# LABS

Cílem cvičení je vyzkoušet si jednotlivé části ASP.NET Core aplikace na reálných příkladech.

Vzhledem k složitosti úloh, mohou být některá uvedená řešení zkrácena a nebýt tak kompletním řešením problému.

# LAB – CLI – dotnet version

## Zadání úkolu:

1. Zapněte powershell
2. Vytvořte novou složku Bratislava20191202
   1. *mkdir Bratislava20191202*
   2. *cd .\Bratislava20191202*
3. Otevřete nápovědu
   1. *dotnet --help*
4. Vyzkoušejte si různé příkazy
   1. *dotnet --info*
   2. *dotnet --list-sdks*
   3. *dotnet --list-runtimes*
5. Zjistěte verzi SDK, která se teď použije
   1. *dotnet version*
6. Vytvořte nový globaljson soubor a upravte verzi na 3.0.100
   1. *dotnet new globaljson*
   2. *code .\global.json*
   3. Změňte version na 3.0.100
7. Zjistěte verzi SDK, která se nově použije
   1. *dotnet version*

# LAB – CLI – create .NET project

## Zadání úkolu:

1. Vytvořte novou konzolovou aplikaci do složky ConsoleApp
   1. *dotnet new console --output ConsoleApp –name ConsoleApp*
2. Otevřete složku s konzolovou aplikací
   1. *cd ConsoleApp*
3. Proveďte build aplikace
   1. *dotnet build .\ConsoleApp.csproj*
4. Vyzkoušejte spuštění aplikace
   1. *dotnet run*
5. Otevřete Program.cs a upravte hlášku, která se vypíše do konzole
   1. *code Program.cs*
   2. Upravte zprávu na Hello Bratislava
6. Vyzkoušejte spuštění aplikace
   1. *dotnet run*

# LAB – CLI – create ASP.NET project

## Zadání úkolu:

1. Vytvořte novou webovou aplikaci do složky WebApp
   1. *dotnet new mvc --name WebApp --output WebApp*
2. Proveďte restore nuget balíčků
   1. *dotnet restore*
3. Proveďte build aplikace
   1. *dotnet build* (implicitně spouští i dotnet restore, předchozí krok je tak možné vynechat)
4. Vyzkoušejte spuštění aplikace
   1. *dotnet run*
   2. Now listening on: <http://localhost:5000> -> redirect to <https://localhost:5001/>
5. Ověřte a případně přidejte https certifikát pro development
   1. Možná se zobrazila hláška **Your connection to this site is not secure / private**
   2. Ověření zda existuje validní certifikát
      1. *dotnet dev-certs https -c*
      2. *dotnet dev-certs https -v*
   3. Druhou možností je ověřit, zda existuje certifikát pro localhost v **certmgr**
   4. Přidání certifikátu
      1. *dotnet dev-certs https –trust*
6. Automatické přesměrování na HTTPS
   1. *code Startup.cs*
   2. Ověřte, zda obsahuje řádek **app.UseHttpsRedirection();**
   3. Zakomentujte **app.UseHttpsRedirection();**
   4. Zkuste znovu spustit aplikaci
      1. *dotnet run* -> Now listening on: <http://localhost:5000> -> NO redirect

# LAB – CLI – SOlution

## Zadání úkolu:

1. Ověřte, že se nacházíte v rootu projektu
   1. *cd .. , …*
   2. *pwd* (zobrazí celou cestu do současné složky)
2. Vytvořte novou solution pro konzolovou aplikaci a webový project
   1. *dotnet new sln*
3. Ověřte, jaké projekty se nacházejí v solution
   1. *code Bratislava20191202.sln*
4. Solution by neměla obsahovat žádné projekty
   1. NEobsahuje Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}")
5. Přidejte projektu do solution
   1. *dotnet sln add .\ConsoleApp\ConsoleApp.csproj*
   2. *dotnet sln add .\WebApp\WebApp.csproj*
6. Solution by měla obsahovat nově přidané projekty
   1. *code Bratislava20191202.sln*

# LAB – .NEt tools

## Zadání úkolu:

1. Vytvořte novou konzolovou aplikaci pojmenovanou GlobalTool
2. Přidejte konzolovou aplikaci do solution
3. Otevřete Program.cs
   1. *code Program.cs*
4. Upravte Console.WriteLine
   1. Změnit na libovolný text
5. Nastavte konzolovou aplikaci jako tool
   1. *code GlobalTool.csproj*
   2. Přidejte do elementu PropertyGroup
      1. *<PackAsTool>true</PackAsTool>*
      2. *<ToolCommandName>myglobaltool</ToolCommandName>*
      3. *<PackageOutputPath>./nupkg</PackageOutputPath>*
6. Vytvořte z projektu NuGet balíček
   1. *dotnet pack*
7. Nainstalujte balíček jako globální nástroj
   1. *dotnet tool install --global --add-source ./nupkg myglobaltool*
8. Ověřte, že je nástroj nainstalován
   1. *dotnet tool list -g*
9. Spusťte global tool
   1. *myglobaltool*

# LAB – Code generation

## Zadání úkolu:

1. Nainstalujte nástroj aspnet-code generator
   1. *dotnet tool install --global dotnet-aspnet-codegenerator*
2. Oveřte, že je balíček úspěšně nainstalován
   1. *dotnet tool list -g*
3. Otevřete složku s webovou aplikací WebApp
4. Vygenerujte nový controller
   1. *dotnet aspnet-codegenerator -name TestController -outDir Controller*
5. Pokud došlo k chybě, nainstalujte potřebný NuGet balíček do projektu
   1. *dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design*
6. Vyzkoušejte přidání controlleru znovu
7. Ověřte, že byl do složky Controllers vygenerovaný nový TestController

# LAB – NuGet

## Zadání úkolu:

1. Vytvořte novou knihovnu (classlib)
2. Přidejte knihovnu do solution
3. Přidejte novou třídu StringExtensions
   1. dotnet new class
   2. Couldn't find an installed template that matches the input, searching online for one that does...
   3. Šablona třídy **neexistuje** a je potřeba jí přidat ručně pomocí vašeho oblíbeného nástroje
4. Otevřete solution pomocí **Visual Studio 2019**
5. Implementujte do třídy extension metodu **WordCount**, která přijme string, rozdělí ho po slovech a vrátí počet nalezených slov.
6. Vytvořte z knihovny NuGet balíček (pomocí CLI nebo nastavením Generate NuGet package on build)
   1. dotnet pack -c Release
7. Pro lokální otestování balíčku, vytvořte novou složku NuGet a vložte do ní vytvořený NuGet balíček
   1. Doporučuji vytvořit jednu složku pro všechny lokální balíčky. Složka NuGet by tak měla být mimo jednotlivé repozitáře/projekty.
8. Zvolenou cestu přidejte jako zdroj NuGet balíčků
   1. Otevřete NuGet Package Manager
      1. Například: Tools -> NuGet Package Manager -> Manage NuGet packages for solution
   2. Otevřete nastavení Package source pomocí ozubeného kola
   3. Přidejte vytvořenou složku jako zdroj pro NuGet balíčky (zdroj pojmenujte Local NuGets)
9. Nainstalujte balíček do vytvořeného webového projektu
   1. dotnet add package NAME -s PATH
   2. nebo
   3. Visual Studio -> Manage NuGet Packages
10. Vyzkoušejte použití balíčku
    1. Pro jednoduchost třeba přímo v Program.cs

# LAB – Dependency injection

## Zadání úkolu:

Vytvořte jednoduchou konzolovou aplikaci na které si vyzkoušíte práci s Dependency Injection v .NET Core. Tato aplikace bude využívat tříd ServiceCollection a ServiceProvider, pro konfiguraci dependency injection.

Zaregistruje do service collection první service, která bude sloužít jako composition root, tedy jako vstupní bod do celé naší aplikace. Tato service bude mít pouze jednu metodu a to metodu void Start(). Obsahem této metody bude vypsání textu **Hello World!** do konzole a zavolání další service, která bude do composition root vložena pomocí DI.

## Řešení:

1. Vytvořte novou konzolovou aplikaci **Console App (.NET Core).**
   1. Aplikaci pojmenujte **ConsoleAppWithoutHost**.
2. Nainstalujte potřebný NuGet balíček
   1. Microsoft.Extensions.DependencyInjection
3. Otevřete soubor **Program.cs** a v něm funkci Main()
4. Přidejte using
   1. using Microsoft.Extensions.DependencyInjection;

# LAB – Console app

## Zadání úkolu:

Rozšiřte aplikaci tak, že bude umět pracovat se základní konfigurací aplikace.

1. Načíst konfiguraci ze souboru appsetting.json
2. Načíst obsah environment variables
3. Použít konfiguraci environment variables k načení další konfigurace

## Řešení:

1. Přidejte do aplikace konfigurační soubory **appsetting.json**
   1. Klikněte pravým tlačítkem na projekt -> Add -> New Item
   2. Vytvořte soubor typu JSON File
   3. Pojmenujte ho **appsettings.json**
   4. Vložte do souboru

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

}

}

1. Vytvořte další json soubor pojmenovaný appsettings.Development.json
   1. Vložte do souboru

{

"Logging": {

"LogLevel": {

"Default": "Debug",

"System": "Information",

"Microsoft": "Information"

}

}

}

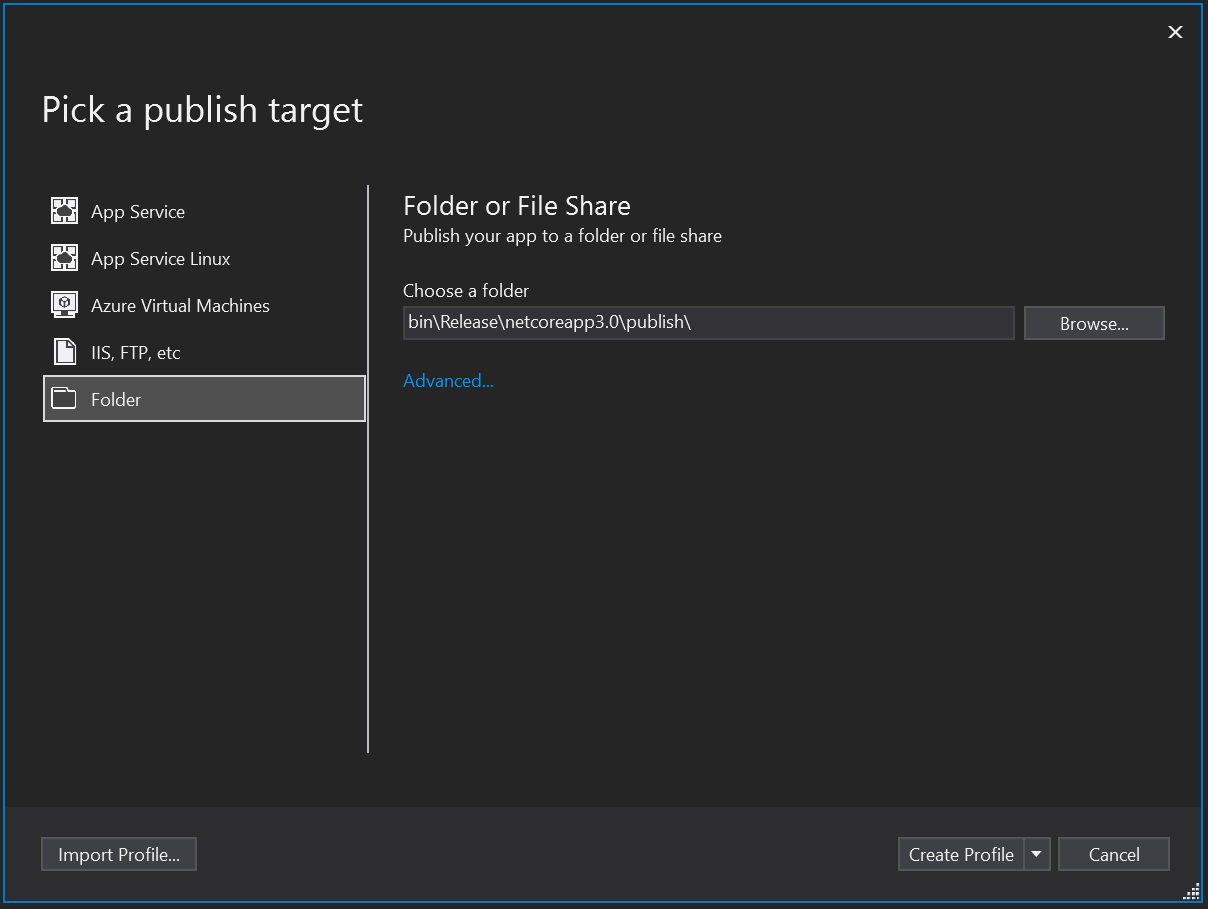
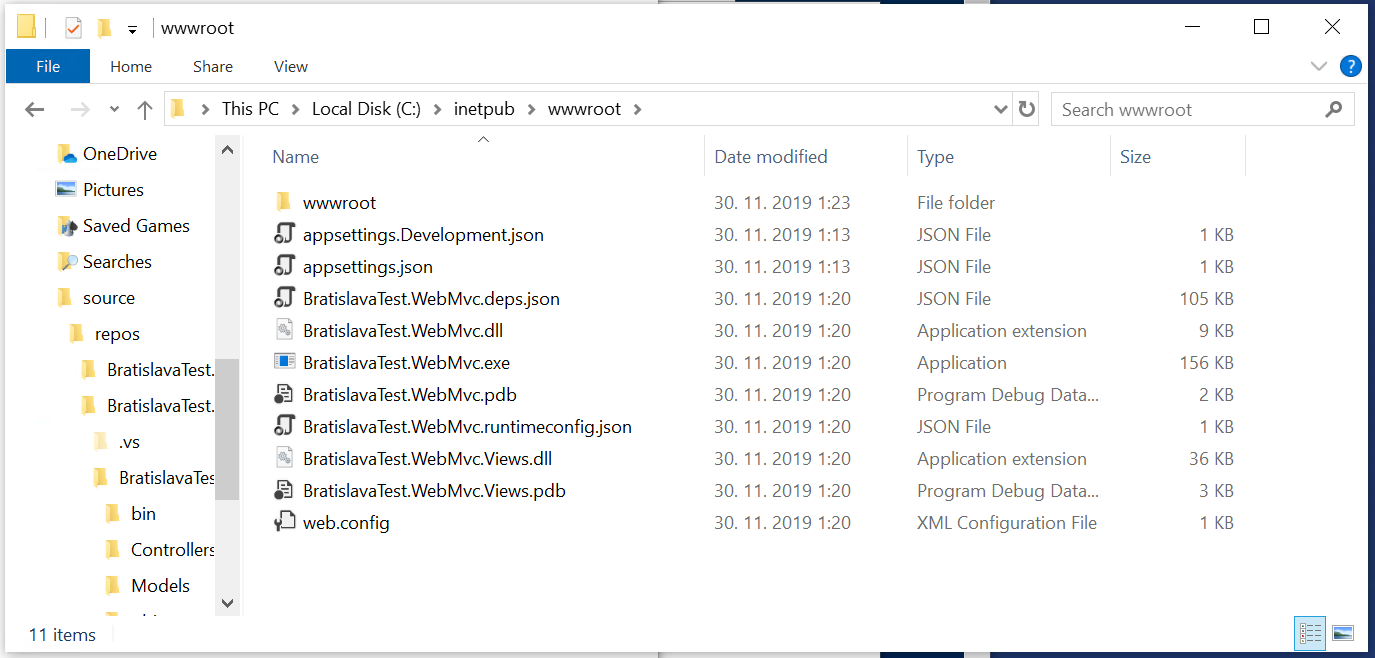
1. Vytvořte nový configuration builder
   1. Nainstalujte potřebný NuGet
      1. Microsoft.Extensions.Configuration
   2. Přidejte using Microsoft.Extensions.Configuration;
   3. Vytvořte novou instanci třídy configurationBuilder
      1. var configurationBuilder = new ConfigurationBuilder();
2. Pomocí třídy ConfigurationBuilder načtěte obsah JsonSouboru
   1. Nainstalujte potřebný NuGet balíček
      1. Microsoft.Extensions.Configuration.Json
      2. **Není** potřeba přidávat žádný nový using.
   2. Zavolejte nad configurationBuilder
      1. .SetBasePath(Directory.GetCurrentDirectory())
      2. .AddJsonFile("appsettings.json", false, false);
3. Načtěte nastavení environment variables
   1. Nainstalujte další NuGet balíček
      1. Microsoft.Extensions.Configuration.EnvironmentVariables
   2. Zavolejte nad configurationBuilder
      1. .AddEnvironmentVariables()
4. Načtěte nastavení z paměti
   1. .AddInMemoryCollection(new[] KeyValuePair.Create("ASPNETCORE\_ENVIRONMENT", "Development") }); // mock setting environment variable
5. Použijte ConfigurationBuilder k vytvoření IConfiguration
   1. IConfiguration configuration = configurationBuilder.Build();
6. Zapněte aplikaci a podívejte se na obsah proměnné *configuration*
   1. Pokud vám aplikace vyhodí chybu, je potřeba nastavit souboru *Copy to Output Directory* na *Copy if newer*
7. Načtěte hodnotu ASPNETCORE\_ENVIRONMENT *z konfigurace*
   1. var environment = configuration["ASPNETCORE\_ENVIRONMENT"];
8. Použijte získanou hodnotu k načtení dalšího appsettings.{environment}.json
9. Zaregistrujte configuration jako Singleton do service collection
   1. services.AddSingleton(configuration);
10. Díky tomuto řešení, je nyní možné si IConfiguration kdekoliv dotáhnout pomocí DI

# LAB – IIS

## Zadání úkolu:

Vytvořte webovou aplikaci v .NET Core (MVC). Aplikaci nasaďte na lokální IIS. Upravte host file tak, aby byla aplikace dostupná pod URL adresou **myaspnetapp.com.**

## Řešení:

1. Otestujte, zda Vám běží lokální IIS.
   1. Otevřete Internet Information Services manager
   2. V levé části rozbalte Site -> Default Web site
   3. V pravé části klikněte na Browse \*80 (http)
   4. Měla by se otevřít v prohlížeči defaultní IIS šablona
      1. <http://localhost/>
      2. C:\inetpub\wwwroot\iisstart.htm
2. Vytvořte novou webovou aplikaci. Případně můžete použít již existující.
3. Proveďte publish aplikace do lokální složky
   1. V horní liště Visual Studio 2019 klikněte na **Build**
   2. Publish [App Name]
   3. Otevře se dialog Pick a publish target
      1. Zvolte v levé části target **Folder**
      2. 
   4. Advanced
   5. Zvolte **Deployment Mode** na **Framework Dependent** nebo **Self-Container**
   6. Save
   7. Create Profile
   8. Rename
      1. Přejmenujte profil třeba na **Local IIS – Folder profile**
   9. Publish
4. Otevřete složku s nově vytvořenou aplikací
   1. Klikněte pravým tlačítkem myši na Project
   2. Open Folder in File Explorer
      1. Bin\Release\netcoreapp3.0\publish
   3. V druhém okně si otevřete File Explorer se složkou pro Default App
      1. C:\inetpub\wwwroot
   4. Nakopírujte celý obsah ze složky publish do složky wwwroot
   5. 
5. Vyzkoušejte znovu otevřít stránku <http://localhost/>
6. Upravte hosts file tak, aby bylo možné přistoupit na stránku přes adresu myaspnetapp.com
   1. Do souboru **C:\Windows\System32\drivers\**etc přidejte

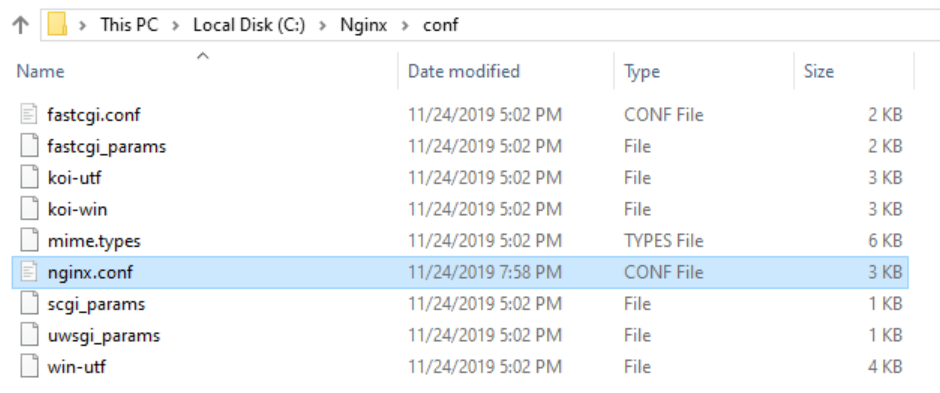
127.0.0.1 myaspnetapp.com

1. Vyzkoušejte otevřít stránku <http://myaspnetapp.com>

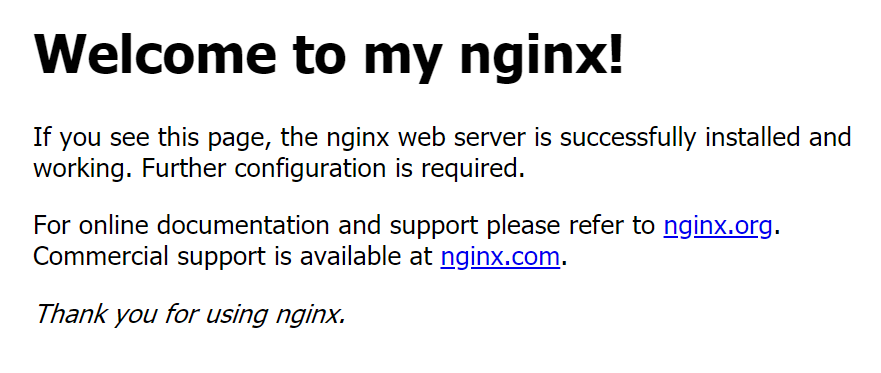
# LAB – Nginx

## Zadání úkolu:

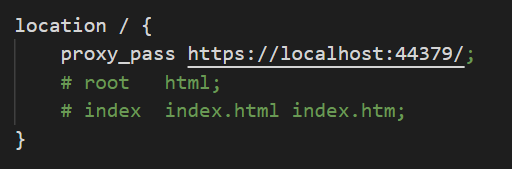
1. Nainstalujte Nginx pro Windows <http://nginx.org/en/download.html>
2. Rozbalte nginx do složky **C:\Nginx**



1. Přidejte složku C:\Nginx do environment variable PATH
2. Otevřete powershell nebo jinou konzoli
3. Otestujte, zda instalace proběhla v pořádku
   1. *nginx -h*
4. *Upravte nginx.conf*
   1. *Na řádku listen 443 změňte port na* ***5005***
5. Otestujte, že je nginx konfigurace v pořádku
   1. *nginx -t*
   2. nginx: the configuration file C:\nginx\nginx-1.17.6/conf/nginx.conf syntax is ok
6. Spusťte nxinx s defaultním nastavením
   1. Otevřete složku C:\Nginx
   2. Zapněte nginx
7. Ověřte, že aplikace v pořádku běží
   1. <http://localhost:5005/>



1. *S*pusťte Vaší webovou aplikaci
2. Upravte konfiguraci nginx v souboru C:\Nginx\conf\nginx.conf
   1. V sekci location zakomentujte nastavení root a index
   2. Přidejte nové nastavení proxy\_pass <https://localhost:PORT_APLIKACE/>



* 1. Uložte změny

1. Aktualizujte konfiguraci nginx
   1. nginx -s reload
2. Otevřete <https://localhost:5005/>. Váš požadavek by měl být přesměrován na spuštěnou webovou aplikaci.

# LAB – Docker – .NET Core

## Zadání úkolu:

1. Přidejte nový projekt Console App (.NET Core)
2. Přidejte do projektu docker file
   1. Pravé tlačitko myši na nově vytvořený projekt
   2. Kliknout na **Add**
   3. Kliknout na **Docker Support**
3. Zvolte variantu **Linux**
4. Smažte celý obsah souboru
5. Vložte do souboru
   1. FROM mcr.microsoft.com/dotnet/core/sdk:3.0-buster AS build
   2. WORKDIR /src
   3. COPY . .
   4. ENTRYPOINT ["dotnet", "run"]
6. Otevřete powershell ve složce s **Dockerfile**
7. **Udělejte build image**
   1. *docker build -t NAME:TAG .*
      1. **NAME** = Vaše libovolné jméno image (např: dockerconsoleapp)
      2. **TAG** = optional verze balíčku (např: 1.0)
8. Zkontrolujte, zda se image úspěšně vytvořil
   1. *docker image ls*
   2. *nebo pomocí Visual Studio*
      1. *View -> Other Windows -> Containers*
      2. *pokud zde taková možnost není, je potřeba nainstalovat do VS extension* ***Visual Studio Container Tools Extension***
9. *Vytvořte z image container a ten spusťte* 
   1. *docker run* ***NAME:TAG***
10. *Mělo by se zobrazit Hello World*

# LAB – Docker – ASP.NET Core – Visual studio

## Zadání úkolu:

1. Přidejte nový projekt ASP.NET Core Web Application (**EMPTY**)
   1. Projekt pojmenujte **DockerWebApp**
2. Přidejte do projektu Dockerfile
3. Spusťte aplikaci
   1. Nastavte nový projekt jako **Startup Project**
   2. 
4. Aplikace by měla zobrazit text **Hello World !**

# LAB – Docker – ASP.NET Core - CLI

## Zadání úkolu:

1. Vyzkoušejte to stejné pomocí docker CLI v PowerShellu
2. Otevřete v PowerShellu složku se solution
3. Vytvořte z aplikace nový image
   1. docker image build . -t IMAGE\_*NAME:TAG -f .\DockerWebApp\Dockerfile*
      1. *POZOR je potřeba být ve složce se solution (soubor .sln)*
      2. *POZOR je potřeba specifikovat Docker file pomocí -f PATH*
4. *Ověřte, že vznikl nový image*
   1. *docker image ls*
5. *Vytvořte z image container a ten spusťte*
   1. *Docker run -d -p 5001:80 --name containername IMAGE\_NAME:TAG*
      1. -d = detached mód
      2. --name = název vytvořeného containeru
6. Vyzkoušejte, zda aplikace běží
   1. <http://localhost:5001/>
7. Zobrazte logy z aplikace
   1. docker logs containername -f
      1. -f = float - vypisuje postupně všechny příchozí zprávy
8. Aplikace nemá žádné nastavení logování
   1. Přidejte logování do aplikace
      1. Otevřete Startup.cs
      2. Upravte funkci MapGet

endpoints.MapGet("/", async context =>

{

var logger = context.RequestServices.GetService<ILogger<Startup>>();

logger.LogError("Test docker logs");

await context.Response.WriteAsync("Hello World!");

});

1. Odstraňte existující container
   1. *docker container stop containername*
   2. *docker container rm containername*
2. Vyzkoušejte znovu vytvořit image, container a aplikaci spustit
   1. *docker image build . -t NAME:TAG -f .\DockerWebApp\Dockerfile*
   2. *docker run -d -p 5001:80 --name containername NAME:TAG*
3. Vyzkoušejte zda aplikace běží
   1. <http://localhost:5001/>
4. *Zobrazte logy z aplikace*
   1. docker logs containername -f

# LAB – Middleware run, use and map

## Zadání úkolu:

1. Vytvořte novou webovou aplikaci, začněte s prázdnou šablonou.
2. Pro začátek odmažte všechno z metody Configure
3. Pomocí metod Run a Map vytvořte aplikaci, která bude sloužit jako jednoduchý kalendář.
   1. Pokud uživatel zadá jakoukoliv cestu, zobrazí text: *$“Dnes je {DATE}”*
   2. Pokud ale uživatel zadá jako cestu **/Yesterday** zobrazí*$“Včera bylo {DATE – 1den}”*
   3. Kde **DATE** je dnešní den ve formátu MM/DD/YYYY

## Řešení:

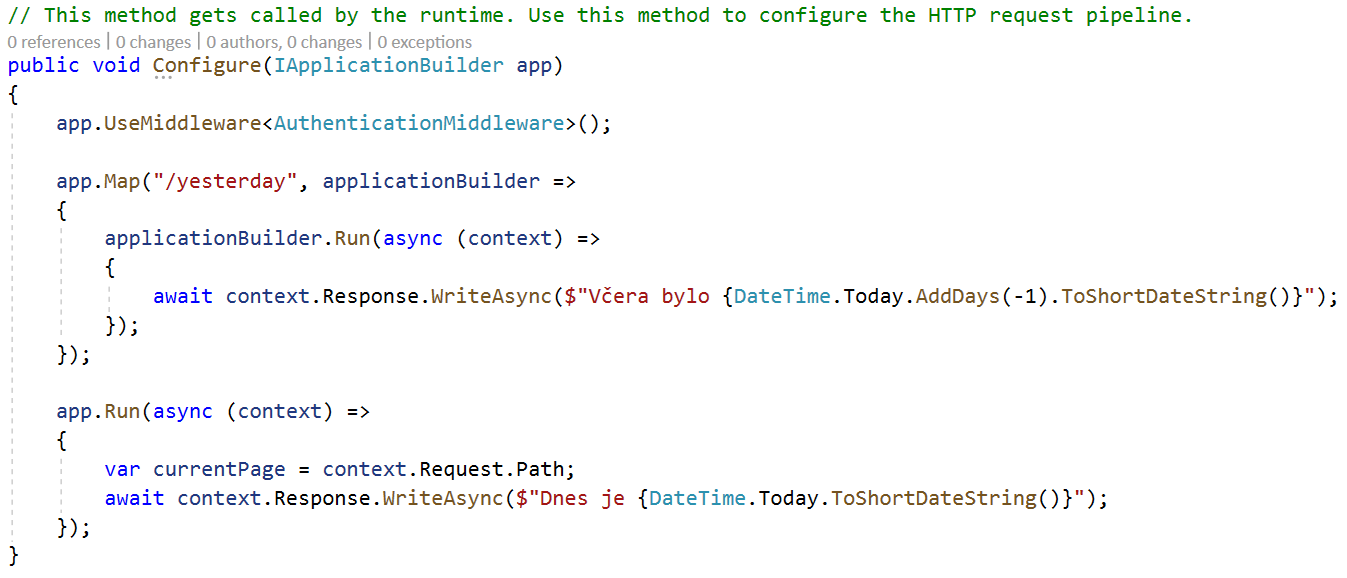


# LAB – Middleware Custom

1. Vytvořte middleware, který zkontroluje, zda má uživatel oprávnění zobrazit kalendář
2. Ověření bude probíhat pomocí Bearer authorization header.
   1. Token může být zadán jak v headeru požadavku, tak v query.
   2. Např: <https://localhost:44394?Authorization=my-secret-token>
3. Pokud požadavek obsahuje authorization hlavičku, zavolejte další middleware.
   1. Pro jednoduchost stačí token porovnat s hodnotou uloženou v const string.
4. Pokud hlavičku neobsahuje, zobrazte stránku 401

## Řešení:





# LAB – Static files

## Zadání úkolu:

1. Zákazník se rozhodl, že do naší aplikace potřebuje ukládat excel soubory. Zákazník požaduje, aby mohl každý uživatel na našem webu tyto soubory procházet a otevírat.

# PRÁZDNÁ STRÁNKA PRO VAŠE POZNámky 1.