -x509 -key p256-private-key.pem -days 365 -subj "/CN=MyP256Cert" -out p256-cert.crt
```

Parameter	Description
req	Certificate request and certificate generating utility.
-new	Generate a new certificate request
-x509	We want a self-signed certificate
-key	name of the private key
-days	When does this certificate expires (days)
-subj	sets subject name
-out	This specifies the output filename

https://www.openssl.org/docs/manmaster/man1/req.html

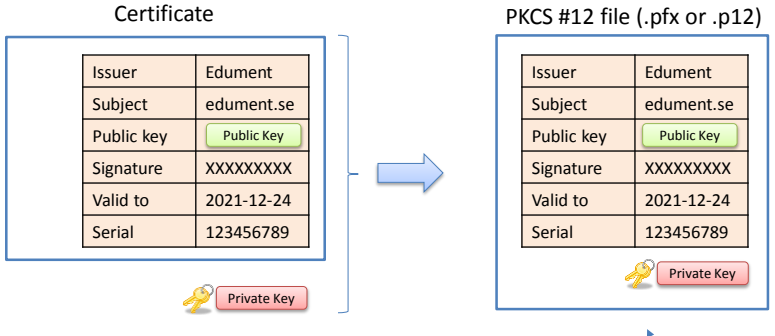
Creating a PKCS 12 file



Creating a PKCS 12 file

The certificate file only contains our **public key**

We can bundle the **cert** and **private key** into a **PKCS #12** file



https://en.wikipedia.org/wiki/PKCS

Creating a certificate

To create the PKCS 12 file:

RSA

```
openssl pkcs12 -export -inkey rsa-private-key.pem -in rsa-cert.crt -out rsa.pfx
```

ECDSA

```
openssl pkcs12 -export -inkey p256-private.pem -in p256-cert.crt -out p256.pfx
```

Parameter	Description
pkcs12	Use the PKCS#12 file utility
-export	Create a PKCS#12 file
-inkey	Name of the private key
-in	Name of the certificate file
-out	This specifies the output filename

https://www.openssl.org/docs/man1.1.0/man1/req.html

Exercise

Let’s do module exercise #10.x

Discussing the exercises with your fellow students is encouraged!

Please write in the chat when you are done

Adding token signing keys

Module #11

Creating the token signing keys

By default, we use this method to create our **signing key**:

```
// not recommended for production -
// you need to store your key material somewhere secure
builder.AddDeveloperSigningCredential();
```

Included in the Identity Server templates

This corresponds to the following defaults:

```
builder.AddDeveloperSigningCredential(
   persistKey: true,
   filename: null,          //use "tempkey.jwk"
   signingAlgorithm: IdentityServerConstants.RsaSigningAlgorithm.RS256);
```

What does this method do? 

Creating the token signing keys

At startup it will create a local file named **tempkey.jwk**

JSON Web Key (JWK) is a standard for representing a cryptographic key.

The JKWS endpoint

Clients can then download **the public signing key** from:

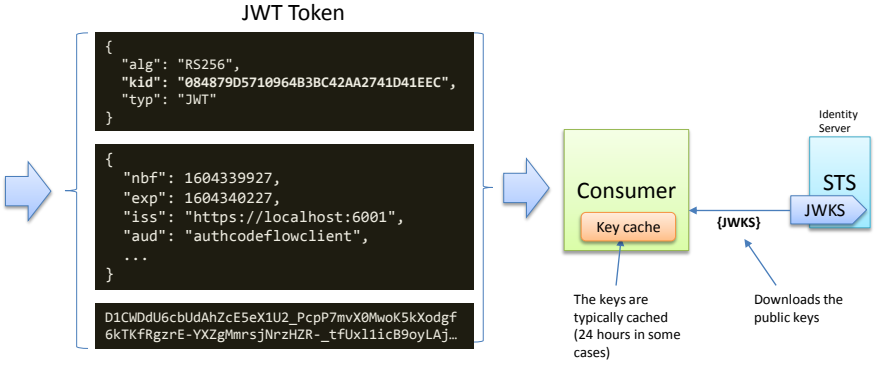
```
https://identity.example.com/.well-known/jwks.json
```

The JWKS contains a set of **JSON Web Keys (JWK)**

The JWKS endpoint can contain multiple keys

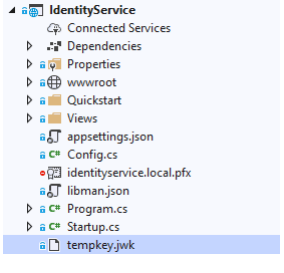
The JKWS endpoint

Then the consumer downloads all the **public keys**



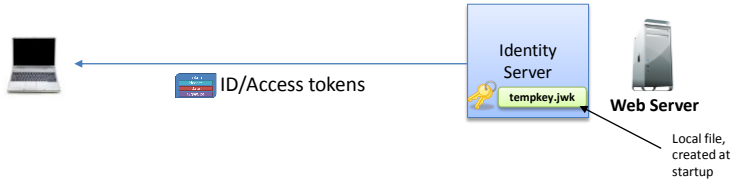
Using the **kid**, it can lookup the key to verify the signature

The problem with **tempkey.jwk**

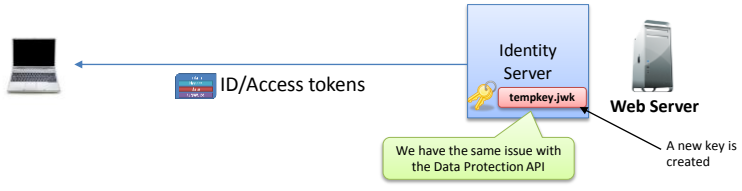


The problem with tempkey.jwk

This key is used to **sign** and **verify** tokens:



If we do a full redeploy, a new **key** is created

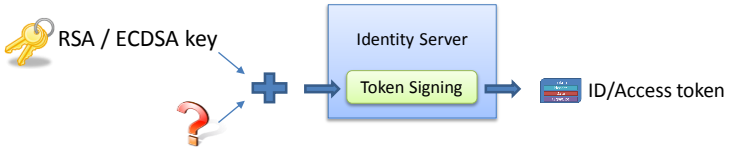


Now existing tokens can't be verified using the new key!

Signature schemes

Signature schemes

Just having a **private key** is not enough to sign tokens



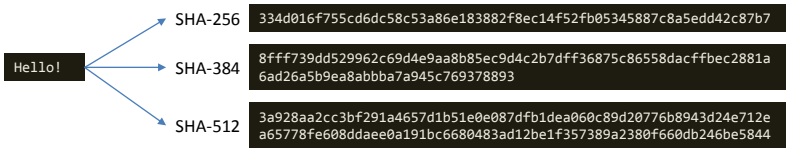
We need to choose a **signature scheme**

Supported signature schemes

IdentityServer supports these **signature schemes**

Algorithm	Description
RS256, RS384, RS512	RSASSA-PKCS1-v1_5 using SHA-256/384/512
PS256, PS384, PS512	RSASSA-PSS using SHA-256/384/512 and MGF1 with SHA-256/384/512
ES256, ES384, ES512	ECDSA using P-256/384/512 and SHA-256/384/512

What does **SHA-256**, **SHA-384**, **SHA-512** mean?



Online hash tool
https://md5calc.com/hash

Supported signature schemes

IdentityServer supports these **signature schemes**

Algorithm	Description
RS256, RS384, RS512	RSASSA-PKCS1-v1_5 using SHA-256/384/512
PS256, PS384, PS512	RSASSA-PSS using SHA-256/384/512 and MGF1 with SHA-256/384/512
ES256, ES384, ES512	ECDSA using P-256/384/512 and SHA-256/384/512

What does **RSxxx**, **PSxxx** and **ESxxx** mean? 

RFC 7518 - JSON Web Algorithms (JWA)
https://tools.ietf.org/html/rfc7518

The RSxxx signature schemes

The **RSxxx** signature schemes:

Name	Algorithm	Hash function
RS256	RSASSA-PKCS1-v1_5	SHA-256
RS384	RSASSA-PKCS1-v1_5	SHA-384
RS512	RSASSA-PKCS1-v1_5	SHA-512

RFC 8017 says:

Although no attacks are known against RSASSA-PKCS1-v1_5,
in the interest of increased robustness:

- **RSASSA-PSS** is REQUIRED in new applications.
- **RSASSA-PKCS1-v1_5 is included only for compatibility with existing applications.**
- A key of size **2048 bits** or larger MUST be used with this algorithm.

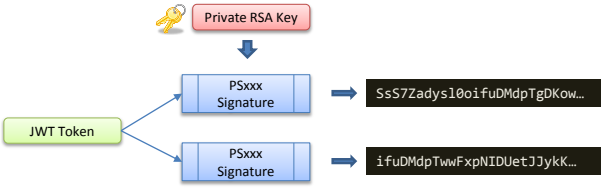
PKCS #1: RSA Cryptography Specifications Version 2.2
https://tools.ietf.org/html/rfc8017

The PSxxx signature schemes

PSxxx is a **Probabilistic** Signature Scheme that includes a bit of randomness in the algorithm

Name	Algorithm	Hash function	Mask function
PS256	RSASSA-PSS	SHA-256	MGF1 SHA-256
PS384	RSASSA-PSS	SHA-384	MGF1 SHA-384
PS512	RSASSA-PSS	SHA-512	MGF1 SHA-512

Multiple signatures for a token will give different results



PKCS #1: RSA Cryptography Specifications Version 2.2
https://tools.ietf.org/html/rfc8017

The ESxxx signature schemes

The **ESxxx** scheme uses the modern **Elliptic Curve Digital Signature Algorithm** (ECDSA)

Name	Algorithm	Curve	Hash function
ES256	ECDSA	P-256	SHA-256
ES384	ECDSA	P-384	SHA-384
ES512	ECDSA	P-521	SHA-512

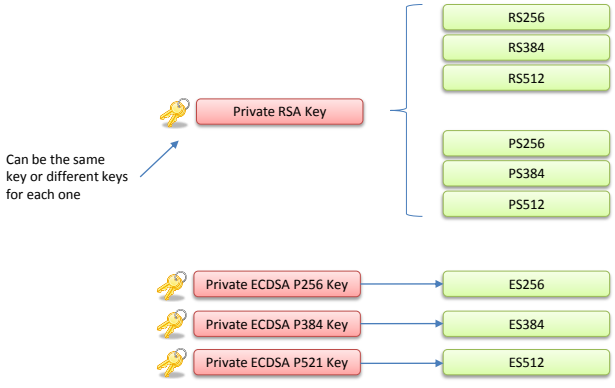
Yes, 521!

Provides equivalent security to RSA cryptography but using shorter key sizes and with greater processing speed for many operations.

JSON Web Algorithms (JWA)
https://tools.ietf.org/html/rfc7518

Signature schemes summary

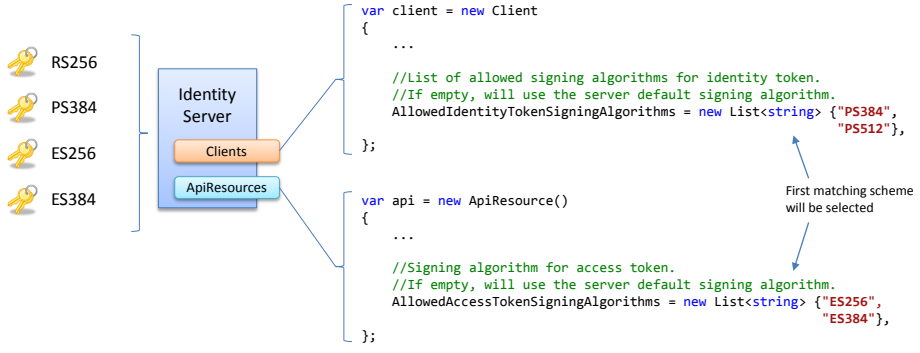
The different signing algorithms and their key-relations



Multiple signing keys

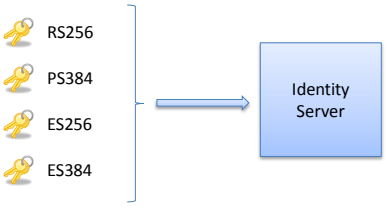
Multiple signing keys

We can add multiple signing keys and schemes



Clients and **ApiResources** can define their preferred schemes

Adding signing keys in production



Adding keys in production

To add the keys, we use the **AddSigningCredential** method:

```
ECDsaSecurityKey ecdkey = ...;
builder.AddSigningCredential(key: ecdkey,
        signingAlgorithm: IdentityServerConstants.ECDsaSigningAlgorithm.ES256);

RsaSecurityKey rsakey = ...;
builder.AddSigningCredential(key: rsakey,
        signingAlgorithm: IdentityServerConstants.RsaSigningAlgorithm.RS256);

SecurityKey seckey = ...;
builder.AddSigningCredential(key: seckey,
        signingAlgorithm: SecurityAlgorithms.RsaSha256);

SigningCredentials signcreds = ...;
builder.AddSigningCredential(credential: signcreds);

X509Certificate2 x509cert = ...;
builder.AddSigningCredential(certificate: x509cert,
                             signingAlgorithm: "RS256");
```

AddSigningCredential has many overloads

IdentityServer Key Management

IdentityServer Key Management

In the documentation it says:

> If in ConfigureServices in your Startup.cs you were previously using AddDeveloperSigningCredential, that can be removed. Automatic key management is now a built-in feature.

Not part of the free Starter edition of IdentityServer.

This is a new feature in version 5 of IdentityServer

What does the key manager do? 

IdentityServer Key Management

The Key Management function handles:

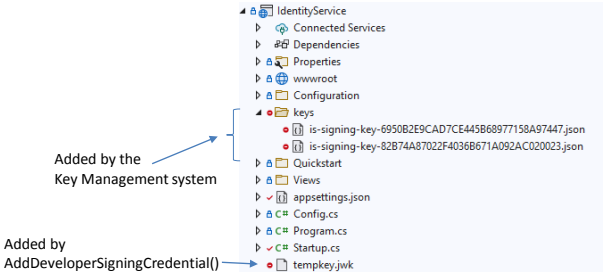
- automatic key rotation
- secure storage of keys at rest
- announcement upcoming new keys in discovery
- maintenance of retired keys in discovery

Do check out the **licensing** if you want to use this feature

https://docs.duendesoftware.com/identityserver/v5/fundamentals/keys/

IdentityServer Key Management

We can see the various key files created:



If you don't use the key manager, then you should disable it:

```
var builder = services.AddIdentityServer(options =>
{
    options.KeyManagement.Enabled = false;
    ...
});
```

https://github.com/DuendeSoftware/IdentityServer/discussions/394

Exercise

Let’s do main exercise #11.x

Discussing the exercises with your fellow students is encouraged!

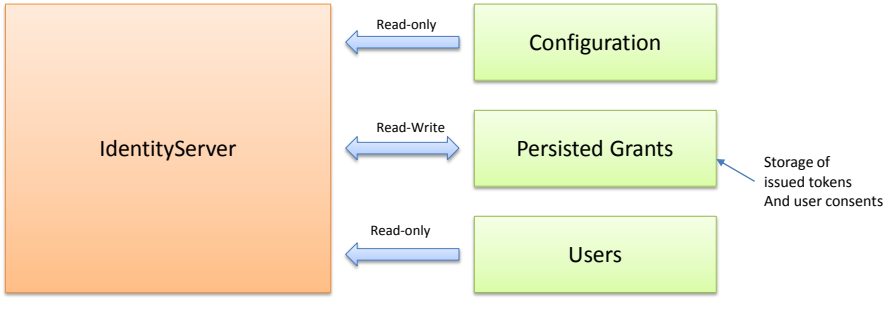
Please write in the chat when you are done

IdentityServer and the database

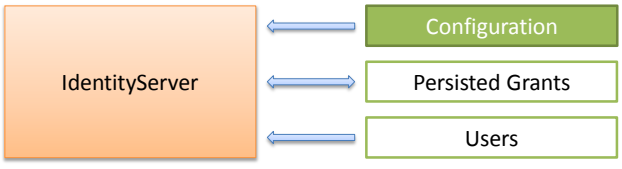
Module #12

Identity Server storage

IdentityServer depends on the following **data sources**

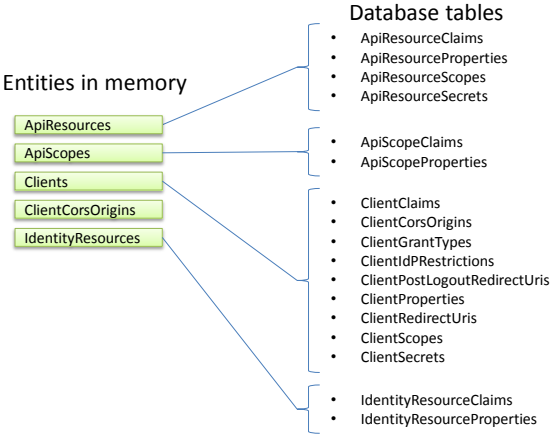


Configuration



Identity Server Configuration

The default configuration data model is quite big and complex



How should we implement this?

Identity Server Configuration

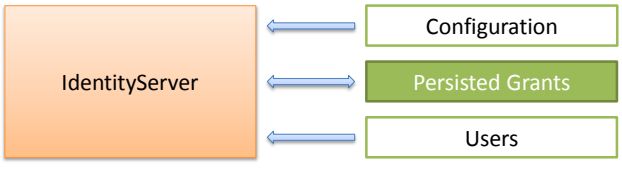
The big question, how often does this data change?

In most systems, the it does not change that often!

So, why should we spend time on adding database support and a fancy database admin GUI?

Let’s keep it in code, with a little refactoring!

Persisted Grants



Persisted Grants

This is about keeping track of **issued items** and **user consent**

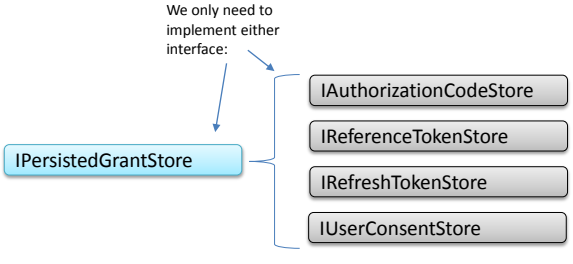
It keeps track of:

- Authorization codes
- Reference tokens
- Refresh tokens
- User consent

How do we implement this?

Persisted Grants

To implement this ,we have two options:



We will choose to implement the **IPersistedGrantStore**

It's a bit simpler and that's what most libraries choose to implement

IPersistedGrantStore

The interface is defined as follows:

```
/// <summary>Interface for persisting any type of grant.</summary>
  public interface IPersistedGrantStore
  {
    /// <summary>Stores the grant.</summary>
    Task StoreAsync(PersistedGrant grant);

    /// <summary>Gets the grant.</summary>
    Task<PersistedGrant> GetAsync(string key);

    /// <summary>Gets all grants based on the filter.</summary>
    Task<IEnumerable<PersistedGrant>> GetAllAsync(PersistedGrantFilter filter);

    /// <summary>Removes the grant by key.</summary>
    Task RemoveAsync(string key);

    /// <summary>Removes all grants based on the filter.</summary>
    Task RemoveAllAsync(PersistedGrantFilter filter);
  }
}
```

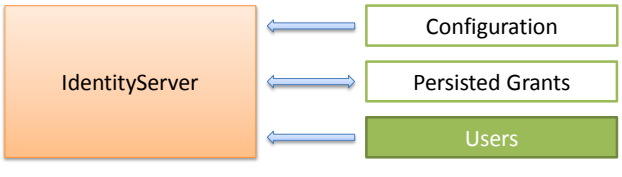
If we use a SQL Server, the table will look like this:

	Key	Type	SubjectId	SessionId	ClientId	Description	CreationTime	Expiration	ConsumedTime	Data
1	7f4RKqcvtcwwXjR...	refresh_token	88421113	0B1F8E33...	authcodeflowclient_dev	NULL	2020-09-26 0...	2020-10-26...	NULL	{"CreationTime":"2020-09-26T07:12
2	9mlYkIfJR0tChtpa...	user_consent	818727	NULL	authcodeflowclient_dev	NULL	2020-09-26 0...	NULL	NULL	{"SubjectId":"818727","ClientId":"a
3	f9O8zKUjmwzUBI...	refresh_token	818727	B306429F...	authcodeflowclient_dev	NULL	2020-09-26 0...	2020-10-26...	NULL	{"CreationTime":"2020-09-26T07:13
4	tVQVgjDlwMGRt7...	user_consent	88421113	NULL	authcodeflowclient_dev	NULL	2020-09-26 0...	NULL	NULL	{"SubjectId":"88421113","ClientId":"
5	wdPtX244RMjYyM...	refresh_token	88421113	F8F41FBE...	authcodeflowclient_dev	NULL	2020-09-26 0...	2020-10-26...	NULL	{"CreationTime":"2020-09-26T07:03

Different types of grants

We will explore this field in the exercises

Users



Persisted Grants

To support users, all we need to do is to implement this:

```
public interface IProfileService
{
    /// <summary>
    /// This method is called whenever claims about the user are requested
    /// (e.g. during token creation or via the userinfo endpoint)
    /// </summary>
    /// <param name="context">The context.</param>
    /// <returns></returns>
    Task GetProfileDataAsync(ProfileDataRequestContext context);

    /// <summary>
    /// This method gets called whenever identity server needs to determine if the user is valid or active
    /// (e.g. if the user's account has been deactivated since they logged in).
    /// (e.g. during token issuance or validation).
    /// </summary>
    /// <param name="context">The context.</param>
    /// <returns></returns>
    Task IsActiveAsync(IsActiveContext context);
}
```

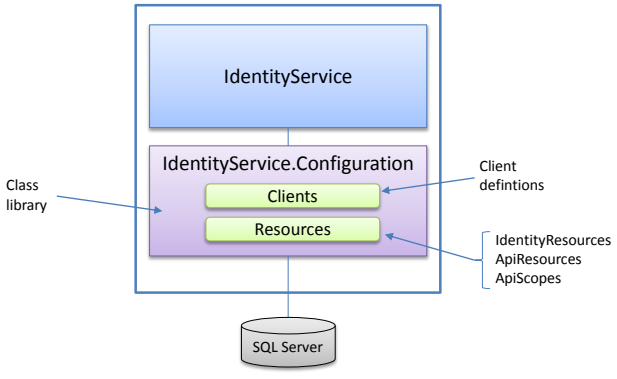
Do notice that IdentityServer never writes to the database

We will deal with users in the next module

Implementation

Implementation

We place the configuration in its own class library



What about the persisted grants?

Implementing Persisted Grants

We will use the one provided by IdentityServer

Duende.IdentityServer.EntityFramework by Duende Software, **1,63K** downloads
EntityFramework persistence layer for Duende IdentityServer

Duende.IdentityServer.EntityFramework.Storage by Duende Software, **1,76K** downloads
EntityFramework persistence layer for Duende IdentityServer

Many other alternatives do exist

These two packages implements:

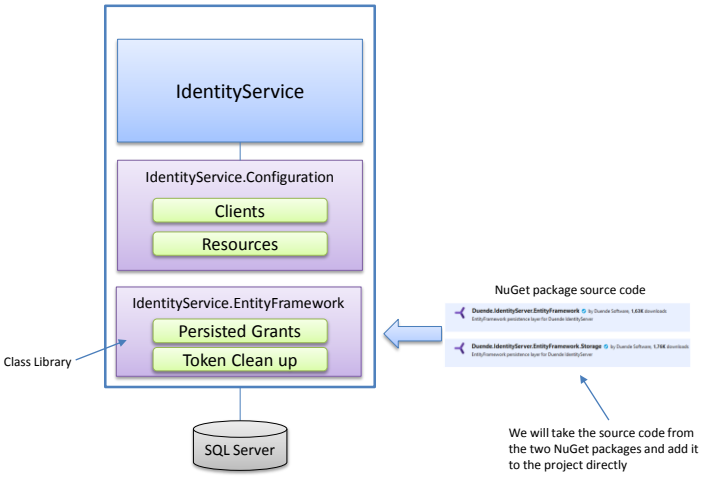
We will not use in this course

- CorsPolicyService
- **Persisted grants**
- Configuration
- Migrations
- **Token cleanup service**

Remove old persisted grants

Implementing Persisted Grants

We place this in its own class library:



Exercise

Let’s do main exercise #12

Discussing the exercises with your fellow students is encouraged!

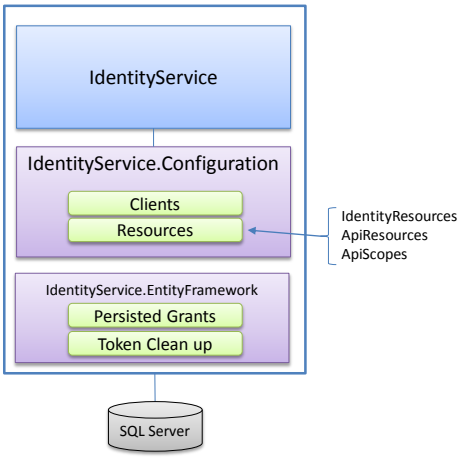
Please write in the chat when you are done

Users

Module #13

IdentityServer and the users

So far, we have this architecture:

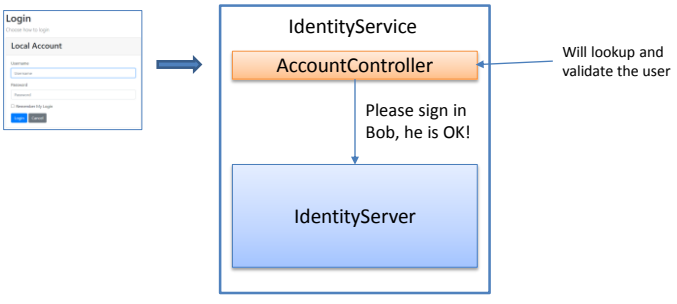


How does IdentityServer deal with **users**?

IdentityServer and the users

IdentityServer does not care that much about users!

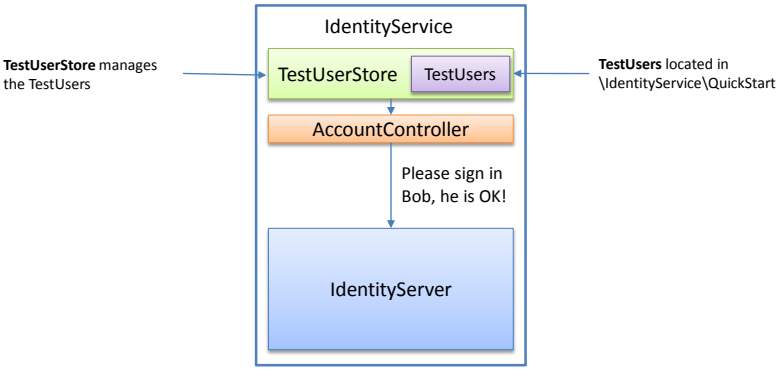
Account Controller tells IdentityServer to sign in the user!



So, where is the users?

IdentityServer and the users

In our setup so far, a **TestUsers** class holds the users

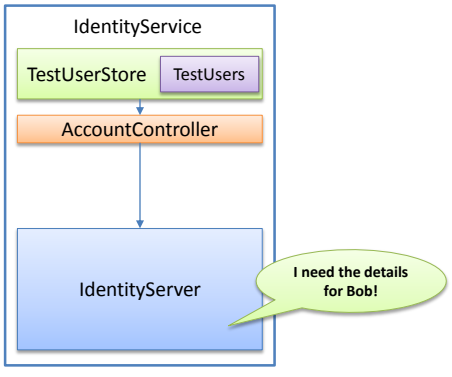


However, we have one problem

https://github.com/DuendeSoftware/IdentityServer/blob/main/src/IdentityServer/Test/TestUserStore.cs

IdentityServer and the users

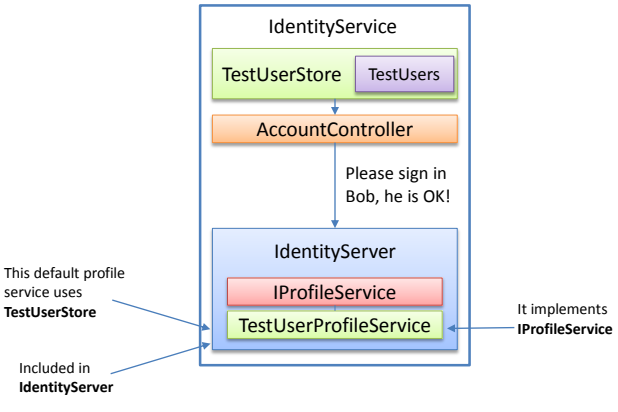
IdentityServer needs to be able to lookup users after login



We to implement a **ProfileService**

IdentityServer and the users

Right now we are using a **TestUserProfileService**

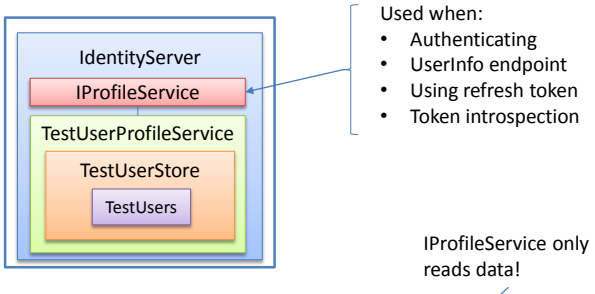


What services does **IProfileService** provide?

https://github.com/DuendeSoftware/IdentityServer/blob/main/src/IdentityServer/Test/TestUserProfileService.cs

IdentityServer and the users

IProfileService allows IdentityServer to get the user details

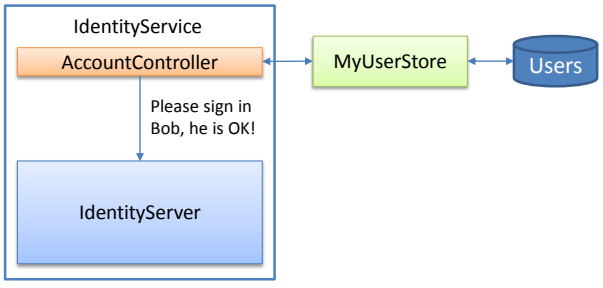



```
/// This interface allows IdentityServer to connect to your user and profile store.
public interface IProfileService
{
    /// This method is called whenever claims about the user are requested
    ///(e.g. during token creation or via the userinfo endpoint)
    Task GetProfileDataAsync(ProfileDataRequestContext context);

    /// This method gets called whenever identity server needs to determine
    /// if the user is valid or active.
    Task IsActiveAsync(IsActiveContext context);
}
```

Adding external users

Adding external users

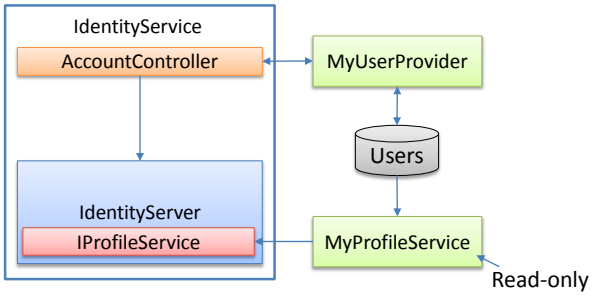
We can let the **AccountController** to use a different source



Is that it?

IdentityServer and the users

We also need to Implement the **IProfileService** interface



Implementation

Implementation

This is the hardest part! so many requirements

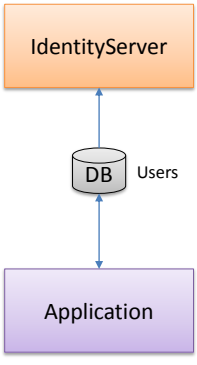
IdentityServer does not care about this!

- Login form
- Register new users
- Email confirmation email
- 2-Factor authentication
- Disabling users
- Managing users
- Change password

How to approach this? 

Implementation

The best is to separate the requirements into two parts



- Login & Logout
- Issue tokens
- OpenID Connect

Keep the Identity Server as clean as possible!

Single responsibility!

- Register new users
- Email confirmation email
- 2-Factor authentication
- Disabling users
- Managing users

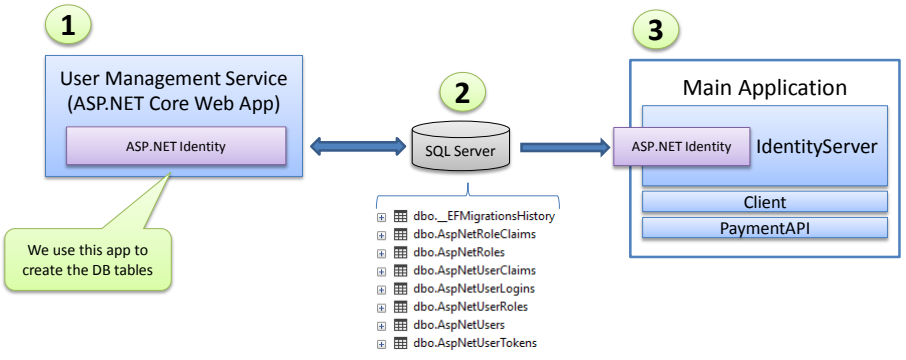
Keep these items in a separate service, perhaps in your application?

Usually very different between implementations

Getting concrete

Getting concrete

What we will do is the following:



What is **ASP.NET Core Identity**?

ASP.NET Core Identity

What does ASP.NET Core Identity do?

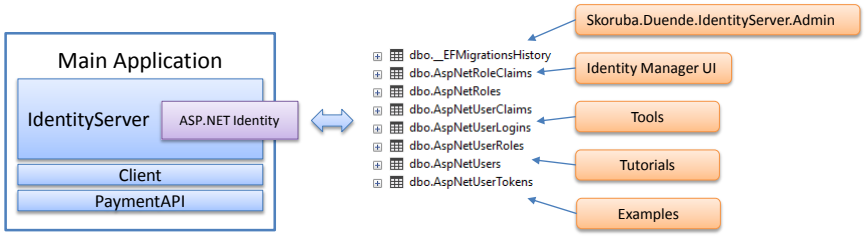
- Adds a fully featured membership system
- Adds user interface (UI) login functionality
- Manages users, passwords, profile data, roles, claims, tokens, email confirmation, and more.

Why do we use it?

ASP.NET Core Identity

What Why do we use it?

- To get access to the standard set of DB tables to store users
- There are many tutorials and projects around it



https://github.com/mguinness/IdentityManagerUI
https://github.com/skoruba/Duende.IdentityServer.Admin

Admin UI

Admin UI

Worth trying out:

…our friends from RSK will also release in **January** a **community edition** of their **AdminUI** tool with the same client limit.

Which means that you can get a full featured Duende IdentityServer + UI absolutely free.

https://blog.duendesoftware.com/posts/20201210_community_edition

Exercise

Let's do **module** and **main** exercise **#13.x**

Discussing the exercises with your fellow students is encouraged!

Please write in the chat when you are done

Tokens and claims

Module #14

Claims

Claims are facts about the user

User: Joe

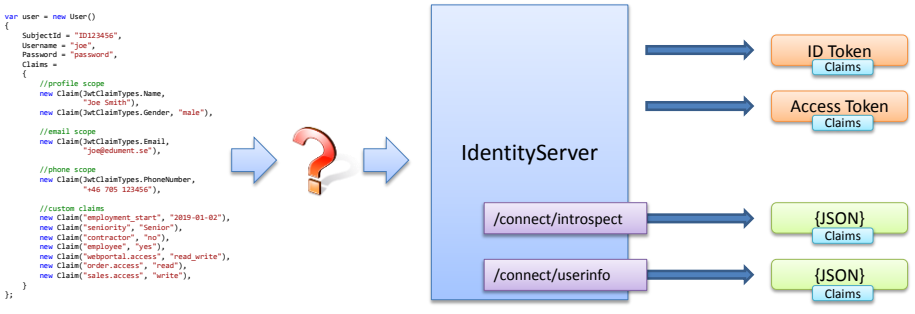
Claims	
Type	**Value**
email	joe@edument.se
name	Joe Smith
contractor	no
order.access	read
sales.access	Write
…	…

```
var user = new User()
{
    SubjectId = "ID123456",
    Username = "joe",
    Password = "password",
    Claims =
    {
        //profile scope
        new Claim(JwtClaimTypes.Name, "Joe Smith"),
        new Claim(JwtClaimTypes.Gender, "male"),

        //email scope
        new Claim(JwtClaimTypes.Email,
                  "joe@edument.se"),

        //phone scope
        new Claim(JwtClaimTypes.PhoneNumber,
                  "+46 705 123456"),

        //custom claims
        new Claim("employment_start", "2019-01-02"),
        new Claim("seniority", "Senior"),
        new Claim("contractor", "no"),
        new Claim("employee", "yes"),
        new Claim("webportal.access", "read_write"),
        new Claim("order.access", "read"),
        new Claim("sales.access", "write"),
    }
};
```

How do **apps / clients** get access to these claims?

Tokens and claims

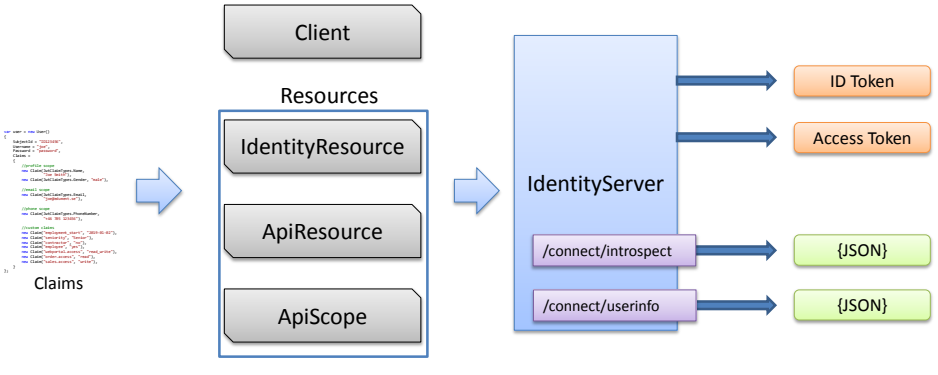
IdentityServer provides the following **tokens** and **endpoints**:



How do we map claims to these endpoints?

Tokens and claims

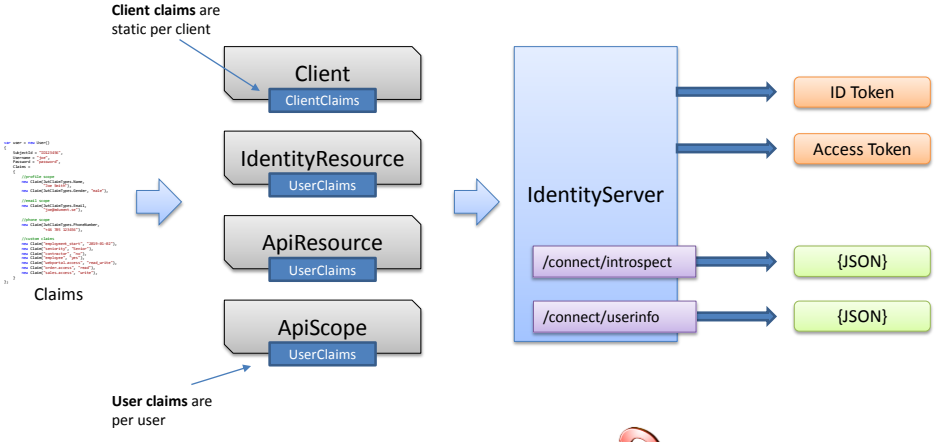
The mapping is defined in the **client** and **resources**



How do does this work?

Tokens and claims

Each resource can select a set of claims

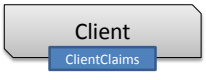


What goes where?

Client Claims

Client Claims

Client claims are static claims, client specific:

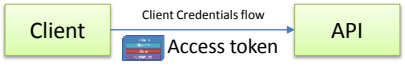



```
var client1 = new Client
{
    ClientId = "myclient",
    ClientName = "My Client",

    AllowedGrantTypes = GrantTypes.CodeAndClientCredentials,

    Claims = new List<ClientClaim>()
    {
        new ClientClaim("mobile", "yes"),
        new ClientClaim("mobiletype", "ios"),
        new ClientClaim("internal", "no")
    },
};
```

Not user specific

What is the main use case for client claims?

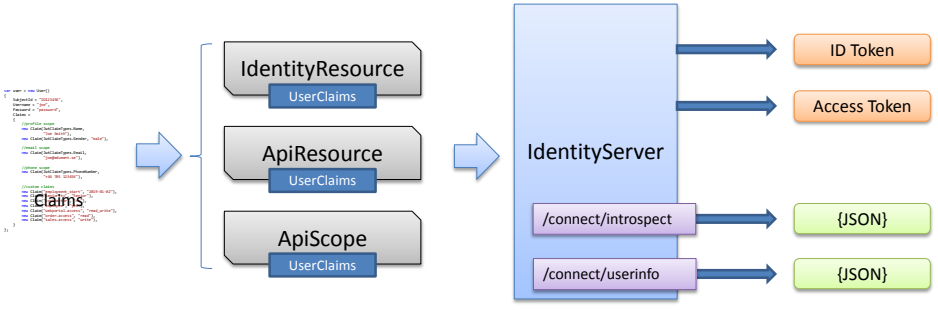
To be used with the **client credentials flow**



User Claims

User Claims

User claims are defined per user:



We can control if the claims should be added to the ID-token

```
// When requesting both an id token and access token, should the user claims always
// be added to the id token instead of requiring the client to use the UserInfo endpoint.
// Defaults to false.
AlwaysIncludeUserClaimsInIdToken = false,
```

Client secrets Frequently Asked Questions

```
var clientDev = new Client
{
    ClientId = "authcodeflowclient_dev",
    ClientName = "My Client application",
    ClientSecrets = new List<Secret>
    {
        new Secret
        {
            Value = "mysecret".Sha512()
        }
    },
    ...
```

Frequently Asked Questions

Why is secrets defined as a list?

```
new ApiResource()
{
    Name = "PaymentAPI",
    ApiSecrets = new List<Secret>
    {
        new Secret("apisecret1".Sha256()),
        new Secret("apisecret2".Sha256()),
        new Secret("apisecret3".Sha256()),
    },
}
```

```
new Client
{
    ClientId = "myclient",
    ClientSecrets = new List<Secret>
    {
        new Secret("mysecret1".Sha512()),
        new Secret("mysecret2".Sha512()),
        new Secret("mysecret3".Sha512()),
    }
};
```

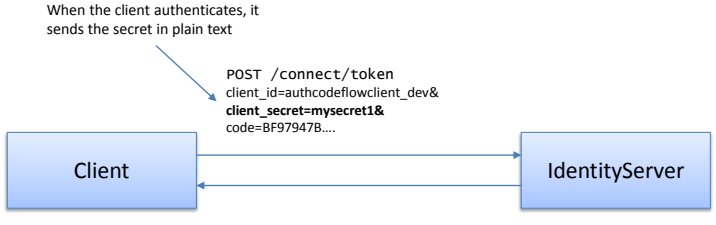
It is useful for secret rotation and rollover

Frequently Asked Questions

Should secrets be stored as **SHA-256** or **SHA-512**?

```
new ApiResource()
{
    Name = "PaymentAPI",
    ApiSecrets = new List<Secret>
    {
        new Secret("apisecret1".Sha256()),
        new Secret("apisecret2".Sha512()),
        new Secret("apisecret3".Sha256()),
    },
    ...
}
```

```
new Client
{
    ClientId = "myclient",
    ClientSecrets = new List<Secret>
    {
        new Secret("mysecret1".Sha512()),
        new Secret("mysecret2".Sha256()),
        new Secret("mysecret3".Sha512()),
    },
    ...
};
```



It doesn't matter, IdentityServer supports both

https://github.com/DuendeSoftware/IdentityServer/blob/main/src/IdentityServer/Validation/Default/HashedSharedSecretValidator.cs

Exercise

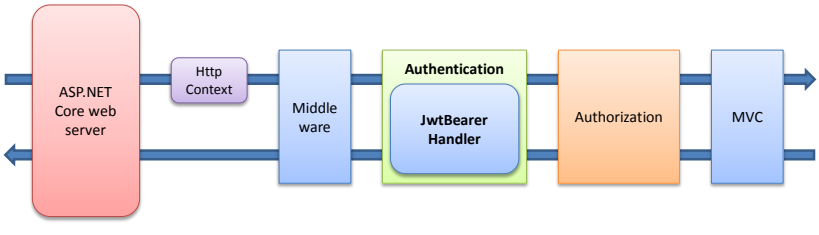
Let’s do **module** and **main** exercise #14.x

Securing API's

Module #15

Securing API's

To secure API's we use the **JwtBearer** handler

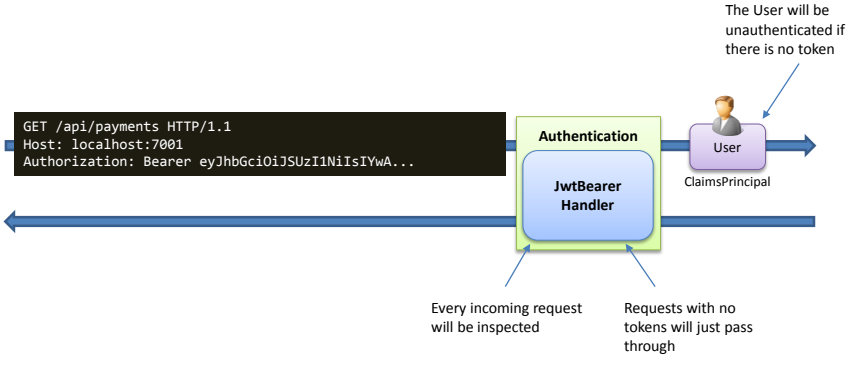


.NET **Microsoft.AspNetCore.Authentication.JwtBearer** by Microsoft, 86,6M downloads
ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.

What does JwtBearer handler do?

Securing API's

It looks for access token in the **Authorization** header

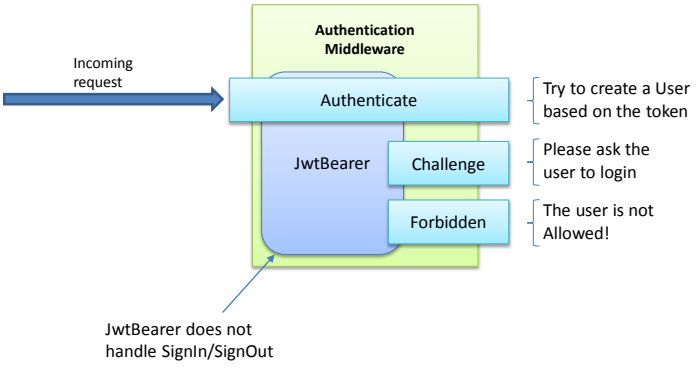


It's main task is to create a **user** from the **token**

How does JwtBearer implement this?

What does JwtBearer do?

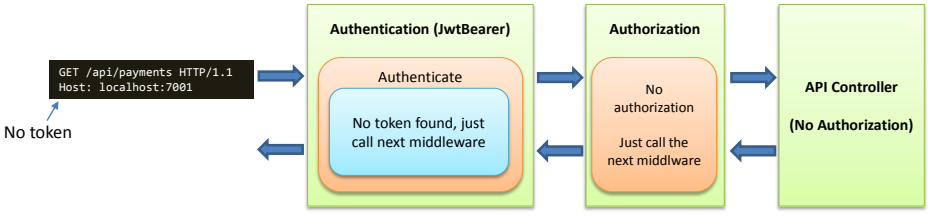
JwtBearer have three core functions:



What happens during authenticate?

Authenticate

For a request without token, the following will happen:

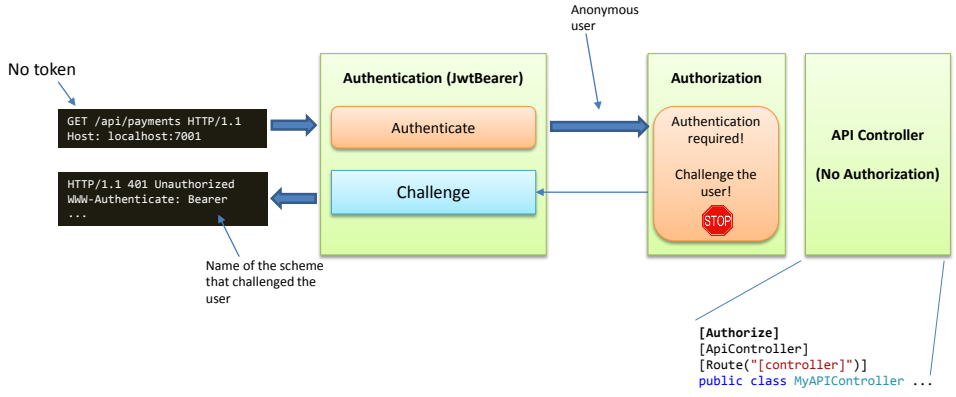


Authenticate is called on every incoming request

What happens if we have authorization rules?

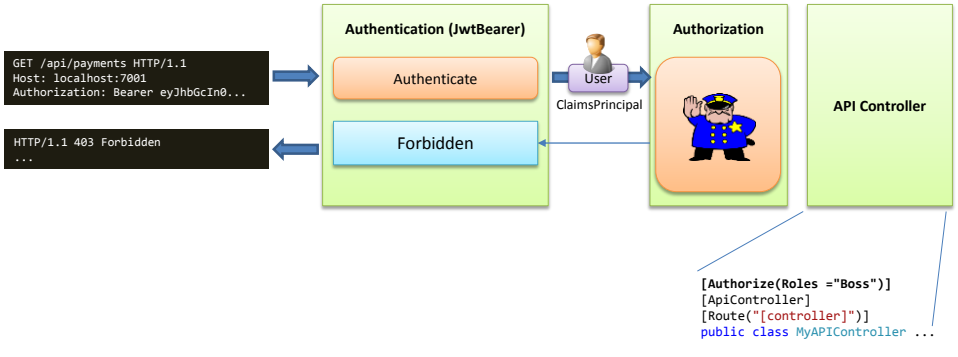
Authenticate

For a request without token to an API with authorization:



Authenticate

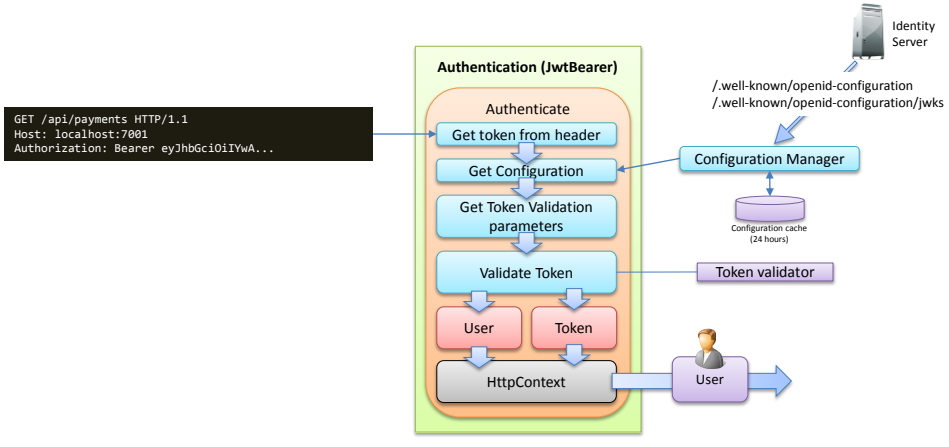
For a request with a token that fails authorization:



API Requests with token

Authenticate

For a request with a token, then the following will happen:

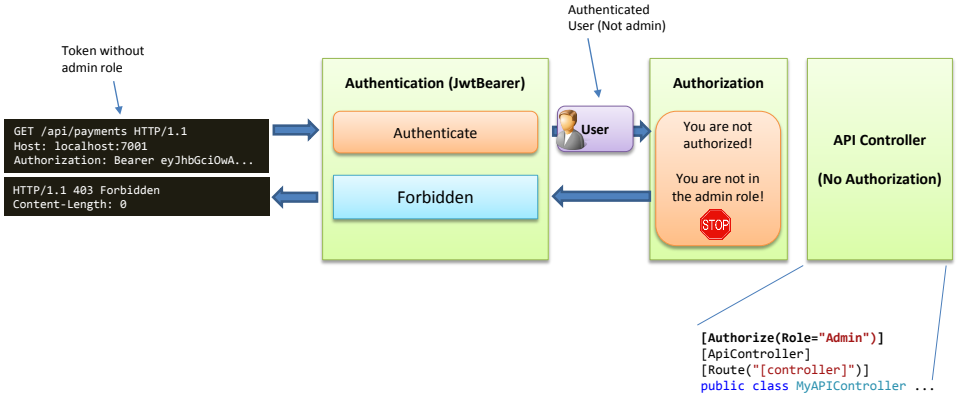


The **HttpContext** is then passed to the authorization

Forbidden

Forbidden

If you are authenticated, but not authorized

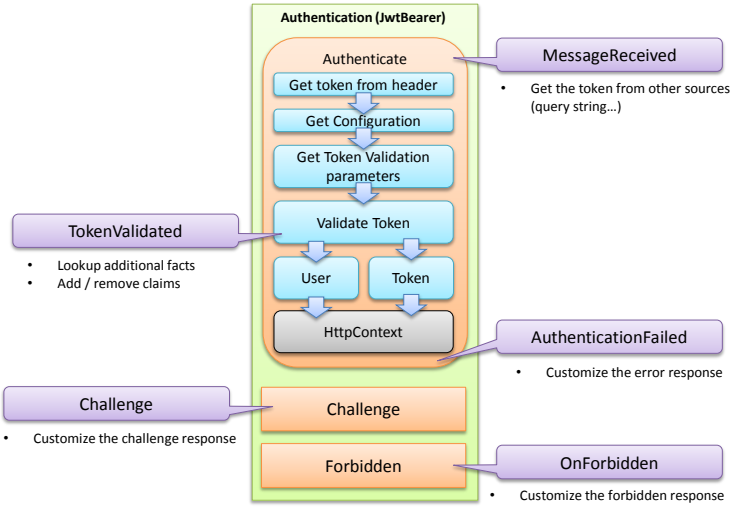


Forbidden will just return a **403** response

JWT Bearer events

Authenticate

We can hook into the following places:



How can we implement this?

JWT Bearer events

We can configure it in two ways:

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
.AddJwtBearer(opt =>
{
    opt.Audience = "payment";
    opt.Authority = "https://localhost:6001";

    opt.Events = new MinimalJwtBearerEvents();
});
```

Replace the current object with a custom one

```
public class MinimalJwtBearerEvents : JwtBearerEvents
{
    public override Task MessageReceived(MessageReceivedContext context)
    {
        return Task.CompletedTask;
    }
    public override Task AuthenticationFailed(AuthenticationFailedContext context)
    {
        return Task.CompletedTask;
    }
    public override Task Challenge(JwtBearerChallengeContext context)
    {
        return Task.CompletedTask;
    }
    public override Task Forbidden(ForbiddenContext context)
    {
        return Task.CompletedTask;
    }
    public override Task TokenValidated(TokenValidatedContext context)
    {
        return Task.CompletedTask;
    }
}
```

JWT Bearer events

Alternatively, we can define them as lambda expressions:

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
.AddMyJwtBearer(opt =>
{
    opt.Audience = "payment";
    opt.Authority = "https://localhost:6001";

    opt.Events.OnMessageReceived = context =>
    {
        // When Authenticate is called
        return Task.CompletedTask;
    };
    opt.Events.OnAuthenticationFailed = context =>
    {
        // When exceptions are thrown during request processing (invalid token...)
        return Task.CompletedTask;
    };
    opt.Events.OnChallenge = context =>
    {
        // Invoked before a challenge is sent back to the caller.
        return Task.CompletedTask;
    };
    opt.Events.OnForbidden = context =>
    {
        // Invoked if Authorization fails and results in a Forbidden response
        return Task.CompletedTask;
    };
    opt.Events.OnTokenValidated = context =>
    {
        // Token has passed validation and a ClaimsIdentity has been generated.
        return Task.CompletedTask;
    };
});
```

Adding and configuring JwtBearer

Adding and Configuring JwtBearer

To get started, we need to set:

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opt =>
    {
       opt.Authority = "https://localhost:6001";
       opt.Audience = "payment";


       opt.TokenValidationParameters.ClockSkew = TimeSpan.FromSeconds(60);

       // IdentityServer emits a type header by default, recommended extra check
       opt.TokenValidationParameters.ValidTypes = new[] { "at+jwt" };
    });
```

Default 5 minutes

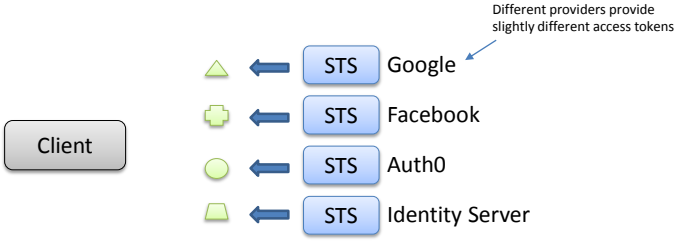
Access token

```
{
  "alg": "RS256",
  "typ": "at+jwt"
}
{
  "iss": "https://localhost:6001",
  "aud": "payment",
  "sub": "818727",
   ...
}
```

What is at+jwt? 

JWT access tokens

The OAuth specification for the **access token** does not specify how it should be constructed



This can cause interoperability issues when we pass tokens between systems

How can we solve this?

JWT access tokens

To help us fix this, a new standard was introduced:

The **typ** header parameter for a JWT access token MUST be **at+jwt**

This is supported by IdentityServer

```
{
  "typ": "at+JWT",
  "alg": "RS256",
  "kid": "RjEwOwOA"
}
{
 "iss": "https://authorization-server.example.com/",
 "sub": " 5ba552d67",
 "aud":   "https://rs.example.com/",
 "exp": 1544645174,
 "client_id": "s6BhdRkqt3_",
 "scope": "openid profile reademail"
}
```

JWT access tokens are regular JWT tokens complying with the requirements described in the specification

JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens
https://tools.ietf.org/html/draft-ietf-oauth-access-token-jwt-04

Additional options

Adding and Configuring JwtBearer

There are plenty of options that we can set:

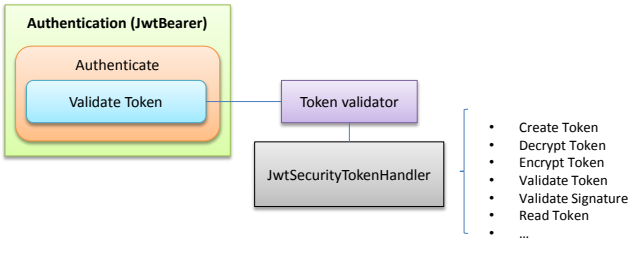
```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
.AddMyJwtBearer(opt =>
{
    opt.Audience = ...;
    opt.Authority = ...;
    opt.AutomaticRefreshInterval = ...;
    opt.BackchannelHttpHandler = ...;
    opt.BackchannelTimeout = TimeSpan.FromMinutes(1);
    opt.Challenge = JwtBearerDefaults.AuthenticationScheme;
    opt.Configuration = ...;
    opt.ConfigurationManager = ...;
    opt.IncludeErrorDetails = true;
    opt.MetadataAddress = ...;
    opt.MapInboundClaims = ...;
    opt.RequireHttpsMetadata = true;
    opt.RefreshInterval = ...;
    opt.RefreshOnIssuerKeyNotFound = true;
    opt.SaveToken = true;
    opt.SecurityTokenValidators = ...;
    opt.TokenValidationParameters = ...;

    opt.Events.OnAuthenticationFailed = ...;
    opt.Events.OnChallenge = ...;
    opt.Events.OnForbidden = ...;
    opt.Events.OnMessageReceived = ...;
    opt.Events.OnTokenValidated = ...;
});
```

However we don’t need to set all of them ☺

SecurityTokenHandlers

SecurityTokenHandler

By default JwtBearer uses **JwtSecurityTokenHandler**



A more optimized version exists named **JsonWebTokenHandler**, but does not work with JwtBearer ☹

Internally asp.net takes a hard dependency on JwtSecurityToken.

Both libraries can be used to manually work with tokens

JwtSecurityToken vs JsonWebToken

JsonWebToken advantages:

- Does not depend of 3rd party libraries
- Faster and less complicated
- Major applications inside Microsoft have switched to using JsonWebToken / JsonWebTokenHandler
- Does not do the annoying claims mapping
- Microsoft.IdentityModel.JsonWebTokens Nuget package

Useful for manual token work!

https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet

JwtSecurityToken vs JsonWebToken

Microsoft says:

*At the current moment, it isn't possible to plug in the **JsonWebTokenHandler** instead of the **JwtSecurityTokenHandler**.*

We plan on adding support for this feature in the future.

***JsonWebTokens** library is an updated, more performant, and easier to use version of **Jwt** that addresses many concerns users had about the previous library.*

https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet/issues/1566#issuecomment-742874826

JwtSecurityToken vs JsonWebToken

Duende made the switch in version 5.0.5

5.0.5

leastprivilege released this 3 days ago · 13 commits to main since this release

As part of this release we had 2 issues closed.

enhancements

- #127 Preserve request objects based on request objects URI
- #118 Replace JwtSecurityTokenHandler with JsonWebTokenHandler
  - this introduces a small breaking change. The `RequestObjectValues` collection on `ValidatedAuthorizeRequest` and `AuthorizeContext` is now a `List<Claim>`

https://github.com/DuendeSoftware/IdentityServer/releases/tag/5.0.5

Claims mapping

Claims mapping

We have plenty of options on how to validate our tokens:

Access token

```
{
  "iss": "https://localhost:6001",
  "aud": "payment",
  "admin": "yes",
  "email": "joe@edument.se",
  "name": "joe",
  "roles": [
    "admin",
    "developer",
    "support"
  ],
  "website": "https://edument.se",
  "scope": [
    "payment"
  ]
  ...
}
```

Name is null because it can't find a claim of type: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

Name: Null

InRole("Admin") = true

Our ClaimsPrincipal

Claim Type	Value
iss	https://localhost:6001
aud	payment
client_id	clientcredentialclient
admin	yes
http://schemas.xmlsoap.org /ws/2005/05/identity/claims/emailaddress	joe@edument.se
name	joe
http://schemas.microsoft.com /ws/2008/06/identity/claims/role	admin
http://schemas.microsoft.com /ws/2008/06/identity/claims/role	developer
http://schemas.microsoft.com /ws/2008/06/identity/claims/role	support
website	https://edument.se
…	…

What is going on here?

Claims Transformations

Microsoft has a different opinion on the naming of claims

The default mapping can be found in **ClaimTypeMapping**:

```
shortToLongClaimTypeMapping = new Dictionary<string, string>
{
  { JwtRegisteredClaimNames.Actort, ClaimTypes.Actor },
  { JwtRegisteredClaimNames.Birthdate, ClaimTypes.DateOfBirth },
  ...
  { "adfs1email", "http://schemas.xmlsoap.org/claims/EmailAddress" },
  { "adfs1upn", "http://schemas.xmlsoap.org/claims/UPN" },
  { "amr", "http://schemas.microsoft.com/claims/authnmethodsreferences" },
  { "authmethod", ClaimTypes.AuthenticationMethod },
  { "certeku", "http://schemas.microsoft.com/2012/12/certificatecontext/extension/eku" },
  { "certissuer", "http://schemas.microsoft.com/2012/12/certificatecontext/field/issuer" },
  ...
```

Where is this mapping used?

https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet/blob/master/src/System.IdentityModel.Tokens.Jwt/ClaimTypeMapping.cs

Claims Transformations

This map is used by the **JwtSecurityTokenHandler** class

It is common to clear this mapping in **Startup.cs**

```
public void ConfigureServices(IServiceCollection services)
{
    // By default, Microsoft has some legacy claim mapping that converts
    // standard JWT claims into proprietary ones. This removes those mappings.
    JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
    JwtSecurityTokenHandler.DefaultOutboundClaimTypeMap.Clear();


    // Or set this flag to false
    .AddJwtBearer(opt =>
    {
        ...
        opt.MapInboundClaims = false;
    });
```

What happens if we do this?

https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet/blob/master/src/System.IdentityModel.Tokens.Jwt/JwtSecurityTokenHandler.cs

Claims mapping

After clearing the type mapping

Name: Null
InRole("Admin") = false

The following claim can not be found anymore:
http://schemas.microsoft.com/ws/2008/06/identity/claims/role

```
{
  "iss": "https://localhost:6001",
  "aud": "payment",
  "admin": "yes",
  "email": "joe@edument.se",
  "name": "joe",
  "roles": [
    "admin",
    "developer",
    "support"
  ],
  "website": "https://edument.se",
  "scope": [
    "payment"
  ]
}
```

Claim Type	Value
iss	https://localhost:6001
aud	payment
client_id	clientcredentialclient
admin	yes
email	joe@edument.se
name	joe
roles	admin
roles	developer
roles	support
website	https://edument.se
…	…

The claim type are no longer renamed

How can we fix the name and role functionality?

Claims mapping

We can tell what the name of the name/roles claim is:

```
.AddMyJwtBearer(opt =>
    {
        ...
        opt.TokenValidationParameters.RoleClaimType = "roles";
        opt.TokenValidationParameters.NameClaimType = "name";
```

Adding this will result make it work as expected:

```
{
  "iss": "https://localhost:6001",
  "aud": "payment",
  "admin": "yes",
  "email": "joe@edument.se",
  "name": "joe",
  "roles": [
    "admin",
    "developer",
    "support"
  ],
  "website": "https://edument.se",
  "scope": [
    "payment"
  ]
}
```

```
[HttpGet]
public IEnumerable<WeatherForecast> Get()
{
    var isAdmin = User.IsInRole("admin");         //true
    var isDeveloper = User.IsInRole("developer"); //true
    var isSupport = User.IsInRole("support");     //true
    var isFinance = User.IsInRole("finance");     //false
    var isSales = User.IsInRole("sales");         //false

    string name = User.Identity.Name;             //joe
}
```

Exercise

Let’s do **main** exercise #15.x

Securing API's – Advanced

Module #16

Error handling

```
/// <summary>
/// Defines whether the token validation errors should be returned to the caller.
/// Default set to true.
/// </summary>
public bool IncludeErrorDetails { get; set; } = true;
```

What does this flag do? 

Error handling

It controls the **WWW-Authenticate** header:

```
opt.IncludeErrorDetails = true;
```

```
HTTP/1.1 401 Unauthorized
Date: Sun, 02 Aug 2020 11:19:06 GMT
WWW-Authenticate: Bearer error="invalid_token", error_description="The signature is invalid"
```

```
opt.IncludeErrorDetails = false;
```

```
HTTP/1.1 401 Unauthorized
Date: Sun, 02 Aug 2020 11:19:06 GMT
WWW-Authenticate: Bearer
```

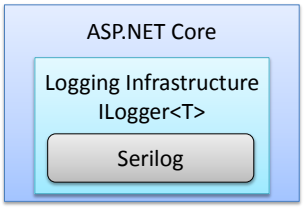
In the OAuth 2.0 specification it says:

> If the protected resource request does not include authentication credentials or does not contain an access token that enables access to the protected resource, the resource server MUST include the HTTP "WWW-Authenticate" response header field;
> https://tools.ietf.org/html/rfc6750#section-3

Advanced logging

Advanced logging

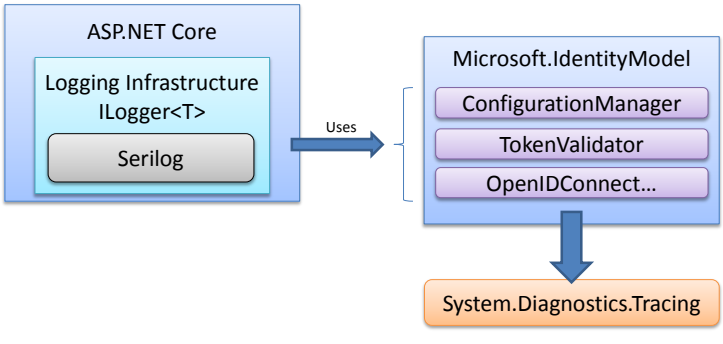
ASP.NET Core contains a very rich logging infrastructure:



So far so good, but will this capture all events?

Advanced logging

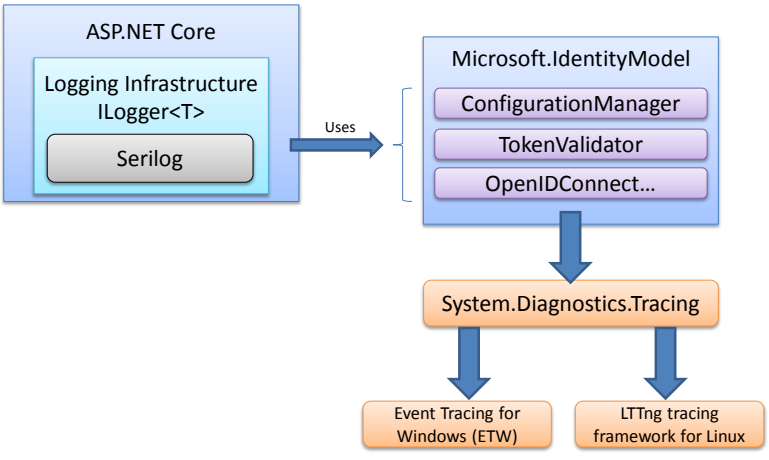
No, some sends their events to **System.Diagnostics.Tracing**



What is System.Diagnostics.Tracing?

System.Diagnostics.Tracing

This is a cross platform diagnostic and event infrastructure

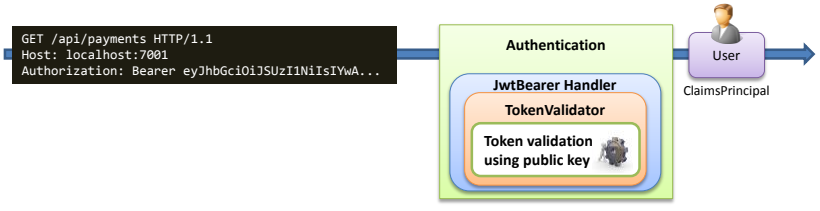


We will explore this in the exercises

Optimizing performance

Optimizing performance

For each request, we will validate the token signature

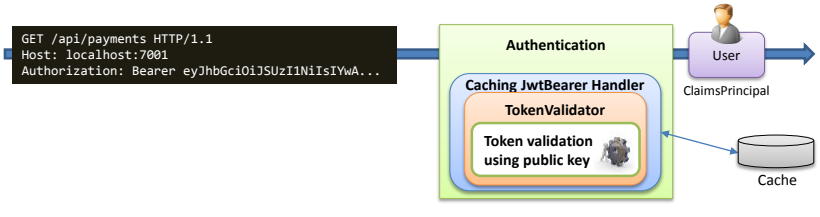


For high-performance API's this is quite expensive

How can we improve this?

Optimizing performance

Why not remember the user for tokens we have seen?



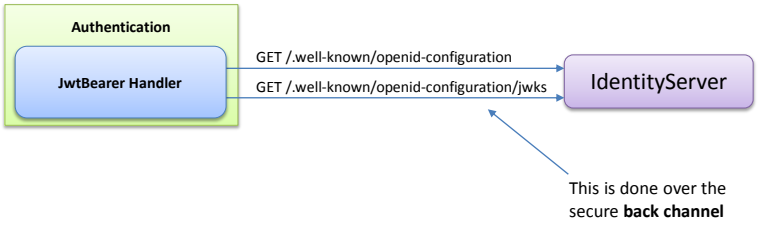
This allows us to only do token validation for new tokens

Demonstration

Logging the back channel requests

Logging the back channel requests

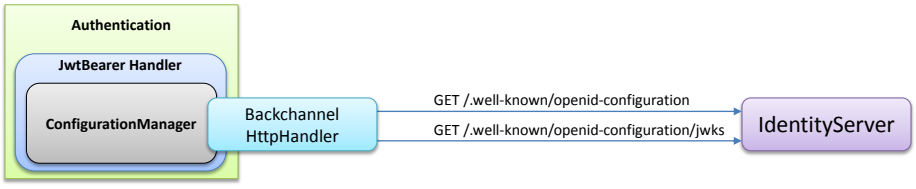
JwtBearer will load the configuration from IdentityServer



How can we log these requests?

Logging the back channel requests

By default, these requests are not logged!



To solve this we can insert our own **HttpMessageHandler**

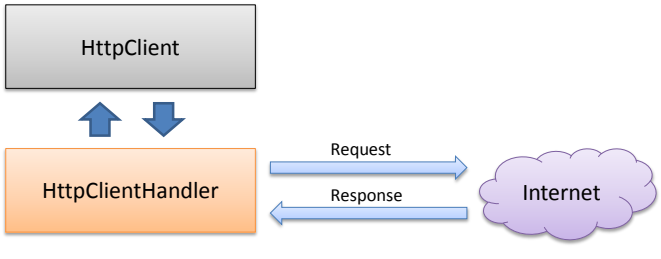
```
.AddJwtBearer(opt =>
{
    opt.BackchannelHttpHandler = new JwtBearerBackChannelListener();
    opt.BackchannelTimeout = TimeSpan.FromSeconds(60);          //default 60s
    ...
}
```

What is a **HttpMessageHandler?** 

What is a HttpMessageHandler?

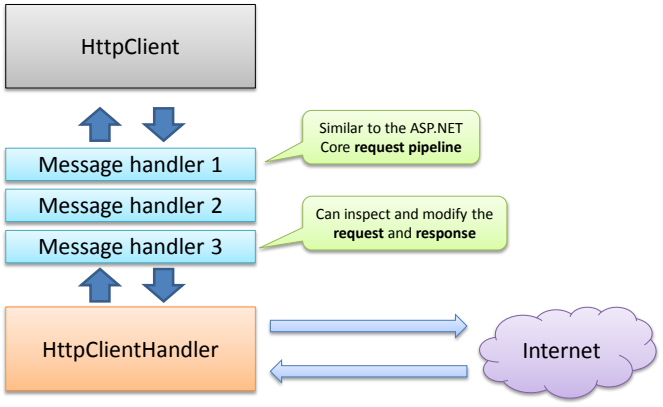
HttpClient does not send any data over the network directly

Instead it delegates this work to a message handler
and the default handler is **HttpClientHandler**



What is a HttpMessageHandler?

We can insert our own **message handlers**

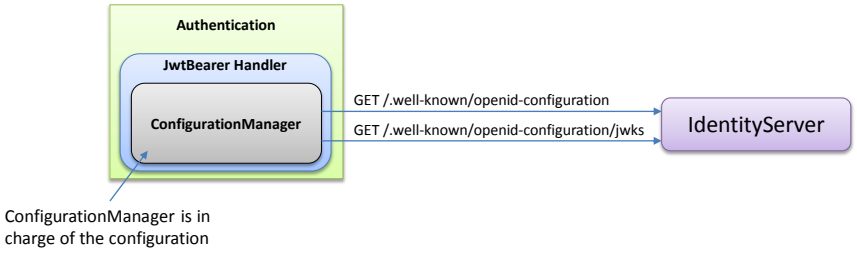


We will explore this in the exercises

Resilience

Resilience

JwtBearer needs to get the configuration from IdentityServer

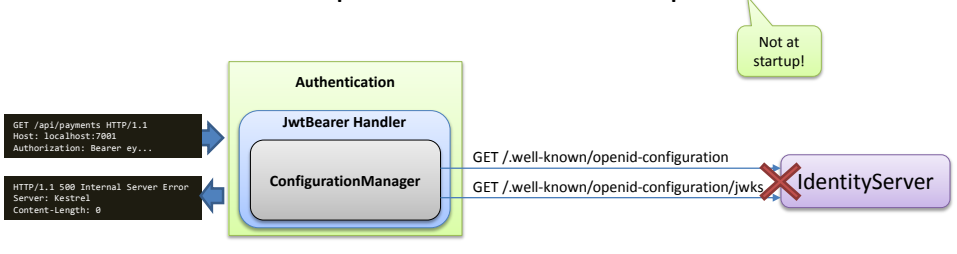


Otherwise tokens can't be accepted!

When is the first time it tries to retrieve the config?

Resilience

The first attempt is when the first request arrives



What happens if this fails?

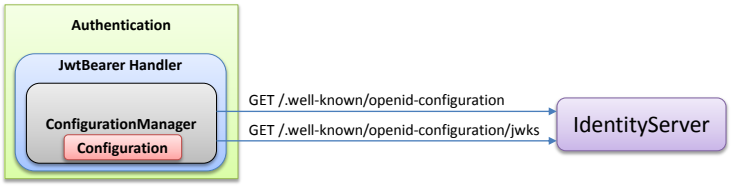
It will try again, but only after 30 seconds

```
.AddMyJwtBearer(opt =>
  {
     opt.RefreshInterval = new TimeSpan(0, 0, 30);
     ...
  }
```

Our API will be unavailable for 30 seconds!

Resilience

If all works, it will get new configuration every **24 hours**



If it then fails, it will reuse existing configuration

```
.AddMyJwtBearer(opt =>
  {
     opt.AutomaticRefreshInterval = new TimeSpan(1, 0, 0, 0);

     opt.BackchannelTimeout = new TimeSpan(0, 0, 10); //10 seconds
...
   }
```

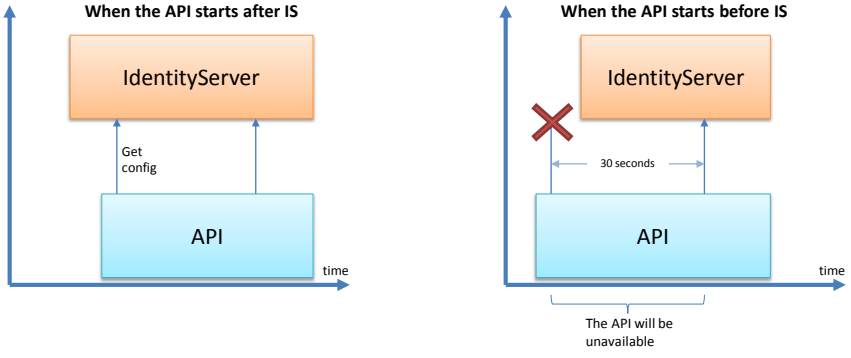
And try again after 30 seconds (RefreshInterval)

The timeout can also be controlled

Improving resilience

Improving resilience

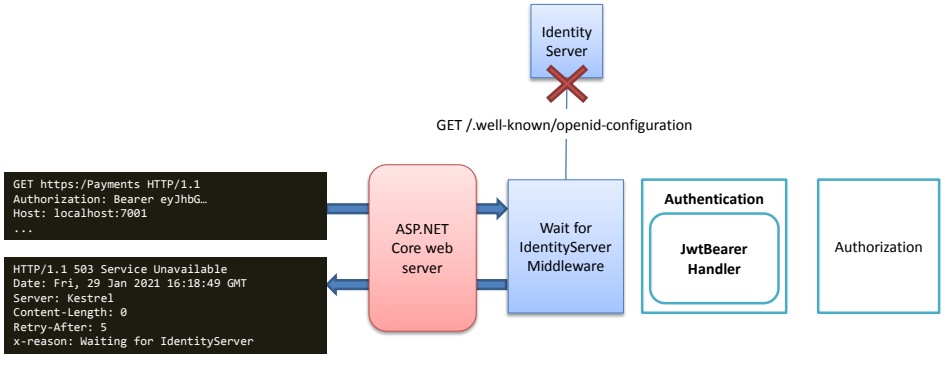
The services startup order matters!



How should we improve this?

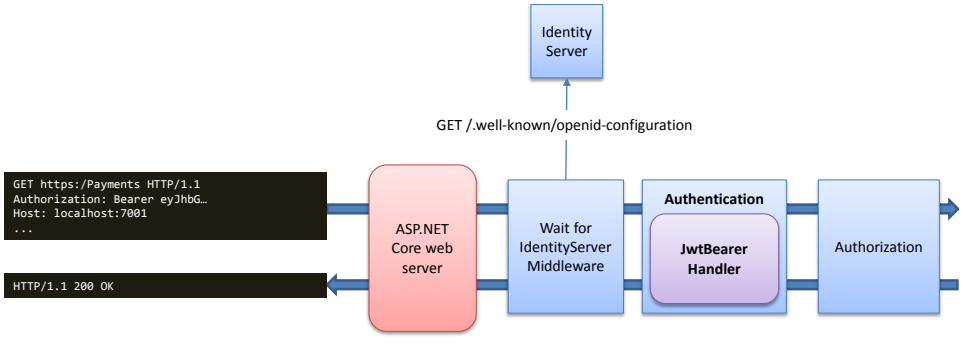
Improving resilience

We can add a filter that blocks the pipeline until IdentityServer responds for the first time



Improving resilience

When it is reachable, then it lets all the traffic through:

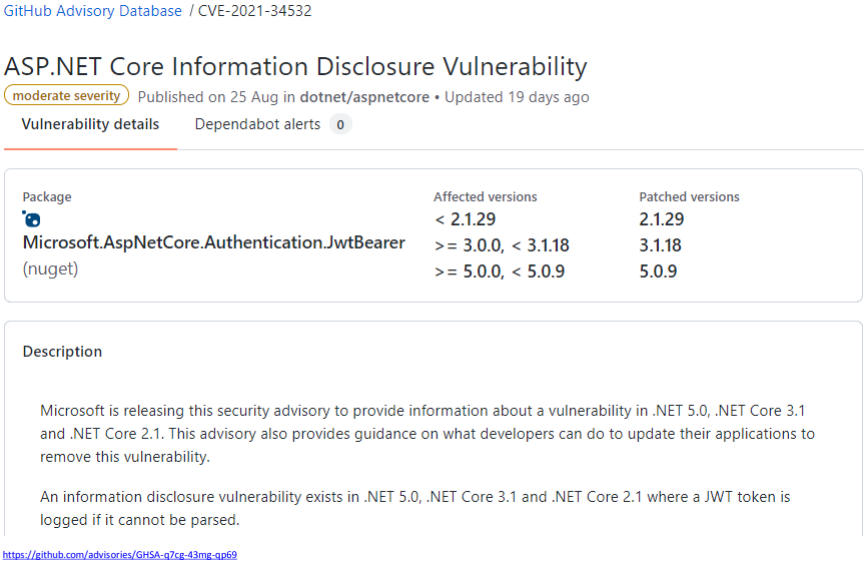


We will explore this in the exercises

Security

Security

Do keep your packages updated!



https://github.com/advisories/GHSA-q7cg-43mg-qp69

Exercise

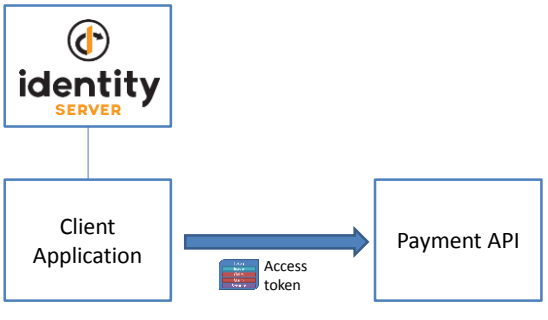
Let’s do main exercise #16.x

Consuming the API

Module #17

Consuming the API

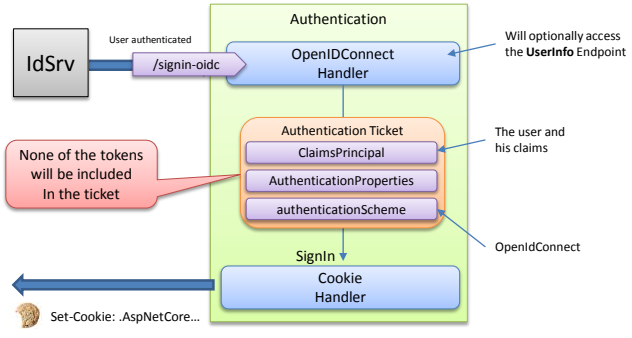
Now let's get the client to call the API using the access token



Where does the client store the tokens?

Where does the client store the tokens?

By default, none of the tokens are saved

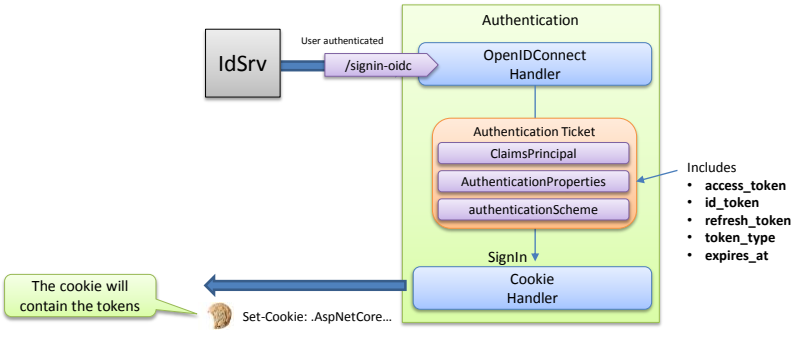


How do we save the tokens?

Where does the client store the tokens?

To keep the tokens, we need to enable **SaveTokens**

```
}).AddOpenIdConnect(options =>
{
    ...
    options.SaveTokens = true;
});
```



How do we access the tokens in code?

Accessing the tokens

The raw tokens can in code be accessed using:

```
string accessToken = await HttpContext.GetTokenAsync("access_token");
string idToken = await HttpContext.GetTokenAsync("id_token");
string refreshToken = await HttpContext.GetTokenAsync("refresh_token");
string tokenType = await HttpContext.GetTokenAsync("token_type");
string accessTokenExpire = await HttpContext.GetTokenAsync("expires_at");
```

Bearer

2021-02-01T10:58:28.0000000+00:00

How can I make a request using these tokens? 

Calling the API manually

Making a request with the token is simple:

```
public async Task<IActionResult> CallApi()
{
    var accessToken = await HttpContext.GetTokenAsync("access_token");

    var client = new HttpClient();

    var authheader = new AuthenticationHeaderValue("Bearer", accessToken);
    client.DefaultRequestHeaders.Authorization = authheader;

    var content = await client.GetStringAsync("https://localhost:7001/api/payment");

    ViewBag.Json = JObject.Parse(content).ToString();
    return View();
}
```

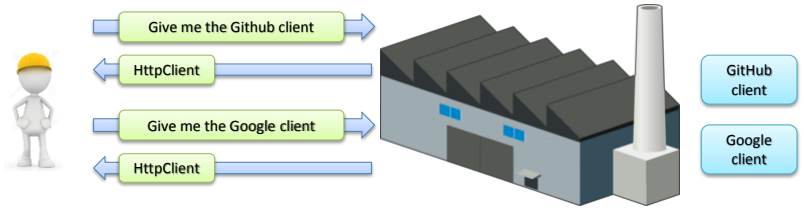
Setting the token can simplified even more:

```
var client = new HttpClient();
client.SetBearerToken(accessToken);
```

Part of the IdentityModel helper library

https://github.com/IdentityModel/IdentityModel

HttpClientFactory



HttpClientFactory

First we can create a named **HttpClient**

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHttpClient("myapi", client =>
    {
        client.BaseAddress = new Uri("https://localhost:7001/api/sales");
        client.Timeout = TimeSpan.FromSeconds(10);
        client.DefaultRequestHeaders.Add("Accept", "application/json");
    });

    services.AddControllersWithViews();
    ...
}
```

Do this before you add MVC

Provides a central location for naming and configuring HttpClients.

HttpClientFactory

Then we can request them in our controllers:

```
public class HomeController : Controller
{
    private readonly IHttpClientFactory _httpClientFactory;

    public HomeController(IHttpClientFactory clientFactory)
    {
        _httpClientFactory = clientFactory;
    }

    public async Task<IActionResult> Index()
    {
        var accessToken = await HttpContext.GetTokenAsync("access_token");

        var client = _httpClientFactory.CreateClient("myapi");

        client.SetBearerToken(accessToken);

        var content = await client.GetStringAsync("");

        ViewBag.Json = JObject.Parse(content).ToString();
        return View();
    }
```

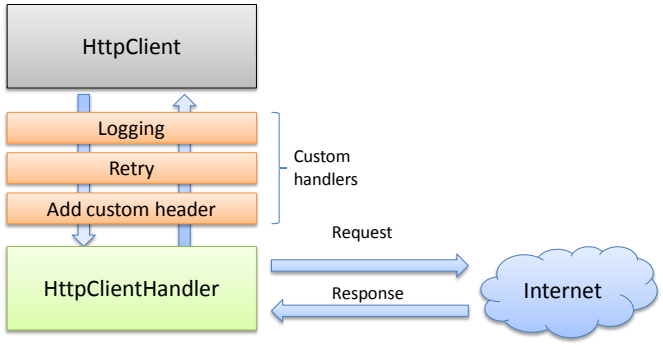
We inject the HttpClientFactory

A new HttpClient instance is returned each time

Improving HttpClient

Improving HttpClient

We saw earlier that we can add our own handlers:



How can we write our own handlers?

Message handlers

We inherit from the **DelegatingHandler** base class:

```
public class AddCustomHeaderHandler : DelegatingHandler
{
    protected async override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
                                                                 CancellationToken cancellationToken)
    {
        //Do something with the request
        request.Headers.Add("CustomHeader", "This is a custom header");

        var response = await base.SendAsync(request, cancellationToken);

        //Do something with the response
        if(response.StatusCode == System.Net.HttpStatusCode.BadRequest)
        { //ohh noo!!
        }
        return response;
    }
}
```

Call the next message handler

We then register it:

Register the type in the DI-container

```
services.AddTransient<AddCustomHeaderHandler>();

services.AddHttpClient("paymentapi", client =>
{
    client.BaseAddress = new Uri("https://localhost:7001/api/payment");
    client.Timeout = TimeSpan.FromSeconds(10);
}).AddHttpMessageHandler<AddCustomHeaderHandler>();
```

Resilience using Polly

Resilience using Polly

Using **Polly** we can make the HttpClient more **resilient**

```
services.AddHttpClient("weatherapi", client =>
{
    client.BaseAddress = new Uri("https://localhost:7001/WeatherForecast");
    client.Timeout = TimeSpan.FromSeconds(10);
})
.AddTransientHttpErrorPolicy(configurePolicy: builder => builder.WaitAndRetryAsync(new[]
        {
                        TimeSpan.FromSeconds(1),
                        TimeSpan.FromSeconds(3),
                        TimeSpan.FromSeconds(5),
                        TimeSpan.FromSeconds(8),
                        TimeSpan.FromSeconds(10)
        },
        onRetry: (outcome, timespan, retryAttempt, context) =>
        {
            Log.Logger.Error("Delaying for {delay}ms, then making retry {retry}.",
                        timespan.TotalMilliseconds, retryAttempt);
        }
    ));
```

Microsoft.Extensions.Http.Polly NuGet package

http://www.thepollyproject.org

Demonstration

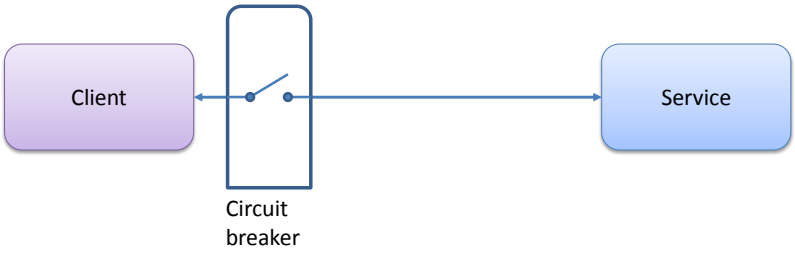
Circuit Breaker pattern

"Coding advice for the day/night: Code defensively, act like every piece of code you don't control will fail (maybe even code you do control)" – David Fowler

Circuit Breaker pattern

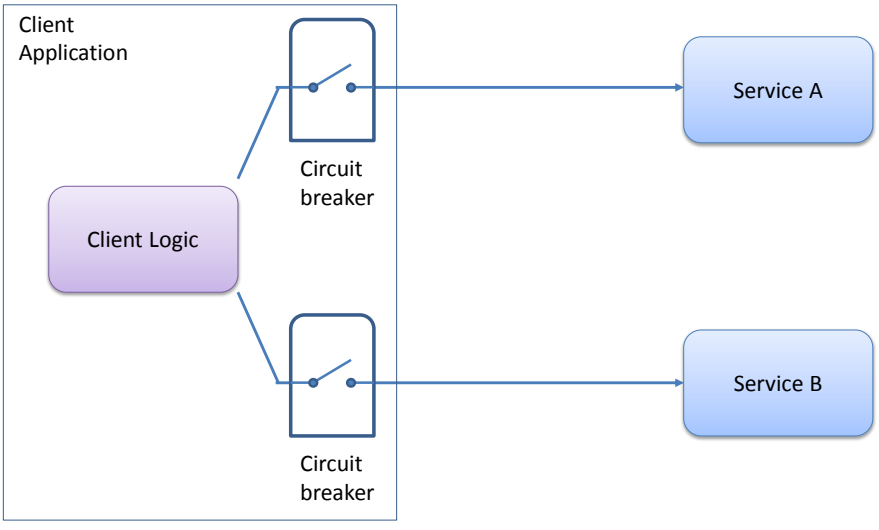
The Circuit Breaker Pattern was first popularized in the book **Release It!** by Michael Nygard

The circuit breaker will break when the number of failures in a given time exceeds a given threshold.



Circuit Breaker pattern

In our demonstration we have the following setup:



Circuit Breaker pattern - resources

- CircuitBreaker
  http://martinfowler.com/bliki/CircuitBreaker.html
- Making the Netflix API More Resilient
  https://medium.com/netflix-techblog/making-the-netflix-api-more-resilient-a8ec62159c2d
- Circuit Breaker Pattern
  https://msdn.microsoft.com/en-us/library/dn589784.aspx
- Polly - resilience framework for .NET
  http://www.thepollyproject.org/

Release It! Second Edition
Design and Deploy Production-Ready Software
ISBN: 978-1-68050-239-8

Exercise

Let’s do main exercise #17.x

Refresh tokens

Module #18

Token lifetime

Eventually our tokens will expire:

Authorization code

```
0WsXtsAY0xR7u5-HGDB93URRR_Pzj-rg2qtycvBJq2Q
```

5 minutes

ID-token

```
{
  "nbf": 1590053955,
  "exp": 1590054255,
  "iss": "https://identityservice.local:6001",
  ...
}
```

5 minutes

Access-token

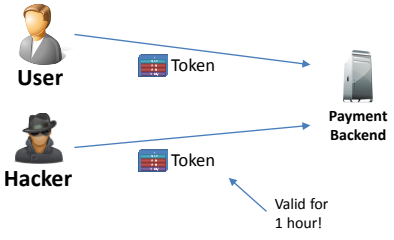
```
{
  "nbf": 1590053955,
  "exp": 1590057555,
  "iss": "https://identityservice.local:6001",
  ...
}
```

60 minutes

Do we want to have **short** or **long-lived** tokens? 

Long-lived access tokens

Having **long-lived** access-tokens means that we will have problem revoking stolen token



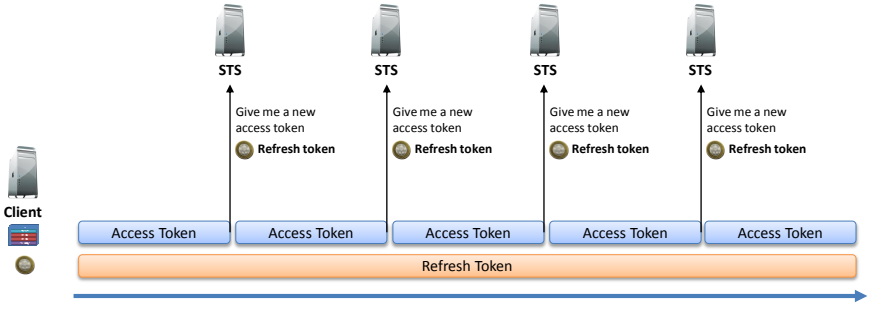
At the same time we don't want the user to have to login every hour.

To solve this we use a **refresh token**

Refresh token

Refresh token

We use the **refresh token** to get new **access tokens**

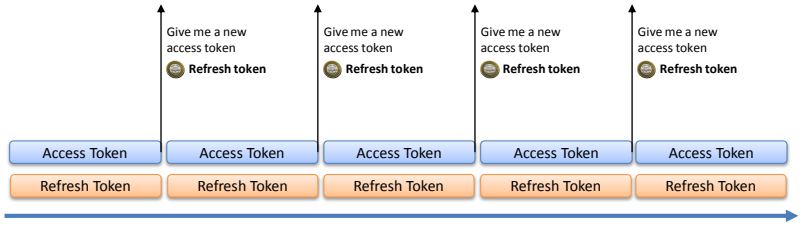


Typically, refresh tokens will be **long-lived** while the access token is **short-lived**.

Refresh token

The default is to use **one-time refresh-tokens**

Each time we use a refresh token we get a new one

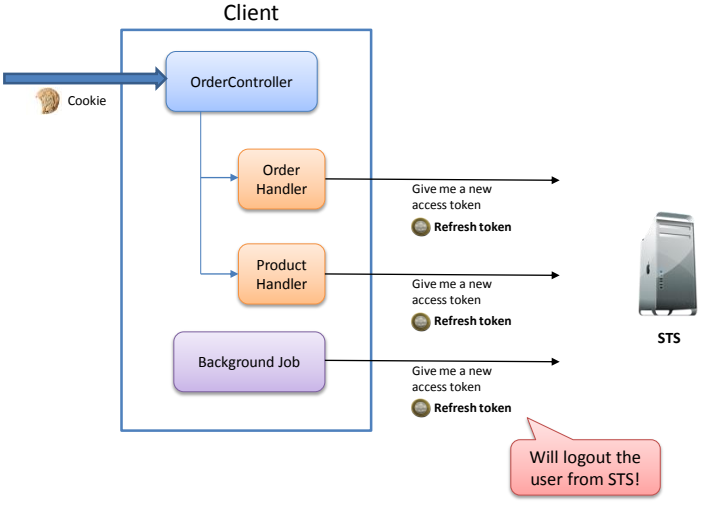


The STS will also prevent reuse of refresh tokens

One-time refresh token gotcha

One-time refresh token gotcha

Watch out for race conditions!



Adding refresh token support in IdentityServer

The offline_access scope

First we need to add support for the **offline_access** scope:

```
AllowedScopes =
    {
        IdentityServerConstants.StandardScopes.OpenId,
        IdentityServerConstants.StandardScopes.Email,
        IdentityServerConstants.StandardScopes.Profile,
        IdentityServerConstants.StandardScopes.OfflineAccess,
    },

AllowOfflineAccess = true,
```

The client definition in IdentityServer

Actually, this one is optional, setting the flag to true is enough.

Then we need to decide if we want to reuse tokens or not:

```
RefreshTokenUsage = TokenUsage.OneTimeOnly
```

A new refresh token is issued after each use **(default)**

```
RefreshTokenUsage = TokenUsage.Reuse;
```

Don't issue new refresh tokens after each use

The offline_access scope

Additional settings that you can control are:

```
AbsoluteRefreshTokenLifetime = 2592000,  //30 days
```

Maximum lifetime of a refresh token in seconds.

```
SlidingRefreshTokenLifetime = 1296000,   //15 days
```

```
UpdateAccessTokenClaimsOnRefresh = true,
```

When asking for a new access token, should the claims be updated?

```
RefreshTokenExpiration = TokenExpiration.Absolute or
                         TokenExpiration.Sliding
```

The token expiration mode

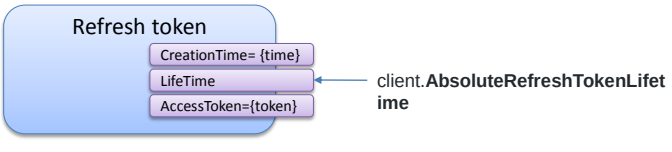
What does the token expiration mode do? 

Token expiration mode

The mode controls the max lifetime of the refresh token

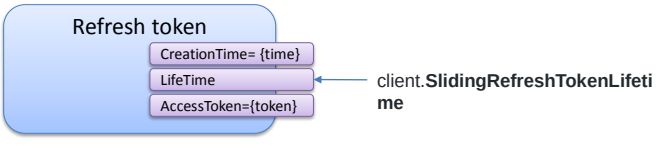
Absolute:

`TokenExpiration.Absolute`



Sliding

`TokenExpiration.Sliding`

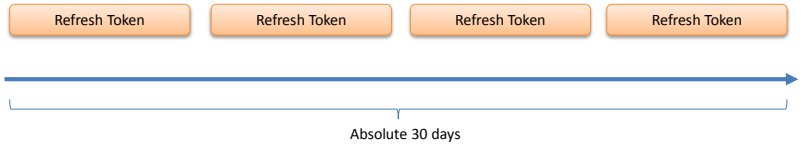


Absolute refresh tokens

Absolute token expiration

The refresh token can be used up to 30 days

```
AbsoluteRefreshTokenLifetime = 2592000,  //30 days
RefreshTokenExpiration = TokenExpiration.Absolute
```



After 30 days, the user must login again

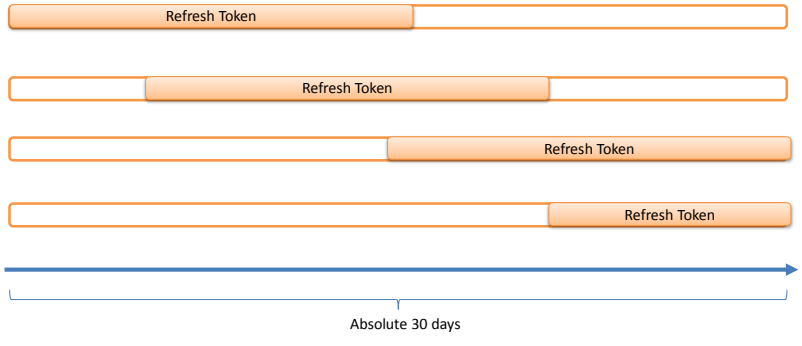
Sliding refresh tokens

Sliding token expiration

With sliding the refresh token must be used within 15 days

```
AbsoluteRefreshTokenLifetime = 2592000,  //30 days
SlidingRefreshTokenLifetime = 1296000,   //15 days

RefreshTokenExpiration = TokenExpiration.Sliding
```

If set to zero, allow indefinite slide



But after 30 days of use, it will expire

Exercise

Let’s do main exercise #18.x

Consuming the API - Advanced

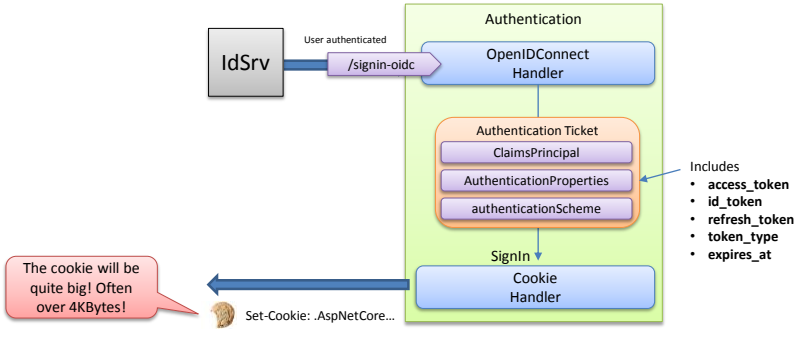
Module #19

Reducing the cookie size

Reducing the cookie size

We have a problem with fat cookies when we save the tokens

```
}).AddOpenIdConnect(options =>
{
    ...
    options.SaveTokens = true;
});
```

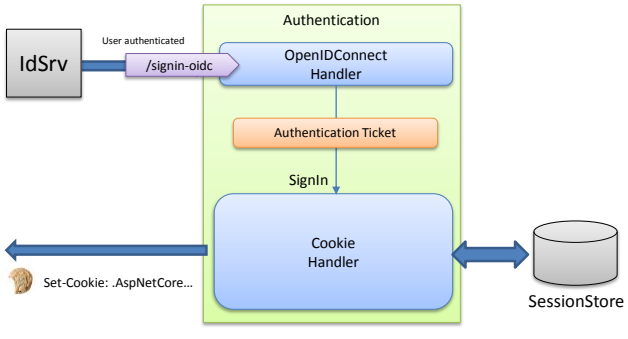


How can we reduce the size of the cookie?

Reducing the cookie size

In the cookie handler we can add a **SessionStore**

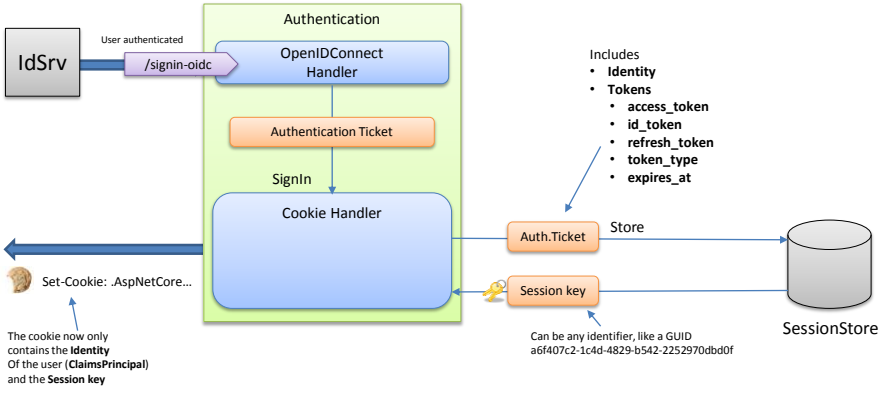
```
}).AddCookie(options =>
{
   ...
   options.SessionStore = new MySessionStore();

})
```



What does the **SessionStore** do?

Reducing the cookie size

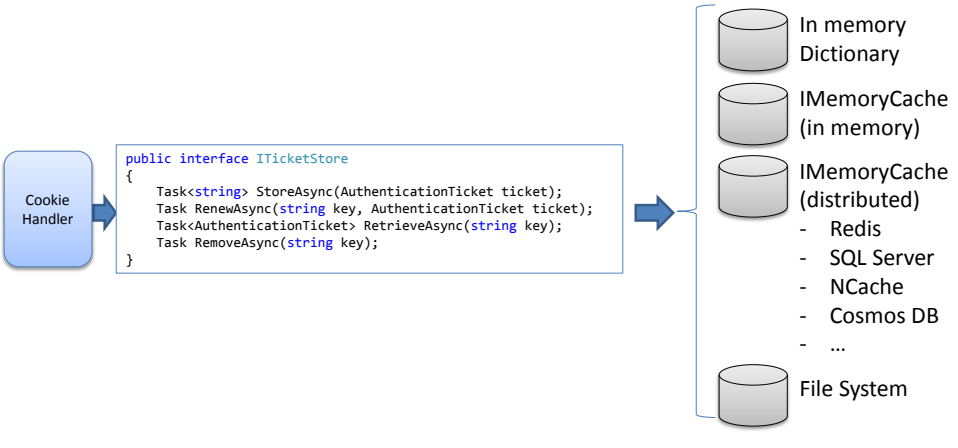
It acts like a **cache** for the tokens



How can we implement a session store?

Reducing the cookie size

All we need is a class that implements **ITicketStore**



We will explore this in the exercises

Are large cookies really a problem?

Large cookies

IT depends!

With HTTP/1.1

Yes, large cookies are sent in every request!

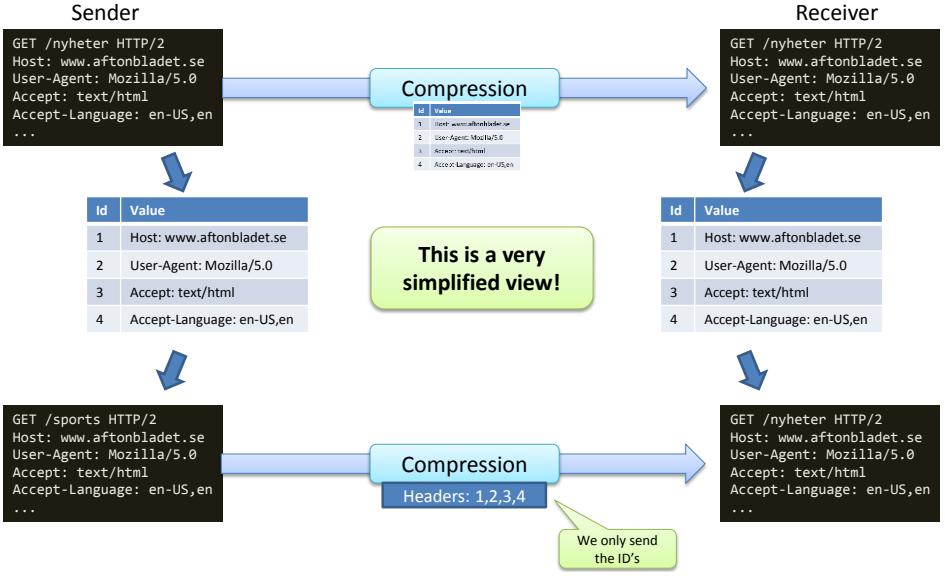
With HTTP/2

Not a major problem!

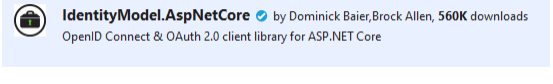
With HTTP/2 he have cookie compressions that only send the header once per connection

Large cookies and HTTP/2

With HTTP/2 we dynamically create lookup tables:



IdentityModel.AspNetCore



IdentityModel.AspNetCore

Helper library for access token lifetime management

It is useful for both **services** and **web** applications

How do I add support for web applications?

https://identitymodel.readthedocs.io/

IdentityModel.AspNetCore

It is simple to add to our application:

```
}).AddMyCookie(options =>
{
    ...
    options.Events.OnSigningOut = async e =>
    {
        // revoke refresh token on sign-out
        await e.HttpContext.RevokeUserRefreshTokenAsync();
    };
})
```

```
services.AddAccessTokenManagement(options =>
{
    //Various options exists
});
```

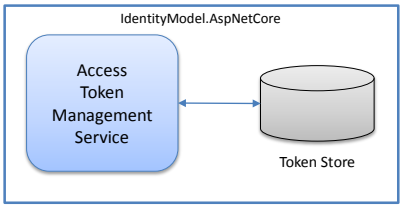
The defaults are fine for most situations

It assumes we use:
- access and refresh tokens
- SaveTokens is true
- We use OpenID Connect

What does it do?

IdentityModel.AspNetCore

Internally, the core consists of:




```
public interface IUserTokenStore
{
    Task StoreTokenAsync(...);
    Task<UserAccessToken> GetTokenAsync(...);
    Task ClearTokenAsync(...);
}
```

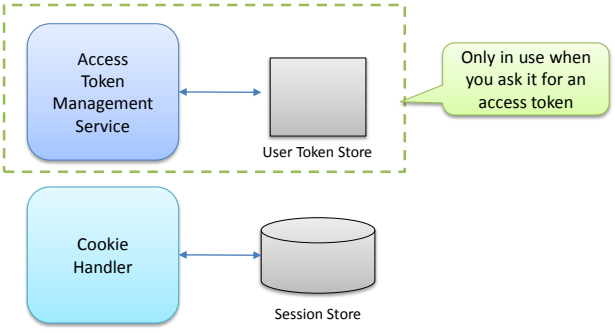
It adds two extension methods for us to use:

```
var token = await HttpContext.GetUserAccessTokenAsync();

HttpContext.RevokeUserRefreshTokenAsync();
```

Session vs **Token** store

Session vs Token store

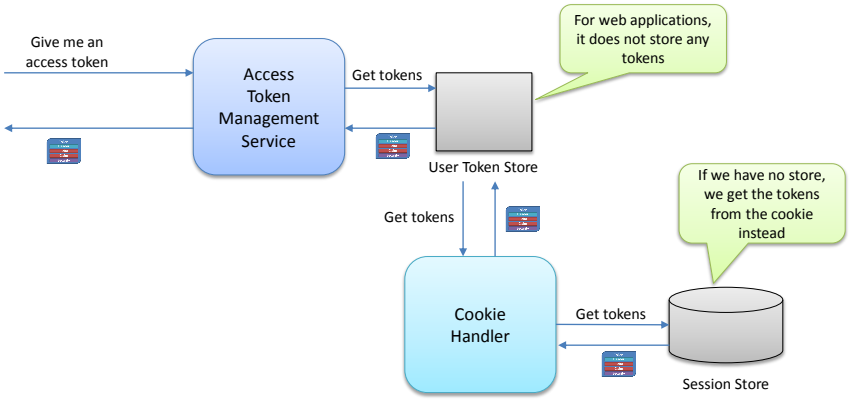
We now have two **token stores** side-by-side in our client:



What is the relationship between these two?

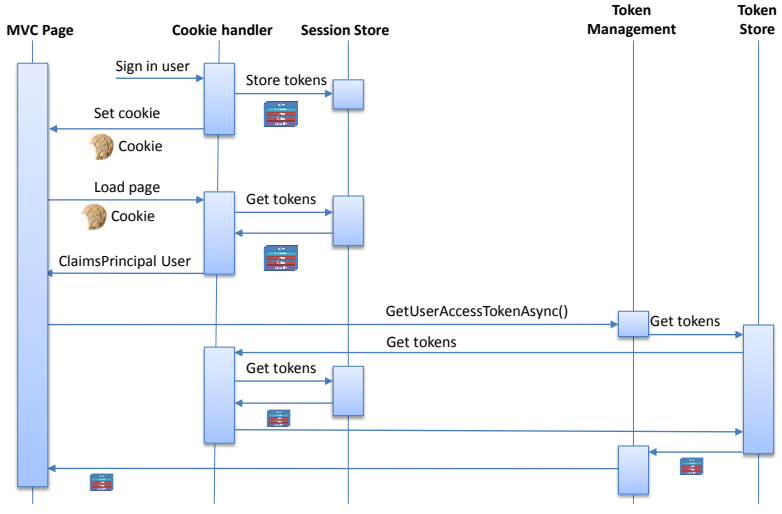
Session vs Token store

When you ask for a token, it will:



Session vs Token store

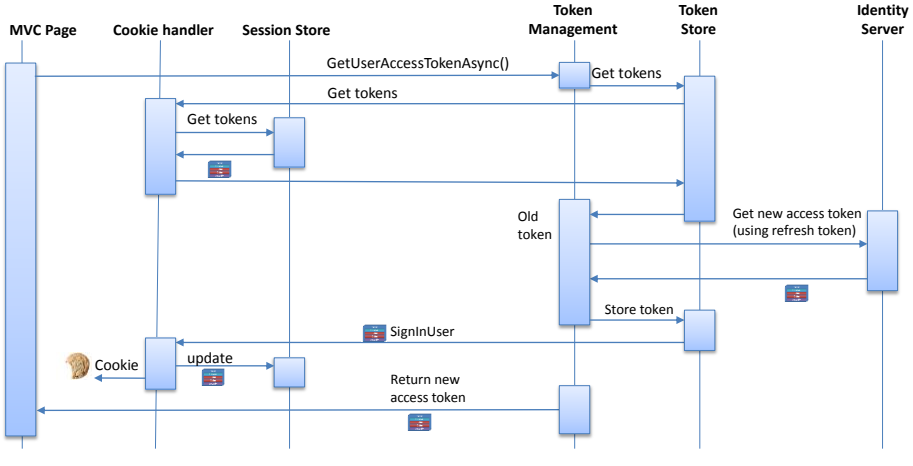
A more detailed view of the flow:



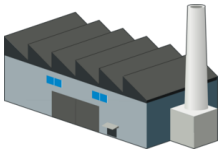
What happens when the access token is about to expire?

Session vs Token store

When the token is about to expire:



Using HttpClientFactory with IdentityModel.AspNetCore



HttpClientFactory

We need to do three things in Startup.cs

1

```
}).AddCookie(options =>
{
    ...
    options.Events.OnSigningOut = async e =>
    {
        // revoke refresh token on sign-out
        await e.HttpContext.RevokeUserRefreshTokenAsync();
    };
})
```

2

```
// adds user and client access token management
services.AddAccessTokenManagement(options =>
{
    options.User.RefreshBeforeExpiration = TimeSpan.FromSeconds(5);
})
.ConfigureBackchannelHttpClient()
.AddTransientHttpErrorPolicy(policy => policy.WaitAndRetryAsync(new[]
        {
            TimeSpan.FromSeconds(1),
            TimeSpan.FromSeconds(2),
            TimeSpan.FromSeconds(3)
        }));
```

HttpClientFactory

We need to do three things in **Startup.cs**

3

```
// registers HTTP client that uses the managed user access token
services.AddUserAccessTokenHttpClient(name: "paymentapi",parameters: null,
                                      configureClient: client =>
{
    client.BaseAddress = new Uri(_configuration["paymentApiUrl"]);
    client.Timeout = TimeSpan.FromSeconds(5);
    client.DefaultRequestHeaders.Add("Accept", "application/json");
});
```

In our controller, we can now ask for a client:

Includes access token

```
HttpClient? client = _httpClientFactory.CreateClient("paymentapi");

var content = await client.GetStringAsync("/payments/get");
```

Exercise

Let’s do main exercise #19.x