Webové technologie

Ondřej Pavlica
PV179

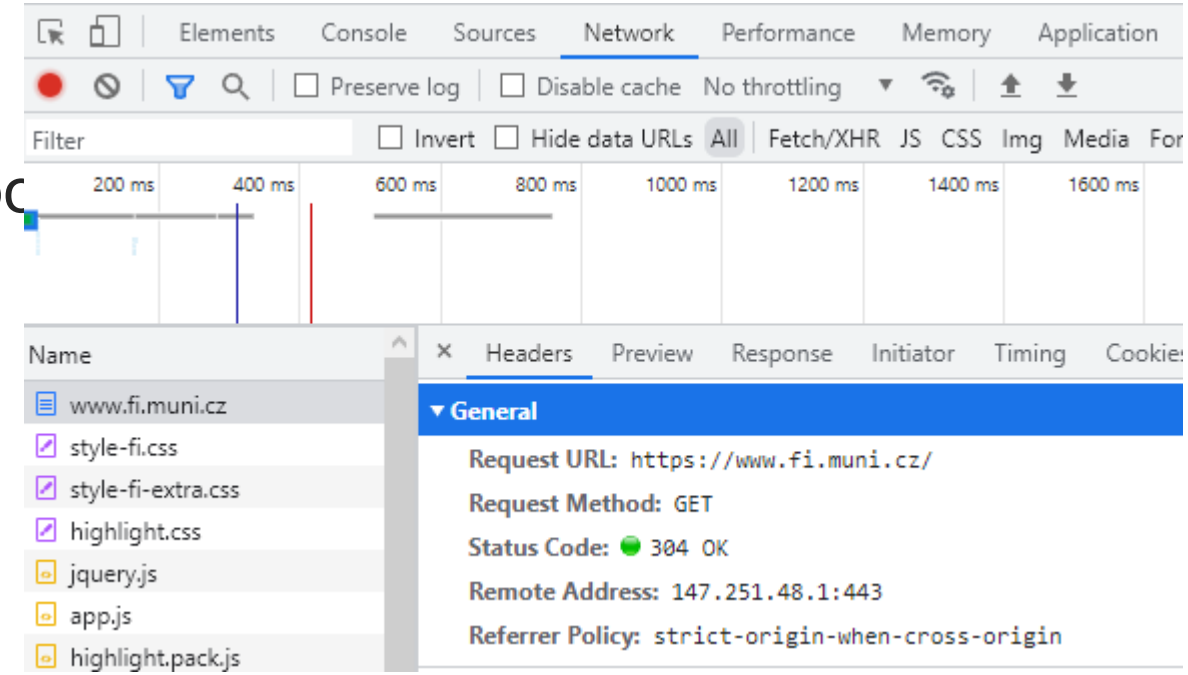
Osnova přednášky

- Základy webu
- *HTTP, REST, HTML*

- ASP.NET Core
- *Web API*
- *MVC*
- *Blazor*

HTTP

- Hypertext transfer protocol
- Formát přenosu dat
- Klient – Server model (Request – Respo



HTTP Request

Obsahuje:

- Metodu (GET, POST, PUT, PATCH, DELETE)
- URI (např. https://fi.muni.cz)
- Tělo (Body) – data odesílaná serveru (při GETu ne)
- Hlavičky
- Cookies

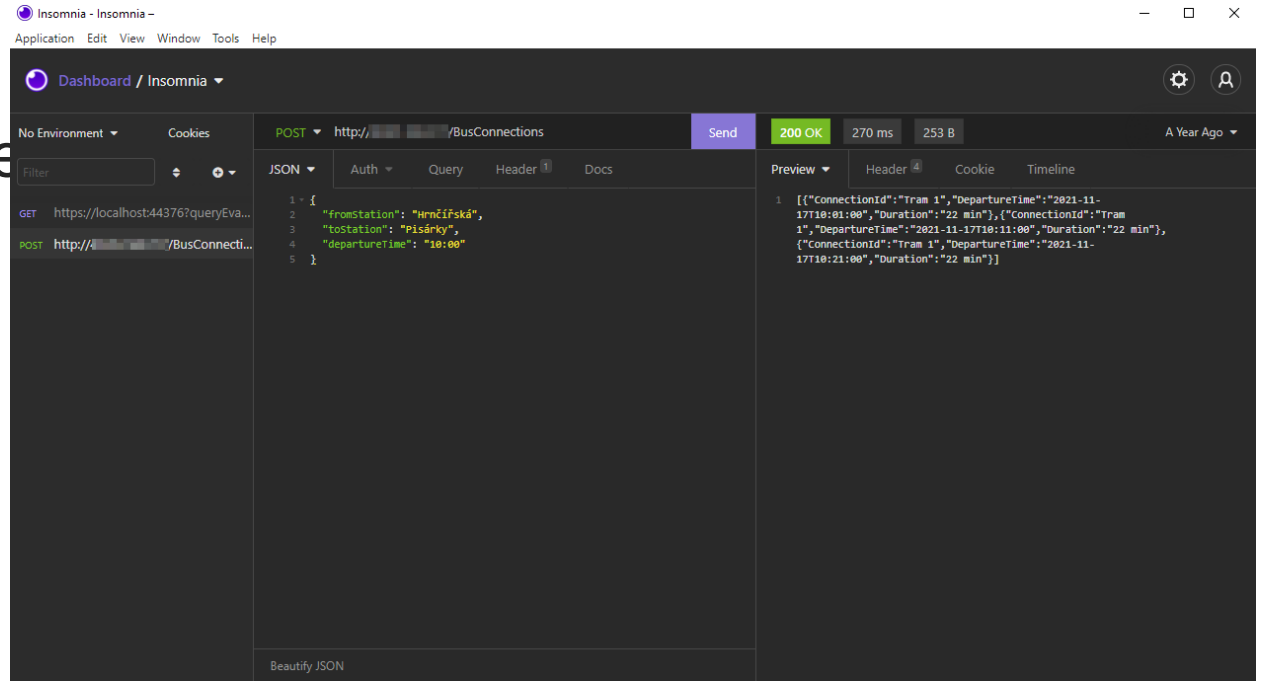
HTTP Response

Obsahuje:

- Statusový kód
- *200 – OK*
- *300+ – přesměrování*
- *400+ – „tvoje chyba“ (bad request, forbidden, atd.)*
- *500+ – „moje chyba“ (internal server error, timeout, atd.)*
- Tělo (Body) – data odesílaná serveru (při GETu ne)
- Hlavičky
- Cookies

Debugování HTTP

- Karta „Network“ v DevTools prohlíže
- API browser poskytuntý serverem
- *Swagger*
- Standalone aplikace
- *Fiddler*
- *Insomnia*



REST

- **RE**presentational **S**tate **T**ransfer
- Rozhraní pro bezestavovou komunikaci po síti

- V praxi znamená, že dostaneme definici HTTP endpointů, jejich metod a parametrů
- Na tyto endpointy pak odesíláme HTTP dotazy, a tím si vyměňujeme data

HTML

- **H**yper**T**ext **M**arkup **L**anguage
- Není programovací jazyk, ale pouze značkovací (nebuďte paní z HR)

- Definice dat na stránce, která se mají vykreslit prohlížečem
- Tato přednáška předpokládá alespoň základní znalosti HTML
- Více na https://www.w3schools.com/html/

ASP.NET Core

- Webový framework
- Spousta věcí out-of-the box
- Coding by convention
- Hodně obecný, lze použít s vícero technologiemi:
  - Web API
  - MVC
  - Razor Pages
  - Blazor
  - …

Obecné koncepty

- Akce
- *.NET metoda, která se zavolá při konkrétním dotazu klienta*

- Controller
- *Třída obsahující jednotlivé akce, spravující závislosti akcí*
- *Akce v controlleru by měly tvořit logický celek (např. UserController, OrderController)*

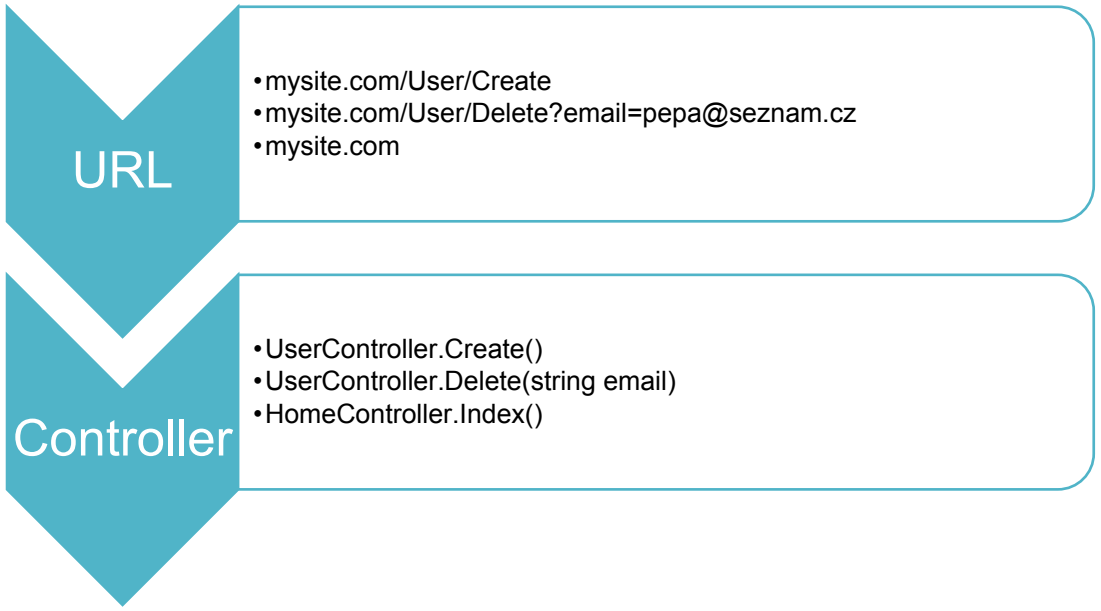
- (View)Model
- *Třída obsahující data k výměně s klientem*

- Routing
- *Logika namapování URL na konkrétní akci*

- Middleware
- *„Mezikroky“ při cestě od přijetí requestu po spuštění akce a zpět*
- *Zde můžeme vkládat věci, které budou mít globální vliv na aplikaci*
- Např. Autorizace, cachování stránek

Controller

- Vstupní bod HTTP dotazu
- Vhodný controller a akce se hledá podle konvence:



Controller (pokr.)

- Konvence nalezení controlleru a akce lze měnit v inicializace přes:
- *`.MapControllerRoute()` (změna konvence hledání controllerů a akcí)*
- *`.UseRouter()` (komplexnější logika mapování URL – akce)*
- Hodí se například při SEO, když chceme mít lokalizovanou i URL
- Dané akci lze nastavit i explicitní cestu přes atribut [Route(„Url“)]

Akce

- Metoda zpracovávající dotaz klienta
- Argumenty metody = data poslaná klientem
- Akce mohou být označeny atributy `[HttpGet]`, `[HttpPost]`, …

- Při výstupu implicitní konverze na HTTP response
- Typy návratových hodnot:
- *Primitivní datové typy (string, int, …)*
- *IActionResult*
- Stavové kódy (`return Ok()`, `BadRequest()`, …)
- Data (`return Ok(data)`) – formát dat (JSON, XML) se domluví s prohlížečem podle hlaviček requestu a nebo explicitně přes atribut – např. `[Produces(„text/xml“)]`
- Soubory (`return File(fileStream)`)
- Přesměrování (`return RedirectToAction(„Error“)`)

Model

- Klasická C# třída, většinou obsahuje jen properties
- V rámci ASP.NET jdou její properties označit validačními atributy
- *Hodí se pro uživatelský vstup, např. registrační formulář*
- *Např. `[Required]`, `[Range(min, max)]`, `[EmailAddress]`*
- *Pokud máme v projektu zapnutou nullability, tak se `[Required]` atribut přidává automaticky na nenullovatelné properties*
- *V controlleru se pak validita kontroluje pomocí `ModelState.IsValid`*

Model binding

- Proces namapování dat poslaných klientem do podoby .NET objektů, které může akce dále zpracovávat

- Zdroje dat mohou být různé:
- *HTTP formulář*
- *URL parametry*
- *Headery*
- *Cookies*

- Tyto namapované objekty se controlleru předávají jako argumenty metody

Model binding (pokr.)

- Model binding defaultně funguje tak, že jednoduché objekty (int, string, apod.) mapuje z URL parametrů a komplexní objekty (třídy) z formulářových dat (request body)

- Tento proces lze ovlivnit:
- *Overridnutím model binderu v nastavení aplikace před startem*
- *Atributy v argumentech metody: `[FromBody]`, `[FromForm]`, `[FromQuery]`, `[FromHeader]`*
- *Explicitně v definici třídy: `[BindProperty]`*
- *Pomocí vlastního model binderu: `[ModelBinder(typeof(MyCustomModelBinder))]`*

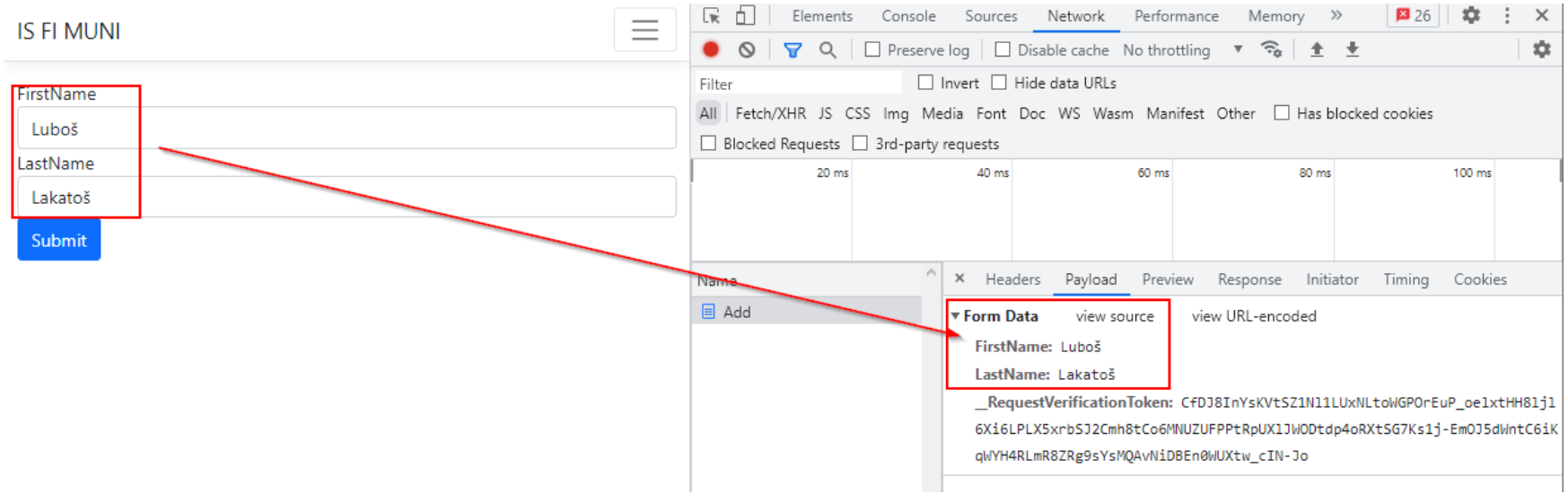
Model binding – komplexní objekty

- Někdy potřebujeme poslat data s více úrovněmi a nebo komplexní objekty, jako jsou např kolekce.

- Pro tyto případy používáme v názvech parametrů notaci níže:
- *Podatribut:* `ParentObject.ChildValue: „someValue“`
- *Indexace:* `MyCoolDictionary[20]: „You just lost the game.“`

- Obecně je s defaultním model binderem špatný nápad posílat enumy, je lepší poslat jejich integerovou reprezentaci a v rámci controlleru si je přecastovat

Model binding (ukázka)




```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Add(StudentUpsertViewModel model)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    var dto = model.ToDto();
    _studentRepository.Create(dto);

    return RedirectToAction(nameof(Index));
}
```

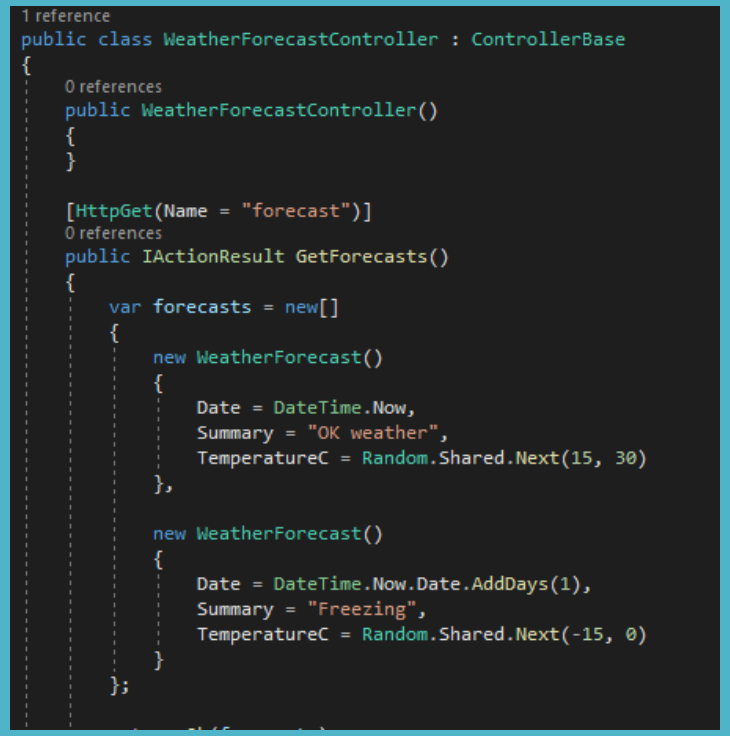
Filtry (typ middleware)

- Návrhový vzor **Interceptor**
- Lze aplikovat na controller a nebo akci
- Využití např. na logování, autentizaci, autorizaci

- `[Authorize]`
- `[ValidateAntiForgeryToken]`

Web API

- Součást ASP.NET pro tvorbu REST rozhraní
- Stačí controller, který dědí z `ControllerBase` a atribut `[ApiController]`



```
1 reference
public class WeatherForecastController : ControllerBase
{
    0 references
    public WeatherForecastController()
    {
    }

    [HttpGet(Name = "forecast")]
    0 references
    public IActionResult GetForecasts()
    {
        var forecasts = new[]
        {
            new WeatherForecast()
            {
                Date = DateTime.Now,
                Summary = "OK weather",
                TemperatureC = Random.Shared.Next(15, 30)
            },

            new WeatherForecast()
            {
                Date = DateTime.Now.Date.AddDays(1),
                Summary = "Freezing",
                TemperatureC = Random.Shared.Next(-15, 0)
            }
        };
```

Live Demo

Web API

Konuzumace API

C#

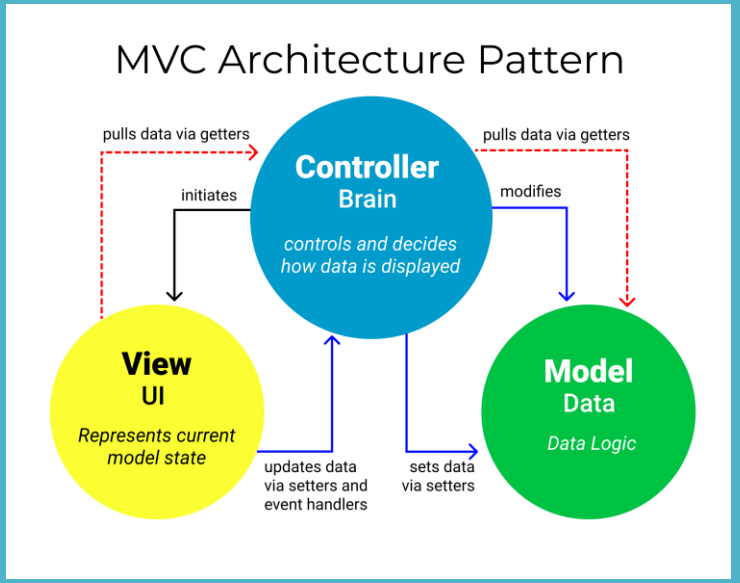
- HttpClient
- WebClient
- HttpWebRequest
- *RestSharp (NuGet)*

JS

- AJAX
- Fetch API

Model-View-Controller

- SW pattern na oddělení domén a snadnou zaměnitelnost kódu
- Nemusí být požit jen u webového vývoje
- Model
  - Data + Aplikační logika
  - V praxi často jen data
    - *Logika pak v službách a příp. jejich fasádách*
- View
  - Vykreslení dat
  - V našem případě HTML (Razor)
- Controller
  - Vstupní bod
  - Načtení modelu, validace, propojení s view



MVC - ukázka

[HttpGet]
UserController.Add()

[Post] UserController.Add(UserAddViewModel model)

GET: mysite.com/user/add
POST: mysite.com/user/add
+ form data

Vytvoření a naplnění daty z HTTP dotazu

Uživatel vyplní data a klikne na „Odeslat“

Uživateli se zobrazí stránka „Úspěšně přidáno“

View

Soubory .cshtml (Razor View Engine)

Jde v nich psát standardní HTML, ale také vložit kousky C# kódu
- Přepnutí do C# kontextu: @(výraz), @{blok kódu}
- Přepnutí zpět do HTML kontextu: @:html, <validni_html_tag>…</v_h_t>

Jdou komponovat
- Např. _Layout.cshtml na společnou kostru všech HTML stránek
- `@Html.Partial(„viewPath“)` vykreslí jiné view uvniř aktuální view
- `@Component.Invoke(„componentName“, parameters)` vykreslí komponentu (= view spojené s „mini-controllerem“)

Nahoře jsou direktivy:
- `@model Namespace.ClassName` – definice modelu
- `@using Namespace` – jako using v C#
- `@inject Service` – injektování servicy – **aplikační logika nemá být ve view!**
  - *Hodí se však např. k ověření privilegií uživatele a podmíněnému vykreslení např. menu*
  - *Nebo lokalizaci textů*

View (pokr.)

- Důležité properties dostupné uvnitř view:
- *`Model` – instance modelu vložená controllerem*
- *`Html` – pomocné metody k renderování HTML*
- `using (Html.BeginForm()) {…}`
- `Html.TextBoxFor(x => x.Name)`
- `Html.ValidationMessageFor(x => x.Name)`
- *`Url` – pomocné metody ke generování URL*
- `Url.Action(„ActionName“)`
- *`User` – právě přihlášený uživatel*

Live Demo

MVC

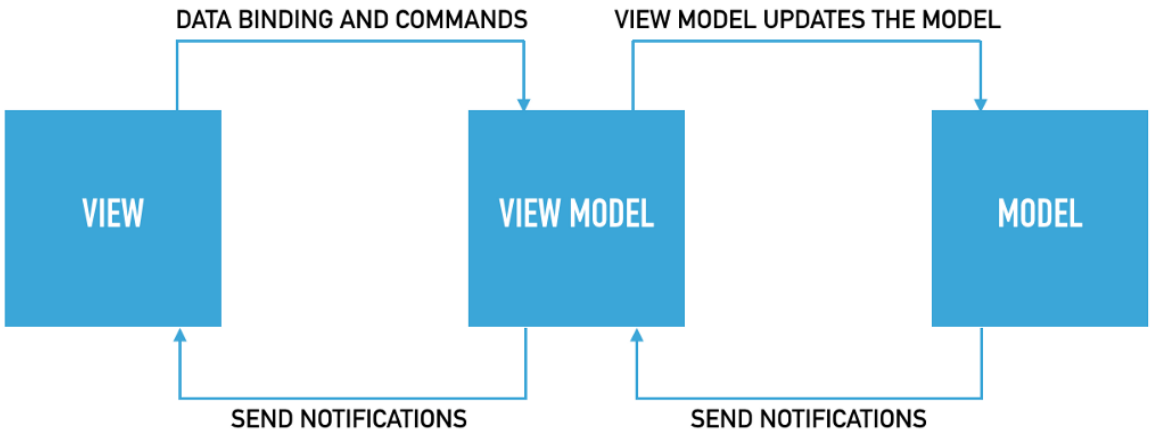
Blazor

- Technologie umožňující vytvářet interaktivní weby bez JS
- 2 módy:
  - Blazor Server – na straně klienta je SignalR klient, který obstarává real-time komunikaci se serverem. Veškerá interaktivita se zpracovává na straně serveru
    - *Rychlejší prvotní načtení stránky*
    - *Podpora starších prohlížečů*
    - *Snadnější debugging*
  - Blazor Client – po připojení se do paměti prohlížeče načte WebAssebly aplikace, která obsahuje veškerou aplikační logiku
    - *Pomalé prvotní načtení stránky*
    - *Následně aplikační logika na straně klienta – responzivnější UI*
    - *Stačí obyčejný webserver, netřeba spouštět .NET -> levnější hosting*
    - *Problematičtější debugging*

Blazor (pokr.)

•Využívá MVVM pattern:

- Model – data, aplikační logika
- View – zobrazení dat z modelu
- ViewModel – „lepidlo“ mezi modelem a viewmodelem, zajišťuje výměnu dat



Live Demo

Blazor

Děkuji za pozornost

- Q&A

Dotazy po přednášce?

- xpavlica@fi.muni.cz / macak@mail.muni.cz
- PV179 Discord