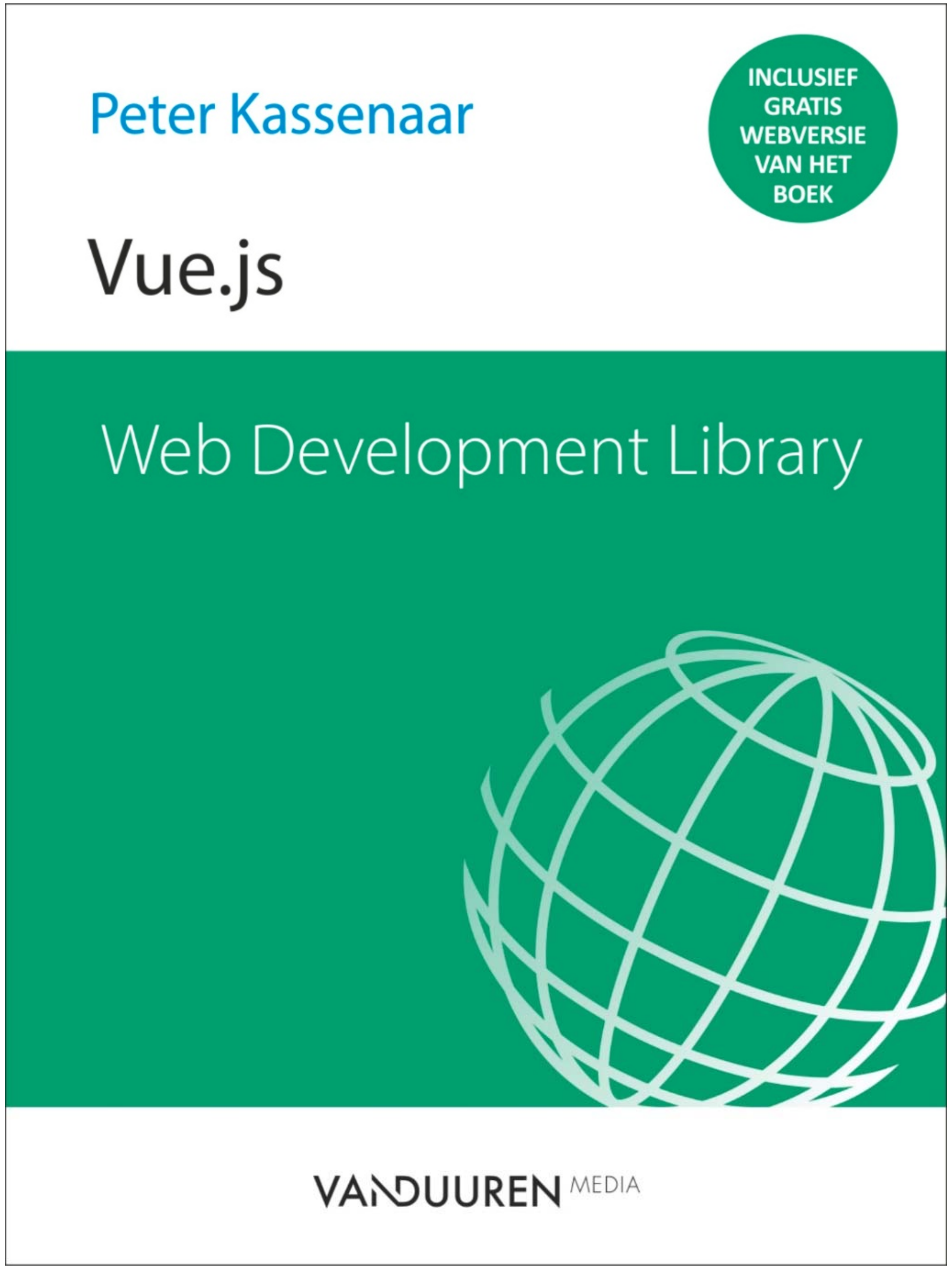
# Vue Fundamentals

# Style Bindings & in depth components

Peter Kassenaar –
info@kassenaar.com

## Using v-model

Two-way databinding with Vue


Peter Kassenaar
INCLUSIEF GRATIS WEBVERSIE VAN HET BOEK
Vue.js
Web Development Library
VANDUUREN MEDIA

P. 122 e.v.

## Using v-model to select changes

*“You can use the `v-model` directive to create two-way data bindings on form input, textarea, and select elements. It automatically picks the correct way to update the element based on the input type.”*

vuejs.org/guide/essentials/forms.html

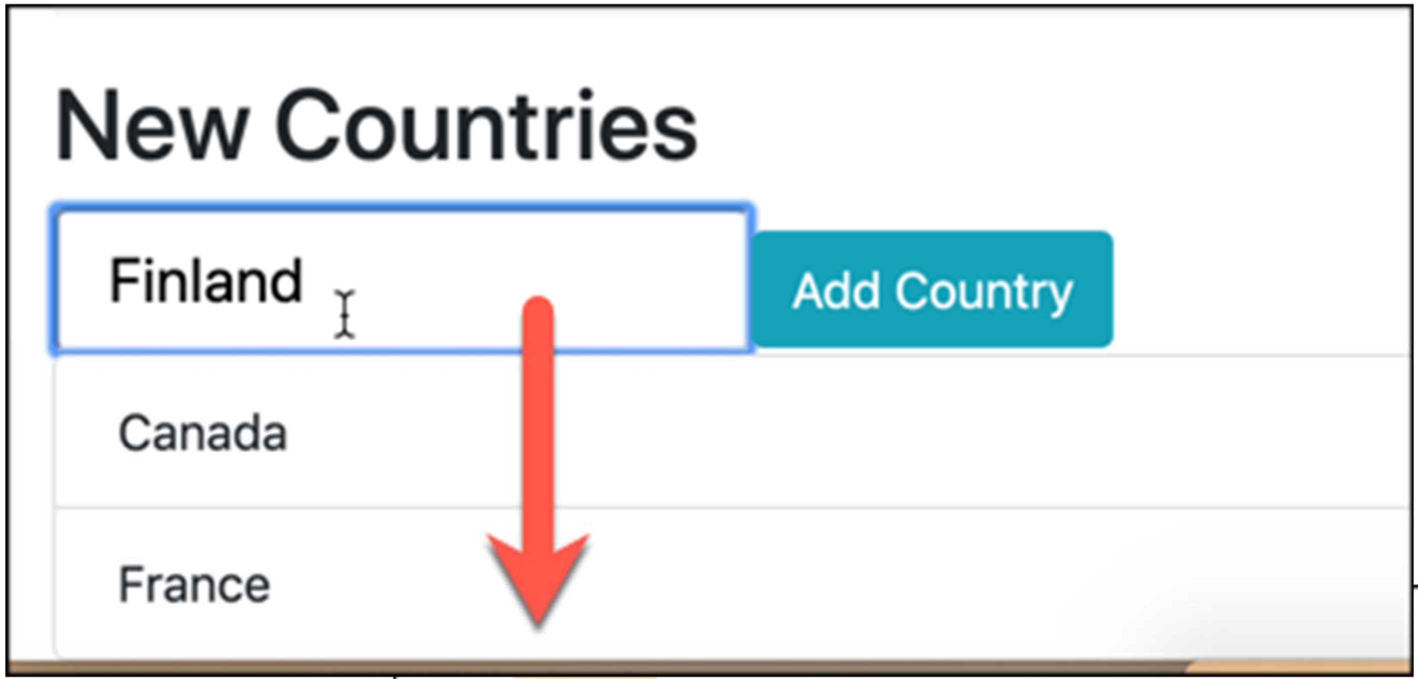
## **Using `v-model`**

*Two-way data binding*

Reflect changes in the UI back to the component and the other way around

```
<input type="text" v-model="…">
```

## Push items to (new) array



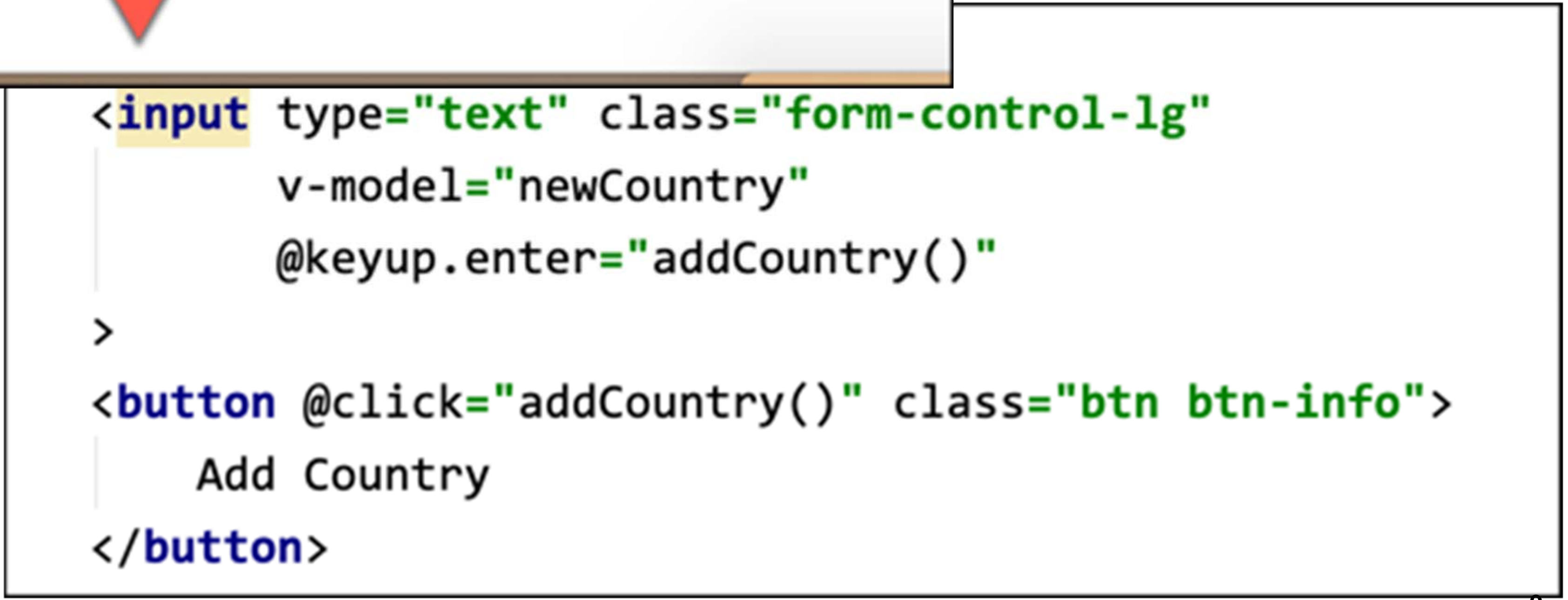
New Countries

Finland

Add Country

Canada

France



```
<input type="text" class="form-control-lg"
       v-model="newCountry"
       @keyup.enter="addCountry()"
>
<button @click="addCountry()" class="btn btn-info">
    Add Country
</button>
```

```
// Array, holding the newly added countries
const newCountries = ref([])
const newCountry = ref('');

// Adding a new country to the array
const addCountry = () => {
  newCountries.value.push(newCountry.value);
}
```

## Using `v-model` on check boxes

```
<input type="checkbox" v-model="…">
```

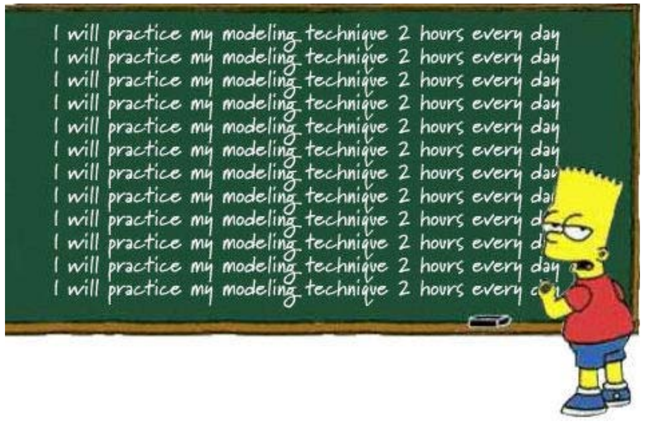
## Using `v-model` on radio buttons

```
<input type=“radio” v-model=“…”>
```

## Workshop v-model

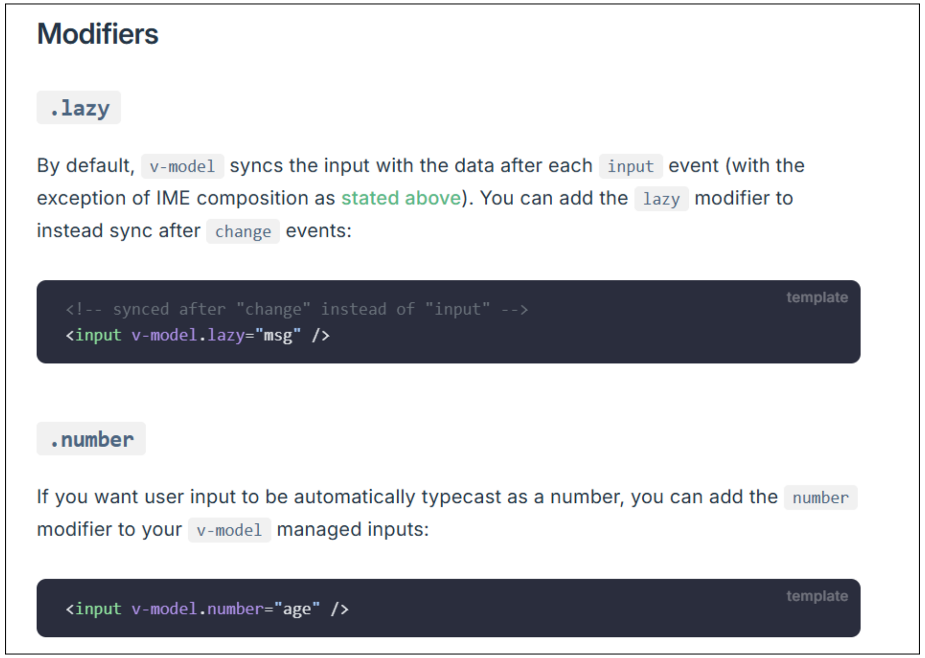
1. Create a component with 2 input fields. The values you type in one field, are copied to the other field and vice versa
2. Add checkboxes to your own data list. If a field is checked, it is added to an array and shown in the user interface
3. Optional: create a textfield on one component.
    1. Text that is typed in, is passed on as a prop to another component
    2. See default `HelloWorld` component as an example for props

- Examples: `…/125-v-model, 126-…, 127-…, 128-…`



## Optional - modifiers for v-model

- Modifying the input, received from a `v-model` textbox
  - `.lazy`
  - `.number`
  - `.trim`
- https://vuejs.org/guide/essentials/forms.html#modifiers



### Modifiers

#### `.lazy`

By default, `v-model` syncs the input with the data after each `input` event (with the exception of IME composition as stated above). You can add the `lazy` modifier to instead sync after `change` events:

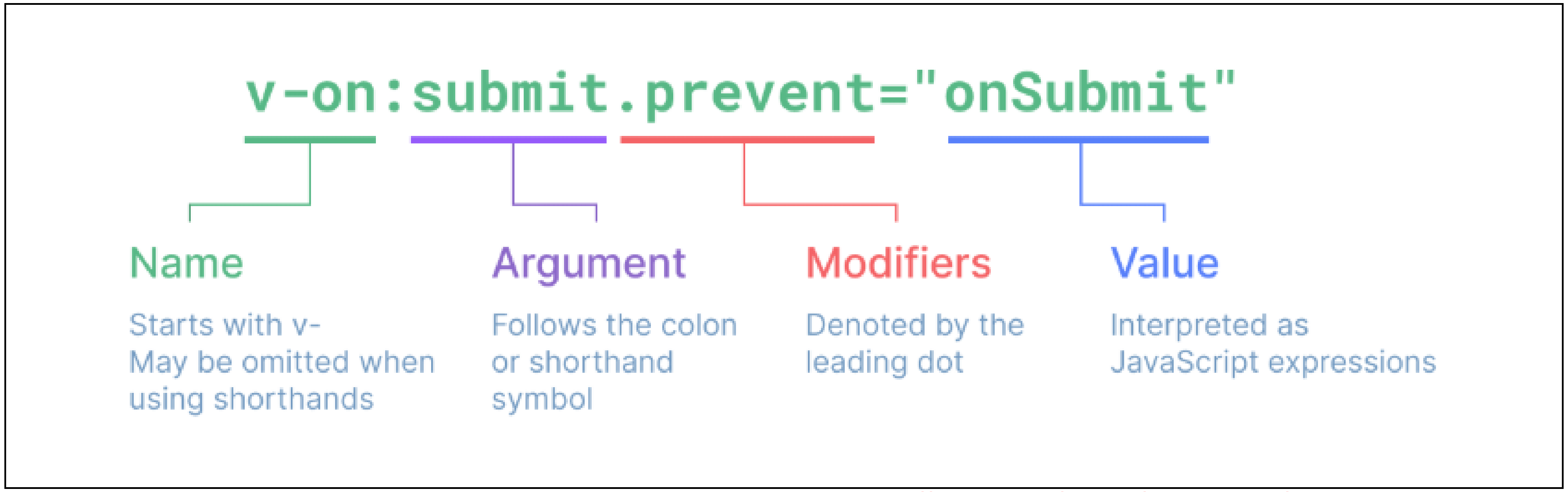
```
template
<!-- synced after "change" instead of "input" -->
<input v-model.lazy="msg" />
```

#### `.number`

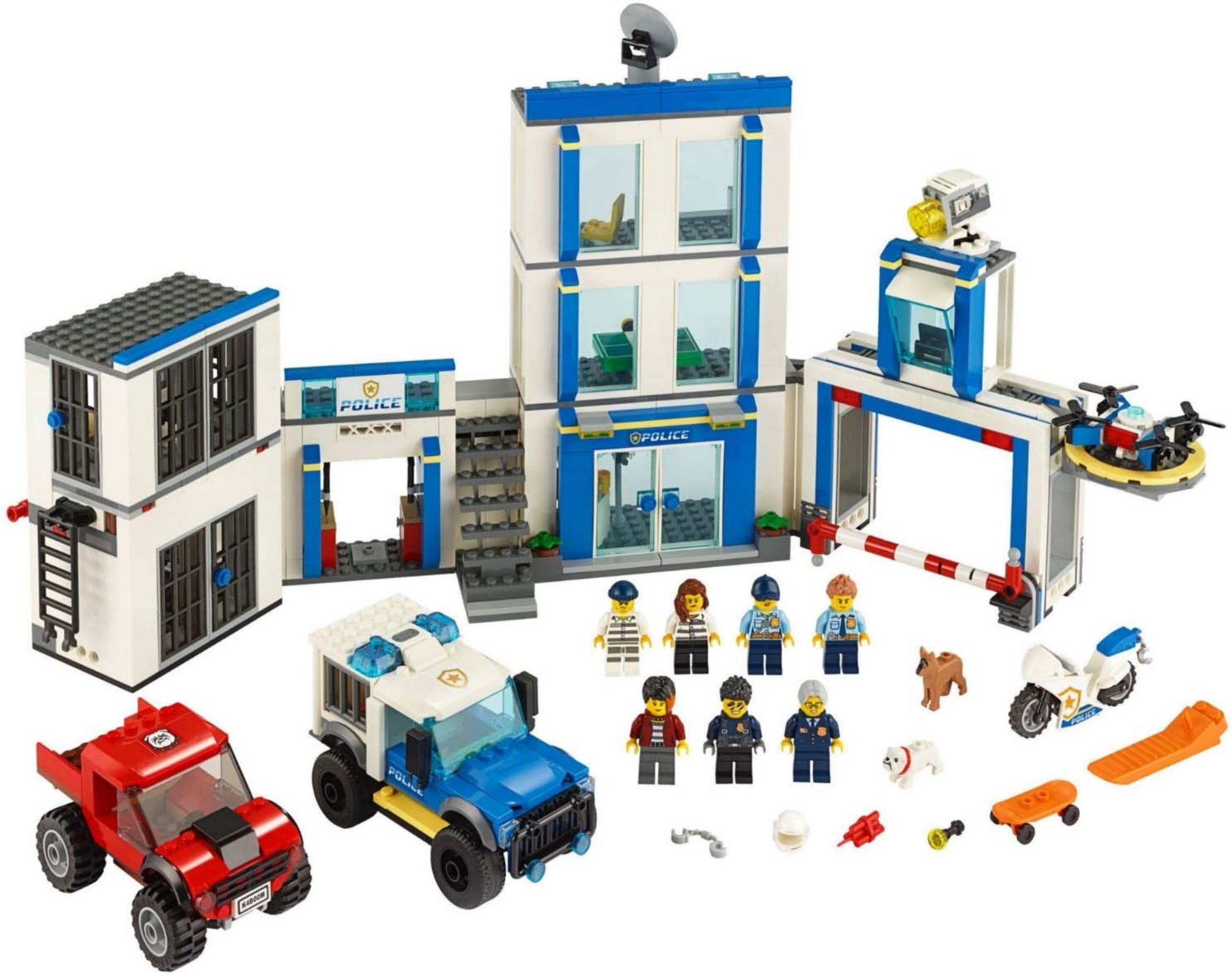
If you want user input to be automatically typecast as a number, you can add the `number` modifier to your `v-model` managed inputs:

```
template
<input v-model.number="age" />
```

## Full directive syntax visualized

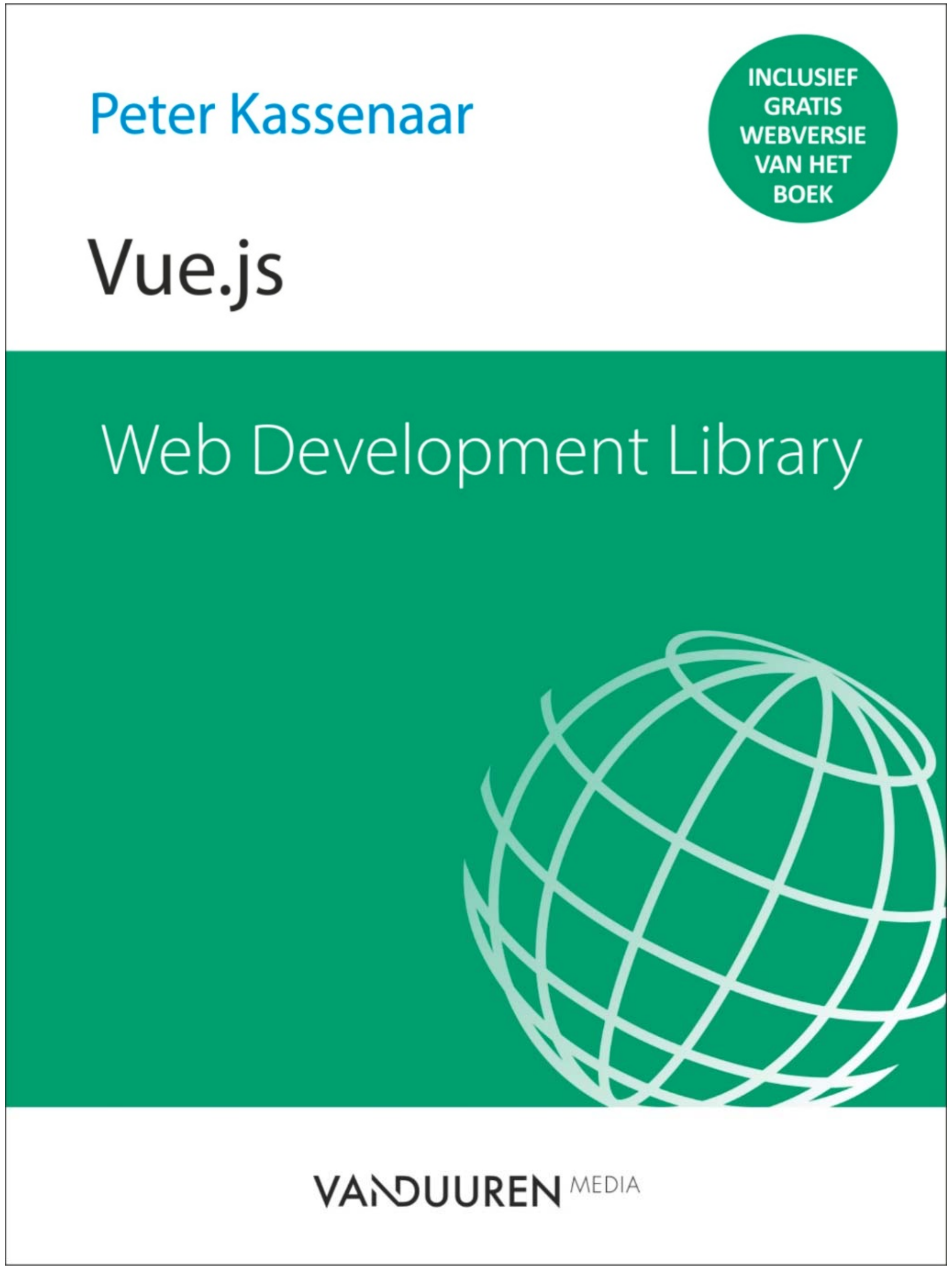

v-on:submit.prevent="onSubmit"
Name
Starts with v-
May be omitted when using shorthands
Argument
Follows the colon or shorthand symbol
Modifiers
Denoted by the leading dot
Value
Interpreted as JavaScript expressions

https://vuejs.org/guide/essentials/template-syntax.html


POLICE
POLICE
POLICE
POLICE

## Component lifecycle hooks

Tapping into the lifecycle of created components


Peter Kassenaar
INCLUSIEF GRATIS WEBVERSIE VAN HET BOEK
Vue.js
Web Development Library
VANDUUREN MEDIA

P. 118 e.v.

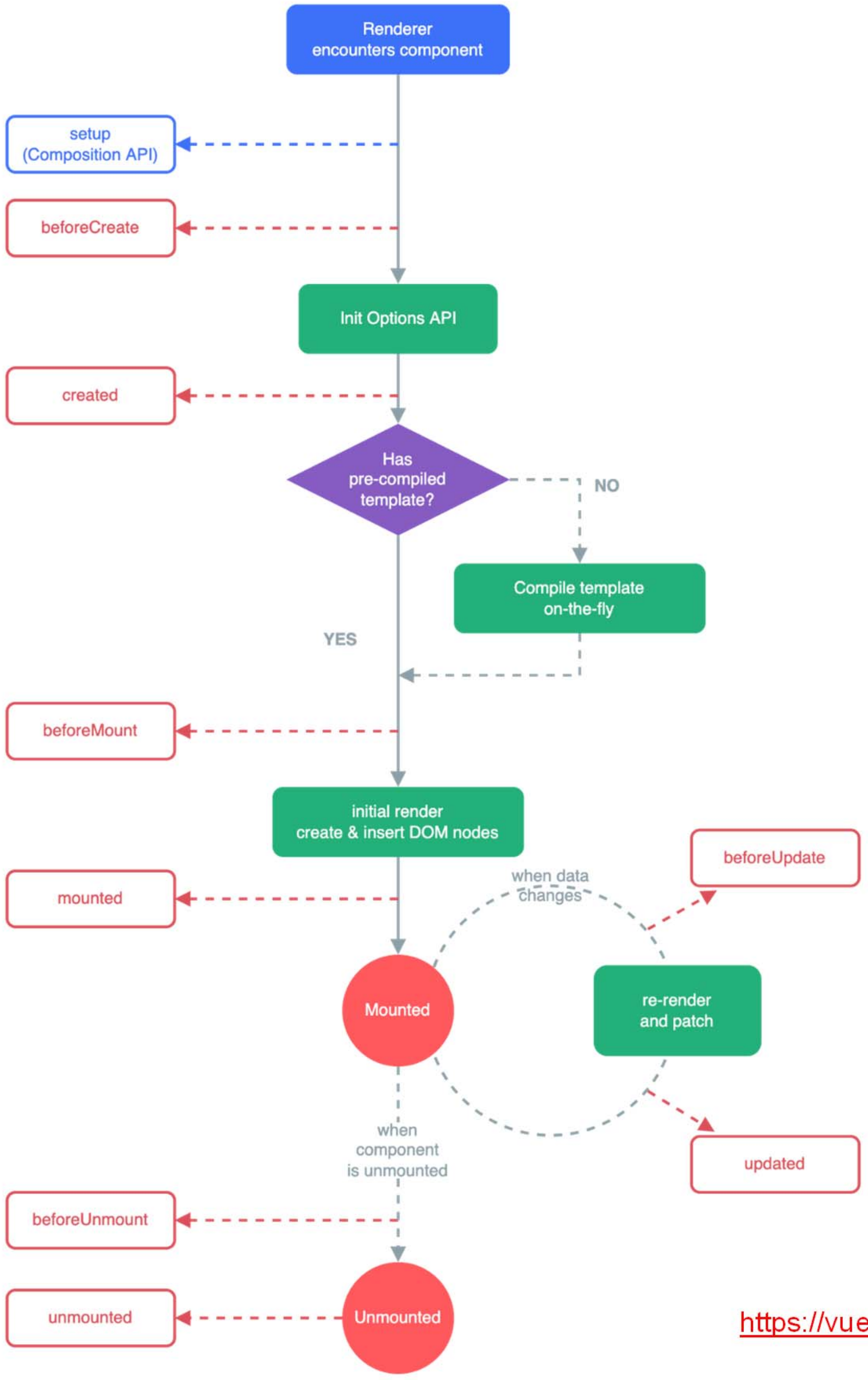
## Lifecycle hooks

- Perform an action automatically when a specific lifecycle event occurs

*“Each component instance goes through a series of initialization steps when it’s created - for example, it needs to set up data observation, compile the template, mount the instance to the DOM, and update the DOM when data changes.”*

https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks

## Official lifecycle diagram



The Red squares are the lifecycle hook methods.

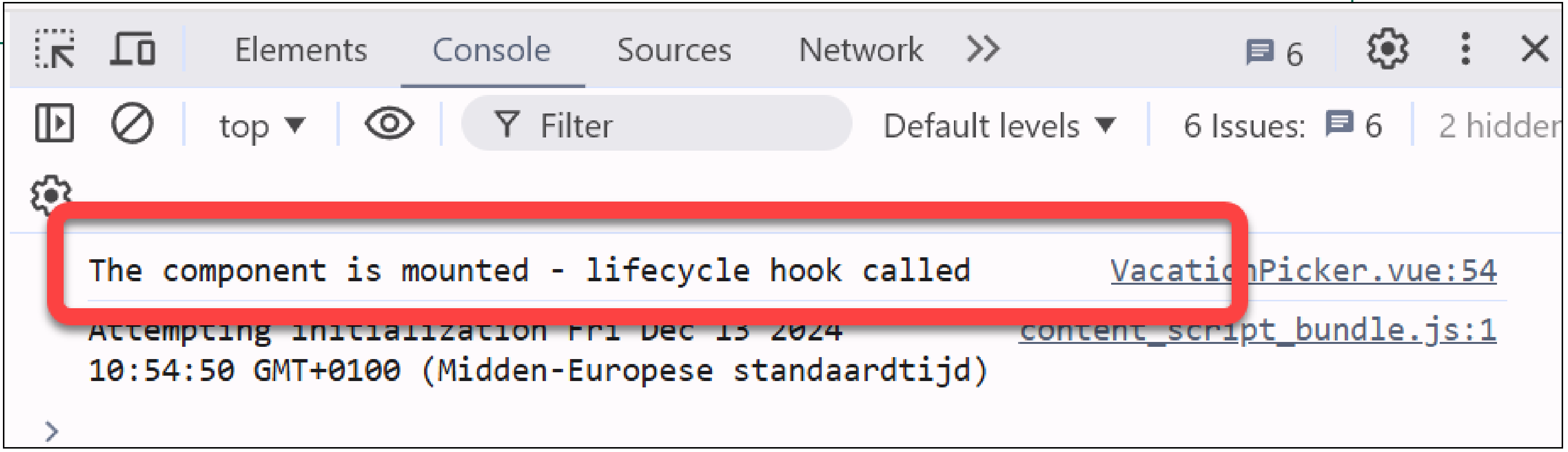
Most used:

- `mounted`
- `updated`
- `unmounted`

https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram

## Using the onMounted hook

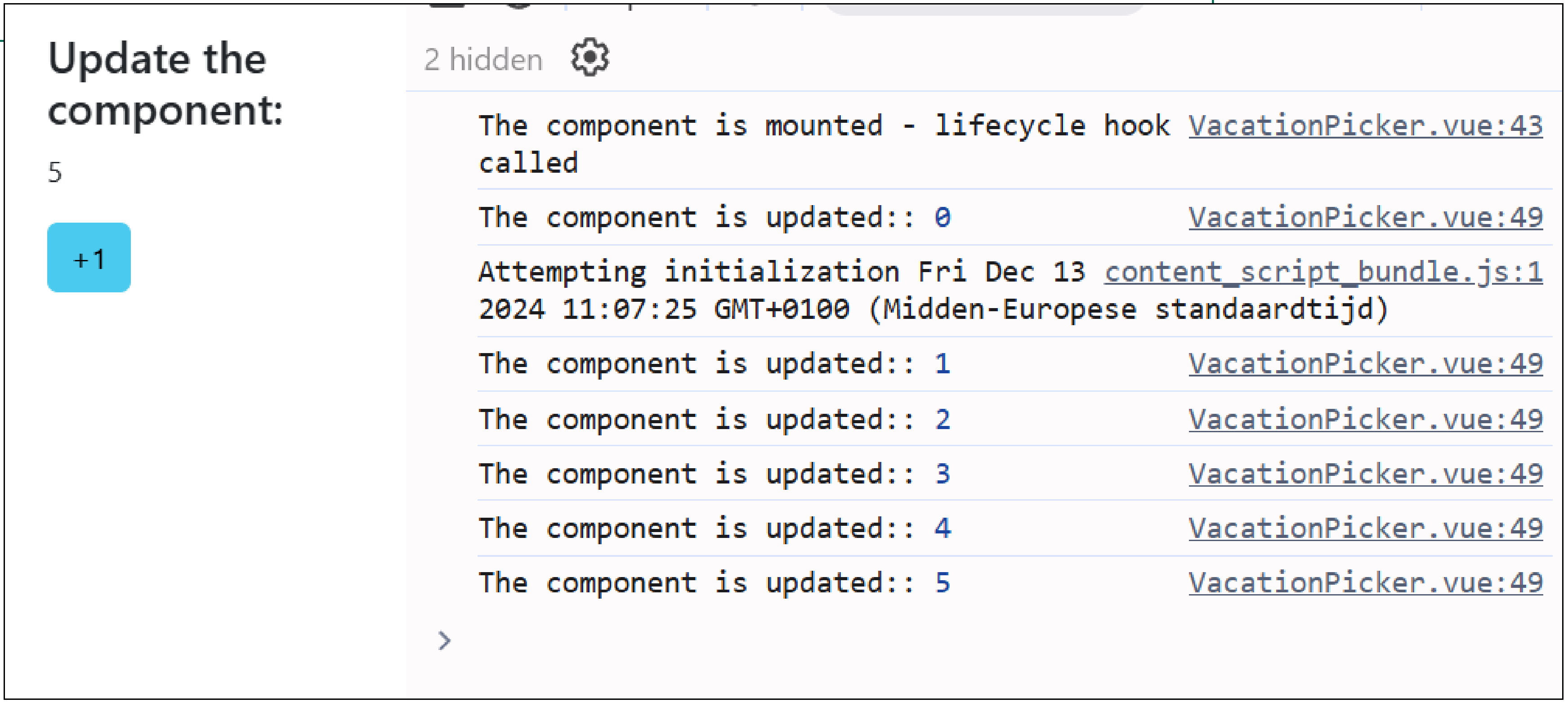
```
<script setup>
import {onMounted, ref} from "vue";
// …
//***************************
// Using the lifecycle hooks
//***************************
onMounted(()=>{
  console.log('The component is mounted - lifecycle hook called');
  header.value = 'Vaction Picker - component is created';
});

</script>
```



```
The component is mounted - lifecycle hook called          VacationPicker.vue:54
Attempting initialization Fri Dec 13 2024              content_script_bundle.js:1
10:54:50 GMT+0100 (Midden-Europese standaardtijd)
```

# Using the onUpdated hook

```
onUpdated(() => {
  console.log('The component is updated::', count.value);
})
```

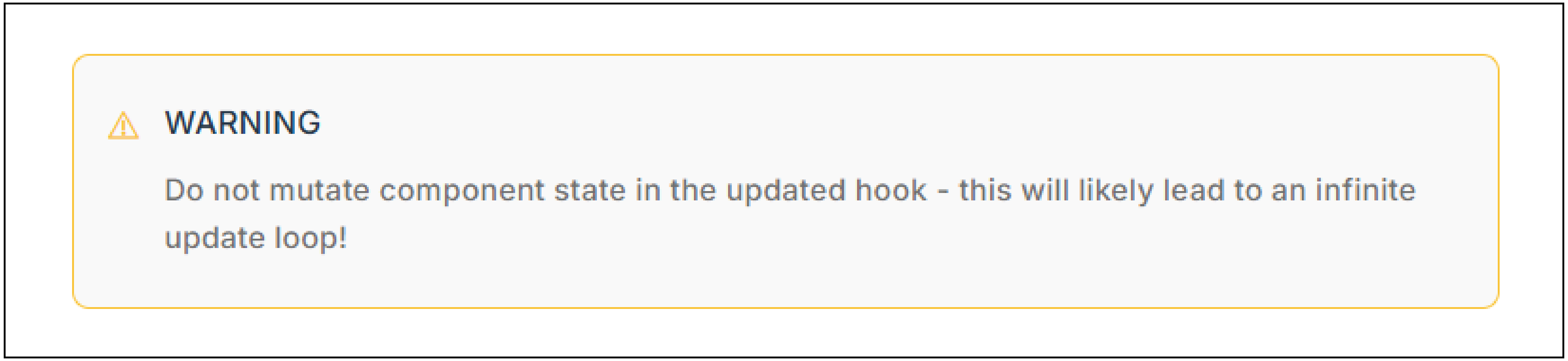


Update the component:

5

```
The component is mounted - lifecycle hook  VacationPicker.vue:43
called
The component is updated:: 0               VacationPicker.vue:49
Attempting initialization Fri Dec 13 content_script_bundle.js:1
2024 11:07:25 GMT+0100 (Midden-Europese standaardtijd)
The component is updated:: 1               VacationPicker.vue:49
The component is updated:: 2               VacationPicker.vue:49
The component is updated:: 3               VacationPicker.vue:49
The component is updated:: 4               VacationPicker.vue:49
The component is updated:: 5               VacationPicker.vue:49
>
```

## When NOT to update ref()’s


⚠ WARNING

Do not mutate component state in the updated hook - this will likely lead to an infinite update loop!

https://vuejs.org/api/composition-api-lifecycle.html

## Usage of lifecycle hooks

- Typical usage
  - `onMounted` – if you want to access or modify the DOM.
  - `onUpdated` – when the component receives new data from the outside (props)
    - Or, when the component variables (like a counter) are updated from within the component
  - `onUnmounted` – to destroy or garbage collect stuff that is not removed automatically
    - Vue 2: `destroyed()`

## Style Bindings

On using global styles and scoped styles

## Global styles and scoped styles

With default styles, CSS is globally available.

For instance, see default `assets/main.css`:

```css
@import './base.css';

#app {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  font-weight: normal;
}
…
```

```js
// main.js
import './assets/main.css'
```

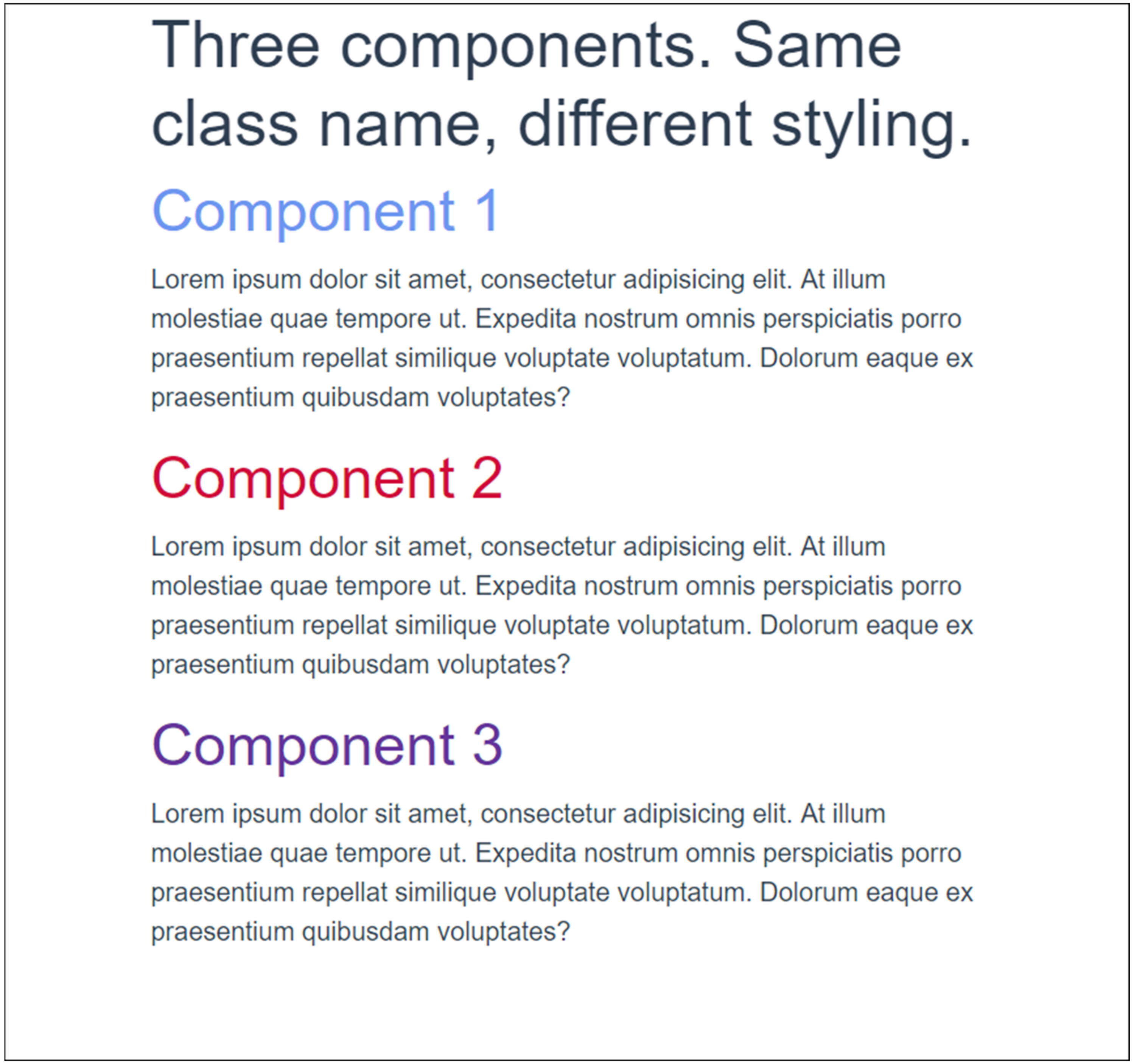
## Using scoped styles

- To avoid naming collisions, it is best to add the scoped attribute to a style block inside a component
- Different components now can reuse the same classname without clashes.

```
<template>
    <div>
        <h2 class="heading">Component 1</h2>
        …
    </div>
</template>

<script>
   export default {
      name: "ComponentOne",
   }
</script>

<style scoped>
    .heading {
        font-size: 36px;
        color: cornflowerblue;
    }
</style>
```

```
<h2 class="heading">Component 2</h2>
<style scoped>
    .heading {
        font-size: 36px;
        color: crimson;
    }
</style>
```

```
<h2 class="heading">Component 3</h2>
<style scoped>
    .heading {
        font-size: 48px;
        color: rebeccapurple;
    }
</style>
```



Three components. Same class name, different styling.

Component 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit. At illum molestiae quae tempore ut. Expedita nostrum omnis perspiciatis porro praesentium repellat similique voluptate voluptatum. Dolorum eaque ex praesentium quibusdam voluptates?

Component 2

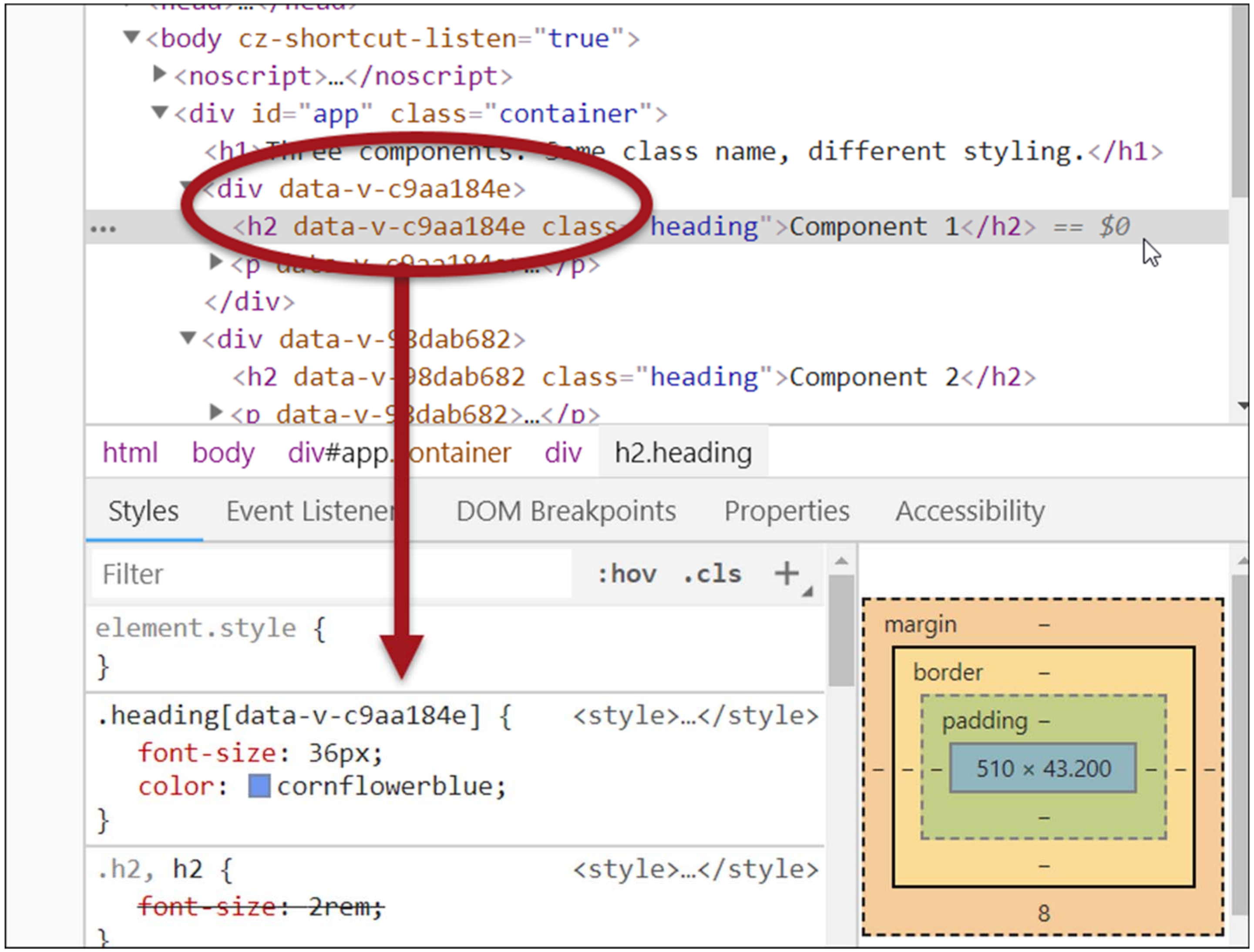
Lorem ipsum dolor sit amet, consectetur adipisicing elit. At illum molestiae quae tempore ut. Expedita nostrum omnis perspiciatis porro praesentium repellat similique voluptate voluptatum. Dolorum eaque ex praesentium quibusdam voluptates?

Component 3

Lorem ipsum dolor sit amet, consectetur adipisicing elit. At illum molestiae quae tempore ut. Expedita nostrum omnis perspiciatis porro praesentium repellat similique voluptate voluptatum. Dolorum eaque ex praesentium quibusdam voluptates?

../examples/120-scoped-styling

## Vue adds (semi random) hashes to elements



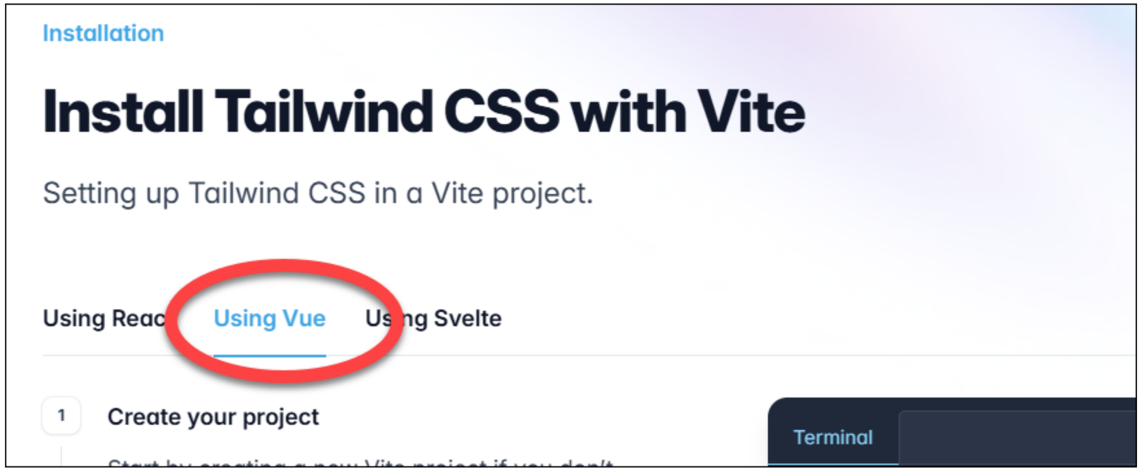
```
<body cz-shortcut-listen="true">
  <noscript>…</noscript>
  <div id="app" class="container">
    <h1>Three components. Same class name, different styling.</h1>
    <div data-v-c9aa184e>
      <h2 data-v-c9aa184e class="heading">Component 1</h2> == $0
      <p data-v-c9aa184e>…</p>
    </div>
    <div data-v-98dab682>
      <h2 data-v-98dab682 class="heading">Component 2</h2>
      <p data-v-98dab682>…</p>
```

html body div#app.container div h2.heading
Styles Event Listeners DOM Breakpoints Properties Accessibility
Filter :hov .cls +
margin –
border –
padding –
510 × 43.200
8

```
element.style {
}
.heading[data-v-c9aa184e] {          <style>…</style>
   font-size: 36px;
   color: cornflowerblue;
}
.h2, h2 {                            <style>…</style>
   ~~font-size: 2rem;~~
}
```

## General rules on styling

- Do not create global styles in components
- Only the top level bootsrapper (`main.js`) should import global styles
- You *can* use a generic CSS-framework like Bootstrap, Foundation, Vuetify, etc.
- See if there are *special instructions* available for your CSS-framework of choice


Installation

Install Tailwind CSS with Vite

Setting up Tailwind CSS in a Vite project.

https://tailwindcss.com/docs/guides/vite

## Conditionally applying styles

- Bind to the style attribute like so:
  - `v-bind:style=“{ …some-style…}”` or just
  - `:style=“{…some-style…}”`
  - For instance `:style=“{ border: ‘2px solid black’}”`
  - These are actually just CSS styles and notation!
- If your CSS-style has a hyphen in them, a special notation is needed:
  - `:style="{['background-color']: 'lightBlue’}”`
  - or use camelCase notation:
  - `:style="{backgroundColor: 'lightBlue'}"`

## Making the style conditional

- For instance: we only want the style to be applied if the cost of a trip is less than 1000
- We can just bind to the HTML `:style` property
- For the value: use a computed property, or method.
- Let the computed property or method return a valid CSS style object

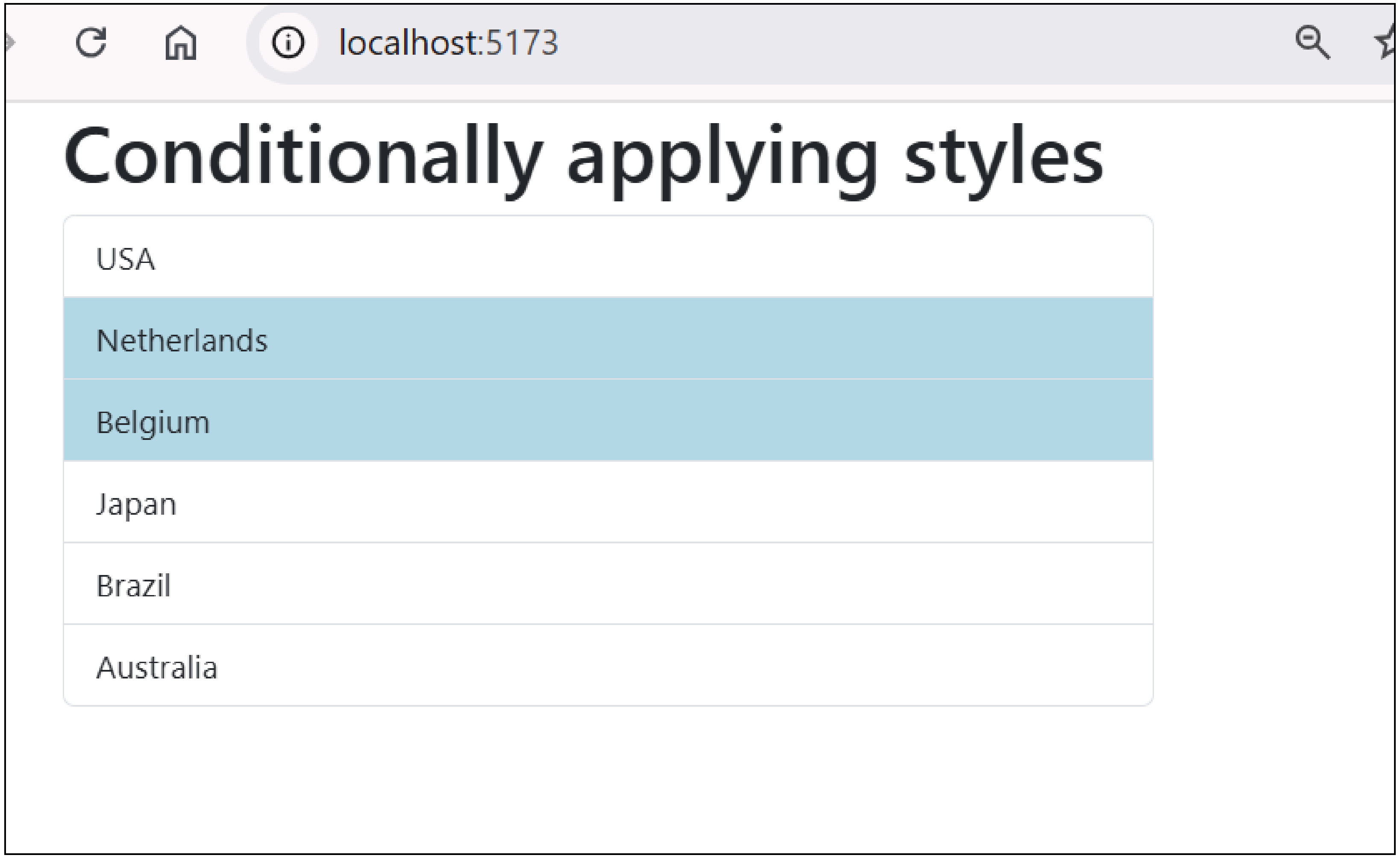
```
:style="{backgroundColor: 'lightBlue'}"
```

This works, but it is not conditional

This example: using a method

```
<li
    class="list-group-item"
    v-for="(country, index) in data.countries"
    :style="highlightBackground(index)"
    :key="country.id">
  {{ country.name }}
</li>
```

```
// Conditional style applied to countries:
const highlightBackground = (index) => {
  return {
    backgroundColor: data.countries[index].cost < 1000 ?
        'lightBlue' :
        'transparent'
  };
}
```



localhost:5173

Conditionally applying styles

USA

Netherlands

Belgium

Japan

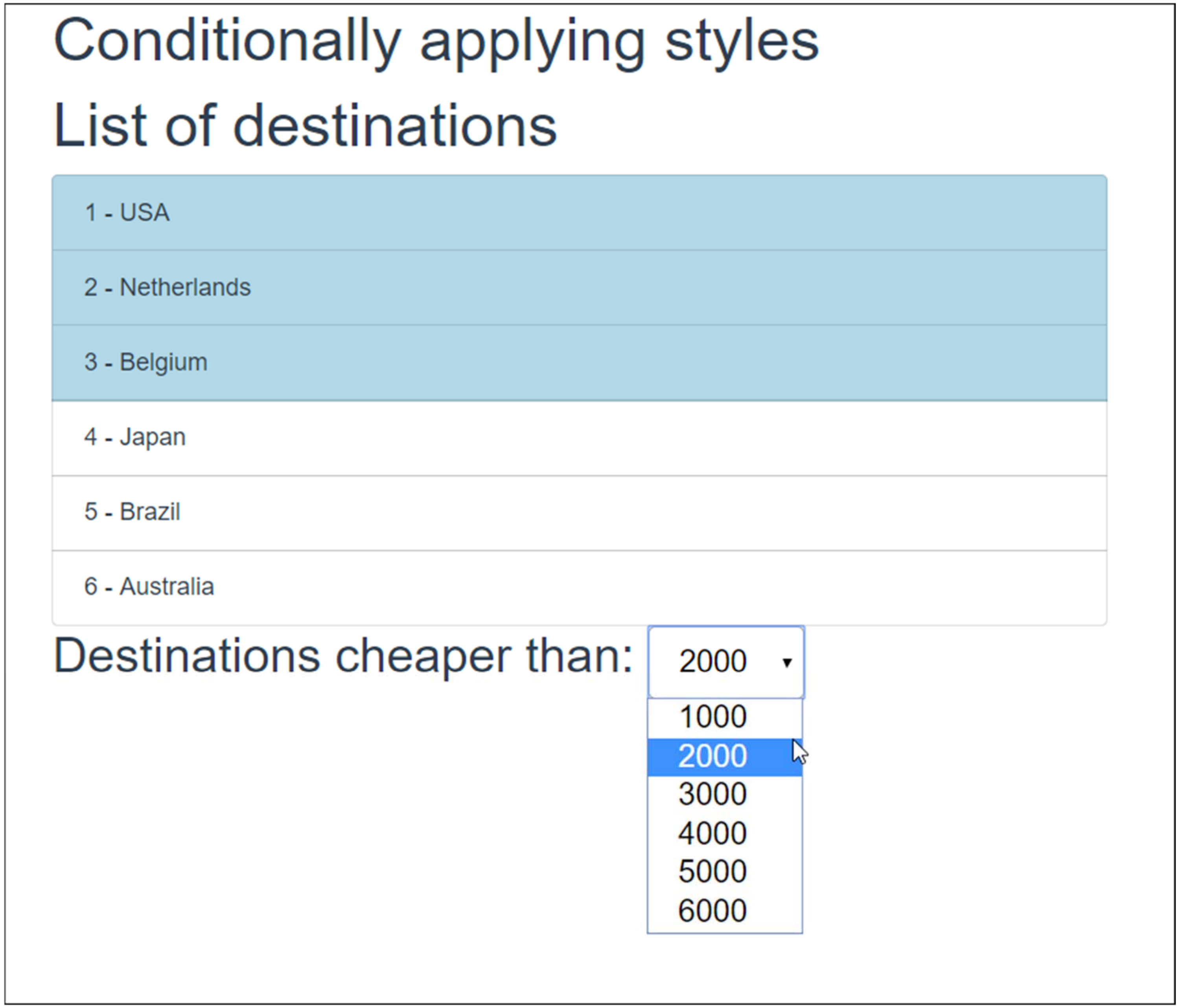
Brazil

Australia

## Using v-model on a selection list

```
<h2>Destinations cheaper than:
    <select class="form-control-lg" v-model="selectedCost">
        <option value="1000">1000</option>
        <option value="2000">2000</option>
        <option value="3000">3000</option>
        <option value="4000">4000</option>
        <option value="5000">5000</option>
        <option value="6000">6000</option>
    </select>
</h2>
```

```
const highlightBackground = (index) => {
  console.log('checking background color for....' + index);
  return {
    backgroundColor: data.countries[index].cost < selectedCost.value ?
        'lightBlue' :
        'transparent'
  };
}
```



## Conditionally applying styles

List of destinations

1 - USA

2 - Netherlands

3 - Belgium

4 - Japan

5 - Brazil

6 - Australia

Destinations cheaper than: 2000

## Conditionally applying classes

- However, often it is better to use CSS *classes* instead of inline styles
- Class binding is an object where the keys are the name of the CSS-class you want to toggle.
- You set the value to a boolean expression that should evaluate to `true` or `false`
  - If `true`, the class is applied
  - If `false`, the class is removed from the element
  - Of course this is all dynamic

## Same functionality – with class binding

Create a CSS class:

```
<style scoped>
    .lightblueBackground {
        background-color: lightblue;
    }
</style>
```

Apply the class conditionally in HTML:

```
:class="{'lightblueBackground': country.cost < selectedCost }"
```

## Workshop

- Create a component with a `<button>` and a `<div>`
- if the button is clicked, the class of the div is toggled
  - First – use conditional styles
  - Second – use conditional classes
- Add a `<div>`. If you hover the mouse over the div, toggle a class to highlight it
- Ready made example: `140…/…/ConditionalClass.vue`
  - (But first try it yourself!)