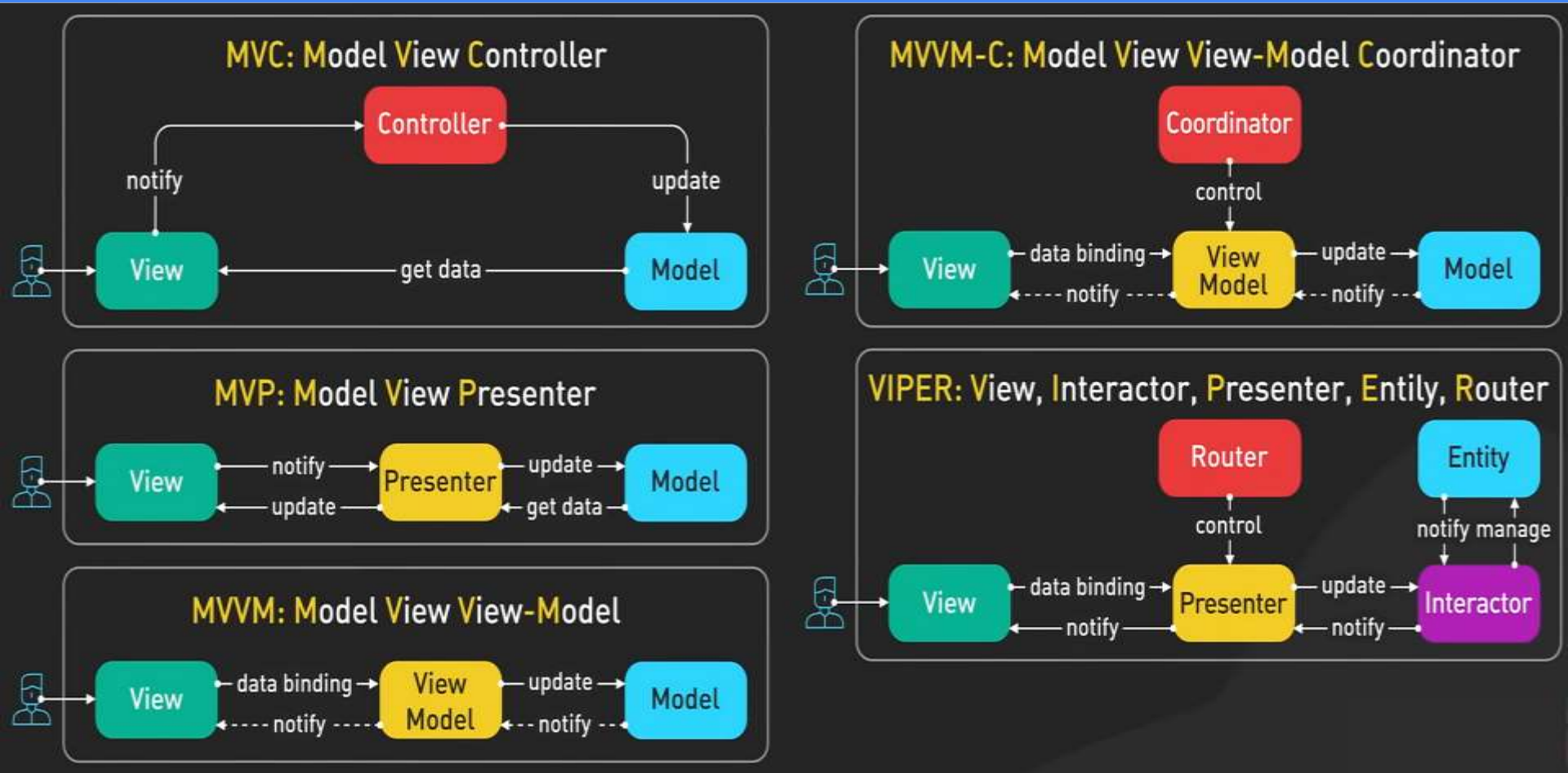
Model
View
Controller

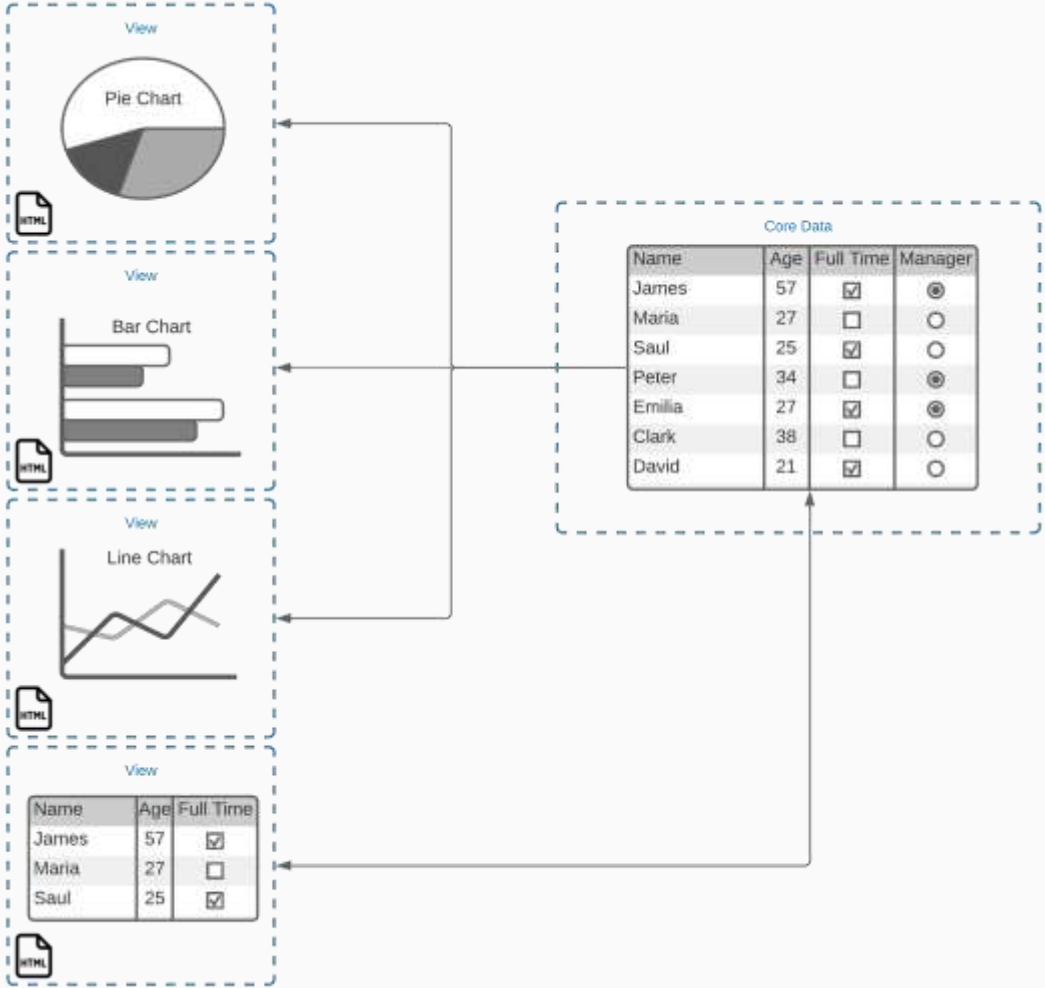
# MVVM

# Srovnání

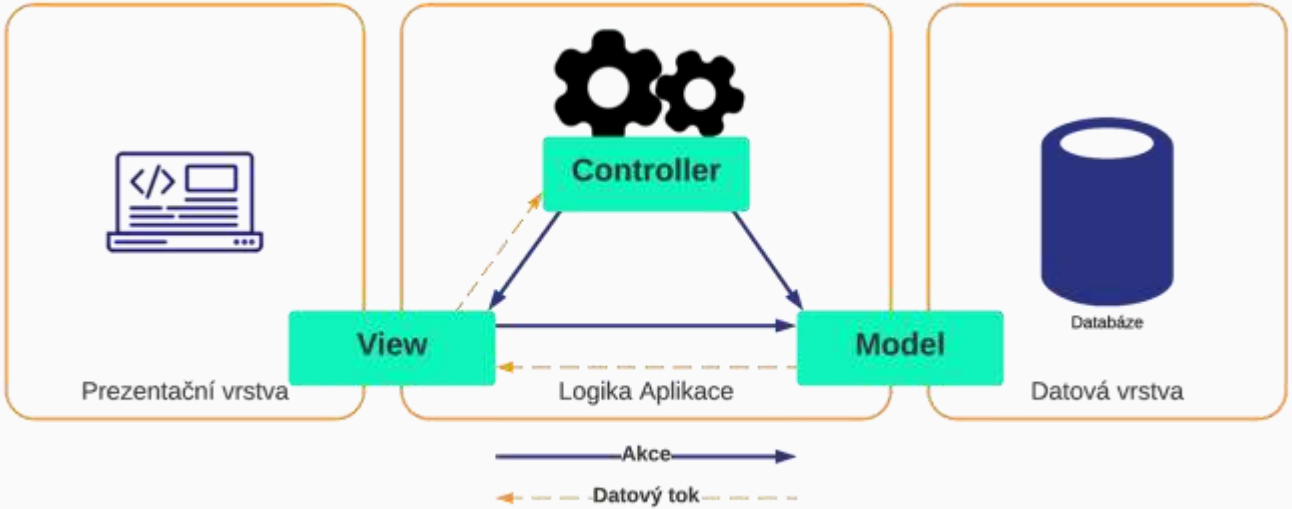

MVC: Model View Controller
Controller
notify
update
View
get data
Model
MVVM-C: Model View View-Model Coordinator
Coordinator
control
View
data binding
notify
View Model
update
notify
Model
MVP: Model View Presenter
View
notify
update
Presenter
update
get data
Model
VIPER: View, Interactor, Presenter, Entily, Router
Router
Entity
control
notify manage
View
data binding
notify
Presenter
update
notify
Interactor
MVVM: Model View View-Model
View
data binding
notify
View Model
update
notify
Model

# Motivace pro použití MVC

- Separace prezentace společného datového zdroje
- Modulární architektura
- Zásahy do implementace bez nutné změny modelu
- Portabilita zdrojového kódu
- Podpora pro SEO
- Podpora asynchrnonních technik



# Model-View-Controller (MVC) - Přehled




**Model-View-Controller** je architektonický vzor, definující návrh softwaru s interaktivním uživatelským rozhraním, který interně odděluje zpracování dat od vstupů a výstupů.

Sestává ze 3 částí (může mít i větší granularitu)

- ***Model*** - logika aplikace + data
- ***View*** - zobrazení informací uživatelům
- ***Controller*** - zpracování uživatelských vstupů

# Model-Vie del

nesmyslně tlustá hlavička - redukovat na třetinu

**Model**

- Společná komponenta pro všechny **Views** a **Controllers**
  - Jednotlivé views přistupují konzistentně ke stejným datům
- Je funkční jádro aplikace (back-end)
  - Návrh struktury aplikace nebo sub-systému v aplikaci
- Obsahuje data a reprezentuje stav systému
  - Umožňuje skrze Observer NV přístup pro **View** a **Controller** komponentu
- Skrze své API komunikuje s **View** i **Controller** komponentou
  - Zajišťuje změnu stavu aplikace a jejich dat
- Pomocí mechanismu *Change-Propagation* si udržuje spojení ke všem závislým částem (**View**, **Controller**)
  - Umožňuje tím propagovat změny v modelu těmto závislým komponentám

# Model-View-Controller (MVC) - View

**View**

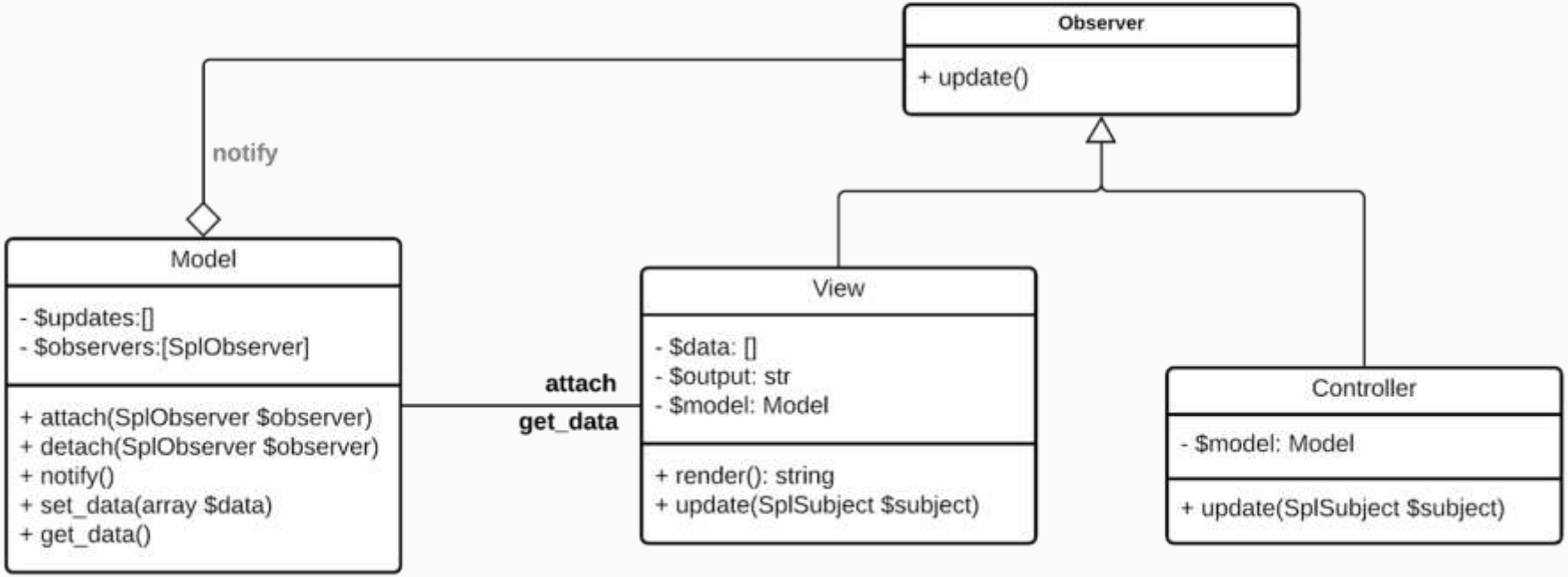
- Zobrazuje informace uživateli
  - Data pro zobrazované informace čerpá z ***Model*** komponenty
- MVC běžně sestává z více ***Views***
  - Každé ***View*** může zobrazovat informace jiným způsobem
- ***Views*** se registrují při inicializaci aplikace do ***Modelu*** a ke každému **View** se přidá ***Controller***
  - Vztah mezi Views a Controllers je tedy 1:1
- ***View*** reprezentuje Front-End aplikace

# Model-View-Controller (MVC) - Controller

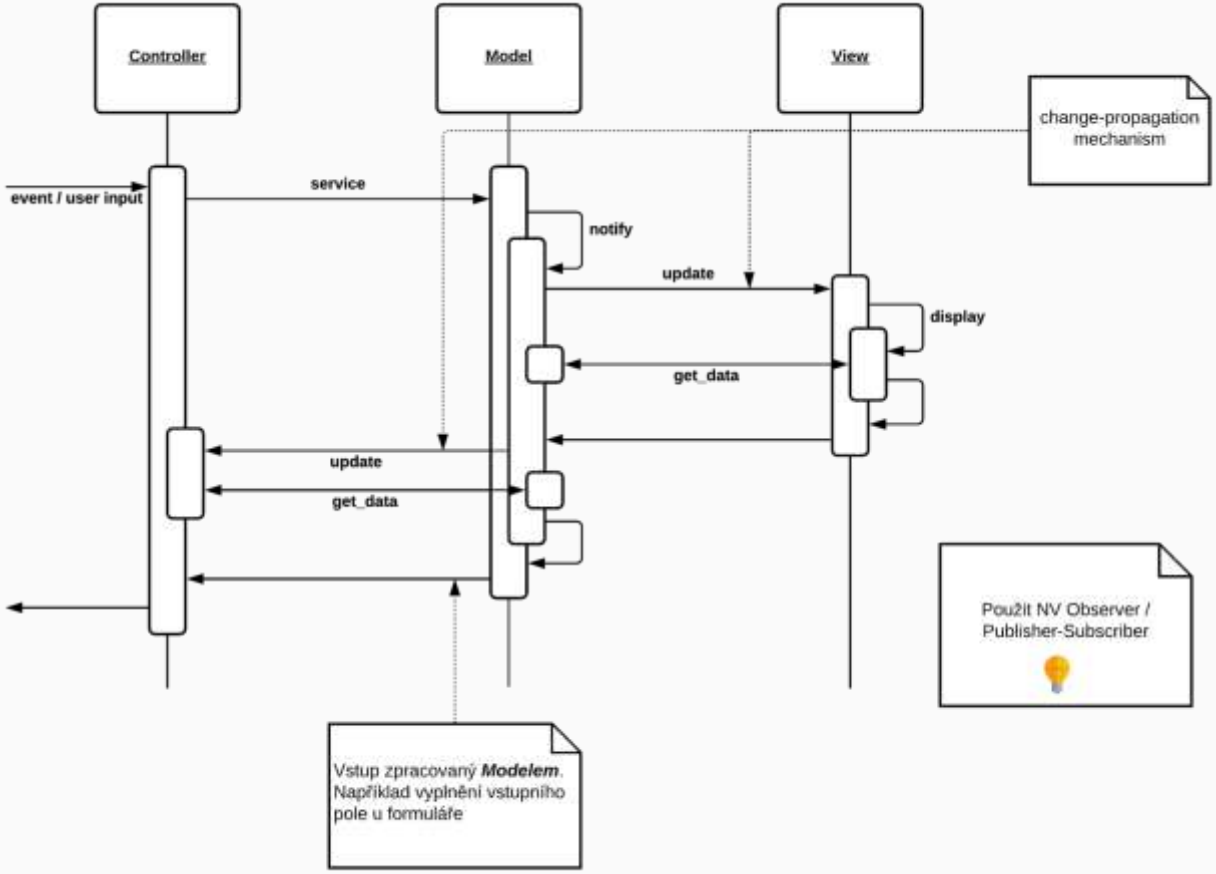
**Controller**

- Přijímá vstupy od uživatele
- Přijatý vstup následně předá jako příkaz pro ***View*** nebo ***Model***
  - Příklad vstupu pro ***View*** - scrolling
  - Příklad vstupu pro ***Model*** - datová položka formuláře
- Může zodpovídat za validaci a sanitizaci vstupu
- U webových aplikací vrací na každou uživatelskou interakci nový ***View***

# MVC - Návrh - Class diagram



# MVC - Návrh - Sequence diagram



# MVC - Ukázková implementace v PHP - Definice Modelu

```php
<?php

/**
* Parent class for custom model classes.
*
* This class was implemented li
*/
class Model implements SplSubje

    private $observers;

    private $updates = [];

    public function __construct(
        $this->observers = new S
    }

    public function attach(SplOb
        if ($observer instanceof
            $this->observers->at
        }
    }
…
```

```php
…

    public function detach(SplObserver $observer) {
        if ($observer instanceof View) {
            ers->detach($observer);


            y() {
            bservers as $value) {
            e($this);


            rray $data) {
            array_merge_recursive($this-


            ata(): array {
        return $this->updates;
    }
}
```

Ne světlé písmo na tmavém pozadí !!!

Kód ne jako obrázky, ale jako text !

# MVC - Ukázková implementace v PHP - Definice View

```php
<?php

/**
* Parent class for custom view classes.
*
* This class was implemented like part
* of the Observer pattern
*/
class View implements SplObserver {

    protected $data = [];

    protected $output = '';

    /**
     * @var Model Model for access data
     */
    protected $model;

    public function __construct(Model $model) {
        $this->model = $model;
    }

…
```

```php
…

    /**
     * Render a template.
     */
    public function render(): string {
        return $this->output;
    }

    /**
     * Update Observer's data.
     *
     * @param SplSubject $subject
     */
    public function update(SplSubject $subject) {
        if ($subject instanceof Model) {
            $this->data = array_merge($this->data, $subject->get());
        }
    }
}
```

# MVC - Ukázková implementace v PHP - Definice Controlleru

```php
<?php

/**
* Parent class for custom controller classes.
*
* This class was implemented like part
* of the Observer pattern
*/
class Controller implements SplObserver {

   /**
    * @var object The model object for current controller
    */
   protected $model = null;


   public function __construct(Model $model) {
       $this->model = $model;
   }

    /**
    * Update Observer's data.
    *
    * @param SplSubject $subject
    */
   public function update(SplSubject $subject) {  // NOP  }
   }
}
```

# MVC - Ukázková implementace v PHP - Kalkulačka

```php
<?php

/**
* Custom Model implementation - Calculator
*/
class CalculatorModel extends Model {

    public function __construct() {
        parent::__construct();
    }

    public function multiply(array $numbers) {
        $this->set([
            'result' => $this->operation('*', $numbers),
            'operands' => $numbers,
        ]);
    }

    public function divide(array $numbers) {
        $this->set([
            'result' => $this->operation('/', $numbers),
            'operands' => $numbers
        ]);
    }

…
```

```php
…

    public function sub(array $numbers) {
        $this->set([
            'result' => $this->operation('-', $numbers),
            'operands' => $numbers
        ]);
    }

    public function add(array $numbers) {
        $this->set([
            'result' => $this->operation('+', $numbers),
            'operands' => $numbers
        ]);
    }
    /**
    * Performs mathematician operation over given arguments.
    *
    * POC example. TODO: Implement
    */
    private function operation(string $operator, array $numbers) {
      //NOP
    }
}
```

# MVC - Ukázková implementace v PHP - Kalkulačka

```php
<?php

/**
* Custom View implementation - Calculator
*/
class CalculatorView extends View {

    public function __construct(CalculatorModel $model) {
        parent::__construct($model);
    }

    public function build_multiply_output() {
        $this->output = $this->generateOutput(
            'Multiplication', $this->data['operands'],
            $this->data['result']);
    }

    public function build_divide_output() {
        $this->output = $this->generateOutput(
            'Division', $this->data['operands'],
            $this->data['result']);
    }

    public function build_add_output() {
        $this->output = $this->generateOutput(
            'Addiction', $this->data['operands'],
            $this->data['result']);
    }
…
```

```php
…

    public function build_sub_output() {
        $this->output = $this->generateOutput(
            'Subtraction', $this->data['operands'],
            $this->data['result']);
    }

    /**
     * Generates the output.
     * Code for POC purposes. No data validation.
     */
    private function generateOutput(string $operation,
                        array $operands, $result): string {
        $string = $operation . ': ';

        foreach ($operands as $value) {
            $string .= $value . ' * ';
        }

        $string = substr($string, 0, strlen($string) - 2);
        $string .= '= ' . $result;

        return $string;
    }
}
```

# MVC - Ukázková implementace v PHP - Kalkulačka

```php
<?php

/**
* Custom Controller implementation - Calculator
*/
class CalculatorController extends Controller {

    public function __construct(CalculatorModel $model) {
        parent::__construct($model);
    }

    public function multiply_endpoint(... $numbers) {

        $this->checkOperands($numbers);
        //manipulate the model
        $this->model->multiply($numbers);
    }

    public function divide_endpoint(... $numbers) {

        $this->checkOperands($numbers);
        //manipulate the model
        $this->model->divide($numbers);
    }
…
```

```php
…

public function sub_endpoint(... $numbers) {

        $this->checkOperands($numbers);
        //manipulate the model
        $this->model->sub($numbers);
    }

    public function add_endpoint(... $numbers) {

        $this->checkOperands($numbers);
        //manipulate the model
        $this->model->add($numbers);
    }


    /**
     * Check minimum numbers required for operations.
     */
    private function checkOperands(array &$numbers) { // NOP }

}
```

# MVC - Ukázková implementace v PHP - Inicializace

```php
<?php

// Create components
$model = new CalculatorModel();
$view = new CalculatorView($model);
$controller = new CalculatorController($model);

// Attach the observers to the subject
$model->attach($view);

$model->attach($controller);

// Call the controller
$controller->multiply(2, 3);

// Notify the observer about changes
$model->notify();

// Call the view
$view->multiply();

// Get the output
echo $view->render(); // Example output: "Multiplication: 2 * 3 = 6"
```

# MVC
## Použití

***Desktopové aplikace***
- Aplikace s GUI a update mechanismem

***Webové aplikace***
- Obvykle bez explicitní implementace update mechanismu
- Většina velkých webových aplikací

***MVC Framework***
- Znovupoužitelné ***Views*** a ***Controllers***
- Předdefinovaná tlačítka, menu, formuláře
- Snižování platformní závislosti pro ***Views*** a ***Controllers***
- Znovupoužití návrhových/architektonických vzorů
  - Bridge pattern - snížení závislosti na platformě
  - Composite pattern - management komponent v rámci ***View***
  - Chain of Responsibility - delegace eventů přes ***Views***
  - Strategy - agregace algoritmů

# MVC
## Výhody použití

+ Možnost vizualizovat stejná data pomocí vícero ***Views***
+ Při updatu ***Modelu*** synchronizace s ***Views***
+ Modifikace ***Views*** a ***Controllers*** bez nutného zásahu do ***Modelu***
+ ***Model*** je nezávislý na platformě, tedy i portabilní
+ Aplikace ve Frameworcích
+ Modularita a paralelní vývoj
+ Možnost jednoduchého rozšíření ***Views***
+ Logické členění
+ Jde ruku k ruce s TDD
+ Modelová komponenta může být testována nezávisle na uživatelích

# MVC

## Nevýhody použití

- Pro malé projekty složitý návrh a implementace
- ***View*** a ***Controller*** jsou obyčejně platformě závislé
- Velké množství variací MVC - čistá forma se již objevuje ojediněle
- Úzké spojení ***View*** a ***Controlleru***
  - Změna API v ***Modelu*** může vést změnu v každém ***View*** a ***Controlleru***
- Častá miskoncepce - ***Controller*** slouží jako separace ***View*** od ***Modelu***
- Neefektivní sběr dat z ***Modelu***
- Implementace *update* mechanismu - jednotlivé ***Views*** a ***Controllers*** nemusí stíhat objem dat a frekvenci *updatů*

# MVC

## Přehled Frameworků

- ASP.NET MVC
- Laravel
- Angular
- React
- Vue
- Django
- Flask
- Ruby on Rails
- Symfony
- Spring MVC

# MVVM

# Motivace

- Zlatý standard u WPF, UWP aplikací
  - Autor John Gossman – WPF Architekt
- Chci backend nezávislý na UI
- Chci testovat
- Chci být nezávislý na platformě

(1)

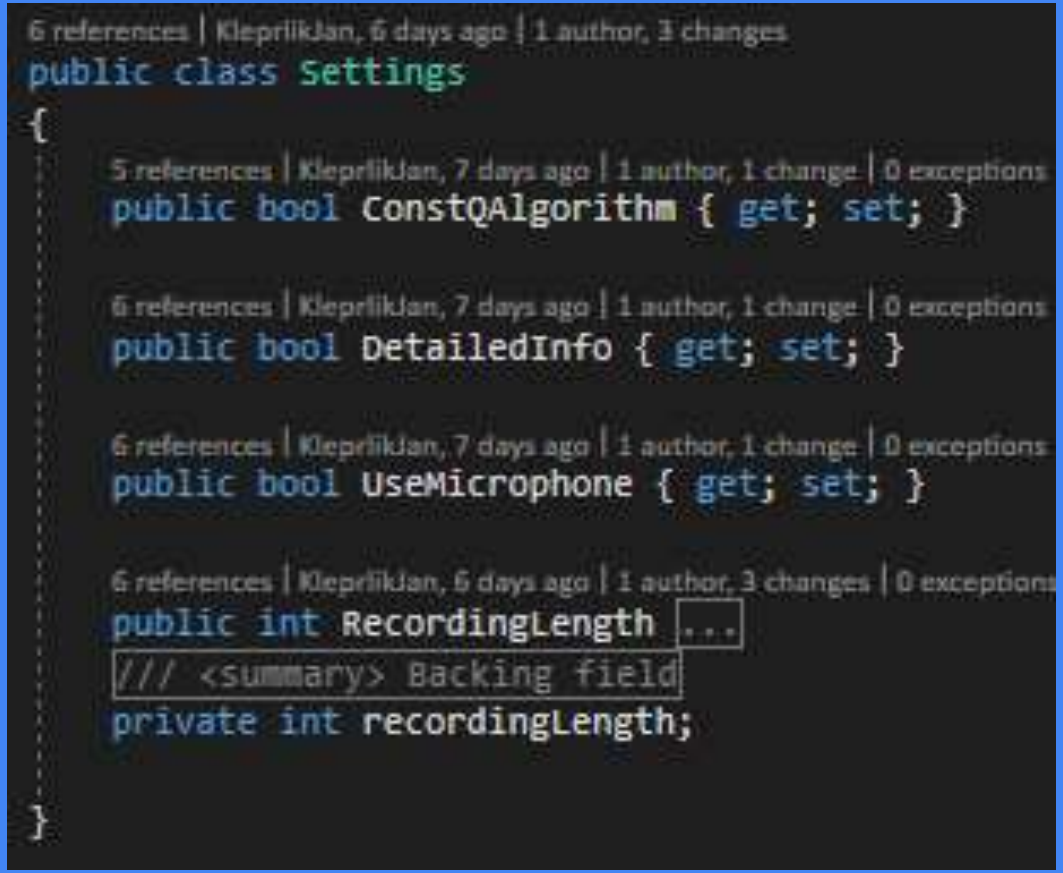
# View

- The UI
- Zobrazuje data

```
<StackPanel x:Name="Options">
    <ToggleSwitch
        x:Name="ConstQAlg"
        OnContent="Constant Q algorithm"
        OffContent="Constant Q algorithm"
        IsOn="{x:Bind SettingsViewModel.ConstQAlg, Mode=TwoWay}"
        />
    <ToggleSwitch
        x:Name="DetailedInfo"
        OnContent="Detailed info"
        OffContent="Detailed info"
        IsOn="{x:Bind SettingsViewModel.DetailedInfo, Mode=TwoWay}"
        />
    <Slider
        x:Name="RecordingLength"
        Minimum="3"
        Maximum="8"
        Header="{x:Bind SettingsViewModel.RecordingLengthText, Mode=OneWay}"
        Value="{x:Bind SettingsViewModel.RecordingLength, Mode=TwoWay}"
        />
</StackPanel>
```

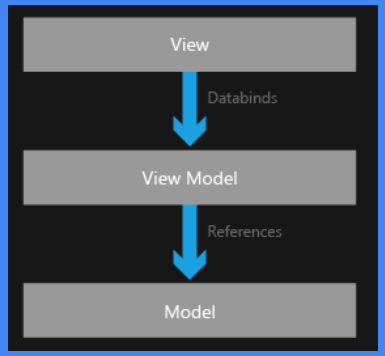
# Model

- Uchovává data



# ViewModel

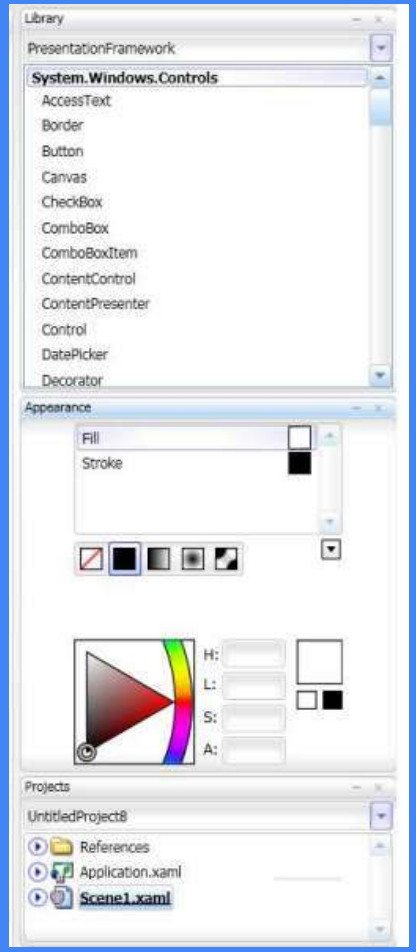
- Zpřístupňuje Model pro View
  - DataBinding
  - Commands
- Model typ -> View typ
- Uchovává stav View



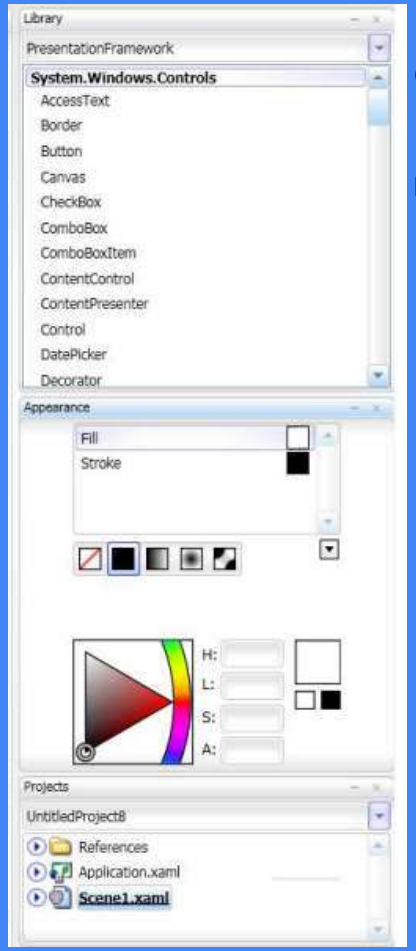

(2)

```
14 references | KleprlikJan, 5 days ago | 1 author, 3 changes
public class SettingsViewModel : BaseViewModel
{
    1 reference | KleprlikJan, 6 days ago | 1 author, 1 change | 0 exceptions
    public SettingsViewModel(Settings settings = null) => Settings = settings
    private Settings settings;
    10 references | KleprlikJan, 6 days ago | 1 author, 1 change | 0 exceptions
    public Settings Settings
    {
        get => settings;
        set
        {
            settings = value;
            OnPropertyChanged(string.Empty);
        }
    }

    8 references | KleprlikJan, 6 days ago | 1 author, 1 change | 0 exceptions
    public bool ConstQAlg
    {
        get => Settings.ConstQAlgorithm;
        set
        {
            Settings.ConstQAlgorithm = value;
            OnPropertyChanged();
        }
    }
```




(3)




(3)

ComboBox

ListBox

Model = List of Assemblies
- List of controls

ViewModel
- selected Control
- InsertIntoScene()

# malý vs VELKÝ projekt

**PRO**

- Jednoduché rozšíření
- Přehledné
- Snadné úpravy
- Podpora testování
- Lze využít jen části, které potřebuji (DataBinding)

**PROTI**

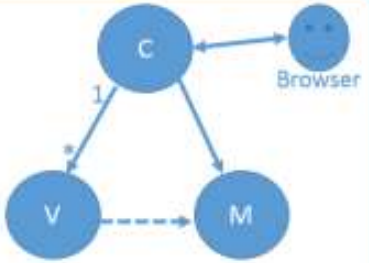
- Learning curve
- Více kódu
- Záhada DataBindigu

# MVVM vs MVC vs MVP

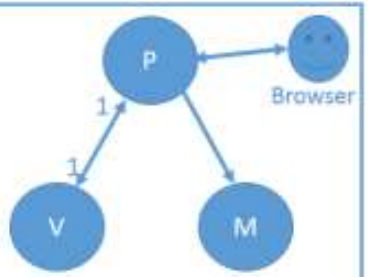
- Doplňují se
  - ViewModel může sloužit jen jako bridge pro data
- ViewModel není závislý na View
- Jedině ViewModel provádí změny na modelech
- View nemá o modelech ponětí a Model nemá ponětí o View


(4)

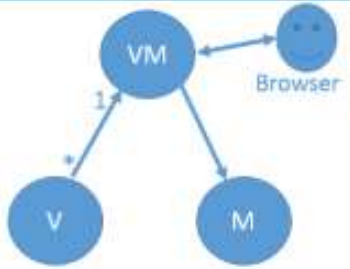
# MVC – MVP - MVVM




- Controller is the entry point to the application
- One to Many relationship between Controller and View
- View does not have reference to the Controller
- View is very well aware of the Model
- Smalltalk, ASP.Net MVC



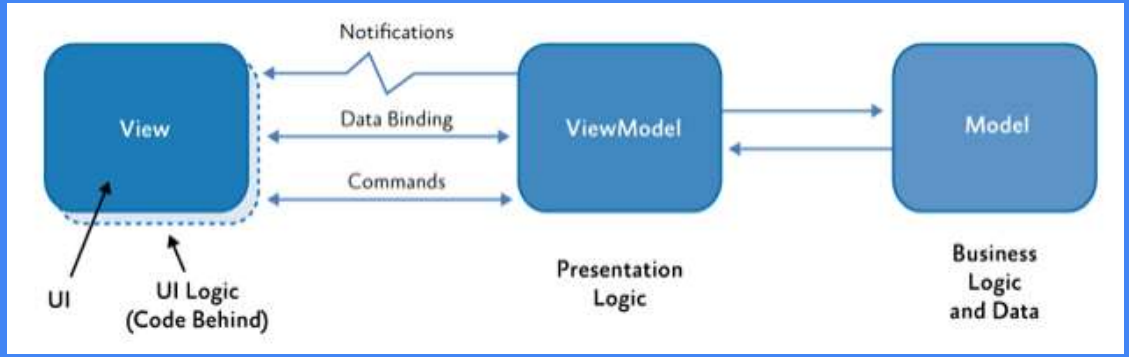

- View is the entry point to the application
- One to One mapping between View and Presenter
- View have the reference to the Presenter
- View is not aware of the Model
- Windows forms




- View is the entry point to the application
- One to Many relationship between View and ViewModel
- View have the reference to the View Model
- View is not aware of the Model
- Silverlight, WPF, HTML5 with Knockout/AngularJS

# Shrunutí

- Architektonický vzor
- Využívá DataBinding
- ViewModel neví co dělá View
- Model neví o View ani ViewModelu
- ViewModel zpřístupňuje data Modelu a commandy pro View




(6)

# Zdroje obrázků:

- (1) - http://www.thomas-weller.de/en/pages/cross-platform-development/
- (2) - https://docs.microsoft.com/en-us/previous-versions/windows/apps/jj721615(v=vs.105)?redirectedfrom=MSDN
- (3) - https://docs.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps
- (4) - https://www.creativereview.co.uk/make-love-not-war-slogan/
- (5) - https://i.stack.imgur.com/ENBf1.png
- (6) - http://web.csulb.edu/~pnguyen/cecs475/pdf/intromvvm%20ver%202.pdf

# Zdroje informací


- https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-and-mvvm
- https://russelleast.wordpress.com/2008/08/09/overview-of-the-modelview-viewmodel-mvvm-pattern-and-data-binding/
- https://stackoverflow.com/questions/667781/what-is-the-difference-between-mvc-and-mvvm
- https://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained
- https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern
- http://www.uniporttogo.com/img/uploads/2015/06/Advanced_MVVM.pdf
- https://stackoverflow.com/questions/19444431/what-is-difference-between-mvc-mvp-mvvm-design-pattern-in-terms-of-coding-c-s
- https://stackoverflow.com/questions/6789236/how-does-wpf-inotifypropertychanged-work