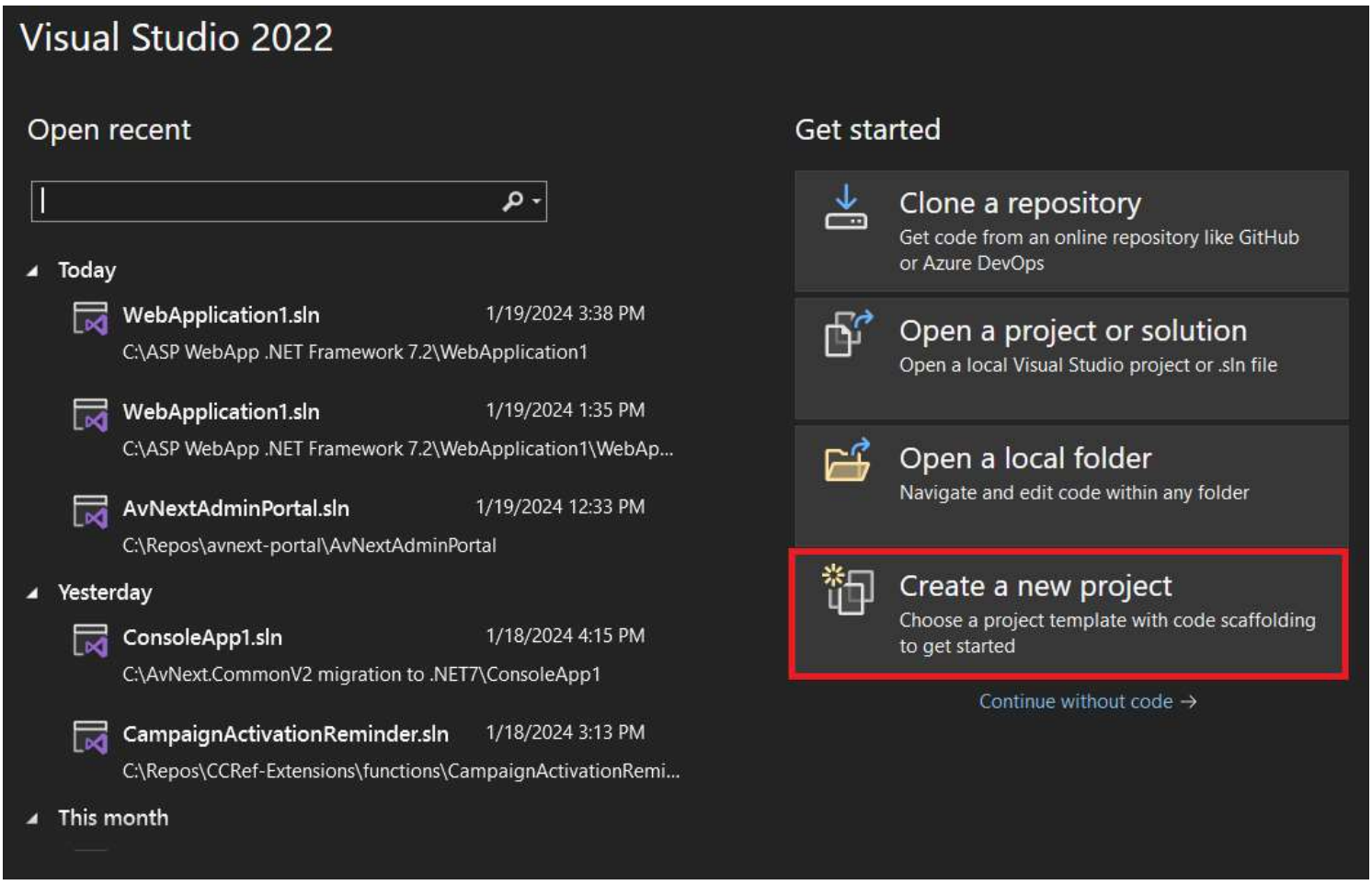
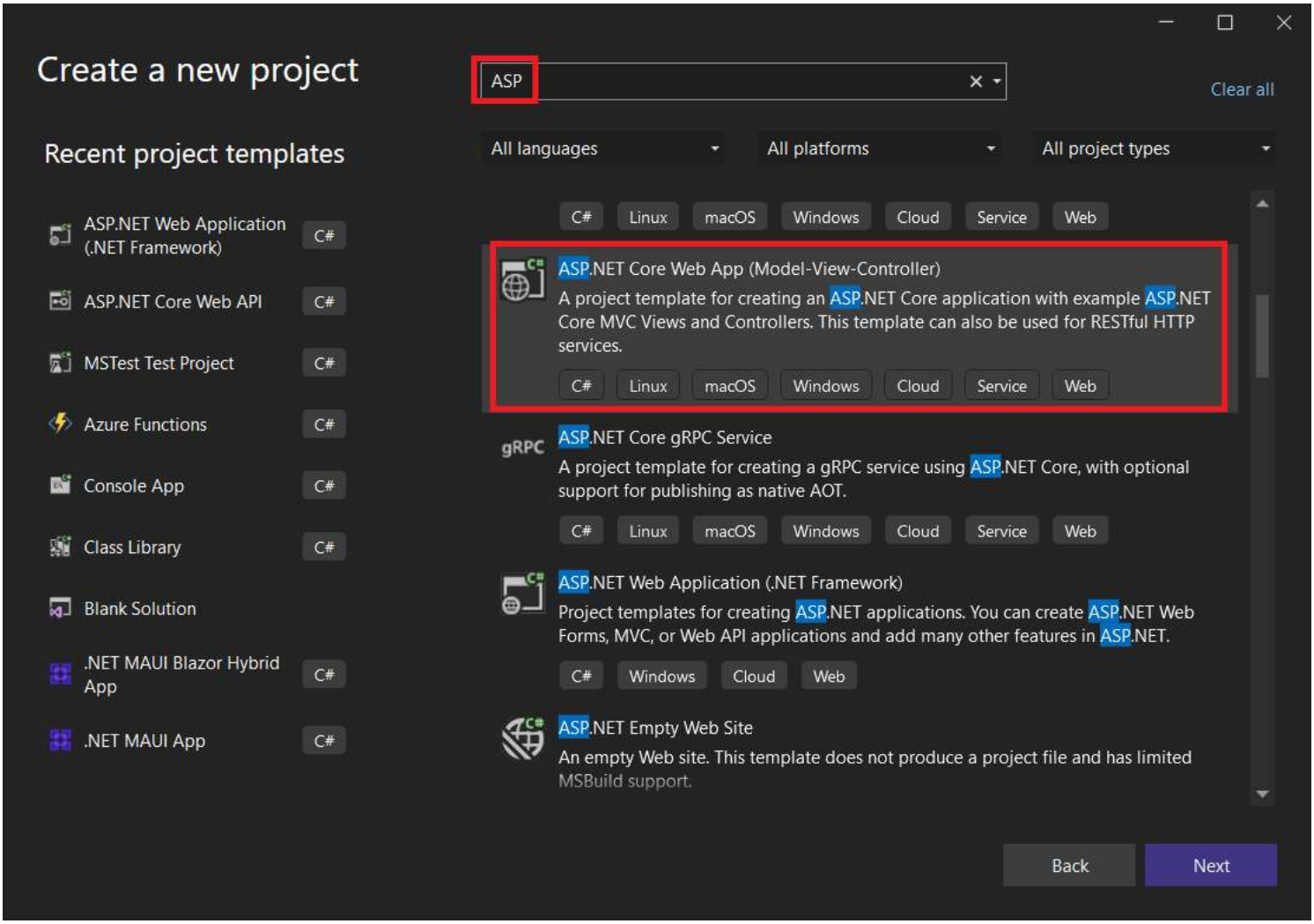
How to create my first ASP WebApp with .NET 8 and deploy to Docker Desktop (in your local laptop)

1. Run Visual Studio and create the WebApp

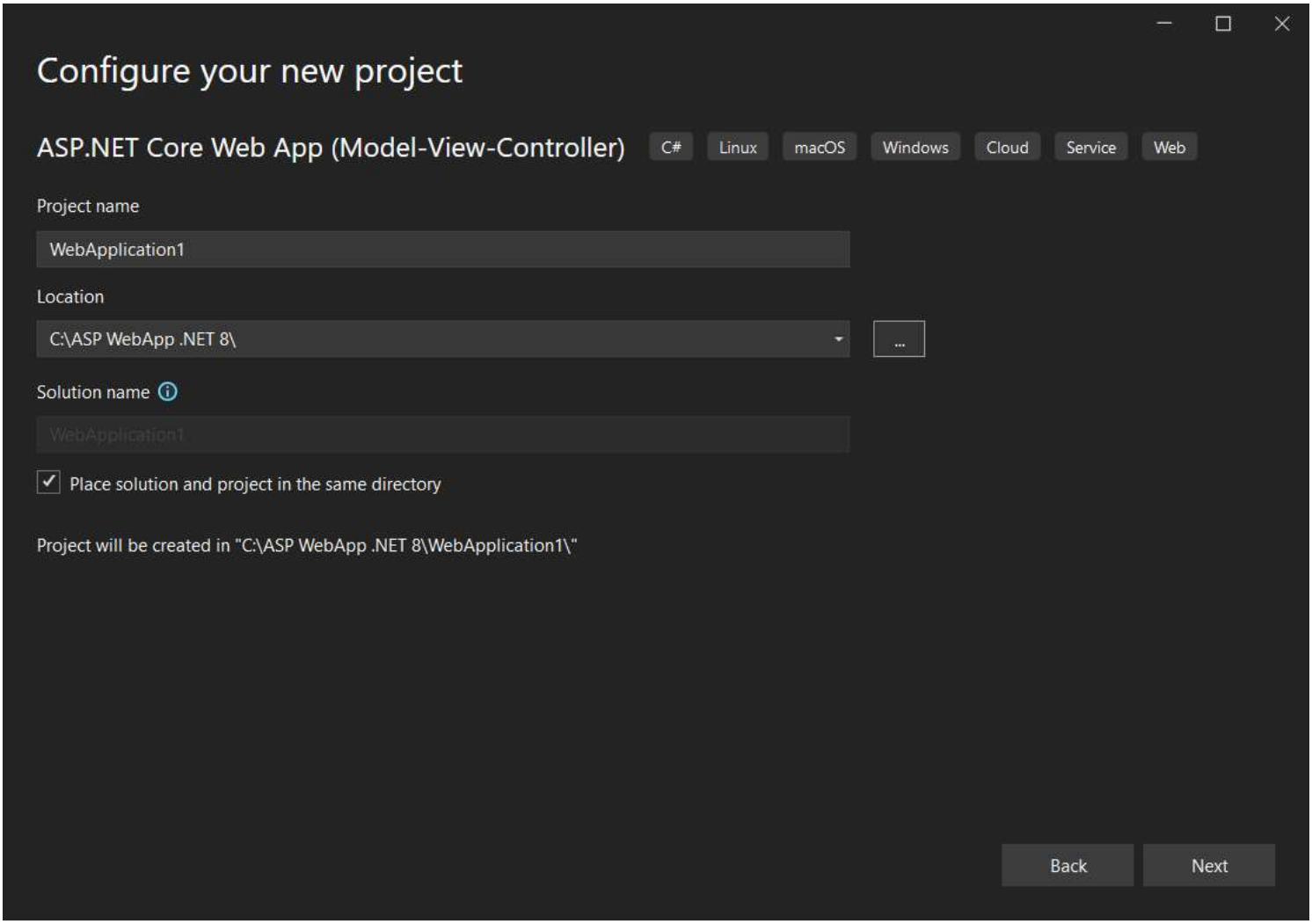
We run Visual Studio 2022 Community Edition and we create a new project



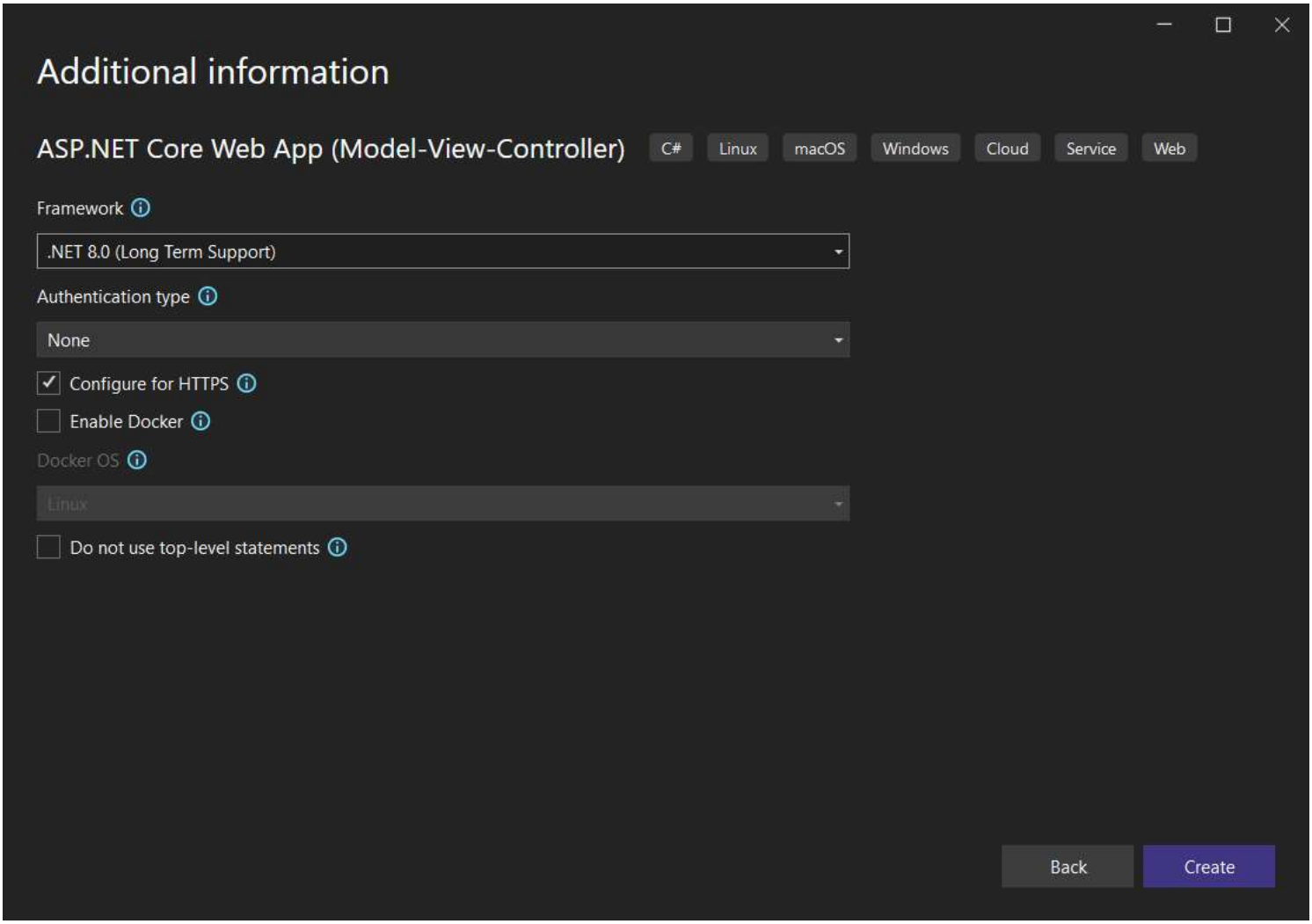
We select the project template



We input the location and the project name



We chose the new project main features



2. Project folder structure and main files description

2.1. Default Folder Structure

wwwroot

This folder contains static files like HTML, CSS, JavaScript, and image files. It's the root for web server requests.

Controllers

In an MVC (Model-View-Controller) application, this folder contains the controller classes. Controllers handle user input and interactions, and return responses.

Models

This folder holds the classes that represent the data of the application and the logic to manage that data. It's part of the MVC pattern.

Views

In MVC, the Views folder contains the Razor view files (.cshtml). These files combine HTML markup with C# code for dynamic rendering of content.

Properties

Contains configuration files, like launchSettings.json, which includes settings for launching the app, like environment variables and application URL.

Dependencies

This section in your solution explorer lists the packages, frameworks, and projects your app depends on.

Optional Folders

Areas: Used for organizing related functionality into a group, like admin panels.

Migrations: For Entity Framework Core, contains database migration files.

Data: Contains data access related code, often includes a **DbContext** class for **Entity Framework Core**.

3. Workflow after running the WebApp

This is the workflow summary:

Program.cs is the application entry point -> **https://localhost:44327/Home/Index** -> Call Index action inside the Home controller **Controllers/HomeController.cs** -> Call Home view (**Views/Home/Index.cshtml**) -> the Index.cshtl view is Rendered inside the **Views/Shared/_Layout.cshtml** as defined in **_ViewStart.cshtml**

3.1. Program.cs file is the entry point of the application

It sets up the web host and starts the application.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();
```

```
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, s
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

3.2. The application entry point (/Home/Index) calls the Index action inside the Home controller (Controllers/HomeController.cs)

```
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using WebApplication1.Models;

namespace WebApplication1.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }
```

3.3. The Index action inside the Home controller calls the Views/Home/Index.cshtml view

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps with ASP.N
</div>
```

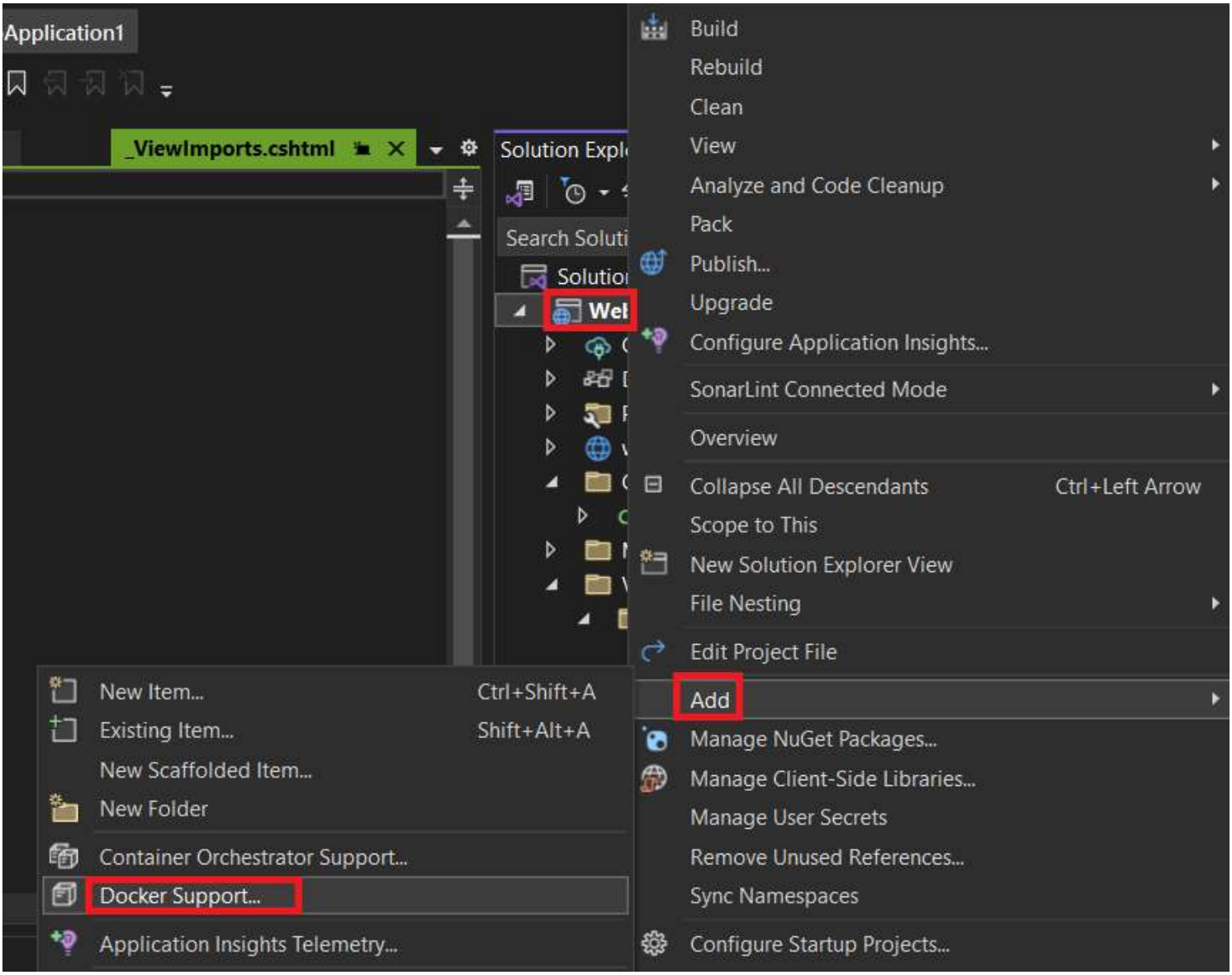
3.4. The Views/Home/Index.cshtml view is rendered inside the Views/Shared/_Layout.cshtml view

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplication1</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/WebApplication1.styles.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bot
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">WebA
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-ta
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-a
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-a
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
```

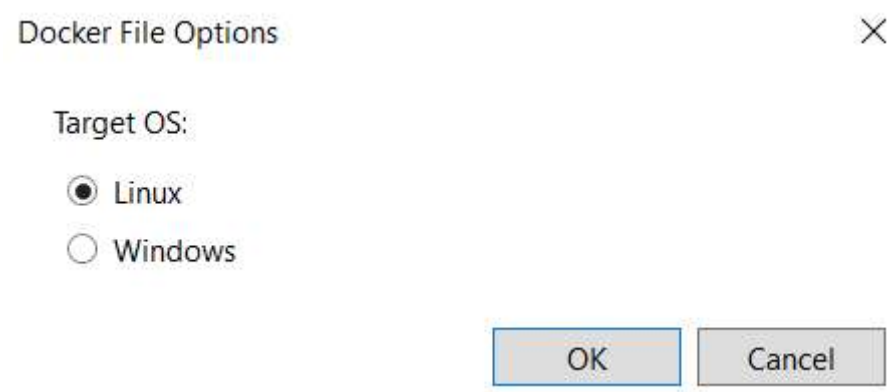
```
            <main role="main" class="pb-3">
                @RenderBody()
            </main>
        </div>

        <footer class="border-top footer text-muted">
            <div class="container">
                &copy; 2024 - WebApplication1 - <a asp-area="" asp-controller="Home" asp-action="Priv
            </div>
        </footer>
        <script src="~/lib/jquery/dist/jquery.min.js"></script>
        <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
        <script src="~/js/site.js" asp-append-version="true"></script>
        @await RenderSectionAsync("Scripts", required: false)
    </body>
    </html>
```

4. Adding docker support to the WebApp application

We right click on the project name and we select the menu option **Add->Docker support...**



We select the Linux operating system



This is the Dockerfile automatically created

```
#See https://aka.ms/customizecontainer to learn how to customize your debug container and how Vis
```

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["WebApplication1.csproj", "."]
RUN dotnet restore "./././WebApplication1.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./WebApplication1.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./WebApplication1.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHo

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

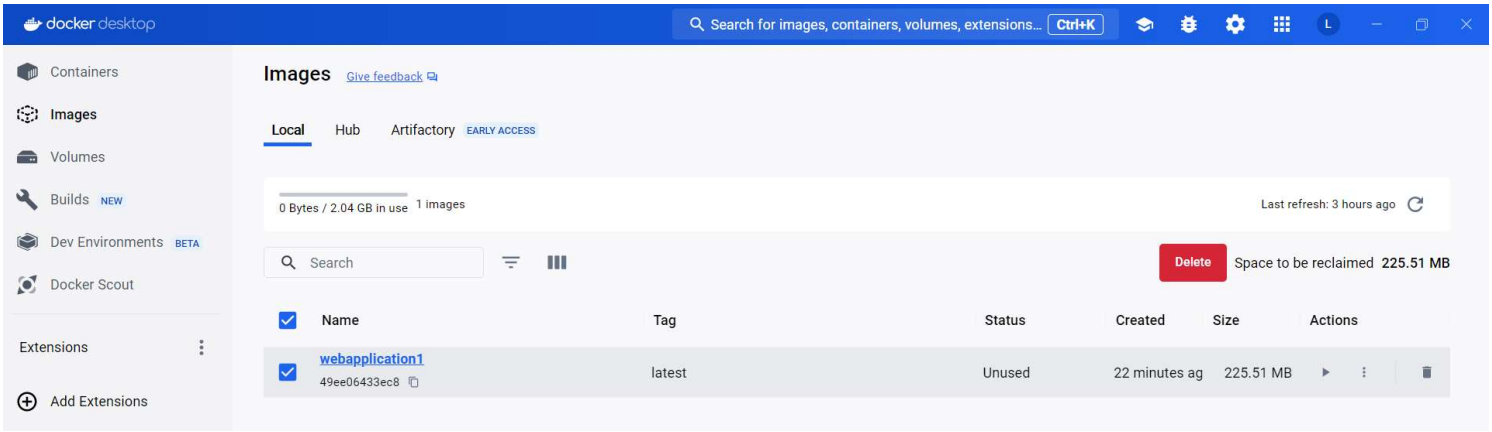
We right click on the project name and we select the option **Open in Terminal**

We build the WebApp docker image running this command

```
docker build -t webapplication1 .
```

```
PS C:\ASP WebApp .NET 8\WebApplication1> docker build -t webapplication1 .
[+] Building 0.8s (18/18) FINISHED                                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                                          0.1s
 => => transferring dockerfile: 892B                                                                                          0.0s
 => [internal] load .dockerignore                                                                                             0.1s
 => => transferring context: 464B                                                                                             0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:8.0                                                             0.3s
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:8.0                                                          0.4s
 => [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:8.0@sha256:7ef41132b2ebe6166bde36b7ba2f0d302e10307c3e0523a4539643a77233f56d  0.0s
 => [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:8.0@sha256:3ff67792728179308c4bf06799d8b45d155c60fddc347acf69465a496d9a20b8  0.1s
 => => resolve mcr.microsoft.com/dotnet/aspnet:8.0@sha256:3ff67792728179308c4bf06799d8b45d155c60fddc347acf69465a496d9a20b8      0.1s
 => [internal] load build context                                                                                             0.1s
 => => transferring context: 5.76kB                                                                                           0.1s
 => CACHED [base 2/2] WORKDIR /app                                                                                            0.0s
 => CACHED [final 1/2] WORKDIR /app                                                                                           0.0s
 => CACHED [build 2/7] WORKDIR /src                                                                                           0.0s
 => CACHED [build 3/7] COPY [WebApplication1.csproj, .]                                                                       0.0s
 => CACHED [build 4/7] RUN dotnet restore "././WebApplication1.csproj"                                                        0.0s
 => CACHED [build 5/7] COPY . .                                                                                               0.0s
 => CACHED [build 6/7] WORKDIR /src/.                                                                                         0.0s
 => CACHED [build 7/7] RUN dotnet build "./WebApplication1.csproj" -c Release -o /app/build                                  0.0s
 => CACHED [publish 1/1] RUN dotnet publish "./WebApplication1.csproj" -c Release -o /app/publish /p:UseAppHost=false        0.0s
 => CACHED [final 2/2] COPY --from=publish /app/publish .                                                                     0.0s
 => exporting to image                                                                                                        0.0s
 => => exporting layers                                                                                                       0.0s
 => => writing image sha256:49ee06433ec87681011aee59b999b46db94851f11cd588035afaca1fececfb2f                                  0.0s
 => => naming to docker.io/library/webapplication1                                                                            0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\ASP WebApp .NET 8\WebApplication1>
```

We can verify in Docker Desktop the new image

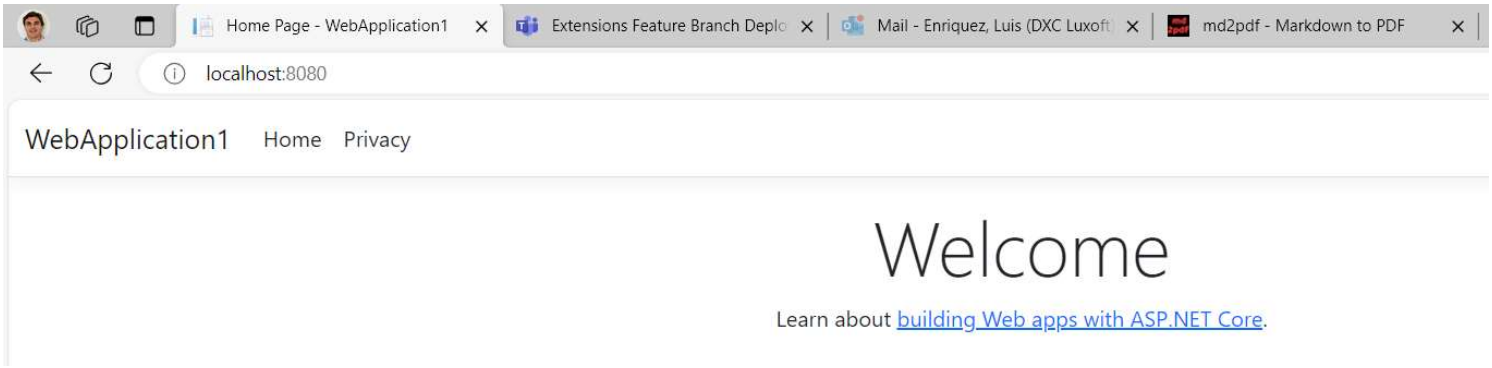


5. Run your application with protocol HTTP

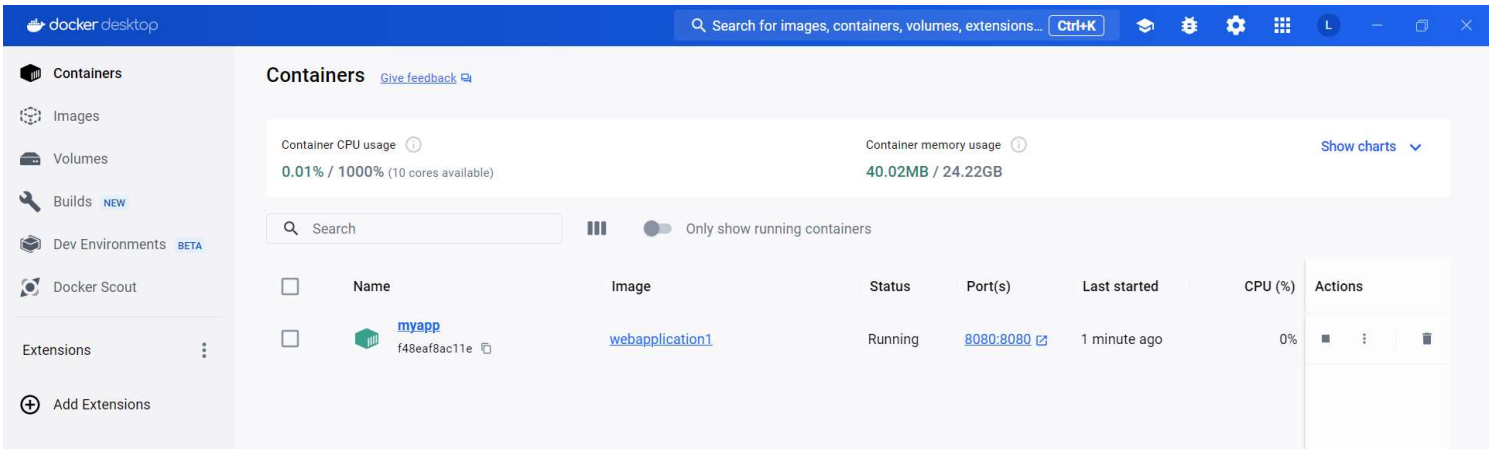
We run the docker image with this command

```
docker run -d -p 8080:8080 --name myapp webapplication1
```

We confirm the application is running in the endpoing: http://localhost:8080



We also see the running docker image in Docker Desktop



6. Run your application with protocol HTTPS

Creating and using a **self-signed certificate** for local development involves a few steps.

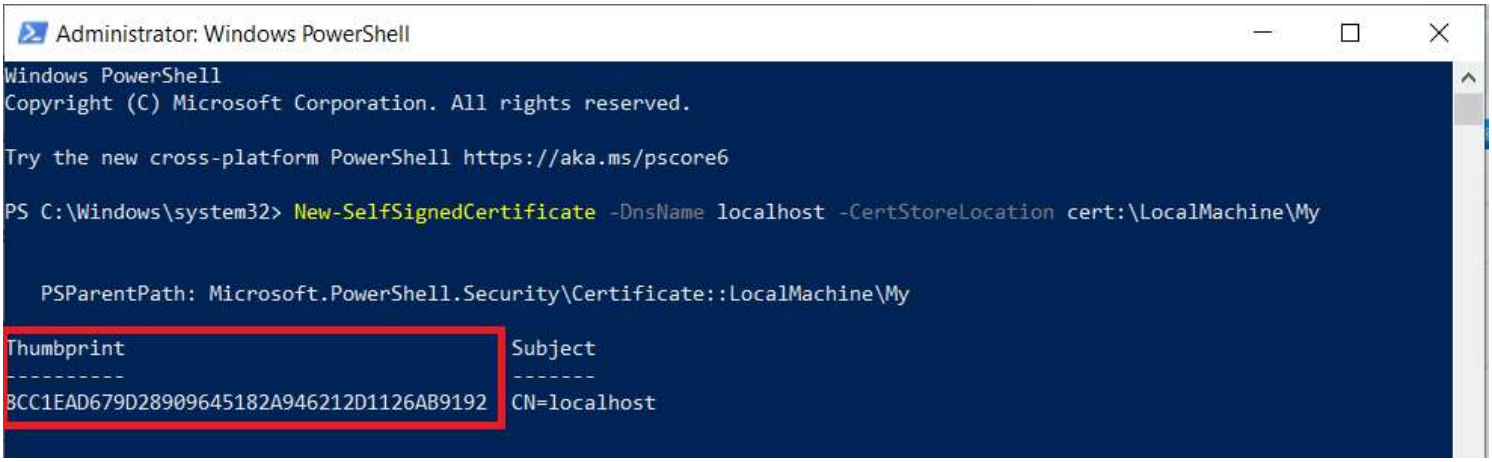
This process can vary slightly depending on your operating system. I'll outline the general steps and provide guidance for **Windows**:

6.1. Generate a Certificate Using PowerShell

You can use PowerShell to create a self-signed certificate:

```
New-SelfSignedCertificate -DnsName localhost -CertStoreLocation cert:\LocalMachine\My
```

We copy the **Thumbprint**. We need it to look for our certificate inside the Manage Certificates Computer application in the **Personal/Certificates** folder

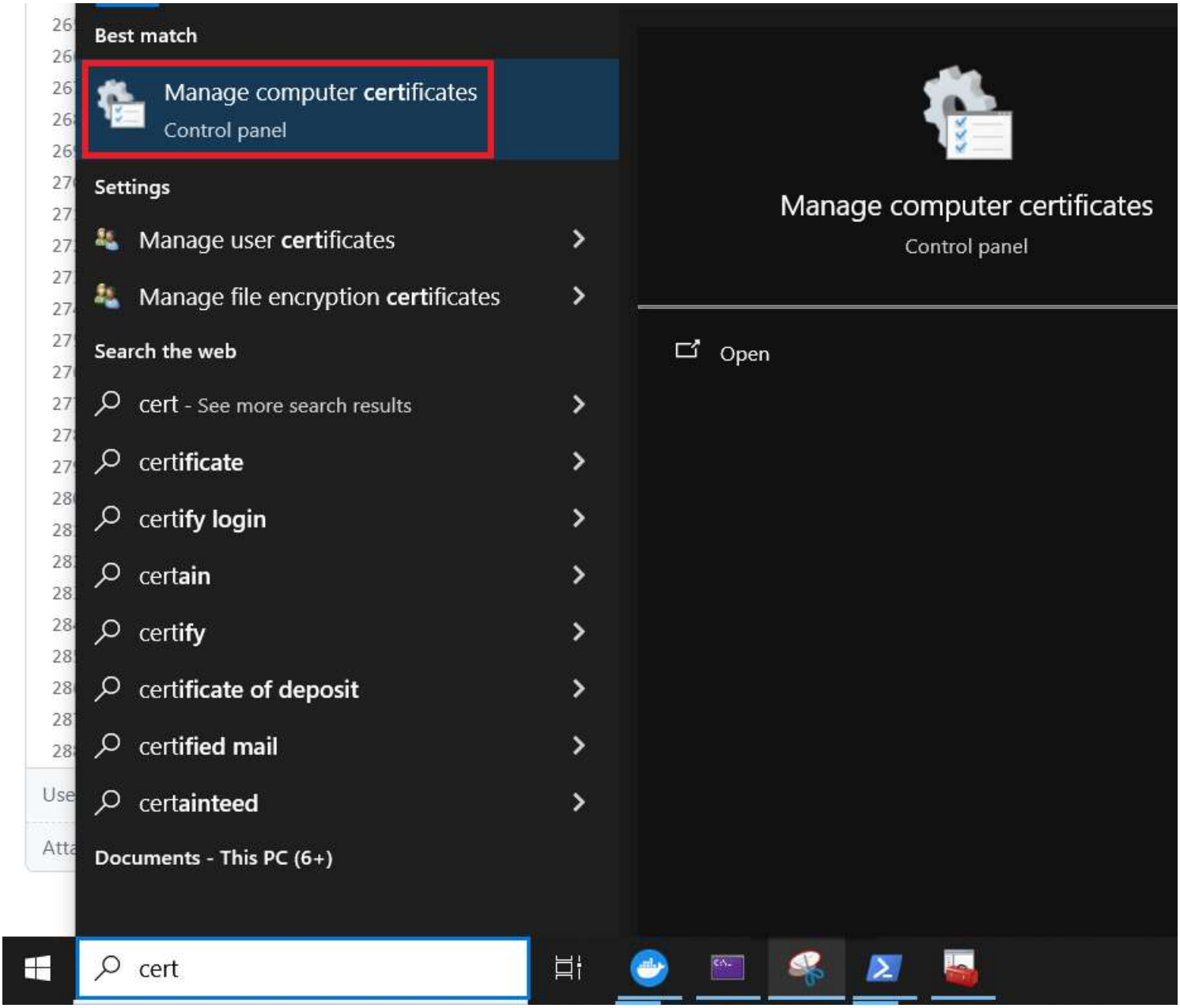


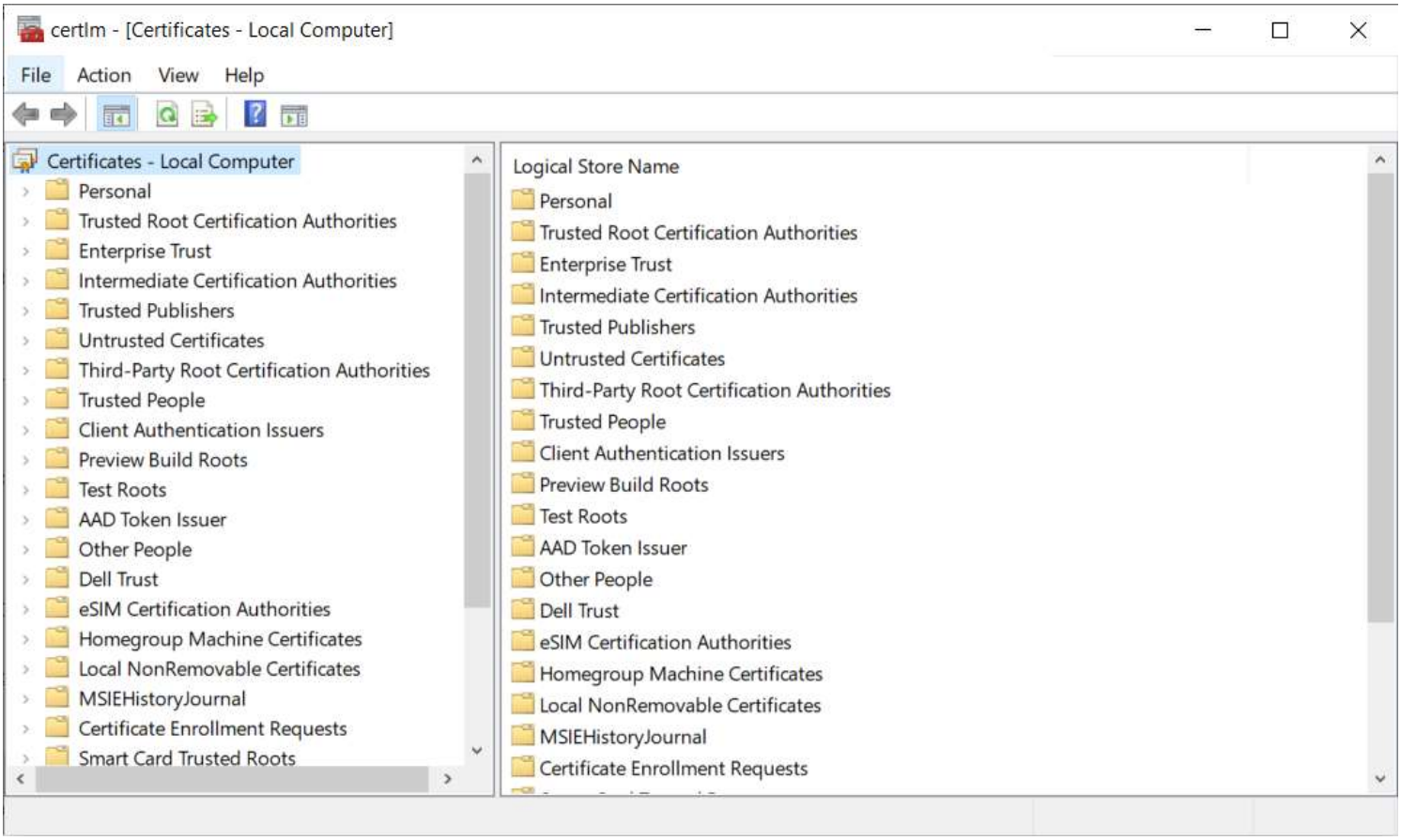
Thumbprint: 8CC1EAD679D28909645182A946212D1126AB9192

This command creates a certificate for localhost and places it in the local machine's certificate store.

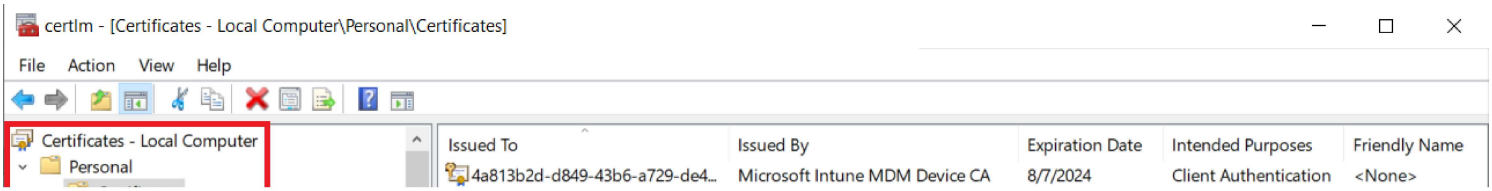
6.2. Export the Certificate

We look for the application **Manage Computer Certificates** (**Microsoft Management Console (MMC)**) and we run it

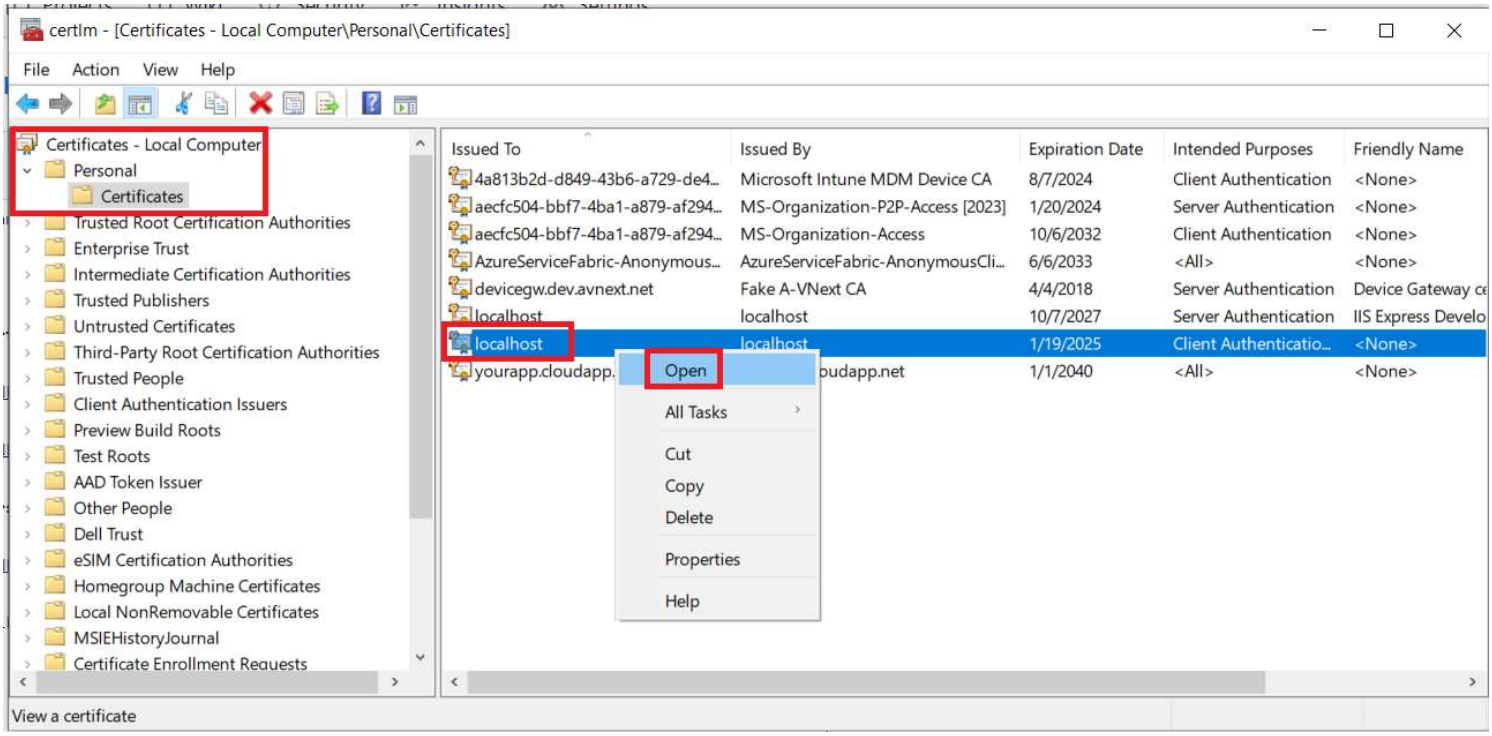

Best match
Manage computer certificates
Control panel
Settings
Manage user certificates
Manage file encryption certificates
Search the web
cert - See more search results
certificate
certify login
certain
certify
certificate of deposit
certified mail
certainteed
Documents - This PC (6+)
Manage computer certificates
Control panel
Open
cert

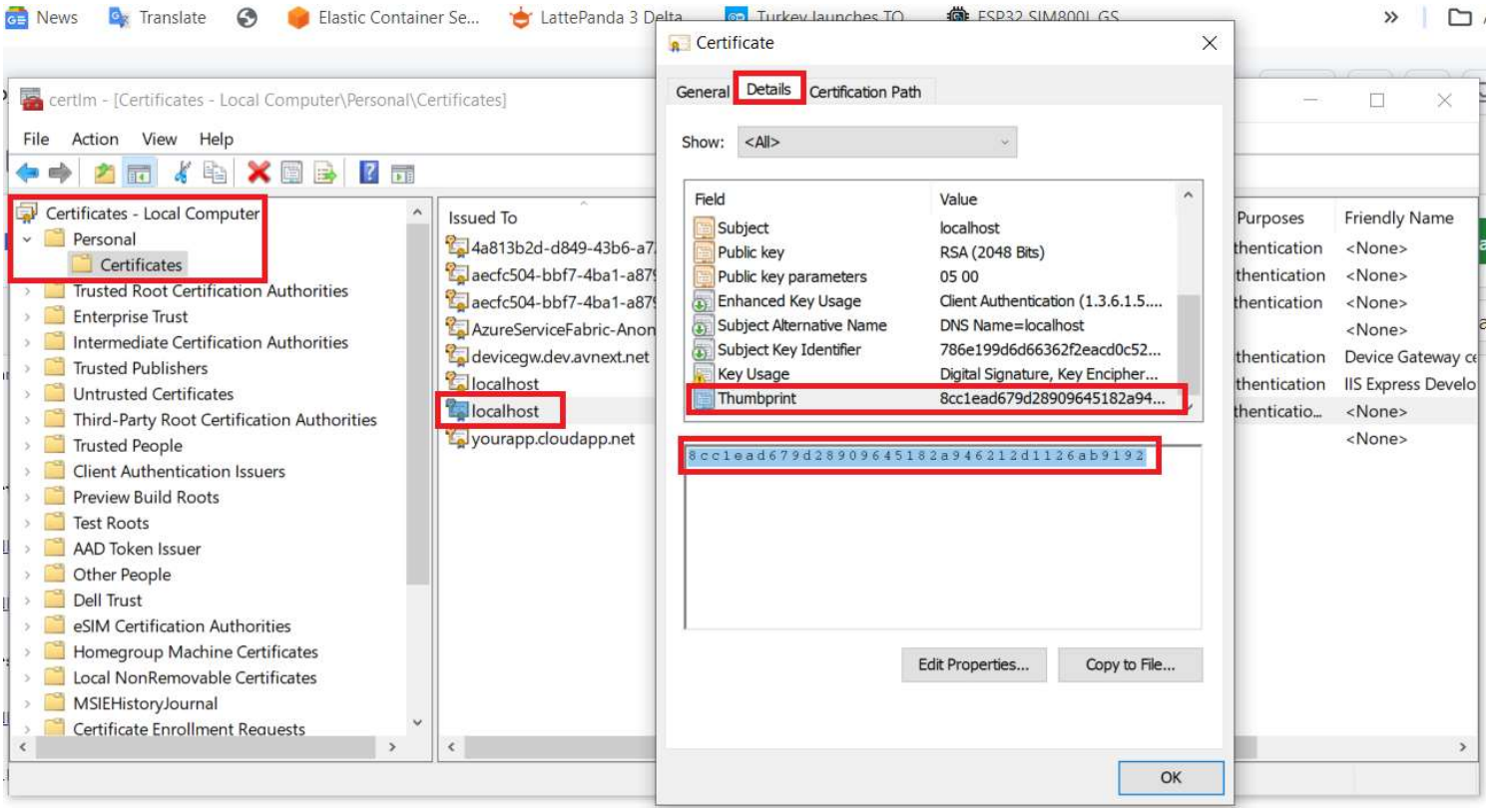


We search for the new certificate inside the **Personal/Certificates** folder, also **Issue To: localhot** and with the **Thumbprint**: 8CC1EAD679D28909645182A946212D1126AB9192



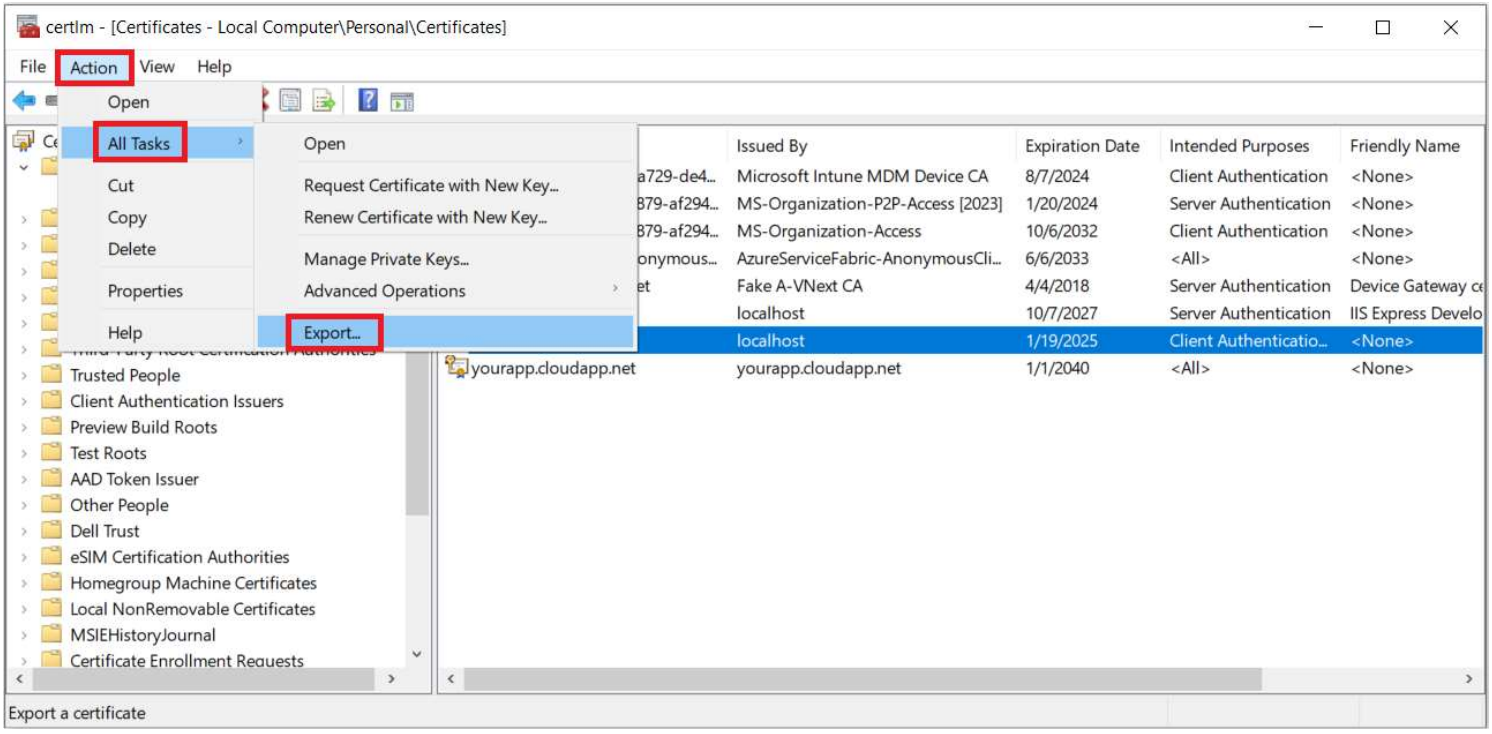
We also check the certificate **Thumbprint**: 8CC1EAD679D28909645182A946212D1126AB9192





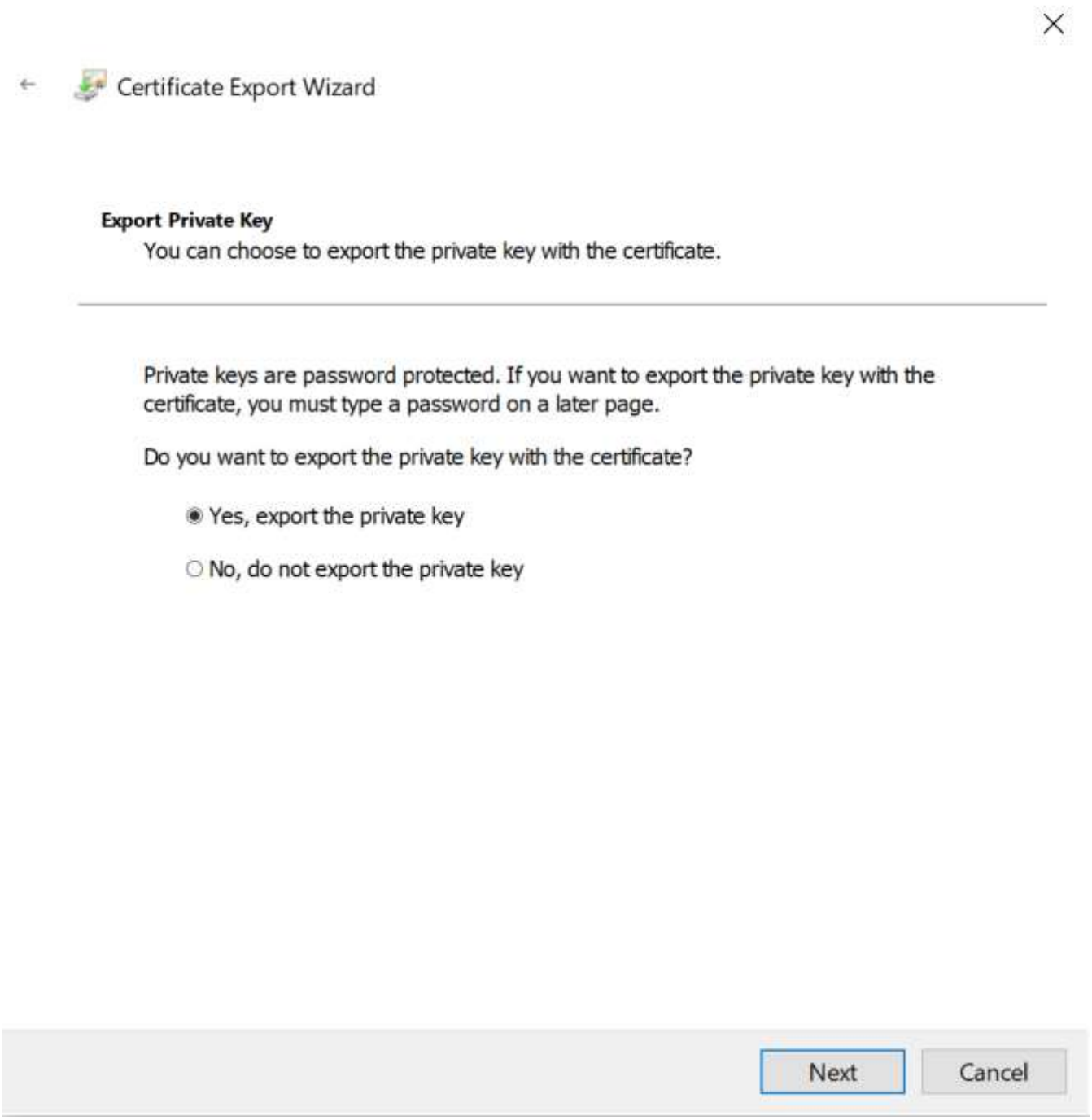
You need to **export the certificate** to a **.pfx** file with a password. You can do this through the **Microsoft Management Console (MMC)** or PowerShell.

We select the certificate and then select the menu option **Action->All Tasks->Export...**

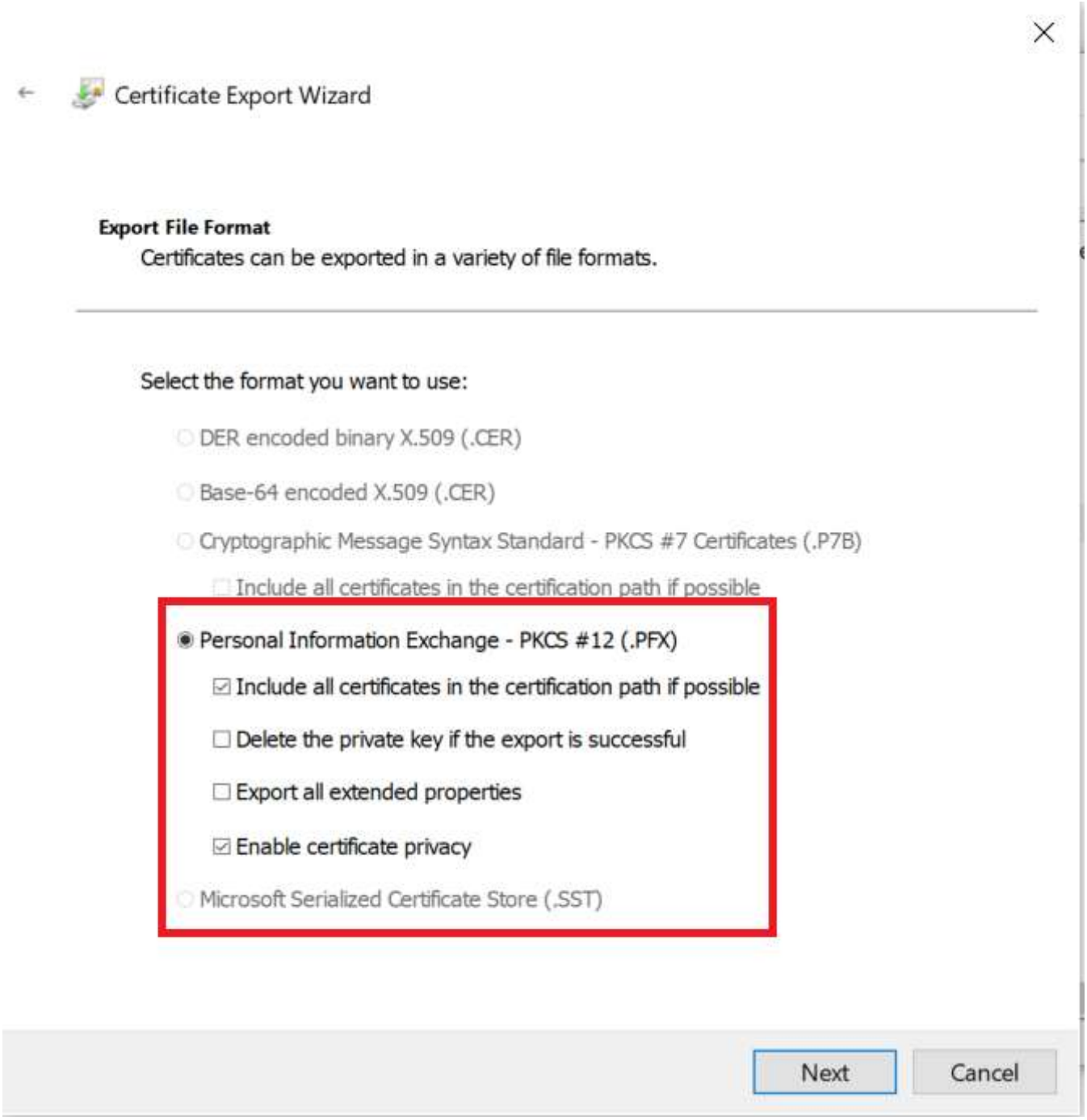


We click on the Next button

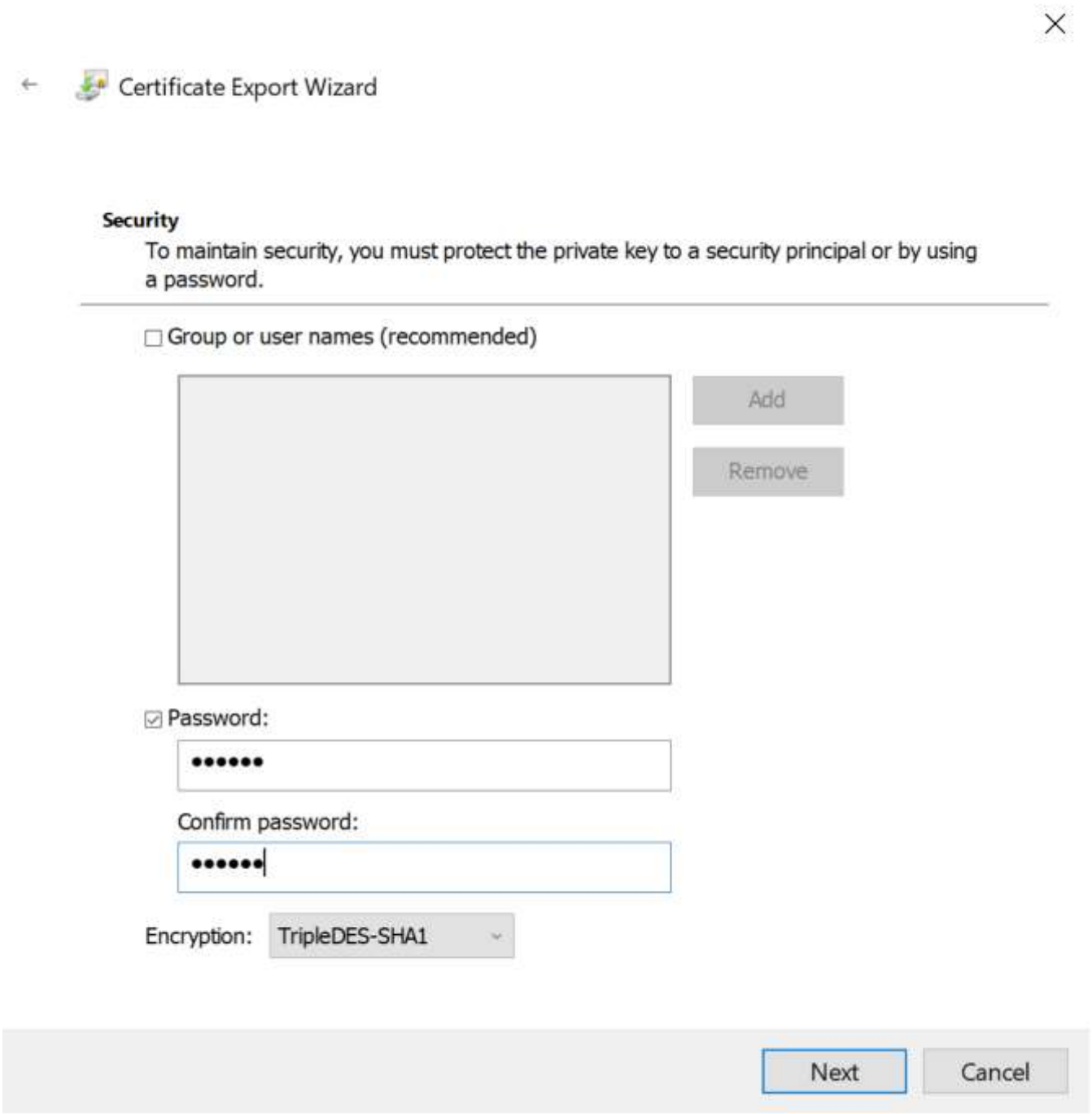
We select the option **Yes, export the private key**



We select the option Personal Information Exchange PFX



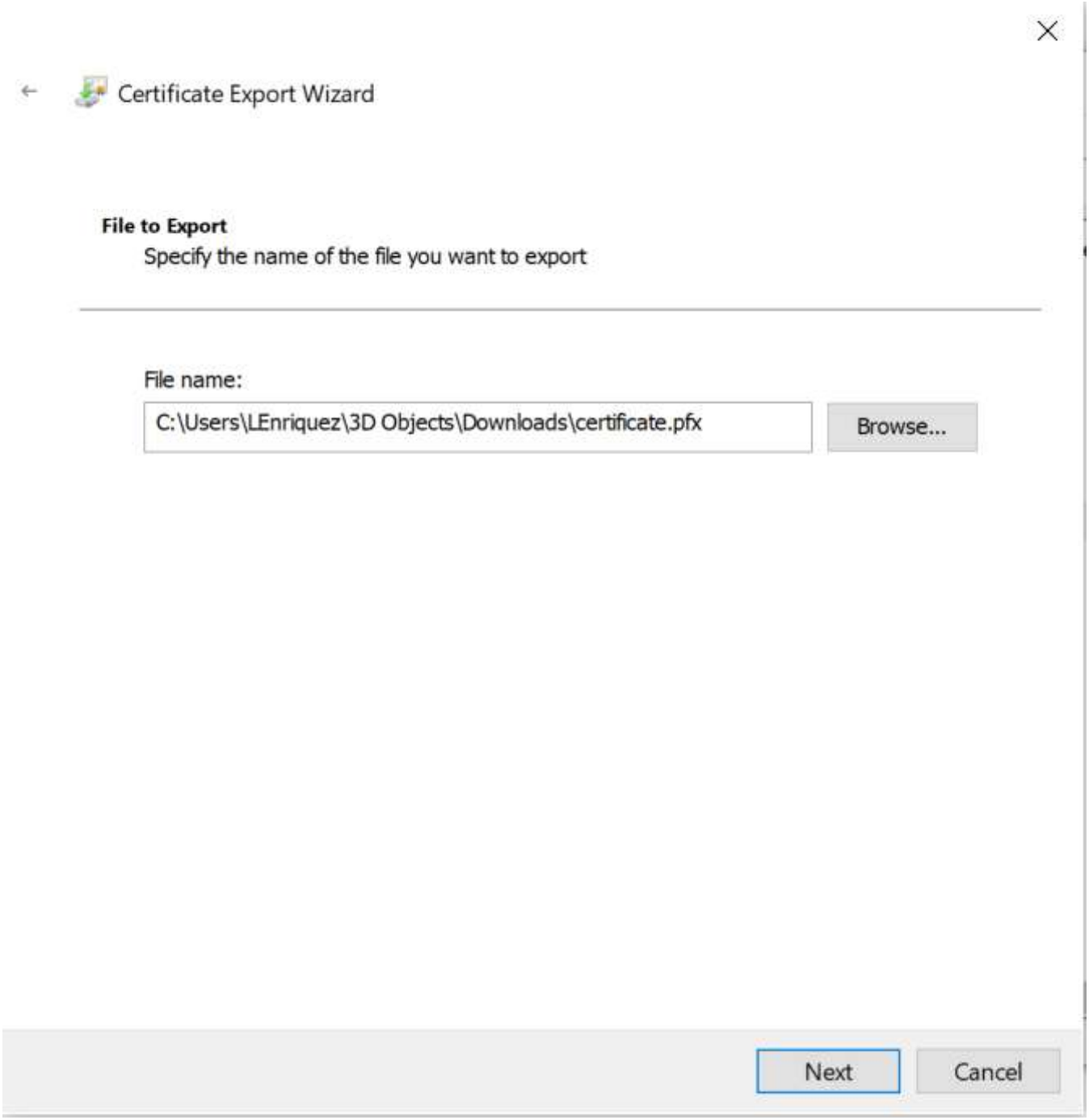
We select and input the Password: 123456



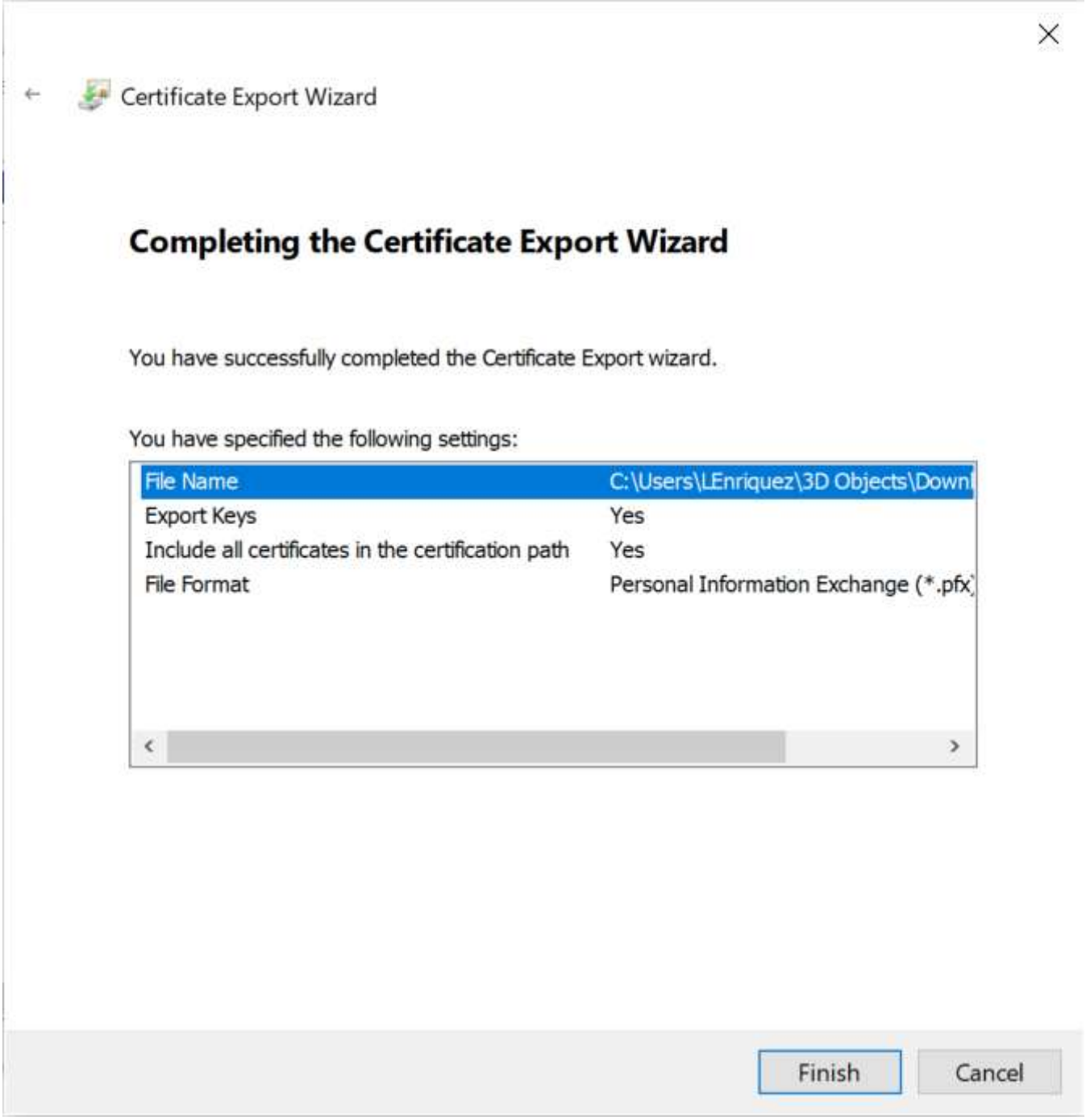
We press in the **Browser** button and select the certificate name and the location to store it

We press the **Save** button

We click on the **Next** button



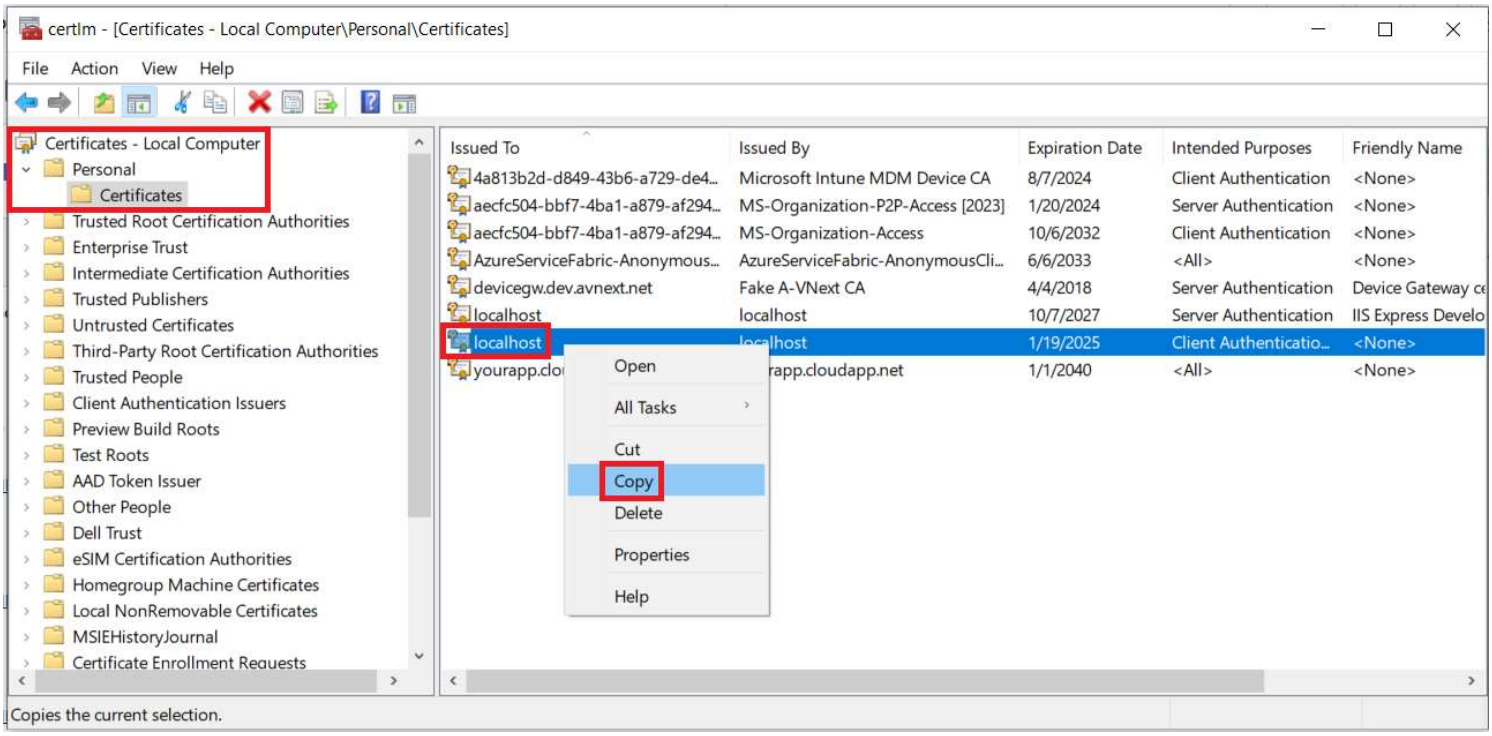
We press the **Finish** button



6.3. Trust the Certificate**

You can trust the certificate by adding it to the "**Trusted Root Certification Authorities**" store, either through MMC or PowerShell.

We select the certificate and right click on it and select the **copy** option



We select the **Certificates** folder inside the folder **Trusted Root Certification Authorities** and we right click on it and select the option **paste**

6.4. Configure ASP.NET Core (appsettings.json)

In your **appsettings.json** or programmatic configuration, specify the path to the .pfx file and the password you used during export:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Kestrel": {
    "Endpoints": {
      "Https": {
        "Url": "https://*:8081",
        "Certificate": {
          "Path": "certificate.pfx",
          "Password": "123456"
        }
      },
      "Http": {
```

```
      "Url": "http://*:8080"
    }
  }
}
}
```

6.5. Update the Dockerfile

Also modify the Dockefile to copy the certificate file into the Docker image: **COPY ["certificate.pfx", "."]**

```
# See https://aka.ms/customizecontainer to learn how to customize your debug container and how Vi

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["WebApplication1.csproj", "."]
RUN dotnet restore "./WebApplication1.csproj"
COPY . .
WORKDIR "/src"
RUN dotnet build "WebApplication1.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "WebApplication1.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .

# Copy the certificate file into the Docker image
COPY ["certificate.pfx", "."]

ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

6.6. Update the Program.cs file

For accessing both protocol HTTP and HTTPS comment this line: **app.UseHttpsRedirection();** in **Program.cs** file

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, s
    app.UseHsts();
}

//app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

6.7. Create the Docker image

We create the WebApp docker image running this command

```
docker build -t webapplication1 .
```

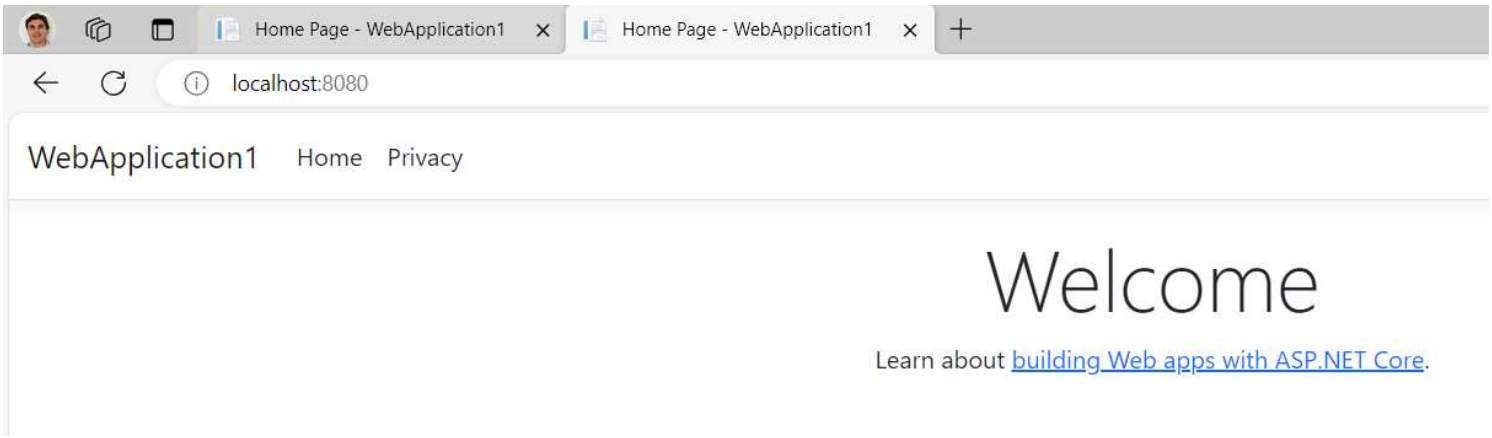
6.8. Run the Docker container

We execute the following command to run the docker container

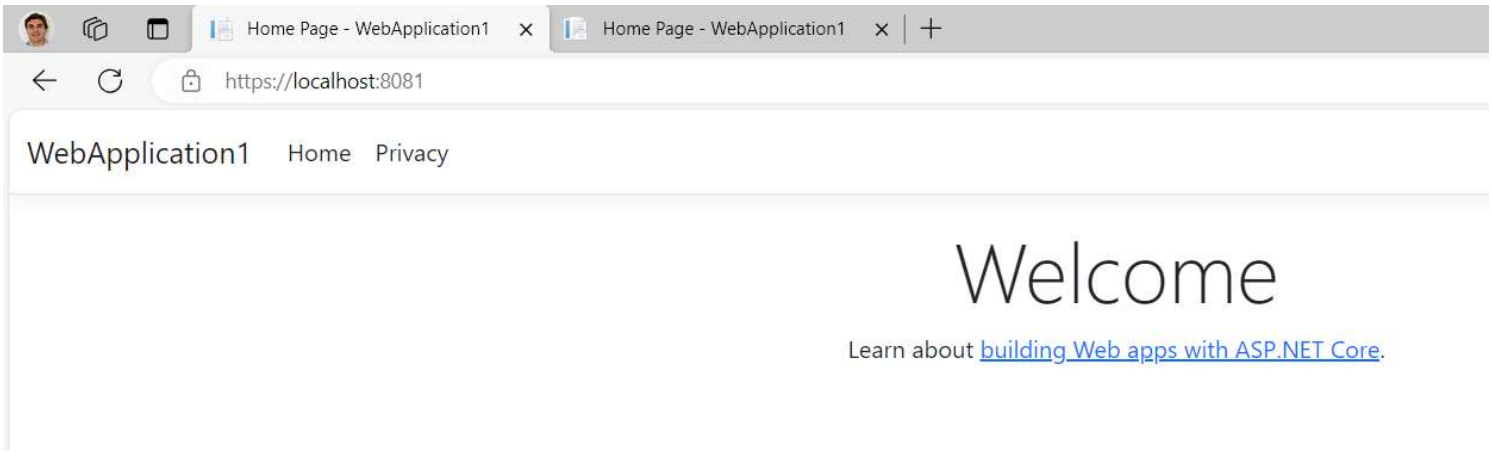
```
docker run -d -p 8080:8080 -p 8081:8081 --name myapp webapplication1
```

6.9. Verify the application endpoints

http://localhost:8080



https://localhost:8081



We also can verify the running container in the Docker Desktop