# 9 - Návrh webové aplikace ASP.NET

# ASP.NET

* Active Server Pages
* Postaven na .NET Frameworku od společnosti Microsoft (framework = softwarová struktura sloužící jako podpora při vývoji sw produktů)
* Běží na serveru a umožňuje vytvářet dynamické webové stránky
* Jazyky C#, F#, VB.NET

# Razor Pages

* Server-side framework umožňující vývoj dynamických webových stránek
* Řešení pro vytvoření jednoduché a velmi kompaktní aplikace pro různé účely

# MVC

* Starší klasičtější architektura
* Populární, kromě ASP.NET MVC také Laravel, Ruby on Rails, Django, ...
* Odděluje logiku od výstupu
* Vhodné pro složitější aplikace, Razor Pages pro začátečnické práce

## Model

* Definuje strukturu dat
* V C# se používají třídy pro popis modelu
* Objekty jsou ukládány v databázi

## View

* Zobrazuje data, uživatel s nimi může pracovat
* HTML, CSS a Razor Syntaxe

## Controller

* Logika aplikace
* View (uživatel) s ním pracuje pomocí http requestů

# Page

* Dle konvencí se ukládají do složky Pages
* Šablona pro vyrenderování

# PageModel

* Připravuje data pro zobrazení Page
* Každá Page má jeden PageModel
* PageModel je třída, do níž jsou připojené obecné služby (přístup k databázi, session, cookie, ...)
* Tyto služby se předávají přes Dependency Injection (základní princip Razor Pages)
* Tyto obecné třídy jsou nadeklarovány ve Startupu do Dependency kontejneru

# Razor Syntaxe

* C# kód je uzavřen v @{}
* Inline výrazy začínají @
* Jsou ukončené ;
* Proměnné začínají var nebo typem

# Startup.cs

* Od verze ASP.NET 6 spojen dohromady se souborem Program.cs
* ConfigureServices
  + Volitelná metoda, která konfiguruje služby aplikace
  + AddService, AddDbContext, AddDefaultIdentity, AddRazorPages
  + Znovu použitelná komponenta, která zprostředkovává funkcionalitu aplikace
  + Služby jsou registrovány v ConfigureServices a používány v aplikaci pomocí Dependency Injection nebo ApplicationService
* Configure
  + Používá se k určení, jak aplikace reaguje na požadavky http

# Služba

* ASP.NET Core používá Dependency Injection
* Lze připojit do naší aplikace a využívat jejích prostředků

# Databáze

* Ukládání permanentních dat (uživatelé, ...)
* Připojení přes connection string v appsettings.json



* DbContext reprezentuje relaci s databázi a zprostředkovává API pro komunikaci s databází a připojení k databázi
* DbSet:
  + Třída DbSet<TEntity> reprezentuje kolekci pro danou entitu v rámci modelu a je branou k databázovým operacím proti entitě
  + Přidávají se jako vlastnosti do třídy DbContext a jsou standardně mapovány na databázové tabulky, které přebírají název vlastnosti DbSet<TEntity>
* Migrace:
  + Umožňují provádět různé změny i po vytvoření databáze
  + Při vytvoření migrace je porovnán stav modelů a databáze
  + Vytvoření migrace  
    Add-Migration name
  + Vymazání poslední migrace  
    Remove-Migration name
  + Aplikace všech vytvořených migrací na databázi  
    Update-Database
* Identita:
  + Sada balíčků, hotové řešení umožňující přihlašování uživatelů
  + Rozšiřuje databázi; potřebuje Entity Framework, "staví na něm" (ukládání loginu, hesla, tokenu, ...)
  + Poskytuje služby UserManager<TUSer>, RoleManager<TRole>
  + Rozšiřuje původní DbContext o několik dalších tabulek – Users, Roles, ...
  + Umožňuje scaffoldovat (generovat) stránky pro identitu

# Entity Framework

* Běží na .NET
* Generuje SQL dotazy pro konkrétní databázi (MySQL, Microsoft SQL Server, SQLite, PostgreSQL, ...)
* Příkazy v syntaxi LINQ překládá do konkrétního SQL, tento dotaz odešle a přijdou mu data, která přeloží zpět do C#