

# Vue.js

kurz Vývoj progresívnych  
webových aplikácií

Eduard Kuric



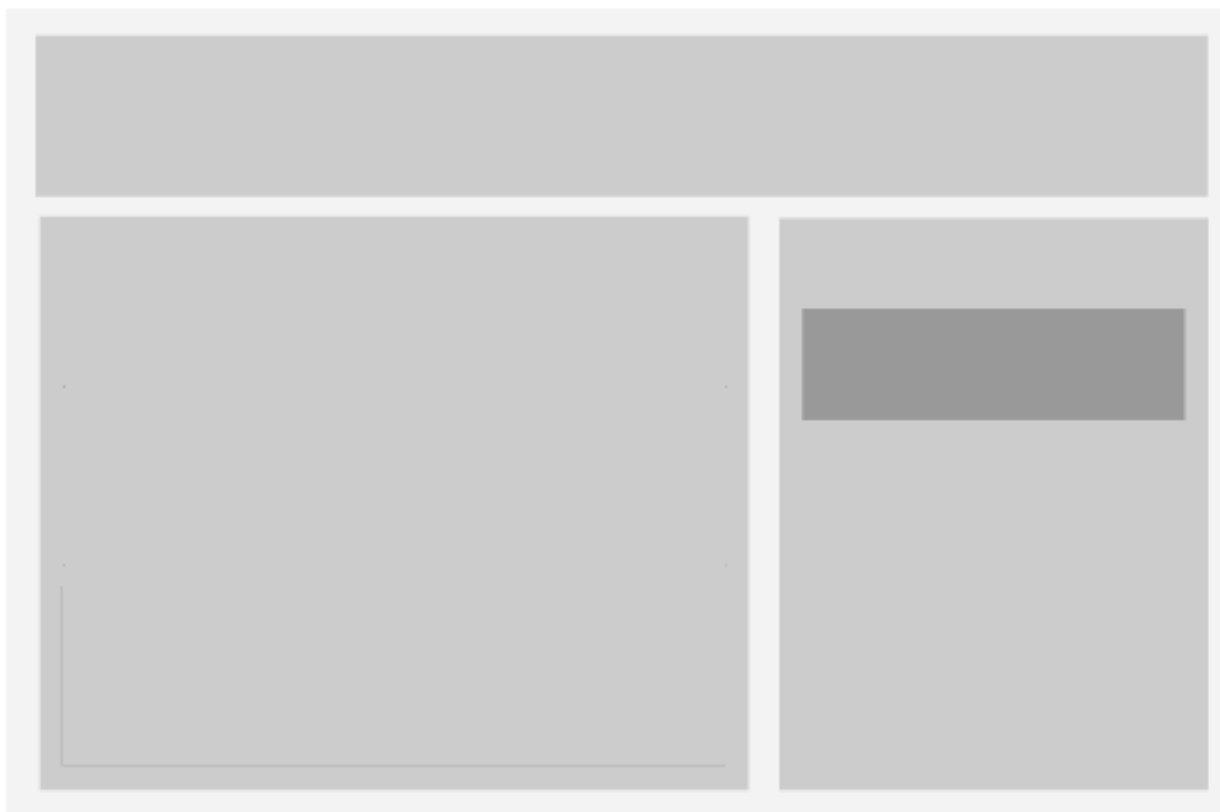


# Vue.js

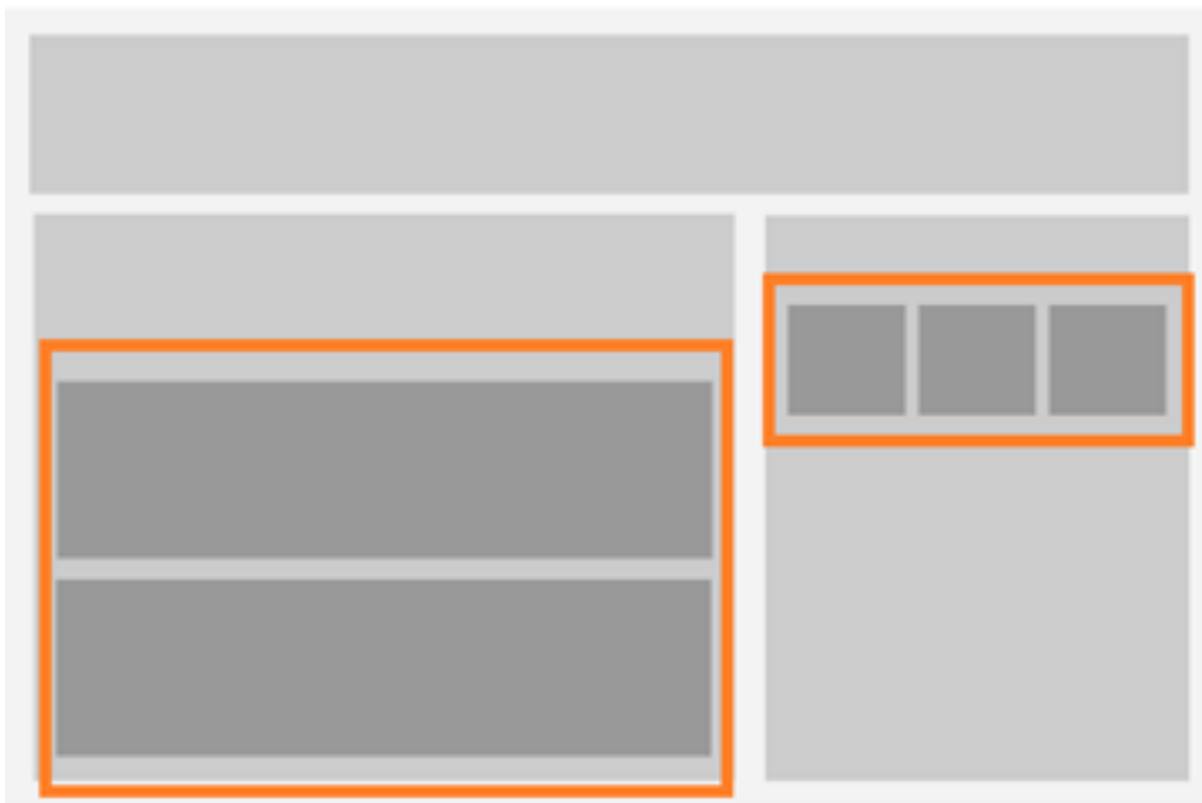
- populárny progresívny JS rámec na tvorbu **plne-interaktívneho používateľského rozhrania** webových stránok/aplikácií
- vytvoril ho Evan You po tom, ako pracoval v Googli, kde používal AngularJS
  - extrahoval a preniesol do Vue najlepšie koncepty z Angularu
  - prvá oficiálna verzia 2014

[porovnanie s inými rámcami](#)

Komponentom môže byť  
jeden fragment

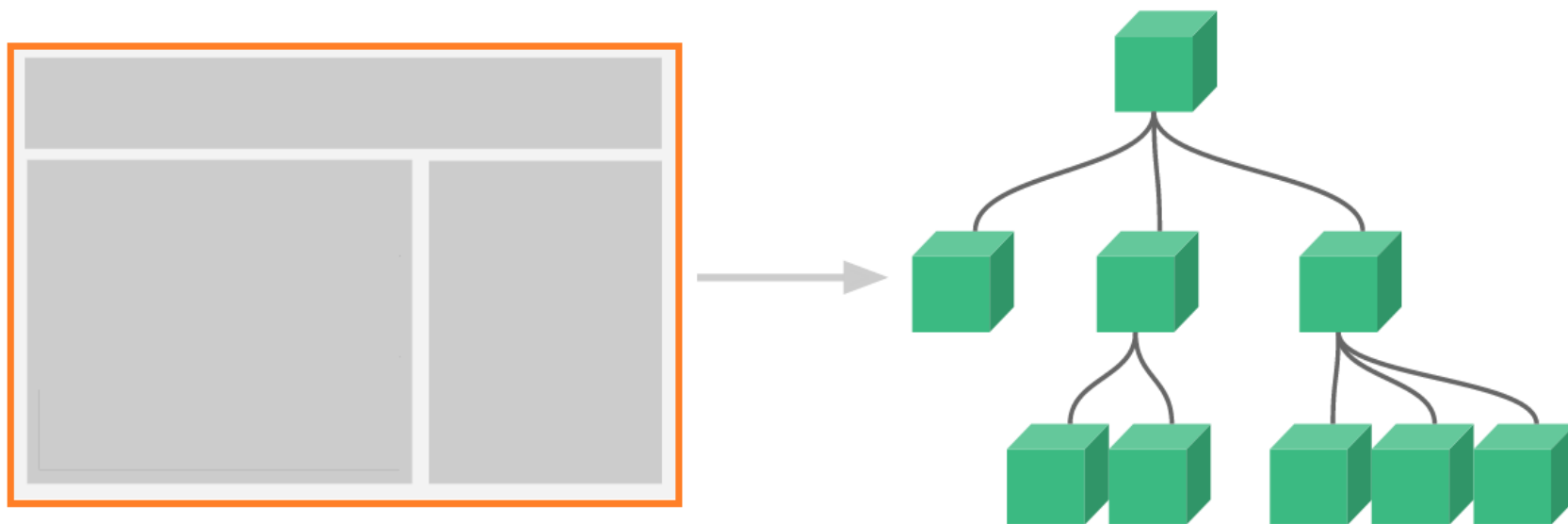


Komponent může být  
zložený z komponentov



# Celá stránka je komponent zložený z komponentov

- **plne interaktívne dynamické stránky/aplikácie**
  - **SPA – Single Page Application**



# Single file components

*In modern UI development, we have found that instead of dividing the codebase into three huge layers (template, logic, style), it makes much more sense to divide them into loosely-coupled components and compose them.*

# Node.js

- **Nainštalujeme si Node.js**
- <https://nodejs.org/>
- **Vytvoríme si Vue project:**
- `npm init vue@latest`
- Odporúčam VS Code plus extension Volar  
<https://code.visualstudio.com/>
- <https://marketplace.visualstudio.com/items?itemName=johnsoncodehk.volar>

# Node.js /2

- `cd MyProjectName`
- `npm install`
- `npm run dev // zbudujeme projekt`



# Single file – hello.vue

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
  </div>
</template>

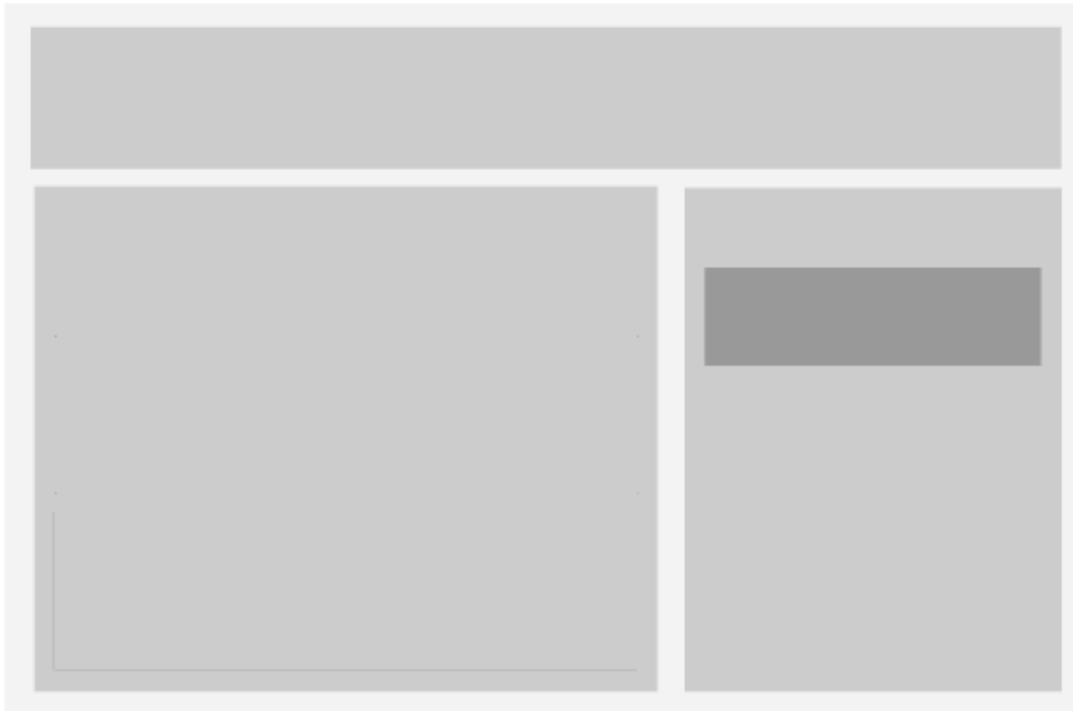
<script>
export default {
  data () {
    return {
      msg: 'Hello World!'
    }
  }
}
</script>

<style scoped>
h1 { color: #42b983; }
</style>
```



# Vue.js – příklad

- chceme na stránce interaktivny (dynamický) komponent
  - příklad, jednoduché počítadlo počtu kliknutí na tlačidlo



# Počítadlo kliknutí na tlačidlo

- vytvoríme koreňový komponent, ktorý zaobaluje ďalšie komponenty...

```
<html>
<head>...</head>
<body>
...
  <div id="counter-widget">
    </div>
...
</body>
</html>
```

# Počítadlo kliknutí na tlačidlo

- vytvoríme Vue.js inštanciu – reprezentáciu koreňového komponentu Vue.js aplikácie
- `#counter-widget` určuje, ktorý element je koreňový elementom komponentu

```
Vue.createApp({  
  }) .mount('#counter-widget')
```

# Počítadlo kliknutí na tlačidlo /2

- **zadefinujeme model komponentu (dátový objekt)** – reaktívne atribúty vystavené v šablóne komponentu

```
Vue.createApp({  
  data() {  
    return {  
      heading: 'Pocitadlo kliknuti',  
      message: 'Tento komponent zobrazuje...',  
      clickCounter: 0  
    }  
  },  
}).mount('#counter-widget')
```

# Počítadlo kliknutí na tlačidlo /3

- **dodefinujeme šablónu komponentu**

```
Vue.createApp({  
  data() {  
    return {  
      heading: 'Pocitadlo kliknutí na tlačidlo',  
      message: 'Tento komponent zobrazuje...',  
      clickCounter: 0  
    }  
  },  
}).mount('#counter-widget')
```

```
<div id="counter-widget">  
  <h1>{{ heading }}</h1>  
  <p>{{ message }}</p>  
</div>
```

# Počítadlo kliknutí na tlačidlo /4

- pridáme tlačidlo

```
Vue.createApp({  
  data() {  
    return {  
      heading: 'I  
      message: 'I  
      clickCounter: 0  
    }  
  },
```

```
<div id="counter-widget">  
  <h1>{{ heading }}</h1>  
  <p>{{ message }}</p>  
  <button v-on:click="buttonClick()">  
    Klikni na mňa  
  </button>  
</div>
```

```
}).mount('#counter-widget')
```

# Počítadlo kliknutí na tlačidlo /5

- **zadefinujeme metódu komponentu**

```
Vue.createApp({  
  data() {  
    return {  
      heading: 'Pocitadl  
      message: 'Tento ko  
      clickCounter: 0  
    }  
  },  
  methods: {  
    buttonClick: function() {  
      this.clickCounter++;  
    }  
  }  
}).mount('#counter-widget')
```

```
<div id="counter-widget">  
  <h1>{{ heading }}</h1>  
  <p>{{ message }}</p>  
  <button v-on:click="buttonClick()">  
    Klikni na mňa  
  </button>  
</div>
```



# Počítadlo kliknutí na tlačidlo /6

- **doplníme šablónu komponentu o zobrazenie aktuálnej hodnoty počítadla**

```
<div id="counter-widget">
  <h1>{{ heading }}</h1>
  <p>{{ message }}</p>
  <button v-on:click="buttonClick()">
    Klikni na mňa
  </button>
  <p>
    Počet kliknutí: {{ clickCounter }}
  </p>
</div>
```

# Reaktivita

- **nikde sme neriešili manipuláciu s DOMom!**
- `buttonClick()` iba inkrementujeme počítadlo `clickCounter`, tým sa ale automaticky zobrazí v rozhraní aj nová hodnota počítadla

<p>

Počet kliknutí: {{ **clickCounter** }}

</p>

# Reaktivita /2

- **zmena atribútov(dátového objektu)  
automaticky vyvolá prekreslenie (update)  
obsahu komponentu**
  - aktualizácia DOMu je vykonávaná asynchrónne pre vyšší výkon
- zriedkavo je potrebné dotknúť sa DOMu ako takého

# model-view-viewmodel

```
<body>
```

```
  <div id="app"></div>
```

```
  <script>
```

```
    Vue.createApp ( {
```

```
      data () {
```

```
    },
```

```
  ) .mount ( '#app' )
```

```
</script>
```

```
</body>
```

## **vm - viewmodel**

- inštancia Vue.js
- koreňovým komponentom
- synchronizácia rozhrania (view) a modelu (dát)

# model-view-viewmodel

```
<body>
  <div id="app"></div>
  <script>
    Vue.createApp({
      data() {
        ,
      }) .mount('#app')
    </script>
  </body>
```

## **data - model**

- dátový reaktívny objekt

# model-view-viewmodel

```
<body>
```

```
  <div id="app"></div>
```

```
  <script>
```

```
    Vue.createApp({  
      data() {  
        },  
    }).mount('#app')
```

```
  </script>
```

```
</body>
```

## **view (rozhranie)**

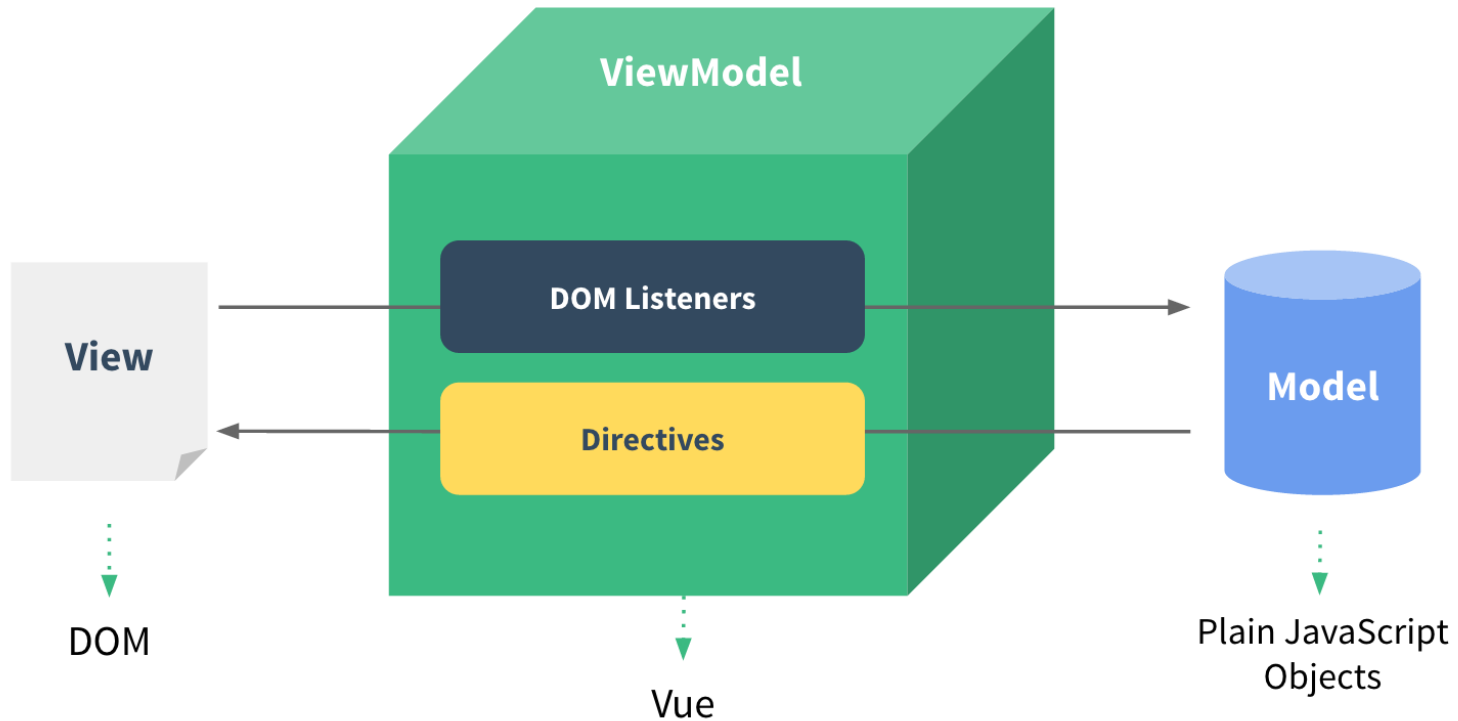
- koreňový DOM element (kontajner) komponentu

# Komponent

- je základnou stavebnou jednotkou
- je určený
  - **šablónou** (html + css) - ako vyzerá, napr. tlačidlo
  - **a správaním** – čo sa má vykonať, napr. keď používateľ klikne na tlačidlo



# Model-view-viewmodel





# Použitie komponentu

- element (otváracia/ukončovacia značka) je názov komponentu

```
<div id="vue-app">
```

```
  <!-- toto Vue transformuje na html sablonu -->
```

```
    <bbutton-counter></bbutton-counter>
```

```
</div>
```

# Znovupoužitie komponentu

```
<div id="vue-app">
```

```
  <button-counter></button-counter>
```

```
  <button-counter></button-counter>
```

```
  <button-counter></button-counter>
```

```
</div>
```

```
var mySPA = new Vue({ el: '#vue-app' })
```

# **data** musí byť funkcia

- každá inštancia komponentu si udržiava vlastný dátový objekt

```
data() {  
  return {  
    count: 0  
  }  
}
```

- ak by neboli dáta funkciou po kliknutí na tlačidlo by sa počítadlo inkrementovalo na všetkých “klonoch” predmetného komponentu

# One-way vs two-way data binding

- One-way: JS premenná je napojená na DOM
  - direktíva `v-bind`
- Two-way: Dáta sú napojené z DOMu späť na JS premennú
  - direktíva `v-model`
  - napr. vstupné pole, pri zmene vstupu sa aktualizuje premenná, pri zmene premnej sa aktualizuje vstup v poli
  - všeobecne formulárové elementy

# Lokálny komponent

```
<script>
import ButtonCounter from './components/ButtonCounter.vue'

export default {
  components: {
    ButtonCounter
  }
}
</script>

<template>
  <ButtonCounter />
</template>
```

# Globálny komponent

```
const app = createApp(App);  
app.component("MyComponent",  
MyComponent);  
app.mount("#app");
```

# Dynamický komponent

...

```
<component :is="currentComponent">  
</component>
```

...

Použitím špeciálneho atribútu **is**:

- môže byť názov registrovaného komponentu
- alebo importovaný component object

# Odovzdanie fragmentov vnoreným komponentom – **slots**

```
<CustomButton>  
  <!-- obsah slotu-->  
  <strong>Click me!</strong>  
</CustomButton>
```

```
<!-- CustomButton.vue component -->  
<template>  
  <button>  
    <b>slot</b>  
  </button>  
</template>
```



# Odovzdanie fragmentov vnoreným komponentom – **slots** /2

- pomenované sloty

```
<CustomLayout>
  <template v-slot:partOne>

    ...

  </template>
  <template v-slot:partTwo>

    ...

  </template>
</CustomLayout>
```

```
<!-- CustomLayout.vue component -->
<slot name="partOne"></slot>
<slot name="partTwo"></slot>
```

# Odovzdanie dát vnoreným komponentom - **props**

ButtonCounter.vue

```
data() {  
  return {  
    count: 0  
  }  
},  
props: {  
  title: String  
}  
  
<template>  
  <span>{{ title }}</span>  
</template>
```

# Posunutie dát vnoreným komponentom – **props** /2

Hodnotu posunieme komponentu cez atribút

```
<ButtonCounter title="Ahoj"></ButtonCounter>
```

- Pre dynamické props použijeme `v-bind` direktívu

# Vue.js – životný cyklus - hooks

1. inicializácia (vytváranie) komponentu
2. zostavenie – vloženie komponentu do DOMu a vykreslenie
3. aktualizácia komponentu – zmena a prekreslenie
4. Zrušenie komponentu

# beforeCreate, created

`beforeCreate() {}`

- počiatok inicializácie komponentu, dáta ešte nie sú reaktívne, udalosti (events) nie sú nastavené

`created() {}`

- možné pristúpiť k dátam, udalosti sú aktívne (šablóny nie, ani DOM)
  - fáza vhodná na získavanie dát (fetch)

# beforeMount, mounted

`beforeMount() {}`

- tesne predtým, ako sa začne komponent vykreslovať

`mounted() {}`

- plný prístup k reaktívnemu komponentu (šablóny, vykreslený DOM)
  - **fáza vhodná na modifikáciu DOMu**

# beforeUpdate, updated

`beforeUpdate() {}`

- keď nastane zmena dát, tesne predtým, ako je DOM zmenený a prekreslený

`updated() {}`

- po zmene dát a prekreslení komponentu
  - **najvhodnejšia fáza, ak chceme pristupovať k DOMu po tom, keď sa zmenili dáta**

# beforeUnmount, unmounted

`beforeUnmount() {}`

- komponent je zatiaľ plne funkčný
  - fáza vhodná, ak chceme napr. vykonať odregistrovanie udalostí

`unmounted() {}`

- komponent bol odobratý
  - fáza vhodná, napr. na informovanie vzdialeného servera



# Syntax šablóny komponentu

- HTML syntax - základ
- najzákladnejší spôsob previazania dát (data binding) je tzv. mustache syntax
  - `<span>Message: {{ msg }} </span>`

// hodnota bude do obsahu vlozena iba raz

// zmena/aktualizacia dat sa v obsahu neprejaví

`<span v-once>This will never change: {{ msg }}</span>`

# mustache vs. v-html

```
rawHtml = '<span style="color:red">This...</span>'
```

Using mustaches: `{{ rawHtml }}`

Using v-html directive

```
<span v-html="rawHtml"></span>
```

Using mustaches: `<span style="color: red">This should be red.  
</span>`

Using v-html directive: **This should be red.**

# Atribúty

- **mustache syntax nemôže byť použitá vo vnútri atribútov**
- v atribútoch použijeme direktívu `v-bind`

```
<div v-bind:id="dynamicId"></div>
```

```
<button v-bind:disabled="isButtDisabled">  
  Click  
</button>
```

```
<div v-bind:id="'list-' + id"></div>
```

# Štýly - triedy

- elementu bude nastavená trieda `active`, ak `isActive` je `true`

```
<div v-bind:class="{ 'active': isActive }"></div>
```

```
<div class="static"
```

```
  v-bind:class="{ 'text-danger': isActive }">
```

```
</div>
```

- zoznam/pole tried

```
<div v-bind:class="[activeClass, errorClass]"></div>
```

- podmienené

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>
```

```
<div v-bind:class="[{ active: isActive }, errorClass]"></div>
```

# Podmienky

```
<div v-if="clickCounter < 5" >
  <button v-on:click="buttonClick()">
    Klikni na mňa
  </button>
  Klikol si {{ clickCounter }}-krát.
</div>
<div v-else-if="clickCounter < 15" >
  <button v-on:click="buttonClick()">
    Klikaj ďalej, zotrvaj!
  </button>
  Klikol si {{ clickCounter }}-krát.
</div>
<div v-else>
  Sorry, už by stačilo klikania...
</div>
```

# Podmienky – v-show

- v-show – element nie je odstránený z DOMu stránky, ale má nastavený CSS `display: none;`

# Cyklus/slučka

```
<div v-for="item in items"  
      v-bind:key="item.id">  
  
</div>
```

- predvolené správanie – stratégia “in-place patch” miesto presúvania elementov, vue “pláta” obsah elementov, je to lacnejšie
- Ak poskytneme vue jednoznačnú identitu položiek – kľúč - :key, potom dokáže relatívne efektívne preusporiadať celé elementy, porovnajte pripojený príklad (s použitím kľúča a bez)
- <https://codepen.io/kurice/pen/ZEamyKw>

# Priority v-for, v-if

```
<li v-for="note in notes" v-if="!note.isImportant">
  {{ todo.name }}
</li>
```

**v-if** má väčšiu prioritu ako **v-for**, musíme **v-for** presunúť do wrappera template:

```
<template v-for="note in notes">
  <li v-if="!note.isImportant ">
    {{ todo.name }}
  </li>
</template>
```



# Udalosti

- použitím direktívy `v-on` môžeme načúvať DOM udalostiam
  - keď udalosť nastane, vykonáme nejaký JS

```
<div id="hello">  
  <button v-on:click="say('hi')">Say hi</button>  
</div>
```

```
methods: {  
  say: function (message) {  
    alert(message)  
  }  
}
```

# vlastnost' computed

```
data () {  
  return {  
    activeBook: 'Fahrenheit 451',  
    books: ['Heidi', 'Animal Farm', 'Don Quixote']  
  }  
},  
computed: {  
  currentBooksCount: function () {  
    return 'Books Count: ' + this.books.length  
  }  
}
```

# vlastnosť computed /2

```
computed: {  
  currentBooksCount: function () {  
    return 'Books Count: ' + this.books.length  
  }  
}
```

- keď sa zmení atribút **books**, je automaticky po ňom zmenený/prepočítaný aj atribút **currentBooksCount**

# computed vs. metóda

```
methods: {  
  currentBooksCount: function () {  
    return 'Books Count: ' + this.books.length  
  }  
}
```

- **computed atribúty sú cacheované**
  - hodnota atribútu (**currentTabComponent**) je aktualizovaná, iba ak sa zmení niektorá z hodnôt jeho závislostí (atribút **books**)
- volanie metódy vždy vypočíta hodnotu daného atribútu (**books**)

# vlastnosť watch

- potrebujeme aktualizovať údaje na základe zmeny iných údajov,
- zmení sa **firstName**, potrebujeme aktualizovať **fullName**, podobne **lastName**

```
watch: {  
  firstName: function (val) {  
    this.fullName = val+' '+this.lastName  
  },  
  lastName: function (val) {  
    this.fullName = this.firstName+' '+val  
  }  
}
```

# watch vs. computed

- kód s `watch` je imperatívny a opakujúci sa
- porovnajme s `computed`

```
computed: {  
  fullName: function () {  
    return this.firstName+' '+this.lastName  
  }  
}
```

# Modifikátory udalostí

<!-- šírenie udalosti bude pozastavené -->

<a @click.**stop**="doThis"></a>

<!-- submit nereloadne stránku -->

<form @submit.**prevent**="onSubmit"></form>

<!-- reťazenie modifikátorov -->

<a @click.stop.**prevent**="doThat"></a>

# Modifikátory klávesov

```
<input v-on:keyup.enter="submit">
```

// skrátený zápis

```
<input @keyup.enter="submit">
```

.enter, .tab., .delete (delete + backspace),  
.esc, .space, .up, .down, .left, .right

// kód klávesu

```
<input v-on:keyup.13="submit">
```

- [možné definovať vlastné aliasy pre kódy](#)



# Modifikátory klávesov /2

`.ctrl, .alt., .shift`

## **.meta**

– windows 

– mac 

## **.exact**

// iba ak click+ctrl a žiadna iná klávesa

```
<button @click.ctrl.exact="onCtrlClick">A</button>
```

# Modifikátory myši

`.left`

`.right`

`.middle`

```
<div v-on:mousedown.left="onDivClick">  
</div>
```

# Vstup z formulára `v-model`

- pre vstupné polia formulára
  - obojsmerné previazanie
    - hodnota vstupného pola s dátovou premennou

```
// premenná message obsahuje aktuálnu hodnotu  
// z textového pola
```

```
<input v-model="message" placeholder="edit me">
```

```
<p>Message is: {{ message }}</p>
```

# Vstup z formulára v-model /2

```
<textarea v-model="message"></textarea>
```

```
<input type="checkbox" v-model="checked">
```

```
<input type="radio" value="One" v-model="picked">
```

```
<select v-model="selected" multiple>
```

```
  <option>A</option>
```

```
  <option>B</option>
```

```
  <option>C</option>
```

```
</select>
```

# Vstup z formulára – modifikátory

- predvolene je hodnota vstupného pola s dátovou premennou synchronizovaná po každej udalosti na vstupné pole

`.lazy` – napr. pri textovom poli, synchronizuje až po zmene hodnoty a strate focusu

```
<input v-model.lazy="msg">
```

`.number`

// automaticky typecast,

// nezabudajme, ze v HTML su to aj tak retazce

```
<input v-model.number="age" type="text">
```

`.trim`

```
<input v-model.trim="msg">
```

# Skrátený zápis

`<a v-bind:href="url"> ... </a>`

`<a :href="url"> ... </a>`

`<a v-on:click="doSomething"> ... </a>`

`<a @click="doSomething"> ... </a>`

# State management /Store

```
import { reactive } from 'vue'

export const store = reactive({
  count: 0,
  increment() {
    this.count++
  }
})
```

# State management /ComponentA

```
<script>
import { store } from './store.js'
export default {
  data() {
    return { store }
  }
}
</script>
<template>
  <div>
    <button @click="store.increment()">
      From A: {{ store.count }}
    </button>
  </div>
</template>
```



# State management /ComponentB

```
<script>
import { store } from './store.js'
export default {
  data() {
    return { store }
  }
}
</script>
<template>
  <div>
    <button @click="store.increment()">
      From B: {{ store.count }}
    </button>
  </div>
</template>
```

# Vuex – Vue Store

- na cvičeniach si ukážeme Vuex
- <https://vuex.vuejs.org/>
- Pinia:
- <https://pinia.vuejs.org/>

# Quasar

- `npm install -g @quasar/cli`
- `quasar create [folder]`
- `quasar dev`