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.
- Laser cutting, cycling and board games.
- Web developer for 7+ 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. Backend Integration
4. Workshop

knowit

Introduction to Declarative DOM Manipulation

Defining Declarative

knowit

↓

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."*

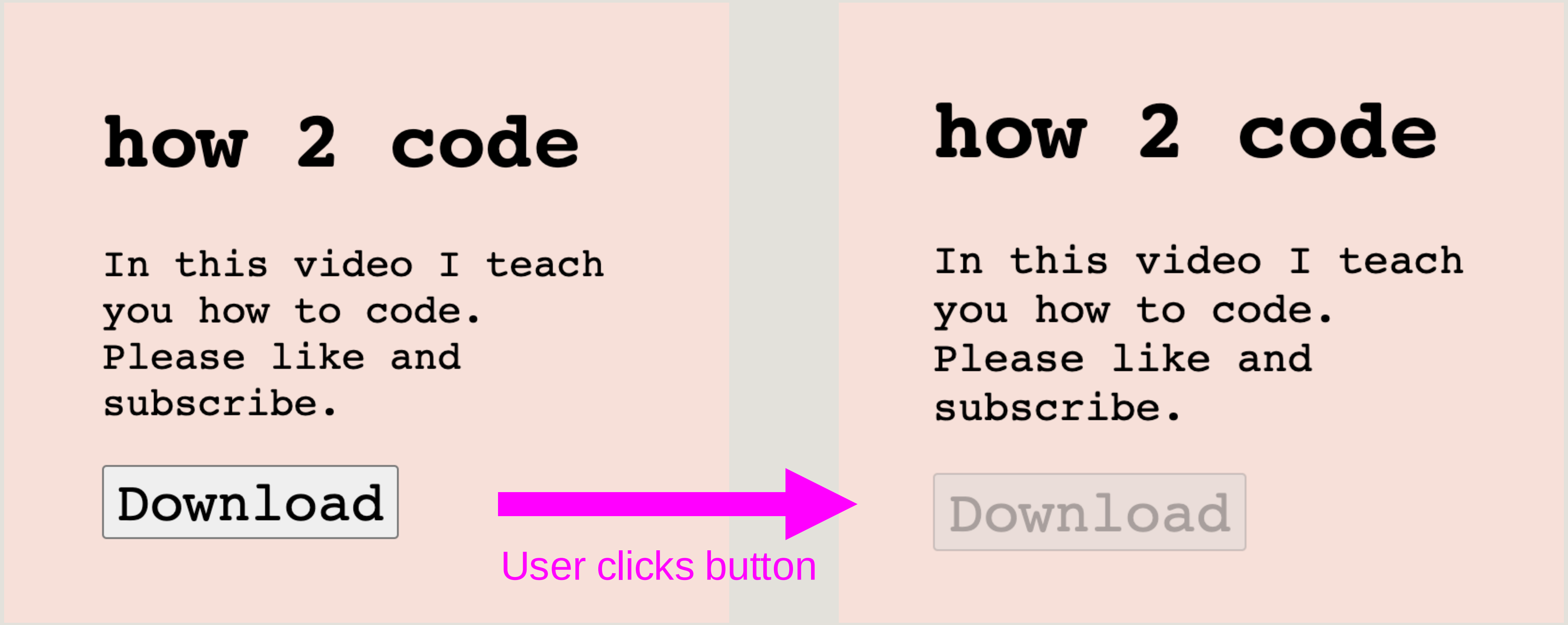
- **Imperative** code describes how to do things.
- *"Blend frozen strawberries and milk.
  Add ice cream and blend.
  Pour into a glass and serve."*

↓

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

knowit

Example


how 2 code

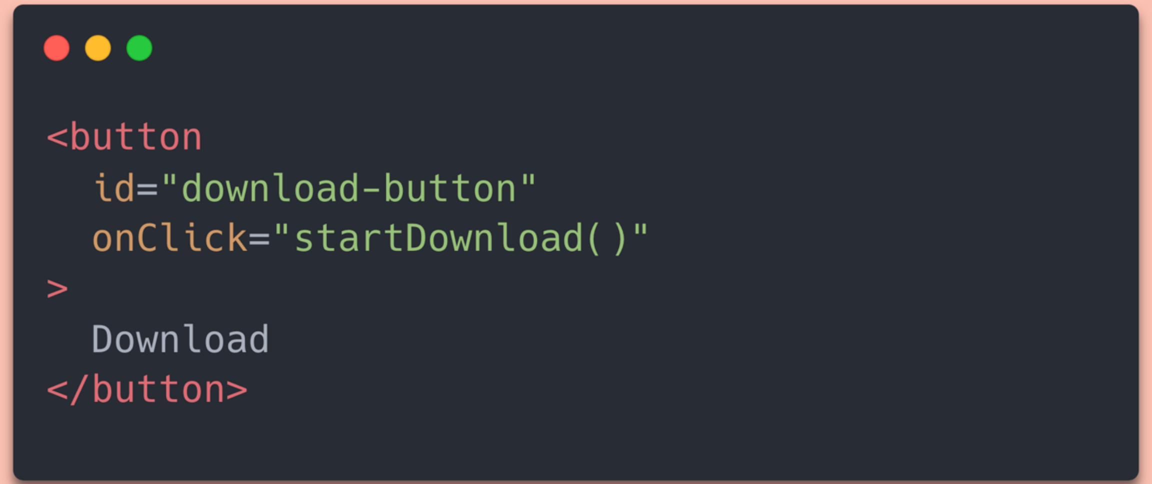
In this video I teach you how to code. Please like and subscribe.

how 2 code

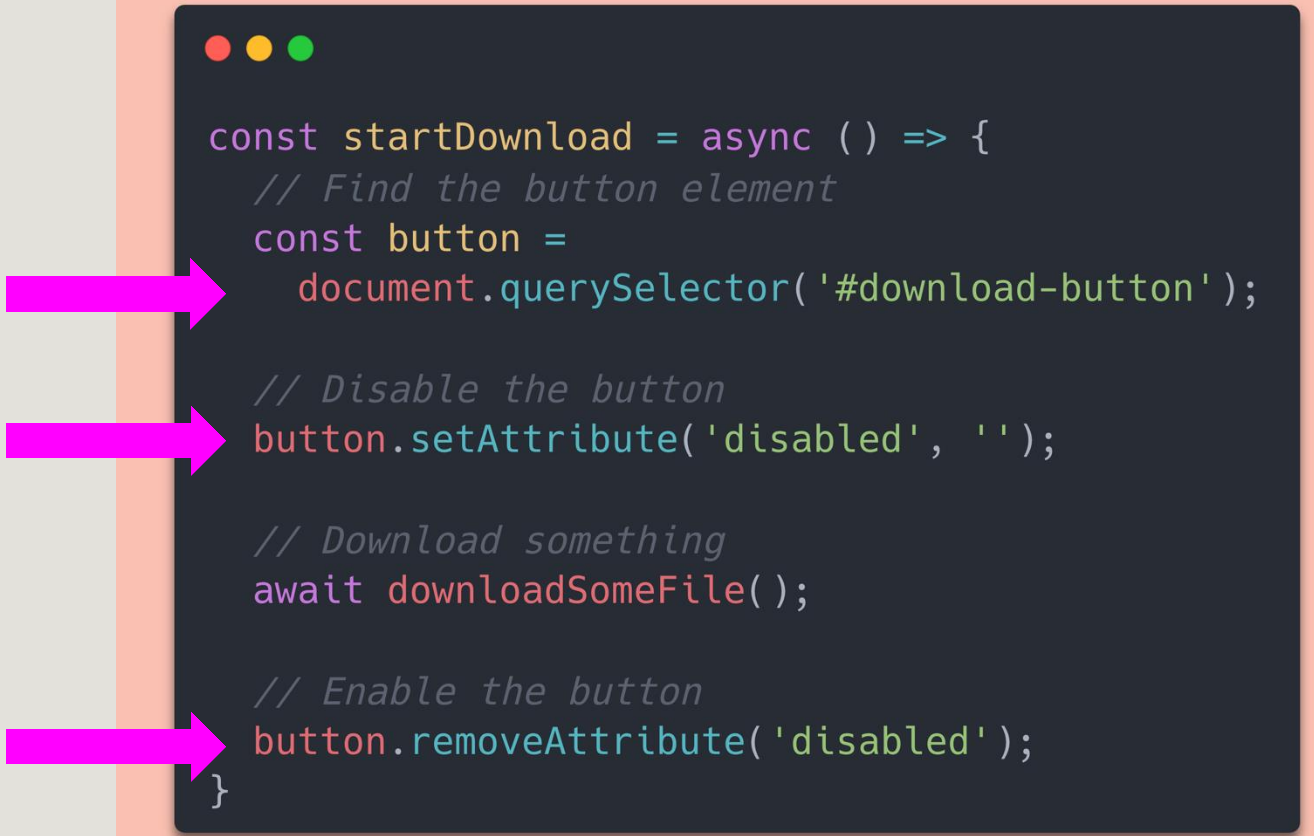
In this video I teach you how to code. Please like and subscribe.

Imperative

- Manually describe the actions to perform.



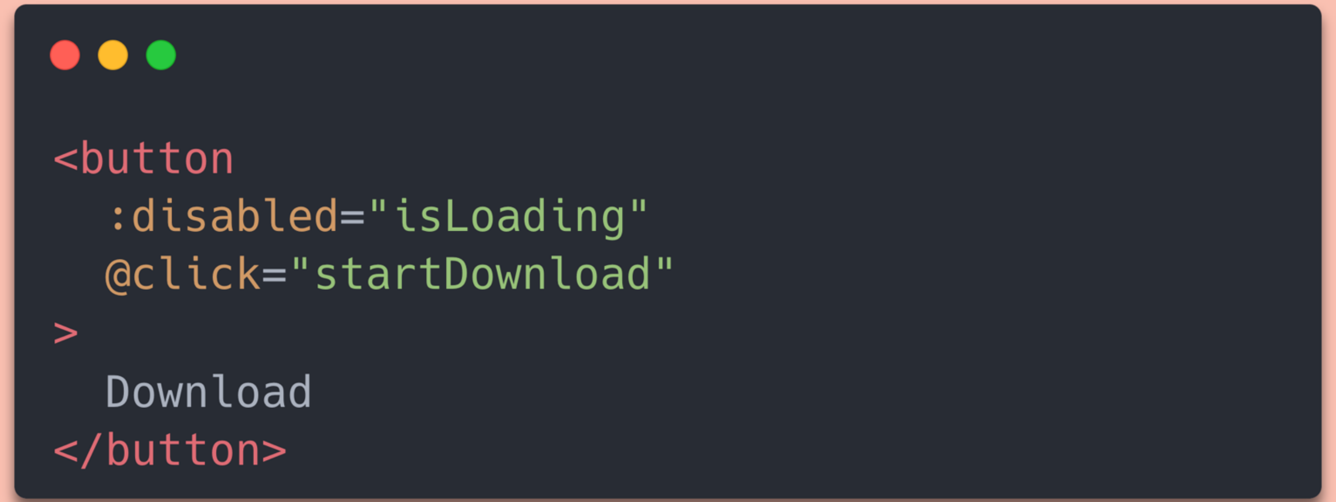
```html
<button
  id="download-button"
  onClick="startDownload()"
>
  Download
</button>
```



```javascript
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');
}
```

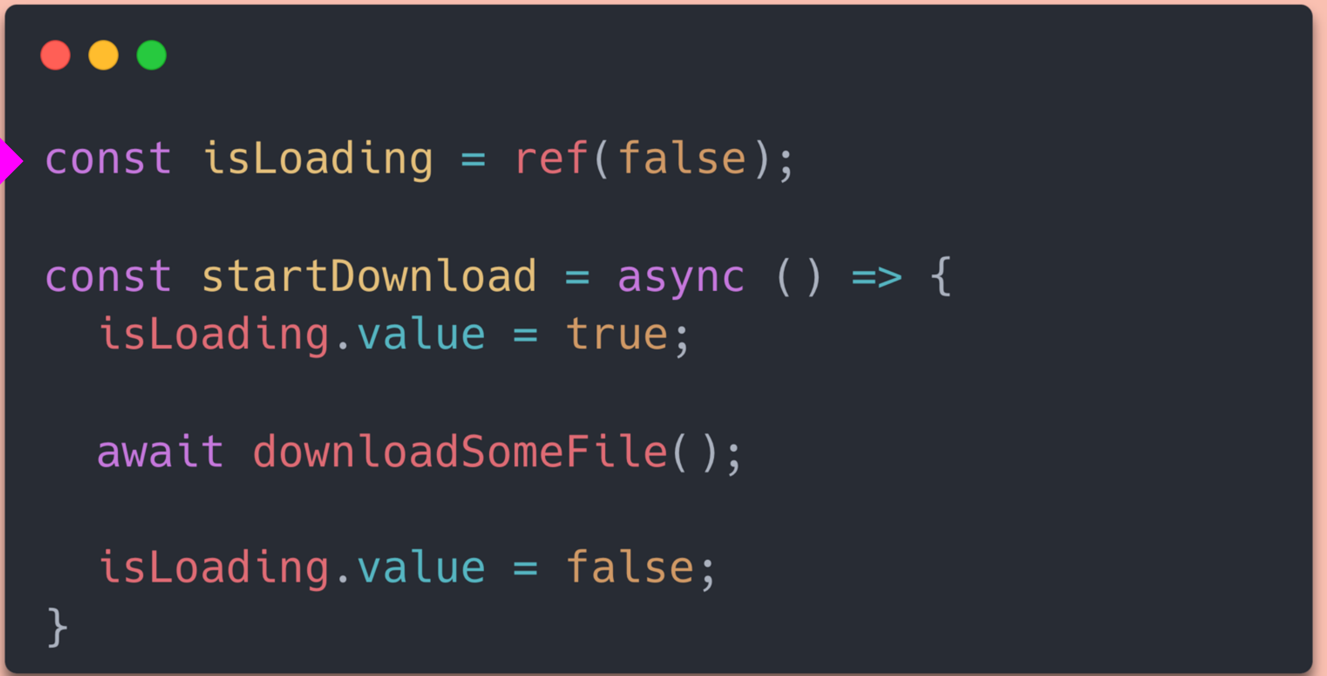
Declarative (Vue.js)

- 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

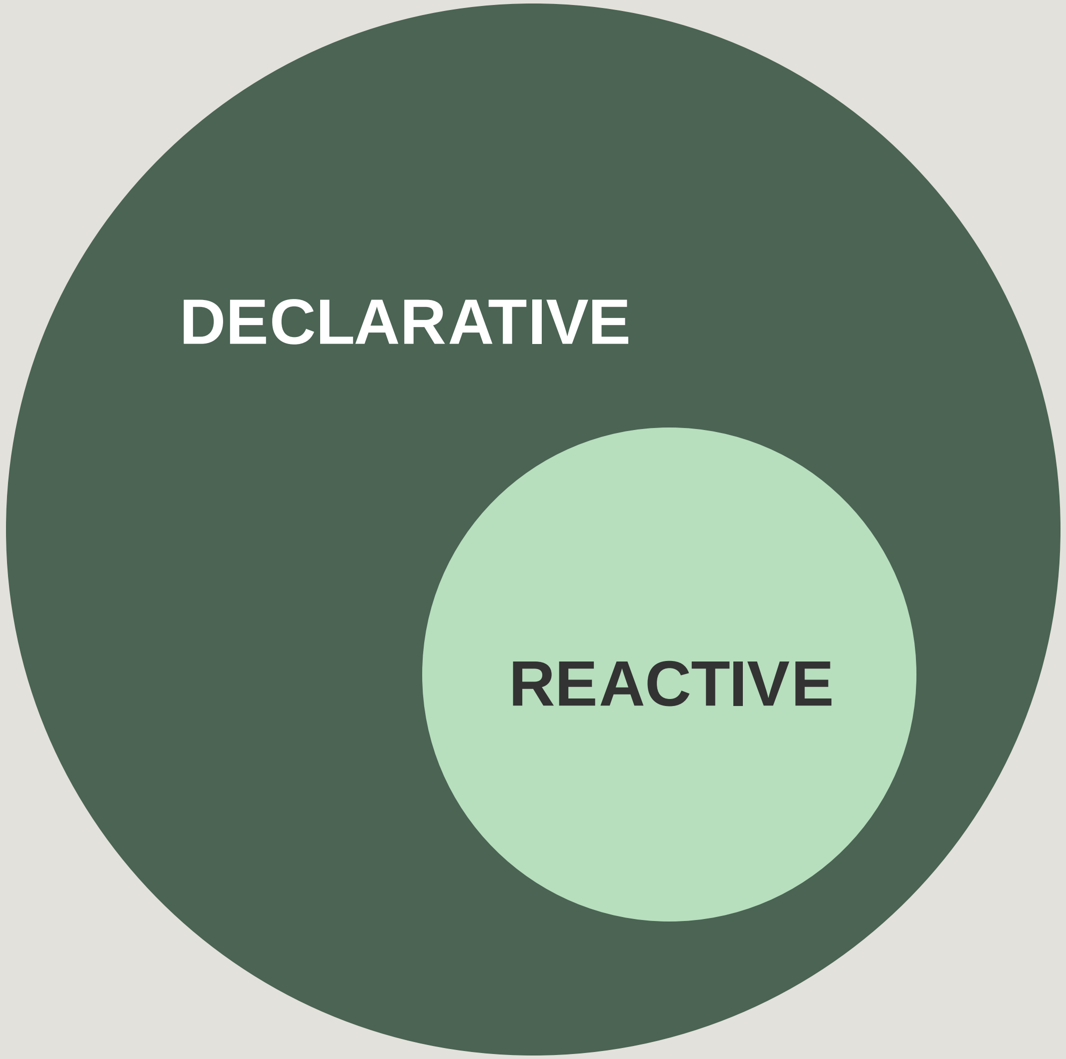
The declarative expression of the relationship between values that change over time.

$$a = b + c$$

knowit

Reactive Programming

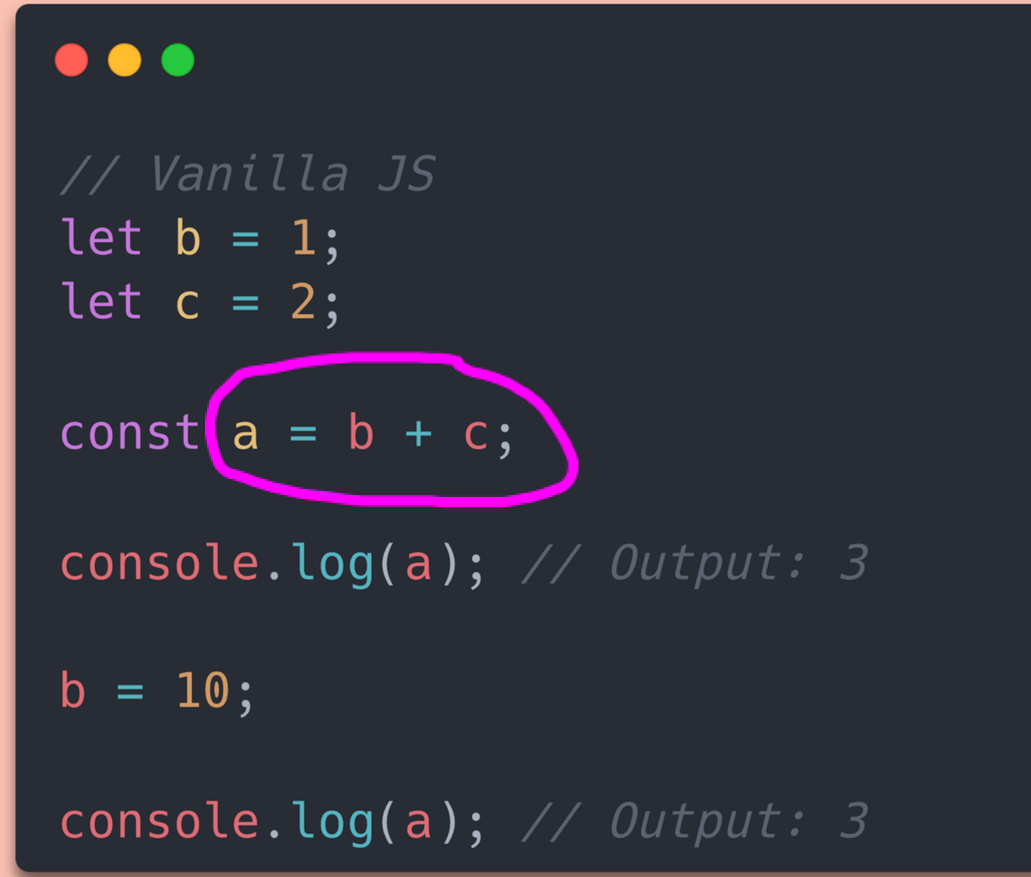
The declarative expression
of the relationship between values that change over time.



knowit

Reactive Programming

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

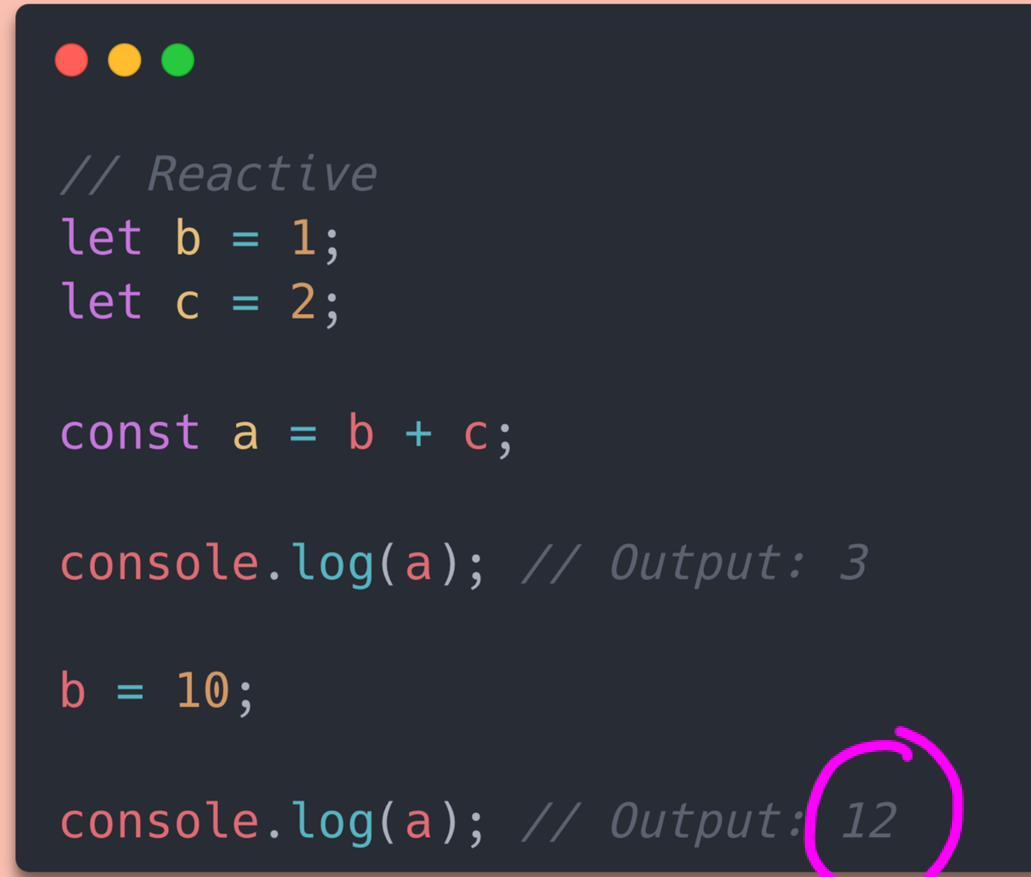


```
// Vanilla JS
let b = 1;
let c = 2;

const a = b + c;

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

b = 10;

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

knowit

Reactive Programming

The declarative expression
of the relationship between values
<u>that change over time</u>.



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

const a = b + c;

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

b = 10;

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

Reactive

Introduction to ~~Declarative~~ DOM Manipulation

Using Vue.js

knowit

Introduction to Declarative DOM Manipulation

Vue.js

knowit

Vue.js

vuejs.org

- JavaScript web UI framework.
- Build reactive applications.

Get started with Vite

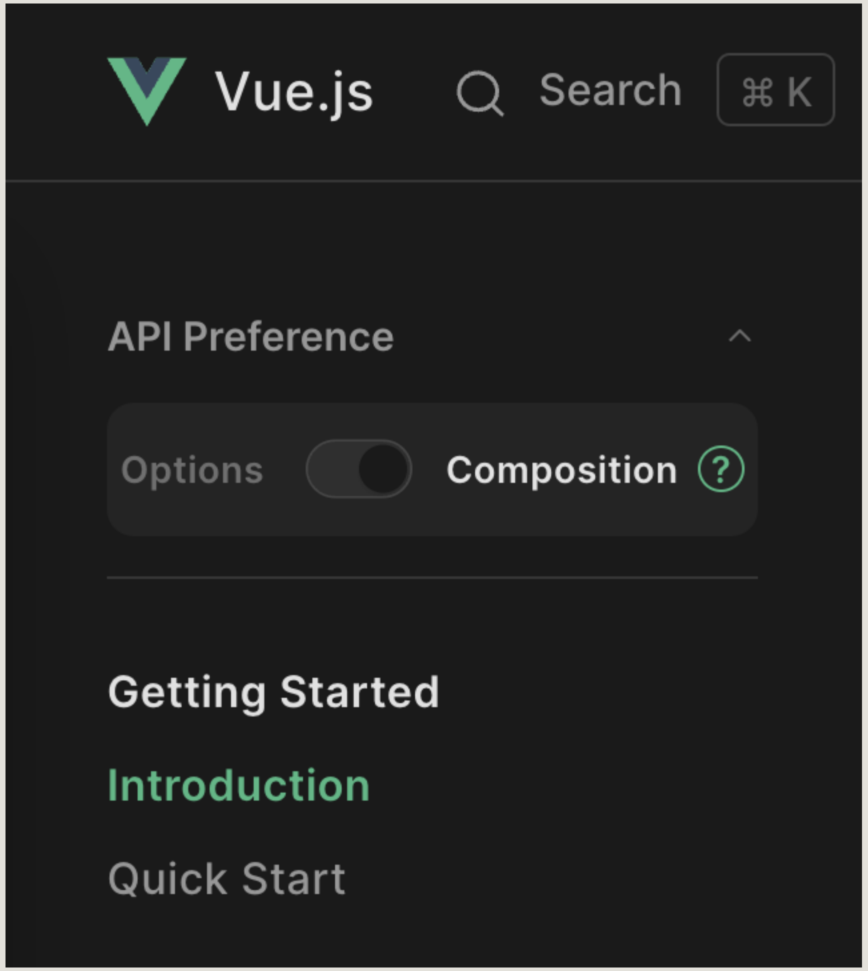
Similar to React

knowit

Today's focus:

~~Options API~~

Composition API



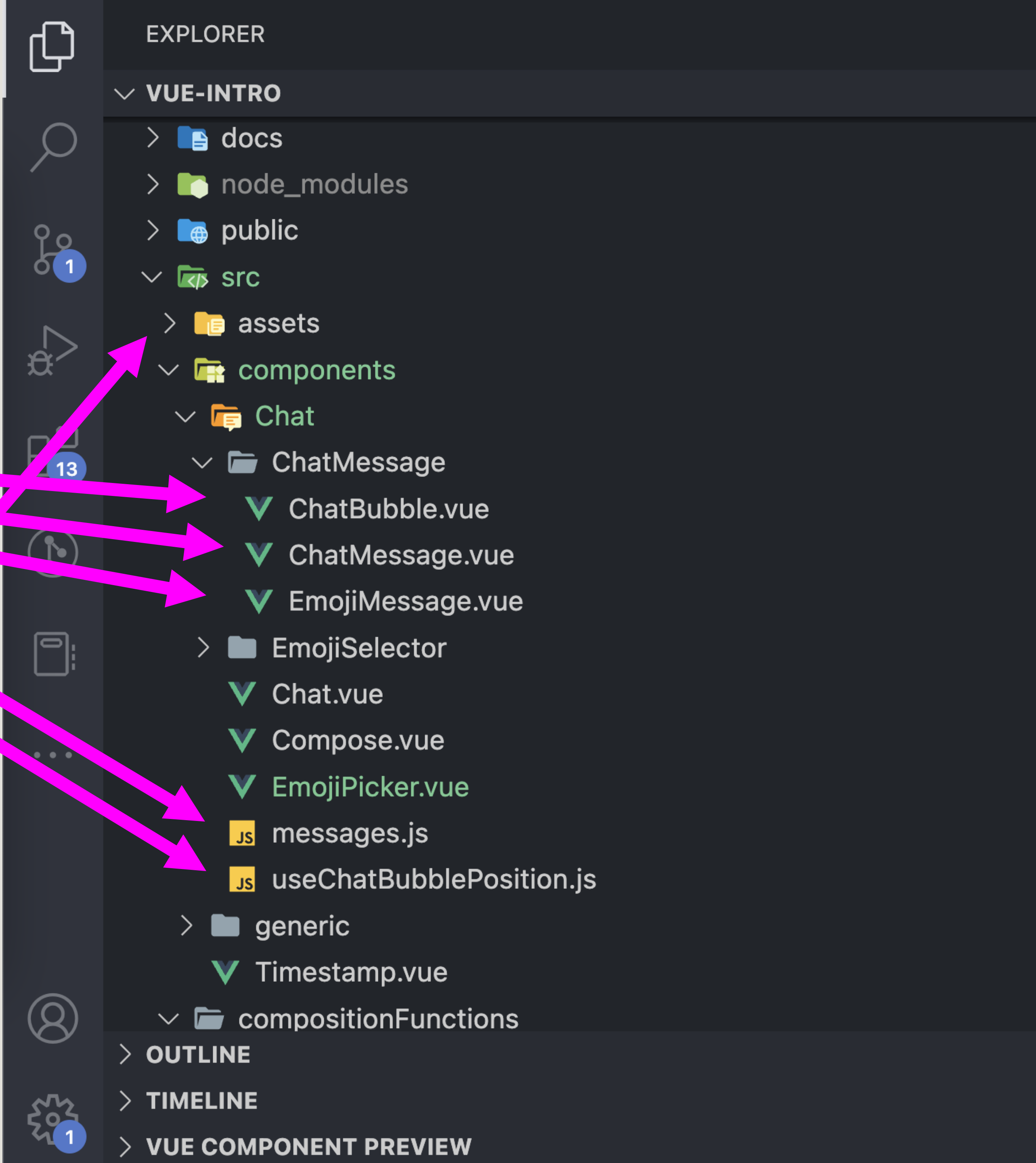
knowit

Vue Crash Course

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

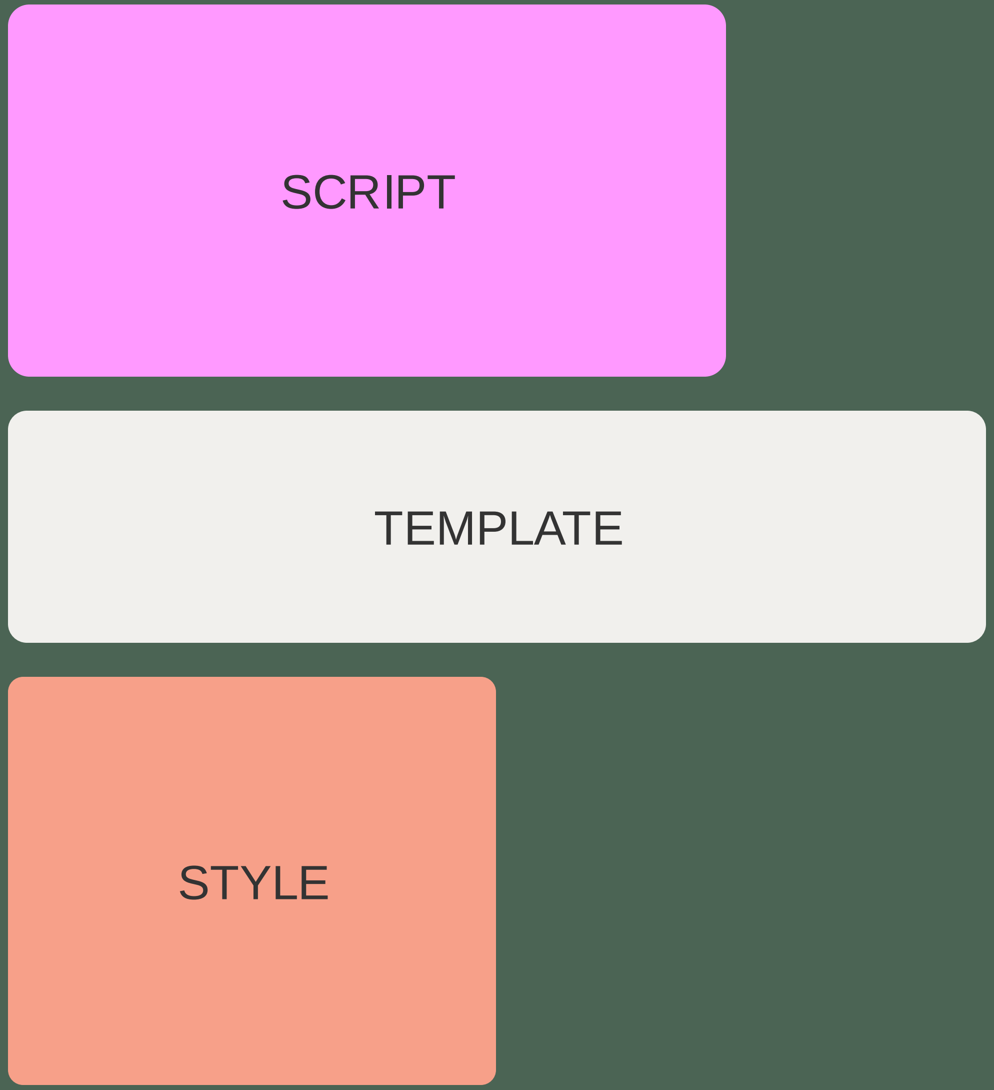
1. Vue Project Structure

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



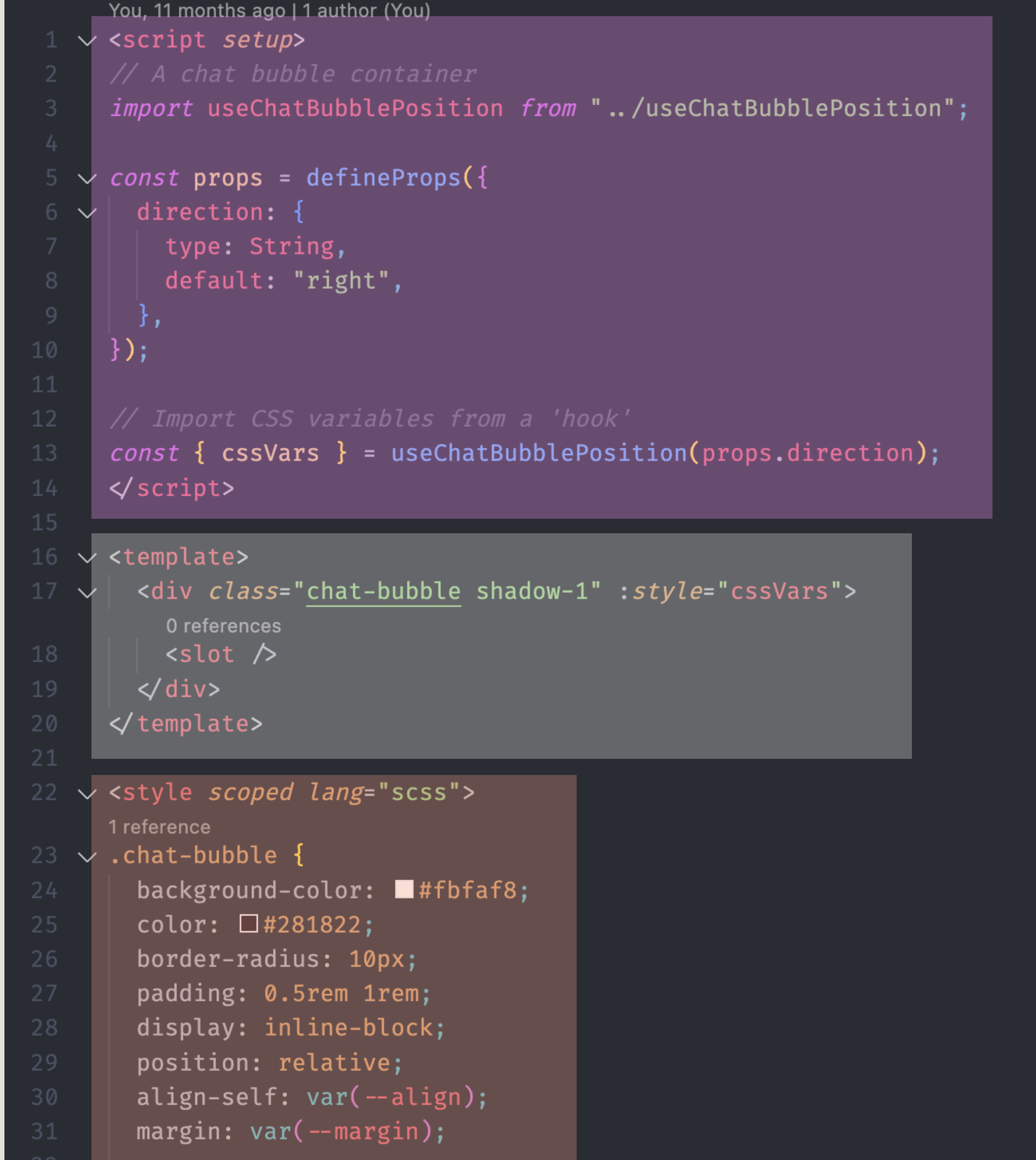
2. Components

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



2. Components

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

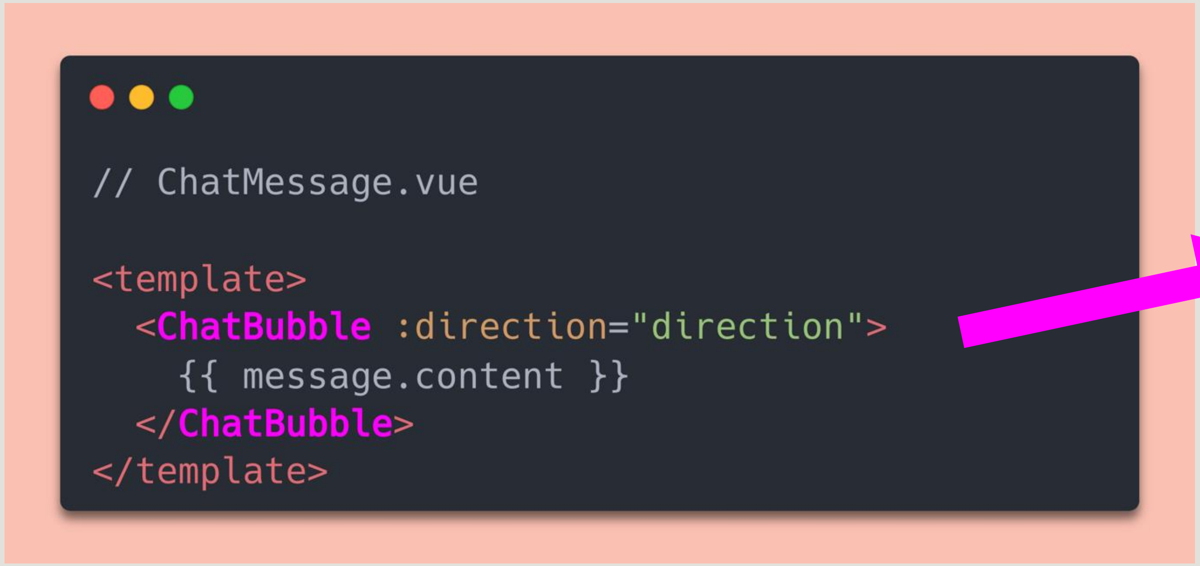


You, 11 months ago | 1 author (You)

```
<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);
```

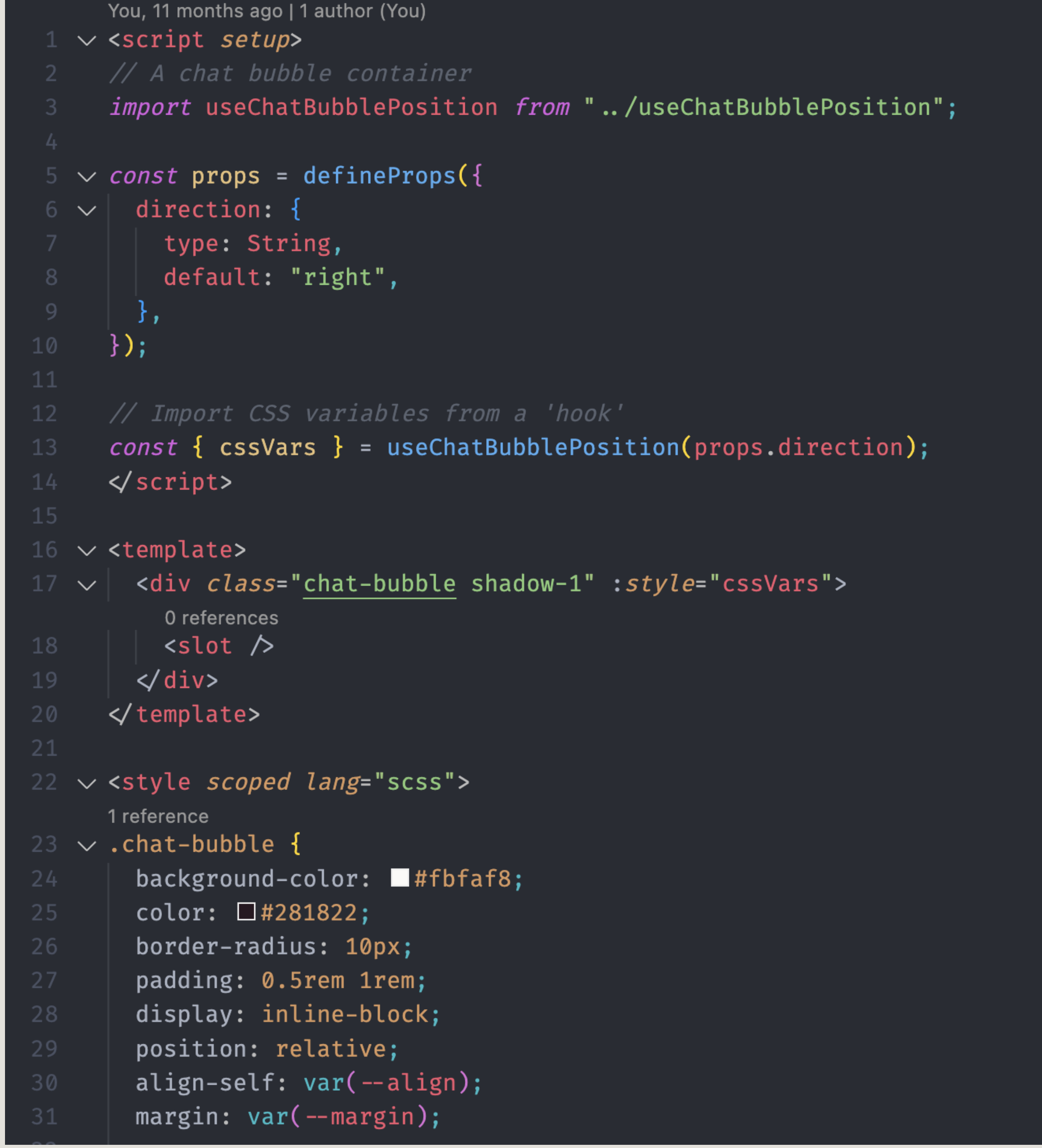
Vue.js

2. Components



```
// ChatMessage.vue

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

ChatMessage.vue



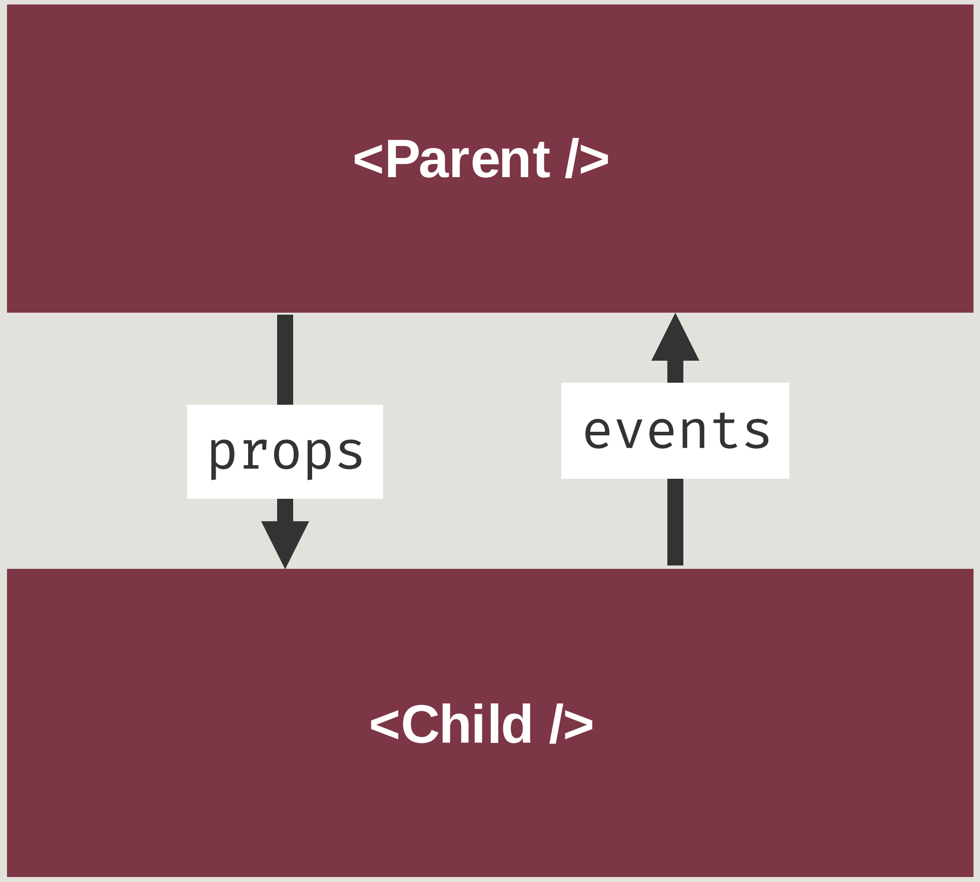
```
You, 11 months ago | 1 author (You)
<script setup>
// A chat bubble container
import useChatBubblePosition from "../useChatBubblePosition";

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

knowit

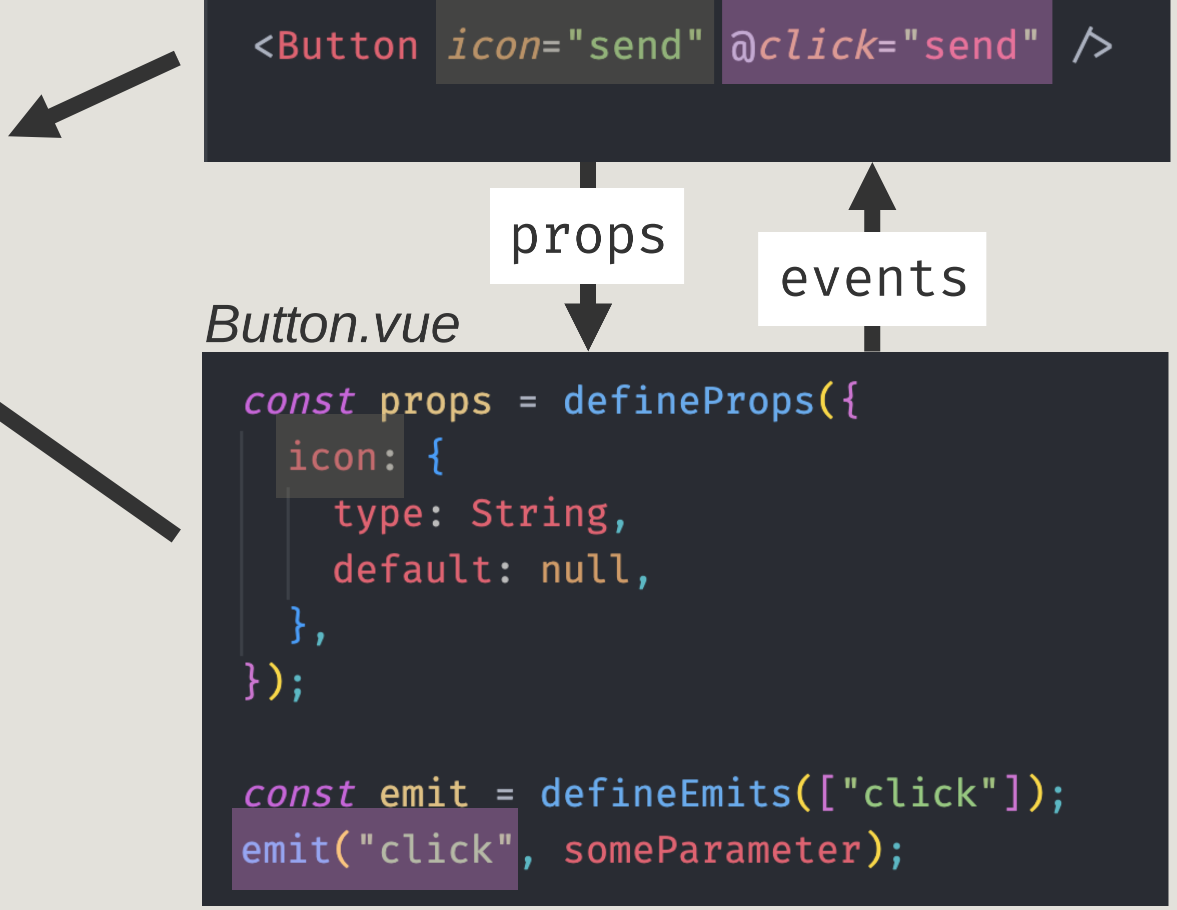
3. Component Communication

- 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

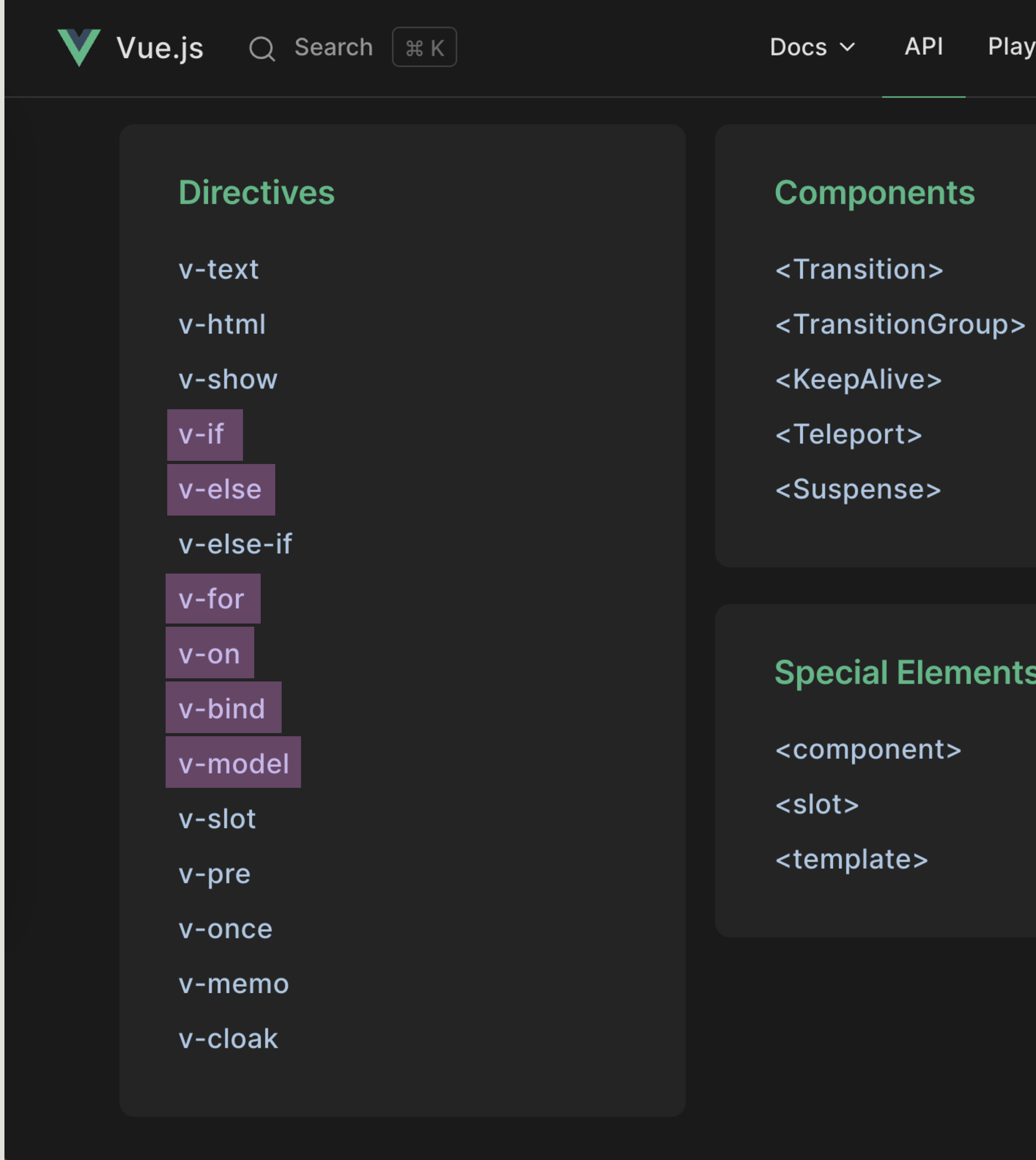
```
<Button icon="send" @click="send" />
```

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!]



Directives

v-text
v-html
v-show
v-if
v-else
v-else-if
v-for
v-on
v-bind
v-model
v-slot
v-pre
v-once
v-memo
v-cloak

Components

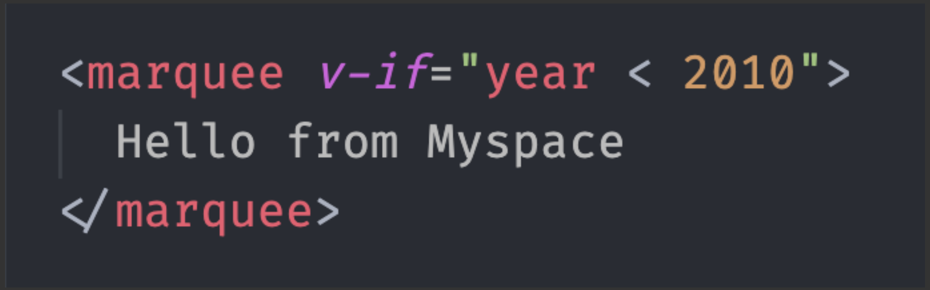
<Transition>
<TransitionGroup>
<KeepAlive>
<Teleport>
<Suspense>

Special Elements

<component>
<slot>
<template>

v-if

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



```html
<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`.

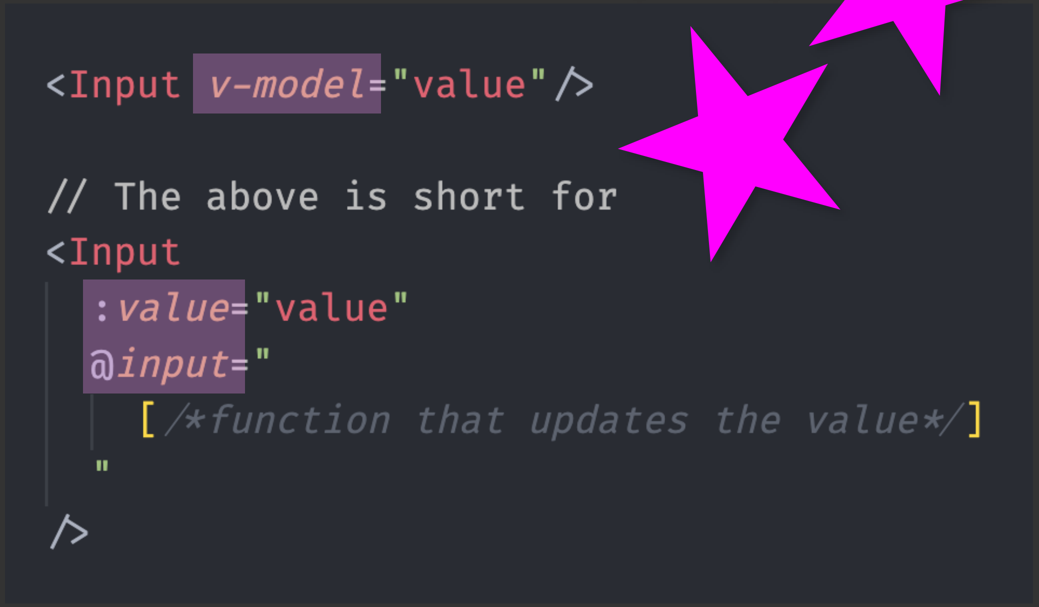
```
<Timestamp v-bind:date="props.message.timestamp" />

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

The time is 14:22

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

v-model

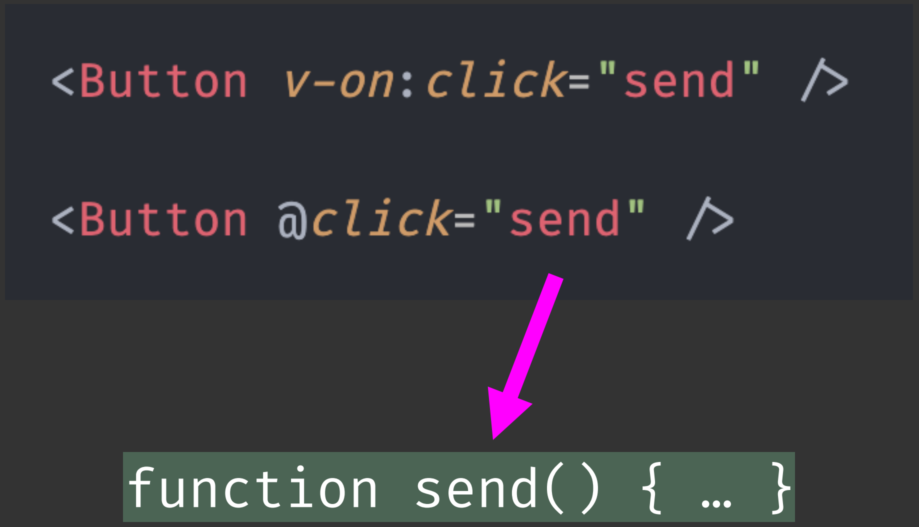
- Binds a variable to a prop or DOM attribute.
- Also adds a **handler** to change that value.
- Works automatically on most elements!



```
<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`

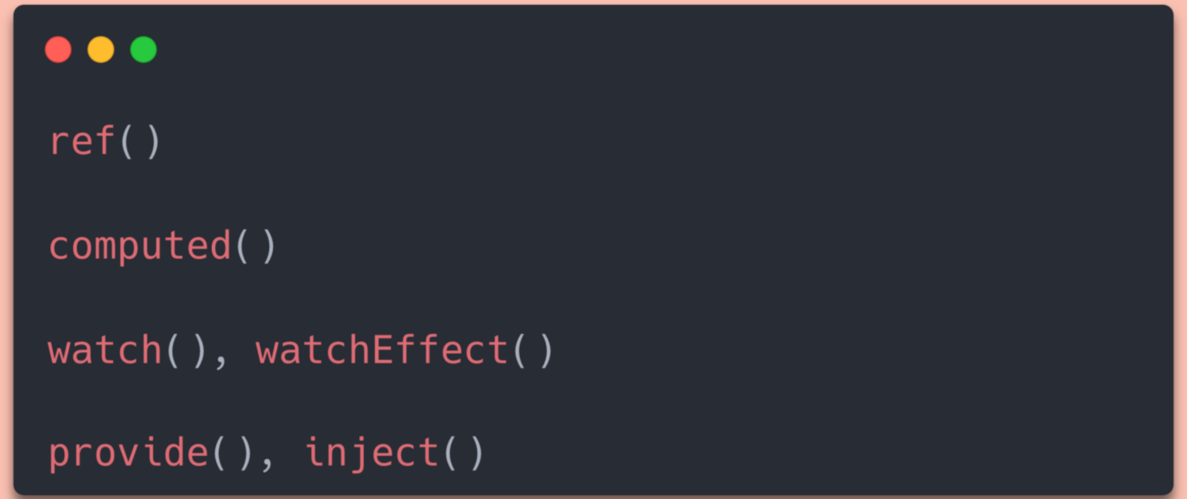


```
<Button v-on:click="send" />

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

```
function send() { … }
```

5. Reactivity Tools

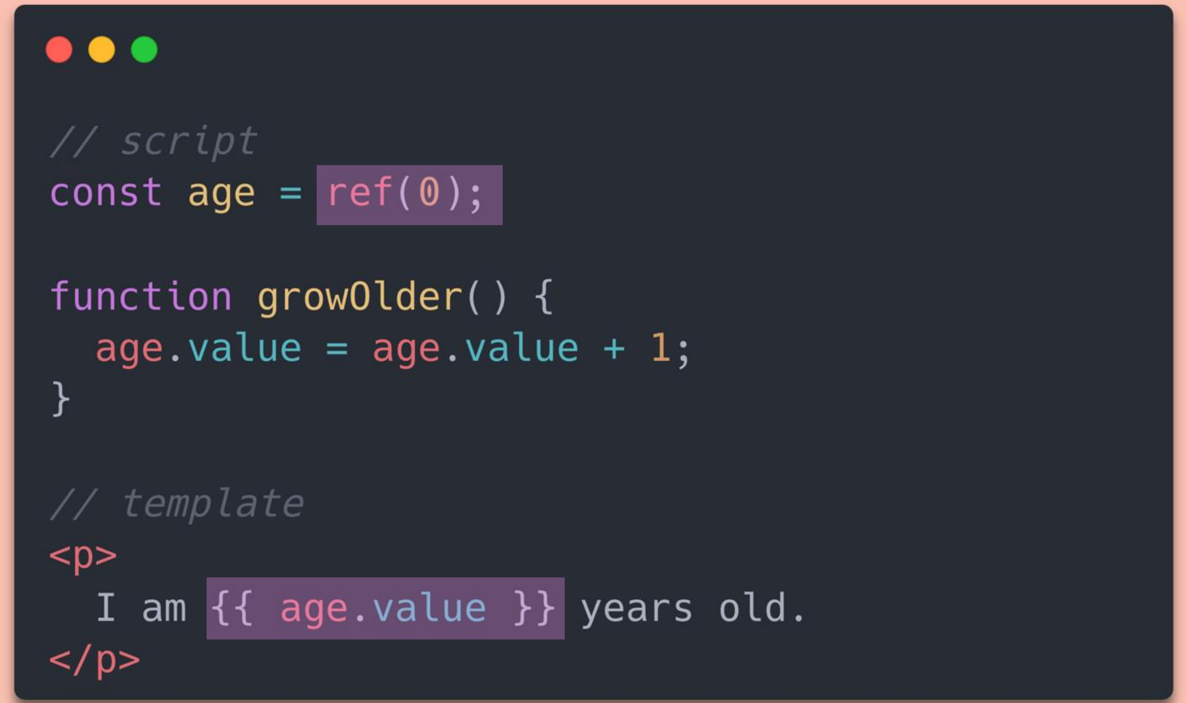
- Make your app reactive.
- See the docs!



```
ref()

computed()

watch(), watchEffect()

provide(), inject()
```

ref()

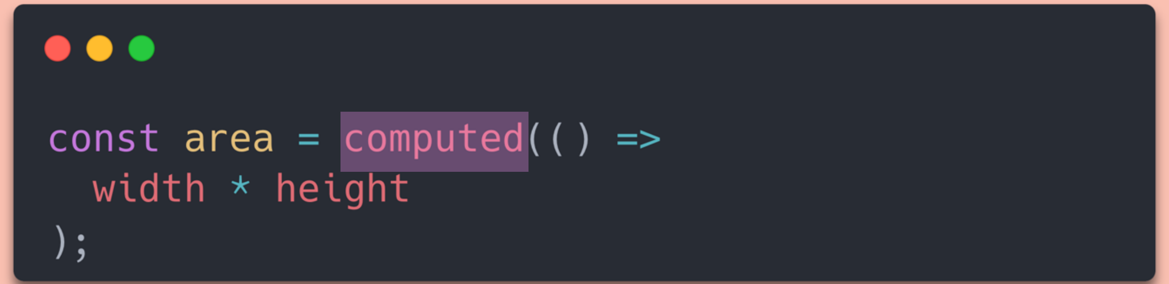
- Declare a reactive variable.
- Use `reactive()` for a reactive object.



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

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

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

computed()

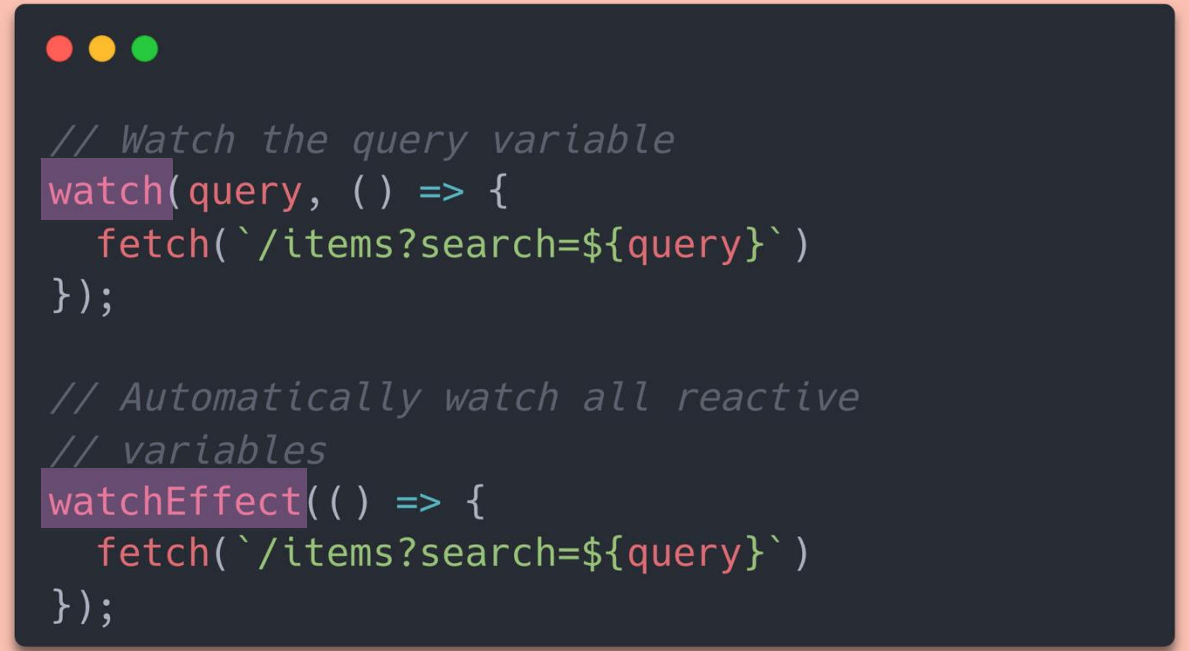
- Store a reactive computed value.



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

watch(), watchEffect()

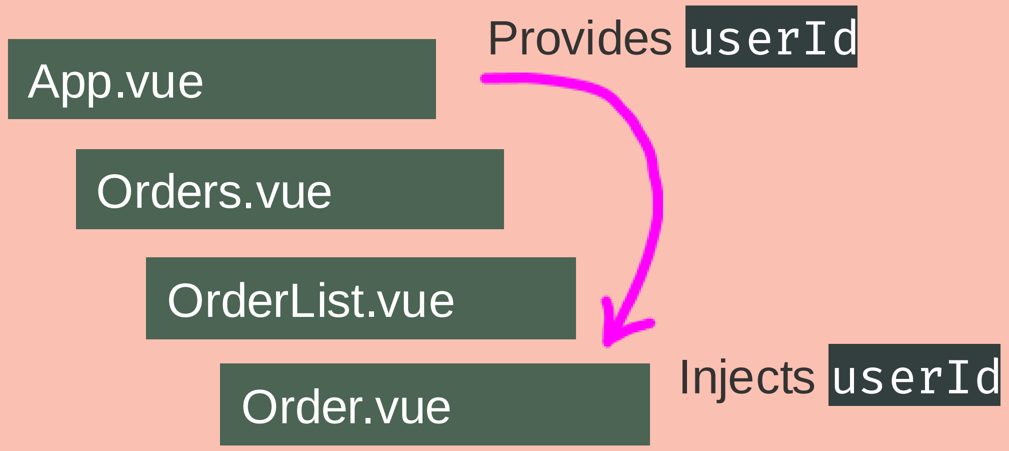
- Watch one or more variables.
- Do something when their values change.



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

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

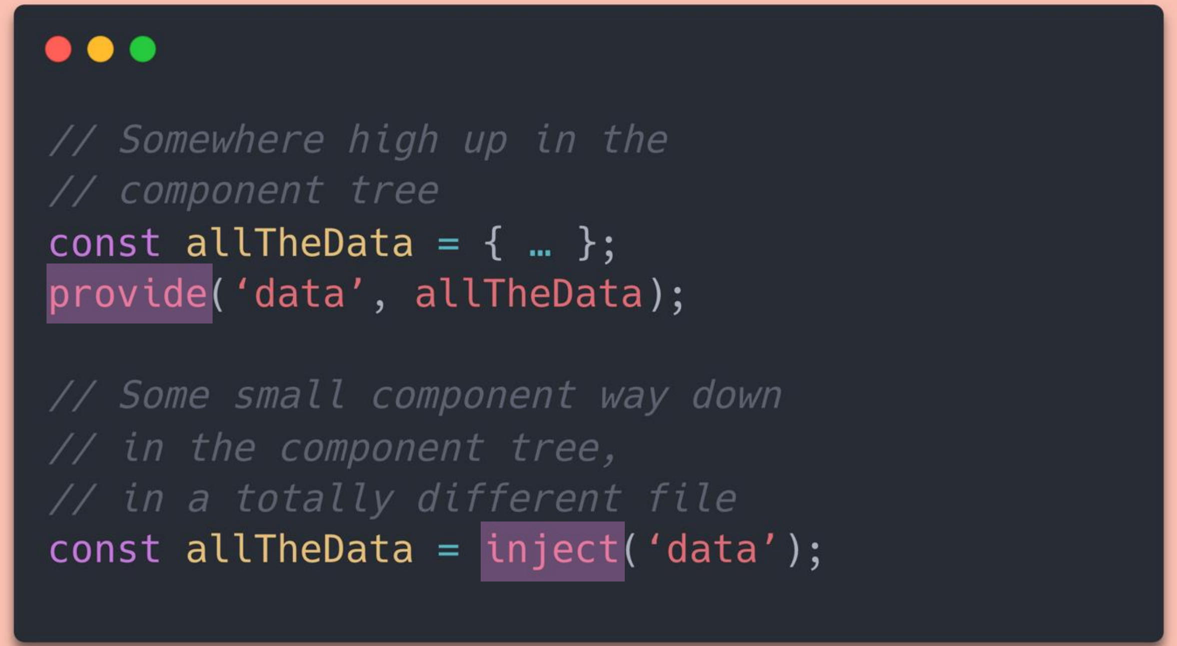
`provide()`, `inject()`

- 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()

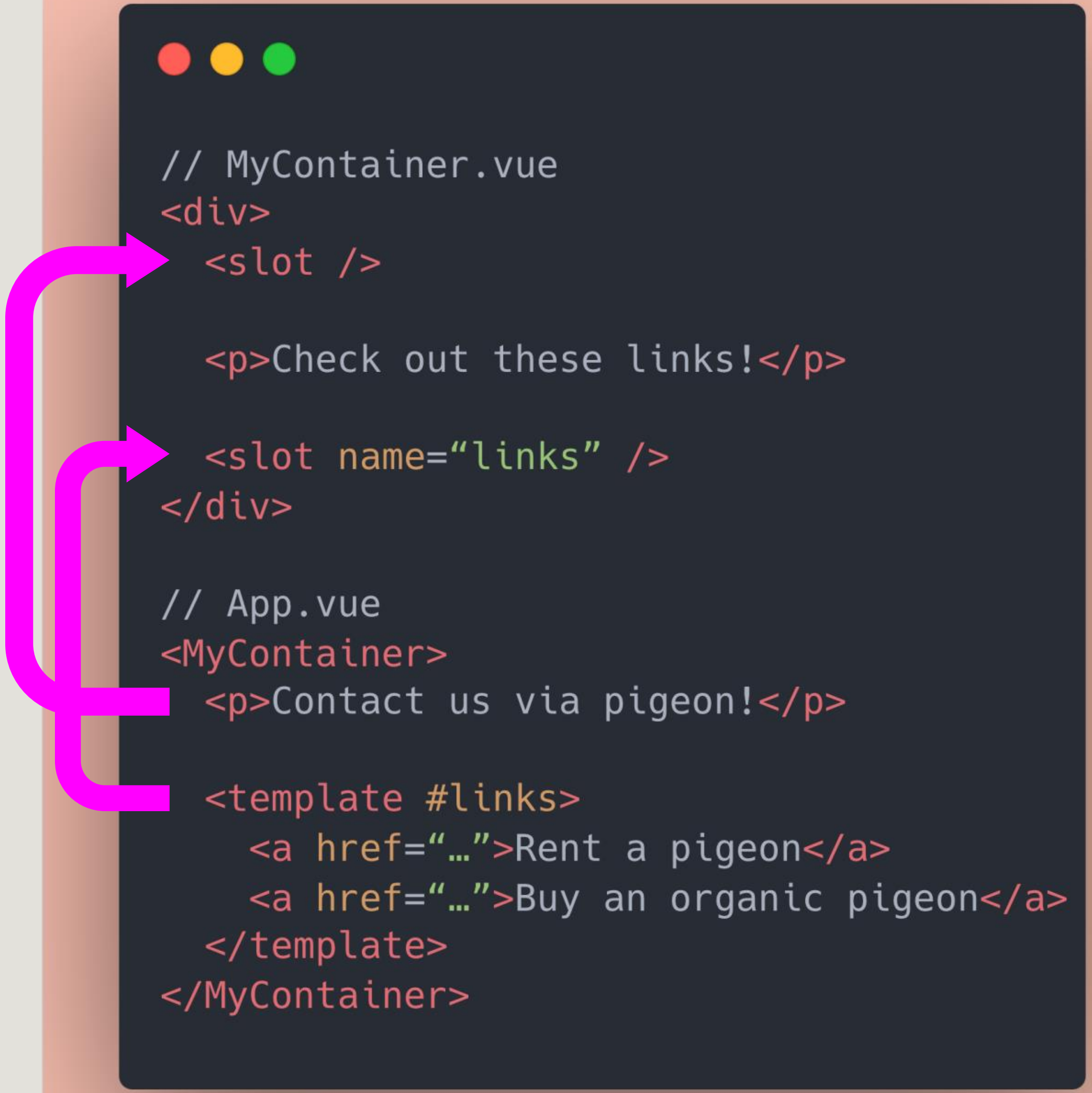
- Provide data to all components in the component tree.
- Inject some provided data into a component.
- No need to drill the data as props through multiple components.



```
// 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.



```
// MyContainer.vue
<div>
  <slot />

  <p>Check out these links!</p>

  <slot name="links" />
</div>

// App.vue
<MyContainer>
  <p>Contact us via pigeon!</p>

  <template #links>
    <a href="…">Rent a pigeon</a>
    <a href="…">Buy an organic pigeon</a>
  </template>
</MyContainer>
```

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!
```

Questions?

Up next: Backend Integration

knowit

Introduction to Declarative DOM Manipulation

Backend Integration

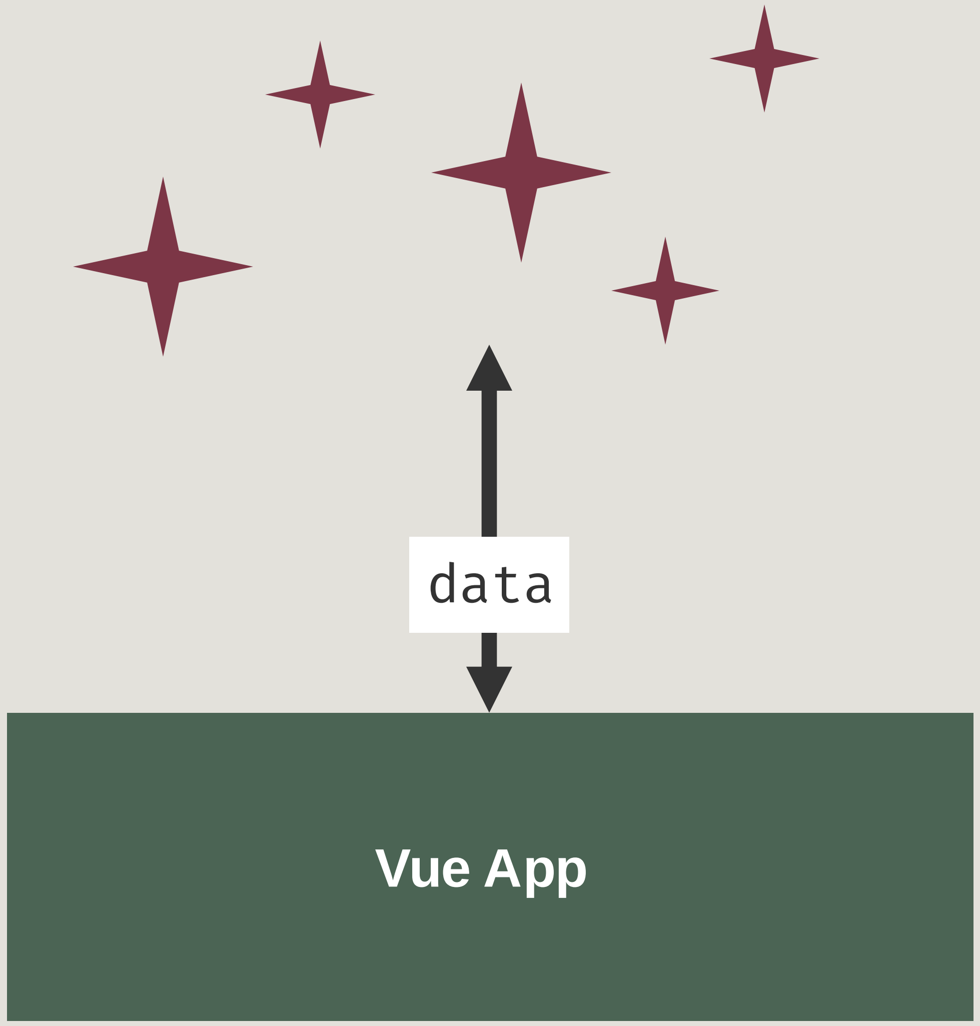
knowit

Backend Integration

1. Where data comes from
2. Fetching data from a REST API
3. Sending data to a REST API

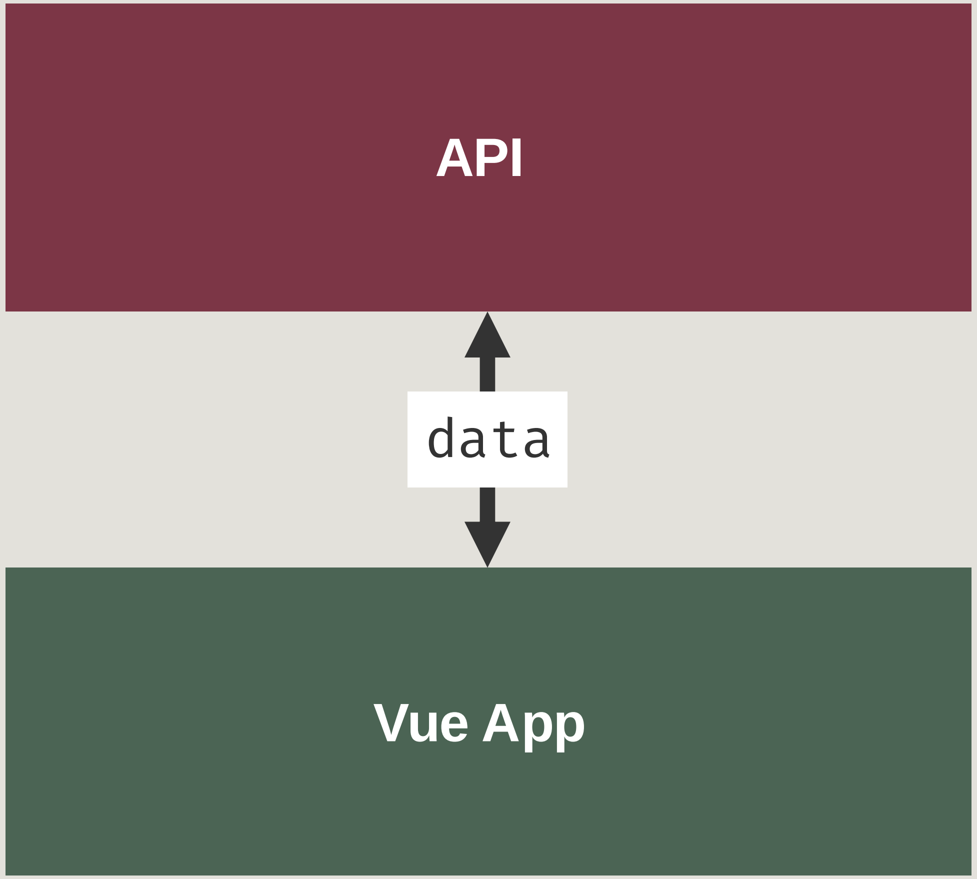
1. Where data comes from

- **Scenario:** An app wants data



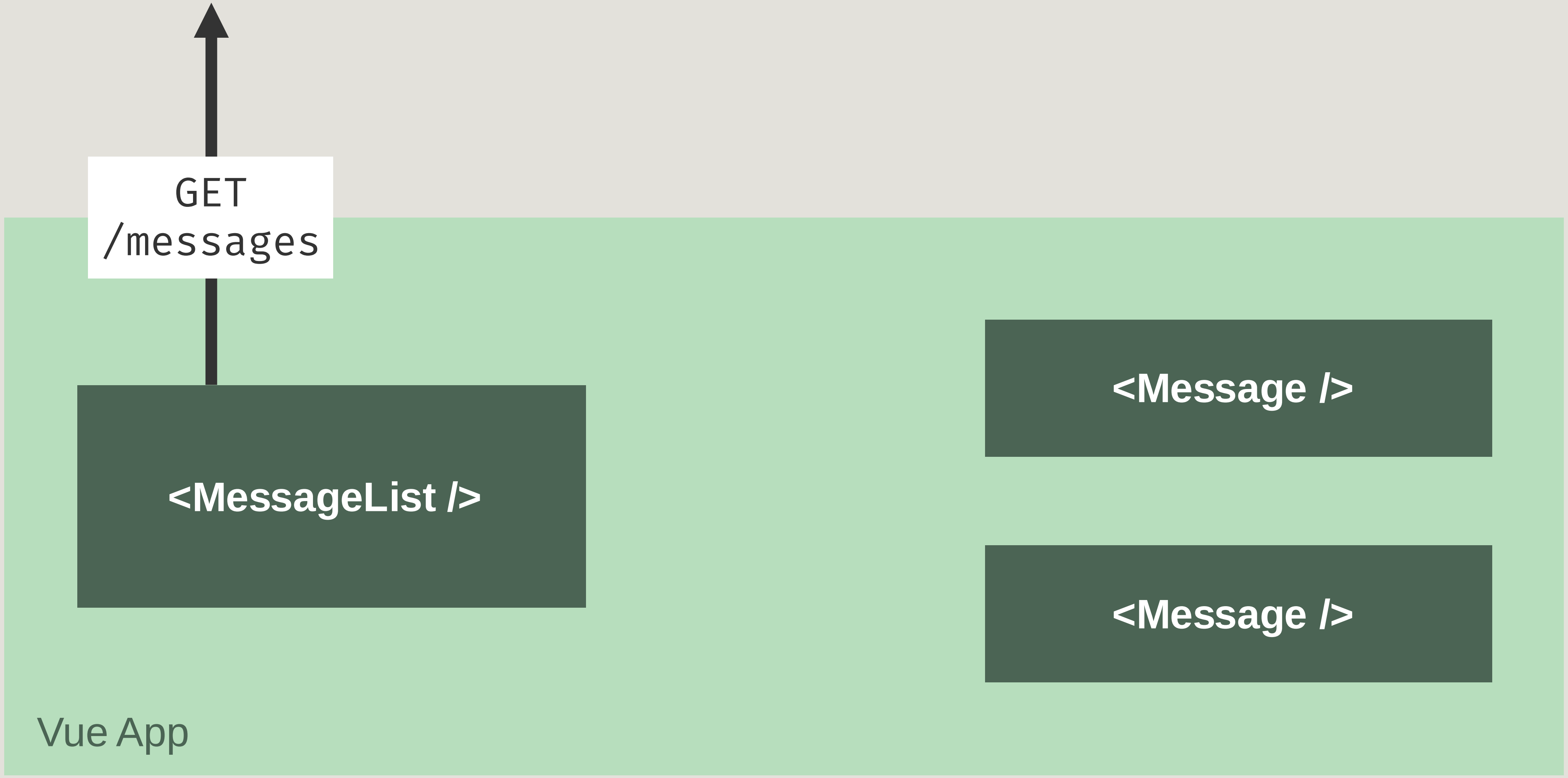
1. Where data comes from

- Usually through an API
  - / REST
  - / GraphQL
  - / WebSocket
  - / etc.



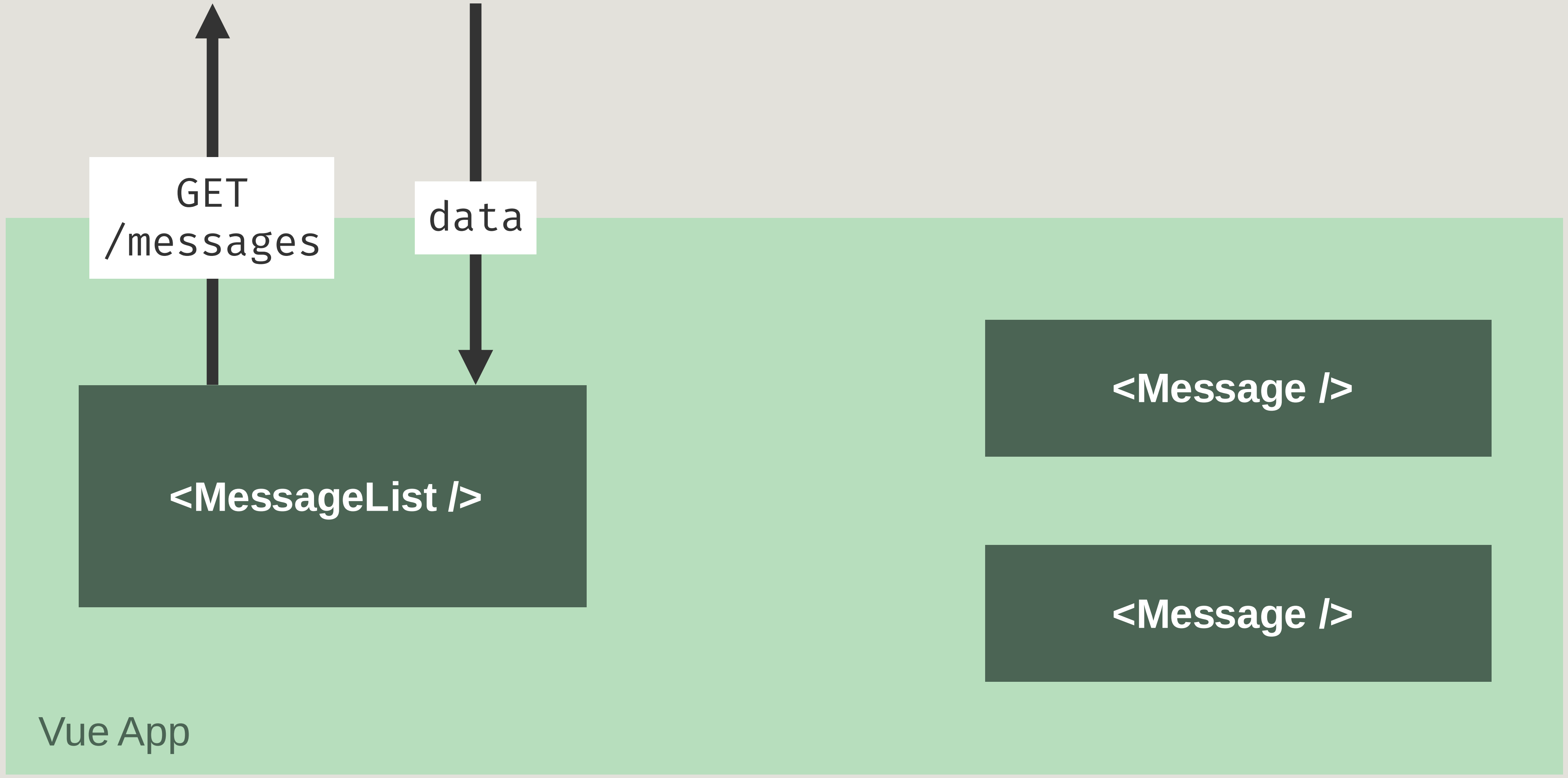
knowit

1. Where data comes from



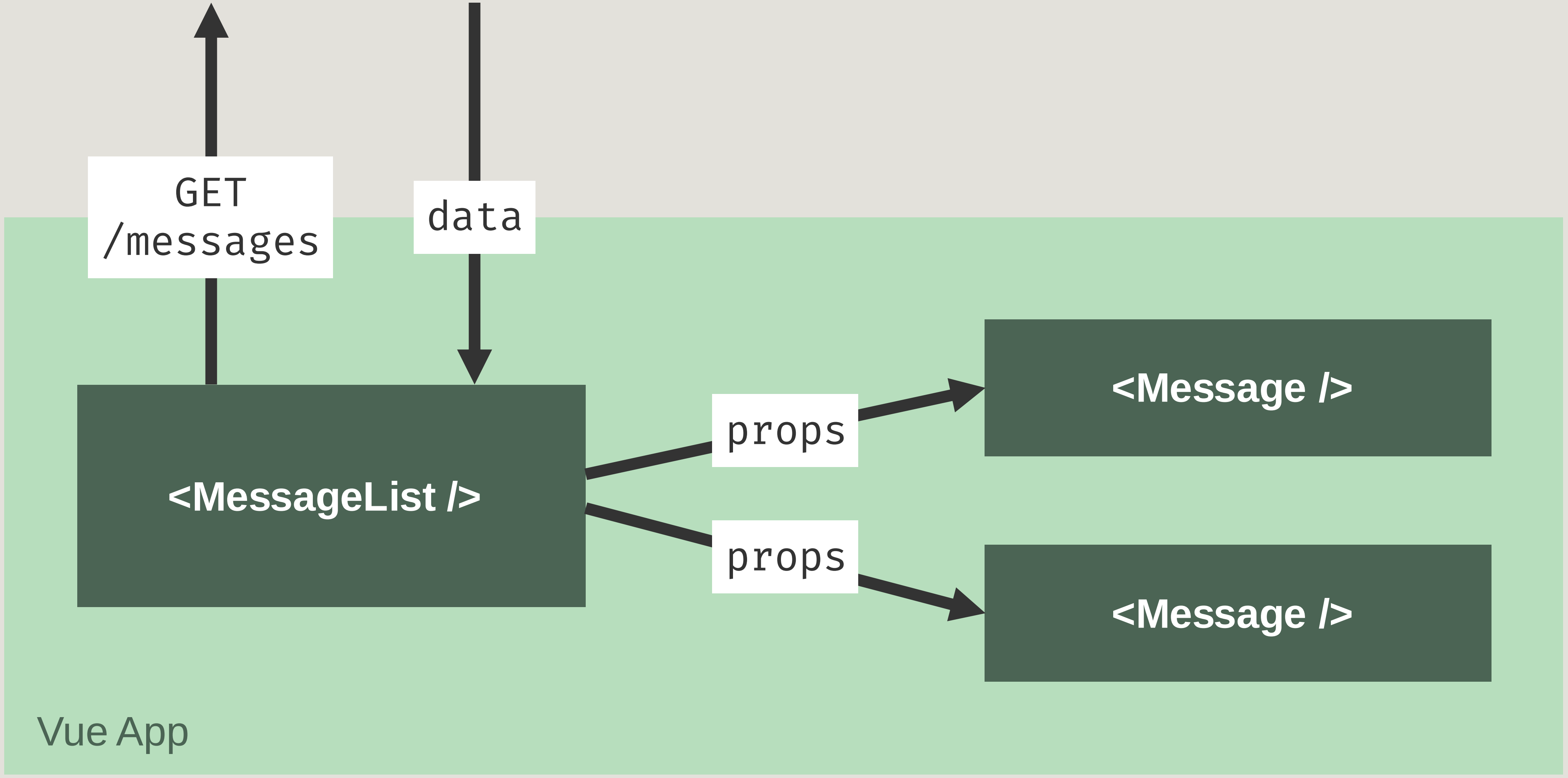
knowit

1. Where data comes from



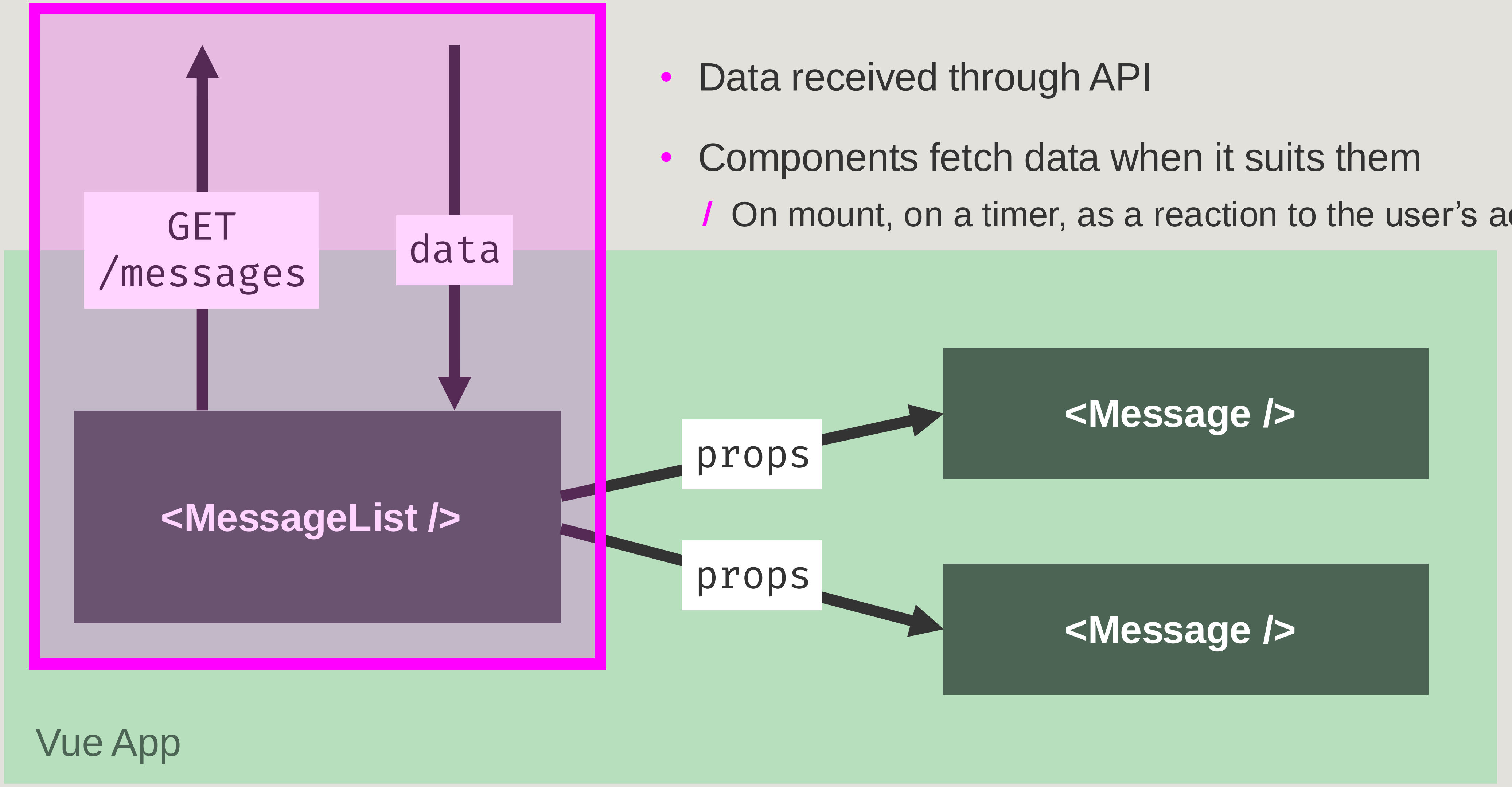
knowit

1. Where data comes from



knowit

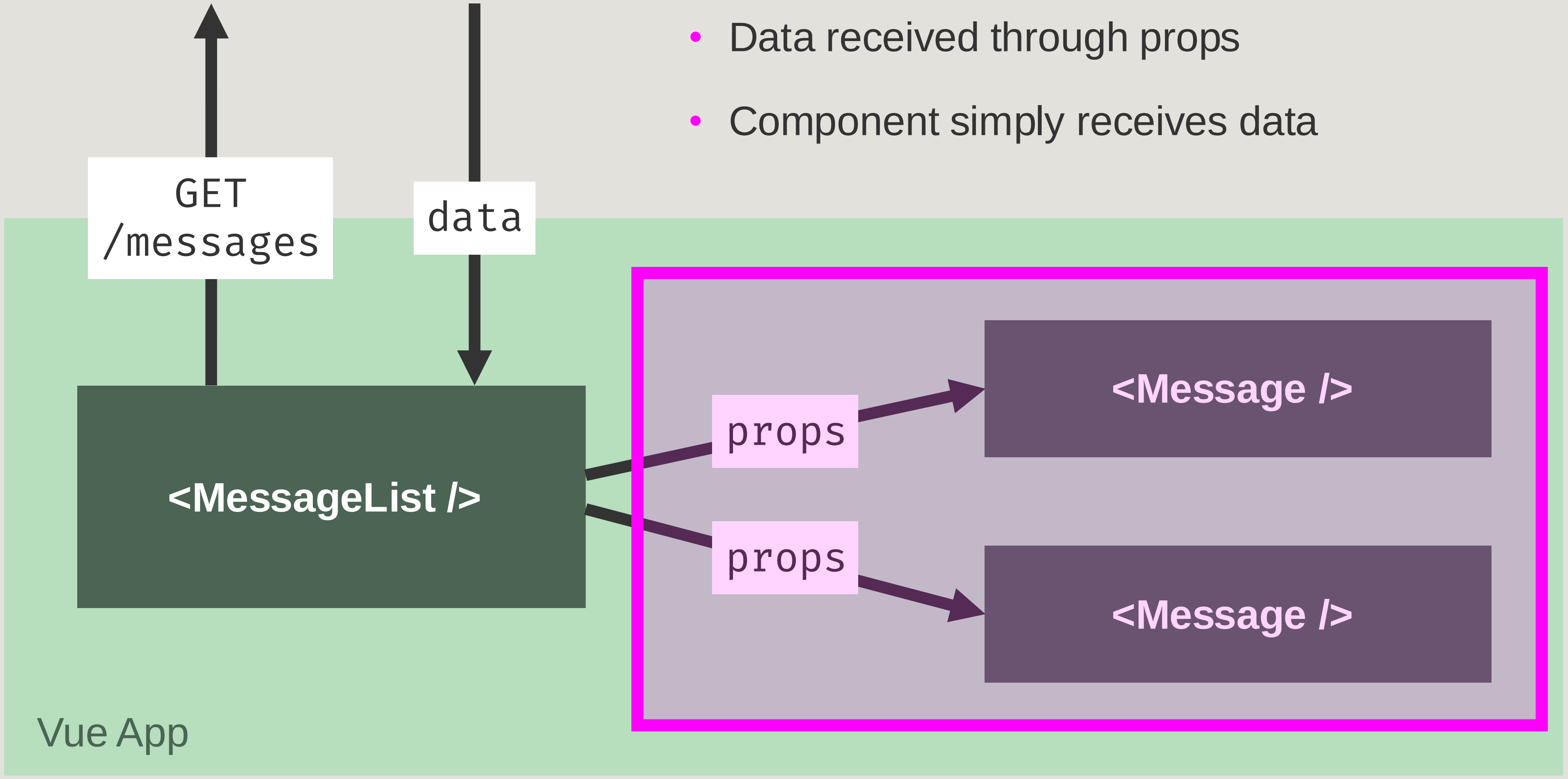
1. Where data comes from



- Data received through API
- Components fetch data when it suits them
  / On mount, on a timer, as a reaction to the user’s actions, …

1. Where data comes from

- Data received through props
- Component simply receives data



knowit

2. Fetching Data From a REST API

1. Construct an HTTP request
2. Send the request
3. Handle the response

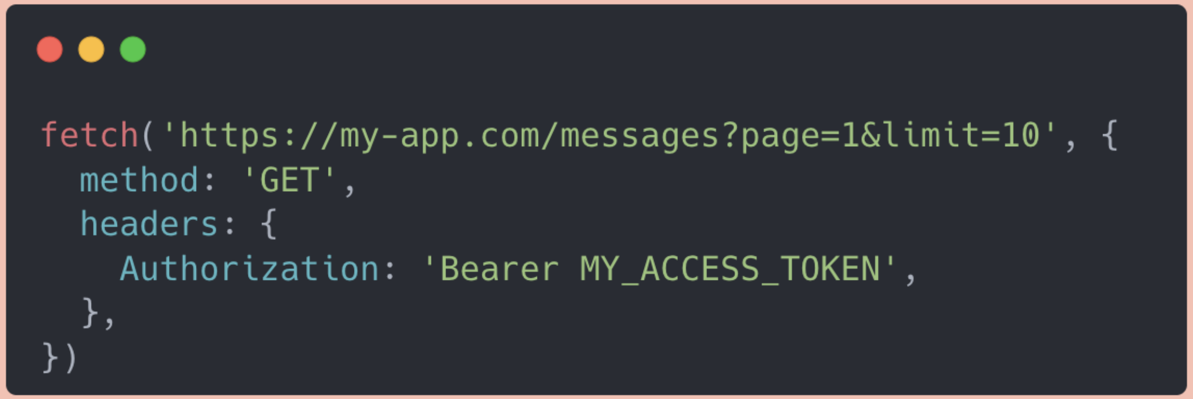
Construct an HTTP Request

- Endpoint URI
- HTTP method
  - / GET, POST, PUT, DELETE, …
- Headers
  - / Authorization, Content-Type, …
- Data
  - / Query parameters, body, …

Endpoint URI	my-app.com/messages
HTTP method	GET
Headers	Authorization: “Bearer MY_ACCESS_TOKEN”
Data	Query parameters • Page: 1 • Limit: 10

Construct an HTTP Request

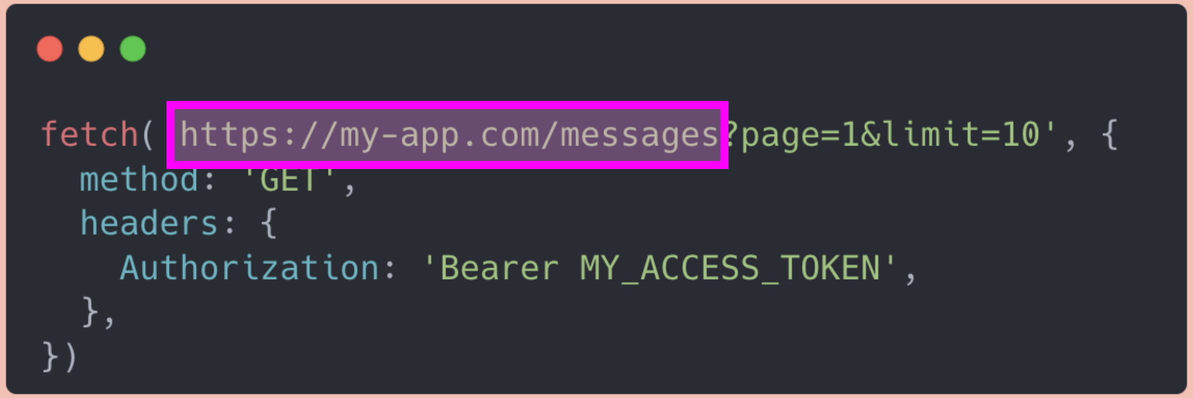
Endpoint URI	my-app.com/messages
HTTP method	GET
Headers	Authorization: “Bearer MY_ACCESS_TOKEN”
Data	Query parameters • Page: 1 • Limit: 10



```
fetch('https://my-app.com/messages?page=1&limit=10', {
  method: 'GET',
  headers: {
    Authorization: 'Bearer MY_ACCESS_TOKEN',
  },
})
```

Construct an HTTP Request

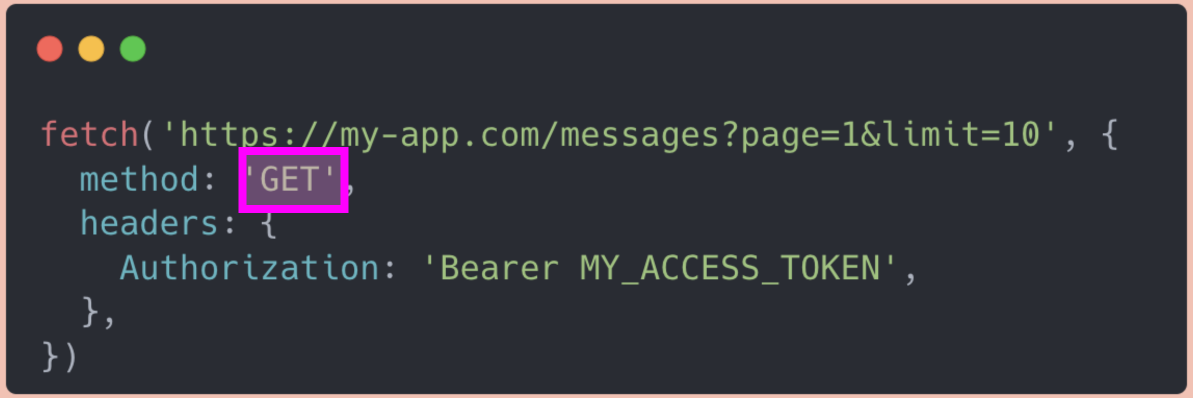
Endpoint URI	my-app.com/messages
HTTP method	GET
Headers	Authorization: “Bearer MY_ACCESS_TOKEN”
Data	Query parameters • Page: 1 • Limit: 10



```
fetch('https://my-app.com/messages?page=1&limit=10', {
  method: 'GET',
  headers: {
    Authorization: 'Bearer MY_ACCESS_TOKEN',
  },
})
```

Construct an HTTP Request

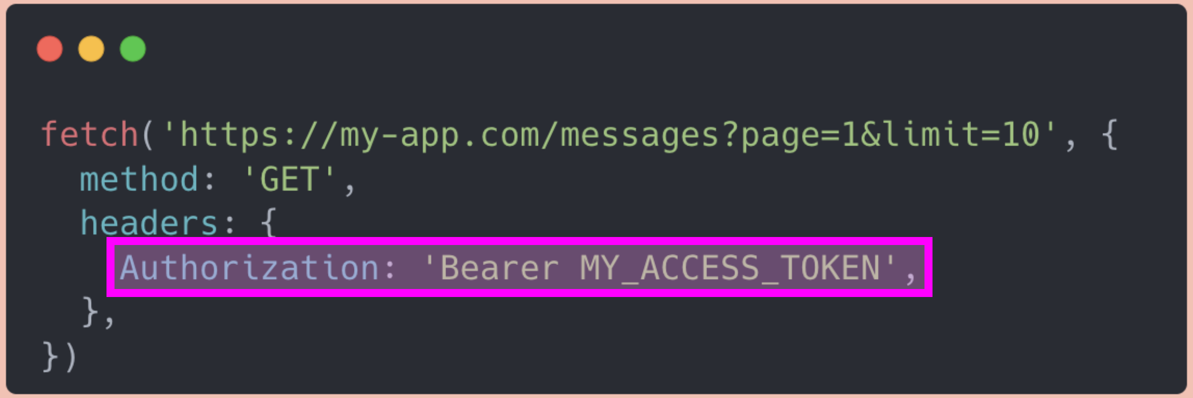
Endpoint URI	my-app.com/messages
HTTP method	GET
Headers	Authorization: “Bearer MY_ACCESS_TOKEN”
Data	Query parameters • Page: 1 • Limit: 10



```
fetch('https://my-app.com/messages?page=1&limit=10', {
  method: 'GET',
  headers: {
    Authorization: 'Bearer MY_ACCESS_TOKEN',
  },
})
```

Construct an HTTP Request

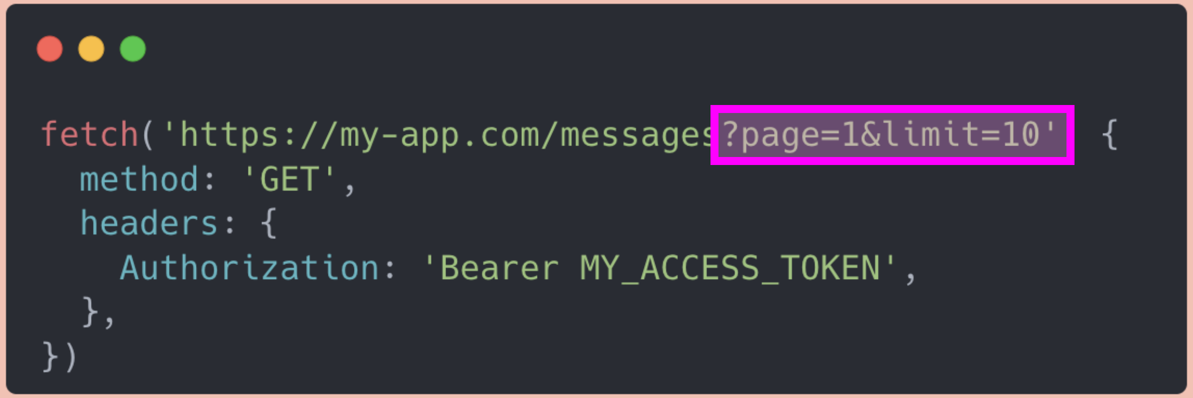
Endpoint URI	my-app.com/messages
HTTP method	GET
Headers	Authorization: “Bearer MY_ACCESS_TOKEN”
Data	Query parameters • Page: 1 • Limit: 10



```
fetch('https://my-app.com/messages?page=1&limit=10', {
  method: 'GET',
  headers: {
    Authorization: 'Bearer MY_ACCESS_TOKEN',
  },
})
```

Construct an HTTP Request

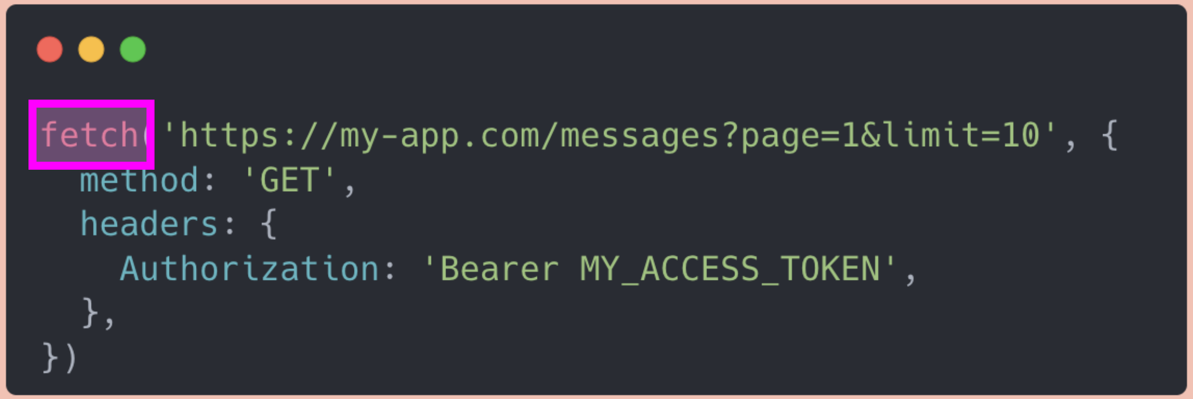
Endpoint URI	my-app.com/messages
HTTP method	GET
Headers	Authorization: “Bearer MY_ACCESS_TOKEN”
Data	Query parameters • Page: 1 • Limit: 10



```
fetch('https://my-app.com/messages?page=1&limit=10' {
  method: 'GET',
  headers: {
    Authorization: 'Bearer MY_ACCESS_TOKEN',
  },
})
```

Send the Request

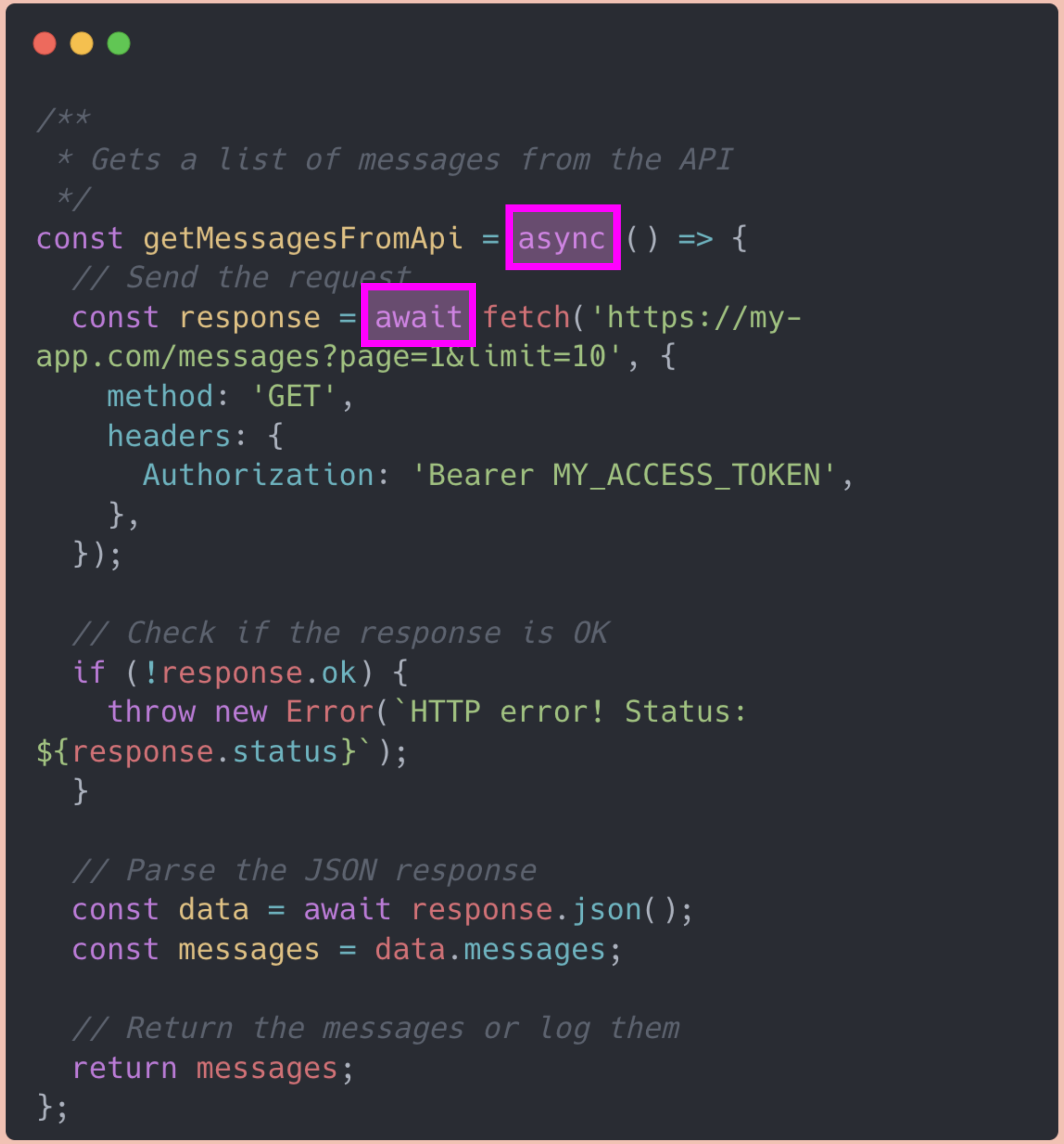
- Use an HTTP client library
  - / Fetch
  - / Axios
  - / …



```
fetch('https://my-app.com/messages?page=1&limit=10', {
  method: 'GET',
  headers: {
    Authorization: 'Bearer MY_ACCESS_TOKEN',
  },
})
```

Handle the Response

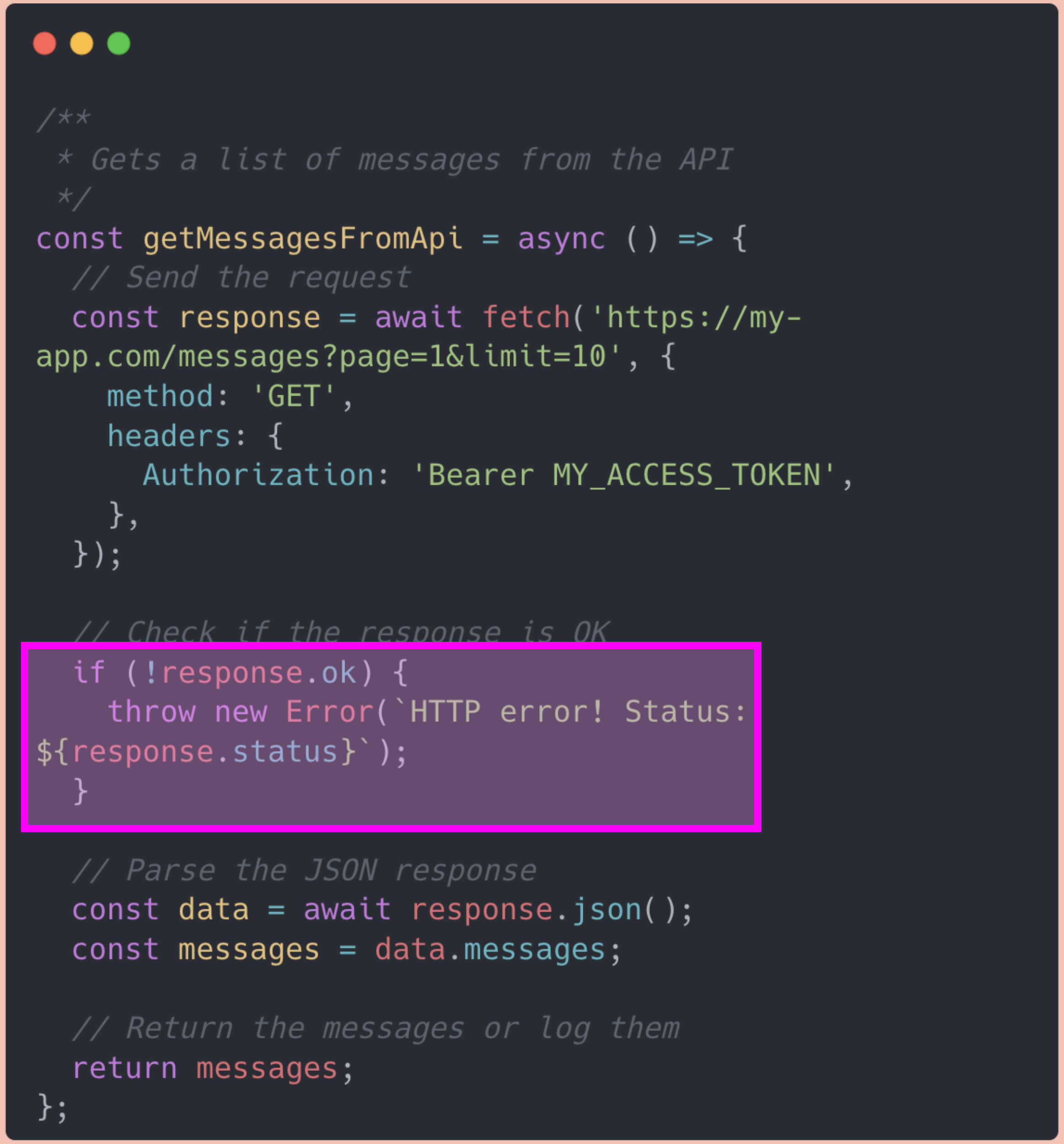
- Wait for the request to complete



```
/**
 * Gets a list of messages from the API
 */
const getMessagesFromApi = async () => {
  // Send the request
  const response = await fetch('https://my-
app.com/messages?page=1&limit=10', {
    method: 'GET',
    headers: {
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
  });

  // Check if the response is OK
  if (!response.ok) {
    throw new Error(`HTTP error! Status:
${response.status}`);
  }

  // Parse the JSON response
  const data = await response.json();
  const messages = data.messages;

  // Return the messages or log them
  return messages;
};
```

Handle the Response

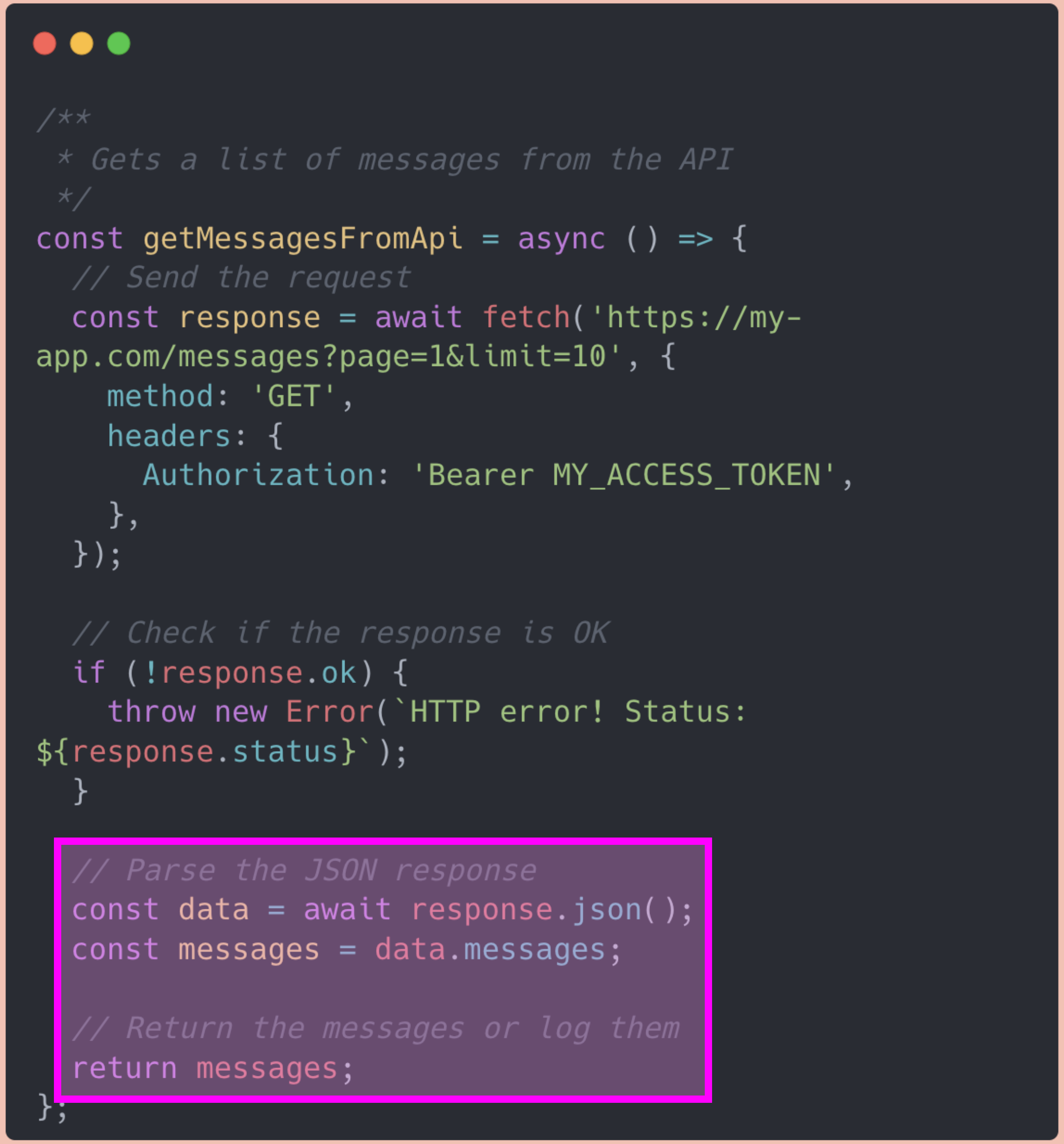
- Wait for the request to complete
- Check if the responses is OK



```
/**
 * Gets a list of messages from the API
 */
const getMessagesFromApi = async () => {
  // Send the request
  const response = await fetch('https://my-
app.com/messages?page=1&limit=10', {
    method: 'GET',
    headers: {
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
  });

  // Check if the response is OK
  if (!response.ok) {
    throw new Error(`HTTP error! Status:
${response.status}`);
  }

  // Parse the JSON response
  const data = await response.json();
  const messages = data.messages;

  // Return the messages or log them
  return messages;
};
```

Handle the Response

- Wait for the request to complete
- Check if the responses is OK
- Get data from the response

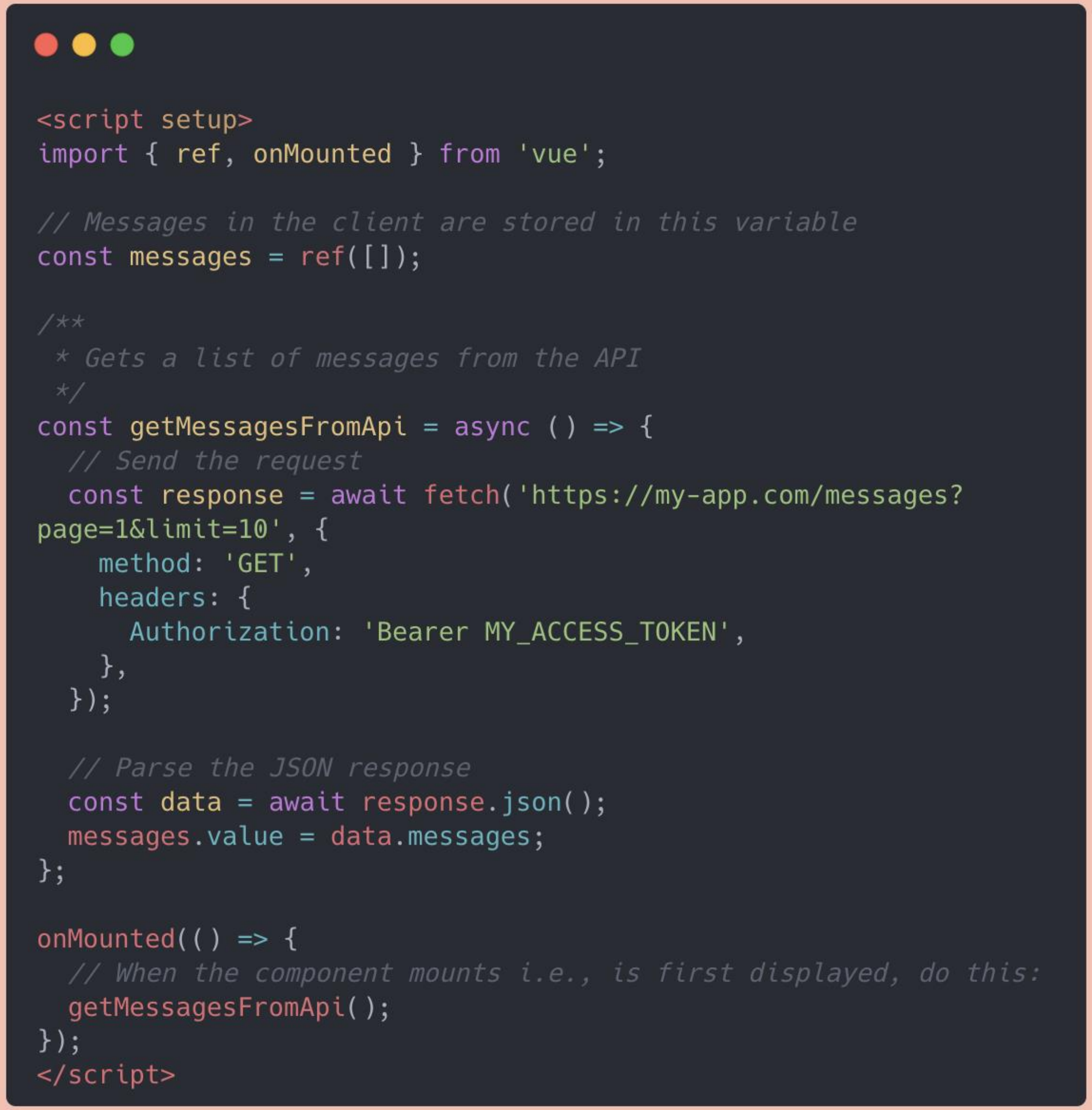


```
/**
 * Gets a list of messages from the API
 */
const getMessagesFromApi = async () => {
  // Send the request
  const response = await fetch('https://my-
app.com/messages?page=1&limit=10', {
    method: 'GET',
    headers: {
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
  });

  // Check if the response is OK
  if (!response.ok) {
    throw new Error(`HTTP error! Status:
${response.status}`);
  }

  // Parse the JSON response
  const data = await response.json();
  const messages = data.messages;

  // Return the messages or log them
  return messages;
};
```

Handle the Response

- Vue example

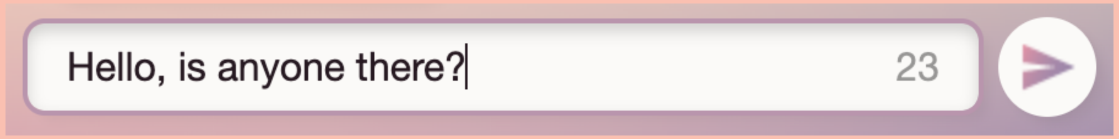
<MessageList />



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

// Messages in the client are stored in this variable
const messages = ref([]);

/**
 * Gets a list of messages from the API
 */
const getMessagesFromApi = async () => {
  // Send the request
  const response = await fetch('https://my-app.com/messages?
page=1&limit=10', {
    method: 'GET',
    headers: {
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
  });

  // Parse the JSON response
  const data = await response.json();
  messages.value = data.messages;
};

onMounted(() => {
  // When the component mounts i.e., is first displayed, do this:
  getMessagesFromApi();
});
</script>
```

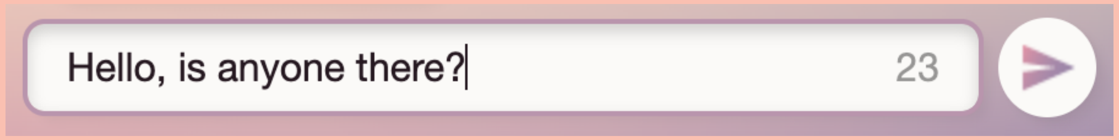
3. Sending Data to a REST API

- **Scenario:** The user wants to send a message to a chat.
  - They type a message and hit “send”.



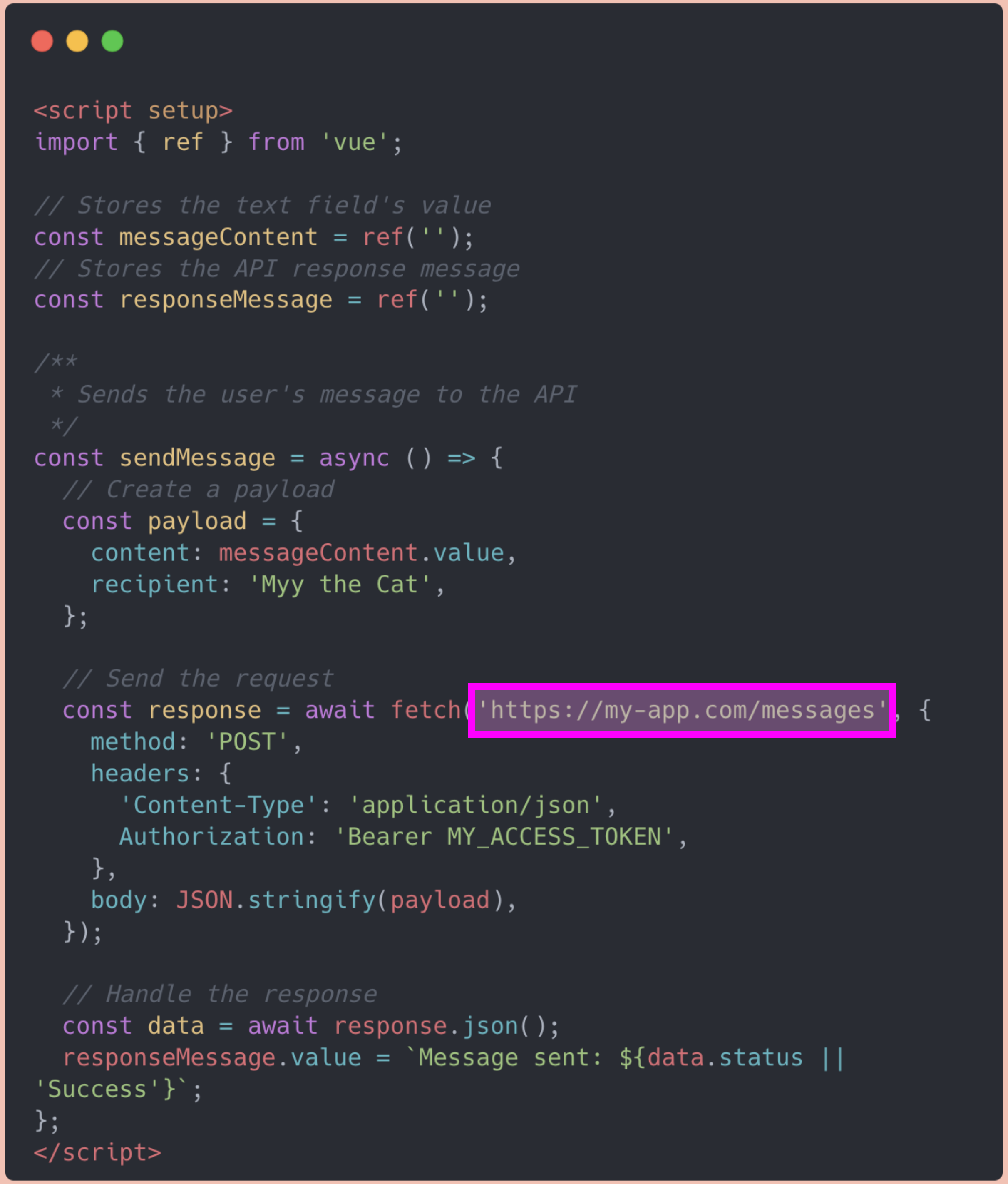
3. Sending Data to a REST API

1. Construct an HTTP request
2. Send the request
3. Handle the response


Hello, is anyone there?

Construct an HTTP Request

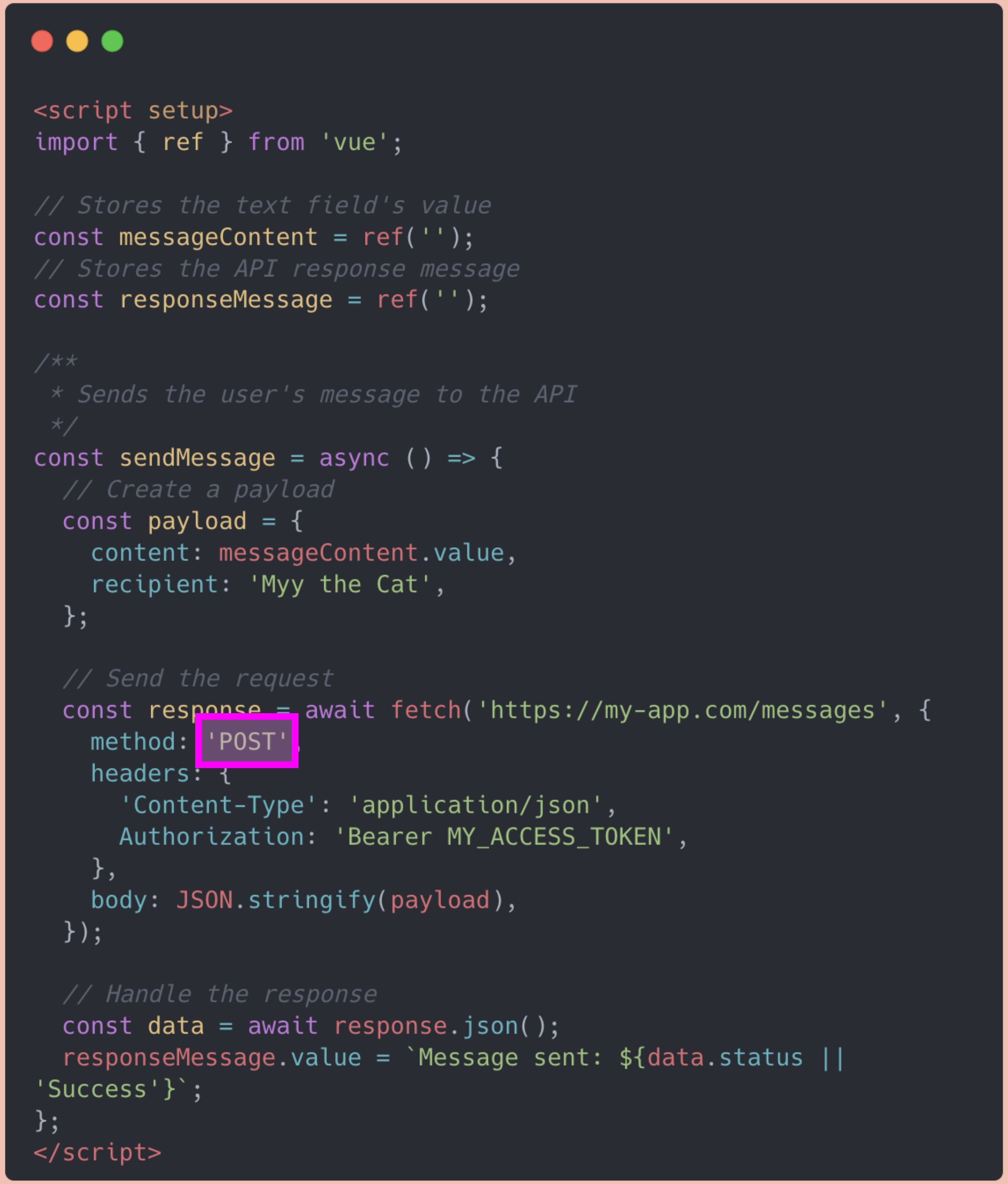
Endpoint URI	my-app.com/messages
HTTP method	POST
Headers	- Content-Type - Authorization
Data	Body - Content - Recipient



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

// Stores the text field's value
const messageContent = ref('');
// Stores the API response message
const responseMessage = ref('');

/**
 * Sends the user's message to the API
 */
const sendMessage = async () => {
  // Create a payload
  const payload = {
    content: messageContent.value,
    recipient: 'Myy the Cat',
  };

  // Send the request
  const response = await fetch('https://my-app.com/messages', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
    body: JSON.stringify(payload),
  });

  // Handle the response
  const data = await response.json();
  responseMessage.value = `Message sent: ${data.status ||
'Success'}`;
};
</script>
```

Construct an HTTP Request

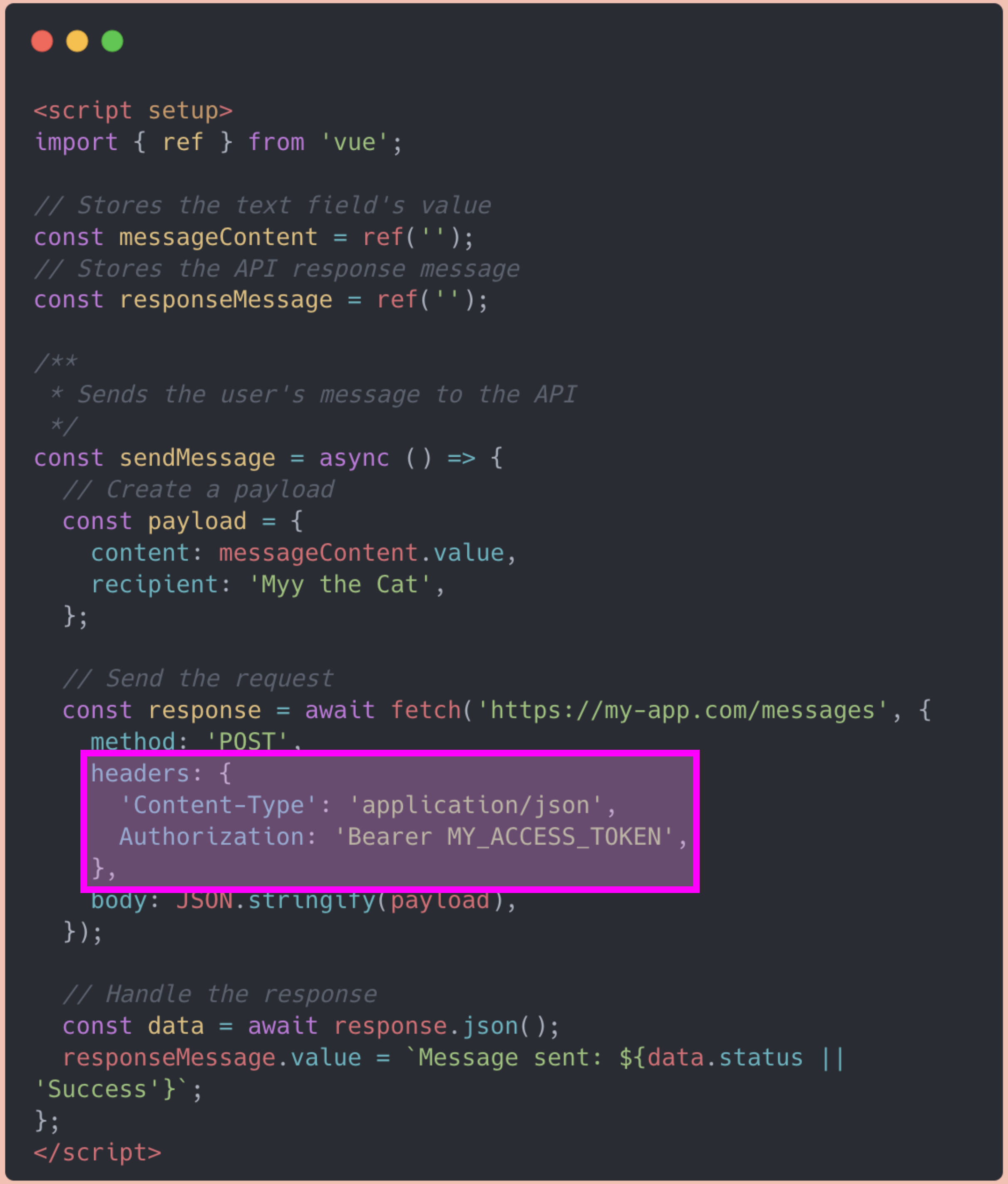
Endpoint URI	my-app.com/messages
HTTP method	POST
Headers	- Content-Type - Authorization
Data	Body - Content - Recipient



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

// Stores the text field's value
const messageContent = ref('');
// Stores the API response message
const responseMessage = ref('');

/**
 * Sends the user's message to the API
 */
const sendMessage = async () => {
  // Create a payload
  const payload = {
    content: messageContent.value,
    recipient: 'Myy the Cat',
  };

  // Send the request
  const response = await fetch('https://my-app.com/messages', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
    body: JSON.stringify(payload),
  });

  // Handle the response
  const data = await response.json();
  responseMessage.value = `Message sent: ${data.status ||
'Success'}`;
};
</script>
```

Construct an HTTP Request

Endpoint URI	my-app.com/messages
HTTP method	POST
Headers	- Content-Type - Authorization
Data	Body - Content - Recipient

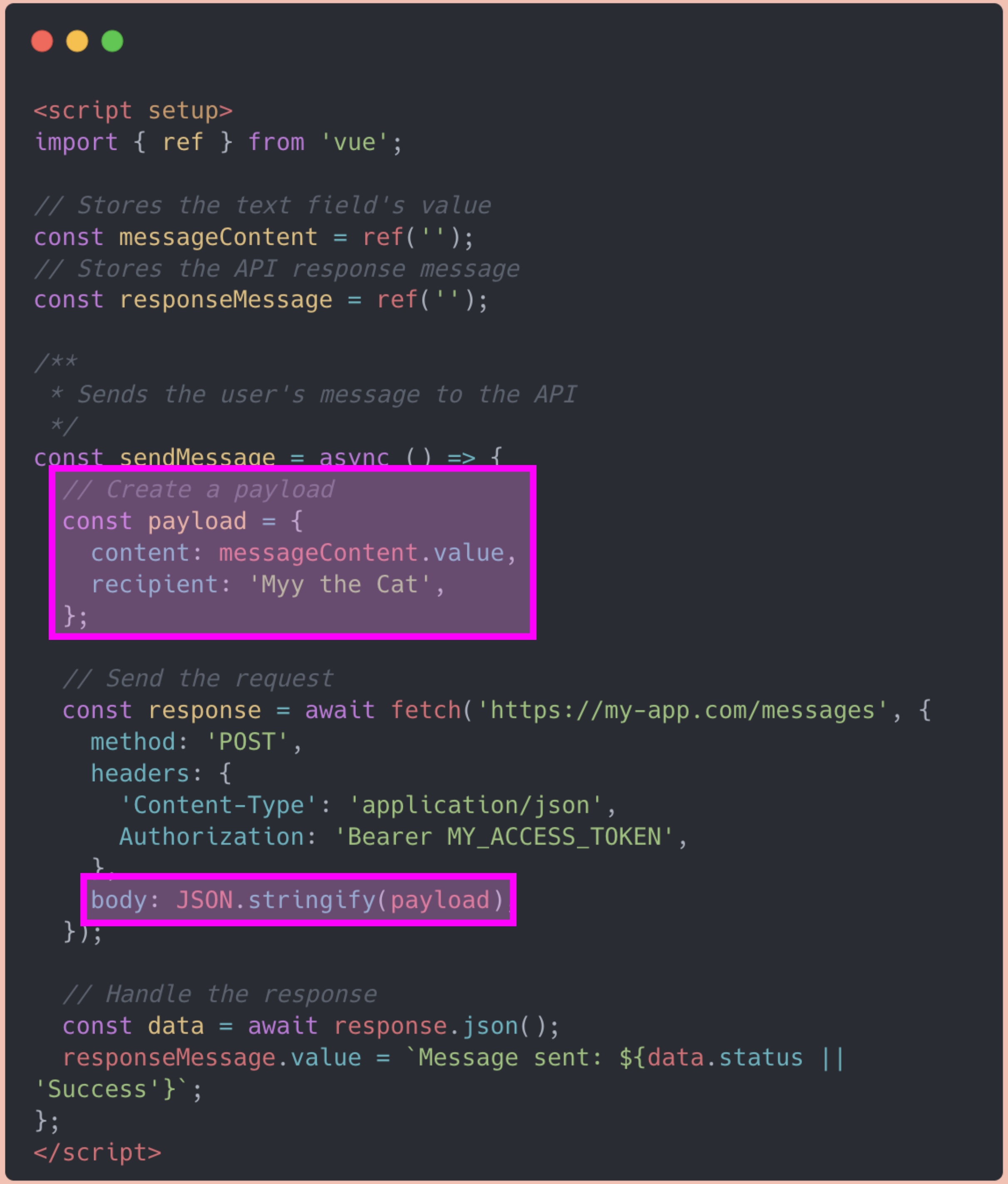


```
<script setup>
import { ref } from 'vue';

// Stores the text field's value
const messageContent = ref('');
// Stores the API response message
const responseMessage = ref('');

/**
 * Sends the user's message to the API
 */
const sendMessage = async () => {
  // Create a payload
  const payload = {
    content: messageContent.value,
    recipient: 'Myy the Cat',
  };

  // Send the request
  const response = await fetch('https://my-app.com/messages', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
    body: JSON.stringify(payload),
  });

  // Handle the response
  const data = await response.json();
  responseMessage.value = `Message sent: ${data.status ||
'Success'}`;
};
</script>
```

/ 3. Sending Data to a REST API

Construct an HTTP Request

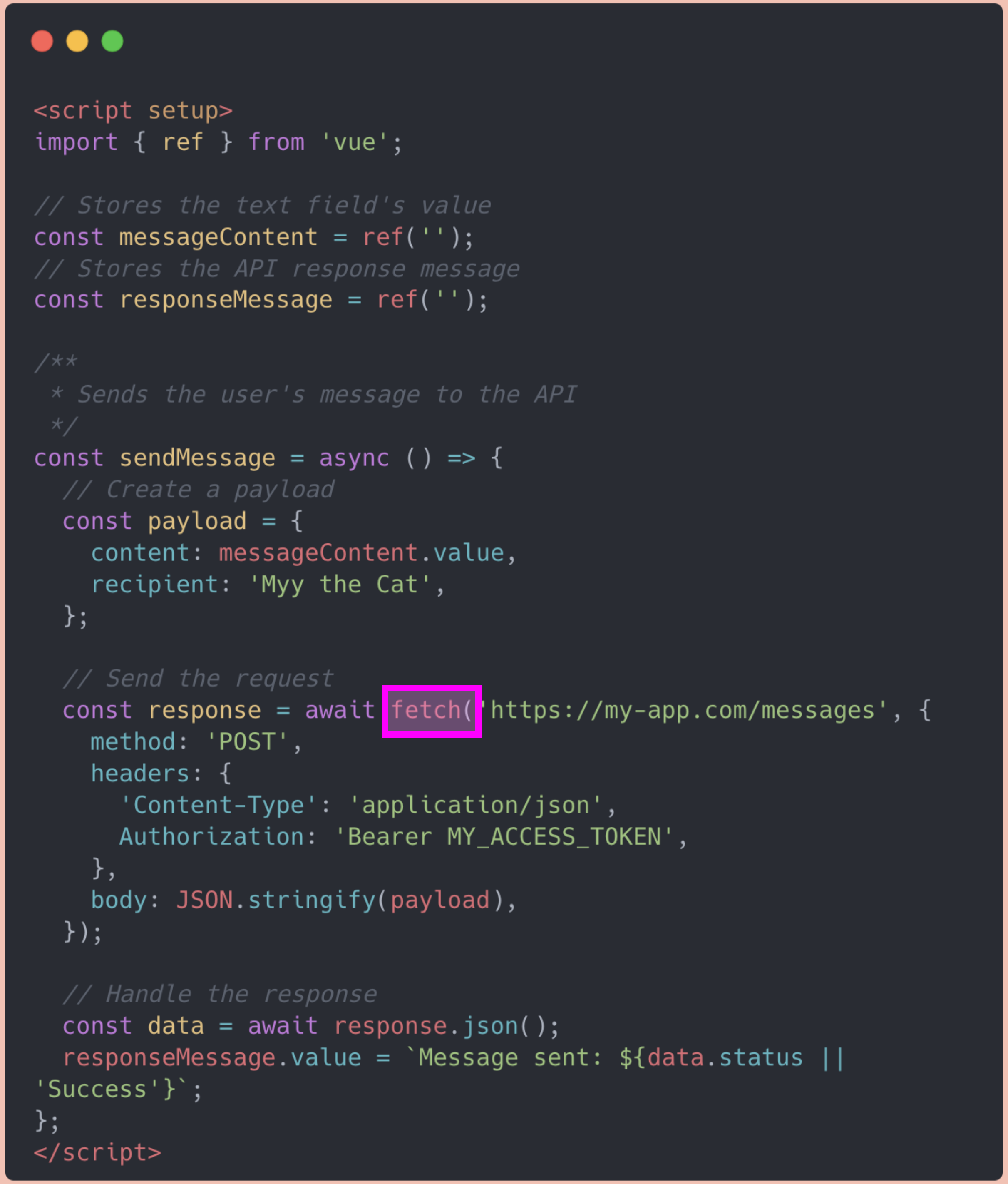
Endpoint URI	my-app.com/messages
HTTP method	POST
Headers	- Content-Type - Authorization
Data	Body - Content - Recipient



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

// Stores the text field's value
const messageContent = ref('');
// Stores the API response message
const responseMessage = ref('');

/**
 * Sends the user's message to the API
 */
const sendMessage = async () => {
  // Create a payload
  const payload = {
    content: messageContent.value,
    recipient: 'Myy the Cat',
  };

  // Send the request
  const response = await fetch('https://my-app.com/messages', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
    body: JSON.stringify(payload)
  });

  // Handle the response
  const data = await response.json();
  responseMessage.value = `Message sent: ${data.status ||
'Success'}`;
};
</script>
```

/ 3. Sending Data to a REST API

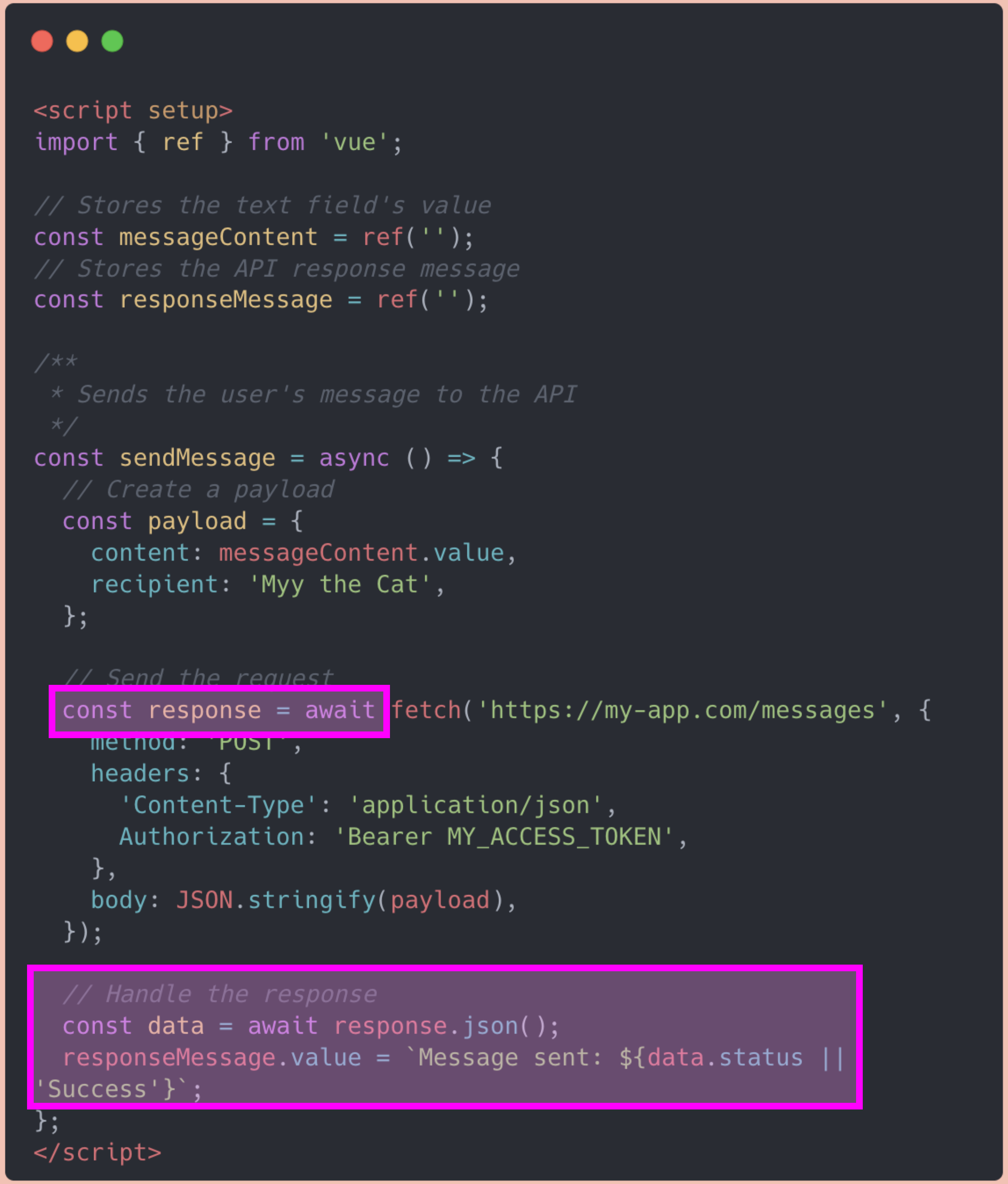
Send the Request



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

// Stores the text field's value
const messageContent = ref('');
// Stores the API response message
const responseMessage = ref('');

/**
 * Sends the user's message to the API
 */
const sendMessage = async () => {
  // Create a payload
  const payload = {
    content: messageContent.value,
    recipient: 'Myy the Cat',
  };

  // Send the request
  const response = await fetch('https://my-app.com/messages', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
    body: JSON.stringify(payload),
  });

  // Handle the response
  const data = await response.json();
  responseMessage.value = `Message sent: ${data.status ||
'Success'}`;
};
</script>
```

/ 3. Sending Data to a REST API

Handle the Response



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

// Stores the text field's value
const messageContent = ref('');
// Stores the API response message
const responseMessage = ref('');

/**
 * Sends the user's message to the API
 */
const sendMessage = async () => {
  // Create a payload
  const payload = {
    content: messageContent.value,
    recipient: 'Myy the Cat',
  };

  // Send the request
  const response = await fetch('https://my-app.com/messages', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer MY_ACCESS_TOKEN',
    },
    body: JSON.stringify(payload),
  });

  // Handle the response
  const data = await response.json();
  responseMessage.value = `Message sent: ${data.status ||
'Success'}`;
};
</script>
```

knowit

↓

Questions?

Up next: Workshop

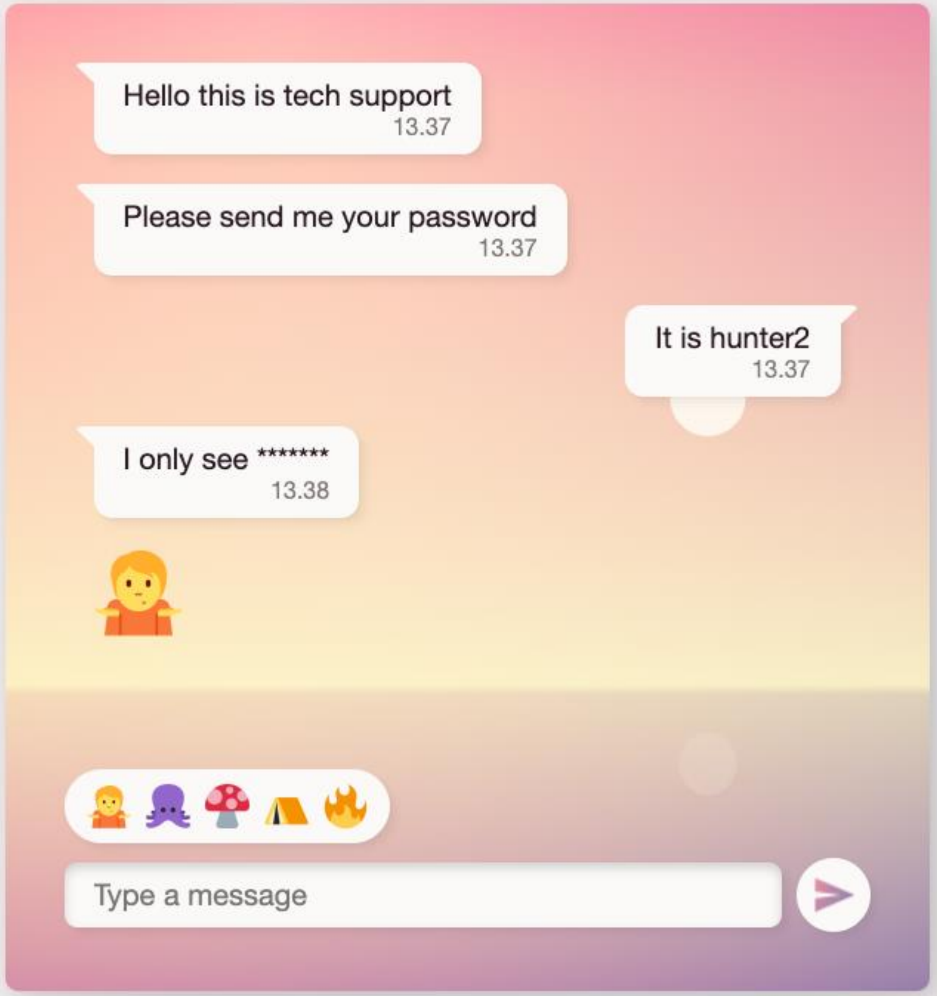
knowit

Introduction to Declarative DOM Manipulation

Workshop

knowit

Finish a chat app


Hello this is tech support
13.37

Please send me your password
13.37

It is hunter2
13.37

I only see *******
13.38

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.

knowit

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

knowit

Thanks