knowit

Introduction to Declarative DOM Manipulation

Using Vue.js

knowit

Hi, I’m Misa Jokisalo

… and the cat is Myy.

- Actually a math and computer science teacher.
- Board games, video games and DnD.
- Web developer for 6+ years.
- Working at Knowit since 2021.

knowit

- Nordic consultancy.
- 500 people in Finland.
- Code, data, cloud and quality.

knowit

Makers of a sustainable future

knowit

Agenda

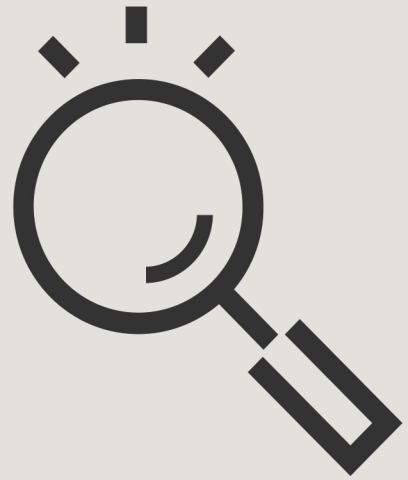
Introduction to Declarative DOM Manipulation

1. Defining Declarative
2. Vue.js
3. Workshop

knowit

Introduction to Declarative DOM Manipulation

Defining Declarative



A programming paradigm that
***expresses the logic** of a computation*
***without describing its control flow**.*

knowit

Defining Declarative

- **Declarative** code describes what we want.
- *“I want a strawberry milkshake.”*

- **Imperative** code describes how to do things.
- *“Blend frozen strawberries and milk.*
  Add ice cream and blend.
  Pour into a glass and serve.”

Declarative programming is a luxury made possible by tons of imperative code.

Example

how 2 code

In this video I teach you how to code. Please like and subscribe.

Download

User clicks button

how 2 code

In this video I teach you how to code. Please like and subscribe.

Download

Imperative

- Manually describe the actions to perform.

```
<button
  id="download-button"
  onClick="startDownload()"
>
  Download
</button>
```

```
const startDownload = async () => {
  // Find the button element
  const button =
    document.querySelector('#download-button');

  // Disable the button
  button.setAttribute('disabled', '');

  // Download something
  await downloadSomeFile();

  // Enable the button
  button.removeAttribute('disabled');
}
```

Declarative (Vue.js)

- Describe the desired behavior.

```
<button
  :disabled="isLoading"
  @click="startDownload"
>
  Download
</button>
```

Reactive

```
const isLoading = ref(false);

const startDownload = async () => {
  isLoading.value = true;

  await downloadSomeFile();

  isLoading.value = false;
}
```

Reactive Programming

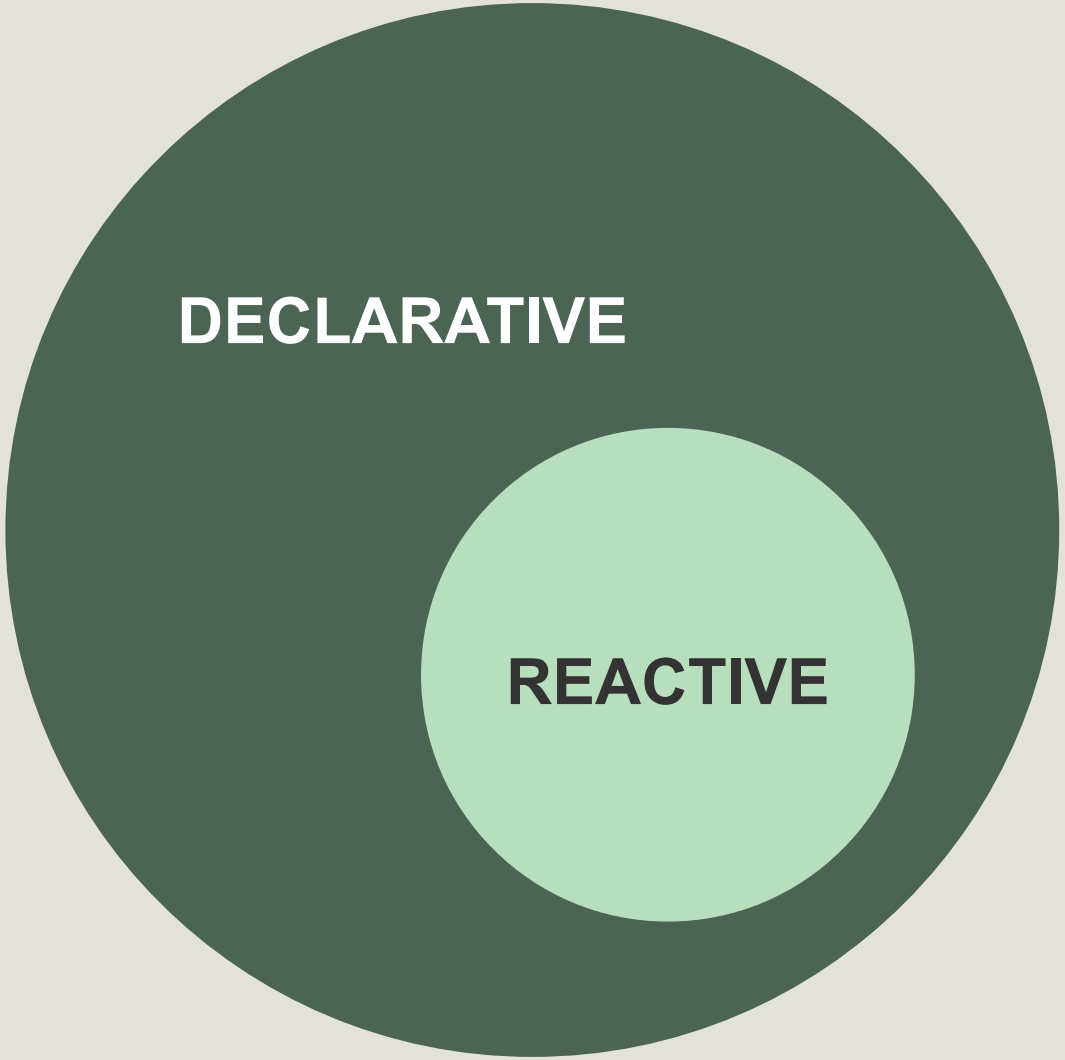
The declarative expression of the relationship between values that change over time.

$$a = b + c$$

knowit

Reactive Programming

The declarative expression
of the relationship between values
that change over time.



Reactive Programming

The declarative expression
of the relationship between values
that change over time.

```
// Vanilla JS
let b = 1;
let c = 2;

const a = b + c;

console.log(a); // Output: 3

b = 10;

console.log(a); // Output: 3
```

Reactive Programming

The declarative expression
of the relationship between values
that change over time.

```
// Reactive
let b = 1;
let c = 2;

const a = b + c;

console.log(a); // Output: 3

b = 10;

console.log(a); // Output: 12
```

knowit

Reactive Introduction to ~~Declarative~~ DOM Manipulation

Using Vue.js

knowit

Introduction to Declarative DOM Manipulation

Vue.js

vuejs.org

knowit

Vue.js

- JavaScript web UI framework.
- Build reactive applications.

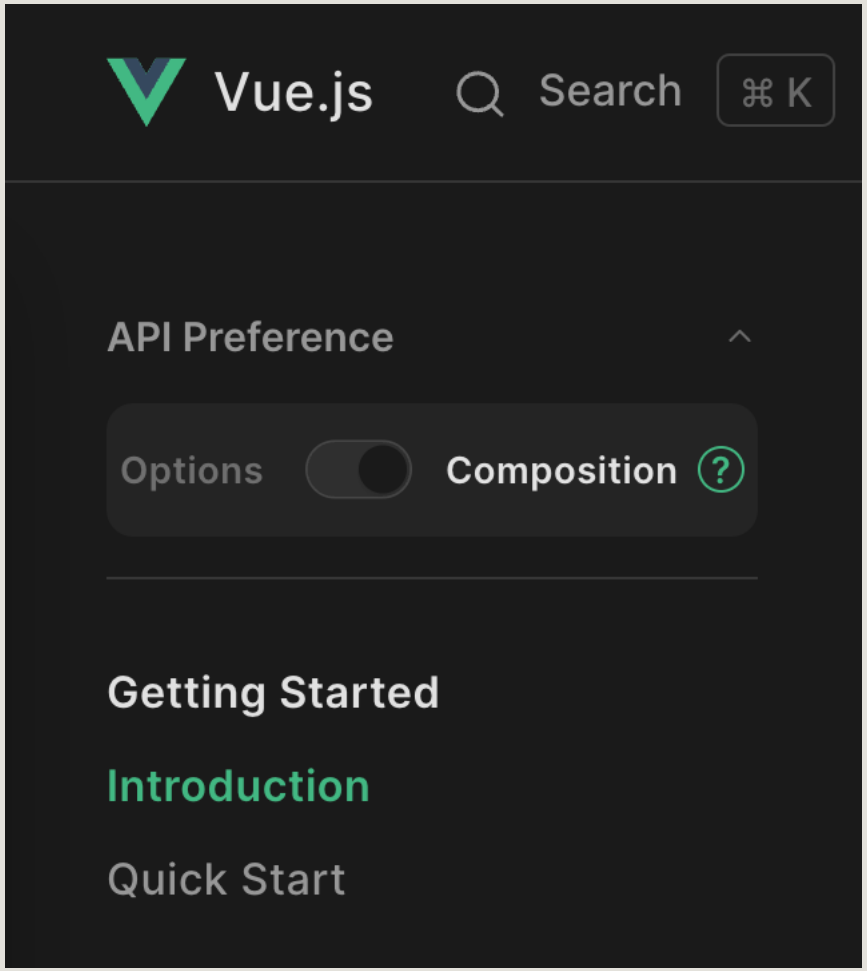
Get started with Vite

Similar to React

Today's focus:

~~Options API~~

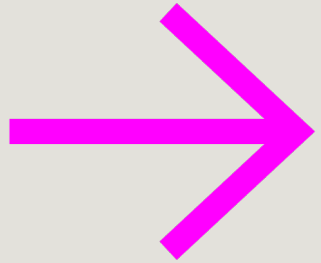
Composition API



knowit

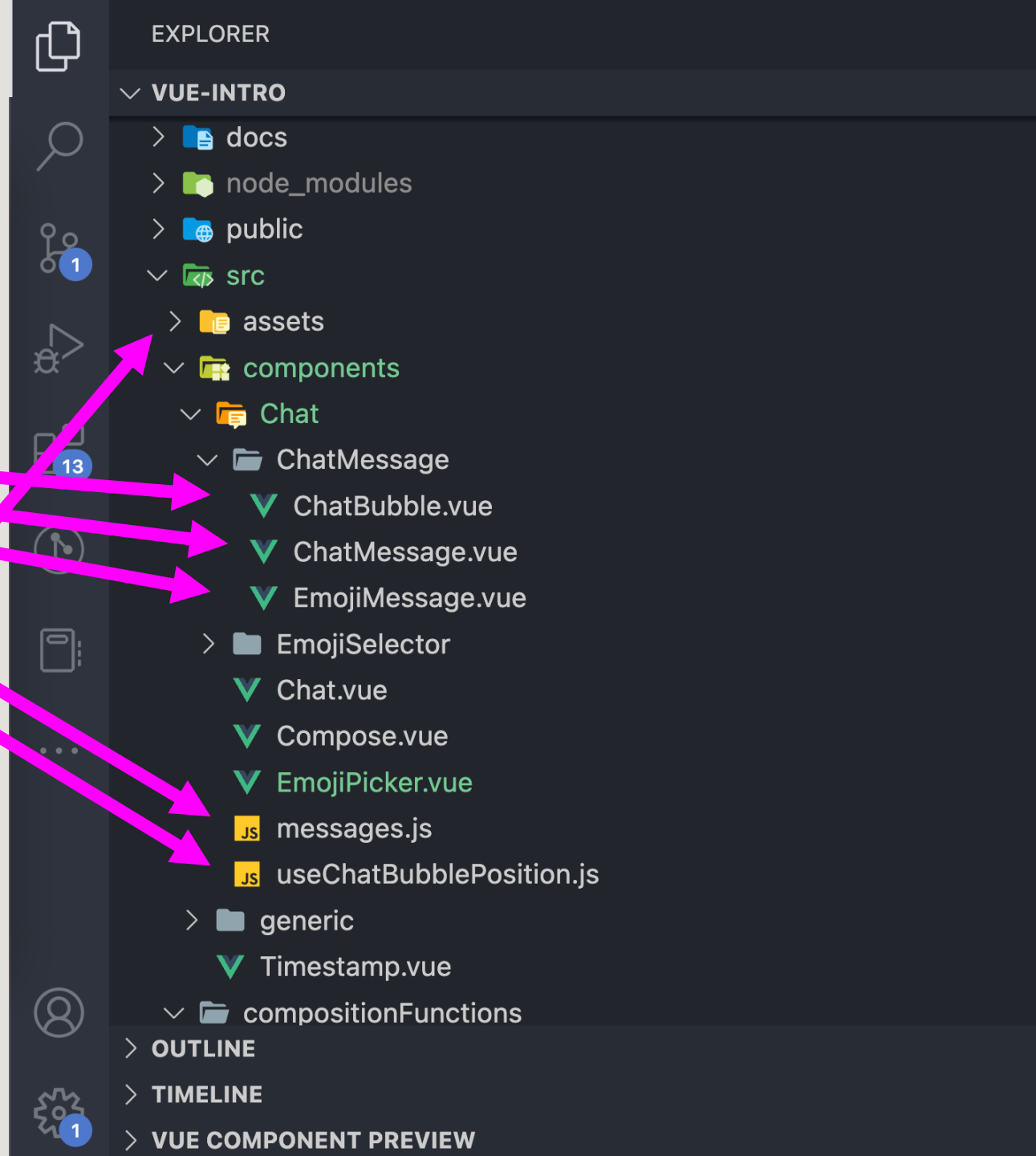
Vue Crash Course

1. Project structure
2. Components
3. Component communication
4. Directives
5. Reactivity tools



1. Vue Project Structure

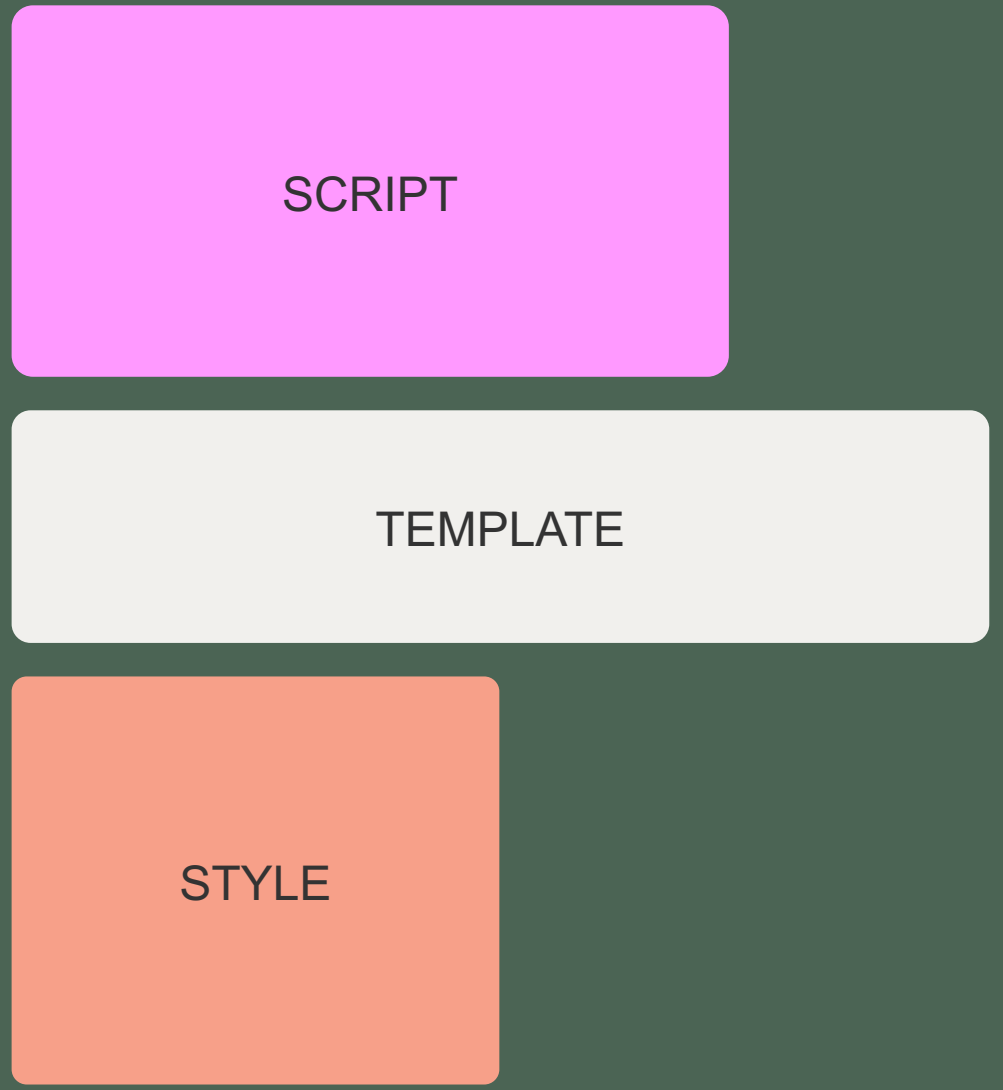
- Hierarchy of **.vue** files
  - Single-file components
- Functions (helpers, utilities, etc.)
- Assets, project configuration files, etc.



2. Components

- A single **.vue** file
- Contains:
  - Script (JavaScript)
  - Template (HTML)
  - Style (CSS)

knowit



2. Components

- A single **.vue** file
- Contains:
  - Script (JavaScript)
  - Template (HTML)
  - Style (CSS)

Vue.js

```
You, 11 months ago | 1 author (You)
<script setup>
// A chat bubble container
import useChatBubblePosition from "../useChatBubblePosition";

const props = defineProps({
  direction: {
    type: String,
    default: "right",
  },
});

// Import CSS variables from a 'hook'
const { cssVars } = useChatBubblePosition(props.direction);
</script>

<template>
  <div class="chat-bubble shadow-1" :style="cssVars">
    0 references
    <slot />
  </div>
</template>

<style scoped lang="scss">
1 reference
.chat-bubble {
  background-color: #fbfaf8;
  color: #281822;
  border-radius: 10px;
  padding: 0.5rem 1rem;
  display: inline-block;
  position: relative;
  align-self: var(--align);
  margin: var(--margin);
```

2. Components

Vue.js

```
// ChatMessage.vue

<template>
  <ChatBubble :direction="direction">
    {{ message.content }}
  </ChatBubble>
</template>
```

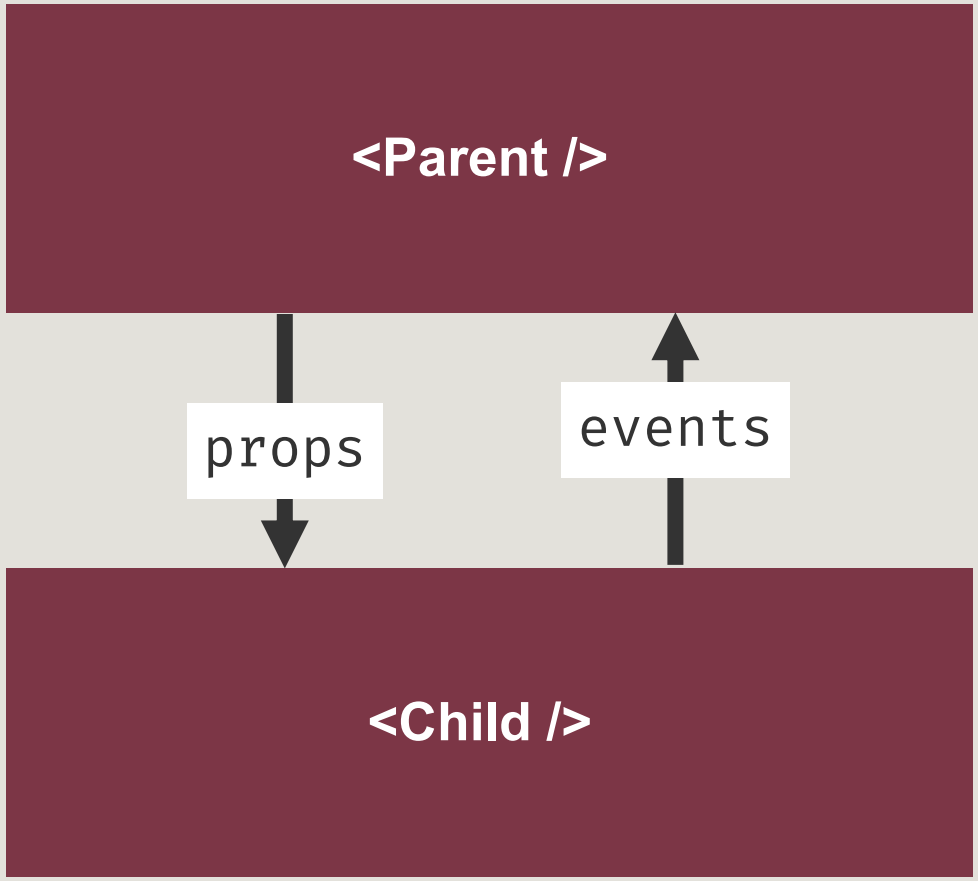
ChatMessage.vue

```
You, 11 months ago | 1 author (You)
<script setup>
// A chat bubble container
import useChatBubblePosition from "../useChatBubblePosition";

const props = defineProps({
  direction: {
    type: String,
    default: "right",
  },
});

// Import CSS variables from a 'hook'
const { cssVars } = useChatBubblePosition(props.direction);
</script>

<template>
  <div class="chat-bubble shadow-1" :style="cssVars">
    0 references
    <slot />
  </div>
</template>

<style scoped lang="scss">
1 reference
.chat-bubble {
  background-color: #fbfaf8;
  color: #281822;
  border-radius: 10px;
  padding: 0.5rem 1rem;
  display: inline-block;
  position: relative;
  align-self: var(--align);
  margin: var(--margin);
```

ChatBubble.vue

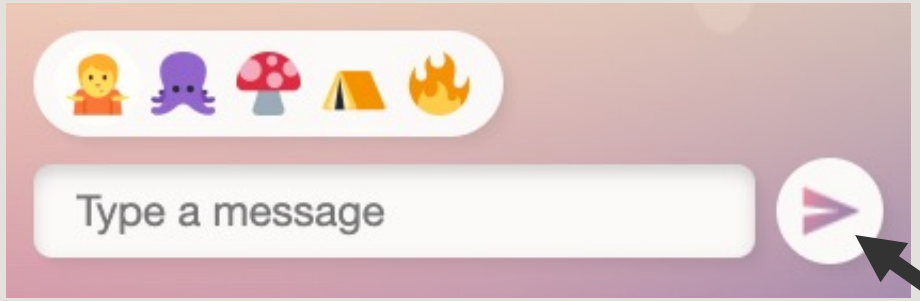
3. Component Communication

- Components pass information to each other with **props** and **events**.



3. Component Communication

- Components pass information to each other with **props** and **events**.



Compose.vue

```
<Button icon="send" @click="send" />
```

props

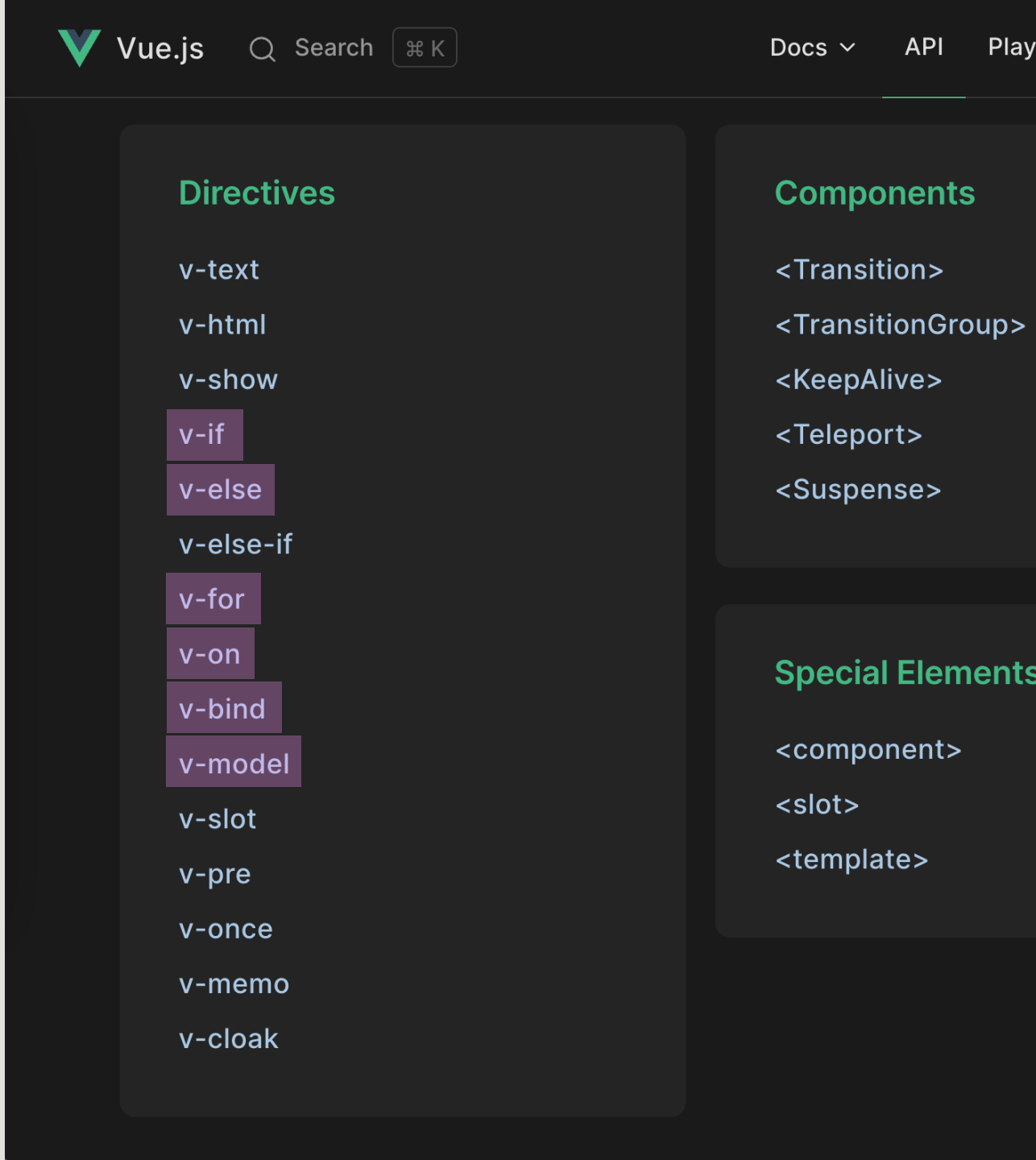
events

Button.vue

```
const props = defineProps({
  icon: {
    type: String,
    default: null,
  },
});

const emit = defineEmits(["click"]);
emit("click", someParameter);
```

4. Directives

- Built-in reusable code or logic.
- Allow developers to manipulate the DOM in many ways.
- E.g. conditional rendering and looping.
- See the docs!



v-if

- Conditionally render an element.
- See also: *v-else* and *v-else-if*.

```
<marquee v-if="year < 2010">
  Hello from Myspace
</marquee>
```

v-for

- Loop through a list.
- Render each element.

```
<Game
  v-for="game in games"
  :title="game.name"
  :description="game.description"
/>
```

v-bind

- Binds a variable to a prop or DOM attribute.
- The ”value” will be the variable’s value, not the literal text entered.
- `v-bind:title` can be shortened to `:title`.

```
<Timestamp v-bind:date="props.message.timestamp" />

<Timestamp :date="props.message.timestamp" />
```

The time is 14:22

~~The time is props.message.timestamp~~

v-model

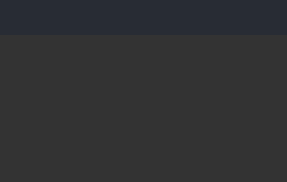
- Binds a variable to a prop or DOM attribute.
- Also adds a **handler** to change that value.
- Works automatically on most elements!

```
<Input v-model="value" />

// The above is short for
<Input
  :value="value"
  @input="
    [/*function that updates the value*/]
  "
/>
```

v-on

- Event handler.
- *“When an event with this name is emitted, do this.”*
- `v-on:click`
  can be shortened to
  `@click`

```
<Button v-on:click="send" />

<Button @click="send" />
```



```
function send() { … }
```

5. Reactivity Tools

- Make your app reactive.
- See the docs!

```
ref()

computed()

watch(), watchEffect()

provide(), inject()
```

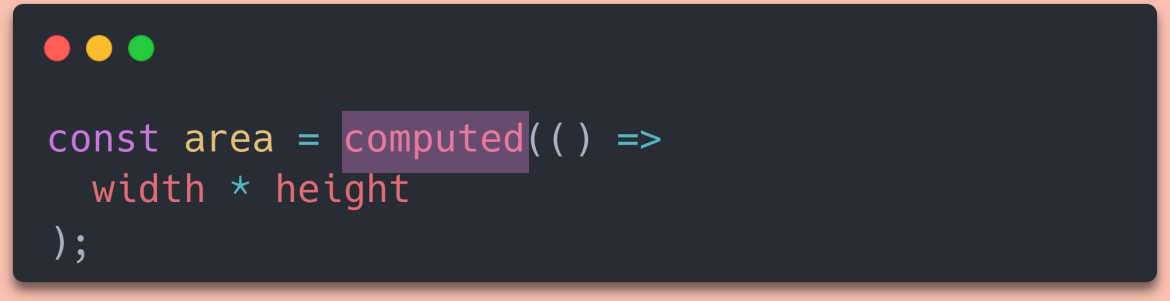
ref()

- Declare a reactive variable.
- Use `reactive()` for a reactive object.

```
// script
const age = ref(0);

function growOlder() {
  age.value = age.value + 1;
}

// template
<p>
  I am {{ age.value }} years old.
</p>
```

computed()

- Store a reactive computed value.



```
const area = computed(() =>
  width * height
);
```

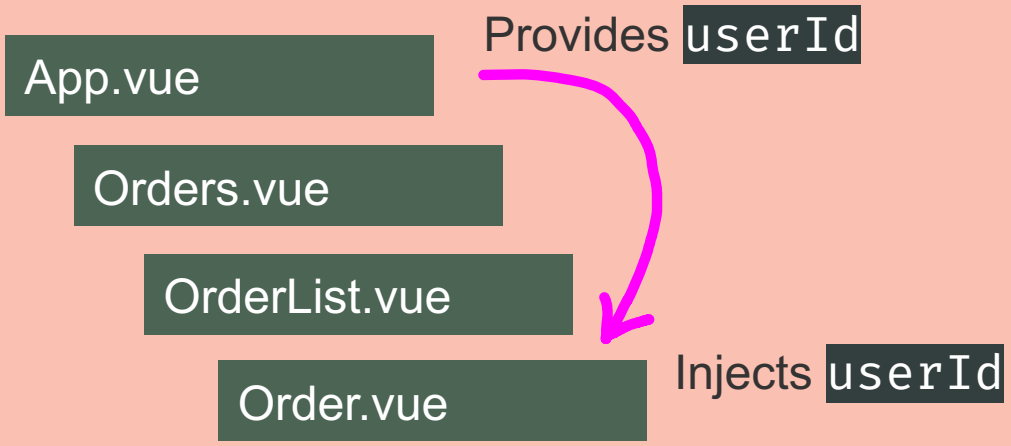
watch(), watchEffect()

- Watch one or more variables.
- Do something when their values change.


```
// Watch the query variable
watch(query, () => {
  fetch(`/items?search=${query}`)
});

// Automatically watch all reactive
// variables
watchEffect(() => {
  fetch(`/items?search=${query}`)
});
```

`provide()`, `inject()`

- Provide data to all components in the component tree.
- Inject some provided data into a component.
- No need to drill the data as props through multiple components.



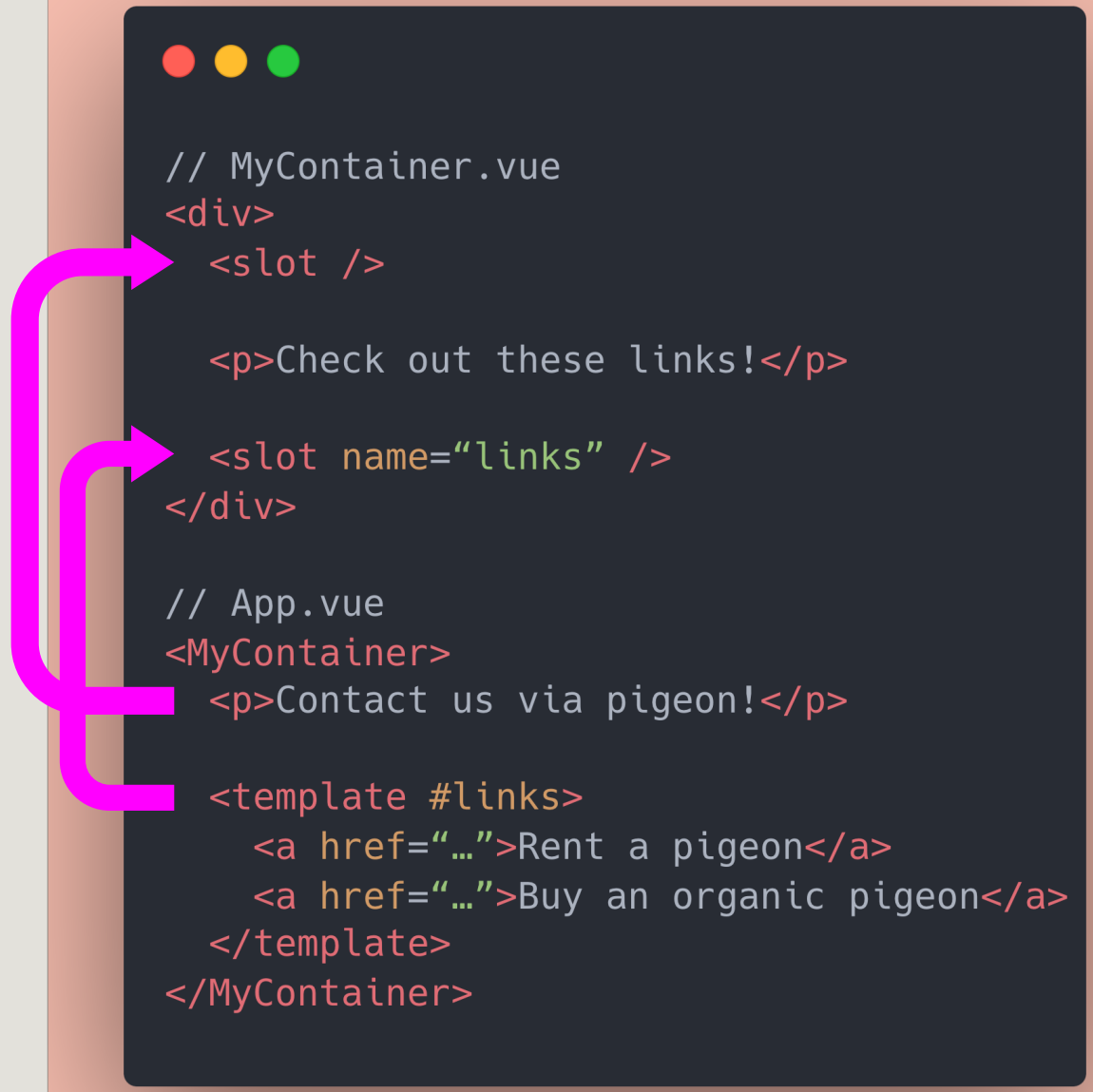
provide(), inject()

- Provide data to all components in the component tree.
- Inject some provided data into a component.
- No need to drill the data as props through multiple components.

```
// Somewhere high up in the
// component tree
const allTheData = { … };
provide('data', allTheData);

// Some small component way down
// in the component tree,
// in a totally different file
const allTheData = inject('data');
```

Slots

- Display content inside a component.
- Slots can be named.



Recap

1. Project structure
2. Components
3. Component communication
4. Directives
5. Reactivity tools


```
// Directives
v-if
v-for
v-bind
v-model
v-on

// Reactivity tools
ref()
computed()
watch(), watchEffect()
provide(), inject()

// See the Vue docs!
```

knowit

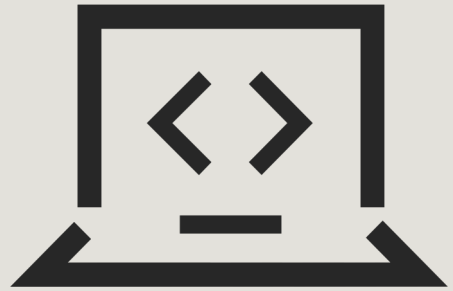
Questions?

Up next: Workshop

knowit

Introduction to Declarative DOM Manipulation

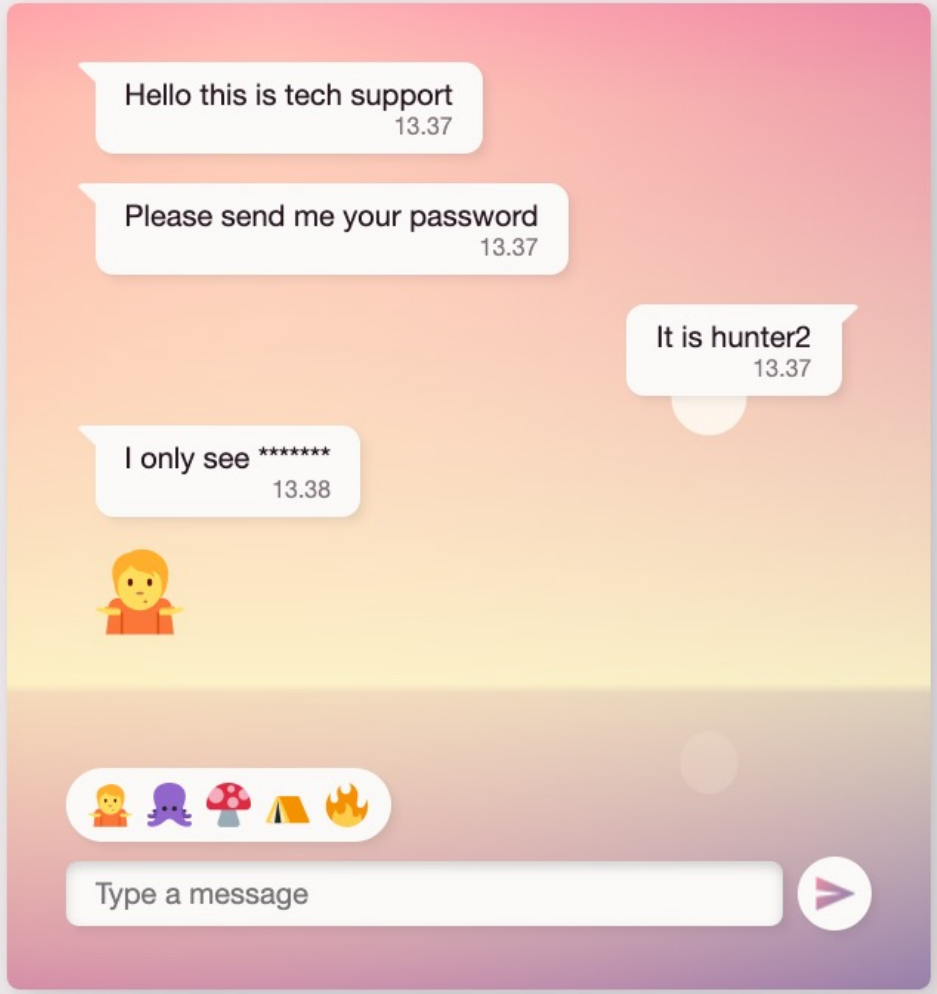
Workshop



Workshop

knowit

Finish a chat app



1. Get the code from GitHub:
   - https://shorturl.at/hoyBM
2. Read the README for instructions.
3. Get coding!
4. Help a friend and ask questions.

Workshop

knowit

github.com/knowit-finland-javascript-guild/vue-for-react-developers

Workshop

knowit

Thanks