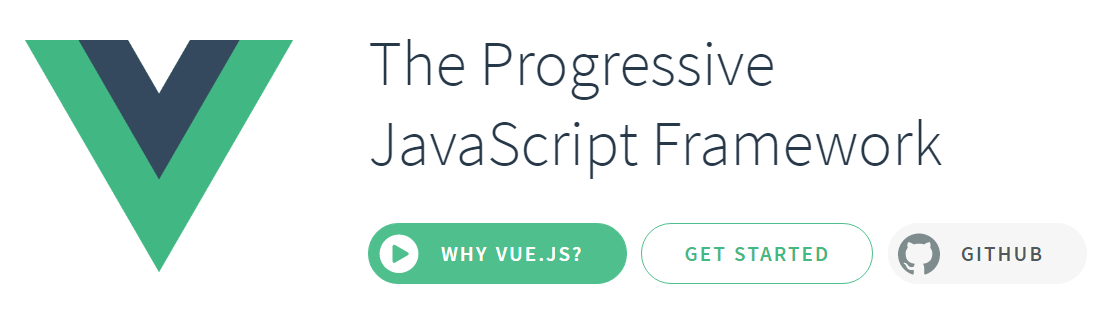
**Poradnik – Rozdział 2**

W tym rozdziale skupimy się na omówieniu struktury Vue.js : dyrektywy, komponenty, routing itd. Omówimy podstawową strukturę Bootstrap’a oraz platformę Firebase, zobaczymy co to jest i jakie zapewnia funkcjonalności. Sprawdzimy jak można połączyć wszystkie wymienione technologie, aby uzyskać prostotę w opracowaniu złożonych rzeczy.

Oficjalna strona Vue.js -- <https://vuejs.org>



Oficjalna witryna Vue.js sugeruje, że Vue jest progresywnym frameworkiem JavaScript

**Co to znaczy?**

Vue pozwala nam powiązać dane z interfejsem, a reszta akcji, które muszą zajść między danymi, a interfejsem, zachodzi sama. Spójrzmy na bardzo prosty przykład, w którym powiążemy wiadomość z tytułem strony. Zacznijmy od definiowania prostej struktury HTML do której dołączymy Bootstrap oraz trochę styli, które będziemy wykorzystywać później.

Index.html

<!DOCTYPE html>

<html lang="pl">

<head>

    <meta charset="UTF-8">

    <title>Vue.js - wiązanie danych</title>

    <style>

        #app {

            width: 60%;

            margin: 20px auto auto auto;

        }

        input {

            padding-left: 10px;

        }

        .animal {

            font-size: 4em;

        }

        .red {

            color: red;

        }

        .green {

            color: green;

        }

    </style>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"

        crossorigin="anonymous">

</head>

<body>

    <div id="app" class="jumbotron">

        <div class="row">

            <div class="col-9">

                <h1>Witaj! Nauczmy się Vue.js</h1>

            </div>

        </div>

    </div>

    <script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"

        integrity="sha384-A7FZj7v+d/sdmMqp/nOQwliLvUsJfDHW+k9Omg/a/EheAdgtzNs3hpfag6Ed950n"

        crossorigin="anonymous"></script>

</body>

</html>

Teraz zainicjujemy na stronie instancję Vue.js i powiążemy jej dane z elementem <h1>.

W tym przykładzie Vue zaimportujemy korzystając z CDN <https://unpkg.com/vue>

Musimy umieścić go w tagu <script>

    ...

    <script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"

    integrity="sha384-A7FZj7v+d/sdmMqp/nOQwliLvUsJfDHW+k9Omg/a/EheAdgtzNs3hpfag6Ed950n"

    crossorigin="anonymous"></script>

    <script src="https://unpkg.com/vue"></script>

</body>

</html>

Zainicjujmy teraz instancję Vue, minimum które potrzebuje to element, do którego ma zostać dołączona oraz obiekt danych.

Chcemy dołączyć naszą instancję do głównego tagu <div> z identyfikatorem **app.** Utworzymy również obiekt danych zawierający wpis z imieniem.

    ...

    <script src="https://unpkg.com/vue"></script>

    <script>

        var data = { name: 'Mati' }

        new Vue({

            el: '#app',

            data

        })

    </script>

</body>

</html>

Powiążemy teraz dane z naszym elementem HTML, zrobimy to za pomocą podwójnych nawiasów klamrowych ({{}}). Wszystko co umieścimy w podwójnych nawisach klamrowych zostanie zinterpretowane. Jeśli więc umieścimy w nich na przykład 2 + 2 to na stronie zostanie wyrenderowane 4. Tak, więc w naszym przypadku, jeśli wstawisz {{name}} w dowolnym miejscu wewnątrz znacznika HTML, zobaczysz imię, które przekazaliśmy do instancji Vue w obiekcie danych.

<div id="app" class="jumbotron">

    <div class="row">

        <div class="col-9">

            <h1>Witaj, {{name}}! Nauczmy się Vue.js</h1>

        </div>

    </div>

    <hr />

</div>

Vue.js wspiera dwukierunkowe wiązanie danych ( **two-way data binding** ), więc zmiany, które zaszły w elemencie na stronie są propagowane do danych z którymi element jest powiązany.

Aby to osiągnąć, wystarczy powiązać dany fragment danych z elementem za pomocą atrybutu v-model. Na przykład dodajmy pole typu input do naszej strony i powiążmy je z naszymi danymi ( z atrybutem name).

<div id="app" class="jumbotron">

    <div class="row">

        <div class="col-9">

            <h1>Witaj, {{name}}! Nauczmy się Vue.js</h1>

        </div>

    </div>

    <hr />

    <div>

        <label for="name">Jak masz na imię? </label>

        <input id="name" type="text" v-model="name">

    </div>

    <hr />

</div>

Jak widać, to nic trudnego. Wszystko, czego potrzebujesz, to przekazanie danych do instancji Vue i powiązanie ich z elementami Twojej strony. Platforma Vue robi wszystko inne.

**Instalacja Vue**

Istnieje wiele sposobów dołączenia Vue do naszego projektu. Przedstawmy więc kilka z nich:

1. **Dołączenie pliku Vue.js do projektu**

Możesz użyć Vue.js, po prostu pobierając go i umieszczając w tagu <script>.

<https://vuejs.org/js/vue.js>

<script src="vue.js"></script>

1. **CDN**

Z tej metody skorzystaliśmy w poprzednim przykładzie, dlatego teraz ją pominiemy.

<script src="https://unpkg.com/vue"></script>

1. **NPM**

Możemy skorzystać z menadżera pakietów NPM i dodać vue do naszego pliku package.json korzystając z polecenia

npm install vue --save

1. **Vue-Cli**

Pierwszym krokiem jest instalacja Vue-Cli poleceniem

npm install --global vue-cli

Teraz możesz rozpocząć nowy projekt za pomocą interfejsu wiersza poleceń Vue. możliwe jest skonfigurowanie projektu przy użyciu różnych szablonów - począwszy od prostego projektu pojedynczej strony HTML, a skończywszy na złożonej konfiguracji projektu w pakiecie internetowym. Polecenie, którego należy użyć do tworzenia rusztowania   
(scaffolding) projektu Vue, jest następujące:

vue init <nazwa-szablonu> <nazwa-projektu>

**Dostępne są następujące szablony:**

• **webpack**: jest to w pełni funkcjonalna konfiguracja pakietu internetowego z programem ładującym vue. Obsługuje hot reload, linting, testowanie, wszelkiego rodzaju preprocesory i tak dalej.

• **webpack-simple**: Jest to prosta konfiguracja pakietu internetowego, która jest przydatna do szybkiego tworzenia prototypów.

• **browserify**: Jest to w pełni funkcjonalna konfiguracja browserify z vueify, która obsługuje również hot reload, linting i testy jednostkowe.

• **browserify-simple**: Jest to prosta konfiguracja browserify z vueify, której można użyć do szybkiego prototypowania.

• **simple**: generuje prostą stronę HTML zawierającą Vue.js. Jest idealny do poznania funkcji.

Możliwe jest również tworzenie własnych szablonów. Dokumentacja dostępna pod adresem <https://github.com/vuejs/vue-cli#custom-templates>

Aby załadować projekt przy użyciu szablonu webpack, po prostu uruchom następujący wiersz kodu:

vue init webpack mój-projekt

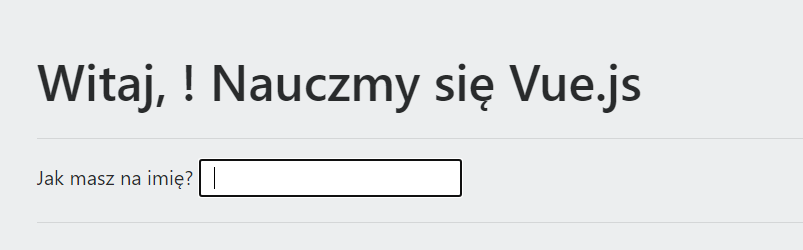
**Dyrektywy Vue**

Dyrektywy Vue to nic innego jak atrybuty dołączone do elementów HTML.

Te dyrektywy zapewniają dodatkowe funkcje szablonu. Wszystkie dyrektywy zaczynają się od przedrostka v-.

**Renderowanie warunkowe**

W przypadku gdy nasz input jest pusty, wiadomość powitalna na stronie nie wygląda estetycznie.



Spróbujmy to teraz naprawić, tak, aby komunikat „Witaj, !” pojawiał się dopiero gdy wpiszemy imię.

Dyrektywy v-show i v-if są używane dokładnie do renderowania warunkowego. Otwórzmy plik index.html i zmieńmy go dodając dyrektywę v-show z wartością name.

<div id="app" class="jumbotron">

    <div class="row">

        <div class="col-9">

            <h1><span v-show="name">Witaj, <strong>{{name}}</strong>! </span>Nauczmy się Vue.js</h1>

        </div>

    </div>

    ...

**Text versus HTML ( różnice w interpolacji )**

Teraz skupimy się na różnicach w renderowania elementów korzystając ze składni wąsów ({{}}) oraz dyrektyw v-text i v-html. Dla przykładu wykorzystamy Unicode emoji. Wróćmy do naszego pliku html i dodajmy do niego następujący kod:

<script>

    const animalsCodes = {

           dog: '&#x1f436;',

           cat: '&#x1f638;',

           monkey: '&#x1f435;',

           unicorn: '&#x1f984;'

   }

   var data = {

       name: 'Mati',

       animalsCodes

   }

   new Vue({

       el: '#app',

       data

   })

</script>

Spróbujmy wyświetlić nasze emoji korzystając z tych trzech sposobów:

Zacznijmy od wykorzystania składni wąsów:

<div id="app" class="jumbotron"></div>

    ...

    <div class="animal">

        <div>{{animalsCodes.cat}}</div>

    </div>

    ...

Po odświeżeniu strony widzimy Unicode, a nie nasze emoji. Dzieje się tak, ponieważ interpolacja wąsów w rzeczywistości interpoluje tekst. Używanie interpolacji wąsów jest takie samo, jak używanie dyrektywy v-text:

<div id="app" class="jumbotron"></div>

    ...

    <div class="animal">

        <div v-text="animalsCodes.cat"></div>

    </div>

    ...

To, co tak naprawdę chcemy tutaj wyrenderować, to nie tekst. Chcemy, aby wartość Unicode dla emoji renderowana była jako HTML. Wystarczy, że zamienimy dyrektywę v-text na dyrektywę v-html, aby otrzymać pożądany efekt:

<div id="app" class="jumbotron"></div>

    ...

    <div class="animal">

        <div v-html="animalsCodes.cat"></div>

    </div>

    ...

Czyli jeżeli chcemy na naszej stronie wyświetlać tekst używamy notacji wąsów lub dyrektywy v-text, natomiast do renderowania czystego html korzystamy z dyrektywy v-html.

**Pętle**

Przyjmijmy sytuację, że mamy kilka zwierząt, chcielibyśmy je wszystkie wyświetlić   
w uporządkowanej liście. Możemy to zrobić używając wcześniej poznanej dyrektywy v-html.

<div class="animal">

    <ol>

        <li v-html="animalsCodes.cat"></li>

        <li v-html="animalsCodes.dog"></li>

        <li v-html="animalsCodes.monkey"></li>

        <li v-html="animalsCodes.unicorn"></li>

    </ol>

</div>

Jednak nie jest to optymalne rozwiązanie. Do tego typu zadań lepiej używać pętli. Stwórzmy więc tablicę, gdzie przechowamy wszystkie nasze zwierzęta (obiekty). Następnie wyeksportujmy tę tablicę do obiektu data.

<script>

    const animalsCodes = {

            dog: '&#x1f436;',

            cat: '&#x1f638;',

            monkey: '&#x1f435;',

            unicorn: '&#x1f984;'

    }

    const animals = ['dog', 'cat', 'monkey', 'unicorn']

    var data = {

        name: 'Mati',

        animalsCodes,

        animals

    }

    new Vue({

        el: '#app',

        data

    })

</script>

Teraz możemy użyć naszej tablicy i dyrektywy v-for, aby wyświetlić wszystkie elementy   
w liście.

<div class="animal">

    <ol>

        <li v-for="animal in animals" v-html="animalsCodes[animal]"></li>

    </ol>

</div>

**Wiązanie danych**

Aby powiązać dane skorzystaliśmy wcześniej z dyrektywy v-model (wiązanie dwukierunkowe). Możemy jej używać tylko z polami typu: input, select oraz textarea. Ta dyrektywa akceptuje kilka modyfikatorów. ( Modyfikatory to specjalne słowa kluczowe, które w jakiś sposób wpływają na dane wejściowe lub ich zachowanie )

Tymi modyfikatorami są:

* .lazy – modyfikator ten sprawia że dane są aktualizowane przy zdarzeniu zmiany   
  ( w przypadku pola input zobaczymy zmianę dopiero po kliknięciu klawisza „ENTER”).
* .number – modyfikator, który rzutuje wartość z pola input na typ liczbowy (konwersja string na numer)
* .trim - modyfikator, który przycina wartość podaną przez użytkownika

Modyfikatory możemy ze sobą łączyć. W tym przykładzie dane zostaną zaktualizowane po kliknięciu klawisza „ENTER”

<input id="name" type="text" v-model.lazy.trim="name">

Możemy używać również dyrektywy v-bind, która pozwala nam wiązać dane z dowolnym elementem strony lub jego właściwością. W tym przykładzie chcemy wyświetlić odpowiednią emotkę w zależności od zawartości pola input

W tym celu pobieramy i umieszczamy dwa obrazki w folderze img. Następnie eksportujemy ich ścieżki do obiektu danych

var data = {

    name: 'Mati',

    animalsCodes,

    animals,

    img1: 'img/emotka1.png',

    img2: 'img/emotka2.png'

}

Teraz umieszczamy tag img w którym korzystamy z dyrektywy v-bind, w której sprawdzamy stan pola input i na jego podstawie wyświetlamy odpowiedni obraz.

<div id="app" class="jumbotron">

        <div class="row">

            <div class="col-9">

                <h1><span v-show="name">Witaj, <strong>{{name}}</strong>! </span>Nauczmy się Vue.js</h1>

            </div>

            <div class="col-3">

                <img width="50%" v-bind:src="name ? img1 : img2">

*<!-- skrócony zapis -->*

*<!-- <img width="50%" :src="name ? img1 : img2">  -->*

            </div>

        </div>

        <hr />

...

**Obsługa zdarzeń**

Vue zapewnia bardzo przyjemny sposób dołączania detektorów do zdarzeń (event listner)   
w dowolnym elemencie DOM i zapewnia metody, które mogą obsłużyć te zdarzenia. Zaletą tych metod jest to, że mają bezpośredni dostęp do danych Vue za pomocą słowa kluczowego this. W ten sposób możemy manipulować danymi metodami, a ponieważ dane te są reaktywne, wszystkie zmiany będą natychmiast propagowane do elementów, z którymi te dane są powiązane.

Aby stworzyć metodę trzeba dodać obiekt metody do sekcji eksport aplikacji Vue. Aby dołączyć tą metodę do detektora zdarzeń użyjemy dyrektywy v-on z odpowiadającym zdarzeniem po dwukropku.

v-on:sumbit="handleSubmit"

v-on:click="handleClick"

v-on:hover="handleHover"

Możemy się spotkać również ze skróconym zapisem.

@sumbit="handleSubmit"

@click="handleClick"

@hover="handleHover"

Teraz skupimy się na praktycznych zastosowaniach wykorzystując projekt z ostatnich zajęć.

Użyliśmy tam detektora v-on:submit, aby wywołać metodę AddMessage. Poniżej przedstawiamy fragment kodu w którym zostało dodane nasłuchiwanie zdarzenia submit, które wywołuje metodę addMessage

<form *@submit*="addMessage">  
 <div *class*="form-group">  
 <input *class*="form-control" *v-model*="newMessage.title" *maxlength*="40" *autofocus placeholder*="Wpisz swoje imię"/>  
 </div>  
 <div *class*="form-group">  
 <textarea *class*="form-control" *v-model*="newMessage.text" *placeholder*="Napisz tu swoją wiadomość" *rows*="3"></textarea>  
 </div>  
 <button *class*="btn btn-primary" *type*="submit">Wyślij</button>  
</form>

*export default* {  
 name: 'app',  
 firebase: {  
 messages: messagesRef  
 },  
 data () {  
 *return* {  
 messages: [],  
 newMessage: {  
 title: '',  
 text: '',  
 timestamp: *null* }  
 }  
 },  
 methods: {  
 dateToString: *dateToString*,  
 addMessage (e) {  
 e.preventDefault()  
 *if* (*this*.newMessage.title === '') {  
 *return* }  
 *this*.newMessage.timestamp = Date.now()  
 messagesRef.push(*this*.newMessage)  
 *this*.newMessage.text = ''  
 *this*.newMessage.title = ''  
 *this*.newMessage.timestamp = *null* }  
 },  
 components: {  
 Card  
 }  
}

Aby lepiej to zrozumieć, dodajmy kilka metod do naszej strony zoo. Dodajmy element wielokrotnego wyboru, który będzie zawierał wszystkie możliwe opcje, a Twoje zoo zostanie wypełnione czymś, co faktycznie wybierzesz. Żeby to zrobić wykonamy poniższe kroki:

1. Dodajmy więcej zwierząt do naszej mapy animalsCodes

 var animalsCodes = {

            dog: '&#x1f436;',

            cat: '&#x1f638;',

            monkey: '&#x1f435;',

            unicorn: '&#x1f984;',

            tiger: '&#x1f42f;',

            mouse: '&#x1f42d;',

            rabbit: '&#x1f430;',

            cow: '&#x1f42e;',

            whale: '&#x1f433;',

            horse: '&#x1f434;',

            pig: '&#x1f437;',

            frog: '&#x1f438;',

            koala: '&#x1f43c;'

        }

1. Teraz wyciągamy klucze z obiektu animalsCodes i tworzymy z nich tablicę.

var animals = Object.keys(animalsCodes)

1. Tworzymy pustą tablicę animalsForZoo w której będziemy przechowywać zwierzęta, które będą w naszym ZOO

var animalsForZoo = []

var data = {

    name: 'Mati',

    animals,

    animalsCodes,

    animalsForZoo,

    img1: 'img/emotka1.png',

    img2: 'img/emotka2.png'

}

1. Zmieniamy tablicę z której wyświetlamy listę zwierząt

<div id="app" class="jumbotron">

    ...

    <hr />

    <div v-if="animalsForZoo.length > 0">

        <h2><span v-if="name">{{name}}! </span>To są zwierzęta w Twoim Zoo: </h2>

        <ol>

            <li v-for="animal in animalsForZoo"><span class="animal" v-html="animalsCodes[animal]"></span></li>

        </ol>

    </div>

</div>

1. Dodajmy element wielokrotnego wyboru, który będzie zawierał wszystkie dostępne zwierzęta

<div id="app" class="jumbotron">

    ...

    <div class="row">

        <select class="col-12" multiple="multiple" name="animals" id="animals">

            <option v-for="animal in animals" :value="animal">{{animal}}</option>

        </select>

    </div>

    <hr />

    <div v-if="animalsForZoo.length > 0">

        <h2><span v-if="name">{{name}}! </span>To są zwierzęta w Twoim Zoo: </h2>

        <ol>

            <li v-for="animal in animalsForZoo"><span class="animal" v-html="animalsCodes[animal]"></span></li>

        </ol>

    </div>

</div>

1. Dodajemy detektor zdarzeń do pola wielokrotnego wyboru, który będzie oczekiwał zdarzenia zmiany (change event), które jeśli się wydarzy wywoła metodę populateAnimalsForZoo

<div class="row">

    <select @change="populateAnimalsForZoo" class="col-12" multiple="multiple" name="animals" id="animals">

        <option v-for="animal in animals" :value="animal">{{animal}}</option>

    </select>

</div>

1. Tworzymy metodę, która pozwoli dodawać wybrane przez nas zwierzątka do stworzonej wcześniej tablicy

new Vue({

    el: '#app',

    data,

    methods: {

        populateAnimalsForZoo(ev) {

            this.animalsForZoo = []

            const selected = document.querySelectorAll('#animals option:checked')

            for (var i = 0; i < selected.length; i++) {

                this.animalsForZoo.push(selected[i].value)

            }

        }

    }

})

**Komponenty Vue**

Vue.js wspiera strukturę aplikacji opartą na komponentach. Tworzenie komponentów jest dosyć proste. Są tylko trzy rzeczy które musisz zrobić:

1. Stwórz komponent, nadaj mu szablon, dane, metody i cokolwiek potrzebujesz
2. Dołącz je w swojej aplikacji Vue w obiekcie components
3. Użyj ich w szablonie aplikacji

Stwórzmy nowy plik html zawierający podstawową strukturę.

<!DOCTYPE html>

<html lang="pl">

    <head>

        <meta charset="UTF-8">

        <meta http-equiv="X-UA-Compatible" content="IE=edge">

        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <title>Vue.js - tworzenie komponentów</title>

    </head>

    <body>

        <div id="app">

        </div>

        <script src="https://unpkg.com/vue"></script>

        <script>

        </script>

    </body>

</html>

Teraz stworzymy nasz przykładowy komponent, który wyrenderuje element nagłówka mówiący „Cześć”. Nazwijmy go HelloComponent.

<script>

    var HelloComponent = {

        template: '<h1>Cześć!</h1>'

    }

    new Vue( {

        el: '#app',

        components: {

            HelloComponent

        }

    })

</script>

Teraz możemy umieścić nasz komponent w div=”app”.

<div id="app">

    <hello-component></hello-component>

</div>

Spróbujmy teraz rozbudować nasz komponent. Dodajmy możliwość wprowadzenia imienia.

<script>

    var HelloComponent = {

        template: '<div>' +

        '<input v-model="name" />' +

        '<h1>Cześć, <strong>{{name}}</strong>!</h1>' +

        '</div>',

        data() {

            return {

                name: 'Tomek'

            }

        }

    }

    new Vue( {

        el: '#app',

        components: {

            HelloComponent

        }

    })

</script>

Każdy komponent może być przechowywany w oddzielnym pliku z własnym kodem html, css, js. Jeśli chcesz tego dokonać musisz użyć jakiegoś modułowego pakietu np. webpacka. Spróbujmy przenieść naszą stronę z zoo do aplikacji z szablonem webpack.

Uruchommy więc skrypt inicjalizujący:

vue init webpack zoo

cd zoo

npm install

npm run dev

Teraz otwórzmy plik App.vue znajdujący się w katalogu src i wklejmy wcześniej utworzony kod w sekcji script:

<script>  
  
*var* animalsForZoo = []  
*var* animalsCodes = {  
 dog: '&#x1f436;',  
 cat: '&#x1f638;',  
 monkey: '&#x1f435;',  
 unicorn: '&#x1f984;',  
 tiger: '&#x1f42f;',  
 mouse: '&#x1f42d;',  
 rabbit: '&#x1f430;',  
 cow: '&#x1f42e;',  
 whale: '&#x1f433;',  
 horse: '&#x1f434;',  
 pig: '&#x1f437;',  
 frog: '&#x1f438;',  
 koala: '&#x1F9CE;'  
}  
*var* animals = Object.keys(animalsCodes)  
*var* data = {  
 name: 'Mati',  
 animals,  
 animalsCodes,  
 animalsForZoo,  
 img1: '../static/images/emotka1.png',  
 img2: '../static/images/emotka2.png'  
}  
  
*export default* {  
 name: 'app',  
 data () {  
 *return* data  
 }  
}  
</script>

Dodajmy wykorzystywane przez nas wcześniej style w sekcji style:

<style>  
#app {  
 width: 60%;  
 margin: 20px auto auto auto;  
}  
  
input {  
 padding-left: 10px;  
}  
  
.animal {  
 font-size: 4em;  
}  
  
.red {  
 color: red;  
}  
  
.green {  
 color: green;  
}  
</style>

Przejdźmy do folderu static, stwórzmy folder images i wkleimy do niego wcześniej wykorzystywane zdjęcia.

Do pliku index.html podpinamy Bootstrap

<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>zoo</title>

    <link

        rel="stylesheet"

        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css"

        crossorigin="anonymous">

  </head>

  <body>

    <div id="app"></div>

*<!-- built files will be auto injected -->*

  </body>

</html>

Teraz stwórzmy nasz komponent. Przechodzimy do folderu components i tworzymy plik Zoo.vue. Utwórzmy w nim podstawową strukturę:

<template>  
   
</template>  
  
<script>  
  
</script>  
  
<style>  
  
</style>

W sekcji template umieśćmy następujący kod:

<div *v-if*="animals.length > 0">  
 <h2><span *v-if*="name">{{name}}! </span>To jest Twoje Zoo</h2>  
 <ol>  
 <li *v-for*="animal *in* animals" *v-bind:key*="animal"><span *class*="animal"  
 *v-html*="animalsCodes[animal]"></span></li>  
 </ol>  
</div>

W sekcji script:

*export default* {  
 props: ['animals', 'animalsCodes', 'name']  
}

Teraz możemy zaimportować stworzony przez nas komponent w App.vue

<script>  
*import* Zoo *from* './components/Zoo'  
  
*var* animalsForZoo = []  
*var* animalsCodes = {  
 dog: '&#x1f436;',  
 cat: '&#x1f638;',  
 monkey: '&#x1f435;',  
 unicorn: '&#x1f984;',  
 tiger: '&#x1f42f;',  
 mouse: '&#x1f42d;',  
 rabbit: '&#x1f430;',  
 cow: '&#x1f42e;',  
 whale: '&#x1f433;',  
 horse: '&#x1f434;',  
 pig: '&#x1f437;',  
 frog: '&#x1f438;',  
 koala: '&#x1F9CE;'  
}  
*var* animals = Object.keys(animalsCodes)  
*var* data = {  
 name: 'Mati',  
 animals,  
 animalsCodes,  
 animalsForZoo,  
 img1: '../static/images/emotka1.png',  
 img2: '../static/images/emotka2.png'  
}  
  
*export default* {  
 name: 'app',  
 components: {  
 Zoo  
 },  
 data () {  
 *return* data  
 }  
}  
</script>

Teraz możemy użyć naszego komponentu wewnątrz szablonu. Zastąpmy całą sekcję template w pliku App.vue następującym kodem :

<template>

  <div id="app" class="jumbotron">

    <div class="row">

      <div class="col-9"><h1><span v-if="name">Witaj, <strong>{{name}}</strong>! </span><span>Nauczmy się Vue.js</span></h1></div>

      <div class="col-3"><img width="50%" :src="name ? img1 : img2"></div>

    </div>

    <hr />

    <div>

      <label for="name" :class="{green: name, red: !name}">Jak masz na imię? </label>

      <input id="name" type="text" v-model.trim="name">

    </div>

    <hr />

    <h2><span v-if="name">{{name}}! </span>Wybierz zwierzę </h2>

    <div class="row">

      <select v-model="animalsForZoo" class="col-12" multiple="multiple" name="animals" id="animals">

        <option v-for="animal in animals" v-bind:key="animal" :value="animal">{{animal}}</option>

      </select>

    </div>

    <hr />

    <zoo :animals="animalsForZoo" :animalsCodes="animalsCodes" :name="name"></zoo>

  </div>

</template>

**Vue router**

W tej sekcji skupimy się na trasowaniu w Vue. Jest to przydatna technika do nawigowania bez przeładowania naszej strony (single page). Vue.js to jeden z frameworków, który oferuje sposób radzenia sobie z routingiem. Po prostu odwzorowujesz ścieżkę adresu URL na swoje komponenty i wszystko działa.

Oficjalną dokumentacja biblioteki vue-router <https://router.vuejs.org/en/>

Tworzymy nowy projekt w którym pokażemy działanie vue-router (nie instalujemy vue-router podczas tworzenia projektu)

Aby móc korzystać z vue-router w istniejącym projekcie musisz go zainstalować poleceniem

npm install vue-router –save

Vue-router można zainstalować również podczas tworzenia projektu



Po uruchomieniu projektu przechodzimy do pliku main.js w katalogu src gdzie zaimportujemy vue-router.

*// The Vue build version to load with the `import` command  
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.  
import* Vue *from* 'vue'  
*import* App *from* './App'  
*import* Router *from* 'vue-router'  
  
Vue.use(Router)  
  
Vue.config.productionTip = *false  
  
/\* eslint-disable no-new \*/  
new* Vue({  
 el: '#app',  
 components: { App },  
 template: '<App/>'  
})

Teraz musimy utworzyć nasze komponenty. Przechodzimy do folderu components i tworzymy trzy pliki:

1. Hello.vue

<template>  
 <h1>Witaj!</h1>  
</template>

1. Home.vue

<template>  
 <h1>Witaj w swojej aplikacji</h1>  
</template>

1. Bye.vue

<template>  
 <h1>Żegnaj!</h1>  
</template>

Po utworzeniu komponentów przechodzimy do pliku main.js w katalogu src i importujemy nowo utworzone komponenty

*import* Home *from* '@/components/Home'  
*import* Hello *from* '@/components/Hello'  
*import* Bye *from* '@/components/Bye'

Teraz musimy utworzyć obiekt naszego routera i przekazujemy go do obiektu Vue.js

*// The Vue build version to load with the `import` command  
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.  
import* Vue *from* 'vue'  
*import* App *from* './App'  
*import* Router *from* 'vue-router'  
*import* Home *from* '@/components/Home'  
*import* Hello *from* '@/components/Hello'  
*import* Bye *from* '@/components/Bye'  
  
Vue.use(Router)  
  
Vue.config.productionTip = *false  
var* router = *new* Router({  
 mode: 'history',  
 routes: [  
 {  
 name: 'home',  
 component: Home,  
 path: '/'  
 },  
 {  
 name: 'hello',  
 component: Hello,  
 path: '/hello'  
 },  
 {  
 name: 'bye',  
 component: Bye,  
 path: '/bye'  
 }  
 ]  
})  
*/\* eslint-disable no-new \*/  
new* Vue({  
 el: '#app',  
 components: { App },  
 template: '<App/>',  
 router  
})

Ostatnim krokiem będzie podpięcie routera w głównym szablonie projektu. Przechodzimy do pliku App.vue i wklejamy następujący kod w sekcji template i script:

<template>  
 <div *id*="app">  
 <img *src*="./assets/logo.png">  
 <hr/>  
 <router-view></router-view>  
 </div>  
</template>  
  
<script>  
  
*export default* {  
 name: 'App'  
}  
</script>

Używając vue-router możemy też stworzyć menu do nawigacji korzystając z <router-link>

Są dwa sposobu:

1. Przez wskazanie ścieżki
2. Odwołanie się do nazwy komponentu i powiązanie ich metodą v-bind:to lub :to

Poniższy kod prezentuje działanie obu z nich. Umieść poniższy fragment w sekcji template w pliku App.vue

<template>  
 <div *id*="app">  
 <img *src*="./assets/logo.png">  
 <hr/>  
 <router-link *to*="/">Home</router-link>  
 <router-link *to*="hello">Hello</router-link>  
 <router-link *to*="bye">Bye</router-link>  
 *<!-- drugi sposób -->* <router-link *:to*="{name: 'home'}">Home</router-link>  
 <router-link *:to*="{name: 'hello'}">Hello</router-link>  
 <router-link *:to*="{name: 'bye'}">Bye</router-link>  
 <router-view></router-view>  
 </div>  
</template>

**Architektura zarządzania stanem Vuex (Vuex state management architecture)**

Według dokumentacji nie możemy zmienić wartości danych przekazanych jako właściwość do komponentu potomnego. Możemy jednak emitować zdarzenia i dołączać detektory zdarzeń do komponentu oraz zmieniać dane dotyczące zdarzenia. Nazwijmy właściwość przekazywaną przez coś, co nie jest nazwą, na przykład initialName. Następnie otwórzmy komponent Introduction w którym umieścimy funkcję danych, która powiąże obiekt name tego komponentu z initialValueprops. Tak więc skrypt komponentu Introduction.vue będzie wyglądał następująco:

(Wracamy do naszego projektu Zoo, przechodzimy do katalogu components i tworzymy plik Introduction.vue)

<script>  
*export default* {  
 props: ['initialName'],  
 data () {  
 *return* {  
 name: *this*.initialName  
 }  
 }  
}  
</script>

Przechodzimy do pliku App.vue i usuwamy div który zwierał pole do wpisywania imienia oraz ustalamy sposób w jaki wiążemy name do komponentu.

<template>  
 <div *id*="app" *class*="jumbotron">  
 <div *class*="row">  
 <div *class*="col-9"><h1><span *v-if*="name">Witaj, <strong>{{name}}</strong>! </span><span>Nauczmy się Vue.js</span></h1></div>  
 <div *class*="col-3"><img *width*="50%" *:src*="name ? img1 : img2"></div>  
 </div>  
 <hr />  
 <introduction *:initialName*="name"></introduction>  
 <hr />  
 <h2><span *v-if*="name">{{name}}! </span>Wybierz zwierzę </h2>  
 <div *class*="row">  
 <select *v-model*="animalsForZoo" *class*="col-12" *multiple*="multiple" *name*="animals" *id*="animals">  
 <option *v-for*="animal *in* animals" *v-bind:key*="animal" *:value*="animal">{{animal}}</option>  
 </select>  
 </div>  
 <hr />  
 <zoo *:animals*="animalsForZoo" *:animalsCodes*="animalsCodes" *:name*="name"></zoo>  
 </div>  
</template>

Teraz musimy zaimportować stworzony przez nas komponent Introduction.vue i dodać go do listy komponentów

<script>  
*import* Zoo *from* './components/Zoo'  
*import* Introduction *from* './components/Introduction'  
...  
...  
*export default* {  
 name: 'app',  
 components: {  
 Zoo,  
 Introduction  
 },  
 data () {  
 *return* data  
 }  
}  
</script>

Teraz musimy stworzyć szablon naszego komponentu, który będzie zawierał metodę, która będzie emitowała zmianę nazwy. Przechodzimy do pliku Introduction.vue i dodajemy sekcję template oraz metodę w sekcji script:

<template>  
 <div>  
 <label *for*="name" *:class*="{green: name, red: !name}">What's your name? </label>  
 <input *id*="name" *type*="text" *v-model.trim*="name" *@input*="onInput">  
 </div>  
</template>  
  
<script>  
*export default* {  
 props: ['initialName'],  
 data () {  
 *return* {  
 name: *this*.initialName  
 }  
 },  
 methods: {  
 onInput () {  
 *this*.$emit('nameChanged', *this*.name)  
 }  
 }  
}  
</script>

Ostatnią rzeczą którą musimy zrobić jest powiązanie detektora zdarzeń nameChange z komponentem Introduction. Dodatkowo wywołamy metodę, która zmieni name w obiekcie danych w pliku App.vue

<template>  
 <div *id*="app" *class*="jumbotron">  
 ...  
 <introduction *@nameChanged*="onNameChanged" *:initialName*="name"></introduction>  
 ...  
 </div>  
</template>  
  
<script>  
...  
*export default* {  
 name: 'app',  
 components: {  
 Zoo,  
 Introduction  
 },  
 data () {  
 *return* data  
 },  
 methods: {  
 onNameChanged (newName) {  
 *this*.name = newName  
 }  
 }  
}  
</script>  
...

Cóż, nie było to bardzo trudne, ale czy chcemy emitować te wszystkie zdarzenia za każdym razem, gdy musimy zaktualizować stan? A co, jeśli mamy komponenty wewnątrz komponentów? A co, jeśli mamy inne komponenty wewnątrz tych komponentów? Jeśli będziemy musieli coś zmienić, czy będziemy musieli iść do wszystkich tych komponentów? Czy nie byłoby lepiej mieć dane aplikacji w jakimś scentralizowanym magazynie, który zapewniłby prosty interfejs API do jego zarządzania, a następnie moglibyśmy po prostu wywołać metody tego magazynu, aby pobrać i zaktualizować dane? Właśnie po to jest Vuex! Vuex to scentralizowane zarządzanie stanem inspirowane Redux.

Oficjalna dokumentacja dostępna pod adresem <https://vuex.vuejs.org/guide/>

Krótko mówiąc, trzy najważniejsze części Vuex to state, getters i mutations:

• state: jest to początkowy stan aplikacji, czyli dane aplikacji

• getters: funkcje, które zwracają dane ze magazynu

• mutations: funkcje, które mogą modyfikować dane w magazynie

Przechodzimy do projektu zoo-vuex-start w którym instalujemy vuex za pomocą polecenia:

npm install vuex --save

Teraz stwórzmy nasz magazyn. Odpalamy projekt i przechodzimy do katalogu src w którym utworzymy folder store wraz z plikiem index.js. Wewnątrz pliku index.js umieszczamy następujący kod:

import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

Teraz tworzymy nową istancję Vuex, która powinna przechowywać state, getters i mutations.

*const* state = {  
}  
*const* getters = {  
}  
*const* mutations = {  
}  
*export default new* Vuex.Store({  
 state,  
 getters,  
 mutations  
})

Teraz przenosimy wszystkie dane z naszej aplikacji do magazynu:

*import* Vue *from* 'vue'  
*import* Vuex *from* 'vuex'  
  
Vue.use(Vuex)  
  
*const* animalsCodes = {  
 dog: '&#x1f436;',  
 cat: '&#x1f638;',  
 monkey: '&#x1f435;',  
 unicorn: '&#x1f984;',  
 tiger: '&#x1f42f;',  
 mouse: '&#x1f42d;',  
 rabbit: '&#x1f430;',  
 cow: '&#x1f42e;',  
 whale: '&#x1f433;',  
 horse: '&#x1f434;',  
 pig: '&#x1f437;',  
 frog: '&#x1f438;',  
 koala: '&#x1f43c;'  
}  
  
*const* animals = Object.keys(animalsCodes)  
*var* state = {  
 name: 'Mati',  
 animals,  
 animalsCodes,  
 animalsForZoo: [],  
 img1: '../../static/images/emotka1.png',  
 img2: '../../static/images/emotka2.png'  
}  
  
*const* getters = {  
  
}  
  
*const* mutations = {  
}  
*export default new* Vuex.Store({  
 state,  
 getters,  
 mutations  
})

Teraz przechodzimy do pliku main.js w którym importujemy nasz magazyn. Od teraz wszystkie komponenty i ich potomkowie będą mieli dostęp do obiektu this.$store

*import* Vue *from* 'vue'  
*import* App *from* './App'  
*import* router *from* './router'  
*import* store *from* './store'  
  
Vue.config.productionTip = *false  
  
/\* eslint-disable no-new \*/  
new* Vue({  
 el: '#app',  
 router,  
 components: { App },  
 template: '<App/>',  
 store  
})

Teraz zamieniamy wszystkie dane wartościami przechowywanymi w naszym magazynie

Przechodzimy do pliku App.vue w którym podmieniamy sekcję script

<script>  
*import* Zoo *from* './components/Zoo'  
*import* Introduction *from* './components/Introduction'  
  
*export default* {  
 name: 'app',  
 components: {  
 Zoo,  
 Introduction  
 },  
 data () {  
 *return* {  
 animalsForZoo: []  
 }  
 },  
 computed: {  
 name () {  
 *return this*.$store.state.name  
 },  
 animals () {  
 *return this*.$store.state.animals  
 },  
 animalsCodes () {  
 *return this*.$store.state.animalsCodes  
 },  
 img1 () {  
 *return this*.$store.state.img1  
 },  
 img2 () {  
 *return this*.$store.state.img2  
 }  
 },  
 methods: {  
 onNameChanged (newName) {  
 *this*.name = newName  
 }  
 }  
}  
</script>

Dodajemy nową mutację updateName w pliku index.js i przekazujemy newName jako jej drugi argument

*const* mutations = {  
 updateName (state, newName) {  
 state.name = newName  
 }  
}

Ostatnim krokiem jest zmiana metody onInput w pliku Introduction.vue (folder componentes)

<script>  
*export default* {  
 props: ['initialName'],  
 data () {  
 *return* {  
 name: *this*.initialName  
 }  
 },  
 methods: {  
 onInput (ev) {  
 *this*.$store.commit('updateName', ev.currentTarget.value)  
 }  
 }  
}  
</script>

**Bootstrap**

Krótko mówiąc, Bootstrap oferuje szeroki zestaw klas, które pozwalają zbudować prawie wszystko z dowolnym układem w łatwy i bezproblemowy sposób.

Bootstrap zapewnia cztery najważniejsze rzeczy

* Łatwe tworzenie layout’ów
* Szeroki zakres klas umożliwiających stylizację prawie każdego elementu
* Komponenty niezależne, takie jak alerts, budges, modals
* Niektóre narzędzia do stylizacji obrazów, rysunków, pozycjonowania, stylizacji i dodawania obramowań

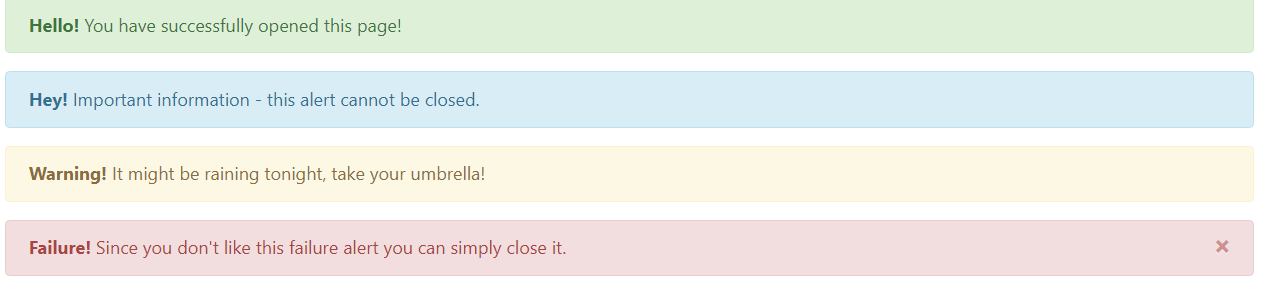
Jak zainstalować Bootstrap?

Możemy go zainstalować tak jak robiliśmy to na początku prezentacji, korzystając z CDN

<https://getbootstrap.com/docs/4.4/getting-started/introduction/>

**Bootstrap Components**

Bootstrap ma wiele komponentów, które można wykorzystać w naszych projektach. Przyjrzyjmy się niektórym z nich. Zaczniemy od alertów. Alerty to okienka które pojawiają się w oknie przeglądarki po wywołaniu jakiegoś zdarzenia. Gdybyś chciał sam utworzyć taki alert prawdopodobnie utworzyłbyś div, dodał do niego jakąś klasę i dodał trochę JavaScript, który usunąłby element z drzewa DOM po okresie karencji. Używając Bootstrap, po prostu dodajesz klasę alertów do swojego div i dodajesz kolejną klasę, taką jak alert-warning lub alert-info, aby określić, jaki to rodzaj alertu

Przykład: Bootsrtap/ex1\_alerts

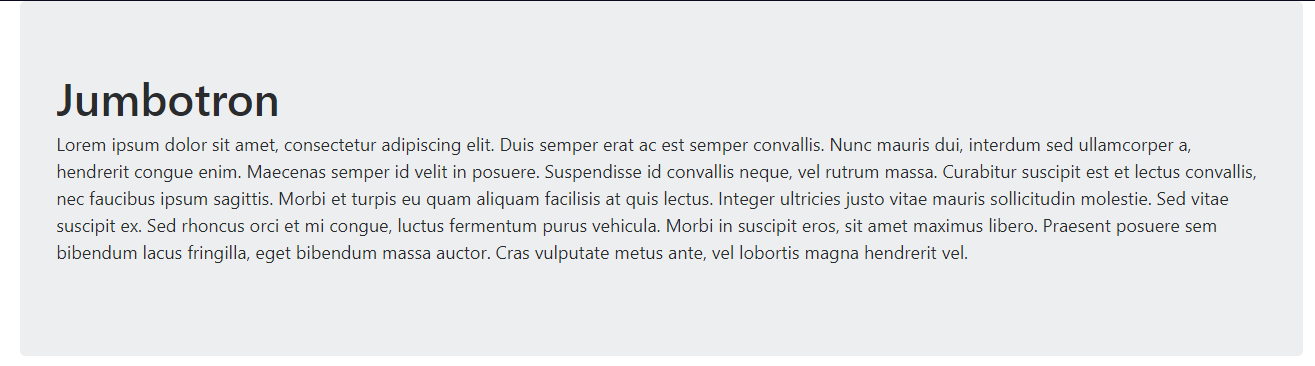
Nawet prosty element, taki jak przycisk, może być stylizowany na setki różnych sposobów za pomocą Bootstrap. Ponownie możesz mieć przyciski wskazujące sukces, strefę zagrożenia, informacyjne lub po prostu szare. Istnieje możliwość grupowania przycisków i nadawania im wyglądu linku.

Przykład: Bootsrtap/ex2\_buttons



Jumbotron tworzy szary prostokąt w którym możemy zawrzeć ważne informacje, wycentrować je, dołożyć przycisk itp. Tworzy się go poprzez dodanie do div’a klasy jumbotron.

Przykład: Bootsrtap/ex3\_jumbotron



**Bootstrap layout**

System siatki Bootstrap jest dość potężny i łatwy do zrozumienia. Składa się z wierszy i kolumn. Wszystko jest kontrolowane przez klasy, które mają dość samoopisowe nazwy, takie jak row i col. Jeśli nadasz swoim kolumnom klasę col, każda kolumna wewnątrz elementu row będzie miała ten sam rozmiar. Jeśli chcesz mieć kolumny o różnych rozmiarach, pamiętaj , że rząd może składać się z 12 kolumn. Więc jeśli chcesz utworzyć jakieś kolumny, powiedzmy na połowę twojego wiersza, nadaj mu klasę col-6.

Przykład: Bootsrtap/ex4\_grid

