



Programowanie aplikacji internetowych  
2019/20

Instrukcja laboratoryjna cz.5

Framework Vue.js



Prowadzący: Tomasz Goluch

Wersja: 2.0

## I. Wprowadzenie<sup>1</sup>

*Cel: Przekazanie podstawowych informacji o zasobach laboratorium.*

Laboratorium odbywa się na maszynach fizycznych, z wykorzystaniem preferowanego edytora lokalnego (np. *Atom*, *Sublime text*, *Neovim*...) albo on-line (np. <https://jsfiddle.net>, <https://codepen.io/>, <https://codesandbox.io>).

## II. Instalacja

*Cel: Zapoznanie z dostępnymi sposobami instalacji Vue.js.*

Aby korzystać z framework'a wystarczy pobrać wybraną wersję skryptu z repozytorium github: <https://github.com/vuejs/vue/releases>. W folderze `dist` znajdują się między innymi wersja deweloperska `vue.js` (zalecana) i zminimalizowana `vue.min.js` (produkcyjna) skryptu. Następnie w pliku `html` należy umieścić odwołanie:

```
<script src="../../vue-2.x.x/dist/vue.js"></script>
```

Można umieścić znacznik `script` pobierający konkretną wersję biblioteki z serwera CDN:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.x.x/vue.js"></script>
<script src="https://unpkg.com/vue@2.x.x/dist/vue.js"></script>
<script src="https://cdn.jsdelivr.net/npm/vue@2.x.x/dist/vue.js"></script>
```

Można również pobrać jej najnowszą wersję:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

Najnowszą wersję biblioteki można również pobrać tutaj:

<https://vuejs.org/v2/guide/installation.html>

Jeśli chcemy korzystać z takich dobrodziejstw Vue.js jak komponenty, wtyczki, gotowe szablony aplikacji musimy zainstalować `vue-cli` do czego wymagane będzie środowisko wykonawcze `Node.js`<sup>2</sup> oraz menedżer pakietów `npm` (dostarczany z `Node.js`) albo `Yarn`. W wierszu poleceń bądź `PowerShell`'u uruchom następujące komendy:

# Instalacja `@vue/cli` oraz `@vue/cli-init`:

```
npm install -g @vue/cli
npm install -g @vue/cli-init
```

albo:

```
yarn global add @vue/cli
yarn global add @vue/cli-init
```

<sup>1</sup> Instrukcja przygotowana na podstawie laboratorium Hands-On lab: [Building your first App for Windows 8.1 and publishing it to the Windows Store](#).

<sup>2</sup> Strona projektu Node.js: <https://nodejs.org/en/>

Uwaga, w przypadku zainstalowanej wcześniejszej wersji vue/cli i vue/cli-init należy ją odinstalować poleceniem:

```
npm uninstall -g vue/cli
npm uninstall -g vue/cli-init
```

albo:

```
yarn global remove vue/cli
yarn global remove vue/cli-init
```

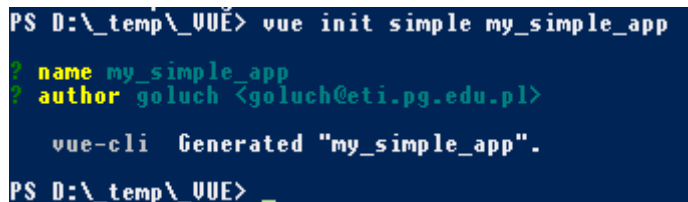
Inicjalizacja szablonu aplikacji:

```
vue init <typ projektu> <nazwa_projektu>
```

Dostępne typy projektów:

- simple – Najprostsza możliwa konfiguracja (plik HTML).
- browserify – W pełni funkcjonalny konfiguracja Browserify.
- browserify-simple – Prosta konfiguracja browserify do szybkiego prototypowania.
- webpack – W pełni funkcjonalna konfiguracja z Webpackiem.
- webpack-simple – Prosta konfiguracja webpack do szybkiego prototypowania.
- pwa – W pełni funkcjonalny szablon PWA (wymagany jest python z dodaną ścieżką dostępu w zmiennej środowiskowej PATH).

Inicjalizacja szablonu prostej aplikacji:



```
PS D:\_temp\_VUE> vue init simple my_simple_app
? name my_simple_app
? author goluch <goluch@eti.pg.edu.pl>
vue-cli Generated "my_simple_app".
PS D:\_temp\_VUE>
```

Po uruchomieniu pliku indeks.html widzimy naszą aplikację:



## Welcome to your Vue.js app!

- To learn more about Vue, visit [vuejs.org/guide](https://vuejs.org/guide)
- For live help with simple questions, check out [the Discord chat](#)
- For more complex questions, post to [the forum](#)

Inicjalizacja szablonu prostej konfiguracji webpack:

```
PS D:\_temp\_VUE> vue init webpack-simple my-webpack-project
? Project name my-webpack-project
? Project description PS D:\_temp\_VUE>
PS D:\_temp\_VUE> vue init webpack-simple my_webpack_app
? Project name my_webpack_app
? Project description A Vue.js project
? Author goluch <goluch@eti.pg.edu.pl>
? License MIT
? Use sass? No

vue-cli Generated "my_webpack_app".

To get started:

  cd my_webpack_app
  npm install
  npm run dev
```

Instalacja i uruchomienie nowego projektu (wymagane uprawnienie administratora):

`cd <nazwa_projektu>`

`npm|yarn install` (tylko jeśli nie wybraliśmy automatycznego uruchomienia tej komendy)

`npm|yarn run dev`

Aplikacja powinna być widoczna w przeglądarce pod adresem: <http://localhost:8080>:

Zatrzymanie projektu (powrót do powłoki):

`Ctrl + C, Y + Enter`

### III. Tworzenie aplikacji Vue.js

*Cel: Utworzenie prostej aplikacji Vue.js.*

Na początek utworzymy prostą aplikację. Nie trzeba jeszcze instalować `vue-cli`, wystarczy sama biblioteka. W pliku `lab1.html` tworzymy element DOM aplikacji Vue:

```
<!DOCTYPE html>
<script src="../vue-2.6.10/dist/vue.js"></script>

<html>
  <body>
    <div id="app" class="container">
      <h2>{{title}}</h2>
    </div>
  </body>
  <script src="lab1.js"></script>
</html>
```

W pliku `lab1.js` dodajmy kod tworzący dane dla aplikacji:

```
var data = { title: 'Moja aplikacja Vue.js'};
```

Tworzymy instancję Vue przekazując jej utworzony wcześniej element DOM oraz dane:

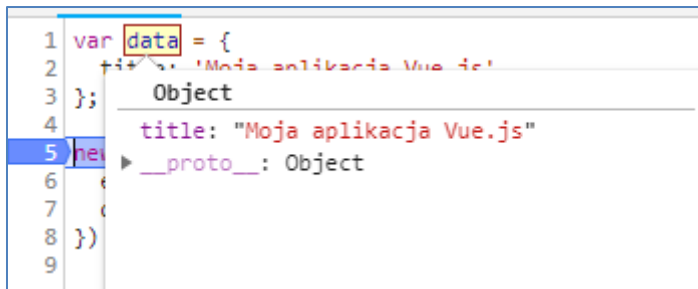
```
new Vue({
  el: '#app',
  data: data
})
```

Po uruchomieniu powinniśmy otrzymać następujący wynik:

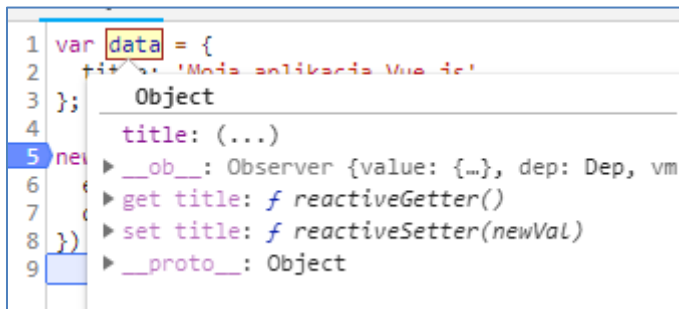
**Moja aplikacja Vue.js**

Aplikacja Vue realizuje wzorzec architektoniczny MVVM. Obiekt `data` to model danych, widok to nasz element DOM z atrybutem `id="app"` a widokmodel to aplikacja Vue realizująca wiązanie danych.

Debuggując nasz kod możemy zauważyć, że po inicjalizacji obiektu Vue obiekt `data` się zmienia. Zawartość `data` przed inicjalizacją aplikacji:



oraz po inicjalizacji:



Dodana metoda `reactiveSetter` pozwala między innymi wywołać powiadomienie o zmianie wartości skutkujące ponownym renderowaniem strony. Pozwala to na wiązanie danych z modelu umieszczonego w kodzie JS do strony HTML.

Vue pozwala na umieszczanie wyrażeń JS w podwójnych nawiasach klamrowych:

```
{{ "Sufit z 7,45 to: " + Math.ceil(7.45) }}
```

Wynik:

Sufit z 7,45 to: 8

Vue.js pozwala na łatwą implementację filtrów:

```
new Vue({
  el: '#app',
  data: data,
  filters: {
    capitalize: item => {
      return item.toUpperCase()
    }
  }
})
```

Wykorzystanie filtru:

```
{{ 'To jest jakiś napis'|capitalize }}
```

Wynik:

TO JEST JAKIŚ NAPIS

Filtry można implementować globalnie. Implementacja musi wystąpić przed utworzeniem instancji Vue:

```
Vue.filter('split', (value) => {
  return value.split(" ")
})

Vue.filter('join', function(value) {
  return value.join("_")
})
```

Symbol `|` pozwala wykorzystać filtry kaskadowo:

```
{{ 'To jest jakiś napis'|split|join }}
```

Wynik:

To\_jest\_jakiś\_napis

#### IV. Wybrane dyrektywy Vue<sup>3</sup>

*Cel: Zapoznanie z działaniem wybranych dyrektyw Vue.js.*

Dyrektywa `v-model` pozwala na dwukierunkowe wiązanie danych, współpracuje z elementami: `input`, `select`, `textarea`. Dodajmy element `input`<sup>4</sup>:

```
<input v-model="title" />
```

Efekt powinien być następujący:

Moja aplikacja ze zmienionym tytułem

akcja ze zmienionym tytułem

Dyrektywa `v-for`<sup>5</sup> pozwala na iteracyjne przetwarzanie, np. elementów tablicy. Dodajmy tablicę stringów do naszego modelu:

```
tab: ['kot', 'pies', 'mysz']
```

Aby je wyświetlić na stronie wystarczy dodać do zwykłej listy wspomnianą dyrektywę:

<sup>3</sup> <https://vuejs.org/v2/api/#Directives>

<sup>4</sup> W przypadku problemów z wiązaniem sprawdź czy dodany element jest wewnątrz elementu `<div id="app" class="container">?`

<sup>5</sup> <https://vuejs.org/v2/guide/list.html>

```
<ul>
  <li v-for="pet in tab">{{ pet }}</li>
</ul>
```

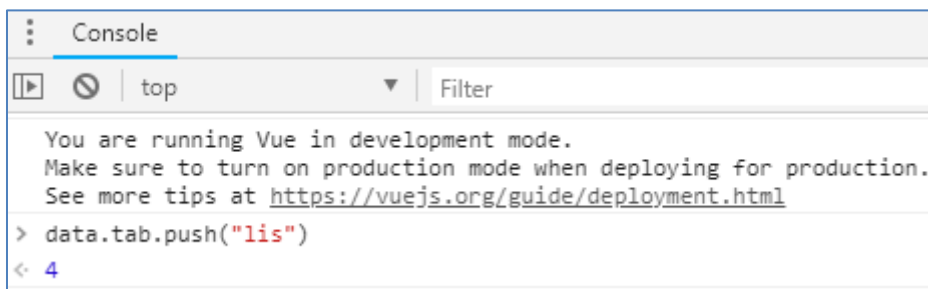
- kot
- pies
- mysz

Vue zapewnia dostęp do bieżącego indeksu tablicy (index), który można wykorzystywać w wyrażeniach:

```
<div v-for="(index, pet) in tab">{{ index }}. {{ pet }}</div>
```

0. kot
1. pies
2. mysz

Dodając w konsoli nowy element do tablicy natychmiast powinien zostać wyświetlony:



Bardzo prawdopodobne, że nasza aplikacja będzie pracować na bardziej złożonych danych, np. tabela obiektów zawierające przynajmniej dwie właściwości:

```
tab: [{ name: 'kot', checked: true },
      { name: 'pies', checked: false },
      { name: 'mysz', checked: false }]
```

```
<div v-for="pet in tab">
  <label>
    <input type="checkbox" v-model="pet.checked">
    {{pet.name}}
  </label>
</div>
```

- ☒ kot
- ☐ pies
- ☐ mysz



Dyrektywa `v-bind:<nazwa_atrybutu>` pozwala na powiązanie atrybutu elementu z wyrażeniem:

```
var data = {  
  ...  
  path: "https://vuejs.org/images/",  
  filename: "logo.png"  
};
```

```

```



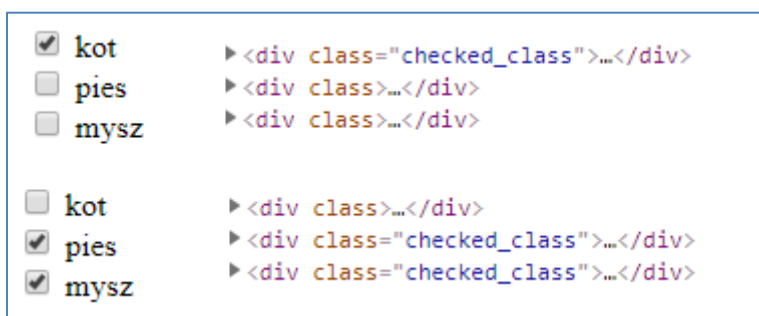
Posiada wersję skróconą, składającą się z pojedynczego znaku `:`.

```

```

Kiedy wiążemy atrybuty `class` lub `style`, możemy to uzależnić od warunku:

```
<div v-for="pet in tab" :class="{ 'unchecked_class': pet.checked }">  
  ...  
</div>
```



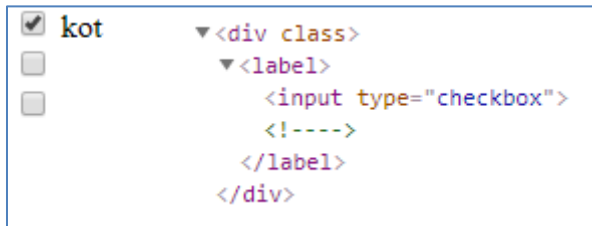
Pozwala to na przełączanie się między różnymi klasami/stylami:

```
<div v-for="pet in tab" :class="[pet.checked? 'checked_class': 'unchecked_class']">  
  ...  
</div>
```

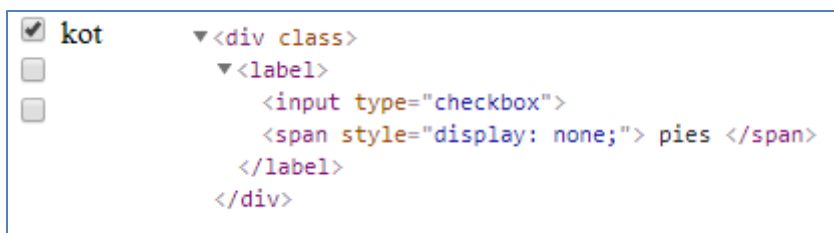
Dyrektywa `v-if` pozwala na warunkowe wyświetlanie elementu. Warunkiem jest dowolne wyrażenie i może zawierać właściwości modelu danych.

```
<div v-for="pet in tab" :class="{ 'checked_class': pet.checked }">  
  <label>  
    <input type="checkbox" v-model="pet.checked">  
    <span v-if="pet.checked"> {{pet.name}} </span>
```

```
</label>
</div>
```

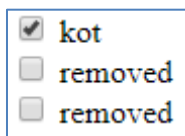


Dyrektywa **v-show** pozwala na to samo co dyrektywa **v-if** ale zawsze renderuje element w DOM. Jeśli warunek nie jest spełniony to ukryje element (właściwość CSS `display: none`) Zalecana podczas częstych zmian stanu.



Wraz z dyrektywą **v-if** można używać dyrektywy **v-else**, która wyświetli element w przypadku niespełnionego warunku:

```
<span v-if="pet.checked"> {{pet.name}} </span>
<span v-else> removed </span>
```



Dyrektywa **v-on** pozwala na obsługę zdarzeń:

```
var data = {
  ...
  counter: 0
};
```

```
new Vue({
  ...
  methods: {
    increment: function (event) {
      data.counter += 1
    }
  }
})
```

```
<button v-on:click="increment">Dodaj 1</button>
<p>Przycisk naciśnięto {{ counter }} razy.</p>
```

W przypadku krótszych metod możemy umieścić ich definicję bezpośrednio w dyrektywie:

```
<button @click="counter++">Dodaj 1</button>
```

Podobnie jak miało to miejsce w przypadku dyrektywy **v-bind**, **v-on** również posiada wersję skróconą, składającą się ze pojedynczego znaku **@**.

W prosty sposób możemy tworzyć niestandardowe dyrektywy. Dostępne jest 5 metod (bind, inserted, update, componentUpdated i unbind):

```
Vue.directive('my-directive', {
  bind: function (el, binding, vnode) {
    // Operacje przygotowawcze
    // np. dołączanie detektorów zdarzeń.
    // Wymaga jednokrotnego uruchomienia.
  },
  inserted: function () {
  },
  update: function (newValue, oldValue) {
    // Wykonanie operacji na podstawie zaktualizowanej wartości.
    // Zostanie również wywołana dla wartości początkowej.
  },
  componentUpdated: function () {
  },
  unbind: function () {
    // Operacje porządkowe
    // np. usunięcie detektorów zdarzeń dodanych w bind().
  }
})
```

Jeżeli interesuje nas jedynie implementacja metody **update** to możemy skorzystać z uproszczonej wersji metody:

```
Vue.directive('ceil', function (value) {
  this.el.innerHTML = Math.ceil(value)
})
```

Wykorzystanie i wynik:

```
Sufit z 7,45 to : <span v-ceil='7.45'></span>
```

Sufit z 7,45 to : 8

## V. Komponenty Vue

*Cel: Zapoznanie z procesem tworzenia komponentów Vue.js.*

Komponenty są ważnym mechanizmem pozwalający na podział funkcjonalności na mniejsze części i na łatwe ich ponowne użycie. Dzięki pośredniej fazie kompilacji możemy użyć

preprocesorów takich jak Pug, Babel czy Stylus aby tworzyć czytelniejsze i bogato wyposażone komponenty.

Utworzenie komponentu:

```
var MyComponent = Vue.extend({
  template: '<h2>To jest komponent</h2>'
});
```

Podczas rejestracji komponentu jako pierwszy parametr przekazujemy nazwę nowego elementu HTML reprezentującego nasz komponent. Najlepiej z wykorzystaniem notacji Kebab case, ponieważ HTML jest językiem case insensitive:

```
Vue.component('my-component', MyComponent);
```

Użycie komponentu (html):

```
<div id='app'>
  <my-component></my-component>
</div>
```

Użycie komponentu (JS):

```
new Vue({
  el: '#app'
})
```

Wynik:

**To jest komponent**

Pierwsze dwa kroki (utworzenie i rejestracja) można wykonać przy pomocy jednej instrukcji:

```
Vue.component('my-component', {
  template: '<h2>To jest komponent</h2>'
});
```

Umieszczanie szablonu w postaci łańcucha znaków jest złą praktyką. Szablony powinny być definiowane w kodzie HTML i służyć do tego element **template**:

```
<template id="component_id">
  <h2>To jest komponent</h2>
</template>
```

W komponencie umieszczamy odwołanie do szablonu.

```
Vue.component('my-component', {
  template: '#component_id'
});
```

Komponent może posiadać własne dane, jednak należy pamiętać, że właściwości `data` i `el` muszą być zaimplementowane w postaci metod:

```
Vue.component('my-component', {
  template: '#component_id',
  data: function () {
    return {
      msg: 'To jest komponent'
    }
  }
});
```

Teraz możemy skorzystać w szablonie z wiązania:

```
<template id="component_id">
  <h2>{{msg}}</h2>
</template>
```

Wszystkie komponenty posiadają dostęp do globalnego zasięgu aplikacji, dodajmy zatem właściwość `elem`:

```
new Vue({
  el: '#app',
  data: {
    elem: 'To jest komponent'
  }
});
```

Należy jawnie definiować w komponencie – za pomocą atrybutu `props` – które właściwości modelu danych komponentu nadrzędnego będą widoczne.

```
Vue.component('my-component', {
  template: '#component_id',
  props: ['msg']
});
```

Do wiązania właściwości modelu danych z instancją komponentu służy dyrektywa **`v-bind`**:

```
<my-component :msg="elem"></my-component>
```

Nic nie stoi na przeszkodzie aby komponent składał się z wielu różnych komponentów. Dodajmy szablon zawierający kontrolkę **`input`**:

```
<template id="inp_id">
  <input v-bind:value="msgIn" />
</template>
```

Implementacja i rejestracja komponentu pozwalającego na wprowadzanie danych:

```
Vue.component('inp-component', {
  template: '#inp_id'
});
```

Szablon multikomponentu zawierający dwa powyższe. Dane komponentu `inp-component` są powiązane z komponentem `my-component` właściwością `elem`:

```
<template id='multi_comp_id'>
  <div>
    <inp-component v-model="elem"></inp-component>
    <my-component :msg="elem"></my-component>
  </div>
</template>
```

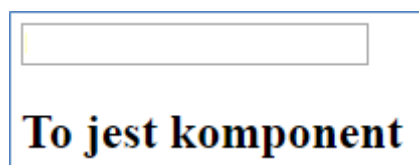
Teraz on będzie on zawierał właściwość `elem`:

```
Vue.component('multi-component', {
  template: '#multi_comp_id',
  data: function () {
    return {
      elem: 'To jest komponent'
    }
  },
});
```

Użycie multikomponentu w kodzie jest bardzo proste i czytelne:

```
<div id='app'>
  <multi-component></multi-component>
</div>
```

Niestety dane z komponentu nie przepływają pomiędzy komponentami tak ja byśmy tego oczekiwali:



Dzieje się tak ponieważ domyślnie dane propagowane są tylko w kierunku rodzic → potomek. Aby móc przesłać dane w drugą stronę komponent `inp-component` musi emitować zdarzenie `input`:

```
Vue.component('inp-component', {
  template: '#inp_id',
  methods: {
    onInput: function (event) {
```

```

        this.$emit('input', event.target.value)
    }
}
});

```

W szablonie komponentu `inp-component` wiążemy metodę `onInput` ze zdarzeniem `input`:

```

<template id="inp_id">
  <input :value="elem" @input="onInput" />
</template>

```

Teraz zachowanie się komponentów jest takie jak przewidywaliśmy:



Jeśli chcielibyśmy aby tekst kontrolki `input` po uruchomieniu zawierał napis zmiennej `elem`, musimy zadeklarować w komponencie `inp-component`, że właściwość `msg_in` będzie w nim widoczna:

```

<template id="inp_id">
  <input :value="msg_in" @input="onInput" />
</template>

```

```

Vue.component('inp-component', {
  template: '#inp_id',
  props: ['msg_in'],
  methods: {
    onInput: function (event) {
      this.$emit('input', event.target.value)
    }
  }
});

```

I powiązać ją za pomocą `elem`:

```

<template id='multi_comp_id'>
  <div>
    <inp-component :msg_in="elem" v-model="elem"></inp-component>
    <my-component :msg="elem"></my-component>
  </div>
</template>

```

Nowy komponent

## Nowy komponent

## VI. Komponenty jedno plikowe Vue

*Cel: Zapoznanie z procesem tworzenia komponentów jedno plikowych Vue.js.*

W bardziej złożonych projektach preferowane jest zamykanie kodu HTML, JS i styli powiązanych z jednym komponentem w jednym pliku, tzw. komponent jedno plikowy. Rozszerzenie takiego pliku, jak łatwo się domyślić to .vue. Przykład prostego, kompletnego komponentu App.vue:

```
<template>
  <h2>{{ msg }}</h2>
</template>

<script>
export default {
  data () {
    return {
      msg: 'To jest komponent'
    }
  }
}
</script>

<style scoped>
</style>
```

Domyślnie styl komponentu widoczny jest globalnie, do zawężenia jego zasięgu na komponent służy atrybut **scoped**. Działanie komponentu można sprawdzić przy używając edytora online: <https://codesandbox.io/s/8x2m79lqp9>.

Jeśli chcemy lokalnie uruchomić komponent musimy zainstalować odpowiedni szablon np.:

```
vue init webpack-simple <nazwa_projektu>
```

Import komponentu wewnątrz komponentu App.vue:

```
<template>
  <div id="app">
    <single-page-comp></single-page-comp>
    <other-name></other-name>
  </div>
</template>

<script>
```



```
import SinglePageComp from './SinglePageComp.vue'
export default {
  name: 'app',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  },
  components: {
    SinglePageComp,
    'other-name': SinglePageComp
  }
}
</script>
```

W kolejnym kroku musimy zainstalować i uruchomić serwer:

```
cd <nazwa_projektu>
npm|yarn run dev
```

Powinien wyświetlić się dobrze już nam znany widok:

**To jest komponent**

## VII. Vue Router<sup>6</sup>

Vue Router pozwala na łatwe pisanie aplikacji SPA (Single-page Application). Instalacja to dodanie kolejnej biblioteki dostępnej np.:

```
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
```

Następnie wystarczy zmapować nasze komponenty na **ścieżki** i poinformować Vue Router, gdzie je renderować:

```
<div id="app">
  <h1>Hello App!</h1>
  <p>
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>
  </p>
  <router-view></router-view>
</div>
```

```
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }
```

<sup>6</sup> <https://router.vuejs.org/guide/>

```
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

const router = new VueRouter({
  routes // short for `routes: routes`
})

const app = new Vue({
  router
}).$mount('#app')
```

Wynik:



## VIII. Wtyczki Vue<sup>7</sup>

*Cel: Zapoznanie z procesem tworzenia wtyczek Vue.js.*

Wtyczki – jak ma to miejsce w innych produktach – pozwalają na poszerzanie możliwości frameworka: globalne metody, właściwości lub zasoby: nowe dyrektywy, filtry, efekty przejścia itp...

Instalacja wtyczki:

```
npm install|yarn add <nazwa-wtyczki> --save-dev
```

Wykorzystanie wtyczki globalne:

```
import VueSomePlugin from 'vue-some-plugin';

// import the styles
import 'vue-some-plugin/dist/vue-some-plugin.css'

Vue.use(VueSomePlugin);
```

Wewnątrz komponentu:

```
import { VueSomePlugin } from 'vue-some-plugin';

// add to component
```

<sup>7</sup> <https://vuejs.org/v2/guide/plugins.html>

```
components: {  
  VueSomePlugin,  
}
```

Jako przykład zainstalujemy wtyczkę udostępniającą łatwą w użyciu i wydajną tabelę pozwalającą na: sortowanie, filtrowanie kolumn, paginacja itp.:<sup>8</sup>

```
npm install|yarn add vue-good-table --save-dev
```

Dodajmy kod komponentu `VueGoodTable.vue`:

```
<template>  
  <div>  
    <vue-good-table :columns="columns"  
                    :rows="rows" />  
  </div>  
</template>  
  
<script>  
  export default {  
    name: 'my-component',  
    data() {  
      return {  
        columns: [  
          {  
            label: 'Name',  
            field: 'name',  
          },  
          {  
            label: 'Age',  
            field: 'age',  
            type: 'number',  
          },  
          {  
            label: 'Created On',  
            field: 'createdAt',  
            type: 'date',  
            dateInputFormat: 'YYYY-MM-DD',  
            dateOutputFormat: 'MMM Do YY',  
          },  
          {  
            label: 'Percent',  
            field: 'score',  
            type: 'percentage',  
          },  
        ],  
      }  
    }  
  }  
</script>
```

---

<sup>8</sup> <https://xaksis.github.io/vue-good-table/guide/#basic-example>

```

      rows: [
        { id: 1, name: "John", age: 20, createdAt: '201-10-31:9: 35 am', score: 0.03343 },
        { id: 2, name: "Jane", age: 24, createdAt: '2011-10-31', score: 0.03343 },
        { id: 3, name: "Susan", age: 16, createdAt: '2011-10-30', score: 0.03343 },
        { id: 4, name: "Chris", age: 55, createdAt: '2011-10-11', score: 0.03343 },
        { id: 5, name: "Dan", age: 40, createdAt: '2011-10-21', score: 0.03343 },
        { id: 6, name: "John", age: 20, createdAt: '2011-10-31', score: 0.03343 },
      ],
    };
  },
};
</script>

```

Zaimportujemy wtyczkę globalnie i wyświetlmy komponent (plik `main.js`):

```

import Vue from 'vue'
import VueGoodTable from './VueGoodTable.vue'
import VueGoodTablePlugin from 'vue-good-table';

// import the styles
import 'vue-good-table/dist/vue-good-table.css'

Vue.use(VueGoodTablePlugin);

new Vue({
  el: '#app',
  render: h => h(VueGoodTable)
})

```

Wynik powinien być następujący:

localhost:8080			
Name	Age	Created On	Percent
John	20	Oct 31st 01	3.34%
Jane	24	Oct 31st 11	3.34%
Susan	16	Oct 30th 11	3.34%
Chris	55	Oct 11th 11	3.34%
Dan	40	Oct 21st 11	3.34%
John	20	Oct 31st 11	3.34%

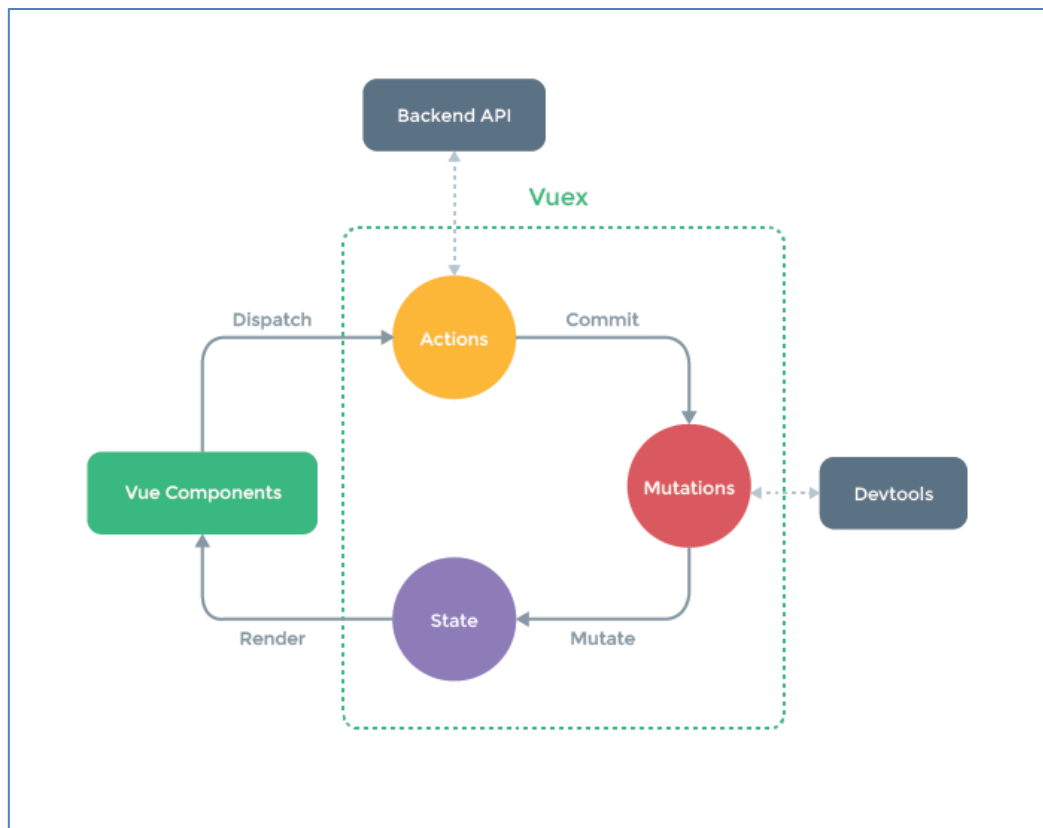
## IX. Stan aplikacji Vue (Vuex)

*Cel: Zapoznanie z zarządzaniem stanem aplikacji Vue.js.*

Wzrost złożoności dużych aplikacji często może być spowodowany dużą liczą elementów stanów rozproszonych na wiele komponentów oraz interakcje między nimi. Z pomocą przychodzi wzorzec Single source of truth (SSOT) realizowany przez Vuex. Magazyn Vuex składa się z:

- stanów – obiekt reprezentujący początkowy stan aplikacji,
- mutacji – obiekt zawierający funkcje akcji, które wpływają na stan.

Można zaimportować go do dowolnego komponentu: `Vue.use(Vuex)`. Zaimportowany w głównym pliku (`App.vue`) będzie dostępny we wszystkich komponentach (zmienna: `this.$score`). Jest reaktywny – zmiana stanu powoduje aktualizację widoków przez komponenty.



Rysunek 1 - Vuex<sup>9</sup>

Komponenty mogą jedynie zatwierdzać mutacje, nie mogą bezpośrednio zmieniać stanu. Popelnienie mutacji zamiast bezpośrednio zmieniać stan, pozwala ją jawnie śledzić. Ta prosta konwencja sprawia, że intencje programisty są bardziej wyraźne, łatwiej można rozumieć zmiany stanu w aplikacji, czytając kod. Ułatwia zrozumienie, jaki rodzaj mutacji może się zdarzyć i jak są one wyzwalane. Ponadto jeśli coś pójdzie nie tak, będzie to rejestrowane w dzienniku. Daje to możliwość wdrożenia narzędzi, które mogą rejestrować każdą mutację, wykonywać migawki stanu a nawet przeprowadzać debugowanie.

Instalacja Vuex'a:

```
npm install vuex --save|yarn add vuex
```

Prosty kod wykorzystujący magazyn zawierający jedną mutację, wraz z jej popelnieniem:

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

const store = new Vuex.Store({
  state: {
    count: 0
  },
```

<sup>9</sup> <https://vuex.vuejs.org/>

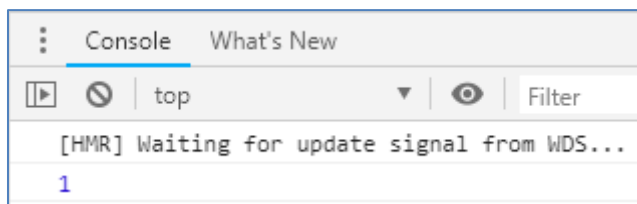
```

mutations: {
  increment(state) {
    state.count++
  }
}
})

store.commit('increment')
console.log(store.state.count) // -> 1

```

Wynik w oknie konsoli:



## X. Testowanie aplikacji Vue

*Cel: Zapoznanie z testowaniem aplikacji Vue.js.*

Dla testów jednostkowych domyślnym środowiskiem jest Karma. Dla E2E wykorzystywany jest framework Nightwatch.js z Selenium WebDriver. Testy End-to-End pozwalają na całosciowe testowanie aplikacji. Nie ma atrap ani imitacji, testowany jest rzeczywisty system. Można sprawdzać takie aspekty aplikacji jak:

- API,
- kod po stronie klienta/serwera,
- bazy danych,
- obciążenie serwera.

Zapewniona jest wysoka jakość integracji systemu.

Uruchomienie testów jednostkowych:

```
npm|yarn run unit
```

Kod przykładowego testu jednostkowego (domyślnie dodany w zaawansowanych szablonach):

```

import Vue from 'vue'
import Hello from 'src/components/Hello'

describe('Hello.vue', () => {
  it('should render correct contents', () => {
    const vm = new Vue({
      template: '<div><hello></hello></div>',
      components: { Hello }
    }).$mount()
  })
})

```

```

    expect(vm.$el.querySelector('.hello
h1').textContent).toContain('Hello World!')
  })
})

```

Dostępny jest również przykładowy test E2E:

```

// For authoring Nightwatch tests, see
// http://nightwatchjs.org/guide#usage

module.exports = {
  'default e2e tests': function (browser) {
    browser
      .url('http://localhost:8080')
      .waitForElementVisible('#app', 5000)
      .end()
  }
}

```

Uruchomienie testów End-to-End:

```
npm|yarn run e2e
```

Wynik poprawnie przeprowadzonego testu:

```

yarn run v1.12.1
$ node test/e2e/runner.js
Starting selenium server... started - PID: 47400

[Test] Test Suite
=====

Running:  default e2e tests
  ✓ Element <#app> was visible after 53 milliseconds.
  ✓ Testing if element <.hello> is present.
  ✓ Testing if element <h1> contains text: "Welcome to Your Vue.js App".
  ✓ Testing if element <img> has count: 1

OK. 4 assertions passed. (5.493s)

Done in 11.30s.

C:\Users\goluch\OneDrive\17.sem-zim_201819\RAI\Vue\code\labVue>

```