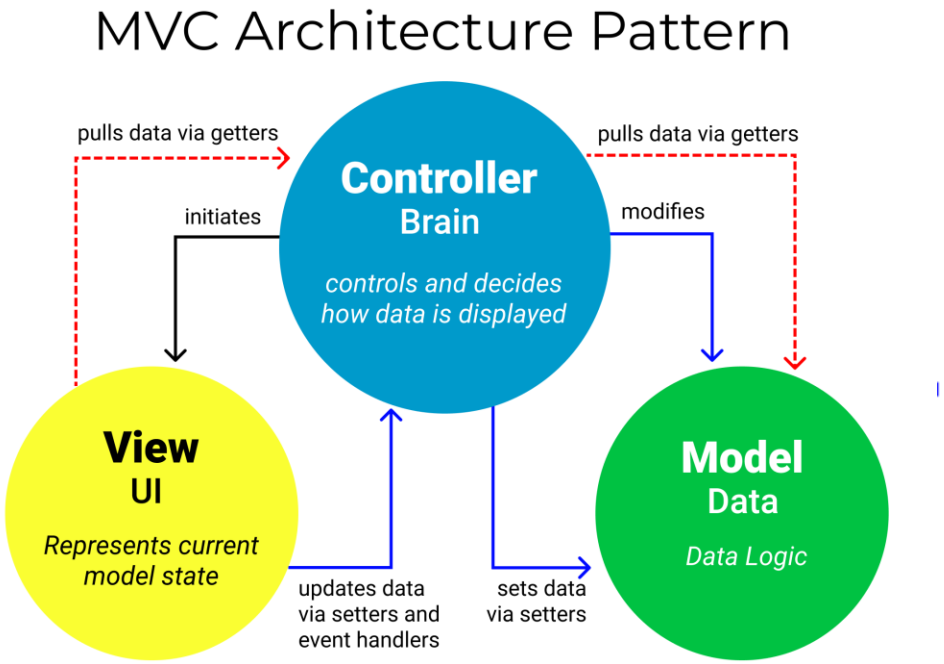

MVC Architektura
Ondřej Pavlica

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



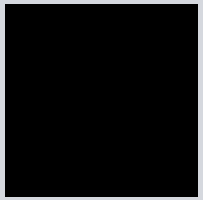
MVC - ukázka

[HttpGet]
UserController.Add()

[Post] UserController.Add(UserAddViewModel model)

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

Request



Controller

Response

Model

Automatické naplnění daty z HTTP dotazu

HTML

View

Uživatel vyplní data a klikne na „Odeslat“

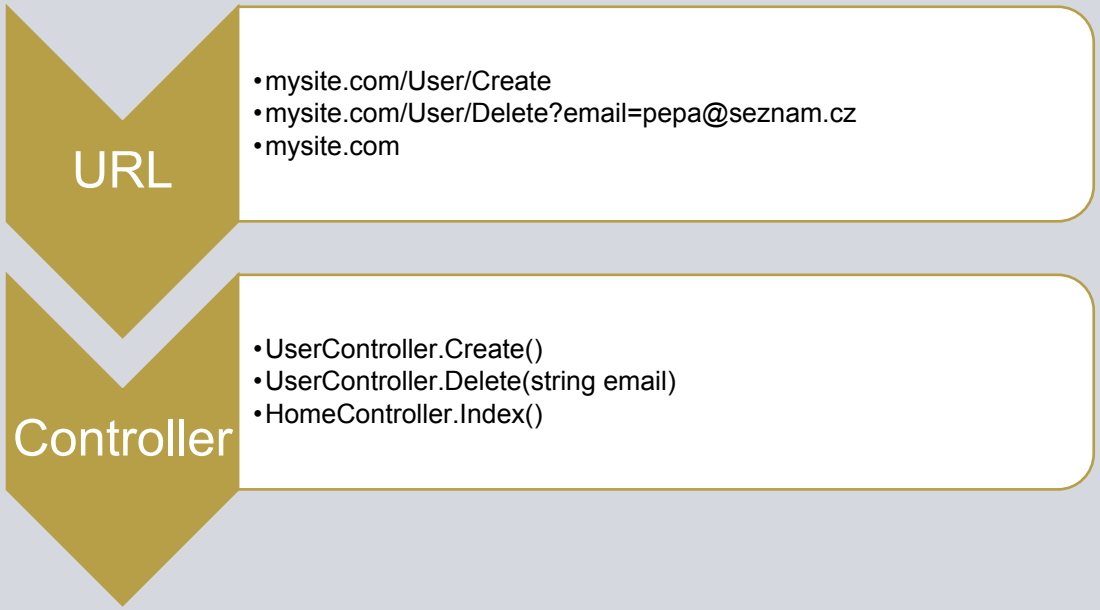
Uživateli se zobrazí stránka „Úspěšně přidáno“

Model

- Klasická C# třída, nemá moc smysl se tu dlouho zastavovat
- 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`

Controller

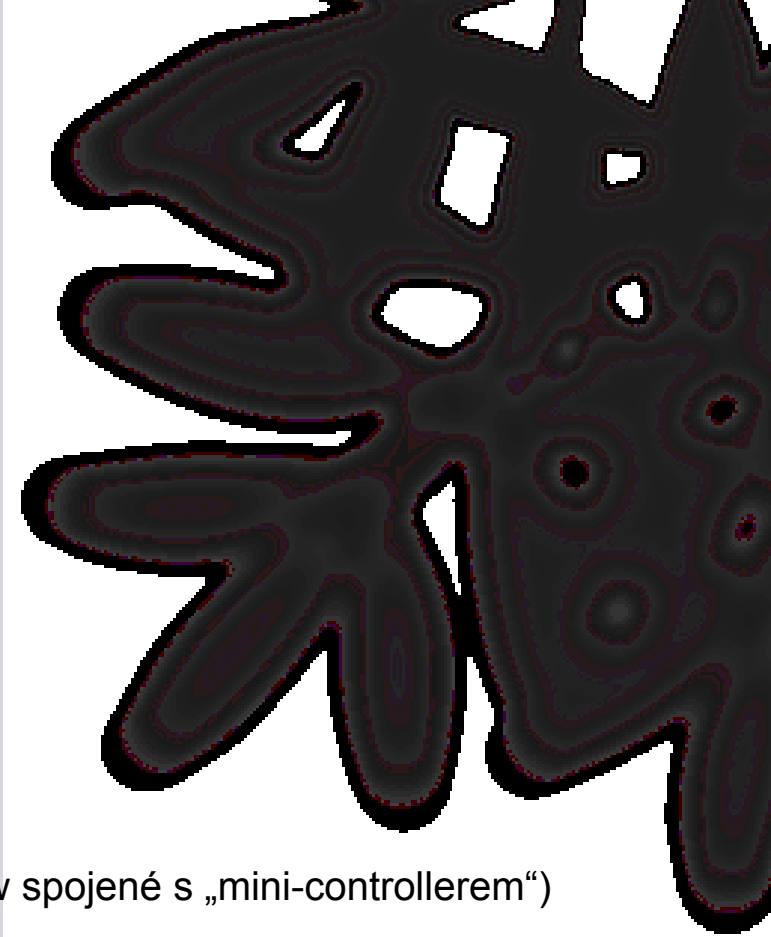
- Vstupní bod HTTP dotazu
- Atributy `[HttpGet]`, `[HttpPost]`, …
- Vhodný controller a akce se hledá podle konvence:



Controller (pokr.)

- Konvence nalezení controlleru a akce lze měnit v inicializaci 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“)]

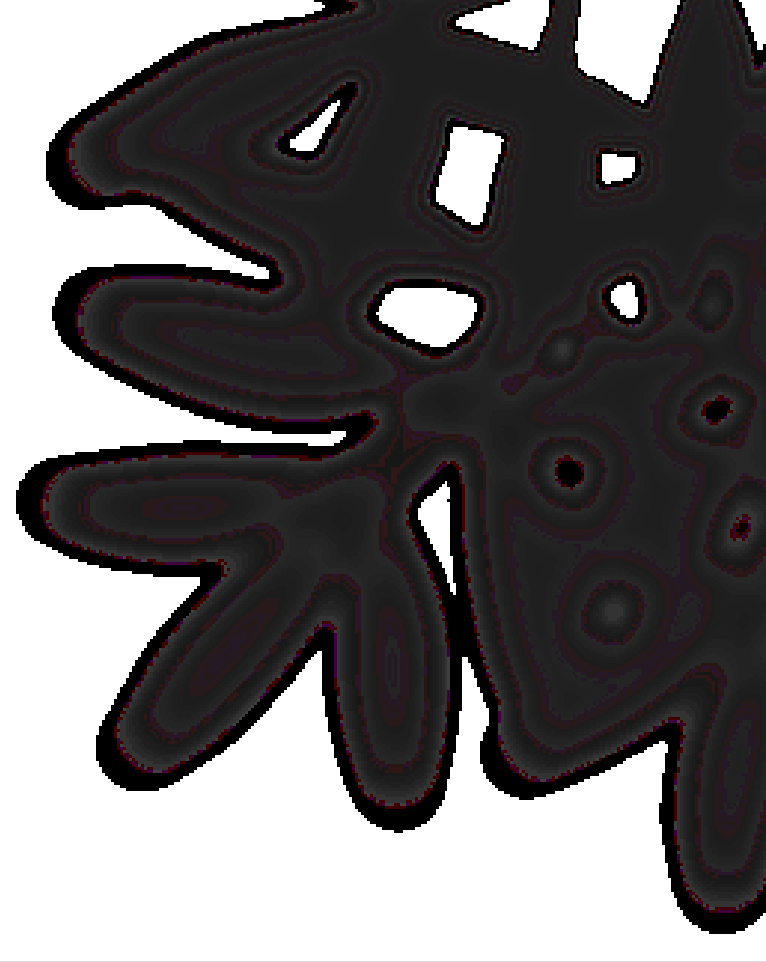
View



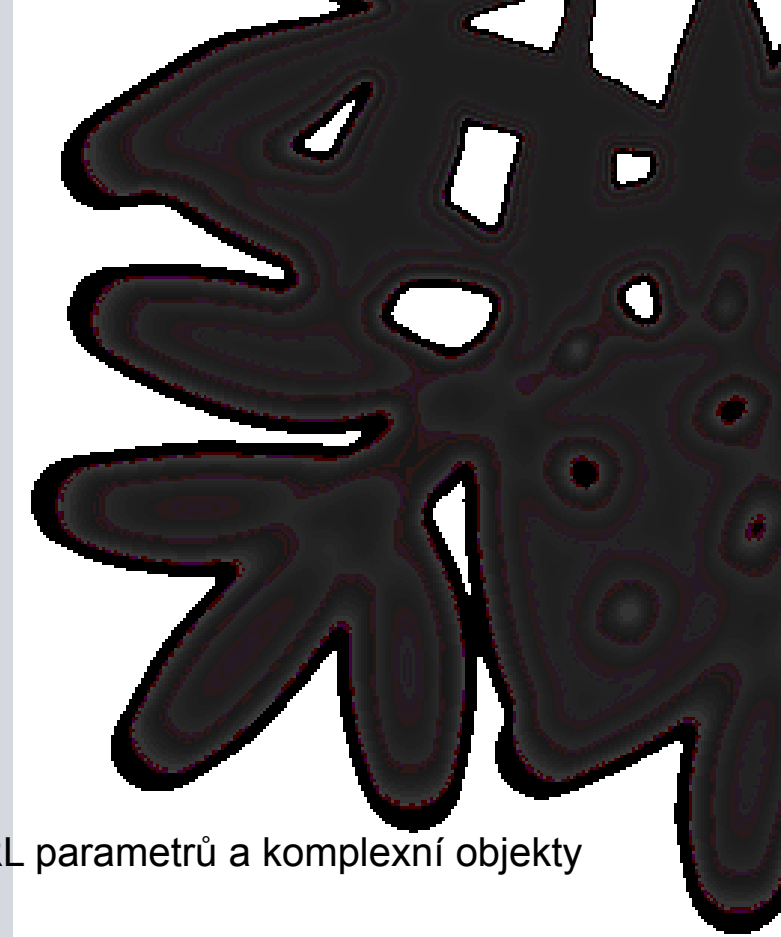
- Soubory .cshtml (Razor View Engine)
- Jde v nich psát standardní HTML, ale také vložit kousky C# kódu
  - Přepnutí do C# kontextu: @(výraz), @{blok kódu}
  - Přepnutí zpět do HTML kontextu: @:html, <validni_html_tag>…</v_h_t>
- Jdou komponovat
  - Např. _Layout.cshtml na společnou kostru všech HTML stránek
  - `@Html.Partial(„viewPath“)` vykreslí jiné view uvnitř 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
  - `Url` – pomocné metody ke generování URL
  - `User` – právě přihlášený uživatel



Model binding

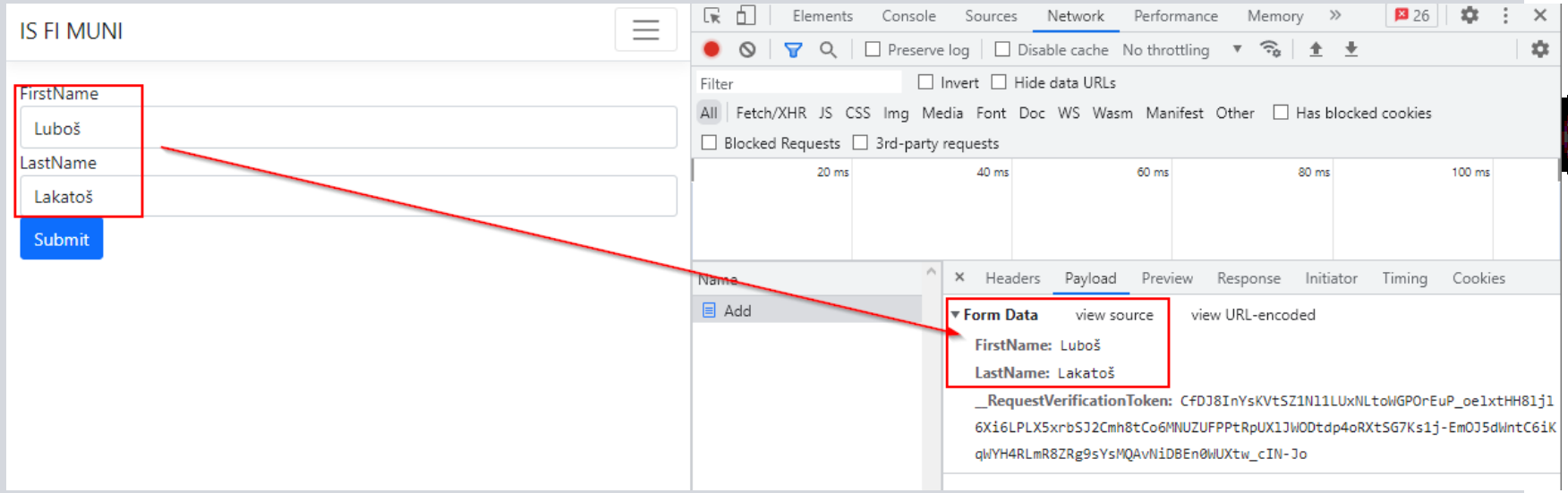


- Proces namapování dat poslaných klientem do podoby .NET objektů
- Zdroje dat mohou být různé:
  - HTTP formulář
  - URL parametry
  - Headery
  - Cookies
- Tyto namapované objekty se controlleru předávají jako argumenty metody
- Model binding defaultně funguje tak, že jednoduché objekty (int, string, apod.) mapuje z URL parametrů a komplexní objekty (třídy) z formulářových dat
- 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));
}
```

model {Lab08_Solution.Models.StudentU
FirstName View "Luboš"
Id 0
LastName View "Lakatoš"