SECURITY

Custom Local Domain using HTTPS, Kestrel & ASP.NET Core

LES JACKSON 8TH JUL '20 0

Learn how to create a custom domain, (*not* "localhost"), for your local ASP.NET Core development environment using Kestrel and secure it with HTTPS and a Self-Signed Certificate.

What You'll Learn

This step by step tutorial will teach you:

- Create a self-signed certificate for your custom domain
- Ensure the certificate is a Trusted Root Certificate
- How to ingest the certificate into Kestrel

Note: This article is geared towards a “Windows” environment, so unfortunately Linux & OSX users may feel a little left out.

Ingredients

If you want to follow along you’ll need:

- VS Code (or Visual Studio / another text editor)
- .NET Core 3.1 SDK
- PowerShell
- Postman / Curl / Web Browser

Source Code

The source code for this tutorial can be found on **here on GitHub**.

Usecase

- We want to use HTTPS in our local development environment.

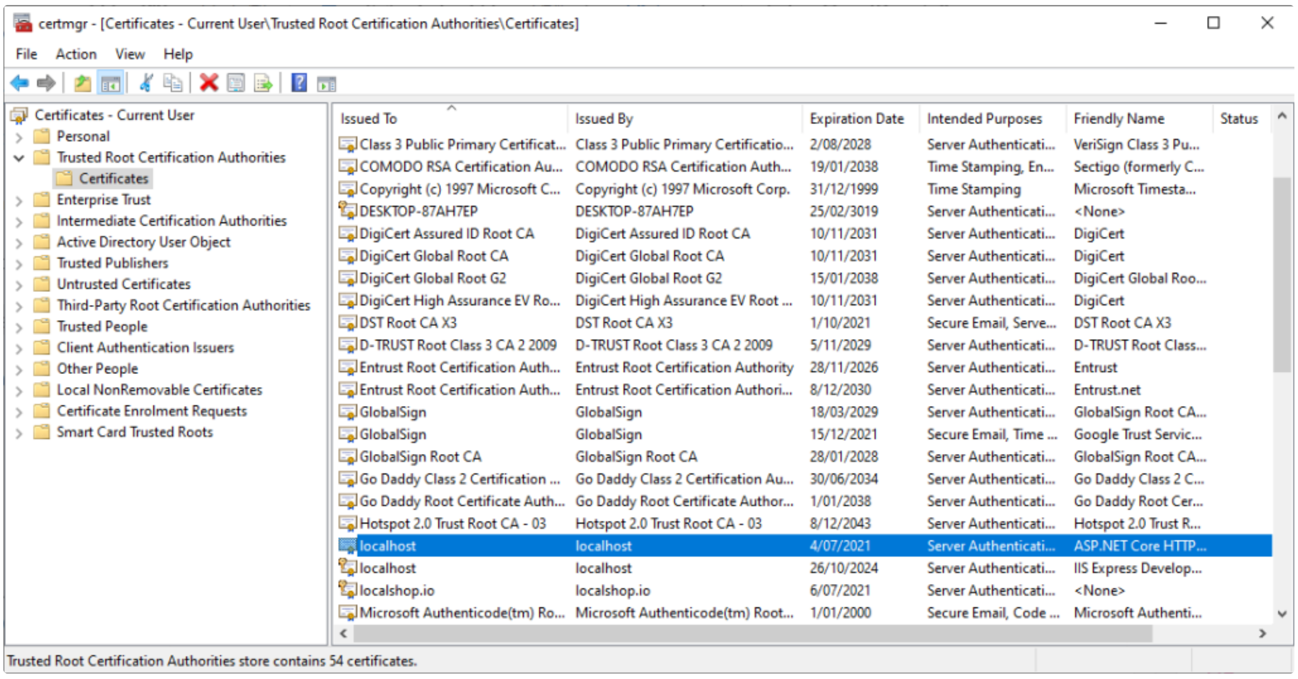
But why?

Dotnet dev-certs

Taking a step back, you can very easily, (and probably already have either directly or indirectly), created a default / standard self-signed localhost development certificate using the *dotnet dev-certs* tool. Running this command:

```
dotnet dev-certs https --trust
```

Will add a self-signed certificate to the Windows as shown below:



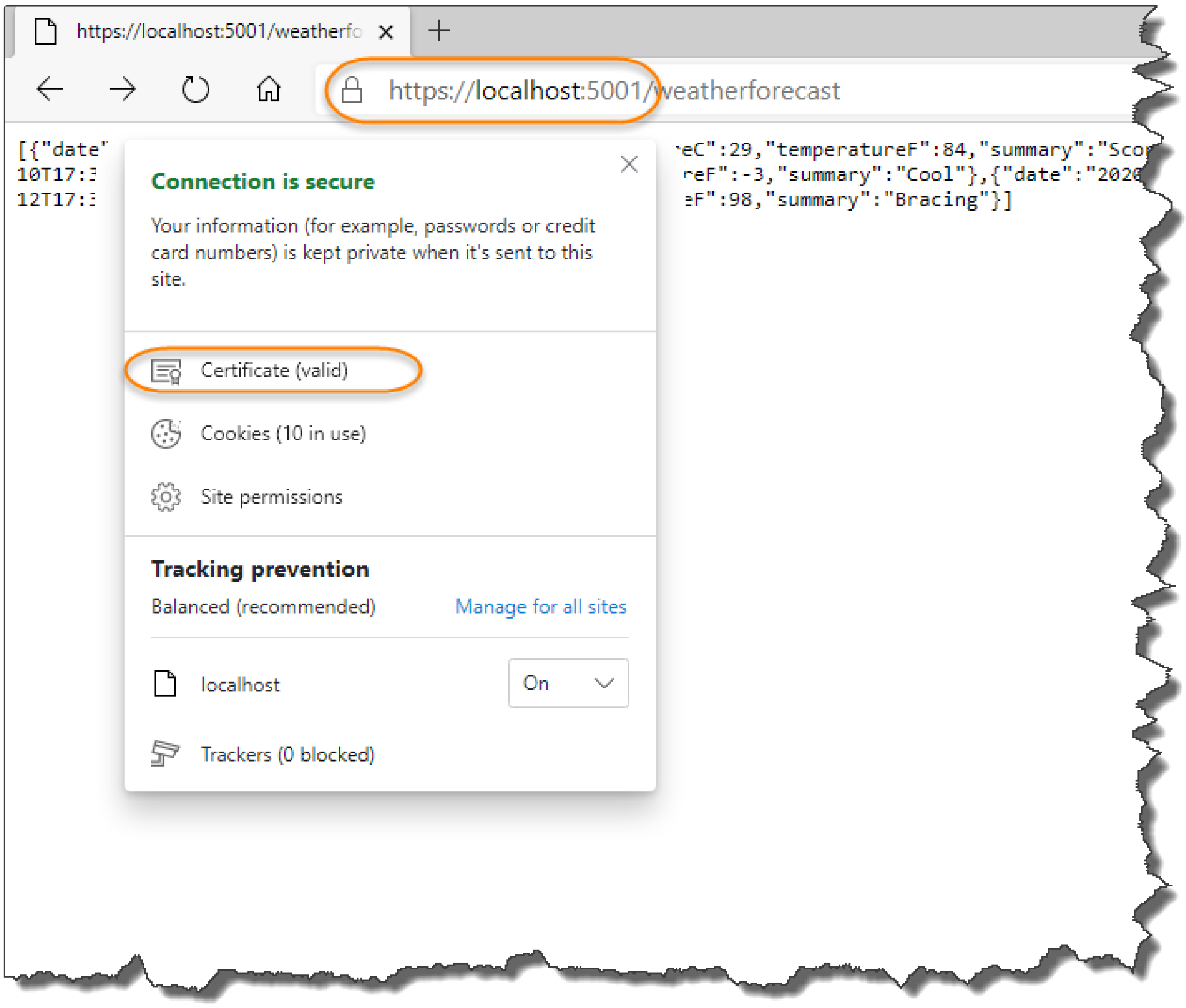
certmgr - [Certificates - Current User\Trusted Root Certification Authorities\Certificates]

- Certificates - Current User
  - Personal
  - Trusted Root Certification Authorities
    - Certificates
  - Enterprise Trust
  - Intermediate Certification Authorities
  - Active Directory User Object
  - Trusted Publishers
  - Untrusted Certificates
  - Third-Party Root Certification Authorities
  - Trusted People
  - Client Authentication Issuers
  - Other People
  - Local NonRemovable Certificates
  - Certificate Enrolment Requests
  - Smart Card Trusted Roots

Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Name	Status
Class 3 Public Primary Certificat...	Class 3 Public Primary Certificatio...	2/08/2028	Server Authenticati...	VeriSign Class 3 Pu...	
COMODO RSA Certification Au...	COMODO RSA Certification Auth...	19/01/2038	Time Stamping, En...	Sectigo (formerly C...	
Copyright (c) 1997 Microsoft C...	Copyright (c) 1997 Microsoft Corp.	31/12/1999	Time Stamping	Microsoft Timesta...	
DESKTOP-87AH7EP	DESKTOP-87AH7EP	25/02/3019	Server Authenticati...	<None>	
DigiCert Assured ID Root CA	DigiCert Assured ID Root CA	10/11/2031	Server Authenticati...	DigiCert	
DigiCert Global Root CA	DigiCert Global Root CA	10/11/2031	Server Authenticati...	DigiCert	
DigiCert Global Root G2	DigiCert Global Root G2	15/01/2038	Server Authenticati...	DigiCert Global Roo...	
DigiCert High Assurance EV Ro...	DigiCert High Assurance EV Root ...	10/11/2031	Server Authenticati...	DigiCert	
DST Root CA X3	DST Root CA X3	1/10/2021	Secure Email, Serve...	DST Root CA X3	
D-TRUST Root Class 3 CA 2 2009	D-TRUST Root Class 3 CA 2 2009	5/11/2029	Server Authenticati...	D-TRUST Root Class...	
Entrust Root Certification Auth...	Entrust Root Certification Authority	28/11/2026	Server Authenticati...	Entrust	
Entrust Root Certification Auth...	Entrust Root Certification Authori...	8/12/2030	Server Authenticati...	Entrust.net	
GlobalSign	GlobalSign	18/03/2029	Server Authenticati...	GlobalSign Root CA...	
GlobalSign	GlobalSign	15/12/2021	Secure Email, Time ...	Google Trust Servic...	
GlobalSign Root CA	GlobalSign Root CA	28/01/2028	Server Authenticati...	GlobalSign Root CA...	
Go Daddy Class 2 Certification ...	Go Daddy Class 2 Certification Au...	30/06/2034	Server Authenticati...	Go Daddy Class 2 C...	
Go Daddy Root Certificate Auth...	Go Daddy Root Certificate Author...	1/01/2038	Server Authenticati...	Go Daddy Root Cer...	
Hotspot 2.0 Trust Root CA - 03	Hotspot 2.0 Trust Root CA - 03	8/12/2043	Server Authenticati...	Hotspot 2.0 Trust R...	
localhost	localhost	4/07/2021	Server Authenticati...	ASP.NET Core HTTP...	
localhost	localhost	26/10/2024	Server Authenticati...	IIS Express Develop...	
localshop.io	localshop.io	6/07/2021	Server Authenticati...	<None>	
Microsoft Authenticode(tm) Ro...	Microsoft Authenticode(tm) Root...	1/01/2000	Secure Email, Code ...	Microsoft Authenti...	

Trusted Root Certification Authorities store contains 54 certificates.

An “out the box” ASP.NET Core application will then use this certificate by default, (well, the Kestrel web server will), and the site will appear secure when browsed to using


https://localhost:5001/weatherfo
https://localhost:5001/weatherforecast

```
[{"date"                                    eC":29,"temperatureF":84,"summary":"Sco
10T17:3                                     reF":-3,"summary":"Cool"},{"date":"2020
12T17:3                                     eF":98,"summary":"Bracing"}]
```

Connection is secure

Your information (for example, passwords or credit card numbers) is kept private when it's sent to this site.

Certificate (valid)

Cookies (10 in use)

Site permissions

Tracking prevention

Balanced (recommended) Manage for all sites

localhost On

Trackers (0 blocked)

If that works for you, (and in most cases it will), then you can stop reading here and get on with your life!

However, there are situations where the use of “localhost” as your Domain name doesn’t cut it, and you need to use a “proper” custom domain. The reason for this is that “localhost” can be treated in a *unique* way by some applications which may mean that your code does not operate as expected, (or more likely the client applications calling your app, don’t work as expected…)

I had this exact issue then preparing another article. The software I was writing about would not route to my .NET Core API that was running on localhost. Moreover as I

Create a Custom Domain

So in order to get a “custom” local domain, I’m just going to update our local Hosts file. The Hosts file in case you’re wondering is just a simple file with IP Address to Host Name entries that we can use for this exact purpose – it’s simple and quick.

Microsoft TCP/IP Host Name Resolution Order

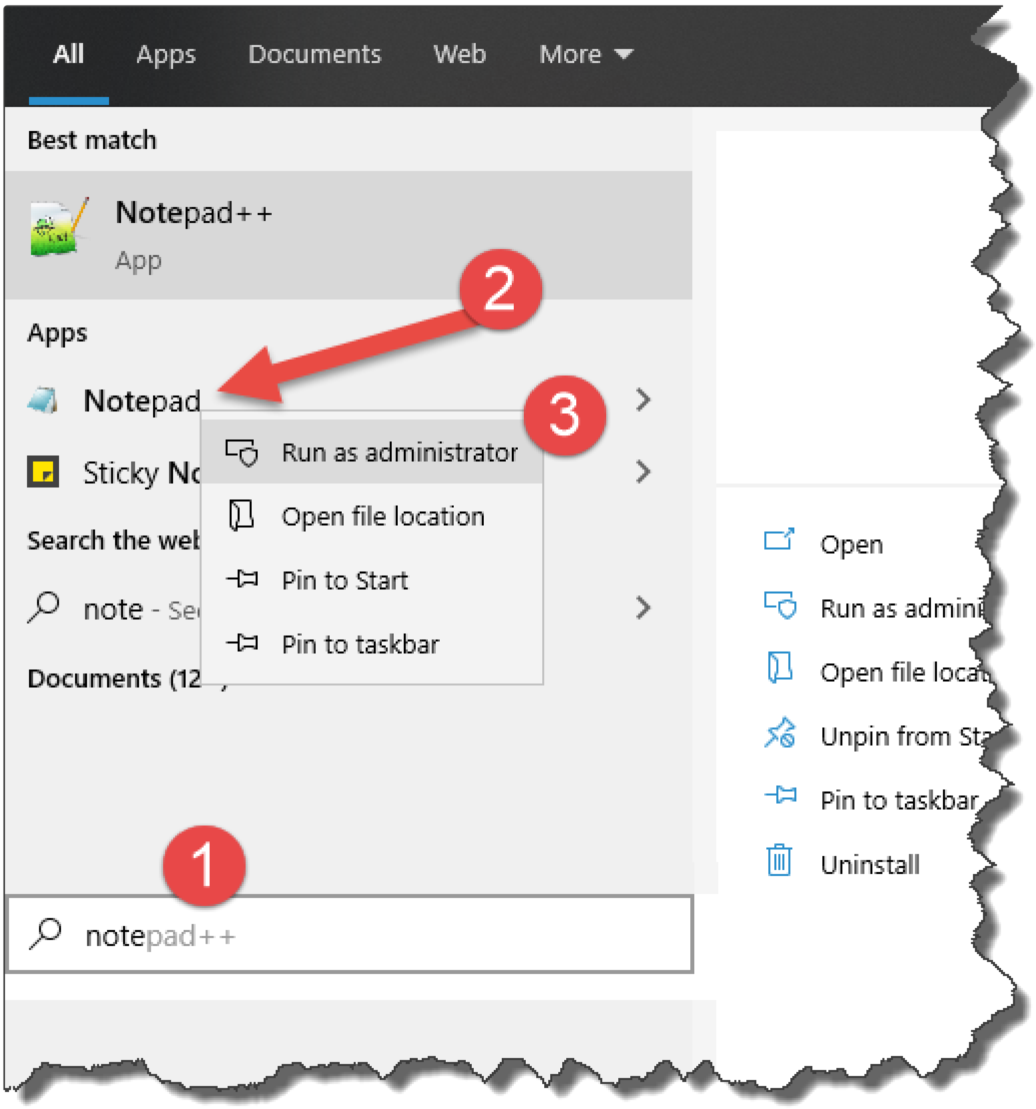
Just before we update the Hosts file it’s worth understanding the order in which “domain names” are resolved to IP addresses. **This article by Microsoft**, explains the Host Name Resolution Order on a Windows PC, basically the order in which your PC will attempt to resolve a Domain Name to an IP Address, are:

1. Check if the name queried is its own
2. Check the Hosts File (that we’re going to update)
3. Domain Name System (DNS) Servers
4. NetBIOS

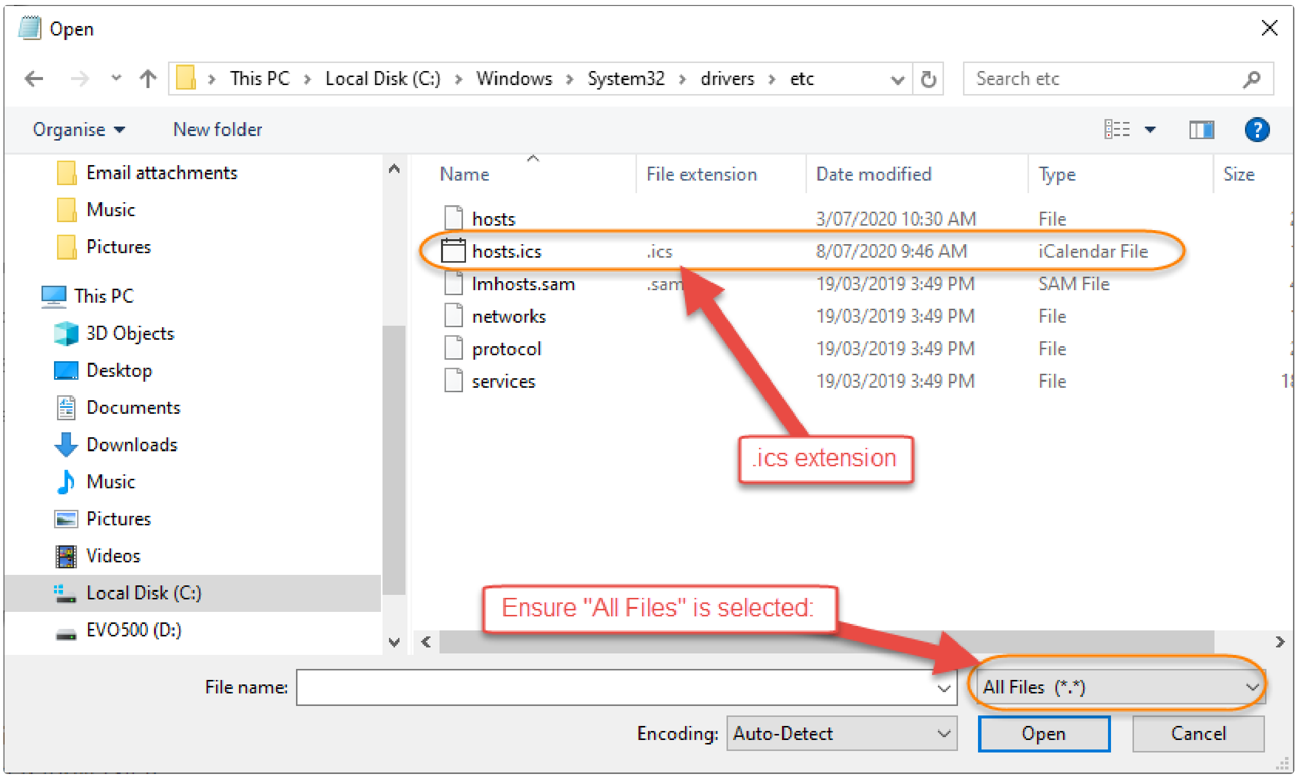
In most cases the domain name will be resolved by the DNS Servers you are using, ordinarily this will be the DNS Servers that your ISP or Company Network have decided to use. You’ll note though that the Hosts file will be queried *before* the DNS servers – so just be aware of that. So we can in effect put any domain name in here and it will be resolved before the DNS server entries, but obviously for our PC only, (don’t worry your not going to end up redirecting Google’s traffic to your laptop).

1. Search for Notepad
2. Right Click
3. Select Run As Administrator

As shown below:

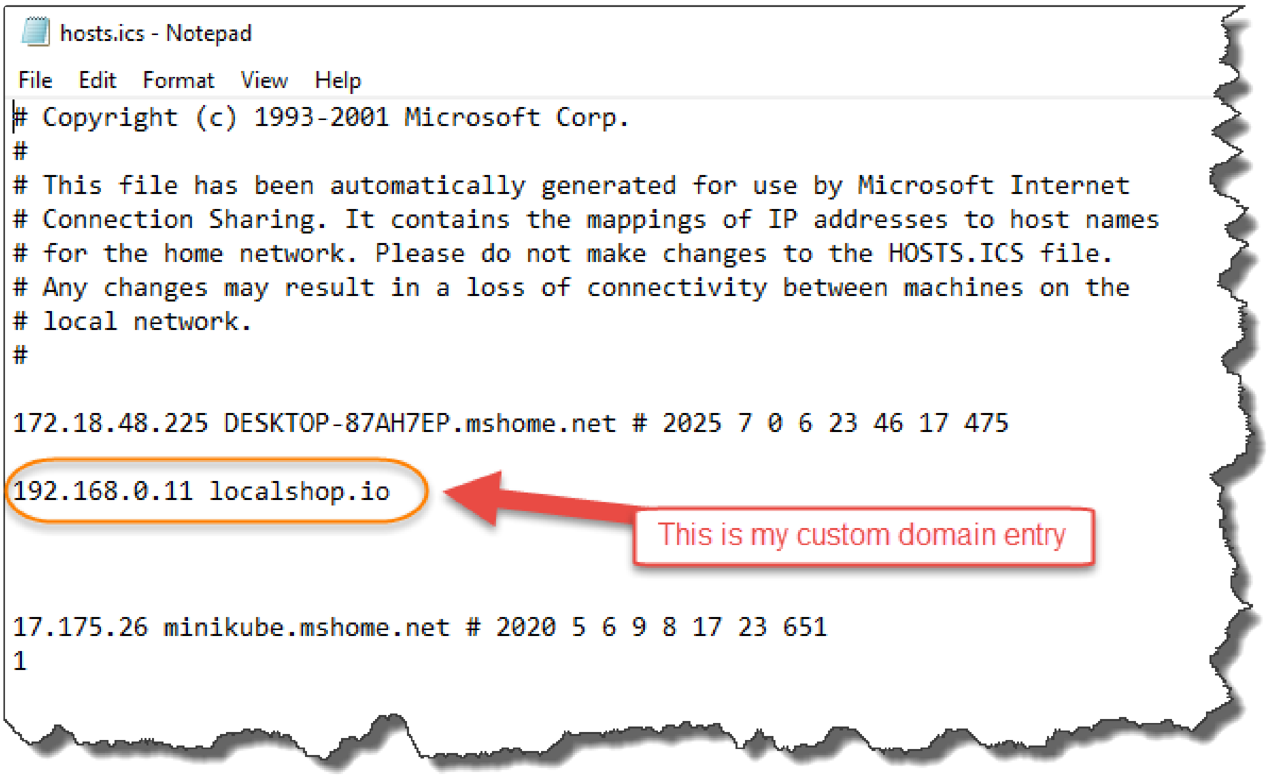


The from Notepad, open the hosts file from the following location, (if your Windows installation is not on the C: drive then swap out): ***C:\Windows\System32\drivers\etc***



Name	File extension	Date modified	Type	Size
hosts		3/07/2020 10:30 AM	File	
hosts.ics	.ics	8/07/2020 9:46 AM	iCalendar File	
lmhosts.sam	.sam	19/03/2019 3:49 PM	SAM File	
networks		19/03/2019 3:49 PM	File	
protocol		19/03/2019 3:49 PM	File	
services		19/03/2019 3:49 PM	File	

Depending on how your PC is set up will depend on the contents of this file, but it should look something like this, (although not exactly for obvious reasons):

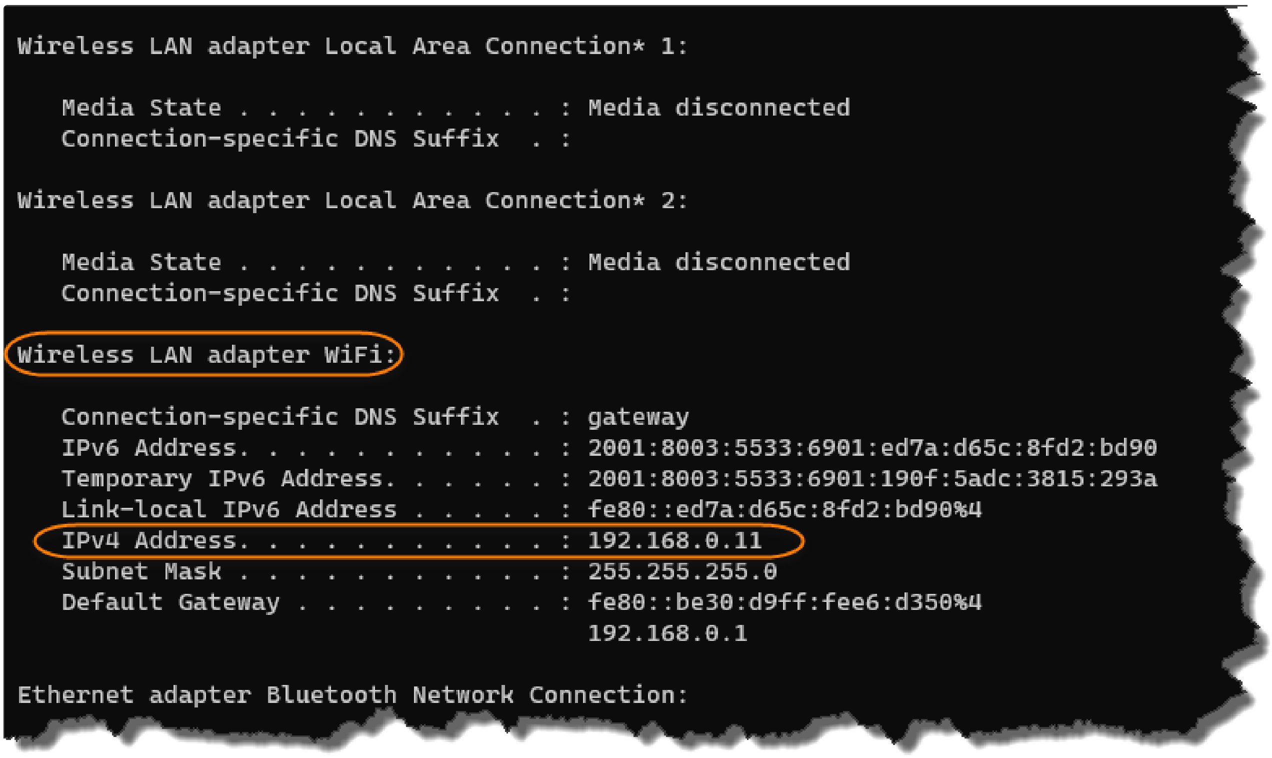


```
# Copyright (c) 1993-2001 Microsoft Corp.
#
# This file has been automatically generated for use by Microsoft Internet
# Connection Sharing. It contains the mappings of IP addresses to host names
# for the home network. Please do not make changes to the HOSTS.ICS file.
# Any changes may result in a loss of connectivity between machines on the
# local network.
#

172.18.48.225 DESKTOP-87AH7EP.mshome.net # 2025 7 0 6 23 46 17 475

192.168.0.11 localshop.io


17.175.26 minikube.mshome.net # 2020 5 6 9 8 17 23 651
1
```

To get your IP Address, open PowerShell or a Command Line and type:

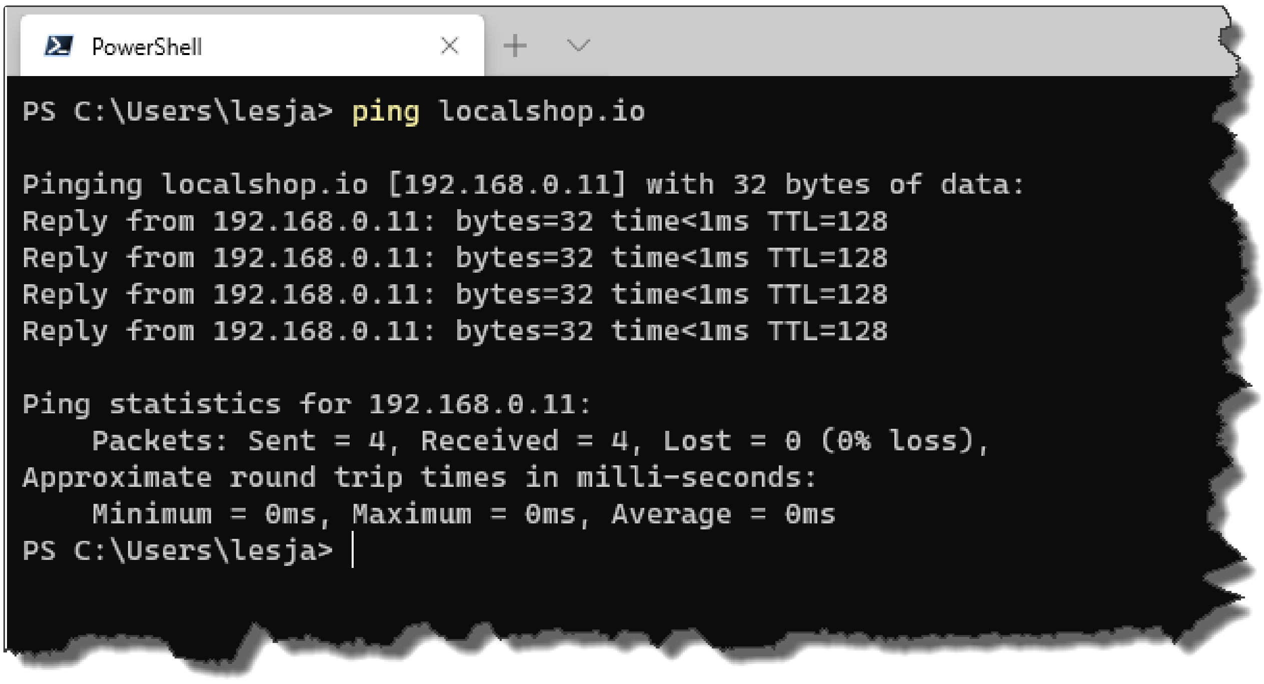
```
ipconfig
```

Again depending on how things have been set up will depend on what results you get back, (you'll probably have a load of "virtual" adapters in addition to the physical adapters you have). As shown below I've selected the IP address, (IPv4), for my physical Wireless Network card that's connecting me to the rest of my home network and the internet:



```
Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter WiFi:

   Connection-specific DNS Suffix  . : gateway
   IPv6 Address. . . . . . . . . . . : 2001:8003:5533:6901:ed7a:d65c:8fd2:bd90
   Temporary IPv6 Address. . . . . . : 2001:8003:5533:6901:190f:5adc:3815:293a
   Link-local IPv6 Address . . . . . : fe80::ed7a:d65c:8fd2:bd90%4
   IPv4 Address. . . . . . . . . . . : 192.168.0.11
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::be30:d9ff:fee6:d350%4
                                       192.168.0.1

Ethernet adapter Bluetooth Network Connection:
```

Save the hosts file, and I always re-open to make sure my changes have taken place. You should now be able to "ping" your chosen custom domain, and it should resolve to



```
PS C:\Users\lesja> ping localshop.io

Pinging localshop.io [192.168.0.11] with 32 bytes of data:
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.11:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
PS C:\Users\lesja>
```

If you only want to use HTTP, (and are not using HTTPS), you can stop here and go on your merry way, after you've updated the *applicationUrl* section in the ***launchSettings.json*** file to reflect your custom domain name. If you **do want to use HTTPS**, we have more work to do, but before we move on, a word of caution…

DHCP – Changing IP Address

If like me your using Dynamic Host Control Protocol, (DHCP), to dynamically allocate IP addresses on your network, just be aware that the IP Address you've configured in the Hosts file will need to be manually updated anytime your network adapter is assigned a new IP address via DHCP. So once you're up and running and then one day everything stops working – check the IP address in the Hosts file Vs the IP Address assigned to your network adapter and make sure they are the same. There are ways around this, (e.g. assign yourself a static IP address), but that's outside the scope of this article.

I was originally going to detail a bit about the mechanics of HTTPS and Certificates, but there's already a load of great info out there that I'd just be duplicating. If you want an overview I highly recommended reading the stuff **Cloudflare has put up here,** it's really readable and to the point.

I'd also highly recommend this article by **Panayotis Vryonis** on Public Key Encryption, it's one of the best explanations I've read, (it's also referenced by the Cloudflare docs).

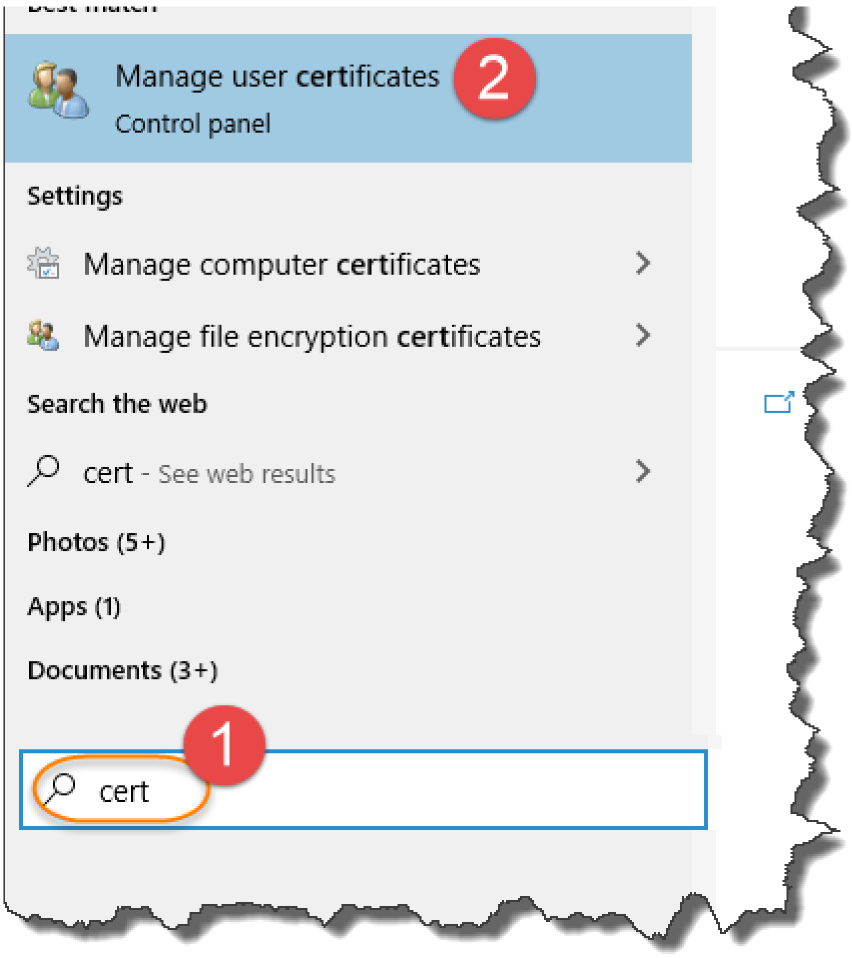
Self-Signed Certificate

So I've already mentioned the creation of a self-signed certificate using the *dotnet dev-certs* tool, however we are now going onto use a different technique to create a new self-signed certificate for our custom domain. To do this we are going to use PowerShell, and specifically the ***New-SelfSignedCertificate*** "cmdlet".

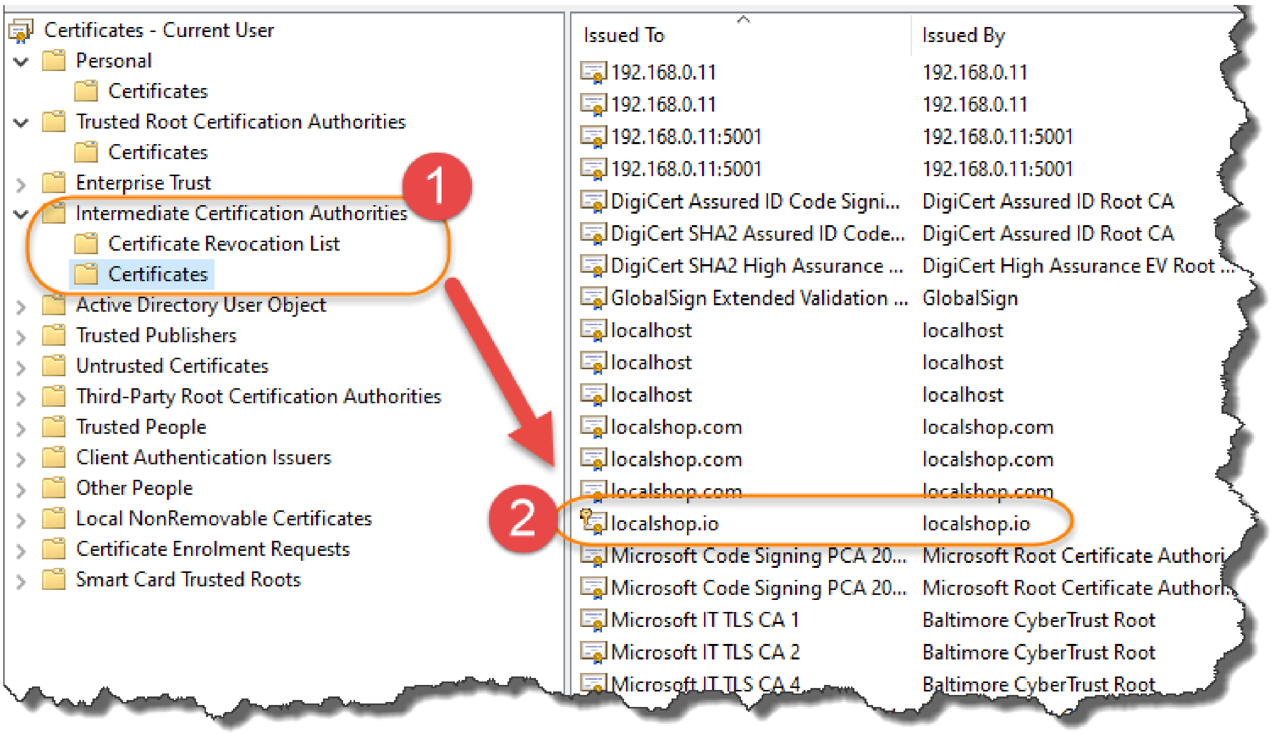
So open a new PowerShell session with Administrator rights, (as we had to do when we edited the hosts file), and enter the following to create a new self-signed certificate, being sure to replace the the domain name with the one you want to use:

```
$cert = New-SelfSignedCertificate -certstorelocation cert:\localmacl
```

This will create a new certificate and place it in *Intermediate Certification Authorities*. To view it, (although we're not yet done so ***don't close*** your PowerShell session), type "cert" into the windows search box and select Manage User Certificates:



Once CertMan is open browse to Intermediate Certification Authorities:



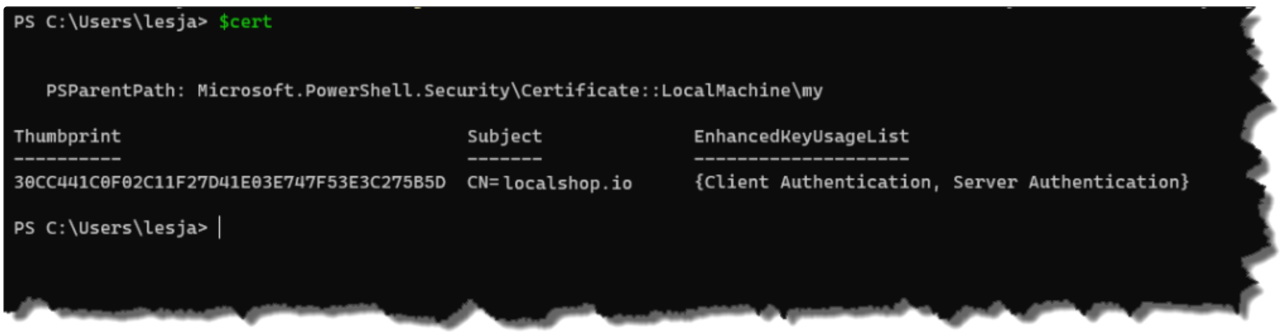
- Certificates - Current User
  - Personal
    - Certificates
  - Trusted Root Certification Authorities
    - Certificates
  - Enterprise Trust
  - Intermediate Certification Authorities
    - Certificate Revocation List
    - Certificates
  - Active Directory User Object
  - Trusted Publishers
  - Untrusted Certificates
  - Third-Party Root Certification Authorities
  - Trusted People
  - Client Authentication Issuers
  - Other People
  - Local NonRemovable Certificates
  - Certificate Enrolment Requests
  - Smart Card Trusted Roots

Issued To	Issued By
192.168.0.11	192.168.0.11
192.168.0.11	192.168.0.11
192.168.0.11:5001	192.168.0.11:5001
192.168.0.11:5001	192.168.0.11:5001
DigiCert Assured ID Code Signi...	DigiCert Assured ID Root CA
DigiCert SHA2 Assured ID Code...	DigiCert Assured ID Root CA
DigiCert SHA2 High Assurance ...	DigiCert High Assurance EV Root ...
GlobalSign Extended Validation ...	GlobalSign
localhost	localhost
localhost	localhost
localhost	localhost
localshop.com	localshop.com
localshop.com	localshop.com
localshop.com	localshop.com
localshop.io	localshop.io
Microsoft Code Signing PCA 20...	Microsoft Root Certificate Authori
Microsoft Code Signing PCA 20...	Microsoft Root Certificate Author
Microsoft IT TLS CA 1	Baltimore CyberTrust Root
Microsoft IT TLS CA 2	Baltimore CyberTrust Root
Microsoft IT TLS CA 4	Baltimore CyberTrust Root

Back in the PowerShell session, if you issue:

```
$cert
```

You should see some thing like this:



```
PS C:\Users\lesja> $cert


   PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\my

Thumbprint                                Subject         EnhancedKeyUsageList
----------                                -------         --------------------
30CC441C0F02C11F27D41E03E747F53E3C275B5D  CN= localshop.io     {Client Authentication, Server Authentication}

PS C:\Users\lesja> |
```

Take note of:

- The correct domain name
- The capabilities of this certificate, (this certificate can identify both clients and the server)

Next we're going to create a password that will be used to protect the Certificate, (specifically the *Private Key*), when we come to using it later, so ensure you keep this safe.

Still in the same PowerShell session, enter:

```
$pwd = ConvertTo-SecureString -String "pa55w0rd!" -Force -AsPlainTe
```

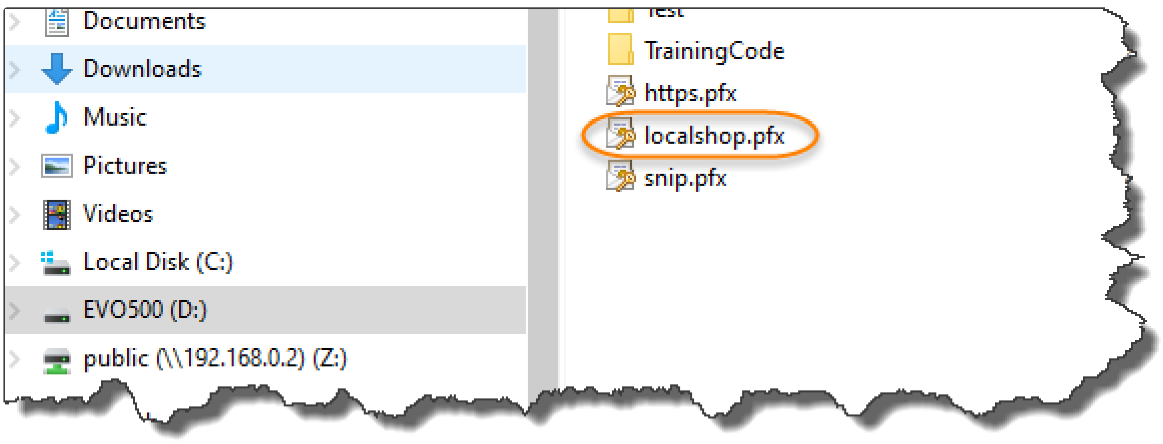
This will create a secure password string, (feel free to update the "pa55w0rd!" value with something of your own – just remember it!), and places it in a local PowerShell variable called ***$pwd***.

Next we're going to create another local variable that will contain the path to our certificate, this makes use of the unique certificate thumbprint that we reference via our existing ***$cert*** variable:

```
$certpath = "Cert:\localMachine\my\$($cert.Thumbprint)"
```

```
Export-PfxCertificate -Cert $certpath -FilePath d:\localshop.pfx -Pa
```

In this case I've just exported the certificate to the root of my D: drive:

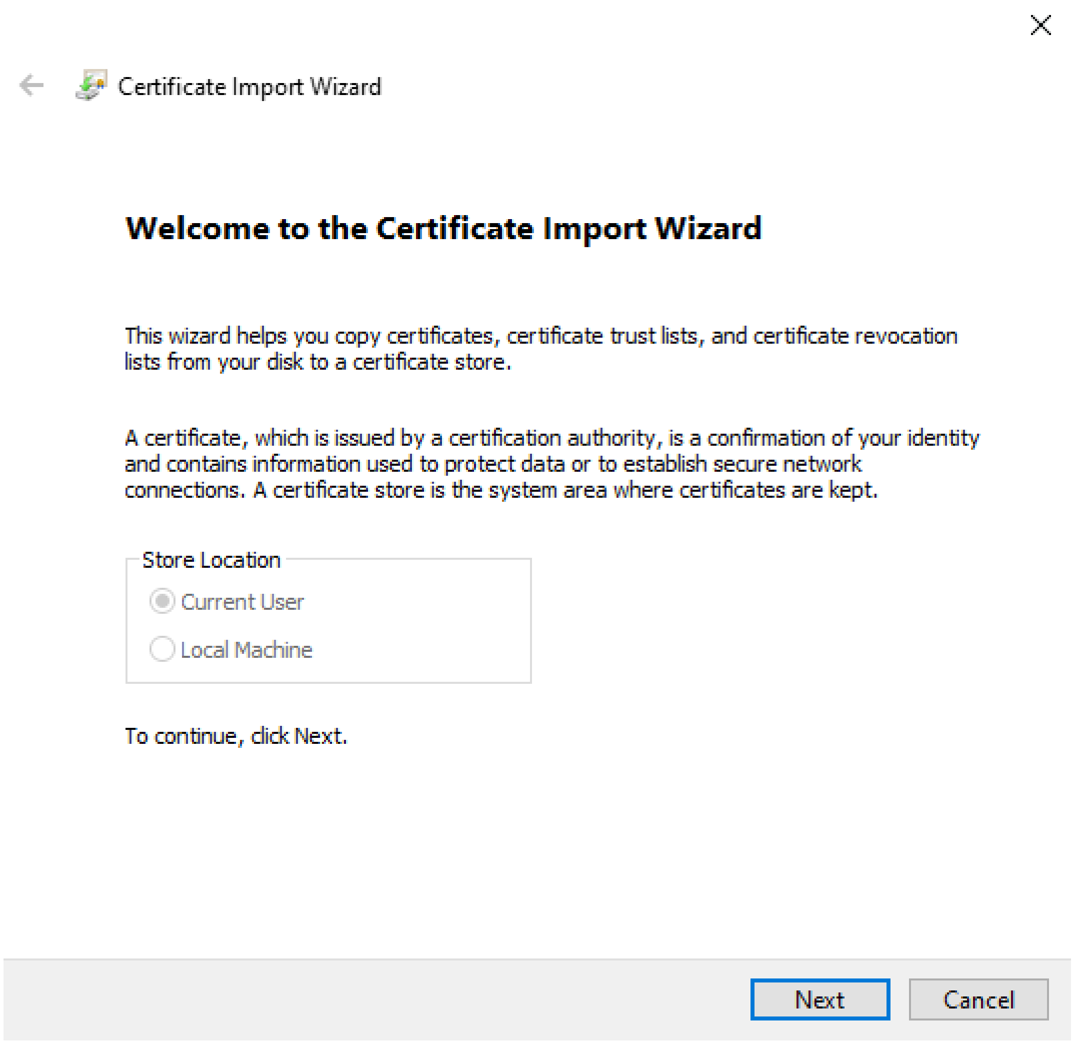


Add To Trusted Root

We'll be ingesting this certificate into Kestrel, which it will then present it as the secure certificate for our ASP.NET Core site. In order for our client, (which in this case will be the Chrome web browser), to actually *trust this certificate*, we'll need to add it to *Trusted Root Certification Authorities*, (as *dotnet dev-certs* did with the localhost certificate).

Note: Both Chrome, Edge and Internet Explorer will look in this location to see if the certificate can be trusted, (i.e. that it has come from a trusted authority). Firefox maintains it's own list of Trusted Authorities so you'd have to perform additional steps, (not shown here), to import the certificate so Firefox clients can trust it.

Certificate Import Wizard:



Certificate Import Wizard

Welcome to the Certificate Import Wizard

This wizard helps you copy certificates, certificate trust lists, and certificate revocation lists from your disk to a certificate store.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

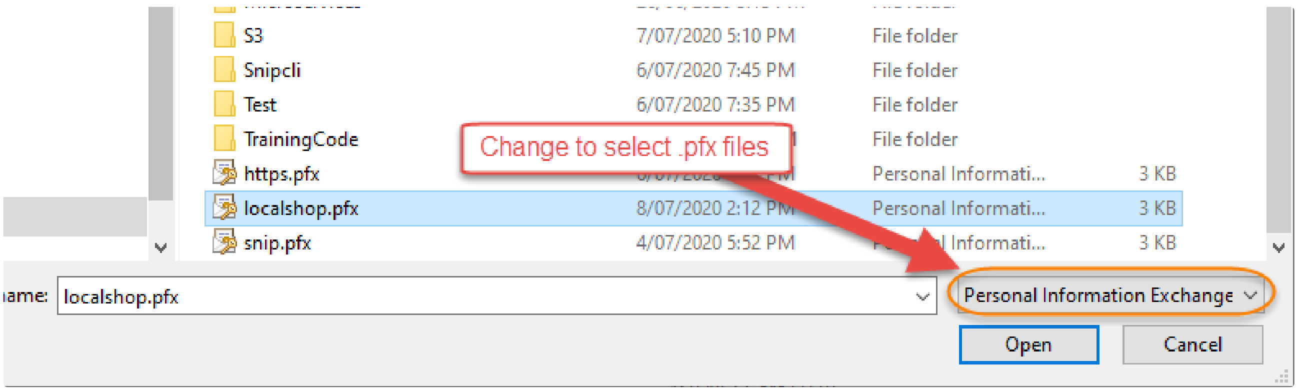
Store Location

[✓ Current User]

Local Machine

To continue, click Next.

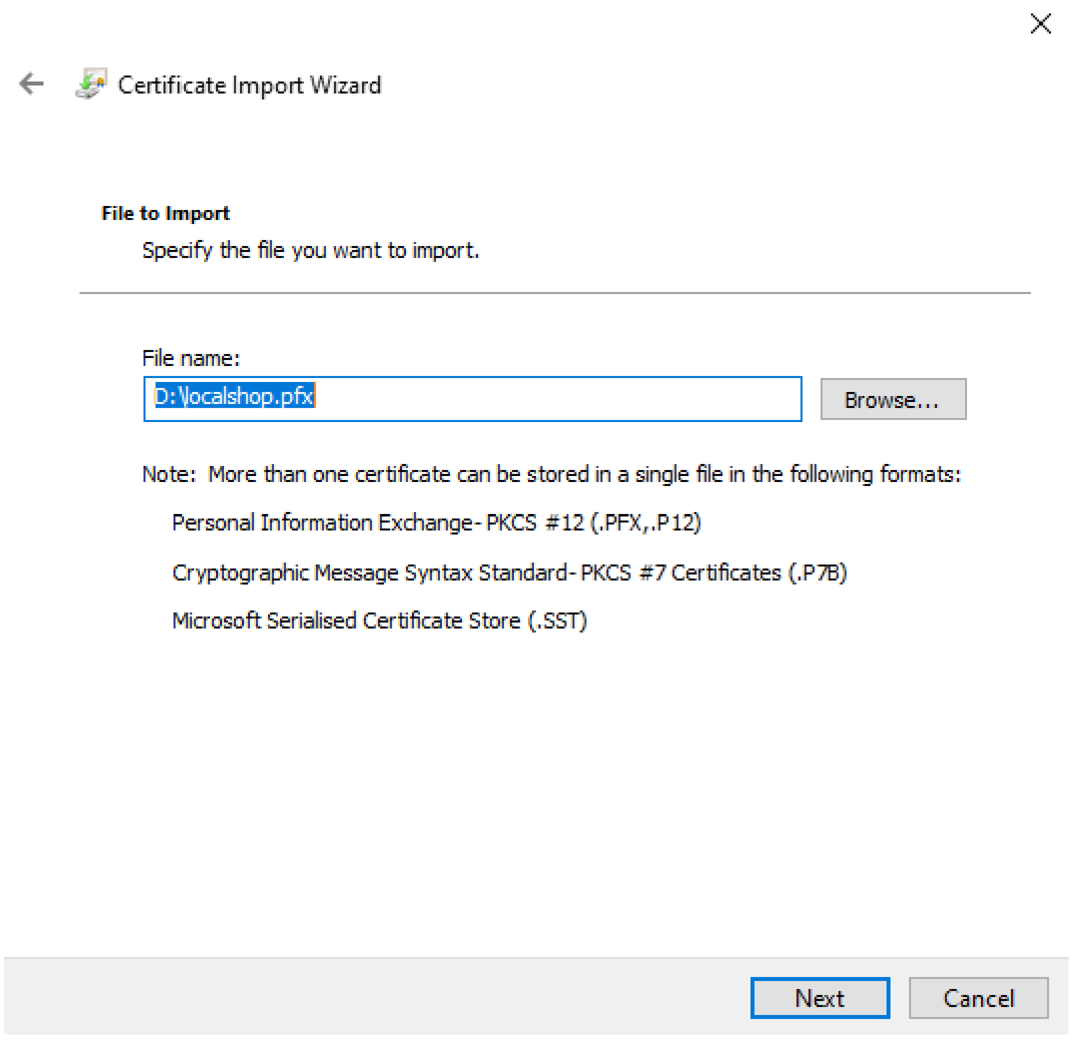
Click Next, browse to the location of the newly created .pfx file, then click Open, (note you may have to change the file type selector on the File Browse dialog):



S3	7/07/2020 5:10 PM	File folder	
Snipcli	6/07/2020 7:45 PM	File folder	
Test	6/07/2020 7:35 PM	File folder	
TrainingCode		File folder	
https.pfx		Personal Informati...	3 KB
localshop.pfx	8/07/2020 2:12 PM	Personal Informati...	3 KB
snip.pfx	4/07/2020 5:52 PM	Informati...	3 KB

ame: localshop.pfx

Personal Information Exchange



File to Import

Specify the file you want to import.

File name:

D:\localshop.pfx

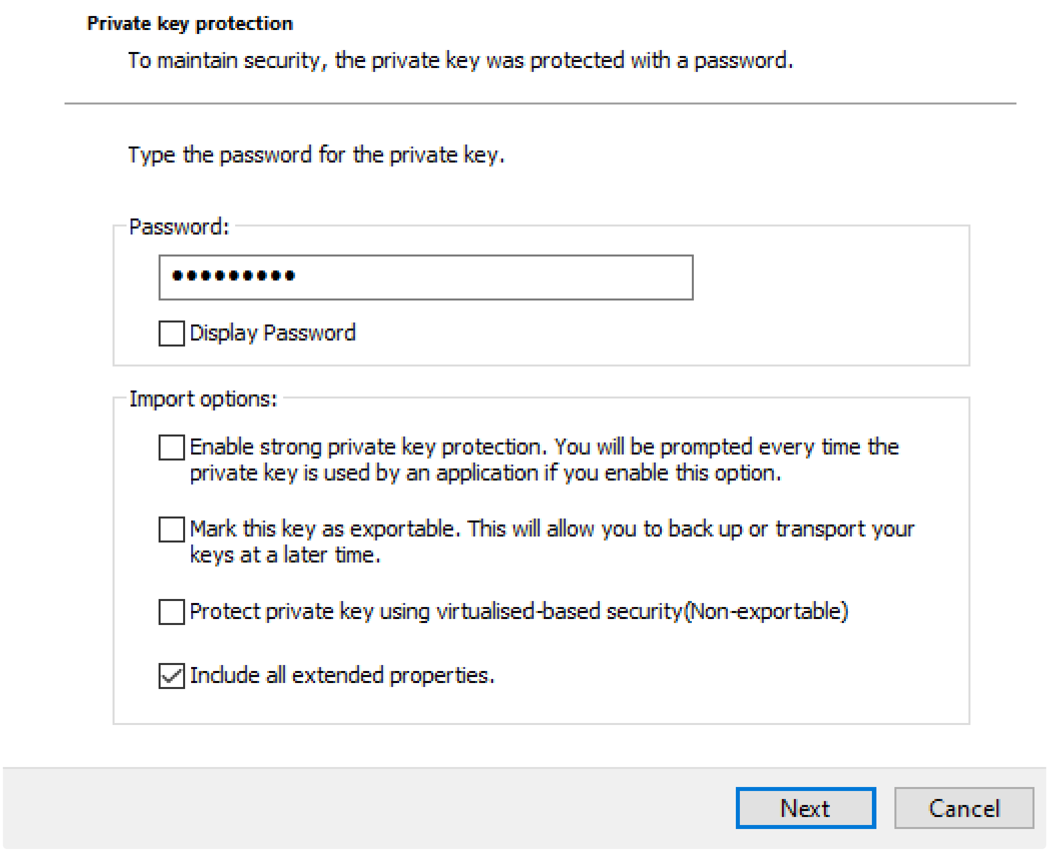
Note: More than one certificate can be stored in a single file in the following formats:

Personal Information Exchange- PKCS #12 (.PFX,.P12)

Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)

Microsoft Serialised Certificate Store (.SST)

Type the password in that you used when you created our certificate:



Private key protection

To maintain security, the private key was protected with a password.

Type the password for the private key.

Password:

••••••••••

☐ Display Password

Import options:

☐ Enable strong private key protection. You will be prompted every time the private key is used by an application if you enable this option.

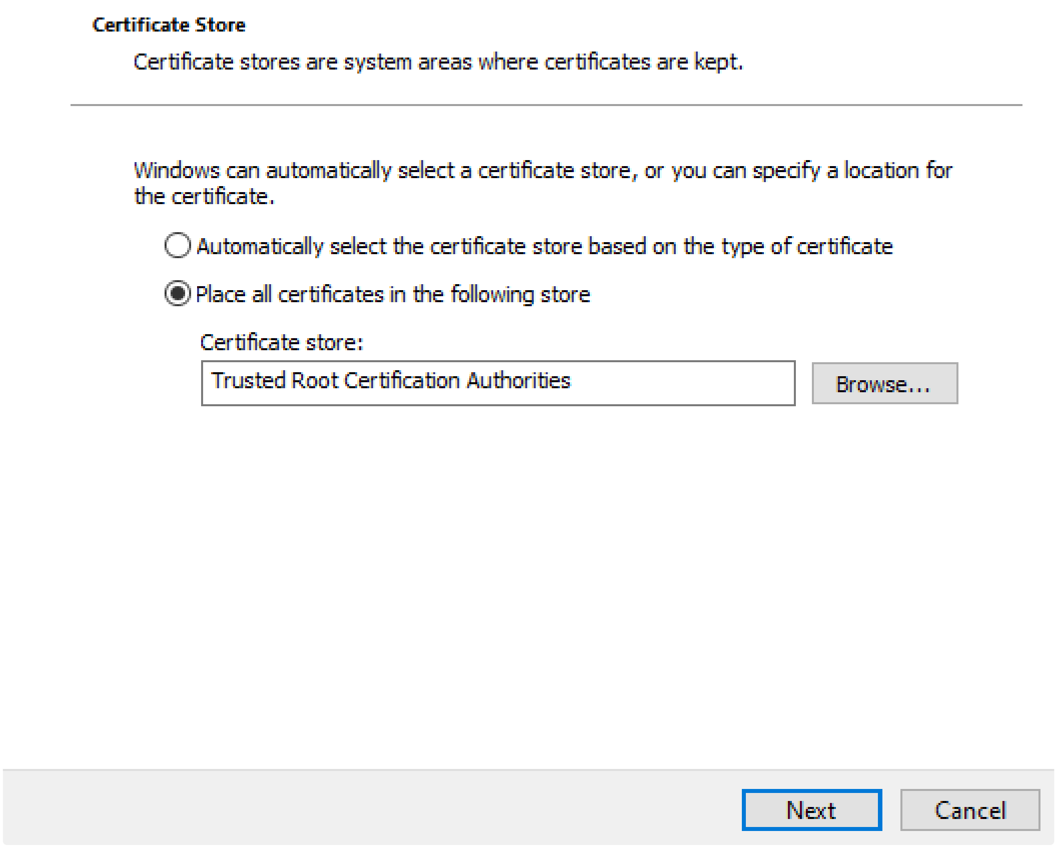
☐ Mark this key as exportable. This will allow you to back up or transport your keys at a later time.

☐ Protect private key using virtualised-based security(Non-exportable)

☑ Include all extended properties.

Click Next, and ensure that the correct location is selected for the import, this should be:

Trusted Root Certification Authorities as shown below:


Certificate Store

Certificate stores are system areas where certificates are kept.

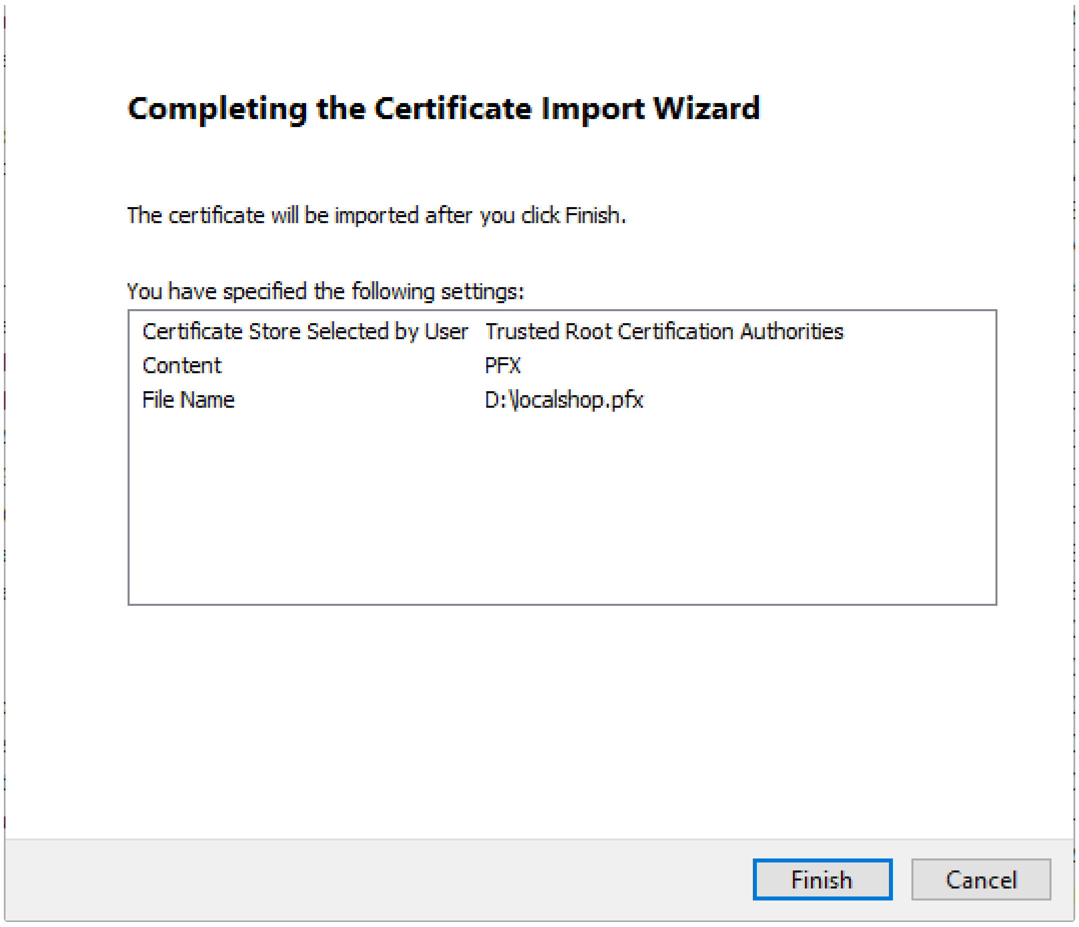
Windows can automatically select a certificate store, or you can specify a location for the certificate.

○ Automatically select the certificate store based on the type of certificate

● Place all certificates in the following store

Certificate store: Trusted Root Certification Authorities

Click Next, and on the resulting screen, review your selections and if happy click Finish:


Completing the Certificate Import Wizard

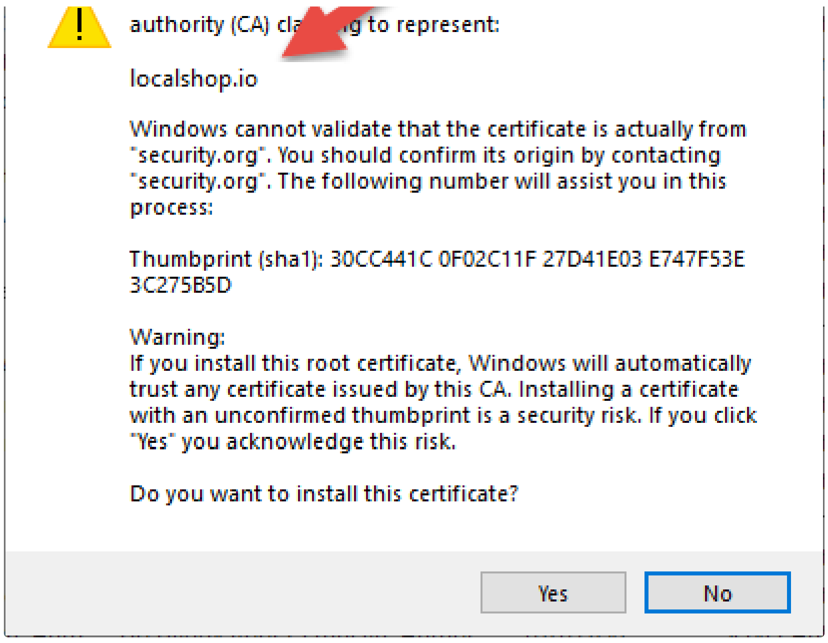
The certificate will be imported after you click Finish.

You have specified the following settings:

Certificate Store Selected by User	Trusted Root Certification Authorities
Content	PFX
File Name	D:\localshop.pfx

You'll then get a Security Warning challenge, asking if you're sure you want to do this.

Remember we are now going to "trust" any server that presents this certificate to us...


authority (CA) cla... g to represent:

localshop.io

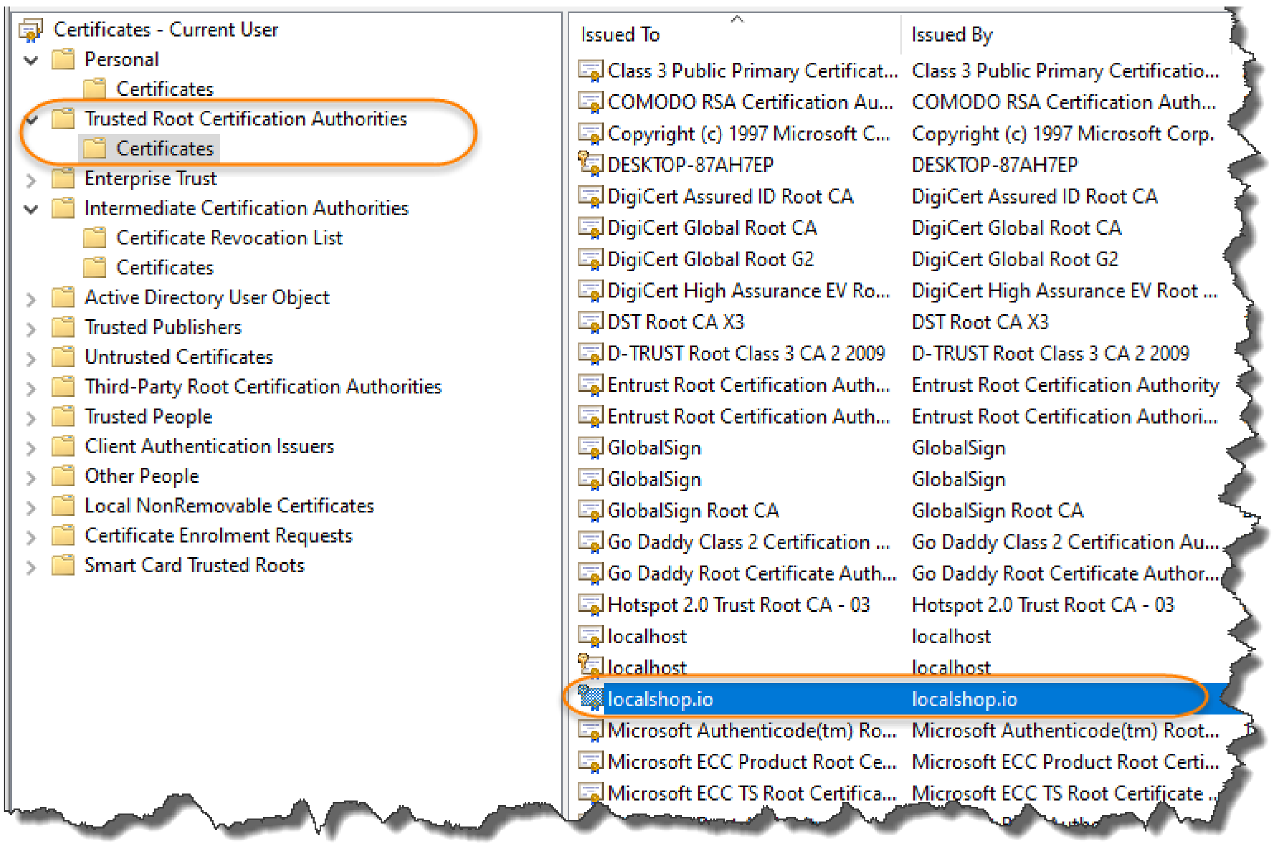
Windows cannot validate that the certificate is actually from "security.org". You should confirm its origin by contacting "security.org". The following number will assist you in this process:

Thumbprint (sha1): 30CC441C 0F02C11F 27D41E03 E747F53E 3C275B5D

Warning:
If you install this root certificate, Windows will automatically trust any certificate issued by this CA. Installing a certificate with an unconfirmed thumbprint is a security risk. If you click "Yes" you acknowledge this risk.

Do you want to install this certificate?

Assuming you trust yourself, click Yes and the Certificate will be imported into our Trusted Root Certificate Authority folder, double check by ensuring the certificate has been installed here:


Certificates - Current User
Personal
Certificates
Trusted Root Certification Authorities
Certificates
Enterprise Trust
Intermediate Certification Authorities
Certificate Revocation List
Certificates
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
Trusted People
Client Authentication Issuers
Other People
Local NonRemovable Certificates
Certificate Enrolment Requests
Smart Card Trusted Roots

Issued To	Issued By
Class 3 Public Primary Certificat...	Class 3 Public Primary Certificatio...
COMODO RSA Certification Au...	COMODO RSA Certification Auth...
Copyright (c) 1997 Microsoft C...	Copyright (c) 1997 Microsoft Corp.
DESKTOP-87AH7EP	DESKTOP-87AH7EP
DigiCert Assured ID Root CA	DigiCert Assured ID Root CA
DigiCert Global Root CA	DigiCert Global Root CA
DigiCert Global Root G2	DigiCert Global Root G2
DigiCert High Assurance EV Ro...	DigiCert High Assurance EV Root ...
DST Root CA X3	DST Root CA X3
D-TRUST Root Class 3 CA 2 2009	D-TRUST Root Class 3 CA 2 2009
Entrust Root Certification Auth...	Entrust Root Certification Authority
Entrust Root Certification Auth...	Entrust Root Certification Authori...
GlobalSign	GlobalSign
GlobalSign	GlobalSign
GlobalSign Root CA	GlobalSign Root CA
Go Daddy Class 2 Certification ...	Go Daddy Class 2 Certification Au...
Go Daddy Root Certificate Auth...	Go Daddy Root Certificate Author...
Hotspot 2.0 Trust Root CA - 03	Hotspot 2.0 Trust Root CA - 03
localhost	localhost
localhost	localhost
localshop.io	localshop.io
Microsoft Authenticode(tm) Ro...	Microsoft Authenticode(tm) Root...
Microsoft ECC Product Root Ce...	Microsoft ECC Product Root Certi...
Microsoft ECC TS Root Certifica...	Microsoft ECC TS Root Certificate ...

Configuring Kestrel

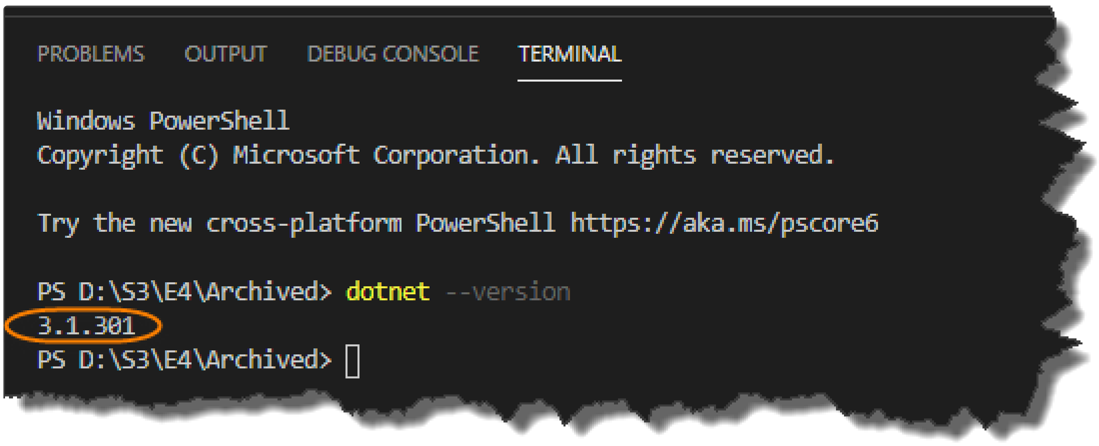
In this final section we are going to configure the Kestrel Web server to ingest our new self-signed certificate, which it communicates via HTTPS.

If you already have an existing ASP.NET Core app that you want to use – feel free to do so, however I'm going to create a very simple out the box *webapi* for the purposes of this article.

Create Test API

At a command prompt, type the following to ensure you have the .NET Core 3.1 SDK installed as you'll need it to follow along:

You should see something like:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\S3\E4\Archived> dotnet --version
3.1.301
PS D:\S3\E4\Archived>
```

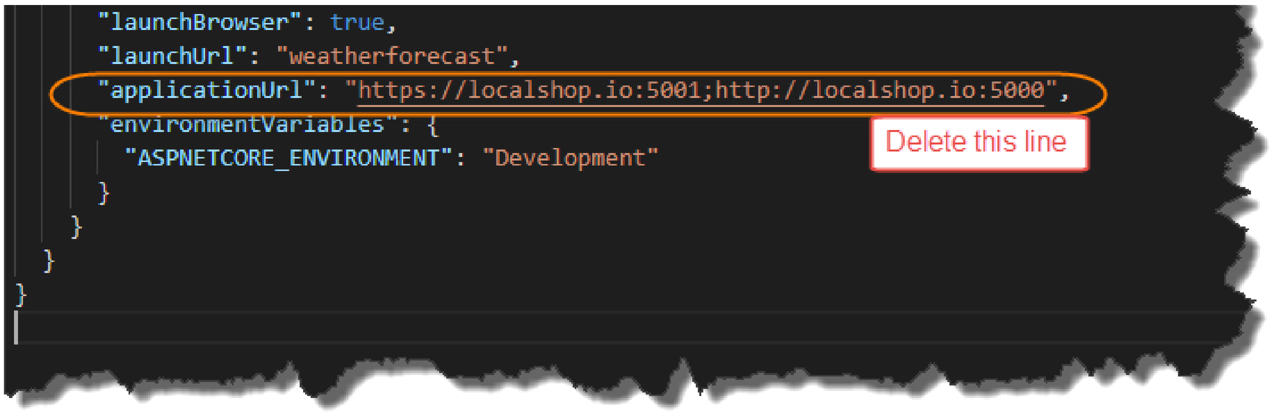
Next, (assuming you have this installed), move to a directory where you want to create your app and enter:

```
dotnet new webapi -n HttpsAPI
```

This will go off and create an out the box ASP.NET Core WebApi called "HttpsAPI", once it's created, open the resulting project folder, (not surprisingly called ***HttpsAPI***), in VS Code, (or whatever development environment you are using).

Update launchSettings.json

We are going to configure Kestrel in the *Program.cs* class so we should remove the "applicationUrl" from ***launchSettings.json*** as highlighted below:


```
      "launchBrowser": true,
      "launchUrl": "weatherforecast",
      "applicationUrl": "https://localshop.io:5001;http://localshop.io:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Remember to save the ***launchSettings.json*** file before moving on.

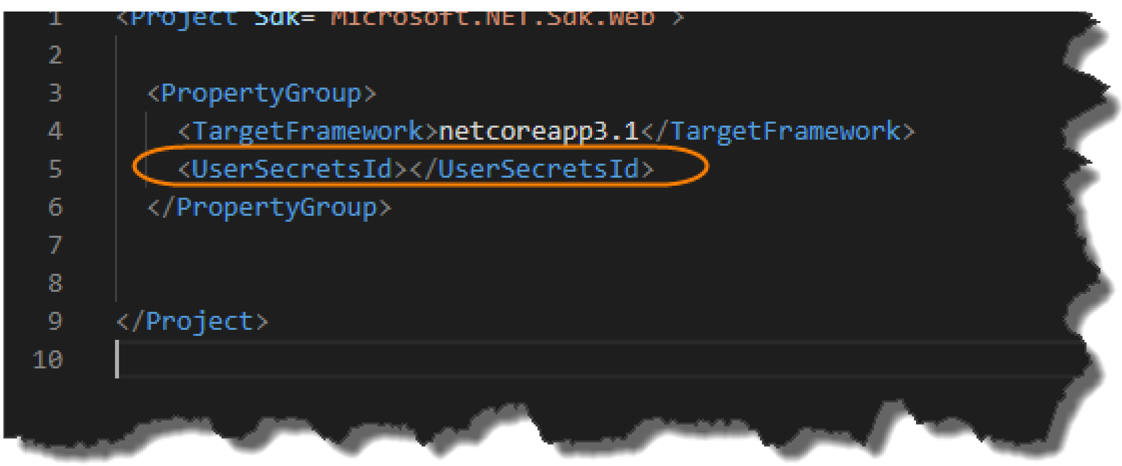
Add User Secrets

When we come to read in our Certificate for Kestrel to use, we need to supply:

- The Certificate File Path, (which we'll retrieve from ***appsettings.Development.json***)
- The password for the Certificate – which we need to keep "secret"

It is this 2nd point that requires the use of *User Secrets* which allow us to store sensitive information, (like passwords), in a file called ***secrets.json***. This file is unencrypted but is stored in a file-system protected user profile folder on the local development machine. It is therefore only available to *you*, the developer, (unless you give someone your Windows login – not advised).

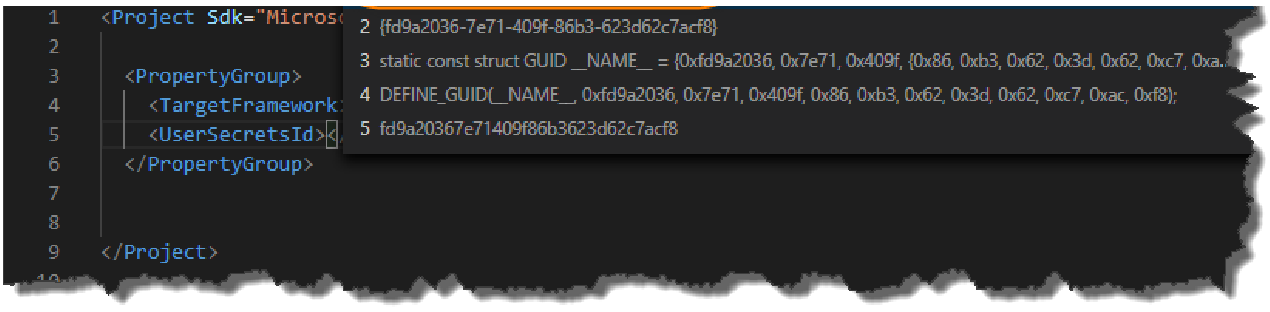
To use User Secrets, move over to the .csproj file, (mine is called ***HttpsAPI.csproj***), and insert a pair of *<UserSecretsID>* tags to the existing *<PropertyGroup>* tags as shown below:



```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UserSecretsId></UserSecretsId>
  </PropertyGroup>


</Project>

```

We now need to generate and insert a GUID value in between these tags, this ensures we retrieve the correct user secrets for a given project. As I'm using VS Code, I've installed an extension called "Insert GUID", which as the name suggests allows you to generate and insert GUIDs into your application code. To insert a GUID using this extension:

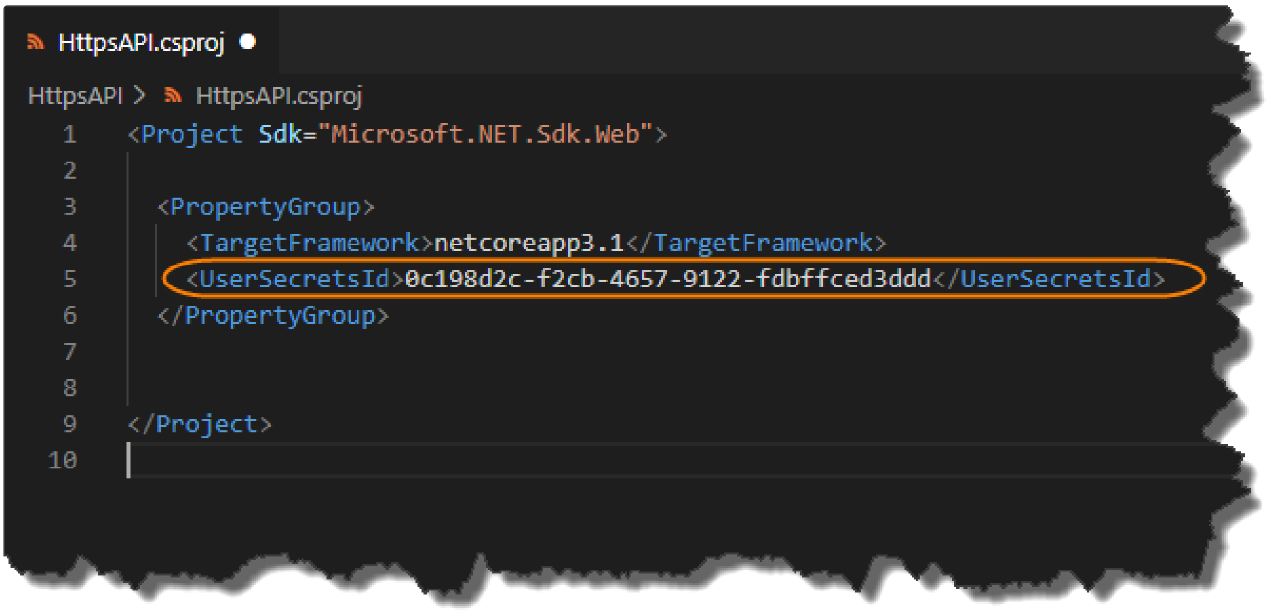
- Install the extension, (search "Insert GUID")
- Place your cursor in between the *<UserSecretsId>* tags
- Press F1 (in VS Code)
- Type "Insert Guid" and select it
- You should then have a choice of GUID formats to insert:



```
<Project Sdk="Micros

  <PropertyGroup>
    <TargetFramework
    <UserSecretsId><
  </PropertyGroup>


</Project>
```

```
2 {fd9a2036-7e71-409f-86b3-623d62c7acf8}
3 static const struct GUID __NAME__ = {0xfd9a2036, 0x7e71, 0x409f, {0x86, 0xb3, 0x62, 0x3d, 0x62, 0xc7, 0xa...
4 DEFINE_GUID(__NAME__, 0xfd9a2036, 0x7e71, 0x409f, 0x86, 0xb3, 0x62, 0x3d, 0x62, 0xc7, 0xac, 0xf8);
5 fd9a20367e71409f86b3623d62c7acf8
```

Pick the 1st Option, and a GUID will be inserted as shown below:



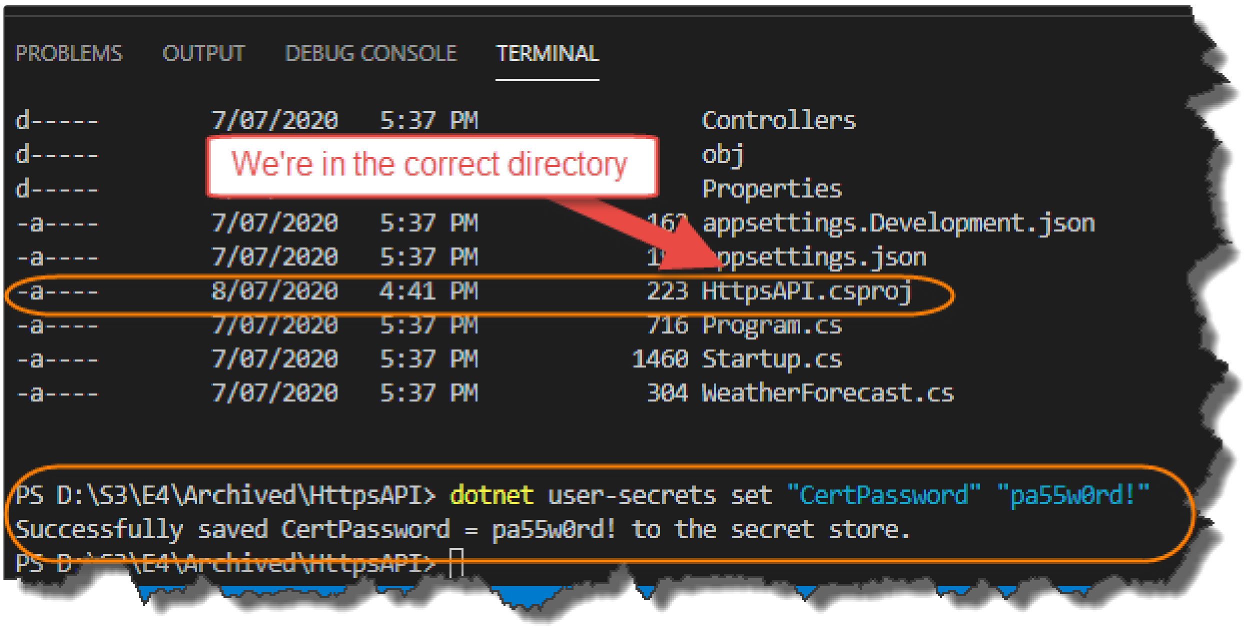
```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UserSecretsId>0c198d2c-f2cb-4657-9122-fdbffced3ddd</UserSecretsId>
  </PropertyGroup>


</Project>
```

Make sure you save your file before you continue.

If you're not using VS Code or don't want to install the Insert GUID extension, there are plenty of free GUID generators online, pick one, then generate and insert a GUID before moving on.

Now open a command prompt "inside" your project, (if you do a directory listing you should see the .csproj file we just updated), and issue the following command to insert a new User Secret:

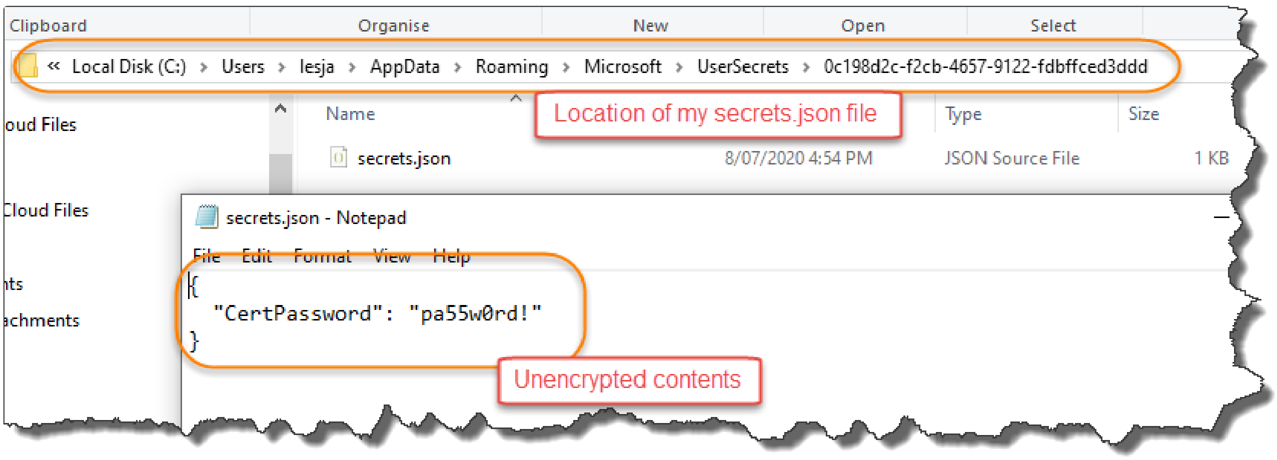
Make sure to update the password to whatever you used when you created the Certificate, you should see something like this:



```
d-----        7/07/2020   5:37 PM                Controllers
d-----                                           obj
d-----                                           Properties
-a----        7/07/2020   5:37 PM            162 appsettings.Development.json
-a----        7/07/2020   5:37 PM            1   appsettings.json
-a----        8/07/2020   4:41 PM            223 HttpsAPI.csproj
-a----        7/07/2020   5:37 PM            716 Program.cs
-a----        7/07/2020   5:37 PM           1460 Startup.cs
-a----        7/07/2020   5:37 PM            304 WeatherForecast.cs

PS D:\S3\E4\Archived\HttpsAPI> dotnet user-secrets set "CertPassword" "pa55w0rd!"
Successfully saved CertPassword = pa55w0rd! to the secret store.
PS D:\S3\E4\Archived\HttpsAPI>
```

On Windows the ***secrets.json*** file is stored in: C:\Users\<User Name>\AppData\Roaming\Microsoft\UserSecrets\<ProjectGUID>

I've shown this location and the opened file below:


Clipboard
Organise
New
Open
Select
« Local Disk (C:) › Users › lesja › AppData › Roaming › Microsoft › UserSecrets › 0c198d2c-f2cb-4657-9122-fdbffced3ddd
Location of my secrets.json file
Name
Type
Size
oud Files
Cloud Files
nts
achments
secrets.json - Notepad
File Edit Format View Help
Unencrypted contents

secrets.json 8/07/2020 4:54 PM JSON Source File 1 KB

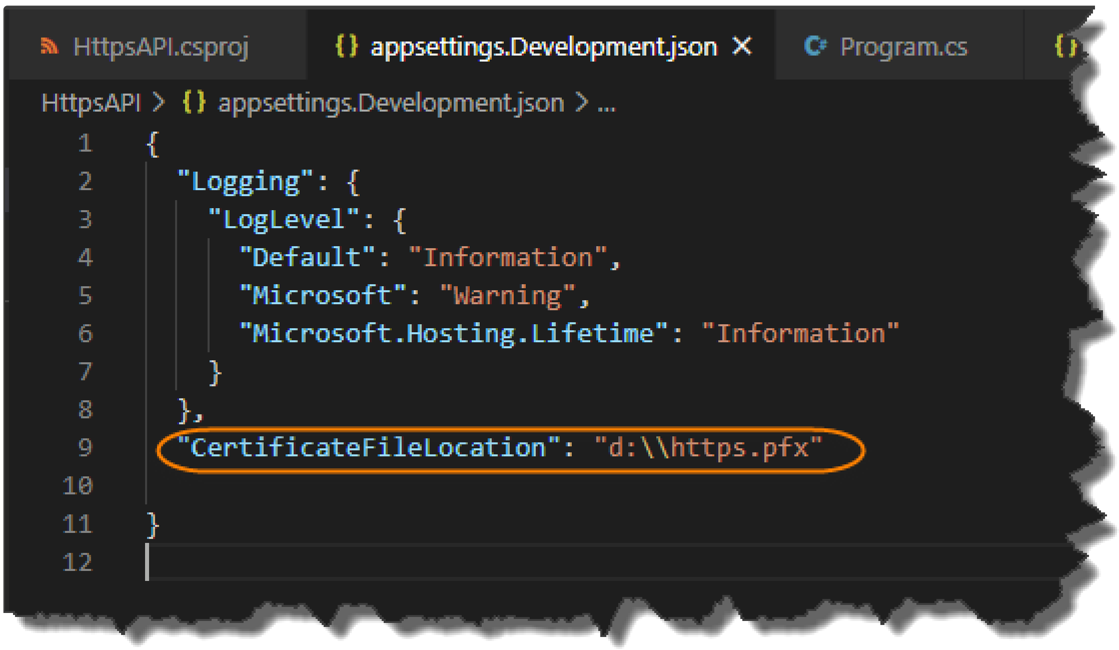
```
{
  "CertPassword": "pa55w0rd!"
}
```

Update appsettings.Development.json

The last bit of config we require is to store the file path location of the .pfx file we created earlier on, we're going to store that in ***appsettings.Development.json***, so add a new JSON attribute called *CertificateFileLocation* and assign the relevant value, (that corresponds to where the file it located), as shown below, (noting the escaped back-slash in the file path):

```
"CertificateFileLocation": "d:\\https.pfx"
```

To put the change in context, your ***appsettings.Development.json*** file should look like this:


HttpsAPI.csproj
appsettings.Development.json
Program.cs
HttpsAPI > appsettings.Development.json > ...

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "CertificateFileLocation": "d:\\https.pfx"

}
```

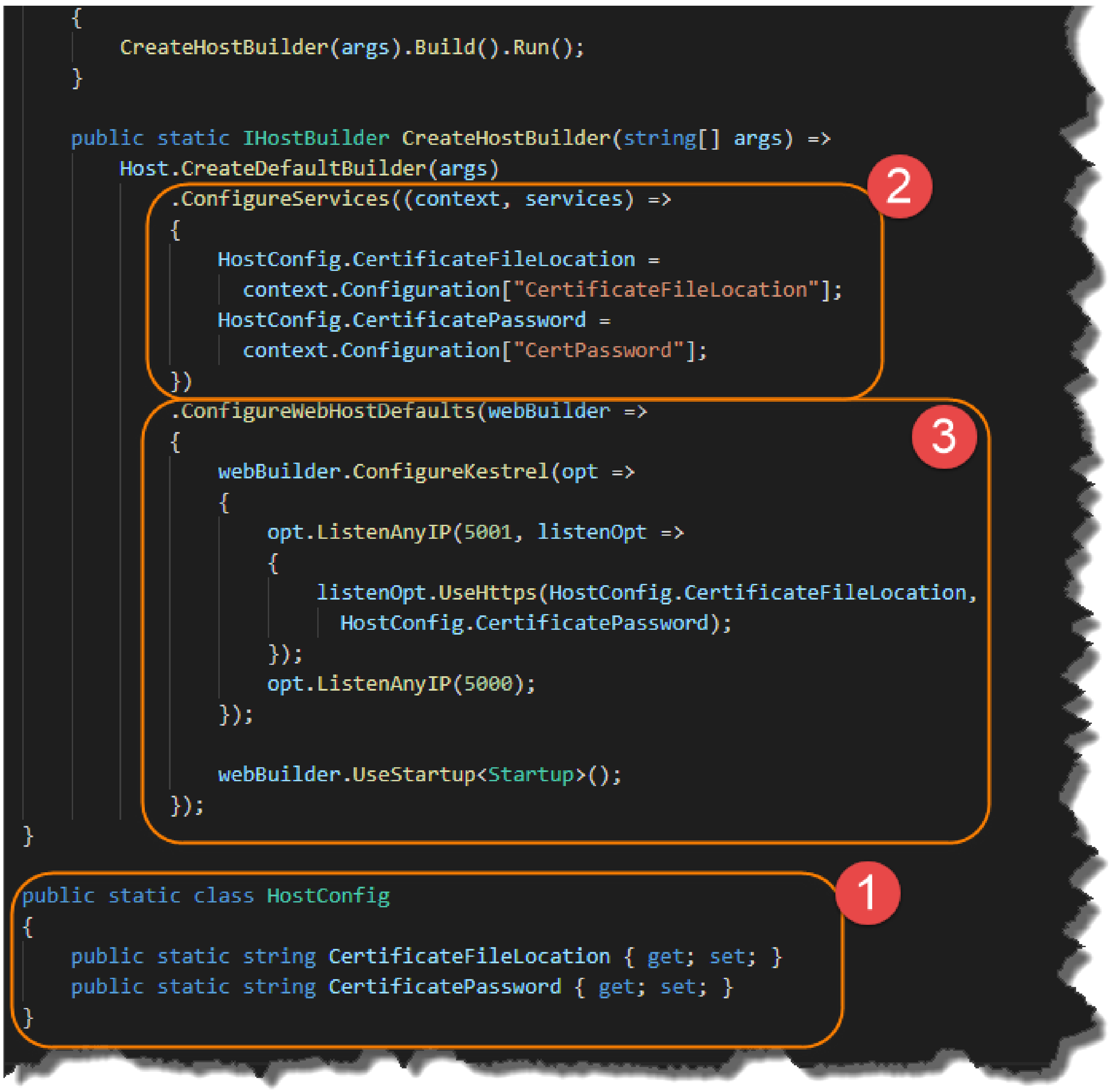
Finally we need to add the code to the Program class to:

- Read in our 2 config elements (Our Password & Certificate File Path)
- Configure Kestrel accordingly

I've included the code for our *Program* and simple *HostConfig* classes below, but remember the **code is also available on GitHub**.

```
public class Program
{
  public static void Main(string[] args)
  {
    CreateHostBuilder(args).Build().Run();
  }

  public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
      .ConfigureServices((context, services) =>
      {
        HostConfig.CertificateFileLocation =
          context.Configuration["CertificateFileLocation"];
        HostConfig.CertificatePassword =
          context.Configuration["CertPassword"];
      })
      .ConfigureWebHostDefaults(webBuilder =>
      {
        webBuilder.ConfigureKestrel(opt =>
        {
          opt.ListenAnyIP(5001, listenOpt =>
          {
            listenOpt.UseHttps(
              HostConfig.CertificateFileLocation,
          opt.ListenAnyIP(5000);
        });

        webBuilder.UseStartup<Startup>();
      });
}

public static class HostConfig
{
  public static string CertificateFileLocation { get; set; }
  public static string CertificatePassword { get; set; }
}
```

To quickly run through the code, I've labelled the major sections as shown below:



```
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices((context, services) =>
            {
                HostConfig.CertificateFileLocation =
                    context.Configuration["CertificateFileLocation"];
                HostConfig.CertificatePassword =
                    context.Configuration["CertPassword"];
            })
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.ConfigureKestrel(opt =>
                {
                    opt.ListenAnyIP(5001, listenOpt =>
                    {
                        listenOpt.UseHttps(HostConfig.CertificateFileLocation,
                            HostConfig.CertificatePassword);
                    });
                    opt.ListenAnyIP(5000);
                });

                webBuilder.UseStartup<Startup>();
            });
}

public static class HostConfig
{
    public static string CertificateFileLocation { get; set; }
    public static string CertificatePassword { get; set; }
}
```

Section 1: HostConfig

This is just a simple static class that has 2 properties used to store our 2 configuration elements.

Section 2: ConfigureServices

We add the *ConfigureServices* extension method to enable is to get access to our standard configuration sources, which in this case are:

- ***secrets.json*** (holds our Certificate Password)

You'll notice that we only reference the *name* of the configuration elements, (*CertificateFileLocation* and *CertPassword*), and not the configuration sources themselves. This is because the .NET Core Configuration sub-layer abstracts, (or aggregates), all of the configuration sources so that we only need to reference the config elements by name.

We then simply assign our read-in elements to the properties of our static class for use in the last section.

Section 3: ConfigureWebHostDefaults

Here we access the *ConfigureKestrel* extension method and configure our 2 ports for listening. You'll notice that for our HTTPS Port, (5001), we perform some additional configuration to "UseHttps" and in doing so pass in the Certificate Path and Password.

This means when we serve up anything on port 5001, (our HTTPS Port), our self-signed certificate will be used. Indeed as we are using *HttpsRedirection* (have a look in the *Configure* method in the *Startup* class), we'll serve everything on HTTPS.

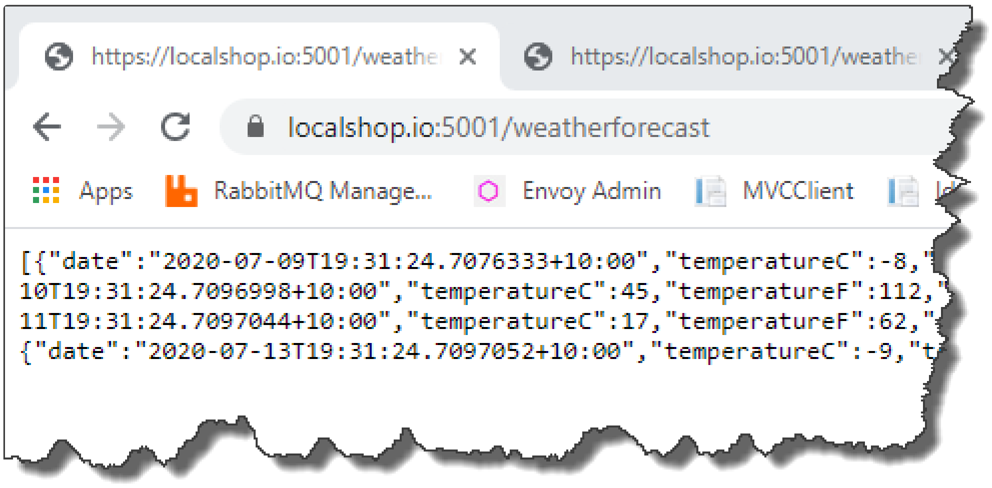
Bring it together

Save everything, and issue a:

```
dotnet build
dotnet run
```

To run it up.

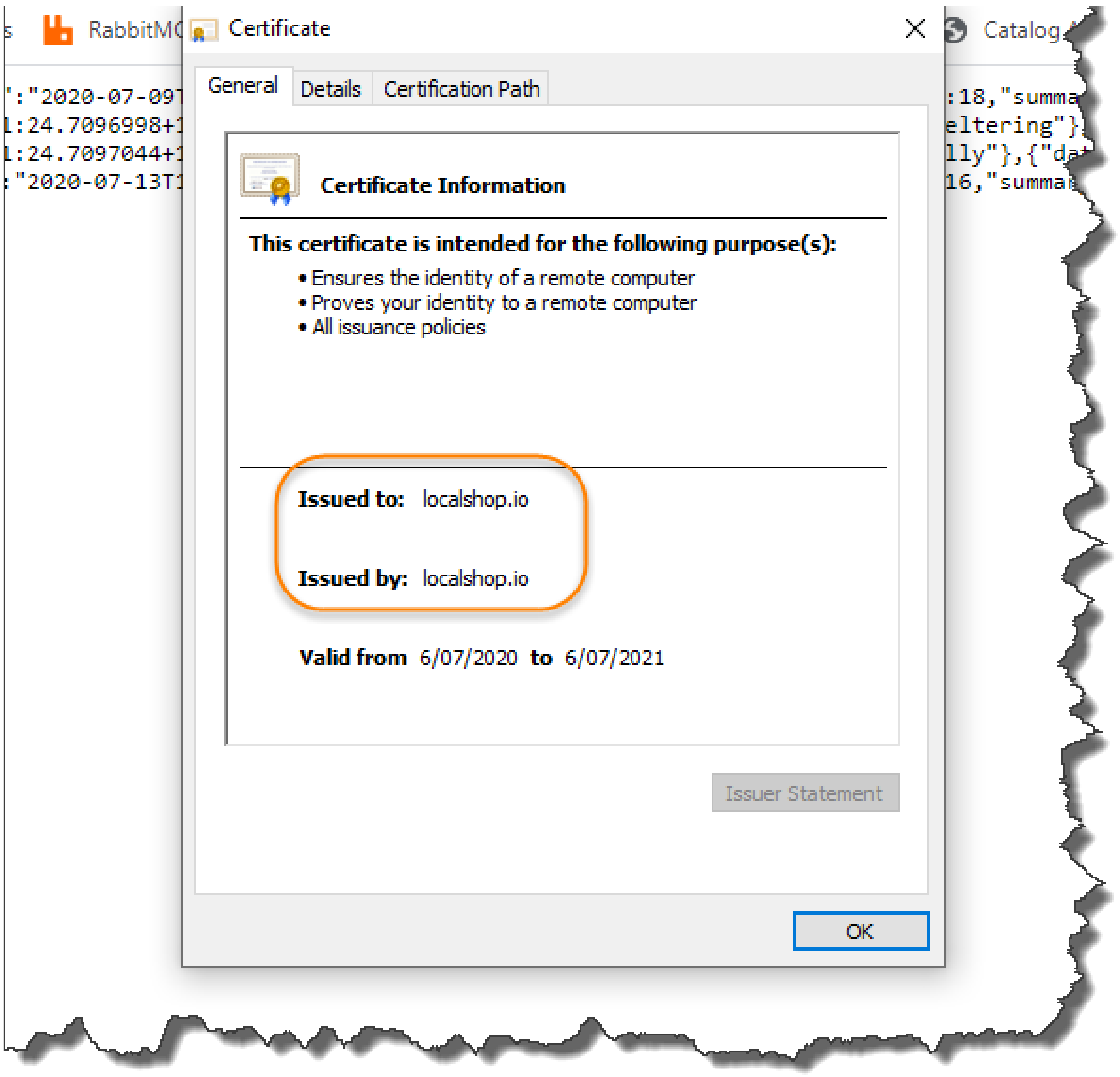
Move over to Chrome and enter the custom domain name, followed by port 5001 and the controller action for our API, (weatherforecast), and you should be served up a HTTPS Session with a valid Certificate:


https://localshop.io:5001/weathe
https://localshop.io:5001/weathe
localshop.io:5001/weatherforecast
Apps
RabbitMQ Manage...
Envoy Admin
MVCClient

```
[{"date":"2020-07-09T19:31:24.7076333+10:00","temperatureC":-8,"
10T19:31:24.7096998+10:00","temperatureC":45,"temperatureF":112,"
11T19:31:24.7097044+10:00","temperatureC":17,"temperatureF":62,"
{"date":"2020-07-13T19:31:24.7097052+10:00","temperatureC":-9,"t
```

Clicking on the “padlock” icon and take a look at the certificate being used:

Dotnet Playbook



Certificate Information

This certificate is intended for the following purpose(s):
- Ensures the identity of a remote computer
- Proves your identity to a remote computer
- All issuance policies

Issued to: localshop.io

Issued by: localshop.io

Valid from 6/07/2020 **to** 6/07/2021

SHARE:

Les Jackson

He's just obtained an MCSD accreditation after almost a year, so now has more time for writing this blog, making YouTube videos, as well as enjoying the fantastic beer, wine, coffee and food Melbourne has to offer.

RELATED ARTICLES

SECURITY

Secure a .NET Core API using Bearer Authentication

PREVIOUS POST
Secure a .NET Core API using Bearer Authentication

NEXT POST
Fault Handling with Polly – A Beginners Guide

Dotnet Playbook

Dotnet Playbook

CATEGORIES

.NET Core	2
Certification	2
Devops	1
Docker	2
Entity Framework	1
JSON	1
Microsoft Azure	2
REST API	4
Security	2
WebSockets	1

META

- Log in
- Entries feed
- Comments feed
- WordPress.org

Dotnet Playbook

≡

YOUTUBE