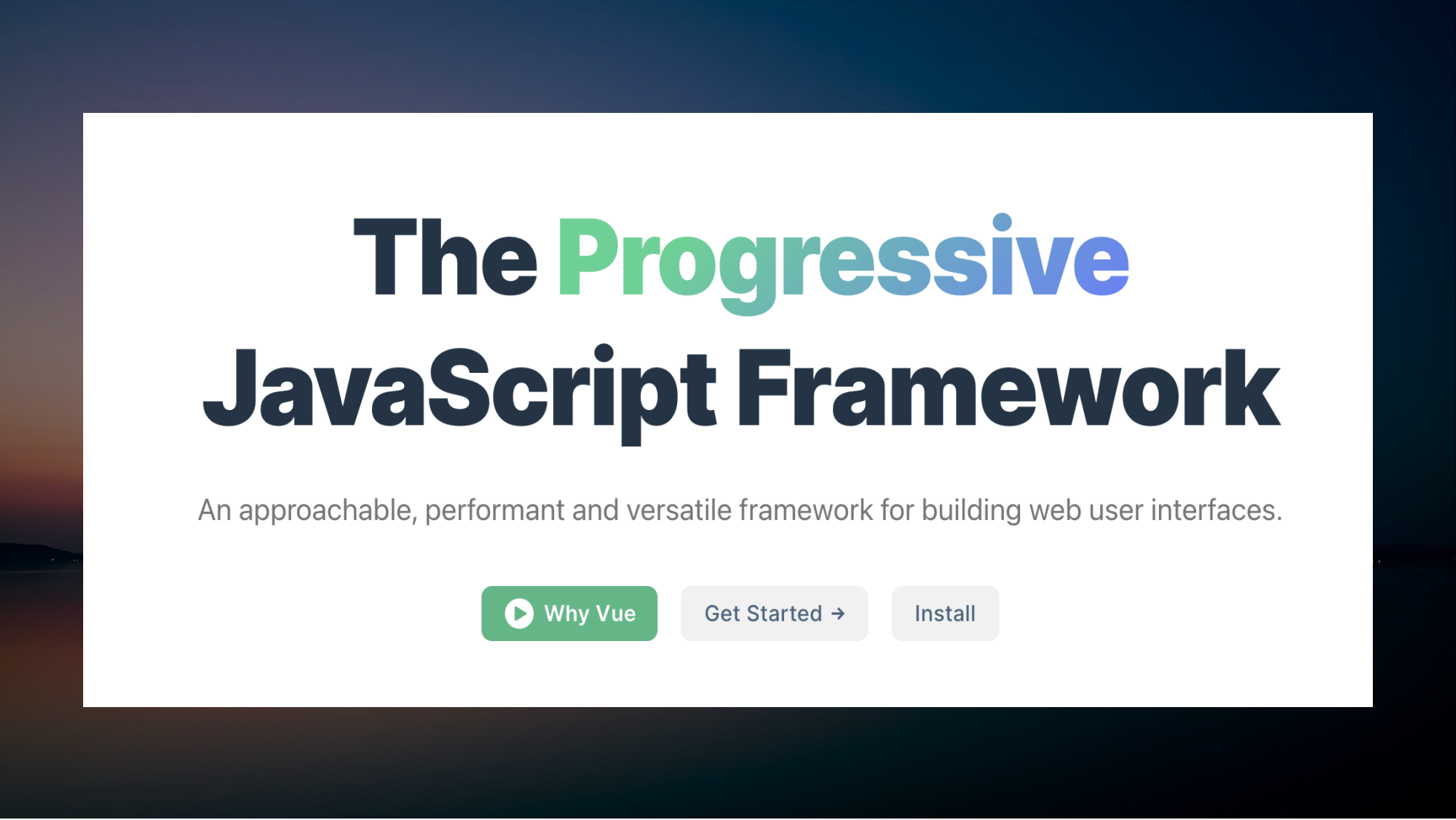
VueJS Fundamentals

by Peter Cosemans

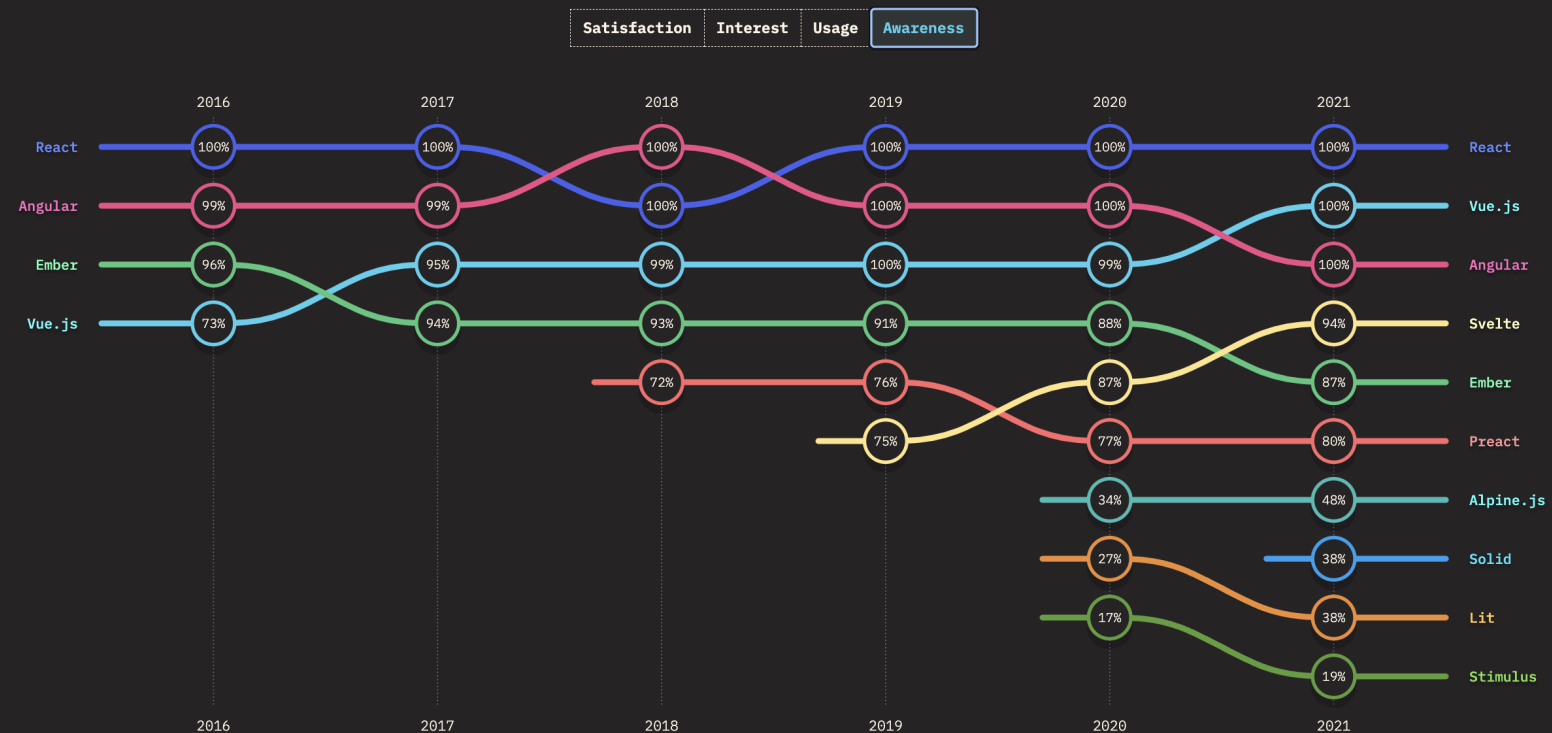

The Progressive
JavaScript Framework
An approachable, performant and versatile framework for building web user interfaces.
Why Vue
Get Started →
Install

Why Vue.js

It is a lightweight progressive framework, which is **easy to learn**, **extremely flexible**, and **simple to use**. Having **179,253 stars** on Github, Vue is one of the most popular Javascript frameworks, surpassing even React and Angular having 163,431 and 70,544 stars each.

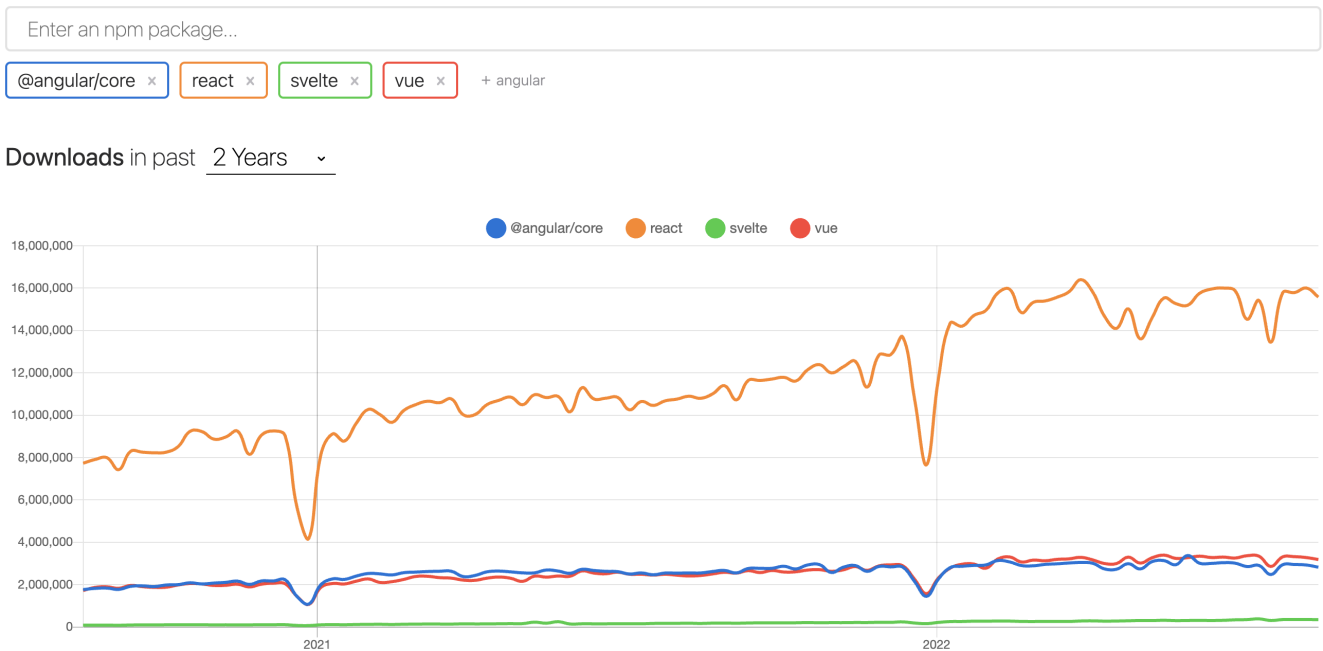
RANKINGS

Satisfaction, interest, usage, and awareness ratio rankings.

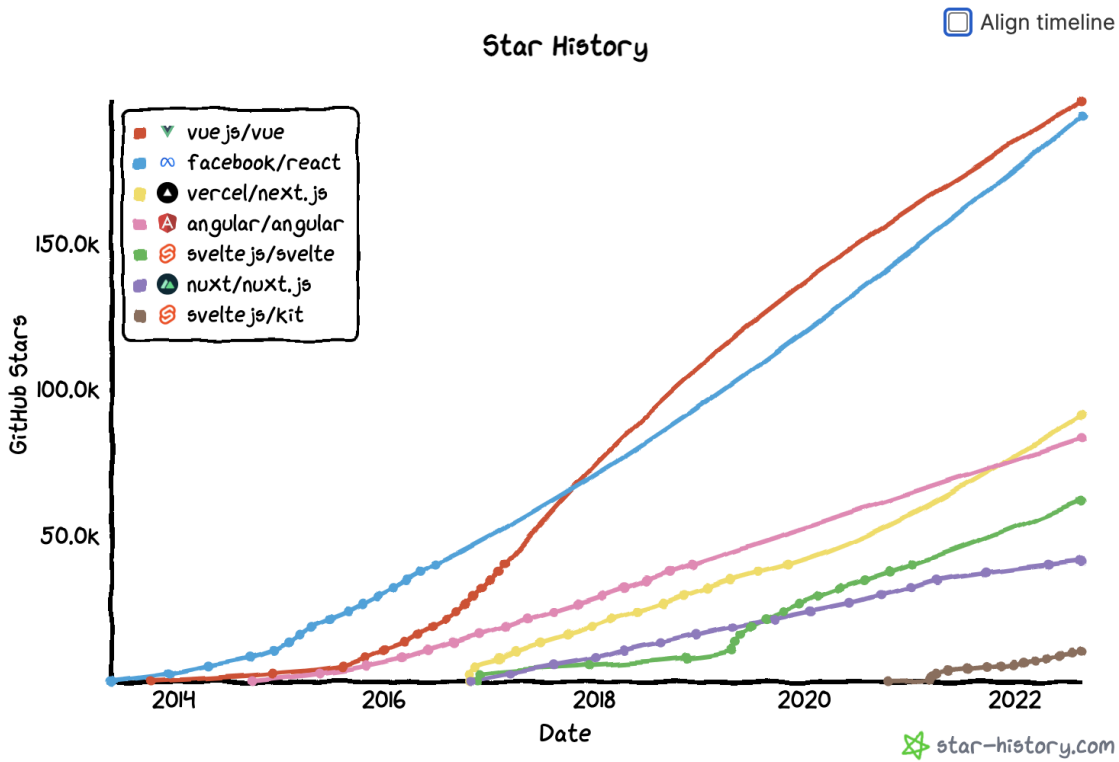


Npm downloads

@angular/core vs **react** vs **svelte** vs **vue**



Github Stars




Getting started

Documentation

https://vuejs.org/

Make sure you have the correct version

- Vue 2 documentation
- Vue 3 documentation

Training

- Vue Mastery
- Vue School

Related

- awesome-vue

Manual setup 1/4

Start new project

```
npm init --force
```

Install dependencies

```
npm install vue
npm install typescript vite @vitejs/plugin-vue --save-dev
```

Manual setup 2/4

Create vite setup (vite.config.js)

```
import { defineConfig } from "vite";
import Vue from "@vitejs/plugin-vue";

export default defineConfig({
  plugins: [Vue()],
});
```

index.html

```
<!DOCTYPE html>
<html>
  <body>
    <script src="./src/main.js" type="module"></script>
  </body>
</html>
```

Manual setup 3/4

Create the app

```
import { createApp } from 'vue'
// import the root component App.
import App from './App.vue'

const app = createApp(App)
```

And mount it

```
<body>
  <div id="app"></div>
  <script src="./src/main.js" type="module"></script>
</body>
```

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

Manual setup 4/4

App.vue

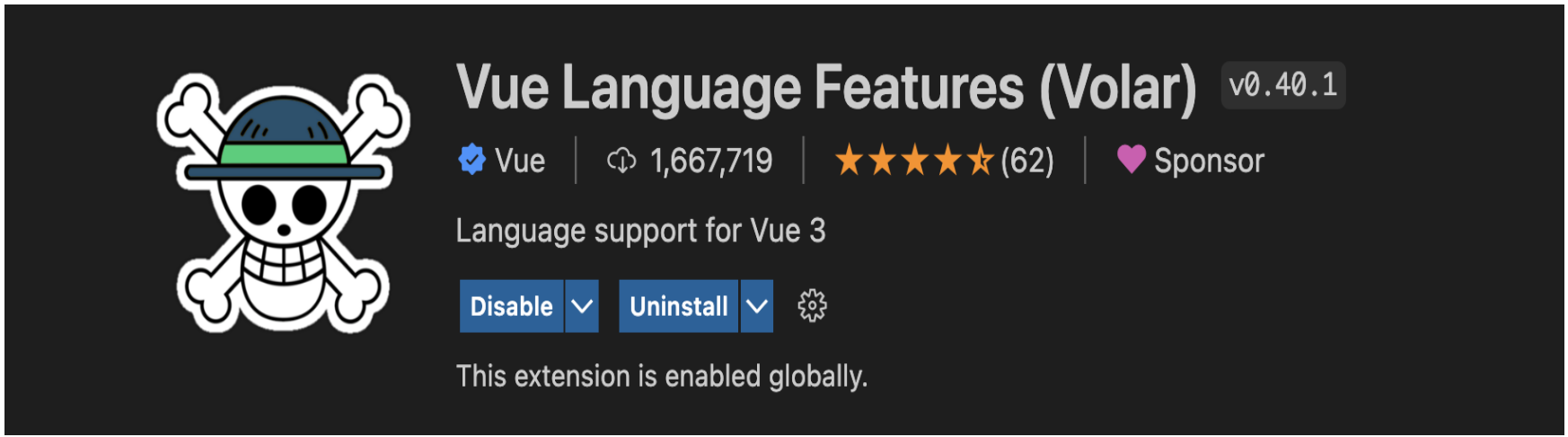
```
<script setup>
const greeting = 'Hello Vue';
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>
```

And start the dev server

```
npm run dev
```

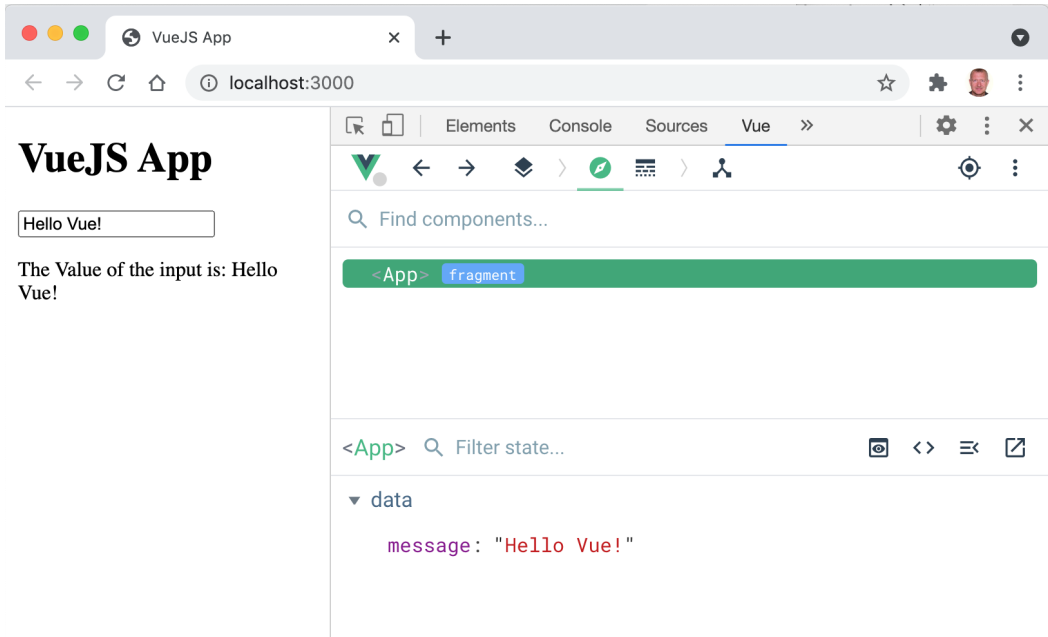
Debug tools 1/2

VSCode 'Vue Language Features (Volar)' extension



- Color syntax highlighting
- Typescript support
- Etc ...

Debug tools 2/2



Vue.js devtools

Quick start with VueCLI (vue2)

```
# install Vue CLI
install -g @vue/cli

# create a new project
vue create hello-vue2

# open and run
cd hello-vue2
npm run serve
```

Vue CLI is more or less outdated, but it’s still a good tool to get started with Vue 2.

Quick start with 'create-vue'

```
# install and execute create-vue
npm init vue@latest
```

```
✔ Project name: … <your-project-name>
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit testing? … No / Yes
✔ Add Cypress for both Unit and End-to-End testing? … No / Yes
✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

The now, official Vue project scaffolding tool.

See docs for more info. https://vuejs.org/guide/quick-start.html#local

Quick start with 'vue-starter-kit'

```
./vue-starter-key/
```

Features

- ⚡ Vue 3, Vite 3, pnpm, ESBuild - born with fastness
- 🍍 State Management via Pinia
- 🎨 Tailwindcss
- 🔥 Use the new `<script setup>` syntax
- 👍 Reactivity Transform enabled
- 🦾 TypeScript, of course
- ⚙️ Unit Testing with Vitest


The structure of a Vue App

Single-File Components (SFC)

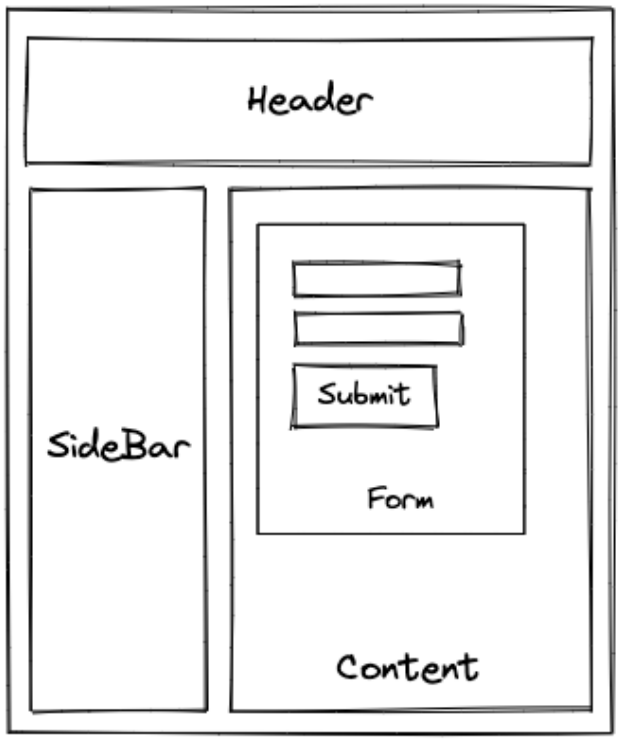
App.vue

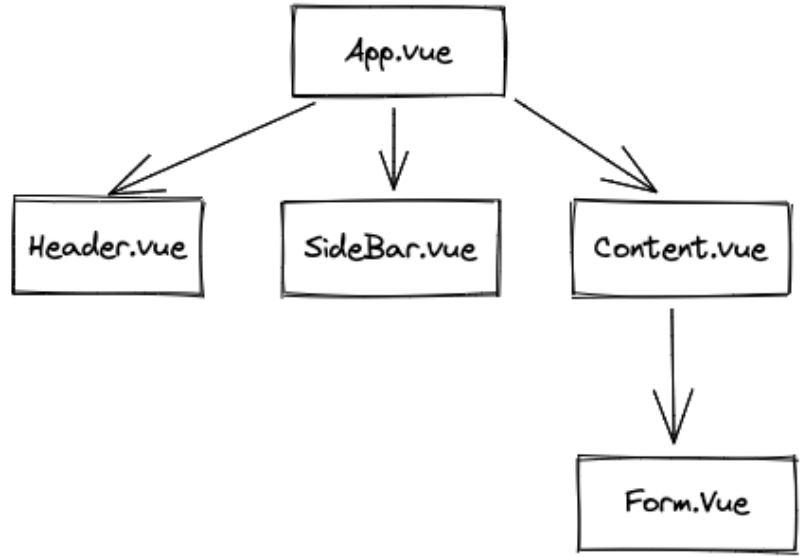
```vue
<script setup>
const greeting = 'Hello World';
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<style>
.greeting {
  color: red;
  font-weight: bold;
}
</style>
```

You can specify the language of your choice

```
<script setup lang="ts">
  const greeting:string = 'Hello World';
</script>

<template lang="pug">
  div
    p Hello World
</template>

<style lang="scss">
  $font-stack: Helvetica, sans-serif;
  $primary-color: #030303;

  .box {
    font: 100% $font-stack;
    color: $primary-color;
  }
</style>
```

A composition of components





A composition of components

App.vue

```vue
<script setup>
import Header from './components/Header.vue'
import SideBar from './components/SideBar.vue'
import Content from './components/Content.vue'
</script>

<template>
  <div class="app">
    <Header />
    <div class="container">
      <SideBar />
      <Content />
    </div>
  </div>
</template>
```

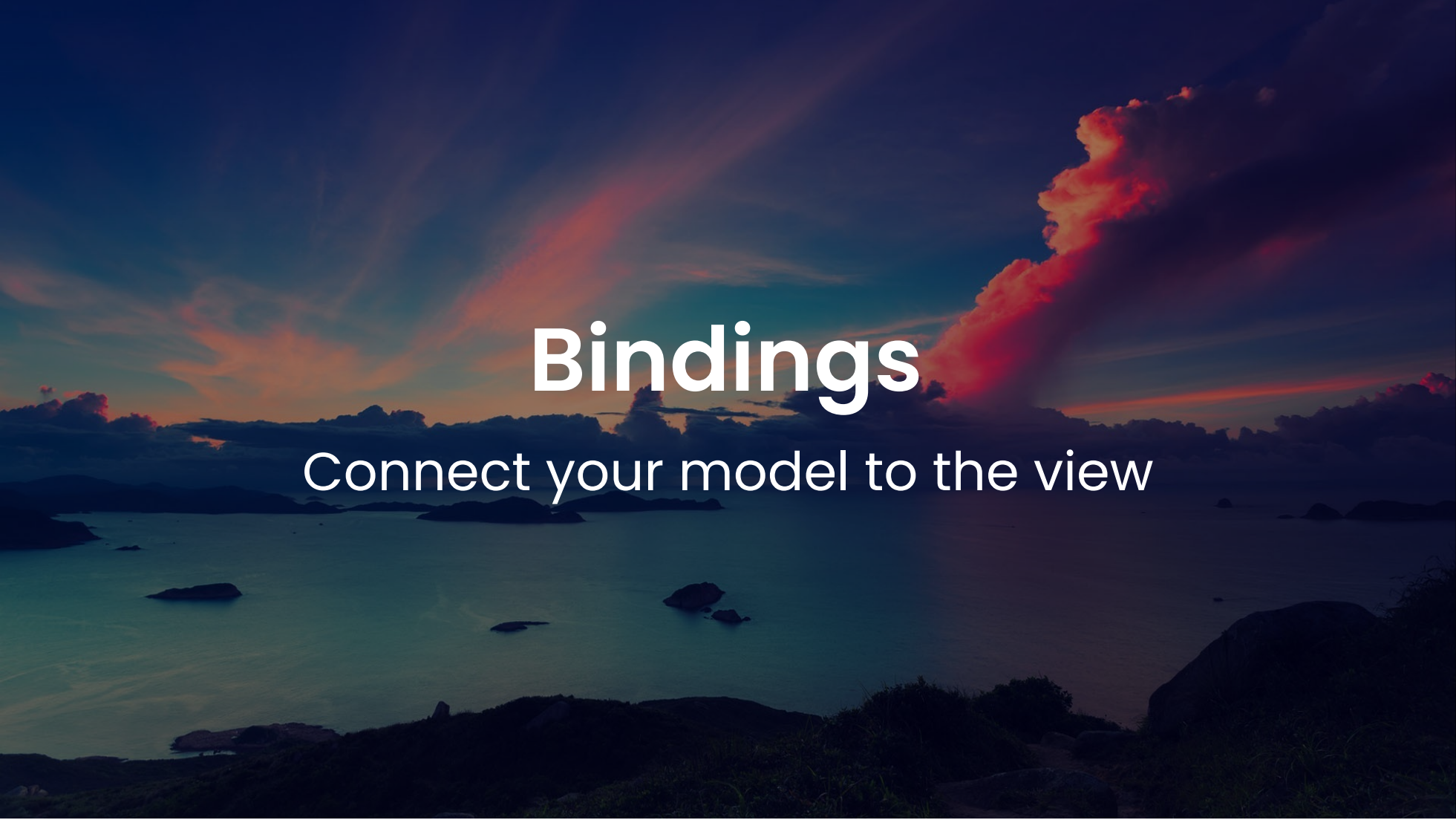
A composition of components

Header.vue

```
<template>
  <h1>header</h1>
</template>
```

SideBar.vue

```
<template>
  <h1>header</h1>
</template>
```


Bindings
Connect your model to the view

Value Binding

```
<script setup>
const message = 'Hello from VueJS'
</script>
```

Simple binding (mustaches)

```
<p>
  The Value of the input is:
  <span>{{ message }}</span>
</p>
```

v-text directive

```
<p>
  The Value of the input is:
  <span v-text="message"></span>
</p>
```

Template Expressions

The 'Mustaches' syntax can contain any Javascript expressions.

```
<span>{{ number + 1 }}</span>

<h2>{{ ok ? 'YES' : 'NO' }}<h2>

<div class="alert">
  {{ message.split(' ').reverse().join('.') }}
</div>
```

But template expressions are sandboxed
and only have access to a whitelist of globals such as Math and Date

Attributes binding

'Mustaches' cannot be used inside HTML attributes Instead, use a v-bind directive:

```
<script setup>
  const isButtonDisabled = true;
</script>

<template>
  <button v-bind:disabled="isButtonDisabled">
    Button
  </button>
</template>
```

Shorthand syntax `:`

```
<button :disabled="isButtonDisabled">
  Button
</button>
```

Event Binding

Used for binding to events like click, mousemove & etc...

```
<template>
  <button v-on:click="handleClick">ClickMe</button>
</template>

<script setup>
  const handleClick = () => {
    console.log('clicked')
  }
</script>
```

Shorthand syntax @

```
<button @click="handleClick">
  Inc
</button>
```

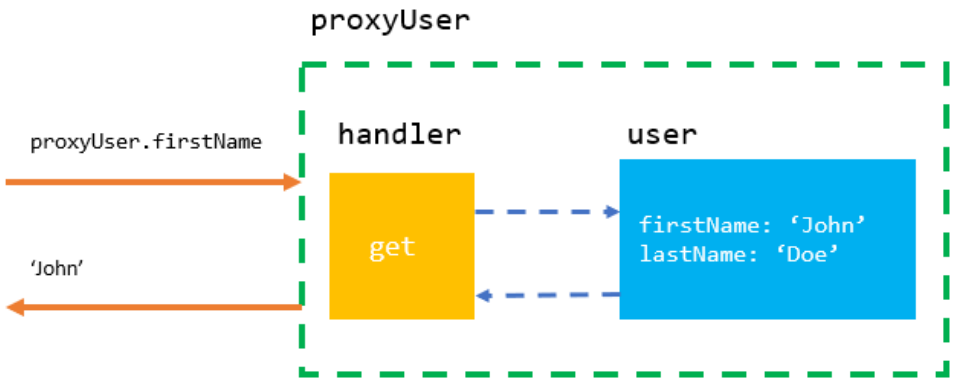
Reactivity Fundamentals

Response to changes

Declaring Reactive State

```
<script setup>
import { reactive } from 'vue';

const state = reactive({ count: 0 });
const handleClick = () => {
  state.count++;
};
</script>

<template>
  <div class="flex gap-3 m-10">
    <button class="btn" @click="handleClick">
      normal
    </button>
    counter: {{ state.count }}
  </div>
</template>
```

The proxy 1/2



The returned value from reactive() is a Proxy of the original object, which is not equal to the original object

```
1  const raw = {}
2  const proxy = reactive(raw)
3
4  // proxy is NOT equal to the original.
5  console.log(proxy === raw) // false
```

The proxy 2/2

```
const diner = {
  meal: 'burger',
};
const handler = {
  get(target, prop) {
    console.log('intercepted get');
    return target[prop];
  },
  set(target, prop) {
    console.log('intercepted set');
    return target[prop];
  },
};
const proxy = new Proxy(diner, handler);
console.log(proxy.meal);
```

Limitations of reactive()

1. It only works for object types (objects, arrays, and collection types such as Map and Set). It cannot hold primitive types such as string, number or boolean.

```
import { ref } from 'vue'

// reactive primitive variable
const count = ref(0)

// access value
count.value++
console.log(count.value) // 1
```

2. Destructuring will lose the reactivity

```
const state = reactive({ count: 0 });

// state is NOT reactive
const { count } = state;
```

Ref Unwrapping

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

const count = ref(0)

function increment() {
  count.value++
}
</script>

<template>
  <button @click="increment">
    {{ count }} <!-- no .value needed -->
  </button>
</template>
```

Directives

Extend the behavior of existing elements

What's a directive

A directive is some special token in the markup
that tells the library to do something to a DOM element.

```
1   <div v-text="message"></div>
```

Some directives can take an "argument",
denoted by a colon after the directive name

```
1   <a v-bind:href="url"> ... </a>
2   <a v-on:click="doSomething"> ... </a>
```

Directives is the glue of VueJS templates

v-text

Updates the textContent of the element.

```
<span v-text="message"></span>
```

`{{ message }}` is a shortcut for the `v-text` directive.

```
<span>{{ message }}</span>
```

v-text

Can use string, number, boolean, object & array.

```
<script script>
  const counter = ref(1);
  const user = reactive({ name: 'Peter', id: '123456' })
  const hasAccess: ref(false);
</script>
<template>
  <div>{{ counter }}</div>
  <div>{{ hasAccess }}</div>
  <code>
    <pre>
      {{ user }}
    </pre>
  </code>
</template>
```

v-bind

Dynamically bind one or more attributes,
or a component prop to an expression.

```
1  <img v-bind:src="imageSrc" /> <br />
2  <img :src="imageSrc" /> <br />
3  <div :class="myClassName"></div>
```

v-on

Bind to an event

```
<template>
  <button v-on:click="handleClick">Click</button>
  <button @click="handleClick">Click</button>
  <button @click="handleOtherClick(15, $event)">Click</button>
</template>

<script setup>
const handleClick(event) {
  console.log('clicked', event)
}
const handleOtherClick(arg, event) {
  console.log('clicked', arg, event)
}
</script>
```

v-on

```
<template>
  <form @submit="submit($event)">
    <input type="text" v-model="name" />
  </form>
</template>

<script script>
const submit = (event) => {
  event.preventDefault();
  console.log('submit');
}
</script>
```

'event.preventDefault' is required to avoid the default browser behavior (post and reload)

v-on modifiers

```
<form @submit.prevent="submit($event)">
  <input type="text" v-model="name" />
</form>
```

Other modifiers

modifier	description
@submit.prevent	call event.preventDefault()
@mousemove.stop	call event.stopPropagation()
@keyup.112	only react on keyCode (deprecated in v3)
@keyup.enter	only react on 'enter' key

More see doc's

v-view, v-hide

Are conditionals that will hide/display information (html elements).
`v-show` toggles the display CSS property of the element.

```
1  <span v-show="toggle">Just some text </span>
2
3  <span v-hide="!error">Error: {{ error.message }}</span>
```

v-if, v-else

Is a conditional that will render/not render information.

```
<span v-if="toggle">Just some text </span>
<span v-else>Other text</span>
```

Conditional group

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

See also `v-elseif`

v-for

Render the template block multiple times

```
<script setup>
  const users: [
    {id: 1, name: 'Joe'}
    {id: 2, name: 'Mary'}
    {id: 3,  name: 'Jane'}
  ];
</script>
<template>
  <ul>
    <li v-for="user in users" :key="user.id">
      {{user.name}}
    </li>
  </ul>
</template>
```

v-model

Relation between model and form input

```
<script setup>
  import { ref } from 'vue'
  const message = ref('Hello World');
  const toggle = ref(false);
</script>

<template>
  <input type="text" v-model="message" />
  <input type="checkbox" v-model="toggle" />
  <span>Message: {{ message }} - {{ toggle }}</span>
</template>
```

This is real 2 way binding on every input event

v-model modifiers

```
1  <template>
2    <input type="text" v-model.trim="message" />
3  </template>
```

modifier	description
v-model.trim	strip leading or trailing whitespace
v-model.number	change string to number
v-model.lazy	model is updated after change events

v-html

The double mustaches interprets the data as plain text, not HTML.

```
const rawHtml = '<strong>Info</strong><br/>Error Message'
```

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

Dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily lead to XSS vulnerabilities. Only use HTML interpolation on trusted content and never on user-provided content.

Directives Overview 1/2

Directive	Description	Sample
v-text	Updates the element's textContent	`<span v-text="msg" />`
v-bind	Dynamically bind one or more attributes	`<img v-bind:src="imageSrc" />`
v-on	Attaches an event listener to the element	`<button v-on:click="doThis"> </button>`
v-if	Conditionally render the element	`<span v-if="test">Show Me</span>`
v-show	Conditionally render the element	`<span v-show="test">Show Me</span>`

Directives Overview 2/2

Directive	Description	Sample
v-for	Render the template block multiple times	`<div v-for="item in items"> {{item.text }}<div>`
v-model	Create a two-way binding on a form input element	`<input type="text" v-model="user" />`
v-html	Updates the element's innerHTML	`<div v-html="html"></div>`


Style binding and css

Global styling

Just import the css file

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

// load external css
import 'bootstrap/dist/css/bootstrap.css'

// load app styling
import './main.css';
// or import './main.scss';
// or import './main.less';
// or import './main.stylus';

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

```
# install dependency
npm install bootstrap
```

Local CSS

Place you css in the `style` tag

```
<template>
  <div class="classroom">...</div>
</template>
<style>
  .classroom {
    background: #232323;
  }
</style>
```

You can specify the style language (less, sass, stylus)

```
<style lang="scss">
  $bg-classroom: #232323;
  .classroom {
    background: $bg-classroom;
  }
</style>
```

Scoped style

When a `<style>` tag has the `scoped` attribute, its CSS will apply to elements of the current component only.

```
<template>
  <h1>Component Title</h1>
</template>

<style>
  /* global styles */
</style>

<style scoped>
  /* scoped styles */
  h1 {
    color: red;
  }
</style>
```

Class Bindings

```vue
<script setup>
  const isActive = ref(true);
  const activeClass = ref('active')
  const errorClass = ref('text-danger')
</script>

<template>
  <!-- static class & bind with object -->
  <div class="box" :class="{ active: isActive }"></div>

  <!-- binding to array -->
  <div :class="[activeClass, errorClass]"></div>
</template>
```

Result

```vue
<div class="box active"></div>
<div class="active text-danger"></div>
```

Exercise

Use a button to toggle (hide/show) a paragraph of text

- Look for multiple solutions


Computed Properties &
Watchers

Computed properties

Are calculation based on the state. We also call this the derived state. Computed properties are cached and will only update when state changes.

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

const users = ref(['peter', 'john', 'jan', 'pieter']);
const filter = ref('');
const filteredUsers = computed(() => {
  return users.value.filter((item) => {
    return item.startsWith(filter.value);
  });
});
</script>

<template>
  Filter: <input v-model="filter" type="text" class="w-80">
  <code>{{ filteredUsers}}</code>
</template>
```

Watchers

Gets called when data changes.

```
1  import { ref, watch } from 'vue';
2
3  const counter = ref(0);
4  const increment = () => {
5    counter.value += 1;
6  };
7
8  watch(counter, (newValue, oldValue) => {
9    console.log(newValue, oldValue);
10 });
```

Watchers

Be aware of reactive objects

```
const obj = reactive({ count: 0 })

// this won't work because we are passing a number to watch()
watch(obj.count, (count) => {
  console.log(`count is: ${count}`)
})

// instead, use a getter:
watch(
  () => obj.count,
  (count) => {
    console.log(`count is: ${count}`)
  }
)
```

See more in docs

Exercise

Create a small TODO list

- Show list of tasks
- Text `input` to enter a task
- Press `enter` key to add the task
- Add buttons (per task) to remove task
- Show the total count of tasks
- Apply some style


Vue API Styles
Which one to use

Option API

Vue2

```vue
<script>
export default {
  data() {
    return {
      name: 'John Doe',
      age: 30,
    }
  },
  methods: {
    handleClick() { ... }
  }
}
</script>
```

Composition API

Vue3 (backported to Vue2.7)

```vue
<script>
import { defineComponent, ref } from 'vue'
export default defineComponent({
  setup() {
    const name = ref('John Doe');
    const age = ref(30);

    const handleClick = () => {
      ...
    }
    return {
      name,
      age,
      handleClick
    }
  }
})
</script>
```

script

Vue 3.0

```vue
<script>
import { defineComponent, ref } from 'vue'
export default defineComponent({
  setup() {
    const name = ref('John Doe');
    const age = ref(30);

    const handleClick = () => {
      ...
    }
    return {
      name,
      age,
      handleClick
    }
  }
})
</script>
```

script setup

Vue 3.2+

```vue
<script setup>
import { ref } from 'vue'
const name = ref('John Doe');
const age = ref(30);

const handleClick = () => {
  ...
}
</script>
```


Components In-Depth

Props

Person.vue

```vue
<script setup>
  const props = defineProps({
    name: String,
    age: Number,
    isActive: Boolean,
  })
</script>

<templates>
  <div>
    Name: {{ name}}
    Age: {{ age }}
  </div>
</templates>
```

App.vue

```vue
<template>
  <!-- static props -->
  <Person name="John Doe"
          is-active />

  <!-- dynamic props -->
  <Person :name="user.name"
          :age="user.age"  />
</template>
```

Props are direct available in `<template>`

Props Validation

It’s always good to add additional props validation

```
<script setup>
  defineProps({
    // Basic type check
    propA: Number,
    // Multiple possible types
    propB: [String, Number],
    // Required string
    propC: {
      type: String,
      required: true,
    },
    // Number with a default value
    propD: {
      type: Number,
      default: 100,
    },
  });
</script>
```

Props - Typescript

```vue
<script setup lang="ts">
  interface PersonProps {
    propA?: number;
    propB?: string | number;
    propC: string;
    propD?: number;
  }
  const props = defineProps<PersonProps>();

  // with default value
  // const props = withDefaults(defineProps<PersonProps>(), {
  //   propD: 100,
  // }
</script>
```

Events

You can emit an event from the child component to the parent.

```
<script setup>
const emit = defineEmits(['inFocus', 'selected'])

function handleClick() {
  emit('selected')
}
</script>
```

Usage (as any other attr binding)

```
<MyComponent @selected="handleSelect"></MyComponent>
```

Event - Typescript

```vue
<script setup lang="ts">
  const emit = defineEmits<{
    (e: 'change', id: number): void
    (e: 'update', value: string): void
  }>()

  function handleClick(id) {
    // you can pass any argument to the event
    emit('change', id)
  }
</script>
```

Lifecycle events

```
1  import { onMounted, onUpdated, onUnmounted } from 'vue';
2
3  // Element is created
4
5  onMounted(() => {
6    // Element is mounted in the html
7  });
8
9  onUpdated(() => {
10   // Called after a data change
11 });
12
13 onUnmounted(() => {
14   // Called after a Vue instance has been destroyed
15 });
```

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

Slots

Using the `<slot>` to pass the inner content from the parent to the child

```
<!-- MyButton.vue -->
<template>
  <button class="btn btn-primary">
    <slot></slot>
  </button>
</template>
```

```
<MyButton>
  <strong>Add todo</strong>
</MyButton>
```

result

```
<!-- rendered HTML -->
<button class="btn btn-primary">
  <strong>Add todo</strong>
</button>
```

Named Slots

```vue
<!-- MyComponent.vue -->
<template>
  <div class="container">
    <header>
      <slot name="header"></slot>
    </header>
    <main>
      <slot>Default content</slot>
    </main>
  </div>
</template>
```

Specify the name of the slot

```vue
<MyComponent>
  <h1 v-slot:header>Page title</h1>
  <p>the main content.</p>
</MyComponent>
```

Shorthand syntax: '#header'

Exercise

Create dismissible bootstrap alert component

- Use bootstrap styling:

  http://getbootstrap.com/components/#alerts-dismissible
- Create VueJS component

```
<!-- default alert: warning -->
<Alert> Almost out of stock </Alert>

<!-- custom alert with event -->
<Alert type="alert" @closed="handleClosed" closable>
  <strong>Alert!</strong> We have a problem.
</Alert>
```

- Don’t use jquery or the bootstrap js library
- Log a message to the console if the dialog is closed


Data Access

Using Axios

```vue
<template>
  <ul>
    <li v-for="user of users">{{user.name}}</li>
  </ul>
</template>

<script>
  import axios from 'axios';
  import { ref, onMounted } from 'vue';

  const users = ref([]);

  onMounted(async () => {
    const res = await axios.get('./users.json');
    users.value = res.data;
  }
</script>
```

Using Fetch

```
<template>
  <ul>
    <li v-for="user of users">{{user.name}}</li>
  </ul>
</template>

<script>
  import { ref, onMounted } from 'vue';

  const users = ref([]);

  onMounted(() => {
    fetch('./users.json')
      .then((res) => res.json())
      .then((users) => {
        users.value = users;
      });
  });
</script>
```

Exercise

> Build an app to show a list of users

- Use 'https://euricom-test-api.herokuapp.com' to get a list of products
- Use 'fetch' api or 'axios' to access data
- Use bootstrap for styling
- Show image, title, sku and price in table format.
- Optional
  - Provide load more, paging or infinite scrolling
  - Make the headers clickable to sort rows

What’s next

- Routing: Vue-router
- Form Validation: (vee-validate & Zod)
- State Management (pinia)
- Style: Vue.js Component Style Guide