27 Helpful Tips for Vue Developers

Michael Hoffmann

27 Helpful Tips for Vue Developers

Bring your Vue knowledge to the next level!

Michael Hoffmann

This book is for sale at http://leanpub.com/27-helpful-tips-for-vue-developers

This version was published on 2022-05-15

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Michael Hoffmann by spreading the word about this book on Twitter!

The suggested hashtag for this book is #27-vuejs-tips.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#27-vuejs-tips

Contents

Introduction

Do you want to bring your Vue knowledge to the next level? In this book, I will show you 27 helpful Vue tips to help you reach that goal.

I wrote this book because I moved my technical focus to Vue.js and collected some tips & tricks to document my learning process.

The tips include basic recommendations like preferring slots over props and more specific tips like creating a custom v-model modifier.

Some words about my background: I'm a Senior Frontend Developer and Freelancer based in Munich/Germany. I hold a Masters in Electrical Engineering from the renowned Technical University of Munich (TUM) and have 7+ years of professional experience. In the last few years, I moved my focus from Angular and React to Vue.js.

But let's not waste more time: Onto Tip 1!

Michael Hoffmann

Web: mokkapps.de[1]
Email: mail@mokkapps.de[2]
Twitter: @mokkapps[3]
GitHub: github.com/mokkapps[4]

[1] https://mokkapps.de
[2] mailto:mail@mokkapps.de
[3] https://twitter.com/mokkapps
[4] https://github.com/mokkapps
[5] mailto:mail@mokkapps.de

Tip 1: Prefer Slots Over Props

Slots[6] in Vue.js give more flexibility than props[7].

It would be best to use slots to give the parent the freedom to customize components.

On the other hand, you should use props if you have a defined design and need to change some values.

Let's take a quick look at a simple Vue component that accepts a message as property:

```
<template>
  <div>{{ msg }}</div>
</template>

<script>
export default {
  name: 'HelloWorldProps',
  props: {
    msg: String,
  },
};
</script>

<style scoped></style>
```

The following Vue component uses a slot instead of a property to show the message:

```
<template>
  <div><slot /></div>
</template>

<script>
export default {
  name: 'HelloWorldSlots',
  props: {},
};
</script>

<style scoped></style>
```

[6] https://v3.vuejs.org/guide/component-slots.html#slots
[7] https://v3.vuejs.org/guide/component-props.html#props

The following code shows how both components can be used in your Vue application:

```
<template>
  <div>
    <HelloWorldProps msg="Welcome to Your Vue.js App" />
    <HelloWorldSlots>
      <h2>Welcome to Your Vue.js App</h2>
    </HelloWorldSlots>
  </div>
</template>
```

As you can see, the slot provides more flexibility as you can pass any HTML code into the slot.

Tip 2: Reactive Values in CSS

Since Vue 3[8] it is possible to use reactive values in the `<style>` block:

```
<template>
  <span class="label">Hello World!</span>
</template>

<script>
export default {
  props: {
    color: {
      type: String,
      default: 'blue',
    },
  }
};
</script>

<style>
  .label {
    color: v-bind(color);
  }
</style>
```

In this example, we bind the `color` property of our Vue component to the CSS color property of our label.

Internally, Vue uses CSS variables that are scoped to each component.

Fore more details, visit the official documentation[9].

[8] https://v3.vuejs.org/
[9] https://v3.vuejs.org/api/sfc-style.html#state-driven-dynamic-css

Tip 3: Detailed Prop Definitions

Your Vue prop definitions should always be as detailed as possible, specifying at least type(s):

Let's take a look at a simple prop definition:

```
const props = defineProps<{status: string}>();
```

A more detailed prop definition would be:

```
const props = defineProps<{
  status: {
    type: String,
    default: 'warning',
    required: true,
    validator(value) {
      // The value must match one of these strings
      return ['success', 'warning', 'danger'].includes(value)
    }
  }
}>();
```

The second example has two advantages:
- API documentation: It's easy to see how the component is meant to be used.
- Development Warnings: In development mode, you will get warnings if you pass incorrectly formatted props to a component.

The official documentation[10] provides further information on how to add more details to your prop definitions.

[10] https://vuejs.org/guide/components/props.html

Tip 4: Props and Context in Setup Method

When using the setup function in Vue 3's[11] Composition API, it will take two arguments: `props` and `context`:

```
<script>
export default {
  props: {
    title: String,
  },
  setup(props, context) {
    /**
     * Setup does not have the same access to `this` as
     * the Options API methods.
     * Thankfully, the setup method accepts a component's props
     * as its first argument
     */
    console.log(props);

    /**
     * Context object exposes three properties of a Vue instance
     * 1. attrs - a component's attributes
     * 2. slots - a component's slots
     * 3. emit - allows us to emit an event from this component
     */
    console.log(context);
    context.emit('eventName');
  },
};
</script>
```

The first argument in the `setup` function is the `props` argument. `props` are reactive and will be updated when new props are passed in.

You cannot use ES6 destructuring because it will remove `props` reactivity. If you want to destructure `props` you need to use toRefs[12].

[11] https://v3.vuejs.org/

[12] https://v3.vuejs.org/guide/reactivity-fundamentals.html#destructuring-reactive-state

The second argument in the `setup` function is the `context` argument. It is a normal JavaScript object that exposes some useful values:

```
export default {
  setup(props, context) {
    // Attributes (Non-reactive object, equivalent to $attrs)
    console.log(context.attrs)

    // Slots (Non-reactive object, equivalent to $slots)
    console.log(context.slots)

    // Emit events (Function, equivalent to $emit)
    console.log(context.emit)

    // Expose public properties (Function)
    console.log(context.expose)
  }
}
```

Tip 5: Avoid Empty Class Attributes

It is best practice to avoid empty class attributes.

For example, the following code will render the `div` with an empty class attribute:

```
<div :class="isDisabled ? 'disabled' : ''"></div>
<!-- Will render <div class></div> -->
```

By passing `null` instead of the empty string we can avoid the rendering of the empty class attribute:

```
<div :class="isDisabled ? 'disabled' : null"></div>
<!-- Will render <div></div> -->
```

Tip 6: Destructure in v-for

I'm a big fan of ES6 destructuring[13] and recently discovered that you can use it in `v-for`.

First, let's take a look at a `v-for` without using ES6 destructuring:

```
<div v-for="article in articles" :key="article.id">
  <h2>{{ article.title }}</h2>
</div>
```

`articles` is a list of `article` JavaScript objects and has the type `Article[]`:

```
interface Article {
  title: string;
  id: string;
}
```

We can use ES6 destructuring which results in a cleaner template code:

```
<div v-for="{id, title} in articles" :key="id">
  <h2>{{ title }}</h2>
</div>
```

[13] https://hacks.mozilla.org/2015/05/es6-in-depth-destructuring/

Tip 7: Avoid Unwanted Re-Renders of an Element Using v-once

The v-once directive[14] is a Vue.js directive used to avoid unwanted re-renders of an element.

It optimizes the update performance by rendering the element and component only once.

Vue will treat the element/component and its children as static content on any subsequent re-render.

```
<p v-once>This text will never change: {{ message }}</p>
```

If we put the `v-once` directive on an element with children, these would also be treated as static content:

```
<section v-once>
  <h1>My Headline</h1>
  <p>{{ message }}</p>
</section>
```

Since 3.2[15], you can also memoize part of the template with invalidation conditions using `v-memo`.

[14] https://v3.vuejs.org/api/directives.html#v-once
[15] https://v3.vuejs.org/api/directives.html#v-memo

Tip 8: Use Multiple v-model Bindings

The v-model[16] gives the flexibility to
use multiple v-model directives on a single component instance.

It is possible to rename `modelValue` to whatever we want, and therefore we can use multiple `v-model` directives.

Let's take a look at an exemplary `App.vue` component that uses `HelloWorld` component, which provides two `v-model` directives:

```
// App.vue

<template>
  <HelloWorld v-model:firstName="firstName" v-model:lastName="lastName" />
</template>

<script setup lang="ts">
import HelloWorld from './components/HelloWorld.vue';

interface Props {
  firstName: string;
  lastName: string;
}

const props = withDefaults(defineProps<Props>(), {
  firstName: 'Michael',
  lastName: 'Hoffmann',
});
</script>
```

And here is the code for `HelloWorld.vue`:

[16] https://v3.vuejs.org/guide/component-basics.html#using-v-model-on-components

```
// HelloWorld.vue

<template>
  <div class="container">
    <label>First Name:</label>
    <input :value="firstName" />
    <label>Last Name:</label>
    <input :value="lastName" />
  </div>
</template>

<script setup>
const props = defineProps({
  firstName: String,
  lastName: String,
});

const emit = defineEmits(['update:firstName', 'update:lastName']);
</script>
```

The source code for this demo is available at StackBlitz[17].

[17] https://stackblitz.com/edit/vitejs-vite-tt9c3z?file=src/components/App.vue

Tip 9: Trigger Watcher Immediately

The watch hook[18] provides an option that its callback function is called immediately:

```
<template>
  <input v-model="counter">
</template>

<script setup>
import { ref, watch } from "vue";

const counter = ref(0);

watch(
  counter,
  (newValue, oldValue) => {
    console.log("The new counter value is: " + counter.value);
  },
  { immediate: true }
);
</script>
```

This code will generate the output `New counter value is: 0` after the setup method is executed initially. It will also create a log message for all subsequent counter-value changes.

[18] https://v3.vuejs.org/guide/composition-api-introduction.html#reacting-to-changes-with-watch

Tip 10: Simple Expressions in Templates

Your Vue component templates should only include simple expressions as they make your template more declarative.

If you have more complex expressions in your Vue.js component templates, you should refactor them into computed properties or methods. This allows us to reuse the code.

For example, you should avoid such a template:

```
<template>
    <span>{{
    fullName.split(' ').map((word) => {
        return word[0].toUpperCase() + word.slice(1)
    }).join(' ')
    }}</span>
</template>
```

A cleaner solution:

```
<template>
    <span>{{ normalizedFullName }}</span>
</template>
```

We have moved the complex expression to a computed property:

```
<script setup>
const props = defineProps({ fullName: String });

const normalizedFullName = computed(() =>
  props.fullName
    .split(' ')
    .map((word) => word[0].toUpperCase() + word.slice(1))
    .join(' ')
);
</script>
```

Tip 11: Assign Handler for Uncaught Errors

You should monitor your errors in production. Vue, therefore, provides the errorHandler[19] for uncaught errors during component render function and watchers. The handler gets called with the error and the application instance.

```
/**
* Handles Vue errors
* @param err - the error trace
* @param vm - VM component
* @param info - Vue-specific error info, e.g., which lifecycle hook the error was fo\
und in
*/
app.config.errorHandler = (err, vm, info) => {
  // handle error
}
```

Error tracking services Sentry[20] and Bugsnag[21] provide official integrations for the `errorHandler`.

It's also possible to add the errorCaptured[22] lifecycle hook to your component. It is called when an error from any descendent component is captured.

[19] https://v3.vuejs.org/api/application-config.html#errorhandler
[20] https://sentry.io/for/vue/
[21] https://docs.bugsnag.com/platforms/browsers/vue/
[22] https://v3.vuejs.org/api/options-lifecycle-hooks.html#errorcaptured

Tip 12: Use Teleport to Render a Component in a Different Place

The built-in Vue 3 Teleport component[23] allows us to render HTML outside the Vue.js application. This can be useful to position modals, toast notifications, overlays, and more.

Sometimes, a component's template belongs logically to the component itself. At the same time, from a technical point of view, it would be preferable to move this part of the template somewhere else in the DOM, outside of the Vue app.

A full-screen modal is a common scenario where you'd want to handle the modal's logic inside the component, but it should be positioned differently.

Teleport provides a clean way to allow us to control under which parent in the DOM we want a piece of HTML to be rendered, without having to resort to a global state or splitting this into two components:

```
<template>
  <teleport to="body">
    This will be appended to the body tag
  </teleport>
</template>
```

The `to` attribute should be a valid CSS selector outside the Vue application.

Read more about Teleport in the official documentation[24].

[23] https://v3.vuejs.org/guide/teleport.html#teleport
[24] https://v3.vuejs.org/guide/teleport.html#teleport

Tip 13: Automatic Global Registration of Base Components

Base components are relatively generic components, like basic inputs or buttons. These components are frequently used across our application inside other components.

A typical component import might look like this:

```
import BaseButton from './BaseButton.vue';
import BaseIcon from './BaseIcon.vue';
import BaseInput from './BaseInput.vue';
export default {
  components: {
    BaseButton,
    BaseIcon,
    BaseInput,
  },
};
```

These imports are necessary to be able to reference the components inside our template:

```
<BaseButton>
  <BaseIcon name="search" />
</BaseButton>
<BaseInput v-model="searchText" />
```

Using webpack[25] or Vue CLI[26] we can use `require.context` to globally register only these very common base components.

Here's a code example from the official Vue docs[27] to globally import base components in your app's entry file (e.g. `src/main.js`):

[25] https://webpack.js.org/
[26] https://github.com/vuejs/vue-cli
[27] https://v3.vuejs.org/cookbook/automatic-global-registration-of-base-components.html#base-example

```
import { createApp } from 'vue';
import upperFirst from 'lodash/upperFirst';
import camelCase from 'lodash/camelCase';
import App from './App.vue';

const app = createApp(App);

const requireComponent = require.context(
  // The relative path of the components folder
  './components',
  // Whether or not to look in subfolders
  false,
  // The regular expression used to match base component filenames
  /Base[A-Z]\w+\.(vue|js)$/
);

requireComponent.keys().forEach(fileName => {
  // Get component config
  const componentConfig = requireComponent(fileName);

  // Get PascalCase name of component
  const componentName = upperFirst(
    camelCase(
      // Gets the file name regardless of folder depth
      fileName
        .split('/')
        .pop()
        .replace(/\.\w+$/, '')
    )
  );

  app.component(
    componentName,
    // Look for the component options on `.default`, which will
    // exist if the component was exported with `export default`,
    // otherwise fall back to module's root.
    componentConfig.default || componentConfig
  );
});

app.mount('#app');
```

Tip 14: Display Raw HTML

The double mustaches `{{ text }}` interprets the data as plain text, not HTML:

```
<p>Using text interpolation: {{ rawHtml }}</p>
```

In order to output real HTML, you will need to use the `v-html` directive:

```
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

The contents of the `span` will be interpreted as plain HTML, and all data bindings are ignored.

We cannot use `v-html` to compose template partials because Vue is not a string-based templating engine. Instead, components are preferred as the fundamental unit for UI reuse and composition.

[[warning]]
| Dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily lead to XSS vulnerabilities. Only use `v-html` on trusted content and never on user-provided content.

Tip 15: Scroll to Top When Navigating to a New Route

When using client-side routing using Vue Router[28] we may want to scroll to top when navigating to a new route or preserve the scrolling position of history entries.

Vue Router allows us to customize the scroll behavior on route navigation completely.

 This feature only works if the browser supports `history.pushState`.

When creating the router instance, we can provide the `scrollBehavior` function:

```
const router = new VueRouter({
  routes: [...],
  scrollBehavior (to, from, savedPosition) {
    return { x: 0, y: 0 }
  }
})
```

The following code scrolls to the top of the page for all route navigations:

```
scrollBehavior (to, from, savedPosition) {
  return { x: 0, y: 0 }
}
```

To achieve a native-like behavior when navigating with back/forward buttons we can return `savedPosition`:

```
scrollBehavior (to, from, savedPosition) {
  if (savedPosition) {
    return savedPosition
  } else {
    return { x: 0, y: 0 }
  }
}
```

Official documentation[29]

[28] https://router.vuejs.org/
[29] https://router.vuejs.org/guide/advanced/scroll-behavior.html

Tip 16: Speed Up Initial Load Using Async Components

The traditional, synchronous way of loading components is:

```
import MyComponent from './MyComponent.vue';
```

In large applications, it makes sense to split the code into smaller chunks and only load it from the server when it's needed.

Vue 3 therefore provides the `defineAsyncComponent` function:

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

const AsyncComp = defineAsyncComponent(() => {
  return new Promise((resolve, reject) => {
    // ...load component from server
    resolve(/* loaded component */);
  });
});
```

Now we can use `AsyncComp` like a typical component in Vue.

Here are the docs for Async Components in Vue 3.[30]

i The syntax for Vue 2 is not that different:

```
const AsyncComponent = () =>
import('./components/LazyLoadingFTW.vue');
```

[30] https://v3.vuejs.org/guide/migration/async-components.html#async-components

Tip 17: Use Two Script Blocks

If you are using the new `<script setup>` syntax, you may run into a case where you need some Options API functionality.
In this case, it's possible to add an `<script>` block to your component. Vue will mix the two together for you, so the Composition API code and Options API code can remain separate.

You may need a regular `<script>` block in these cases:

- Use the Options API to declare options that cannot be expressed in `<script setup>`, for example, `inheritAttrs` or custom options enabled via plugins.
- Run side effects or create objects that should only execute once because `setup()` is run for every component.
- Declare named exports which allow exporting multiple things from one file.

```
<script>
// Normal <script> using Options API, executed in module scope (only once)
runSideEffectOnce()

// Here you can declare additional options
export default {
  name: 'MyComponent',
  inheritAttrs: false,
  customOptions: {}
}
</script>

<script setup>
// Composition API: executed in setup() scope (for each instance)
import { ref } from 'vue';
console.log('Setting up new component instance');
const count = ref(0);
</script>
```

This feature is officially supported and documented[31].

[31] https://vuejs.org/api/sfc-script-setup.html#usage-alongside-normal-script

Tip 18: Use Optional Chaining in Templates

It's possible to use Optional Chaining[32] in Vue 3 templates:

> The optional chaining operator (?.) enables you to read the value of a property located deep within a chain of connected objects without checking that each reference in the chain is valid.

Here's a simple Vue component that uses the optional chaining operator:

```
<template>
  <p v-if="data?.user?.name">Name: {{ data?.user?.name }}</p>
  <p>Age: {{ data?.user?.age ?? 0 }} years old</p>
</template>

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

const data = ref({ user: { name: 'Michael' } });
</script>
```

This Vue SFC playground[33] shows the rendered output of this Vue component.

---

[32] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining

[33] https://sfc.vuejs.org/#eyJBcHAudnVlIjoiPHNjcmlwdCBzZXR1cD5cbmltcG9ydCB7IHJlZiB9IGZyb20gJ3Z1ZSdcblxuY29uc3QgZGF0YSA9IHJlZih7dXNlcjoge25
XG5cbjx0ZW1wbGF0ZT5cbiAgPHAgdi1pZj1cImRhdGE/LnVzZXI/Lm5hbWVcIj5OYW1lOiB7eyBkYXRhPy51c2VyPy5uYW1lIH19PC9wPlxuICA8cD5BZ2U6IHt7IG
LnVzZXI/LmFnZSA/PyAwIH19IHllYXJzIG9sZDwvcD5cbjwvdGVtcGxhdGU+IiwiaW1wb3J0LW1hcC5qc29uIjoie1xuICBcImltcG9ydHNcIjoge1xuICAgIFwidnVlXC

Tip 19: Special CSS Selectors

Deep Selectors

The `:deep()` CSS pseudo-class can be used if you want styles to apply to some "deeply" nested child components:

```
<style scoped>
  .a :deep(.b) {
  /* ... */
}
</style>
```

Vue will compile this code into:

```
.a[data-v-f3f3eg9] .b {
/* ... */
}
```

i DOM content created with `v-html` is not affected by scoped styles, but you can still style them using `deep` selectors.

Slotted Selectors

By default, scoped styles do not affect contents rendered by `<slot/>` but we can use the `:slotted` pseudo-selector to apply styles to slot content:

```
<style scoped>
:slotted(div) {
  color: red;
}
</style>
```

Global Selectors

The `:global` pseudo-class can be used to globally apply a rule, even within the `<style scoped>` block:

```
<style scoped>
:global(.red) {
  color: red;
}
</style>
```

To be more specific, I recommend adding a second `<style>` block if you need to apply multiple global styles:

```
<style scoped>
:global(.red) {
  color: red;
}
</style>

<style>
  body {
  margin: 0;
  padding: 0;
}
</style>
```

Official documentation[34]

[34] https://vuejs.org/api/sfc-css-features.html

Tip 20: Use Vuex in Vue Router Navigation Guards

Vue Router[35] allows us to define Navigation Guards[36].

For example, we can register them globally:

```
const router = createRouter({ ... })

/**
* to: the target route location in a normalized format being navigated to.
* from: the current route location in a normalized format being navigated away from.
*/
router.beforeEach((to, from) => {
  // ...
  // explicitly return false to cancel the navigation
  return false
})
```

A common scenario is to redirect an unauthenticated user to the login page if he tries to visit a protected route:

```
router.beforeEach(async (to, from) => {
  if (
    // make sure the user is authenticated
    !isAuthenticated &&
    // ☠ Avoid an infinite redirect
    to.name !== 'Login'
  ) {
    // redirect the user to the login page
    return { name: 'Login' };
  }
});
```

If you are using Vuex[37] you probably store the authenticated state in the Vuex store.

You can easily access that store in your guard navigation:

[35] https://router.vuejs.org/
[36] https://router.vuejs.org/guide/advanced/navigation-guards.html
[37] https://vuex.vuejs.org/

```
import store from '@/store';

router.beforeEach(async (to, from) => {
  const authenticated = store.getters['auth/authenticated'];

  // redirect the user to login page if he is not authenticated
  if (!authenticated && to.name !== 'Login') {
    return { name: 'Login' };
  }
});
```

Tip 21: Watch Nested Values

The watch[38] option supports a dot-delimited path as key to watch nested values:

```
<script setup lang="ts">
import { watch } from 'vue';
import { useRoute } from 'vue-router';

const route = useRoute();

watch(
  () => route.query.id,
  queryId => {
    console.log('Route Query ID', queryId);
  }
);
</script>
```

The above example assumes that you are using the Composition API[39] with TypeScript[40].

If you are using Options API[41] the code looks like this:

```
<script>
export default {
  watch: {
    // Note: only simple paths. Expressions are not supported.
    '$route.query.id'(queryId) {
      console.log('Route Query ID', queryId);
    },
  },
};
</script>
```

[38] https://vuejs.org/guide/essentials/watchers.html
[39] https://vuejs.org/guide/introduction.html#composition-api
[40] https://www.typescriptlang.org/
[41] https://vuejs.org/guide/introduction.html#options-api

Tip 22: Use v-bind to Pass Multiple Props to Components

It's totally valid to bind multiple props to a Vue component:

```
<template>
  <Button
    :type="type"
    :color="color"
    :disabled="disabled"
    :test-id="testId"
    :icon="icon"
    @click="onClick"
    @custom="onCustom"
  >
    Login
  </Button>
</template>
```

For a cleaner template you can also `v-bind` to bind an object including all your property information to your component:

```
<template>
  <Button v-bind="button">Login</Button>
</template>

<script setup lang="ts">
const button = {
  type: 'submit',
  color: 'blue',
  disabled: false,
  testId: 'login-button',
  icon: 'arrow-right',
};
</script>
```

If you need to handle a lot of events you might want to group them using `v-on`:

```
<template>
  <Button v-on="buttonEventHandlers">Login</Button>
</template>

<script setup lang="ts">
const buttonEventHandlers = {
  onClick: () => console.log('Click'),
  onCustom: () => console.log('Custom Event'),
};
</script>
```

i `v-bind` and `v-on` work with Vue 2 and Vue 3.

Tip 23: Animate Child Component Before Route Leave

Vue Router[42] provides a way to use transitions[43] on your route components and animate navigations.

But sometimes, you might want to animate a specific (sub-)component before a route is left.

Therefore, you can use the onBeforeRouteLeave navigation guard[44] combined with a ref[45] to toggle the visibility of the component:

i The `v-if` is necessary to trigger the animation. More at the official docs[46].

```
<template>
  <section class="body">
    <Transition name="image">
      <img v-if="showUI" src="../assets/image.svg" alt="Image" />
    </Transition>
  </section>
</template>

<script setup lang="ts">
import { ref } from 'vue';
import { onBeforeRouteLeave } from 'vue-router';

const showUI = ref(true);

onBeforeRouteLeave((to, from, next) => {
  if (to.name === 'Home') {
    showUI.value = false;
    setTimeout(() => {
      next();
    }, 1000);
  } else {
    next();
  }
});
```

[42] https://router.vuejs.org
[43] https://router.vuejs.org/guide/advanced/transitions.html
[44] https://router.vuejs.org/guide/advanced/composition-api.html#navigation-guards
[45] https://vuejs.org/api/built-in-special-attributes.html#ref
[46] https://vuejs.org/guide/built-ins/transition.html#the-transition-component

```
</script>

<style lang="scss">
.image-leave-active {
  transition: transform 1s ease;
}

.image-leave-to {
  transform: translateY(30vw) translateX(40vw) rotate(30deg);
}
</style>
```

Let's take a closer look at the code inside `onBeforeRouteLeave`:

```
onBeforeRouteLeave((to, from, next) => {
  if (to.name === 'Home') {
    showUI.value = false;
    setTimeout(() => {
      next();
    }, 1000);
  } else {
    next();
  }
});
```

If the following route is anything other than `Home`, we allow the navigation and call `next`.

If the following route is `Home`, we toggle the component's visibility by setting `showUI` to `false`. Next, we call `setTimeout` with 1 second to wait until the animation is done and then proceed with the route change by calling `next`.

The animation time is defined in our `<style>` block:

```
<style lang="scss">
.image-leave-active {
  // highlight-next-line
  transition: transform 1s ease;
}

.image-leave-to {
  transform: translateY(30vw) translateX(40vw) rotate(30deg);
}
</style>
```

Tip 24: Measure Performance

What I love about Vue[47] is that usually, it is performant without any manual optimization effort. Of course, there exist
unique scenarios where we need to optimize the performance, but therefore we first need to know how we can measure it.

Several great tools can help us to measure performance. If you want to profile the load performance of your production deployment, you can use one of these tools:

- PageSpeed Insights[48]
- WebPageTest[49]

If you want to profile the performance during local development, you can use one of these tools:

- Chrome DevTools Performance Panel[50]

i You can enable `app.config.performance` for Vue-specific performance markers in Chrome DevTools' performance timeline, see official docs[51].

- Vue DevTools Extension[52]

The official documentation[53] provides more information about Vue's performance.

[47] https://vuejs.org/
[48] https://pagespeed.web.dev/
[49] https://www.webpagetest.org/
[50] https://developer.chrome.com/docs/devtools/evaluate-performance/
[51] https://vuejs.org/api/application.html#app-config-performance
[52] https://vuejs.org/guide/scaling-up/tooling.html#browser-devtools
[53] https://vuejs.org/guide/best-practices/performance.html#overview

Tip 25: Check Version at Runtime

It's possible to check Vue's version at runtime by importing `version` from the Vue npm package and splitting the string at the `.` character:

```
<template>
  <h1>Vue Version: {{ vueVersion }}</h1>
</template>

<script setup>
import { version } from 'vue';

const vueVersion = version.split('.')[0];
</script>
```

Demo[54]

[54] https://sfc.vuejs.org/#eyJBcHAudnVlIjoiPHRlbXBsYXRlPlxuICA8aDE+VnVlIFZlcnNpb246IHt7IHZ1ZVZlcnNpb24gfX08L2gxPlxuPC90ZW1wbGF0ZT5cblxuPHNjcmlwdCBzZXR1cFxuaW1wb3J0IHsgdmVyc2lvbiB9IGZyb20gJ3Z1ZSc7XG5cbmNvbnN0IHZ1ZVZlcnNpb24gPSB2ZXJzaW9uLnNwbGl0KCcuJylbMF07XG48L3NjcmlwdD4iLCJpbXBvcnQtbWFwIjoiIn0=

Tip 26: Query Inner Elements in Third-Party Components

In some cases, we have to deal with third-party Vue components where we cannot put a ref on inner HTML elements to access them in our Vue code.

In these cases we can access $el[55] on a template ref[56] on the third-party component:

```
<template>
  <ThirdPartyComponent ref="comp"/>
</template>

<script setup lang="ts">
  const compRef: Ref<typeof ThirdPartyComponent | null> = ref(null);
  const input: HTMLInputElement | null = compRef.value?.$el.querySelector('input');
</script>
```

i In general and for consistency, you should use template refs[57] for direct access to elements instead of relying on `$el`.

[55] https://vuejs.org/api/component-instance.html#el
[56] https://vuejs.org/guide/essentials/template-refs.html#template-refs=
[57] https://vuejs.org/guide/essentials/template-refs.html#template-refs=

Tip 27: Create a Custom v-model Modifier

`v-model` has some built-in modifiers[58] like `.lazy`, `.number` and `.trim`. But in some cases, you might need to add your custom modifier.

In this simple demo, I want to create a custom modifier called `no-underscore` that removes all underscores _ from an input:

```
<Comp v-model.no-underscore="text" />
```

Inside our component we can access the modifier via the `modelModifiers` prop. We manipulate the value if an `input` event is fired and the modifier is available:

```
<script setup>
const props = defineProps({
  modelValue: String,
  // highlight-next-line
  modelModifiers: { default: () => ({}) },
});

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

function emitValue(e) {
  let value = e.target.value;
  // highlight-next-line
  if (props.modelModifiers['no-underscore']) {
    value = value.replace('_', '');
  }
  emit('update:modelValue', value);
}
</script>

<template>
  <input type="text" :value="modelValue" @input="emitValue" />
</template>
```

If your `v-model` binding includes both modifiers and argument then the generated prop name will be `arg + "Modifiers"`:

[58] https://vuejs.org/guide/essentials/forms.html#modifiers

```
<template>
  <Comp v-model:name.no-underscore="text" />
</template>

<script setup>
  const props = defineProps(['name', 'nameModifiers']);
  defineEmits(['update:name']);
</script>
```

Demo[59]

[59] https://sfc.vuejs.org/#eyJBcHAudnVlIjoiPHNjcmlwdCBzZXR1cD5cbmltcG9ydCB7IHJlZiB9IGZyb20gJ3Z1ZSdcbmltcG9ydCBDb21wIGZyb20gJy4vQ29tcC52d
e3sgdGV4dCB9fTwvaDE+XG4gIDxwIHYtbW9kZWwubm8tdW5kZXJzY29yZT1cInRleHRcIi8+XG48L3RlPiIsImltcG9ydC1tYXAuanNvbiI6IntcbiAg
XG5jb25zdCBwcm9wcyA9IGRlZmluZVByb3BzKFsnbmFtZScsICduYW1lTogU3RyaW5nLFxuICBtb2RlbE1vZGlmaWVyczogeyBkZWZhdWx0OiAoKSA9PiAo
XG5cdDxpbnB1dCB0eXBlPVwidGV4dFwiIDp2YWx1ZT1cIm1vZGVsVmFsdWVcIiBAaW5wdXQ9XCJlbWl0VmFsdWVcIiAvPlxuPC90ZW1wbGF0ZT4ifQ=
=

Conclusion

Thanks for purchasing and reading my first book. You are awesome!

I shared some helpful Vue tips that you can use in your current or future Vue projects.

If you liked the book, please support my work and spread the word:

- Subscribe to my weekly Vue.js newsletter[60]
- Follow me on Twitter[61]
- Follow me on LinkedIn[62]
- Follow me on Instagram[63]
- Follow me on GitHub[64]

Thanks,
Michael

www.mokkapps.de[65]

[60] https://mokkapps.de/newsletter
[61] https://twitter.com/mokkapps
[62] https://linkedin.com/mokkapps
[63] https://instagram.com/mokkapps
[64] https://github.com/mokkapps
[65] https://mokkapps.de