Why to not change props in components:

It's important that props serve as one-way data binding — a component should never alter the value of its own props. There are a lot of reasons for this. In part, components editing props can make debugging a challenge. If a value is passed to multiple children, it could be hard to track where the changes to that value were coming from. In addition, changing props can cause components to re-render. So mutating props in a component would trigger the component to rerender, which may in-turn trigger the mutation again.

Żeby użyć komponentu w <template> to musisz najpierw go zaimportować i dodać do components:

export default {

  name: 'SearchView',

  components: {

    ClaimMain,

  },

The data property is where you can manage local state in a component, it lives inside the component object alongside the props property and has the following structure:

data() {

return {

key: value

}

}

Because of the way that this works in arrow functions (binding to the parent's context), you wouldn't be able to access any of the necessary attributes from inside data if you used an arrow function. So don't use an arrow function for the data property.

To make a method available to the ToDoForm component, we need to add it to the component object, and this is done inside a methods property to our component, which goes in the same place as data(), props, etc. The methods property holds any methods we might need to call in our component. When referenced, methods are fully run, so it's not a good idea to use them to display information inside the template. For displaying data that comes from calculations, you should use a computed property, which we'll cover later.

export default {

methods: {

onSubmit() {

console.log('form submitted')

}

}

}

To prevent the browser from posting to the server, we need to stop the event's default action while bubbling up through the page ([Event.preventDefault()](https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault), in vanilla JavaScript). Vue has a special syntax called event modifiers that can handle this for us right in our template. Modifiers are appended to the end of an event with a dot like so: @event.modifier. Here is a list of event modifiers:

* .stop: Stops the event from propagating. Equivalent to [Event.stopPropagation()](https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation) in regular JavaScript events.
* .prevent: Prevents the event's default behavior. Equivalent to [Event.preventDefault()](https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault).
* .self: Triggers the handler only if the event was dispatched from this exact element.
* {.key}: Triggers the event handler only via the specified key. [MDN has a list of valid key values](https://developer.mozilla.org/en-US/docs/Web/API/UI_Events/Keyboard_event_key_values); multi-word keys just need to be converted to kebab case (e.g. page-down).
* .native: Listens for a native event on the root (outer-most wrapping) element on your component.
* .once: Listens for the event until it's been triggered once, and then no more.
* .left: Only triggers the handler via the left mouse button event.
* .right: Only triggers the handler via the right mouse button event.
* .middle: Only triggers the handler via the middle mouse button event.
* .passive: Equivalent to using the { passive: true } parameter when creating an event listener in vanilla JavaScript using [addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener).

v-model to syntax sugar dla :value I @input robi to samo co te 2 właściwości ale za pomocą samego v-model.

In a similar fashion to event modifiers, we can also add modifiers to change the behavior of v-model. In our case, there are two worth considering. The first, .trim, will remove whitespace from before or after the input. We can add the modifier to our v-model statement like so: v-model.trim="label".

The second modifier we should consider is called .lazy. This modifier changes when v-model syncs the value for text inputs. As mentioned earlier, v-model syncing works by updating the variable using events. For text inputs, this sync happens using the [input event](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/input_event). Often, this means that Vue is syncing the data after every keystroke. The .lazy modifier causes v-model to use the [change event](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change_event) instead. This means that Vue will only sync data when the input loses focus or the form is submitted. For our purposes, this is much more reasonable since we only need the final data.

Zmienna w v-model musi być zadeklarowana w data() inaczej wyjebie błąd:

Obraz zawierający tekst

Opis wygenerowany automatycznie

To use both the .lazy modifier and the .trim modifier together, we can chain them, e.g. v-model.lazy.trim="label".

Computed Properties work similarly to methods, but only re-run when one of their dependencies changes.

Dwa sposoby deklarowania emitów:

<button type="button" @click="deleteToDo">

  methods: {

    deleteToDo() {

      this.$emit('item-deleted');

    },

}

Lub:

<button type="button" click="$emit('item-deleted')">

Jak stworzyć nowy projekt za pomocą vue-cli?

vue create my-project

Zawsze jak odnosimy się do zmiennych wewnątrz tagów <script> to trzeba dodać this.zmienna

Importowanie axiosa:

import axios from 'axios';

wcześniej trzeba oczywiście zainstalować axiosa w konkretnym folderze.

Lodash bardzo przydatne.

.debounce() – czeka odpowiednią ilość czasu po ostatnim evencie i dopiero wtedy wykonuje funkcję. Przydatne np. jak masz inputa, żeby nie robił zapytań do serwera po każdej literze wpisanej.

Nie ma sensu instalować całego loadash, żeby użyć funkcji .debounce() instalujesz tylko tę funkcję, czyli:

npm i lodash.debounce

użycie debounce:

 methods: {

    handleInput: debounce(() => {

handleInput nie może być z nawiasami(), bo mamy debounce jako funkcję i w niej zamykamy wszystko i jako drugi argument po }, 500) czyli, że czekamy 500ms

Dzięki serverowi vue można się połączyć ze stronką np. na telefonie jeśli korzystamy z tego samego wifi.

Webpack w vue działa tak, że jak masz ścieżkę do importowanego pliku to możesz postawić @/components/…

@ oznacza folder rootowy folder np. src

Można dzięki temu podawać ścieżki względem folderu src.

<style lang="scss" scoped>

Scoped oznacza, że style mają scope tylko na pliku, w którym są

Mimo to jeśli mamy 2 elementy zagnieżdżone w sobie z tą samą klasą to dziecko może dziedziczyć style po rodzicu tak samo nazwanym.

W Vue dobrpraktyką jest robienie osobnego komponentu np. tylko dla background-image. Dzięki temu możemy łatwo wyłączyć i włączyć komponent i mamy większą kontrolę nad tłem no i kod jest czystszy.

Jak mieć dostęp do np. wartości imputa jeśli jest w innym komponencie, a chcemy wyświetlić to jeszcze w innym komponencie?

Do wywołania komponentu dodajesz v-model=’nazwa zmiennej z komponentu, w którym chcemy mieć wartość inputa’:

<SearchInput v-model=’searchValue’ />

v-model zawiera w sobie :value=’searchValue’

W propsach w komponencie który zawiera wartość inputa tworzymy propsa o nazwie value: {} ponieważ jest to domyślna nazwa propsa przy użyciu v-model. Bo v-model zawiera :value i to nazwa dwukropku mówi jak będzie się nazywał props. Jeśli nie użylibyśmy v-model tylko zwykłe bindowanie zamiast :value to :wartość to props też musiałby mieć nazwę wartość: {}

Dodawanie klasy do elementu html warunkowo:



Klasa zostanie dodana jeśli step === 1

Jeśli mamy dodatkowo zwykłą klasę na stałę to oplatamy wszystkie klasy w tablicę i klasę stałą w stringa

Z czego składa się dyrektywa:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeśli nie podasz w nawiasie argumentu to wywoła się funkcja greet i argumentem będzie natywny event.

<button @click="greet">Greet</button>

Można też podać coś w nawiasie wtedy jest to custom event który jest przekazywany do funkcji.

<button @click="say('bye')">Say bye</button>

methods: {

say(message) {

alert(message)

}

}

Jeśli potrzebujesz eventu danego elementu DOM ale przekazujesz jeszcze inne argumenty do funkcji ($event lub event ) to używasz:

<!-- using $event special variable -->

<button @click="warn('Form cannot be submitted yet.', $event)">

Submit

</button>

<!-- using inline arrow function -->

<button @click="(event) => warn('Form cannot be submitted yet.', event)">

Submit

</button>

methods: {

warn(message, event) {

// now we have access to the native event

if (event) {

event.preventDefault()

}

alert(message)

}

}

Funkcja submit wykona się dopiero gdy naciśnie się enter:

<input @keyup.enter="submit"/>

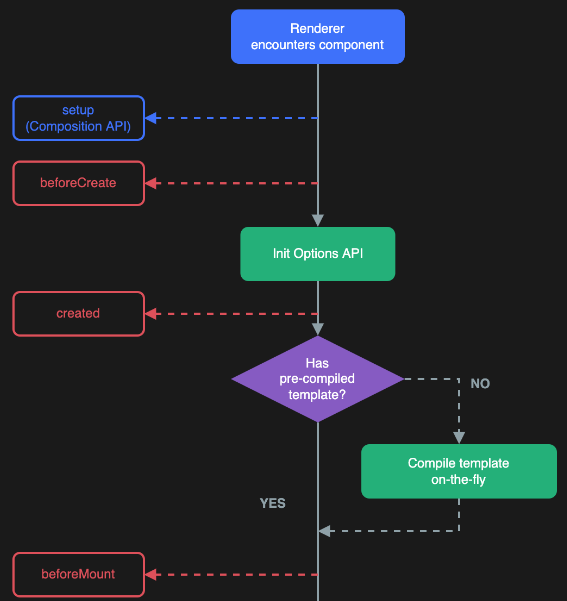
<input @keyup.ctrl.i="submit"/> => działa po kliknięciu ctrl + i

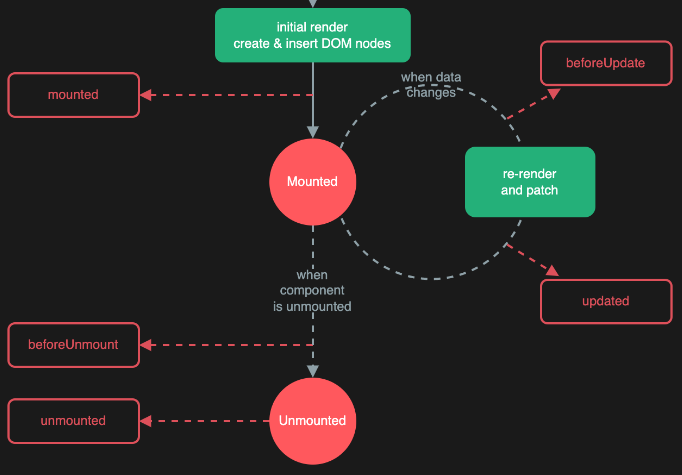
<!-- this will only fire when Ctrl and no other keys are pressed -->

<button @click.ctrl.exact="onCtrlClick">A</button>

All lifecycle hooks are called with their this context pointing to the current active instance invoking it. Note this means you should avoid using arrow functions when declaring lifecycle hooks, as you won't be able to access the component instance via this if you do so.

Lifecycle diagram:





Jeśli chcesz żeby wywołała się jakaś funkcja po zmianie danej zmiennej to używasz watch: {}

// camelCase in JavaScript

const BlogPost = {

props: ['postTitle'],

emits: ['updatePost'],

template: `

<h3>{{ postTitle }}</h3>

`

}

template

<!-- kebab-case in HTML -->

<blog-post post-title="hello!" @update-post="onUpdatePost"></blog-post>

Różnica między dom template a SFC template => https://stackoverflow.com/questions/68578348/what-exactly-is-a-dom-template-in-vue-js

Komponent można zarejestrować lokalnie i globalnie:

Globalny komponent:

Obraz zawierający tekst

Opis wygenerowany automatycznieNie jest to rozwiązanie idealne, bo np. jeśli używamy Vite to nawet jeśli przestaniemy używać takiego komponentu, to będzie on uwzględniony w finalnej paczce przez co użytkownicy muszą ściągać więcej kodu js

Lokalny komponent:

Obraz zawierający tekst

Opis wygenerowany automatycznieWystępuje w środku innego komponentu. Więc w template nie możemy użyć tego komponentu na zewnątrz komponentu, w którym go zaimportowaliśmy.