

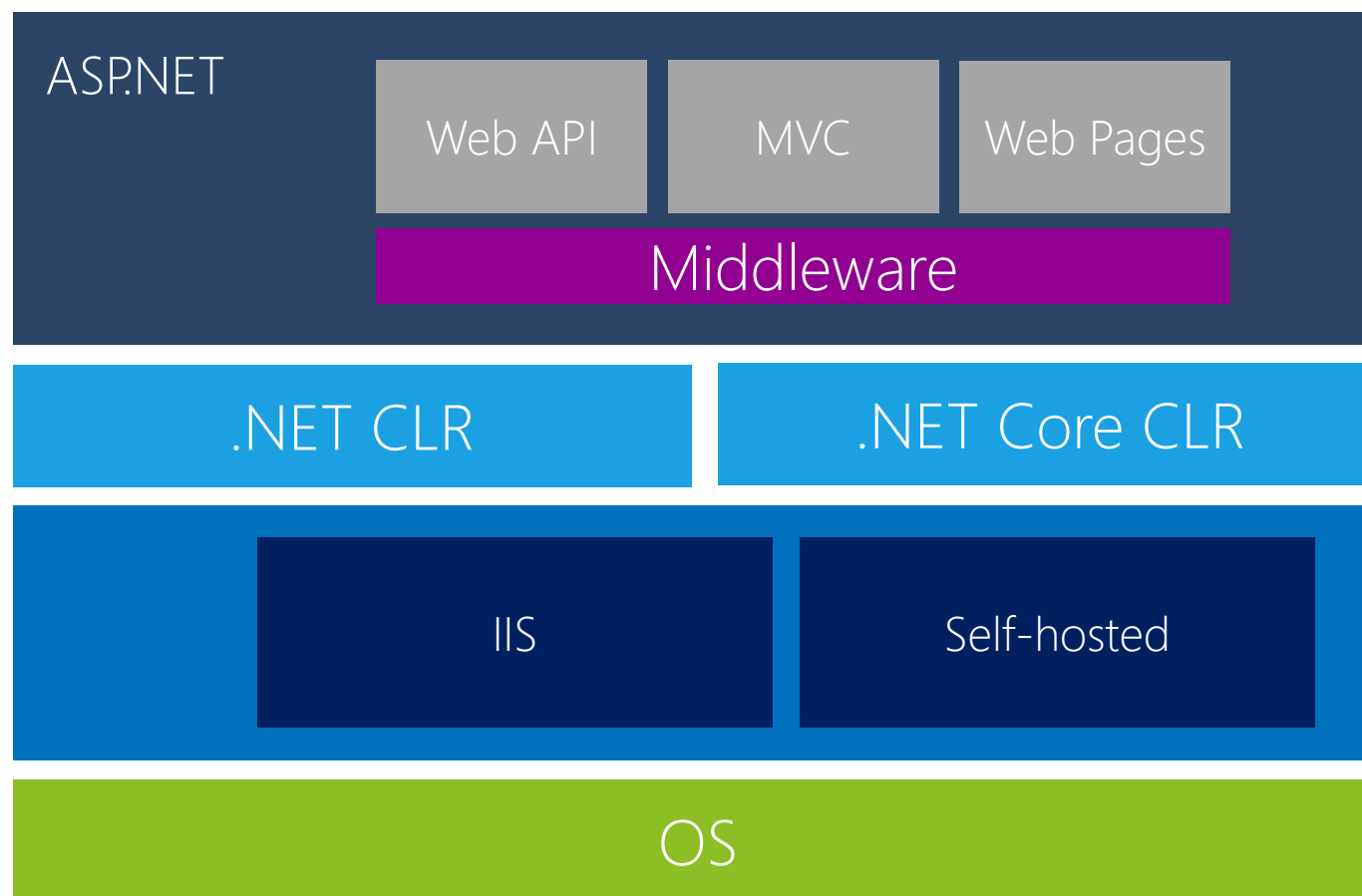
ASP.NET Core

Ing. Jan Janoušek

NuGet

- Správce balíčků pro .NET (pro jakýkoliv typ aplikace).
- Není závislý na ASP.NET.
- Zjednodušuje instalaci, správu a aktualizaci závislostí.
- <https://www.nuget.org>
- Integrace do VS – Package Manager Console a Package Manager.
- ASP.NET Core je distribuován právě pomocí NuGet balíčků - <https://www.nuget.org/packages?q=Microsoft.AspNetCore&sortBy=relevance>

Architektura ASP.NET Core



Základní ASP.NET Core aplikace

```
// Program.cs

public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // builder.Services.AddXXXX();

        var app = builder.Build();

        // app.UseXXXX();

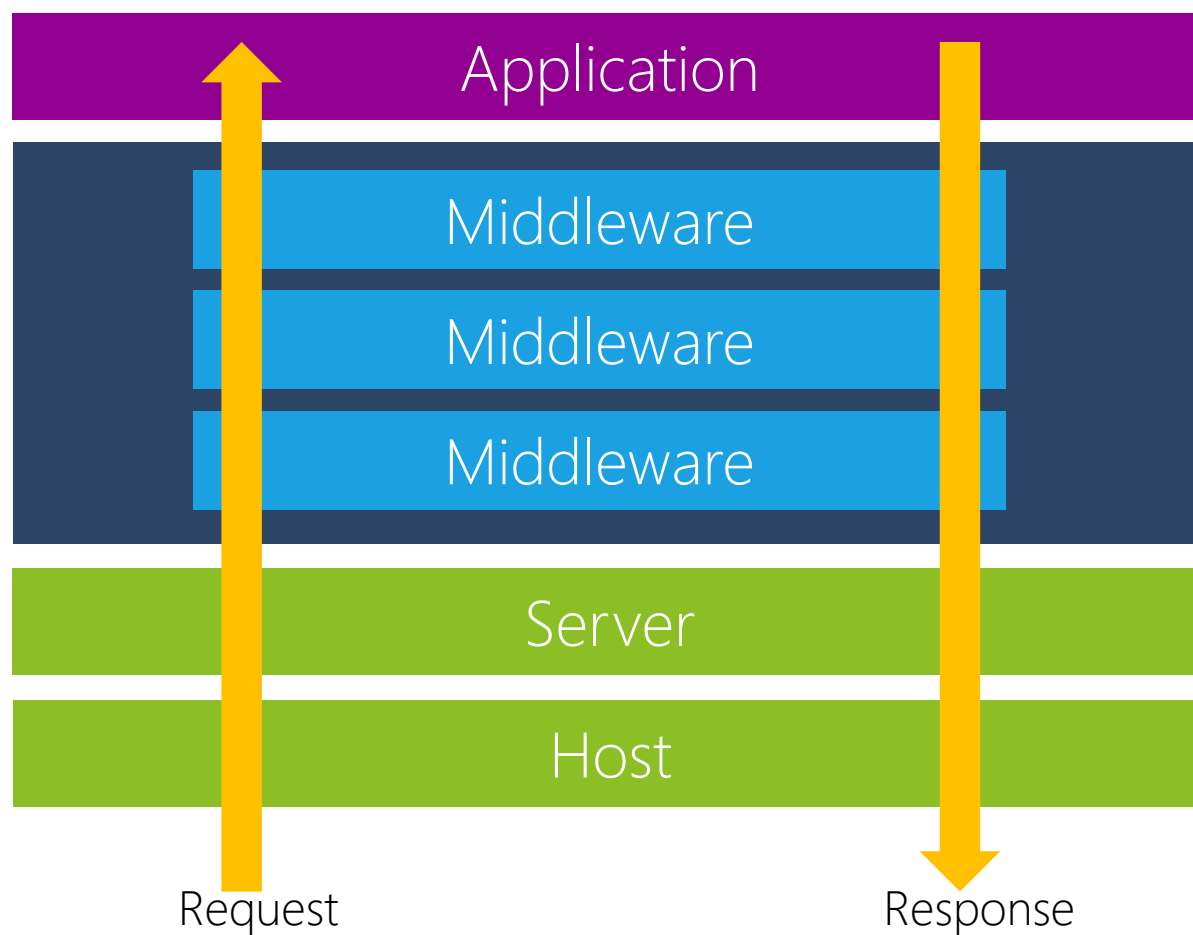
        app.MapGet("/", () => "Hello World!");

        app.Run();
    }
}
```

Základní vlastnosti ASP.NET Core

- Nevyžaduje IIS server (na rozdíl od ASP.NET).
- **Kestrel** – jednoduchý, multiplatformní webový server (preferovaný server pro ASP.NET Core)
- Konfigurace pomocí JSON.
- Nejde o monolitní framework, jako tomu bylo u předchozích verzí. Aplikace se skládá z jednotlivých balíčků (NuGet)
Tato vlastnosti umožňuje nezávislý vývoj jednotlivých komponent.

Middleware



Middleware

- Nahrazuje HTTP moduly, handlery, atd... z přechozích verzí ASP.NET

```
// Program.cs
```

```
app.UseExceptionHandler("/Home/Error");  
app.UseStaticFiles();  
app.UseSession();  
app.UseIdentity();  
app.UseMiddleware<MyCustomMiddleware>();  
app.UseMvc(routes => {  
    routes.MapRoute(  
        name: "default",  
        template: "{controller=Home}/{action=Index}/{id?}");  
});
```

Standardní Middlewares

- Authorization
- CORS (Cross-Origin Resource Sharing)
- Routing
- Session
- Static files
- Spoust dalších, které lze nainstalovat přes NuGety - včetně MVC!

Vlastní middleware

- Jakákoliv třída, které splňuje následující:
 - Má konstruktor s parametrem typu *RequestDelegate* (může mít i další, které budou získány pomocí DI).
 - Má metodu *Invoke*.
- Další middleware v řadě se volá pomocí *RequestDelegate* z konstruktoru.

```
public class MyCustomMiddleware {  
    private readonly RequestDelegate next;  
    public MyCustomMiddleware(RequestDelegate next) {  
        this.next = next;  
    }  
  
    public async Task Invoke(HttpContext context) {  
        await context.Response.WriteAsync("My first middleware");  
        await this.next.Invoke(context);  
    }  
}
```

Dependency Injection

- Podporován napříč celou platformou ASP.NET (a MVC 6). Závislosti lze předávat kamkoliv – do jakékoliv třídy (toto byl velký problém ve starých verzích MVC).
- Používá se pro například pro konfiguraci middleware.
- Definují se v rámci Startup.cs.
- Závislosti se označují jako „služby“ (návrhový vzor Service Locator).
- Typy služeb:
 - Transient – nová instance je vytvořena pokaždé, když je služba vyžádána (dokonce i v rámci stejného HTTP požadavku).
 - Scoped – nová instance je vytvořena pouze jednou pro každý HTTP požadavek (v rámci jednoho požadavku se služba znovupoužívá).
 - Singleton – nová instance je vytvořena pouze jednou. Následně se znovu používá pro všechny HTTP požadavky (singleton).

Dependency Injection

```
// Program.cs
IServiceCollection services = builder.Services;

services.AddIdentity<ApplicationUser, UserRoleEntity>()
services.AddUserStore<UserStore>();
services.AddRoleStore<RoleStore>();
services.AddUserManager<ApplicationUserManager>()

services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = true;
    .....
});

services.AddSingleton<ISmsSender, SmsSenderService>();
services.AddSingleton<IEmailSender, EmailSenderService>();
services.AddScoped<AuthorizationService>();

services.AddSession();

services.AddMvc();
```

```
// nějaký controller
public class AppController : Controller
{
    private ISmsSender smsSender;

    public AppController(ISmsSender smsSender){
        this.smsSender = smsSender;
    }

    public IActionResult SomeAction()
    {
        this.smsSender.Send("+420721.....", "text SMS");

        IEmailSender emailSender = this
            .HttpContext
            .RequestServices
            .GetService(typeof(IEmailSender));

        emailSender.Send("email@email.cz", "text");
        return View();
    }
}
```

appsettings.json (appsettings.[environment].json)

- Konfigurační soubor

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "cors": {
    "rules": [
      {
        "origin": "https://manage.iis.net",
        "allow": true
      }
    ]
  },
  "myData": {
    "ConnectionString": "Server=tcp:.....",
    "FbId": "123456789"
  }
}
```

```
// Program.cs
IConfiguration configuration = app.Configuration;

string dbConnectionString = configuration["myData:ConnectionString"];

...

// Nebo přes DI „IConfiguration“
```

Cookies

- Soubory/řetězce které se ukládají u uživatele a posílají s každým požadavkem na server.
- Uživatel je může číst i modifikovat!
- **Response.Cookies.Append(...)**
 - Vytvoří novu cookie
 - Lze použít i pro smazání (smazání = vytvoření nové expirované cookie)
- **Response.Cookies.Delete(...)**
 - Smaže cookie – vytvoří novou expirovanou
- **Request.Cookies.TryGetValue(...)**
 - Získá hodnotu z cookie

Minimal APIs

- Abstrakce nad middlewary.
- Hlavní metody:
 - MapGet(string path, Delegate handler)
 - MapPost (string path, Delegate handler)
 - MapPut (string path, Delegate handler)
 - MapDelete (string path, Delegate handler)
 - MapOptions (string path, Delegate handler)
- Delegát může vracet přímo data – string vs. Object - výchozí serializer
- Třída Results, TypedResults

Minimal APIs

- Routing – „product/{id}/{slug}“.
- Parameter Binding
- Dependency injection
- Explicitní parameter binding:
 - [FromRoute]
 - [FromQuery],
 - [FromServices]
 - [FromHeader]
 - [FromForm]
 - [FromBody]

Sessions

- Data, která se ukládají na serveru a jsou svázána s aktuálním uživatelem pomocí cookies.
- Fungují následovně:
 - Na serveru se vygeneruje unikátní klíč a k němu se uloží data (na uložišti nezáleží – DB, paměť, souborový systém).
 - Uživateli se zašle cookie obsahující klíč.
 - Při opětovném požadavku na server se na základě klíče načtou data.
- Uživatel je nemůže číst ani modifikovat (vše je na severu).
- ~~Je nutné nainstalovat balíček „Microsoft.AspNetCore.Session“ (v nových verzích neplatí).~~
- Konfigurace:
 - **services.AddDistributedMemoryCache();**
 - Program.cs / Services - **services.AddSession();**
 - Program.cs / App - **app.UseSession();**
- HttpContext.Session.Set(...);
- HttpContext.Session.TryGetValue(...);
- HttpContext.Session.Remove(...);
- HttpContext.Session.SetXXXXXX(...); - extension metody (SetInt32, SetString, ...)
- HttpContext.Session.GetXXXXXX(...); - extension metody (GetInt32, GetString, ...)

WebUtility

- WebUtility.HtmlEncode("...");
 - Převeďte HTML na entity - ** = **
 - Slouží k ošetření vstupů – nikdy nevypisovat neošetřené vstupy od uživatele do stránky!!!
- WebUtility.HtmlDecode("...");
 - Převeďte entity na HTML.
- WebUtility.UrlEncode("...");
 - Převeďte „čistý text“ na URL kódování „**10 % 2 = 0**“ = „**10+%25+2+%3D+0**“
- WebUtility.UrlDecode("...");
 - Převeďte text v URL kódování na „čistý text“.

IStringLocalizer<T>

- Služba pro překlad.
- T = libovolná třída se kterou se překlad „sváže“.

```
public HomeController(IStringLocalizer<MainRes> loc) {  
    string txt = loc.GetString("test");  
}
```

- Lokalizace se zapíná v Program.cs / Services:

- ResourcesPath = cesta k *.resx souborům

```
services.AddLocalization(options => options.ResourcesPath = "Resources");
```

*.resx soubor

- XML soubor s překlady – klíč/hodnota
- Vždy musí existovat základní/výchozí překlad v souboru „[nazev].resx“
- Jednotlivé překlady musí být v souborech s názvem „[nazev].[kultura].resx“ – název se musí shodovat.
- Pokud soubor s požadovanou kulturou neexistuje, použije se výchozí.
- K resx souborům se automaticky generuje C# kód – třída se statickými property odpovídajícími názvům klíčů.
- Při použití s `IStringLocalizer<T>`, musí být název resx souboru = názvu typu **T**.

Výběr kultury

- Jaká kultura se má aktuálně použít?
- Program.cs / Configure

```
app.UseRequestLocalization((RequestLocalizationOptions options) => {  
    // výchozí kultúra  
    options.DefaultRequestCulture = new RequestCulture("cs-CZ");  
    // Kultury, které se mohou použít pro formátování čísel, dat, atd...  
    options.SupportedCultures = new List<CultureInfo>() { new CultureInfo("cs-CZ"), new CultureInfo("en-US")  
};  
  
    // Kultury, které se mohou použít pro resx soubory  
    options.SupportedUICultures = new List<CultureInfo>() { new CultureInfo("cs-CZ"), new CultureInfo("en-US") };  
  
    // Vlastní poskytovatel kultury  
    options.RequestCultureProviders.Insert(0, new CustomCultureProvider());  
});
```

RequestCultureProvider

- Slouží k výběru kultury – vlastní logika

```
class CustomCultureProvider : RequestCultureProvider
{
    public override Task<ProviderCultureResult> DetermineProviderCultureResult(
        HttpContext httpContext
    )
    {
        bool isChrome = httpContext.Request.Headers["User-Agent"]
            .First()
            .IndexOf("chrome", StringComparison.InvariantCultureIgnoreCase) != -1;
        string culture = isChrome ? "cs-CZ" : "en-US";
        return Task.FromResult(new ProviderCultureResult(culture));
    }
}
```

Více o lokalizaci a globalizaci

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization?view=aspnetcore-5.0>

Dokumentace k ASP.NET Core

<https://docs.asp.net/en/latest/index.html>