Notatki z Vue.js: Stan Globalny, Routing, Binding i Inne Funkcje

Kacper Renkel

August 23, 2024

Contents

1 Wprowadzenie do Vue.js **3**

2 Zarzadzanie Stanem Globalnym z Pinia **3**
2.1 Instalacja Pinia . . . 3
2.2 Konfiguracja Pinia . . . 3
2.3 Tworzenie Store'u w Pinia . . . 3
2.4 Użycie Store'u w Komponentach . . . 4

3 Routing z Vue Router **4**
3.1 Instalacja Vue Router . . . 4
3.2 Konfiguracja Routera . . . 4
3.3 Użycie Routera w Aplikacji . . . 5
3.4 Dynamiczne Trasy . . . 5

4 Binding Danych w Vue.js **5**
4.1 Dwukierunkowe Wiazanie Danych (v-model) . . . 5
4.2 Binding Atrybutów . . . 6
4.3 Binding Klas i Stylów . . . 6

5 Reactive i Ref **6**
5.1 Reactive . . . 6
5.2 Ref . . . 6

6 Odczytywanie Danych z Parametrów URL **7**
6.1 Parametry Dynamiczne w URL . . . 7

7 Przekazywanie Parametrów miedzy Komponentami **7**
7.1 Przekazywanie Parametrów z Rodzica do Dziecka . . . 7
7.2 Przekazywanie Parametrów z Dziecka do Rodzica . . . 8

8 Przekazywanie Danych z Dziecka do Rodzica za pomoca `v-model` **8**
8.1 Przekazywanie Danych z Dziecka do Rodzica za pomoca `v-model` i `@update:value` (Tradycyjny Sposób) . . . 8
8.1.1 Komponent Dziecka . . . 8
8.1.2 Komponent Rodzica . . . 9

8.2 Przekazywanie Danych z Dziecka do Rodzica za pomoca `v-model` i `@update:value` z `<script setup>` . . . 9
8.2.1 Komponent Dziecka . . . 9
8.2.2 Komponent Rodzica . . . 10

9 Composables i Serwisy **11**
9.1 Composables . . . 11
9.2 Serwisy . . . 11

10 Obsługa Formularzy i Walidacja **12**
10.1 Tworzenie Formularzy . . . 12
10.2 Walidacja Formularzy . . . 13

11 Obsługa Zdarzeń Globalnych **13**
11.1 Emitowanie i Nasłuchiwanie Zdarzeń . . . 13
11.2 Globalny Bus Zdarzeń . . . 14

12 Lazy Loading Komponentów **14**
12.1 Dynamiczny Import . . . 14
12.2 Użycie `defineAsyncComponent` . . . 14

13 Middleware i Ochrona Tras w Vue Router **15**
13.1 Przykład Middleware Autoryzacji . . . 15
13.2 Ochrona Tras . . . 15

14 Praca z Animacjami i Przejściami **15**
14.1 Typy klas w `transition` . . . 15
14.2 Użycie `transition` . . . 16
14.3 Animacje w Grupach . . . 17

15 Testowanie w Vue.js **17**
15.1 Proste Testy Jednostkowe . . . 17
15.2 Testowanie Emitowania Zdarzeń . . . 17

16 Optymalizacja Aplikacji Vue.js **18**
16.1 Minimalizacja Renderów . . . 18
16.2 Memoizacja . . . 18

17 Obsługa Modułów Vuex **18**
17.1 Tworzenie Modułów . . . 18
17.2 Dostep do Modułów w Komponentach . . . 19

18 Integracja z API i Autoryzacja **19**
18.1 Wysyłanie Żadań HTTP z Axios . . . 19
18.2 Zarzadzanie Tokenami Autoryzacji . . . 19

19 Podsumowanie **20**

1 Wprowadzenie do Vue.js

Vue.js to jeden z najpopularniejszych frameworków JavaScript używanych do budowania nowoczesnych aplikacji webowych. Jego prostota, wydajność oraz modularność sprawiaja, że jest wybierany zarówno przez poczatkujacych, jak i zaawansowanych deweloperów.

2 Zarzadzanie Stanem Globalnym z Pinia

Pinia to oficjalny menedżer stanu dla Vue.js, który jest bardziej nowoczesnym i uproszczonym narzedziem w porównaniu do Vuex.

2.1 Instalacja Pinia

Aby zainstalować Pinia w projekcie Vue.js, należy użyć nastepujacej komendy:

```
npm install pinia
```

2.2 Konfiguracja Pinia

Aby rozpocząć korzystanie z Pinia, należy dodać ja do aplikacji:

```
import { createApp } from 'vue'
import { createPinia } from 'pinia'
import App from './App.vue'

const app = createApp(App)
app.use(createPinia())
app.mount('#app')
```

2.3 Tworzenie Store'u w Pinia

Store w Pinia jest miejscem, gdzie można przechowywać globalny stan aplikacji.

```
import { defineStore } from 'pinia';

export const useMainStore = defineStore('main', {
    state: () => ({
        counter: 0,
        user: null
    }),
    actions: {
        increment() {
            this.counter++;
        },
        setUser(user) {
            this.user = user;
        }
    },
    getters: {
```

```
        doubleCounter(state) {
            return state.counter * 2;
        }
    }
});
```

2.4 Użycie Store'u w Komponentach

```
import { useMainStore } from './store/main';

export default {
  setup() {
    const mainStore = useMainStore();

    function incrementCounter() {
        mainStore.increment();
    }

    return { mainStore, incrementCounter };
  }
}
```

3 Routing z Vue Router

Vue Router to oficjalna biblioteka do zarzadzania routingiem w aplikacjach Vue.js.

3.1 Instalacja Vue Router

```
npm install vue-router
```

3.2 Konfiguracja Routera

Po zainstalowaniu, należy skonfigurować router:

```
import { createRouter, createWebHistory } from 'vue-router';
import Home from './views/Home.vue';
import About from './views/About.vue';

const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About }
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});
```

```
export default router;
```

3.3 Użycie Routera w Aplikacji

Nastepnie router musi zostać dodany do instancji Vue:

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App)
  .use(router)
  .mount('#app');
```

3.4 Dynamiczne Trasy

Vue Router pozwala na definiowanie tras dynamicznych z wykorzystaniem parametrów:

```
const routes = [
  { path: '/user/:id', component: User }
];
```

W komponencie, możemy odczytać parametry trasy:

```
import { useRoute } from 'vue-router';

export default {
  setup() {
    const route = useRoute();
    const userId = route.params.id;

    return { userId };
  }
}
```

4 Binding Danych w Vue.js

Binding danych to jedna z kluczowych funkcji Vue.js, umożliwiajaca łatwe powiazanie danych w aplikacji z widokiem.

4.1 Dwukierunkowe Wiazanie Danych (v-model)

```
<input v-model="message" placeholder="Wpisz wiadomo ">
<p>Wiadomo : {{ message }}</p>
```

Dzieki temu, każde wpisanie tekstu do pola input automatycznie aktualizuje wartość ‘message‘.

4.2 Binding Atrybutów

W Vue możemy dynamicznie wiazać atrybuty HTML z danymi w komponencie:

```
<img :src="imageSrc" :alt="imageDescription">
```

4.3 Binding Klas i Stylów

Możemy dynamicznie przypisywać klasy i style do elementów:

```
<div :class="{ active: isActive, 'text-bold': isBold }"></div>
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

5 Reactive i Ref

5.1 Reactive

Funkcja ‘reactive‘ tworzy reaktywny obiekt, co oznacza, że każda zmiana w obiekcie automatycznie aktualizuje widok:

```
import { reactive } from 'vue';

export default {
  setup() {
    const state = reactive({
      count: 0,
      user: { name: 'John Doe' }
    });

    function increment() {
      state.count++;
    }

    return { state, increment };
  }
}
```

5.2 Ref

Funkcja ‘ref‘ tworzy reaktywna referencje, która może przechowywać wartości prymitywne lub obiekty:

```
import { ref } from 'vue';

export default {
  setup() {
    const count = ref(0);

    function increment() {
      count.value++;
```

```
    }

    return { count, increment };
  }
}
```

6 Odczytywanie Danych z Parametrów URL

6.1 Parametry Dynamiczne w URL

W Vue Router można odczytywać parametry dynamiczne z URL:

```
const routes = [
  { path: '/profile/:id', component: Profile }
];

import { useRoute } from 'vue-router';

export default {
  setup() {
    const route = useRoute();
    const profileId = route.params.id;

    return { profileId };
  }
}
```

7 Przekazywanie Parametrów miedzy Komponentami

7.1 Przekazywanie Parametrów z Rodzica do Dziecka

Komponenty moga przekazywać dane do swoich komponentów potomnych za pomoca atrybutów props.

```
export default {
  props: {
    message: String,
    count: Number,
    user: Object
  }
}
```

Użycie komponentu dziecka w komponencie rodzica:

```
<ChildComponent :message="parentMessage" :count="parentCount" :user="parent
```

7.2 Przekazywanie Parametrów z Dziecka do Rodzica

Aby komponent dziecka mógł przekazać dane z powrotem do rodzica, można użyć emitowania zdarzeń:

```
export default {
  setup(props, { emit }) {
    function sendMessageToParent() {
      emit('messageFromChild', 'Hello from Child');
    }

    return { sendMessageToParent };
  }
}
```

W komponencie rodzica:

```
<ChildComponent @messageFromChild="handleMessage"></ChildComponent>
```

I metoda 'handleMessage' w komponencie rodzica:

```
export default {
  setup() {
    function handleMessage(message) {
      console.log('Received from child:', message);
    }

    return { handleMessage };
  }
}
```

8 Przekazywanie Danych z Dziecka do Rodzica za pomoca `v-model`

8.1 Przekazywanie Danych z Dziecka do Rodzica za pomoca `v-model` i `@update:value` (Tradycyjny Sposób)

8.1.1 Komponent Dziecka

Najpierw tworzymy komponent dziecka, który bedzie emitował wartość do rodzica za pomoca `v-model`.

```
<template>
  <input :value="modelValue" @input="updateValue">
</template>

<script>
export default {
  props: {
    modelValue: String
  },
```

```
  methods: {
    updateValue(event) {
      this.$emit('update:modelValue', event.target.value);
    }
  }
};
</script>
```

W powyższym kodzie:

- `props` zawiera `modelValue`, co jest standardowym schematem dla `v-model`.
- `updateValue` to metoda, która emituje zaktualizowana wartość do rodzica.

8.1.2 Komponent Rodzica

Teraz zdefiniujemy komponent rodzica, który bedzie używał komponentu dziecka z `v-model`.

```
<template>
  <div>
    <ChildComponent v-model="parentMessage" />
    <p>Message from child: {{ parentMessage }}</p>
  </div>
</template>

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

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

W tym przykładzie:

- `v-model` w rodzicu automatycznie zarzadza danymi `parentMessage`, synchronizujac je z wartościa z komponentu dziecka.

8.2 Przekazywanie Danych z Dziecka do Rodzica za pomoca `v-model` i `@update:value` z `<script setup>`

8.2.1 Komponent Dziecka

Przy użyciu `<script setup>`, możemy uprościć kod i nadal emitować dane z dziecka do rodzica.

```
<template>
  <input :value="modelValue" @input="updateValue">
</template>

<script setup>
import { defineProps, defineEmits } from 'vue';

const props = defineProps({
  modelValue: String
});

const emit = defineEmits(['update:modelValue']);

function updateValue(event) {
  emit('update:modelValue', event.target.value);
}
</script>
```

Tutaj:

- `defineProps` definiuje `modelValue` jako wartość przekazywana z rodzica.
- `defineEmits` definiuje zdarzenie `update:modelValue`, które jest emitowane, gdy wartość sie zmienia.

8.2.2 Komponent Rodzica

Komponent rodzica przy użyciu `<script setup>`:

```
<template>
  <div>
    <ChildComponent v-model="parentMessage" />
    <p>Message from child: {{ parentMessage }}</p>
  </div>
</template>

<script setup>
import { ref } from 'vue';
import ChildComponent from './ChildComponent.vue';

const parentMessage = ref('');
</script>
```

W tym przypadku:

- `ref` jest używany do zdefiniowania reaktywnej zmiennej `parentMessage`, która bedzie aktualizowana przez dziecko.

9 Composables i Serwisy

9.1 Composables

Composables to funkcje, które można użyć w wielu komponentach w celu zredukowania powtarzalności kodu. Sa to funkcje, które korzystaja z reaktywności i innych funkcji Vue.

Przykład prostego composable:

```
import { ref } from 'vue';

export function useCounter() {
  const count = ref(0);

  function increment() {
    count.value++;
  }

  return { count, increment };
}
```

Użycie composable w komponencie:

```
import { useCounter } from './composables/useCounter';

export default {
  setup() {
    const { count, increment } = useCounter();

    return { count, increment };
  }
}
```

9.2 Serwisy

Serwisy w Vue.js to pliki, które przechowuja logike zwiazana z interakcjami z zewnetrznymi API. Zazwyczaj wykorzystuja ‘axios‘ lub inna biblioteke do wykonywania żadań HTTP.

Przykład serwisu:

```
import axios from 'axios';

export function fetchUser(userId) {
  return axios.get(`/api/users/${userId}`);
}
```

Użycie serwisu w komponencie:

```
import { fetchUser } from './services/userService';

export default {
  setup() {
    const userId = ref(1);
    const user = ref(null);
```

```
    async function loadUser() {
      const response = await fetchUser(userId.value);
      user.value = response.data;
    }

    loadUser();

    return { user };
  }
}
```

10 Obsługa Formularzy i Walidacja

Formularze sa integralna cześcia wiekszości aplikacji webowych. Vue.js ułatwia tworzenie i zarzadzanie formularzami, a także walidacje danych.

10.1 Tworzenie Formularzy

Formularze w Vue.js moga być zarzadzane za pomoca `v-model`, co ułatwia powiazanie pól formularza z danymi komponentu.

```
<template>
  <form @submit.prevent="submitForm">
    <input v-model="formData.name" placeholder="Imi " />
    <input v-model="formData.email" placeholder="Email" />
    <button type="submit">Submit</button>
  </form>
</template>

<script>
export default {
  data() {
    return {
      formData: {
        name: '',
        email: ''
      }
    };
  },
  methods: {
    submitForm() {
      console.log(this.formData);
    }
  }
};
</script>
```

10.2 Walidacja Formularzy

Do walidacji formularzy możemy użyć zewnetrznych bibliotek, takich jak `Vuelidate` lub `formkit`, lub zaimplementować własne rozwiazania.

```
import useVuelidate from '@vuelidate/core';
import { required, email } from '@vuelidate/validators';

export default {
  setup() {
    const formData = reactive({
      name: '',
      email: ''
    });

    const rules = {
      name: { required },
      email: { required, email }
    };

    const v$ = useVuelidate(rules, formData);

    return { formData, v$ };
  }
};
```

11 Obsługa Zdarzeń Globalnych

Vue.js pozwala na obsługe zdarzeń globalnych, które moga być emitowane i nasłuchiwane w różnych komponentach.

11.1 Emitowanie i Nasłuchiwanie Zdarzeń

Możemy użyć wbudowanych metod `$emit` i `$on`, aby zarzadzać zdarzeniami globalnymi.

```
export default {
  methods: {
    emitEvent() {
      this.$emit('my-event', 'dane');
    }
  },
  created() {
    this.$on('my-event', (data) => {
      console.log('Zdarzenie odebrane:', data);
    });
  }
};
```

11.2 Globalny Bus Zdarzeń

Możemy również stworzyć globalny bus zdarzeń:

```
const EventBus = new Vue();

export default EventBus;
```

Nastepnie, w komponentach:

```
import EventBus from './EventBus';

export default {
  methods: {
    emitEvent() {
      EventBus.$emit('my-event', 'dane');
    }
  },
  created() {
    EventBus.$on('my-event', (data) => {
      console.log('Zdarzenie odebrane:', data);
    });
  }
};
```

12 Lazy Loading Komponentów

Lazy loading to technika ładowania komponentów na żadanie, co pozwala na optymalizacje wydajności aplikacji.

12.1 Dynamiczny Import

Możemy użyć `import()` do dynamicznego ładowania komponentów:

```
const MyComponent = () => import('./MyComponent.vue');

export default {
  components: {
    MyComponent
  }
};
```

12.2 Użycie `defineAsyncComponent`

Vue 3 wprowadza `defineAsyncComponent` do definiowania komponentów asynchronicznych.

```
import { defineAsyncComponent } from 'vue';

const MyComponent = defineAsyncComponent(() =>
  import('./MyComponent.vue')
```

```
);

export default {
  components: {
    MyComponent
  }
};
```

13 Middleware i Ochrona Tras w Vue Router

Middleware to funkcje, które moga być używane do ochrony tras w Vue Router przed nieautoryzowanym dostepem.

13.1 Przykład Middleware Autoryzacji

Możemy stworzyć prosty middleware, który sprawdzi, czy użytkownik jest zalogowany:

```
router.beforeEach((to, from, next) => {
  const isAuthenticated = !!localStorage.getItem('authToken');
  if (to.meta.requiresAuth && !isAuthenticated) {
    next('/login');
  } else {
    next();
  }
});
```

13.2 Ochrona Tras

Nastepnie, możemy oznaczyć trasy, które wymagaja autoryzacji:

```
const routes = [
  {
    path: '/dashboard',
    component: Dashboard,
    meta: { requiresAuth: true }
  },
  { path: '/login', component: Login }
];
```

14 Praca z Animacjami i Przejściami

Vue.js oferuje wbudowane narzedzia do animacji i przejść miedzy komponentami i elementami DOM.

14.1 Typy klas w `transition`

W Vue.js, gdy używamy komponentu `transition`, automatycznie stosowane sa odpowiednie klasy CSS na różnych etapach cyklu życia animacji. Oto pełna lista klas:

- `v-enter` - Stosowana na poczatku fazy wejścia (element jest dodawany do DOM).
- `v-enter-active` - Aktywna przez cały czas trwania fazy wejścia.
- `v-enter-to` - Stosowana na końcu fazy wejścia (Vue 2.x: `v-enter-active`, Vue 3.x: `v-enter-to`).
- `v-leave` - Stosowana na poczatku fazy wyjścia (element jest usuwany z DOM).
- `v-leave-active` - Aktywna przez cały czas trwania fazy wyjścia.
- `v-leave-to` - Stosowana na końcu fazy wyjścia.

Przykład zastosowania tych klas w stylach CSS:

```
.fade-enter-active, .fade-leave-active {
  transition: opacity 0.5s;
}
.fade-enter, .fade-leave-to {
  opacity: 0;
}
```

W powyższym przykładzie klasy `.fade-enter` i `.fade-leave-to` ustawiaja poczatkowa i końcowa przezroczystość elementu, natomiast `.fade-enter-active` i `.fade-leave-active` odpowiadaja za animacje samej zmiany przezroczystości.

14.2 Użycie `transition`

`transition` to wbudowany komponent do obsługi animacji.

```
<template>
  <transition name="fade">
    <p v-if="visible">Animowany tekst</p>
  </transition>
</template>

<script>
export default {
  data() {
    return { visible: true };
  }
};
</script>

<style>
.fade-enter-active, .fade-leave-active {
  transition: opacity 0.5s;
}
.fade-enter, .fade-leave-to {
  opacity: 0;
}
</style>
```

14.3 Animacje w Grupach

`transition-group` pozwala na animowanie list elementów.

```
<template>
  <transition-group name="list">
    <div v-for="item in items" :key="item.id">
      {{ item.text }}
    </div>
  </transition-group>
</template>

<style>
.list-enter-active, .list-leave-active {
  transition: all 0.5s;
}
.list-enter, .list-leave-to {
  opacity: 0;
  transform: translateY(30px);
}
</style>
```

15 Testowanie w Vue.js

Testowanie jednostkowe jest kluczowe dla utrzymania jakości kodu. Vue.js wspiera testowanie za pomoca narzedzi takich jak Jest i Vue Test Utils.

15.1 Proste Testy Jednostkowe

Przykład testu jednostkowego dla komponentu:

```
import { mount } from '@vue/test-utils';
import MyComponent from '@/components/MyComponent.vue';

describe('MyComponent', () => {
  it('renders a message', () => {
    const wrapper = mount(MyComponent, {
      props: { msg: 'Hello Vue' }
    });
    expect(wrapper.text()).toContain('Hello Vue');
  });
});
```

15.2 Testowanie Emitowania Zdarzeń

Testowanie, czy komponent poprawnie emituje zdarzenia:

```
it('emits an event when button is clicked', async () => {
  const wrapper = mount(MyComponent);
```

```
  await wrapper.find('button').trigger('click');
  expect(wrapper.emitted()).toHaveProperty('my-event');
});
```

16 Optymalizacja Aplikacji Vue.js

Optymalizacja aplikacji Vue.js może znaczaco poprawić jej wydajność.

16.1 Minimalizacja Renderów

Unikaj niepotrzebnych renderów za pomoca funkcji `shouldComponentUpdate` i efektywnego korzystania z `v-if` oraz `v-show`.

16.2 Memoizacja

Używaj funkcji `computed` oraz `watch`, aby zoptymalizować powtarzajace sie obliczenia.

```
computed: {
  expensiveCalculation() {
    return this.items.reduce((sum, item) => sum + item.value, 0);
  }
}
```

17 Obsługa Modułów Vuex

W wiekszych aplikacjach warto podzielić Vuex na moduły.

17.1 Tworzenie Modułów

Przykład tworzenia modułu Vuex:

```
const moduleA = {
  state: () => ({ count: 0 }),
  mutations: {
    increment(state) {
      state.count++;
    }
  }
};

export default new Vuex.Store({
  modules: {
    a: moduleA
  }
});
```

17.2 Dostep do Modułów w Komponentach

Dostep do modułów w komponentach:

```
computed: {
  countA() {
    return this.$store.state.a.count;
  }
}
```

18 Integracja z API i Autoryzacja

Integracja z zewnetrznymi API oraz zarzadzanie autoryzacja użytkowników to kluczowe elementy w nowoczesnych aplikacjach.

18.1 Wysyłanie Żadań HTTP z Axios

Vue.js nie ma wbudowanego modułu do obsługi żadań HTTP, ale możemy użyć biblioteki `axios`.

```
import axios from 'axios';

export default {
  data() {
    return {
      userData: null
    };
  },
  methods: {
    async fetchData() {
      try {
        const response = await axios.get('/api/user');
        this.userData = response.data;
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    }
  },
  mounted() {
    this.fetchData();
  }
};
```

18.2 Zarzadzanie Tokenami Autoryzacji

Przykład zarzadzania tokenami autoryzacji:

```
axios.interceptors.request.use(config => {
  const token = localStorage.getItem('authToken');
```

```
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
}, error => {
  return Promise.reject(error);
});
```

19 Podsumowanie

Dodane sekcje rozszerzaja zakres notatek o Vue.js, obejmujac zaawansowane techniki i najlepsze praktyki, które sa kluczowe dla efektywnego tworzenia nowoczesnych aplikacji. Każda z tych sekcji dostarcza praktycznych przykładów i kodu, który może być bezpośrednio zastosowany w rzeczywistych projektach.