

VUE.JS FRAMEWORK

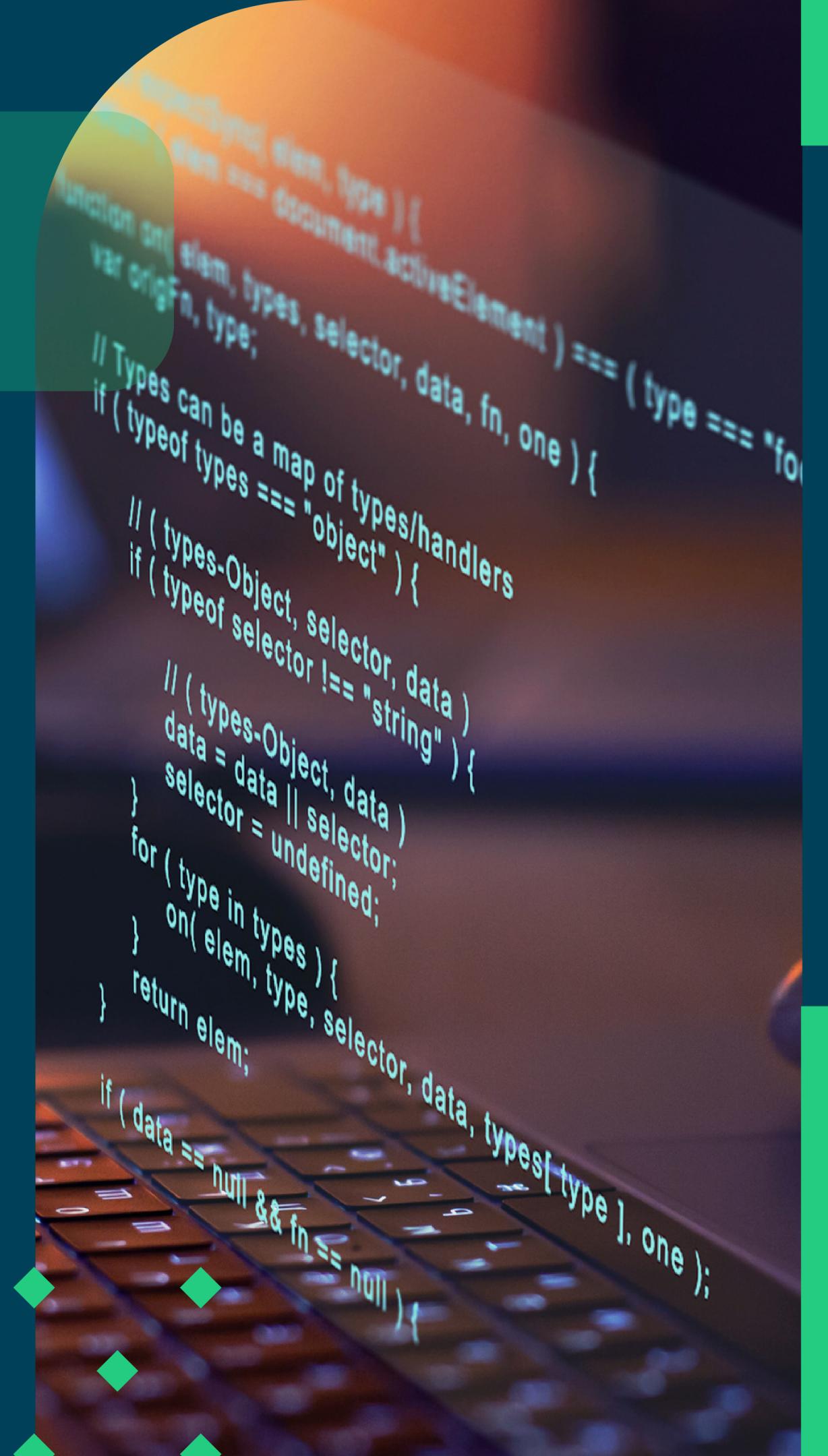
Adrian Szlachta



Co to jest Vue?

Vue.js to progresywny framework JavaScript używany do budowy interfejsów użytkownika oraz aplikacji jednostronicowych.

Zapewnia łatwą integrację dzięki swojej minimalistycznej architekturze, elastyczności i wydajności, co pozwala programistom efektywnie tworzyć dynamiczne aplikacje internetowe. Vue.js opiera się na konceptach reaktywności danych i komponentów, umożliwiając szybkie tworzenie interaktywnych UI.



Zalety i wady

01

Łatwość nauki i zrozumienia:

Dzięki prostocie składni i intuicyjnej dokumentacji Vue.js jest łatwy dla początkujących programistów, umożliwiając im szybkie rozpoczęcie pracy nad projektami.

02

Efektywność i wydajność:

Vue.js oferuje szybkie renderowanie interfejsów użytkownika dzięki swojemu lekkiemu rozmiarowi i zintegrowanemu systemowi reaktywności danych, co przekłada się na płynne działanie aplikacji.

03

Elastyczność i modułowość:

Vue.js umożliwia budowę aplikacji zarówno małych prototypów, jak i dużych, skalowalnych projektów, dzięki czemu jest odpowiedni do różnorodnych zastosowań.

01

Mniejsza popularność w porównaniu z innymi frameworkami:

Pomimo wzrostu popularności, Vue.js nadal pozostaje mniej popularny od niektórych konkurencyjnych frameworków, co może skutkować mniejszą dostępnością zasobów i narzędzi.

02

Brak natywnego wsparcia od niektórych firm i organizacji:

W przeciwieństwie do niektórych innych frameworków, Vue.js nie cieszy się wsparciem niektórych dużych firm, co może być ważne dla niektórych projektów.

03

Nadmiar elastyczności może prowadzić do chaosu w kodzie:

Chociaż elastyczność Vue.js może być zaleta, może też prowadzić do chaotycznego kodu w przypadku braku jasnych wytycznych dla programistów.



04

Aktywna społeczność i wsparcie:

Istnieje duże wsparcie społecznościowe w postaci aktywnych forum, bibliotek i narzędzi wspierających rozwój aplikacji opartych na Vue.js.

05

Łatwa integracja:

Vue.js może być łatwo zintegrowany z istniejącymi projektami, nawet w przypadku aplikacji opartych na innych frameworkach, co ułatwia migrację lub współpracę między zespołami.

06

Dobra dokumentacja:

Vue.js posiada obszerną, przejrzystą i aktualną dokumentację, co ułatwia programistom znalezienie potrzebnych informacji i rozwiązywanie problemów.

04

Mniejsza ilość gotowych komponentów w porównaniu z innymi frameworkami:

W porównaniu z niektórymi innymi frameworkami, Vue.js może mieć mniejszą liczbę gotowych komponentów, co może wymagać większego nakładu pracy przy tworzeniu niestandardowych rozwiązań.

05

Ryzyko zależności od jednego dostawcy:

Korzystanie wyłącznie z jednego frameworka, takiego jak Vue.js, może zwiększać ryzyko zależności od jednego dostawcy, co może wpływać na przyszłą rozbudowę aplikacji.

06

Wymagania dotyczące doświadczenia w obszarze reaktywności danych:

Chociaż Vue.js jest stosunkowo łatwy do nauki, wymaga pewnego poziomu zrozumienia reaktywności danych, co może stanowić wyzwanie dla początkujących programistów.

Vue vs inne frameworki

Temat	Vue.js	React	Angular
Łatwość nauki i zrozumienia	Prosta i przejrzysta składnia, łatwa dla początkujących.	Wymaga bardzo dobrego znajomości js oraz jego mechanizmów, duże community daje duże wsparcie.	Bardziej złożona składnia i struktura, jednakowa dla większych zespołów i projektów.
Wykorzystanie	Do małych i start-upowych projektów.	Duże i średnie projekty.	Duże i średnie projekty.
Zachowanie jakości kodu	Przy zachowaniu wzorców z dokumentacji łatwe.	Ze względu na swoją elastyczność w doborze bibliotek oraz struktury w Reacie łatwo jest zrobić bałagan.	Wymusza pewne struktury i wykorzystanie wewnętrznych narzędzi co powoduje łatwe utrzymanie kodu.



Temat	Vue.js	React	Angular
Pobieranie danych	Axios, fetch oraz działanie na Promisach.	Axios, fetch oraz działanie na Promisach.	Wykorzystanie HttpClient oraz obiektów Observera.
Wirtualne drzewo DOM	Ma wirtualne drzewo DOM, które pomaga w efektywnym renderowaniu zmian i minimalizacji operacji na rzeczywistym drzewie DOM	Wykorzystuje wirtualny DOM do porównywania stanu poprzedniego i obecnego, co pozwala na minimalną manipulację rzeczywistym drzewem DOM.	Korzysta z wirtualnego DOM w celu zoptymalizowania renderowania i poprawy wydajności.

Propsy

Propsy w Vue.js są mechanizmem, który umożliwia komunikację między komponentami poprzez przekazywanie danych z komponentu rodzica do komponentu potomnego. Pozwalają one na dynamiczne przekazywanie danych, co jest kluczowe dla budowania modularnych i reużywalnych komponentów.

Rodzaje propsów:

- **Standardowe propsy:** Są to wartości przekazywane z komponentu nadrzędnego do komponentu potomnego poprzez atrybuty HTML, np. `<my-component :propName="value"></my-component>`.
- **Dynamiczne propsy:** Pozwalają na przekazywanie dynamicznych wartości do komponentu potomnego, co oznacza, że dane te mogą być zmienną wartością, obiektem, tablicą itp.



Walidacja propsów:

Vue.js umożliwia walidację propsów, co pozwala na sprawdzenie, czy przekazane dane spełniają określone kryteria. Można to osiągnąć poprzez zdefiniowanie obiektu props z właściwościami zawierającymi nazwy propsów i ich typy, np.:

```
javascript
props: {
  propName: {
    type: String,
    required: true,
    default: 'default value',
    validator: function(value) {
      // Custom validation logic
      return value.length > 0;
    }
}
```

Typowanie propsów:

W Vue.js można określić typy danych, które propsy powinny przyjmować.

Dostępne typy to:

- String
- Number
- Boolean
- Array
- Object
- Function
- Symbol



```
else {  
    array();  
    i_fetch_assoc($result);  
    $row = $row['Correct'];  
    'A'] = $row['Anum'];  
    'B'] = $row['Bnum'];  
    'C'] = $row['Cnum'];  
    'D'] = $row['Dnum'];  
    'Correct'] = $correctAnswer;  
    'Answer'] = rtrim($row[$correctAnswer], ".");  
    'Query'] = "SELECT * FROM TechTerms WHERE Date='".date("Y-m-d")."';  
    $stArray;  
  
    ['Error'] = 'Quiz load query failed';  
    $stArray:
```



Vuex

Vuex to biblioteka stanu dla aplikacji Vue.js, która pomaga w zarządzaniu globalnym stanem aplikacji. Używa wzorca zarządzania stanem "flux", co ułatwia przewidywanie i śledzenie zmian w aplikacji. Vuex przechowuje dane w jednym globalnym obiekcie stanu, który jest dostępny dla wszystkich komponentów w aplikacji.

Przykład:

Załóżmy, że mamy prostą aplikację, która wyświetla licznik, a użytkownik może zwiększać jego wartość klikając na przycisk. W takim przypadku Vuex może być wykorzystany do przechowywania aktualnej wartości licznika.

javascript

```
// store.js
import Vuex from 'vuex'
import Vue from 'vue'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    counter: 0
  },
  mutations: {
    increment(state) {
      state.counter++
    }
  },
  actions: {
    increment(context) {
      context.commit('increment')
    }
  },
  getters: {
    getCounter(state) {
      return state.counter
    }
  }
})
```

html

```
<!-- App.vue -->
<template>
  <div>
    <p>Counter: {{ counter }}</p>
    <button @click="incrementCounter">Increment</button>
  </div>
</template>

<script>
  import { mapGetters, mapActions } from 'vuex'

  export default {
    computed: {
      ...mapGetters(['getCounter'])
    },
    methods: {
      ...mapActions(['increment']),
      incrementCounter() {
        this.increment()
      }
    }
  }
</script>
```

W tym przykładzie Vuex przechowuje stan licznika, a komponent App.vue korzysta z gettera getCounter do wyświetlenia aktualnej wartości oraz akcji increment do zwiększania licznika. Dzięki temu stan licznika jest globalny i dostępny dla wszystkich komponentów w aplikacji.

Przekazywanie danych za pomocą propsów oraz vuex

PROPS:

Komponent rodzica (ParentComponent.vue):

```
vue

<template>
  <div>
    <p>Parent Message: {{ parentMessage }}</p>
    <input v-model="parentMessage" type="text" placeholder="Edit message" />
    <ChildComponent :message="parentMessage" />
  </div>
</template>

<script>
import ChildComponent from './ChildComponent.vue';

export default {
  components: {
    ChildComponent
  },
  data() {
    return {
      parentMessage: 'Hello from parent component'
    };
  }
};
</script>
```

Komponent potomny (ChildComponent.vue):

```
vue

<template>
  <div>
    <p>Child Message: {{ message }}</p>
  </div>
</template>

<script>
export default {
  props: {
    message: String
  }
};
</script>
```

VUEX - store.js

```
javascript

import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    message: 'Hello from Vuex store'
  },
  mutations: {
    updateMessage(state, newMessage) {
      state.message = newMessage;
    }
  }
});
```

Komponent (ParentComponent.vue):

```
vue

<template>
  <div>
    <p>Message: {{ message }}</p>
    <input v-model="updatedMessage" type="text" placeholder="Update Message">
    <button @click="updateMessage">Update Message</button>
  </div>
</template>

<script>
import { mapState, mapMutations } from 'vuex';

export default {
  computed: {
    ...mapState(['message'])
  },
  data() {
    return {
      updatedMessage: ''
    };
  },
  methods: {
    ...mapMutations(['updateMessage'])
  }
};
</script>
```

Komponent (ChildComponent.vue):

```
vue

<template>
  <div>
    <p>{{ message }}</p>
  </div>
</template>

<script>
import { mapState } from 'vuex';

export default {
  computed: {
    ...mapState(['message'])
  }
};
</script>
```

Change detection

Mechanizm change detection w Vue.js polega na automatycznym wykrywaniu zmian w danych komponentu i aktualizacji widoku w przypadku ich modyfikacji. Vue.js śledzi zależności między komponentami a danymi, co pozwala na efektywne renderowanie tylko tych części widoku, które wymagają aktualizacji. Jednakże Vue.js ma problem z wykrywaniem zmian w tablicach, szczególnie w przypadku zmian dokonywanych przy użyciu indeksów lub metody length.

Aby obejść ten problem, można zastosować następujące rozwiązania:

- Zamiast zmieniać tablicę w miejscu, można użyć metod reaktywnych udostępnianych przez Vue.js, takich jak push(), pop(), splice() itp., które informują framework o zmianach.
- Można również zastosować metodę Vue.set() (lub this.\$set() w komponentach) do dodawania nowych właściwości reaktywnych do obiektów, co umożliwia śledzenie zmian w tablicach.
- Innym rozwiązaniem jest tworzenie kopii tablicy i modyfikowanie jej, co pozwala uniknąć problemów związanych z bezpośrednią modyfikacją oryginalnej tablicy.

Change detection

```
javascript

export default {
  data() {
    return {
      nestedObject: {
        property1: 'value1',
        property2: 'value2'
      }
    };
  },
  watch: {
    nestedObject: {
      handler(newValue, oldValue) {
        console.log('Zmiana w obiekcie zagnieżdżonym', newValue, oldValue);
      },
      deep: true // Ustawienie obserwacji głębokiej
    }
  }
};
```

- Jednym ze sposobów wykrywania zmian jest również stosowanie watch na obiektach oraz zastosowanie opcji deep.

Dziękuję
za uwagę

