Przewodnik po Vue.js

Kacper Renkel

August 11, 2024

Contents

1 Wprowadzenie

Vue.js to progresywny framework JavaScriptowy do budowania interfejsów użytkownika. Zapewnia elastyczność oraz możliwość integracji z różnymi projektami, co czyni go popularnym wyborem wśród deweloperów.

2 Cykl życia komponentów

Cykl życia komponentu składa sie z kilku etapów:

- **beforeCreate** - Inicjalizacja komponentu, brak dostepu do danych i metod.
- **created** - Dostep do danych i metod, komponent nie jest jeszcze zamontowany na DOM.
- **beforeMount** - Komponent jest gotowy do zamontowania na DOM.
- **mounted** - Komponent został zamontowany na DOM.
- **beforeUpdate** - Przed aktualizacja komponentu w odpowiedzi na zmiany w danych.
- **updated** - Po aktualizacji komponentu na DOM.
- **beforeUnmount** - Przed usunieciem komponentu z DOM.
- **unmounted** - Komponent został usuniety z DOM.

```
export default {
  data() {
    return {
      message: 'Hello Vue!'
    }
  },
  created() {
    console.log('Komponent zosta  utworzony');
  },
  mounted() {
    console.log('Komponent zosta  zamontowany na DOM');
  }
}
```

Listing 1: Przykład użycia cyklu życia komponentu

3 Computed Properties i Watch

3.1 Computed Properties

Computed properties to właściwości, które sa obliczane na podstawie innych danych. Vue.js buforuje wyniki obliczeń, wiec sa one aktualizowane tylko wtedy, gdy zależne dane ulegaja zmianie.

```
export default {
  data() {
    return {
      firstName: 'John',
      lastName: 'Doe'
    }
  },
  computed: {
    fullName() {
      return this.firstName + ' ' + this.lastName;
    }
  }
}
```

Listing 2: Przykład computed properties

3.2 Watch

Watch to mechanizm obserwujacy zmiany w danych i uruchamiajacy określona funkcje w odpowiedzi na te zmiany. Jest użyteczny, gdy potrzebujemy wykonać operacje asynchroniczne lub kosztowne obliczenia.

```
export default {
  data() {
    return {
      query: ''
    }
  },
  watch: {
    query(newValue, oldValue) {
      this.fetchResults(newValue);
    }
  },
  methods: {
    fetchResults(query) {
      // Logika do pobierania wynik w wyszukiwania
    }
  }
}
```

Listing 3: Przykład watch

4 Dyrektywy

Dyrektywy w Vue.js umożliwiaja manipulacje DOM poprzez specjalne atrybuty. Poniżej znajduja sie przykłady popularnych dyrektyw.

- `v-bind` - Służy do dynamicznego przypisania atrybutów lub właściwości do elementów.
- `v-if` / `v-else-if` / `v-else` - Renderowanie warunkowe.
- `v-for` - Iteracja po kolekcji danych.
- `v-on` - Obsługa zdarzeń.
- `v-model` - Dwukierunkowe wiazanie danych.

```
<div v-if="isVisible">Widoczne</div>
<ul>
  <li v-for="item in items" :key="item.id">{{ item.name }}</
     li>
</ul>
<button @click="handleClick">Kliknij mnie</button>
<input v-model="message" placeholder="Wpisz wiadomo  ">
```

Listing 4: Przykład dyrektyw w Vue.js

5 Przekazywanie danych miedzy komponentami

5.1 Od rodzica do dziecka (Props)

Dane moga być przekazywane od komponentu rodzica do dziecka poprzez propsy.

```
<!-- Komponent rodzic -->
<template>
  <ChildComponent :message="parentMessage" />
</template>

<script>
export default {
  data() {
    return {
      parentMessage: 'Hello from parent'
    }
  }
}
</script>

<!-- Komponent dziecko -->
<template>
  <p>{{ message }}</p>
</template>

<script>
export default {
```

```
  props: ['message']
}
</script>
```

Listing 5: Przekazywanie danych przez props

5.2 Od dziecka do rodzica (Emitowanie zdarzeń)

Dziecko może komunikować sie z rodzicem poprzez emitowanie zdarzeń.

```
<!-- Komponent dziecko -->
<template>
  <button @click="sendMessage">Send Message</button>
</template>

<script>
export default {
  methods: {
    sendMessage() {
      this.$emit('messageSent', 'Hello from child');
    }
  }
}
```

Listing 6: Emitowanie zdarzenia przez dziecko

¡!– Komponent rodzic –¿ ¡template¿ ¡ChildComponent @messageSent=”handleMessage” /¿ ¡/template¿
¡script¿ export default methods: handleMessage(msg) console.log(msg); ¡/script¿

6 Sloty

Sloty umożliwiaja przekazywanie zawartości HTML z komponentu rodzica do dziecka, co pozwala na wieksza elastyczność w tworzeniu komponentów wielokrotnego użytku.

```
<!-- Komponent dziecko -->
<template>
  <div class="card">
    <slot></slot> <!-- Standardowy slot -->
  </div>
</template>

<!-- Komponent rodzic -->
<template>
  <CardComponent>
    <p>Tre przekazana przez slot</p>
  </CardComponent>
</template>
```

Listing 7: Przykład użycia slotów

7 Mixiny

Mixiny to mechanizm pozwalajacy na współdzielenie funkcjonalności pomiedzy różnymi komponentami. Umożliwiaja one współdzielenie metod, danych, oraz innych opcji.

```
export const myMixin = {
  data() {
    return {
      mixinData: 'To jest dane z mixinu'
    }
  },
  methods: {
    mixinMethod() {
      console.log('To jest metoda z mixinu');
    }
  }
}

export default {
  mixins: [myMixin],
  mounted() {
    console.log(this.mixinData); // 'To jest dane z mixinu'
    this.mixinMethod(); // 'To jest metoda z mixinu'
  }
}
```

Listing 8: Przykład użycia mixinów

8 Provide/Inject

Mechanizm `provide/inject` pozwala na przekazywanie danych z komponentu rodzica do dziecka bez konieczności przekazywania ich przez każdy pośredni komponent (prop drilling).

```
<!-- Komponent rodzic -->
<template>
  <ChildComponent />
</template>

<script>
export default {
  provide() {
    return {
      message: 'Hello from parent'
    }
  }
}
</script>
```

```
<!-- Komponent dziecko -->
<template>
  <p>{{ message }}</p>
</template>

<script>
export default {
  inject: ['message']
}
</script>
```

Listing 9: Przykład użycia provide/inject

9 Dynamiczne komponenty

Vue.js umożliwia dynamiczne ładowanie i renderowanie komponentów w oparciu o zmienne.

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

<script>
export default {
  data() {
    return {
      currentComponent: 'ComponentA'
    }
  },
  components: {
    ComponentA,
    ComponentB
  }
}
</script>
```

Listing 10: Przykład dynamicznych komponentów

10 `script setup` vs tradycyjny `script`

Vue 3 wprowadził nowa składnie `script setup`, która upraszcza kod komponentu.

- `script setup` - Lepsza wydajność, bardziej zwarta składnia, brak potrzeby jawnej deklaracji zwracanych danych.
- **Tradycyjny** `script` - Wieksza elastyczność, szczególnie w skomplikowanych komponentach, lepsza czytelność dla wiekszych zespołów.

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

<script setup>
import { ref } from 'vue'

const message = ref('Hello from script setup')
</script>
```

Listing 11: Przykład użycia `script setup`

W przypadku prostych komponentów i projektów `script setup` jest zalecany ze względu na prostote i wydajność. W bardziej złożonych komponentach, szczególnie gdy potrzebujemy pełnej kontroli nad cyklem życia komponentu, tradycyjny `script` może być lepszym wyborem.

11 Podsumowanie

Vue.js oferuje szeroki wachlarz narzedzi do tworzenia dynamicznych, interaktywnych interfejsów użytkownika. Wybór odpowiednich technik i narzedzi zależy od specyficznych wymagań projektu oraz preferencji zespołu.