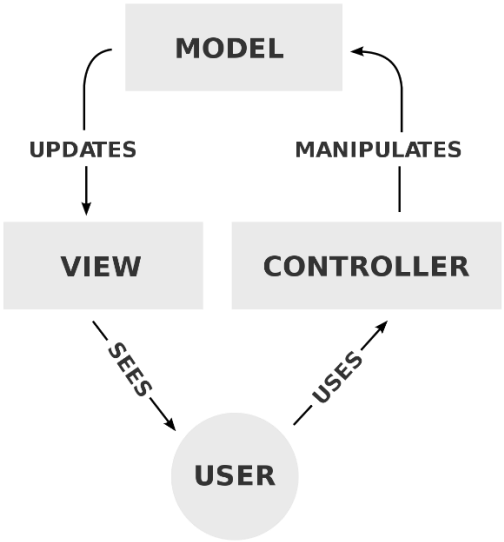
7- Frameworky architektury MVC

MVC – Model-View-Controller

- Architektonický vzor
- Populární zejména na webu – ASP.NET MVC, Laravel, Ruby on Rails, Django
- Odděluje logiku od výstupu
- Rozděluje se na Model, View a Controlle
- Výhody:
    - Díky oddělení logiky od výstupu je aplikace jednoduše udržitelná a rozšiřitelná
    - Frontend vývojáři teoreticky nemusí vědět, jak funguje backend, a obráceně
- Nevýhody
    - Každý request je routován přes kontroler do modelu, který pak renderuje view
    - Tento proces je vyžadován i při menších změnách ve vizuální části
    - Při více těchto requestech může být pomalé a časově náročné



-

Model

- Logika aplikace (práce s databází, výpočty)
- Pouze přijímá a vydává data
- Neví odkud data přišla ani co se s nimi bude dít

View

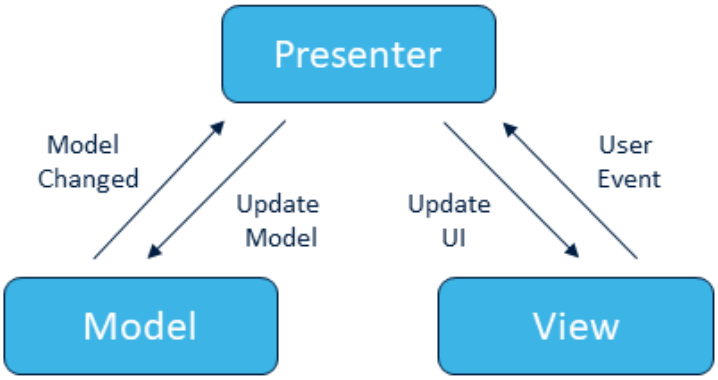
- Zobrazuje data uživateli
- Nezná původ dat
- Obsahuje minimum logiky
- HTML, XAML

Controller

- Propojuje View a Model
- Převádí výstup od uživatele na příkazy pro Model/View

MVP – Model-View-Presenter

- Odvozeno od MVC
- Používání pro Android, PHP aplikace
- Rozděluje se na Model, View a Presenter
- Model – poskytuje data, která mají být zobrazena
- View – zobrazuje data a posílá akce Presenteru
- Presenter:
    - Pracuje s modelem i view, podle dat získaných z modelu formátuje view (model mu hlásí změny opět událostmi)
    - Uchovává proměnné
    - Zpracovává reakce uživatele
- Oproti MVC má každé view svého presentera, v presenterovi se nachází více logiky
    - Controllery v MVC se dělí spíše podle toho, s jakými daty manipulují, a tak view pracují s více controllery, které na ně nejsou tak vázané (a mohou pracovat s více view)
- Výhoda – kompletní separace model a view, presenter je middle man
- Nevýhoda – hodně kódu pro propojení view + presenter; presenter je příliš vázaný na dané view



MVVM (Model-View-ViewModel)

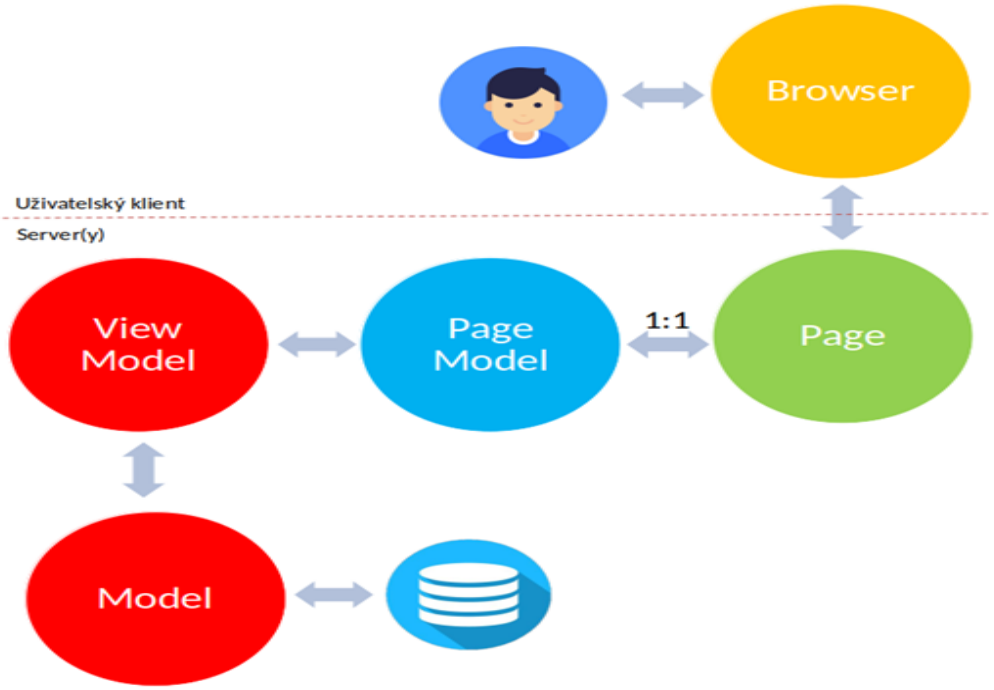
- Známý také jako Model View Binder
- Používáno u UWP a Xamarinu
- Model – poskytuje data, která mají být zobrazena
- View – definuje strukturu a rozložení stránky, s ViewModelem komunikuje pomocí bindingu
- ViewModel – převádí data z modelu do prezentovatelné podoby ve View

Binding

- Vytvoření vazby mezi vlastnostmi poskytovatele (ViewModel) a konzumenta (View)
- Data jsou synchronizována
- Poskytovatel při změně upozorní všechny konzumenty (existuje i obousměrný binding)
- WPF, UWP, Xamarin

PageModel

- Další modifikace MVC
- Controller (Presenter)+View nahrazeno 1:1 entitou Page+PageModel
- Model – data a stav aplikace
- Page – šablona dané stránky
- PageModel – datová a aplikační logika pro danou stránku
- Data připravená pro danou stránku mohou být upřesněna ve ViewModelu



```razor
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<h4>Uživatelé</h4>

<table class="table table-sm">
    <tbody>
        @foreach (var e in Model.GetUsers()) {
            <tr>
                <td>@e.username</td>
            </tr>
        }
    </tbody>
</table>
```

```razor
public class IndexModel : PageModel
{
    private IModel s;

    public IndexModel(IModel s)
    {
        this.s = s;
    }

    public List<User> GetUsers() => s.GetUsers();
}

// Startup.cs:
services.AddScoped<IModel, Model>();
```

Webové aplikace klient-server

- Klient si zobrazí aplikaci běžící na serveru
- Komunikace Request (žádost o data) -> Response (vrácení dat)
- 1** - informační odpověď
- 2** - žádost byla dodána a přijmuta
- 3** - přesměrování
- 4** - chyba na straně uživatele
- 5** - chyby na straně serveru
- Příklady:
    - 200 – OK
    - 201 – Created
    - 400 – Bad Request
    - 401 – Unauthorized
    - 403 – Forbidden

- 404 – Not Found
- 408 – Request Timeout
- 500 – Internal Server Error
- 503 – Service Unavailable