Vue.js

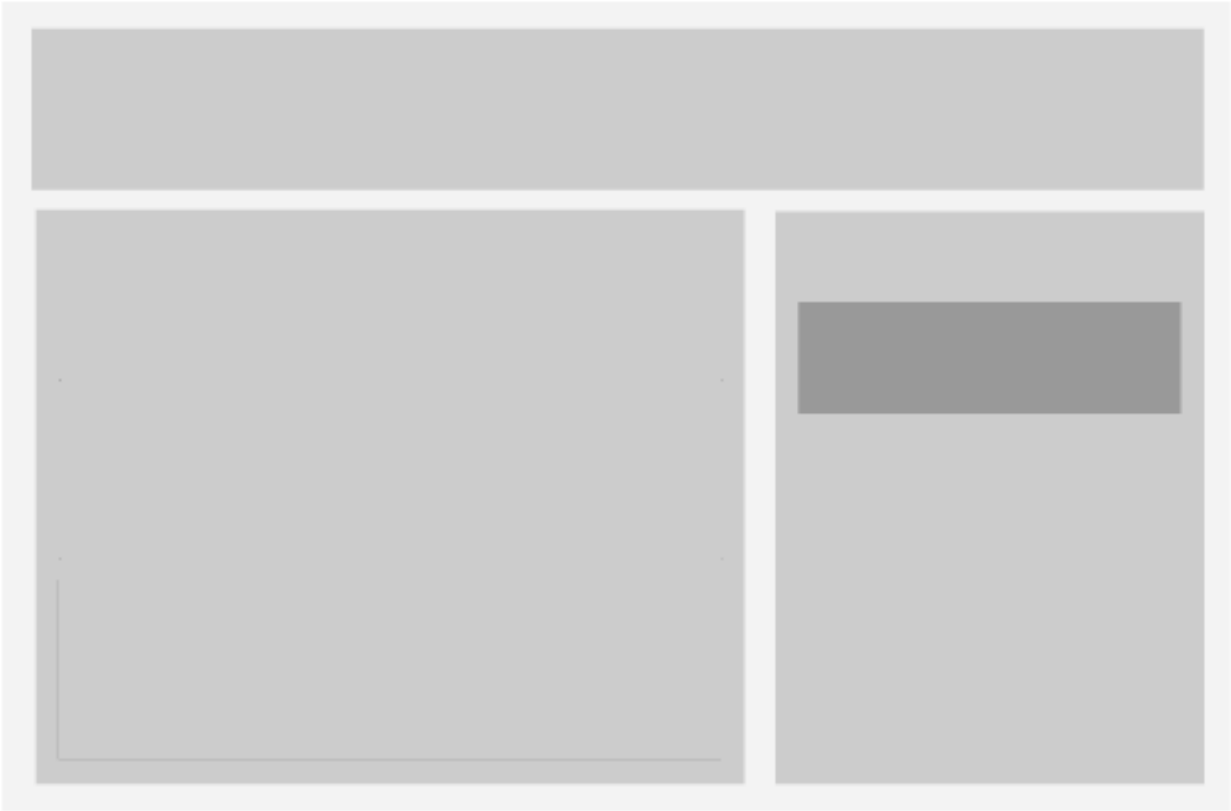
kurz Webové technológie

Eduard Kuric

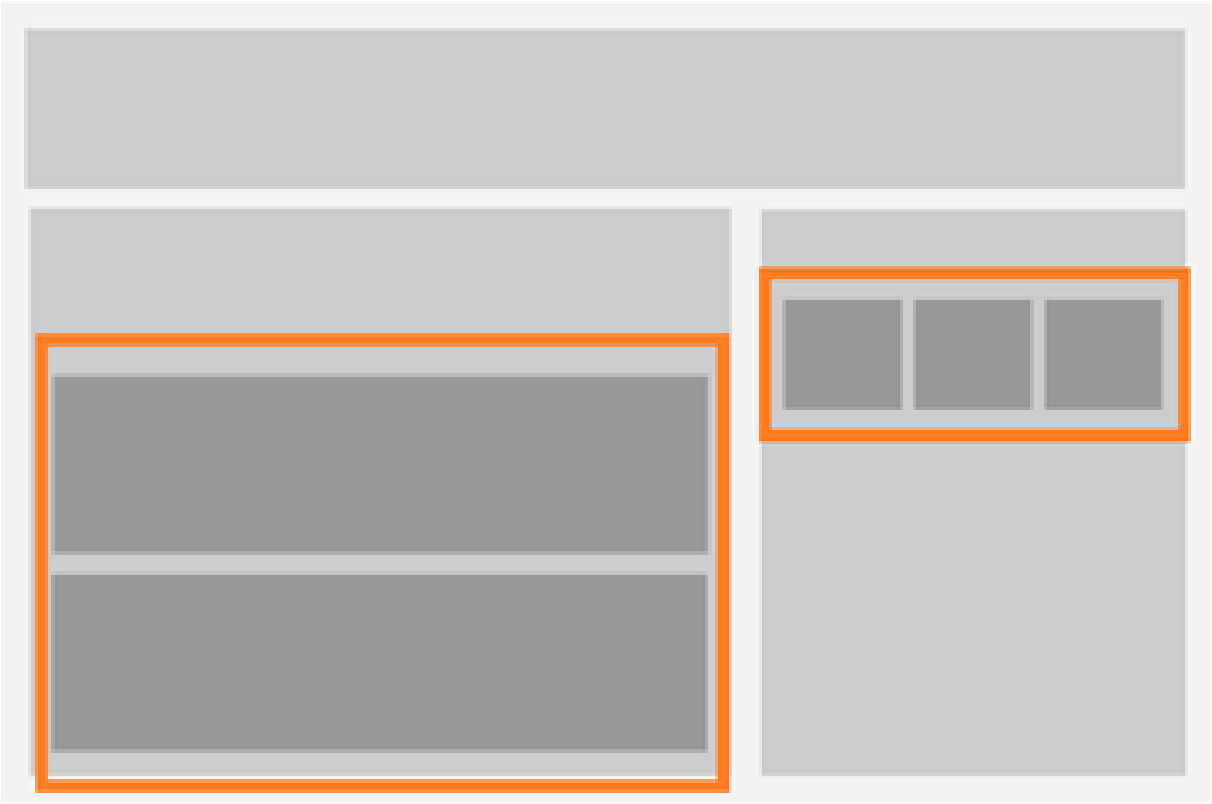
STU FIIT

27. 11. 2018

Komponentom môže byť jeden fragment

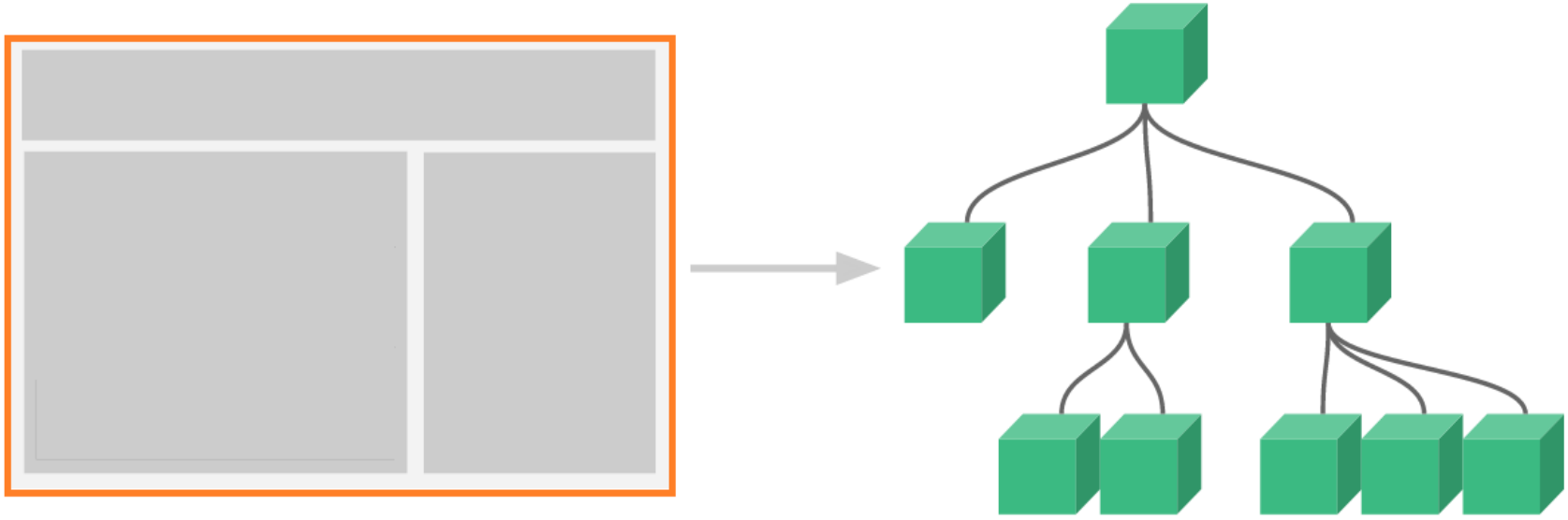


Komponent môže byť zložený z komponentov



Celá stránka je komponent zložený z komponentov

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



Single file components

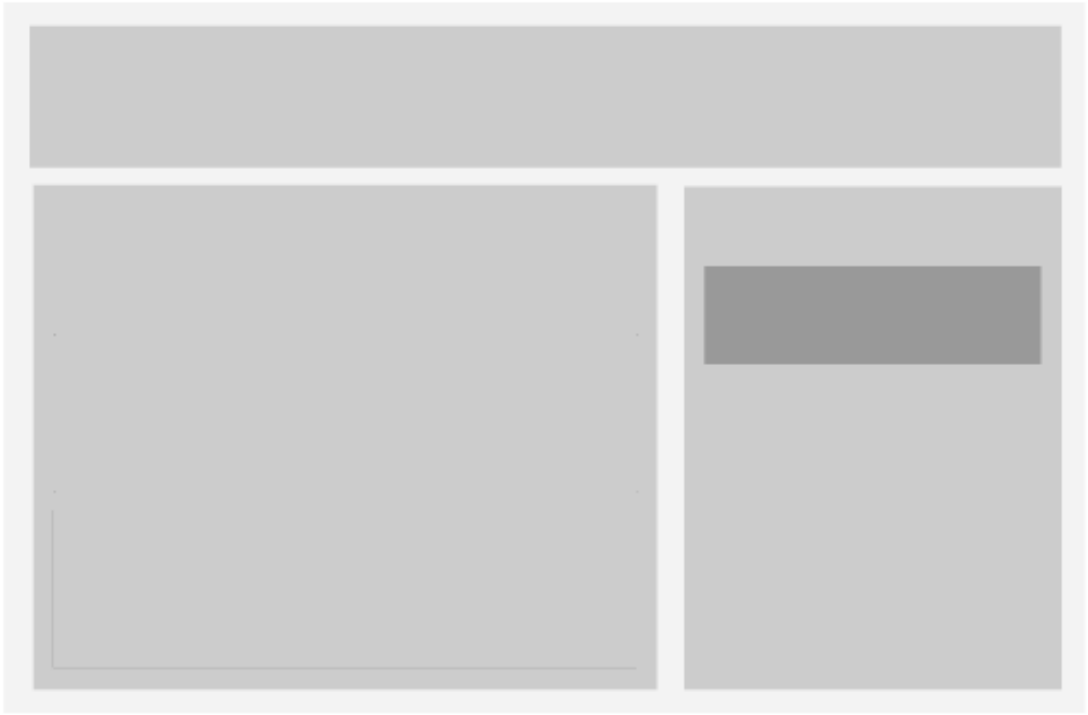
In modern UI development, we have found that instead of dividing the codebase into three huge layers (template, logic, style), it makes much more sense to divide them into loosely-coupled components and compose them.

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 – príklad

- chceme na stránke interaktívny (dynamický) komponent
  - príklad, jednoduché počítadlo počtu kliknutí na tlačidlo



Počítadlo kliknutí na tlačidlo

- vytvoríme koreňový komponent, ktorý zaobaľuje 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
- v atribúte `el` určíme, ktorý element je koreňovým elementom komponentu

```
<script>
var app = new Vue({
    el: '#counter-widget',
});
</script>
```

Počítadlo kliknutí na tlačidlo /2

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

```
var app = new Vue({
    el: '#counter-app',
    data: {
      heading: 'Počítadlo kliknutí',
      message: 'Komponent zobrazuje počet
                kliknutí na tlačidlo'
    }
});
```

Počítadlo kliknutí na tlačidlo /3

- **dodefinujeme šablónu komponentu**

```
var app = new Vue({
    el: '#counter-widge
    data: {
      heading: 'Počítadlo kliknutí',
      message: 'Komponent zobrazuje počet
                kliknutí na tlačidlo'
    }
});
```

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

Počítadlo kliknutí na tlačidlo /4

- **pridáme tlačidlo**

```
var app = new V
    el: '#counte
    data: {
      heading: '
      message: 'Komponent zobrazuje pocet
               kliknutí na tlačidlo'
    }
});
```

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

Počítadlo kliknutí na tlačidlo /5

- **zadefinujeme metódu komponentu**

```
var app = new Vue({
    el: '#counter-widget
    data: {
      heading: 'Počítadl
      message: 'Komponen
                na tlači
      clickCounter: 0
    },
    methods: {
      buttonClick: function() {
         this.clickCounter++;
      }
    }
});
```

```
<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!**
- zavolaním metódy `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">
        {{message}}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue"></script>

    <script>
      var vm = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue!'
        },
        methods: {}
      });
    </script>
</body>
```

vm - viewmodel

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

model–view–viewmodel

```
<body>
    <div id="app">
        {{message}}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue"></script>

    <script>
      var vm = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue!'
        },
        methods: {}
      });
    </script>
</body>
```

data - model

- dátový objekt

model–**view**–viewmodel

```
<body>
    <div id="app">
        {{message}}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue"></script>

    <script>
      var vm = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue!'
        },
        methods: {}
      });
    </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

Jeden koreňový element

- **každý komponent musí mať koreňový element**
- ak chceme mať takúto šablónu príspevku:

```
<h3>{{ post.title }}</h3>
<div v-html="post.content"></div>
```

- musíme elementy zaobaliť

```
<div class="blog-post">
  <h3>{{ post.title }}</h3>
  <div v-html="post.content"></div>
</div>
```

Globálny komponent

```
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">
               Kliknutí {{ count }}.
             </button>'
})
```

Použitie komponentu

- element (otváracia/ukončovacia značka) je názov komponentu

```
<div id="vue-app">
    <!- toto Vue transformuje na html sablonu -->
    <button-counter></button-counter>
</div>

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

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

- ak by neboli dáta funkciou

```
data: {
   count: 0
}
```

po kliknutí na tlačidlo by sa počítadlo inkrementovalo na všetkých “klonoch” predmetného komponentu

Lokálny komponent

```
// normálny JS objekt
var ComponentA = { /* ... */ }

new Vue({
  el: '#app'
  components: {
    'component-a': ComponentA
  }
})
```

Lokálny komponent /2

- **lokálne definované komponenty nie sú dostupné v subkomponentoch**
- ak chceme, aby bol komponent A dostupný v komponente B

```
var ComponentA = { /* ... */ }

var ComponentB = {
  components: {
    'component-a': ComponentA
  },
}
```

Posunutie dát vnoreným komponentom - `props`

```
new Vue({
  el: '#blog-posts',
  data: {
    posts: [
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Blogging with Vue' },
      { id: 3, title: 'Why Vue is so fun' },
    ]
  }
})
```

Posunutie dát vnoreným komponentom – `props` /2

```
// cez props zaregistrujeme custom-atribúty
// komponentu

Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3'
})

// cez v-bind odovzdáme komponentu predmetný atribút
<blog-post
  v-for="post in posts"
  v-bind:key="post.id"
  v-bind:title="post.title">
</blog-post>
```

Posunutie dát vnoreným komponentom – `props` /3

```
// cez props zaregistrujeme custom-atribúty
// komponentu
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})

// priamo cez atribút
<blog-post title="Príspevok 1"></blog-post>
<blog-post title="Príspevok 2"></blog-post>
<blog-post title="Príspevok 3"></blog-post>
```

Sloty

- posunutie obsahu komponentu cez slot

```
<alert-box>
   Nastala chyba #5456.
</alert-box>

Vue.component('alert-box', {
  template: '
    <div class="demo-alert-box">
      <strong>Chyba!</strong>
      <slot></slot>
    </div>
  '
})
```

Sloty /2

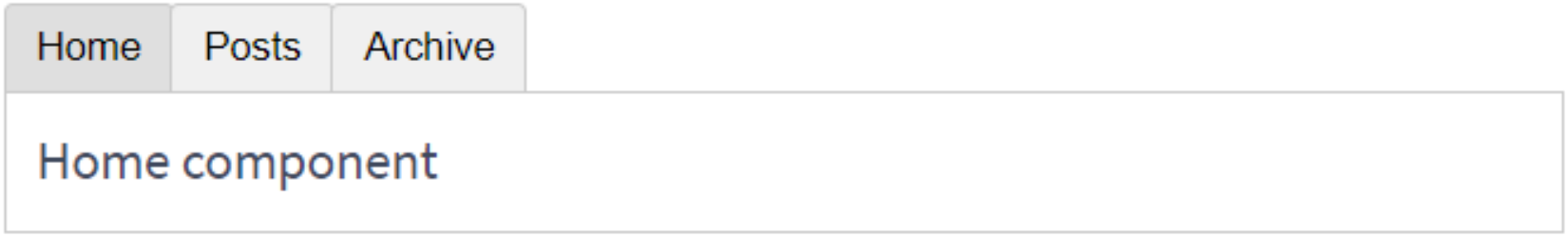
- posunutie obsahu komponentu

```
<alert-box>
    Nastala chyba #5456.
</al
Vue.
  te
      </div>
  '
})
```

```
<div class="demo-alert-box">
  <strong>Chyba!</strong>
  Nastala chyba #5456.
</div>
```

Dynamické komponenty

- chceme zobraziť v kontajneri komponent na základe výberu/podmienky
  - typicky karty



Dynamické komponenty /2

- vytvoríme čiastkové komponenty – jednotlivé karty

```
Vue.component('tab-home', {
      template: '<div>Home component</div>'
})

Vue.component('tab-posts', {
      template: '<div>Posts component</div>'
})

Vue.component('tab-archive', {
      template: '<div>Archive component</div>'
})
```

Dynamické komponenty /3

- vytvoríme komponent, ktorý bude obsahovať tlačidlá na prepnutie medzi kartami a obsah aktuálnej karty

```
<div id="dynamic-component">
  <button
    v-for="tab in tabs" v-bind:key="tab"
    v-on:click="currentTab = tab">
       {{ tab }}
  </button>

  // currentTabComponent - názov aktuálnej tabky
  // registrovaný komponent: tab-home, tab-posts, tab-archive
  <component v-bind:is="currentTabComponent">
  </component>
</div>
```

Dynamické komponenty /4

```
new Vue({
  el: '#dynamic-component',
  data: {
    currentTab: 'Home',
    tabs: ['Home', 'Posts', 'Archive']
  },
  computed: {
    currentTabComponent: function () {
      return 'tab-'+this.currentTab.toLowerCase()
    }
  }
})
```

vlastnosť `computed`

```
new Vue({
  el: '#dynamic-component',
  data: {
    currentTab: 'Home',
    tabs: ['Home', 'Posts', 'Archive']
  },
  computed: {
    currentTabComponent: function () {
      return 'tab-'+this.currentTab.toLowerCase()
    }
  }
})
```

vlastnosť `computed` /2

```
computed: {
    currentTabComponent: function () {
      return 'tab-'+this.currentTab.toLowerCase()
    }
}
```

- keď sa zmení atribút **`currentTab`**, je automaticky po ňom zmenený/prepočítaný aj atribút **`currentTabComponent`**

`computed` vs. metóda

```
methods: {
    currentTabComponent: function () {
      return 'tab-'+this.currentTab.toLowerCase()
    }
}
```

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

vlastnosť `watch`

- potrebujeme zmeniť údaje na základe zmeny iných údajov

```
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 (AngularJS štýl)

- porovnajme s `computed`

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

- sú ale situácie, kedy `watch` je nutný

Asynchrónne komponenty

- veľké aplikácie potrebujeme rozdeliť na menšie kúsky
  - vytvorenie komponentu pomocou tzv. **factory function**
  - **vykoná** sa, až **keď je požiadavka na vykreslenie komponentu** (výsledok sa nacachuje pre ďalšie prekreslenia)

Asynchrónne komponenty /2

```
Vue.component('async-component',
                    function (resolve, reject) {
    setTimeout(function () {
      resolve({
          template: '<div>I am async!</div>'
        })
      }, 1000)
})
```

- **`resolve`** callback zavoláme, keď máme k dispozícii definíciu komponentu (napr. načítaný zo servera)
  - `setTimeout` na ilustráciu
- **`reject`** callback vytvorenie komponentu zlyhalo

Async komponenty + Webpack

- asynchrónnymi komponentami môžeme indikovať webpacku, aby rozdelil kód na balíčky, ktoré budú načítané cez XHR požiadavky

```
Vue.component('async-webpack', function
(resolve) {
   require(['./my-async-component'], resolve)
})
```

Vue.js – životný cyklus - hooks

1. inicializácia (vytváranie) komponentu
2. zostavenie – vloženie komponentu do DOMu a vykreslenie
3. aktualizáca komponentu – zmena a prekreslenie
4. zrušenie/zničenie

`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 virtuálny DOM)
  - **fáza vhodná na získavanie dát (fetch)**

`beforeMount`, `mounted`

`beforeMount() {}`

- tesne predtým, ako sa začne komponent vykreslovať

`mounted() {}`

- plný pristúp k kreaktívnemu komponentu (šablóny, vykreslený DOM cez `this.$el`)
  - **fáza vhodná na modifikáciu DOMu** (knižnice 3tích strán)

`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**

beforeDestroy, destroyed

`beforeDestroy() {}`

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

`destroyed() {}`

- komponent bol zničený
  - **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 neprejavi
<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: This should be red.

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>
```

- 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>
```

Triedy s komponentami

```
Vue.component('my-component', {
  template: '<p class="foo bar">Hi</p>'
})

// <p class="foo bar baz boo">Hi</p>
<my-component class="baz boo"></my-component>

// <p class="foo bar active">Hi</p>
<my-component v-bind:class="{ active: isActive }">
</my-component>
```

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">
  <!-- content -->
</div>
```

- predvolené správanie - keď sa zmení poradie položiek, namiesto toho, aby sa elementy preusporiadali tak, aby zodpovedali novému poradiu, vue nahradí každý prvok na danom indexe
- vue ale môžeme poskytnúť jednoznačnú identitu položiek – kľúč, pomocou ktorého dokáže efektívne znovupoužiť (preusporiadať) existujúce položky
- Odporúčam príklad

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>

new Vue({
  el: '#example-3',
  methods: {
    say: function (message) {
      alert(message)
    }
  }
})
```

Modifikátory udalostí

```
<a v-on:click.stop="doThis"></a>
<form v-on:submit.prevent="onSubmit"></form>

<!- režim zachytenie/capture -->
<div v-on:click.capture="doThis">...</div>

<!-- event.target je samotny element -->
<div v-on:click.self="doThat">...</div>
```

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="number">
```

```
.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>
```

Filtre

- používajú sa **na formátovanie textu**
  - mustache {{ }}
  - v-bind

```
<!-- mustache -->
{{ message | capitalize }}

<!-- v-bind -->
<div v-bind:id="rawId | formatId"></div>
```

Lokálny filter

- súčasťou komponentu

```
filters: {
  capitalize: function (value) {
    if (!value) return ''
    value = value.toString()
    return value.charAt(0).toUpperCase()
          + value.slice(1)
  }
}
```

Globálny filter

```
Vue.filter('capitalize', function (value)
{
  if (!value) return ''
  value = value.toString()
  return value.charAt(0).toUpperCase()
          + value.slice(1)
})
```

Reťazenie filtrov, argumenty

```
{{ message | filterA | filterB }}
```

```
{{ message | filterA('arg1', arg2) }}
```

Mixins

- znovupoužitie funkcionality pre komponenty
- `mixin object` môže obsahovať akúkoľvek vlastnosť komponentu
  - „primiešajú“ sa do daného komponentu

```
var mixin = {
  data: function () {
    return {
       foo: 'abc'
    }
  }
}
```

Mixins /2

```
new Vue({
  mixins: [mixin],
  data: function () {
    return {
      bar: 'def'
    }
  },
  created: function () {
    // { foo: "abc", bar: "def" }
    console.log(this.$data)
  }
})
```

Mixins - konflikt

- v prípade konfliktu, dáta komponentu majú vyššiu prioritu

```
new Vue({
  mixins: [mixin],
  data: function () {
    return {
      foo: 'def'
    }
  },
  created: function () {
    // { foo: "def"}
    console.log(this.$data)
  }
})
```

State management

```
const zdielanyZdrojDat = {}

const vmA = new Vue({
  data: zdielanyZdrojDat
})

const vmB = new Vue({
  data: zdielanyZdrojDat
})
```

State management /2

- vnorené komponenty by mohli prístúpiť k týmto dátam cez `this.$root.$data`
- hocaká časť týchto dát by mohla byť pozmenená kedykoľvek, kýmkoľvek
  - nočná mora na debugovanie

- **riešenie - Store**

Store

- zdieľané dáta presunieme do stóru
- do stóru pridáme aj metódy, ktoré mutujú/menia dáta – stav
- centralizovaný stavový manažment
  - zvyčajne jeden store na aplikáciu

Store – vzor (pattern)

```
var store = {
  debug: true,
  state: {
    message: 'Hello!'
  },
  setMessageAction (newValue) {
    this.state.message = newValue
  },
  clearMessageAction () {
    if (this.debug) console.log('clearMessageAction triggered')
    this.state.message = ''
  }
}
```

Store – vzor (pattern) /2

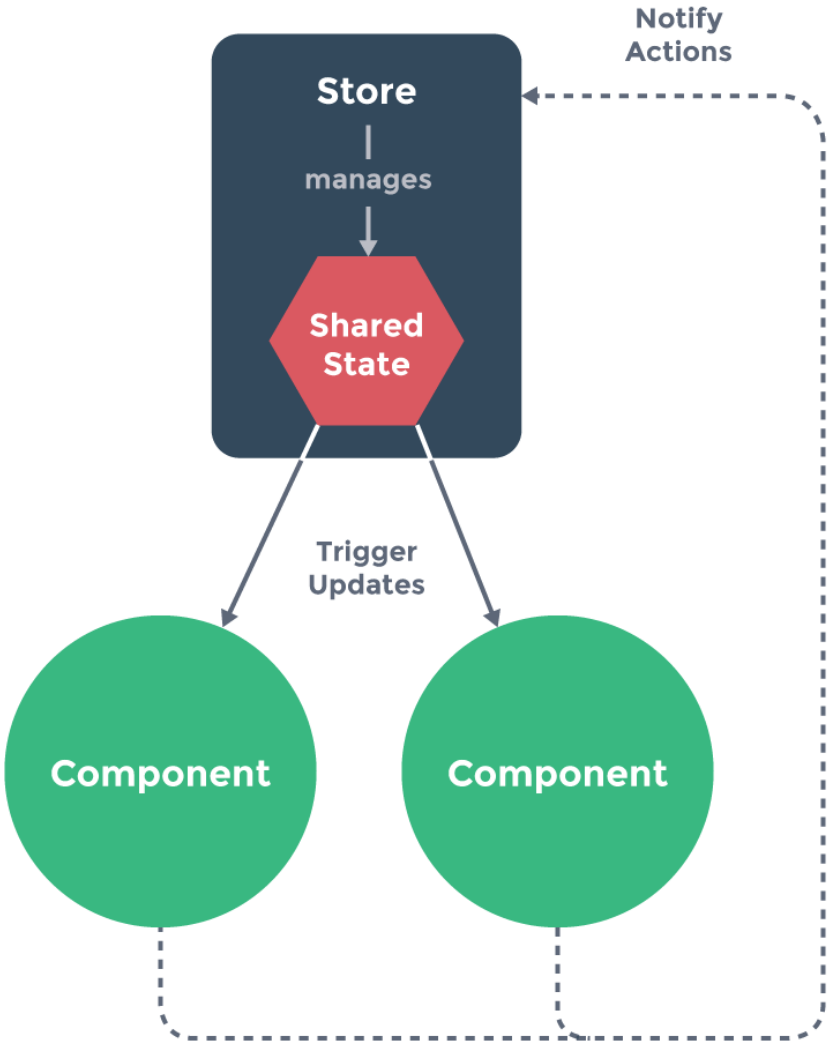
```
var vmA = new Vue({
  data: {
    privateState: {},
    sharedState: store.state
  }
})

var vmB = new Vue({
  data: {
    privateState: {},
    sharedState: store.state
  }
})
```

Store – vzor (pattern) /4

- táto **centralizácia umožňuje ľahšie pochopiť**, aký **typ mutácii** môže nastať
- store **udržiava zdieľané dáta v trasparentnom a predvídateľnom stave** (aj keď sú zdieľané viacerými komponentami)
- komponenty, ktoré „sledujú“ stav budú pri zmene **automaticky aktualizované** – reaktivita

- **eliminuje duplicitu kódu**, alebo extrakciu do helpera
  - napr. ak komponenty realizujú rovnaký výpočet nad zdieľanými dátami

Store – vzor (pattern) /5



Vuex – store

- zdieľané údaje sú uložené v `state`

```
const store = new Vuex.Store({
  state: {
    todos:
     [{ id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }]
  }
})
```

Vuex – store - **getters**

- getters vracajú stav v `store`, sú to v princípe `computed` vlastnosti
  - sú cacheované; sú prepočítané, keď sa zmení niektorá zo závislostí

```
const store = new Vuex.Store({
  state: {
    todos:
     [{ id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }]
  },
  getters: {
    doneTodos: state => {
      return state.todos.filter(todo => todo.done)
    }
  }
})
store.getters.doneTodos
```

Vuex – muttators

- obslužné metódy, ktoré menia (mutujú) stav (`state`) v store
  - zmeny v `state` je možné robiť iba cez mutácie, mutácia je analógia vyvolania udalosti (event); zmenu vykonáme odovzdáním mutácie
- každá obslužná metóda má parameter `state`
- k metódam nie je možné pristupovať priamo, ale cez `commit`

```
const store = new Vuex.Store({
  state: {
    count: 1
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
store.commit('increment')
```

Vuex – muttators /2

- odovzdanie argumentov v mutácii obslužnej metóde

```
mutations: {
   increment (state, n) {
     state.count += n
   }
}
store.commit('increment', 10)
```

Mutácie - synchrónne

- mutácie musia byť synchrónne

```
mutations: {
   someMutation (state) {
     api.callAsyncMethod(() => {
       state.count++
     })
   }
}
```

- callback nie je zavolaný/vykonaný v čase, keď je mutácia odovzdaná (commited); nevieme, kedy bude reálne vykonaný
  - napr. ak by sme chceli zalogovať mutáciu – stav pred a po, nebolo by to možné

Vuex – akcie

- nemutujú stav, ale odovzdávajú mutácie
- môžu obsahovať ľubovoľné asynchrónne operácie
- akcia má parameter `{ commit }`

```
const store = new Vuex.Store({
mutations: {
    increment (state) {
      state.count++
    }
  },
  actions: {
    increment ({ commit }) {
        commit('increment')
    }
  }
})
store.dispatch('increment')
```

Vuex – akcie async /pr.1

```
actions: {
  incrementAsync ({ commit }) {
    setTimeout(() => {
      commit('increment')
    }, 1000)
  }
}

store.dispatch('incrementAsync')
```

Vuex – akcie async / pr.2

```
actions: {
  fetchData ({ commit }) {
    this.$http({
      url: 'some-endpoint',
      method: 'GET'
    }).then(function (response) {
      commit('updateSavedData', response.data)
    }, function () {
      console.log('error')
    })
  }
}
```

Dispečing akcií v komponentoch

```
methods: {
    ...mapActions([
      // this.increment() ->
      // this.$store.dispatch('increment')
      'increment',
    ]),
    ...mapActions({
      // this.add()
      // -> this.$store.dispatch('increment')
      add: 'increment'
    })
  }
```

Vuex – moduly

```
const store = new Vuex.Store({
  modules: {
    a: moduleA,
    b: moduleB
  }
})

store.state.a // -> state v moduleA
store.state.b // -> state v moduleB
```

Vuex – moduly /2

```
const moduleA = {
  state: { ... },
  mutations: { ... },
  actions: { ... },
  getters: { ... }
}

const moduleB = {
  state: { ... },
  mutations: { ... },
  actions: { ... }
}
```

Vuex – moduly /3

```
const moduleA = {
  actions: {
    incrementIfOddOnRootSum ({ state, commit,
                               rootState }) {
      if ((state.count + rootState.count) % 2 === 1) {
        commit('increment')
      }
    }
  }
}
// state.count == rootState.a.count
```

Quasar

- Starter kit
- inštalácia
  - `npm install -g quasar-cli`
  - `quasar init <folder_name>`
- smerovanie - routing
  - `/src/router/routes.js`
- pridanie layouts a pages
  - `quasar new layout main`
  - `quasar new  page user`
- build
  - `quasar dev`
  - `quasar build`

Zdroje

- Fullstack Vue
- Vue.js
- Vuex

- Aligator.io – Vue.js