Návrh webové aplikace ASP.NET

Razor Pages, MVC, Page, PageModel, Razor syntaxe, Setup.cs, služba, databáze, Entity Framework, Identity

Joseph Soukup – P4

Razor Pages

- Razor Pages poskytují řešení, jak vytvořit jednoduchou a velmi kompaktní aplikaci pro různé účely
- Díky víceúčelovým PageModelům lze vytvářet jednoduché webové aplikace se závislostí na menším množství služeb a jednoduchou business logikou soustředěnou do jednoho místa

MVC (Model View Controller)

- Rozděluje aplikaci do 3 částí:
  - Model
  - View
  - Controller

Model

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

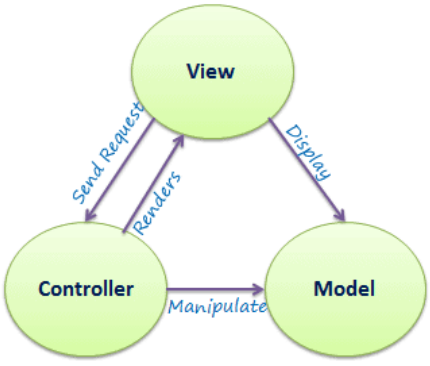
View

- View je v MVC UI
- Znázorňuje data z Modelu a umožňuje uživateli s nimi pracovat
- View v ASP.NET je HTML, CSS a Razor Syntaxe => jednoduchá komunikace s modelem a kontrolérem

Controller

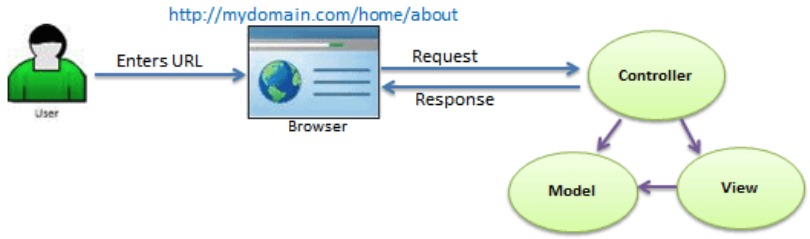
- Kontrolér pracuje s požadavky uživatele
- Typický je, že uživatel přes View podá http request, který se zpracuje kontrolérem
- Kontrolér požadavek zpracuje a pošle odpovídající odpověď

Interakce mezi Modelem, View a Kontrolérem



MVC Architecture

Běh požadavku uživatele



Request Flow in MVC Architecture

Page a PageModel

- Všechny tzv. Razor Pages se dle konvencí vkládají do složky **Pages**
- Klasická Razor Page je vlastně obyčejné view (cshtml) přepnuté do režimu Razor Page pomocí direktivy @page
- Takový view dále umožňuje vkládat inline PageModel, které popisuje chování dané Razor Page
- Pokud je logika Razor Page komplexnější, pak je vhodné zmírněný PageModel extrahovat do samostatného souboru
- Razor Page o tomto modelu ví na základě direktivy @model
- PageModel je v Razor Pages víceúčelový
- Obsahuje data vykreslovaní do View, řeší zpracování požadavků z klienta a obvykle má vazbu na další služby

Razor syntaxe

- C# kód je uzavřen v @{…}
- Inline výrazy začínají @ (proměnný/funkce)
- Ukončeno ;
- Proměnné začínají var nebo typem (int, string, …)
- String je uzavřen do uvozovek
- C# soubory mají koncovky .cshtml

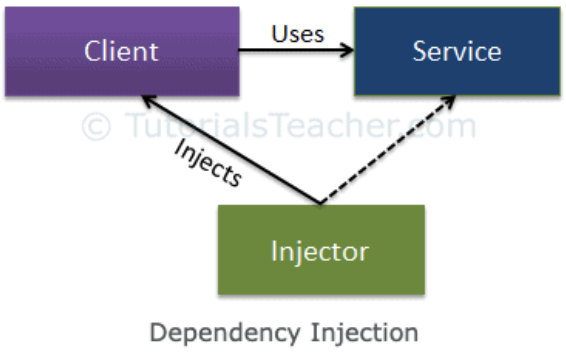
```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

Startup.cs

- ConfigureServices a Configure se volají při startu aplikace

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddRazorPages();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

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

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapRazorPages();
        });
    }
}
```

ConfigureServices:

- Metoda, která konfiguruje služby aplikace
  - Služba je znovupoužitelná komponenta, která zprostředkovává funkcionalitu aplikace
  - Služby jsou registrovaný v ConfigureServices a spotřebována v aplikaci pomocí:
    - Dependency Injection (DI)
      - Návrhový vzor, který nám pomáhá vytvářet **loosely coupled kód**
        - Používání komponent, které spolu komunikují pomocí interface
      - 3 druhy tříd
        - Třídu klienta
        - Třídu služby
        - Třídu injectora
          - Vkládá třídu služby do třídy klienta
      - 

Dependency Injection
    - ApplicationService
- Metoda je:
  - Volitelná
  - Volaná před Configure aby mohla nakonfigurovat služby aplikace
  - Místo, kde se nacházejí možnosti konfigurace

- Nastavují se zde věci jako:
    - AddDbContext
    - AddDefaultIdentity
    - AddRazorPages
    - AddService

Configure

- Používá se k určení, jak aplikace reaguje na požadavky http
- Request pipeline je konfigurovaná přidáním middleware komponent
- Každá **Use** metoda přidá jednu nebo více middleware komponent

Služba

- ASP.NET core používá DI
- Aby DI vědělo, jak rozpoznat závislosti, tak musí být nejdříve nakonfigurovány
- Například:
    - **services**.**AddMvc**() přidá službu, která je potřeba pro veškerou MVC funkcionalitu
    - **services**.**AddAuthentication**() přidá službu, která je potřeba pro funkcionalitu autentifikace
- Je možné taky vytvořit svou službu, která může obsahovat logiku aplikace (metody, práce s databází, …)

Entity Framework

- Je to Framework pro přístup k datům , který se používá pro přístup a testování dat

DbContext

- Reprezentuje relaci s databází a zprostředkovává API pro komunikaci s databází
    - Připojení k databázi
        - Zahájení komunikace
    - Operace
        - Přidání
        - Editování
        - Mazání

DbSet

- Třída DbSet<TEntity> reprezentuje kolekci pro danou entitu v rámci modelu a je bránou 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>

Model

- Model obsahuje jak data, tak chování modelu
- Každá třída reprezentuje Entitu

```
public class Author
{
    public int AuthorId { get; set; }
    public string LastName { get; set; }
    public List<Book> Titles { get; set; } = new List<Book>();
}
public class Book
{
    public int BookId { get; set; }
    public string Title { get; set; }
    public Author Author { get; set; }
}
```

Relace

- A relační databázi je každá entita uložena ve vlastní oddělené tabulce identifikovaná unikátním klíčem
- Relace jsou určeny pomocí Foreign Keys
- One To Many
- Many To Many
- One To One

Connection String

- Obsahuje informace o zdroji dat (databázi)

  The following snippet shows how a connection string can be added to an appsettings.json file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MyConnection": "server=.;database=myDb;trusted_connection=true;"
  }
}
```

  Now you can use the Configuration API to pass the connection string to the `DbContext` in the Startup class:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ConfigurationContext>(options => {
        options.UseSqlServer(Configuration.GetConnectionString("MyConnection"));
    });
}
```

Migrace

- Umožňují provádět různé změny i po vytvoření databáze
-
```
[Package Manager console]
add-migration <name of migration>
```
- Při vytvoření migrace je porovnán stav modelů a databáze

- ```
  [Package Manager Console]
  remove-migration
  ```
- Vymaže poslední migraci
- ```
  [Package Manager Console]
  update-database
  ```
- Všechny vytvořené migrace se aplikují na databázi