



# Production-Grade Vue

Frontend *Masters*

---

Ben Hong

<https://www.bencodezen.io>



**BEN HONG**  
Senior DX Engineer  
Netlify

[@bencodezen](https://twitter.com/bencodezen)

 **Vue.js**

Docs ▾ API Reference Ecosystem ▾ Support Vue ▾ GitHub ↗ Q Search K

**Essentials**

- Installation
- Introduction**
- What is Vue.js?
- Getting Started
- Declarative Rendering
- Handling User Input
- Conditionals and Loops
- Composing with Components
- Relation to Custom Elements
- Ready for More?
- Application & Component Instances
- Template Syntax
- Data Properties and Methods
- Computed Properties and Watchers
- Class and Style Bindings
- Conditional Rendering
- List Rendering
- Event Handling
- Form Input Bindings
- Components Basics

## Introduction

 **NOTE**

Already know Vue 2 and just want to learn about what's new in Vue 3? Check out the [Migration Guide](#)!

### What is Vue.js?

Vue (pronounced /vju:/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with [modern tooling](#) and [supporting libraries](#).

If you'd like to learn more about Vue before diving in, we [created a video](#) walking through the core principles and a sample project.

### Getting Started



# SCHEDULE



Time Slot (EST)	Event
9:30AM - 10:30AM	Part 1
10:30AM - 10:45AM	☕ Short Break ☕
10:45AM - 12:00PM	Part 2
12:00PM - 1:00PM	🥪🌮 Lunch 🍱🍜
1:00PM - 3:00PM	Part 3
3:00PM - 3:15PM	☕ Short Break ☕
3:15PM - 5:30PM	Part 4

Before we get started...

# PARTICIPATION TIPS

Questions are **welcome!**

# PARTICIPATION TIPS

All slides and examples will be **public**.

*No need to hurry and copy down examples.*

# PARTICIPATION TIPS

If you need a **break**, please take one!

# PARTICIPATION TIPS

Most things are applicable to Vue 2 and 3.

*If there is anything Vue 3 specific, it'll be signified with:*



# PARTICIPATION TIPS

All code is **compromise**.

*I encourage you to question and/or disagree.*

*Your opinion and experience matter.*

*Choose what works best for you and your team.*



# LANGUAGES

TOOL

Single File Components (SFCs)

TOOL

# Single File Components (SFCs)



MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```

# **BEST PRACTICES**

## JavaScript

# DISCUSSION

# JavaScript vs TypeScript

## DISCUSSION

# JavaScript vs TypeScript

- Majority of bugs encountered are **not** due to type violations
- TypeScript does **not** inherently guarantee type safety - it requires discipline
- Full type safety in a codebase can be a significant cost to a team in terms of productivity
- Most applications would benefit from better tests and code reviews

## DISCUSSION

# JavaScript vs TypeScript

- Progressive types can be added to a codebase with JSDoc comments
- If the application is in Vue.js 2, probably not worth upgrading to TypeScript
- Starting a new project with Vue.js 3 and the team is interested in trying it out TypeScript? Go for it!



# Questions?

# **BEST PRACTICES**

## **HTML**

## **BEST PRACTICE**

All HTML should exist in .vue files  
in a <template> or render function

## BEST PRACTICE

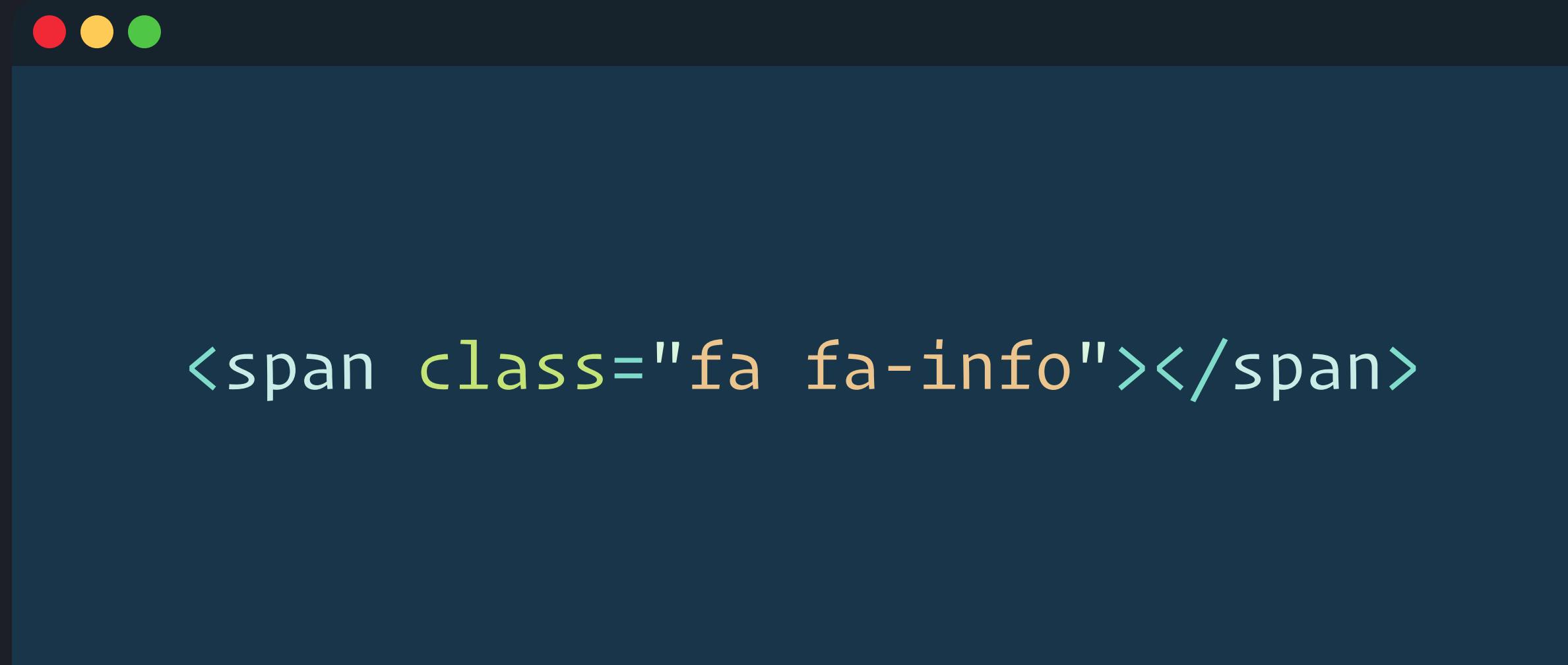
All HTML should exist in .vue files  
in a <template> or render function

- A benefit of doing this is that Vue has an opportunity to parse it before the browser does
- This allows for developer experience improvements such as:
  - Self-closing tags

## BEST PRACTICE

All HTML should exist in .vue files  
in a <template> or render function

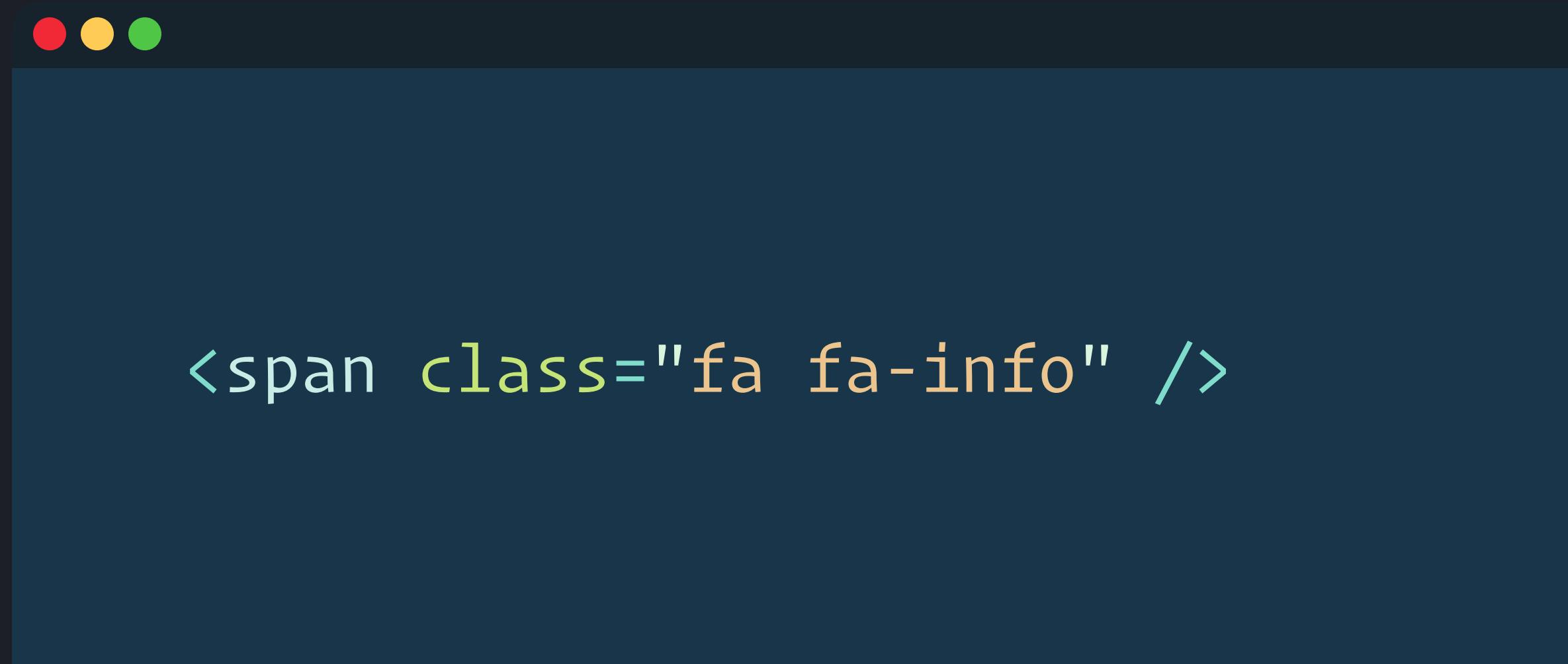
- Self-closing tags



## BEST PRACTICE

All HTML should exist in .vue files  
in a <template> or render function

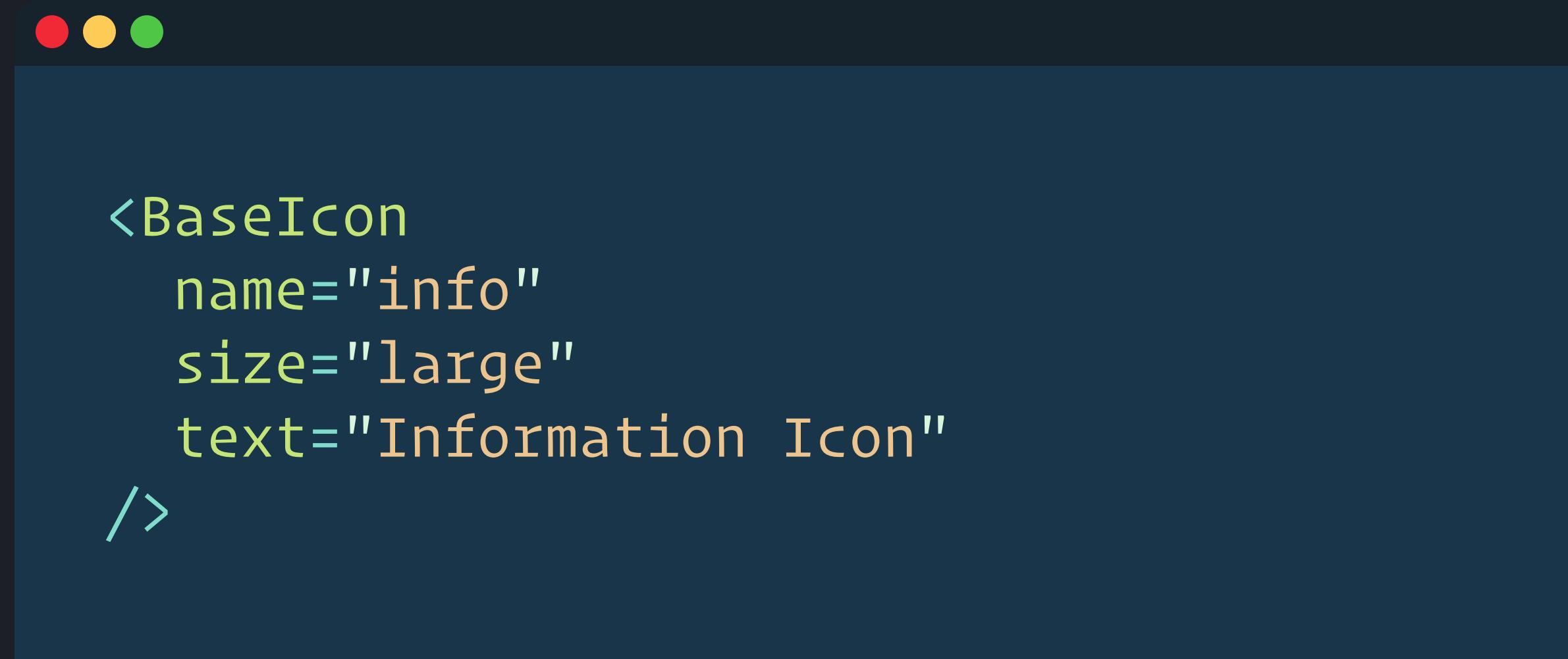
- Self-closing tags



## BEST PRACTICE

All HTML should exist in .vue files  
in a <template> or render function

- Self-closing tags



```
<BaseIcon
  name="info"
  size="large"
  text="Information Icon"
/>
```

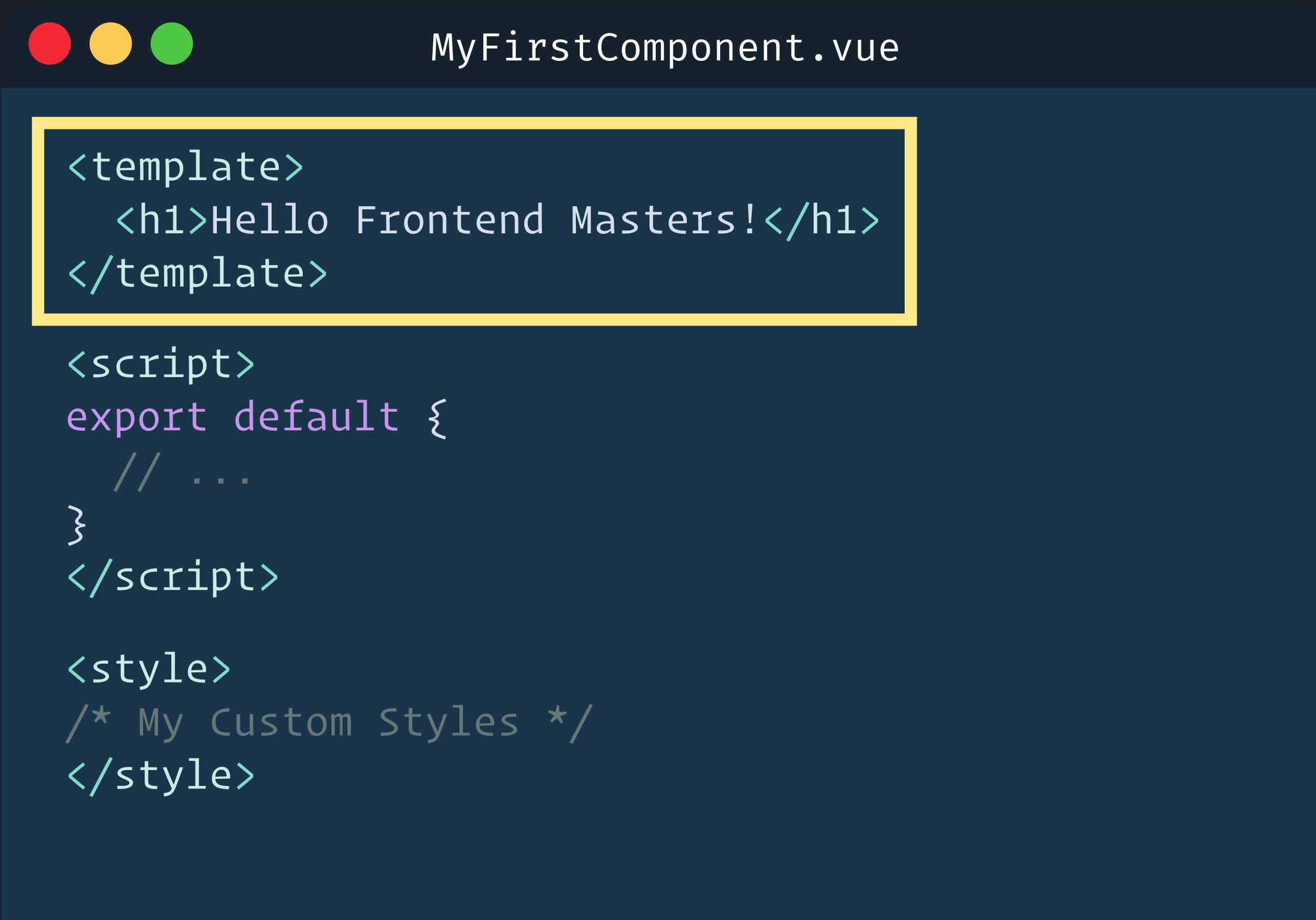
## BEST PRACTICE

# All HTML should exist in .vue files in a <template> or render function

- A benefit of doing this is that Vue has an opportunity to parse it before the browser does
- This allows for developer experience improvements such as:
  - Self-closing tags
  - Easy enhancement path if needed

# TECHNIQUE

## Template



MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}

</script>

<style>
/* My Custom Styles */
</style>
```

# TECHNIQUE

## Render Function



MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```

# TECHNIQUE

## Render Function



MyFirstComponent.vue

```
<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```

# TECHNIQUE

## Render Function



MyFirstComponent.vue

```
<script>
export default {
  render(createElement) {
    return createElement('h1', 'Hello Frontend Masters!')
  }
}
</script>

<style>
/* My Custom Styles */
</style>
```

# TECHNIQUE

## Render Function



MyFirstComponent.vue

```
<script>
import { h } from 'vue'

export default {
  render(createElement) {
    return createElement('h1', 'Hello Frontend Masters!')
  }
}
</script>

<style>
/* My Custom Styles */
</style>
```



Vue.js 3

# TECHNIQUE

## Render Function



MyFirstComponent.vue

```
<script>
import { h } from 'vue'

export default {
  render() {
    return h('h1', 'Hello Frontend Masters!')
  }
}
</script>

<style>
/* My Custom Styles */
</style>
```



Vue.js 3

# DISCUSSION

## Template vs Render Function

## DISCUSSION

# Template vs Render Function

- Templates are the most declarative way to write HTML and is recommended as the default
- Render functions are a valid alternative to templates that are great for programmatic generation of markup

# TECHNIQUE

**v-bind="{}...{}"**

**v-on="{}...{}"**

# When working with props and/or events consider...



```
<VueMultiselect
  v-bind:options="options"
  v-bind:value="value"
  v-bind:label="label"
  v-on:change="parseSelection"
  v-on:keyup="registerSelection"
  v-on:mouseover="registerHover"
/>
```

# When working with props and/or events consider...



```
<VueMultiselect
  v-bind="{
    options: options,
    value: value,
    label: label
  }"
  v-on:change="parseSelection"
  v-on:keyup="registerSelection"
  v-on:mouseover="registerHover"
/>
```

# When working with props and/or events consider...



```
<VueMultiselect
  v-bind="{
    options: options,
    value: value,
    label: label
  }"
  v-on="{
    change: parseSelection,
    keyup: registerSelection,
    mouseover: registerHover
  }"
/>
```

# When working with props and/or events consider...



```
<VueMultiselect
  v-bind="{
    options,
    value,
    label
  }"
  v-on="{
    change: parseSelection,
    keyup: registerSelection,
    mouseover: registerHover
  }"
/>
```

# When working with props and/or events consider...



```
<VueMultiselect
  v-bind="vmsProps"
  v-on="{
    change: parseSelection,
    keyup: registerSelection,
    mouseover: registerHover
  }"
/>>
```

# When working with props and/or events consider...



```
<VueMultiselect  
  v-bind="vmsProps"  
  v-on="vmsEvents"  
/>
```



# Questions?

# **BEST PRACTICES**

## CSS

# **BEST PRACTICES**

## CSS

- Limit global styles to App.vue whenever possible
- Scope all component styles with scoped styles or CSS modules



## MyRedText.vue

```
<template>
  <p class="red">
    This should be red!
  </p>
</template>
```

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

# TECHNIQUE

## Scoped Styles



## MyRedText.vue

```
<template>
  <p class="red">
    This should be red!
  </p>
</template>
```

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



## MyRedText.vue

```
<template>
  <p class="red">
    This should be red!
  </p>
</template>
```

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



## MyRedText.vue

```
<template>
  <p class="red" data-v57s8>
    This should be red!
  </p>
</template>
```

```
<style>
  .red[data-v57s8] {
    color: red;
  }
  .bold[data-v57s8] {
    font-weight: bold;
  }
</style>
```

# TECHNIQUE

# CSS Modules



## MyRedText.vue

```
<template>
  <p class="red">
    This should be red!
  </p>
</template>
```

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



## MyRedText.vue

```
<template>
  <p class="red">
    This should be red!
  </p>
</template>

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



## MyRedText.vue

```
<template>
  <p :class="$style.red">
    This should be red!
  </p>
</template>
```

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



## MyRedText.vue

```
<template>
  <p :class="$style.red">
    This should be red!
  </p>
</template>

<style>
  .MyRedText_red_3fj4x {
    color: red;
  }
  .MyRedText_bold_8fn3s {
    font-weight: bold;
  }
</style>
```



## MyRedText.vue

```
<template>
  <p class="MyRedText_red_3fj4x">
    This should be red!
  </p>
</template>

<style>
  .MyRedText_red_3fj4x {
    color: red;
  }
  .MyRedText_bold_8fn3s {
    font-weight: bold;
  }
</style>
```

# TECHNIQUE

# CSS Modules Exports

# TECHNIQUE

## CSS Modules Exports



```
<template>
  <p>Grid Padding: {{ $style.gridPadding }}</p>
</template>

<style module>
:export {
  gridPadding: 1.5rem;
}
</style>
```



# Questions?

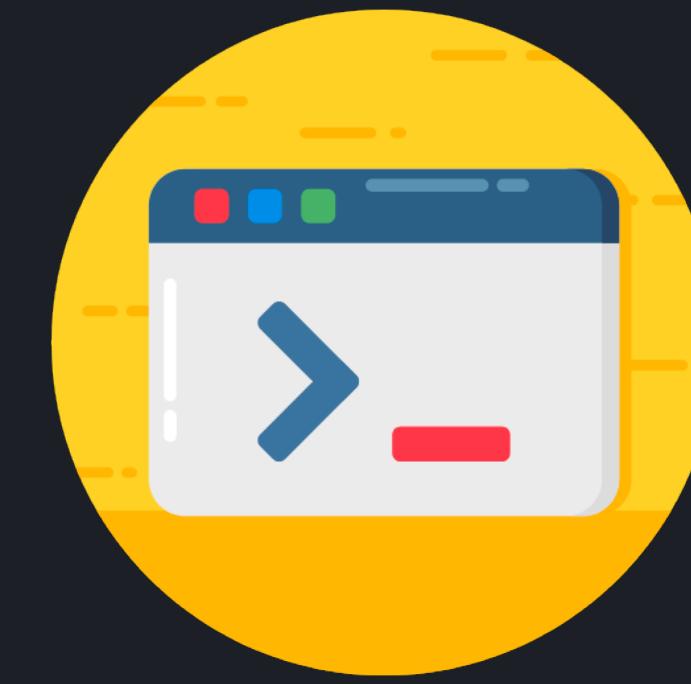
# Practice

## In the repo

- Rewrite the **DynamicHeading** component using the render function
- Refactor the `#app` styles to a CSS class and use CSS modules

## In your app

- Look around to see if there are any components that might benefit from using the render method instead of templates
- Refactor a component to use CSS modules



# VUE CLI

TOOL  
Vue CLI

TOOL

# Vue CLI - GUI Mode

**PRO TIP**

**Vue CLI Modern Mode**

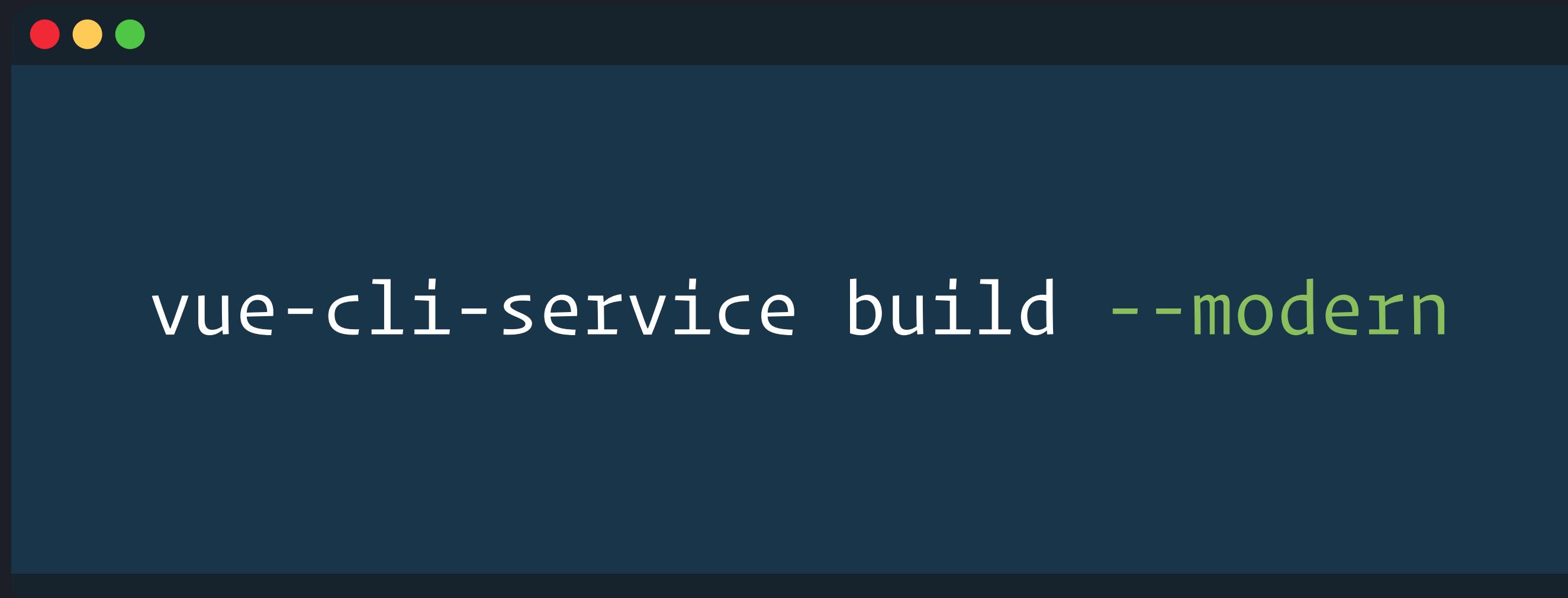
## PRO TIP

# Vue CLI - Modern Mode

- Babel allows us to utilize all the newest language features in ES6+, but this usually meant it gets shipped to all users regardless of their needs
- Vue CLI products two versions of your app:
  - A modern bundle targeting browsers that support ES modules
  - A legacy bundle targeting older browsers that do not

**PRO TIP**

# Vue CLI - Modern Mode



- Once it is enabled, no additional steps are needed!

# DISCUSSION

## Valid alternatives to Vue CLI

## DISCUSSION

# Valid alternatives to Vue CLI

- Micro-frontends
- Legacy migration
- Server-side rendering





# **SHORT BREAK**

**Be back at 10:45AM!**



# COMPONENTS (PT. 1)

# **BEST PRACTICE**

## Naming Components



Actual  
programming



Debating for  
30 minutes on  
how to name a  
variable

## BEST PRACTICE

### Naming Components

Avoid single word components

~~Header.vue~~

~~Button.vue~~

~~Container.vue~~

## BEST PRACTICE

### Naming Components

**App**PrefixedName.vue / **Base**PrefixedName.vue

Reusable, globally registered UI components.

AppButton, AppModal, BaseDropdown, BaseInput

**The**PrefixedName.vue

Single-instance components where only 1 can be active at the same time.

TheShoppingCart, TheSidebar, TheNavbar

## BEST PRACTICE

### Naming Components

Tightly coupled/related components

**TodoList.vue**

**TodoListItem.vue**

**TodoListItemName.vue**

1. Easy to spot relation
2. Stay next to each other  
in the file tree
3. Name starts with the  
highest-level words

**BEST PRACTICE**

Naming Component Methods

## BEST PRACTICE

### Naming Component Methods

Use descriptive names

✗ `onInput`

✓ `updateUserName`

Don't assume where it will be called

```
updateUserName ($event) {  
  this.user.name = $event.target.value  
}
```

✗ `Wrong`

```
updateUserName (newName) {  
  this.user.name = newName  
}
```

✓ `Correct`

## BEST PRACTICE

### Naming Component. Methods

Prefer destructuring over multiple arguments

```
updateUser (userList, index, value, isOnline) {  
  if (isOnline) {  
    userList[index] = value  
  } else {  
    this.removeUser(userList, index)  
  }  
}
```

# BEST PRACTICE

## Naming Component. Methods

Prefer destructuring over multiple arguments

```
updateUser (userList, index, value, isOnline) {  
  if (isOnline) {  
    userList[index] = value  
  } else {  
    this.removeUser(userList, index)  
  }  
}
```

✗ Not recommended

```
updateUser ({ userList, index, value, isOnline }) {  
  if (isOnline) {  
    userList[index] = value  
  } else {  
    this.removeUser(userList, index)  
  }  
}
```

✓ Recommended

## **BEST PRACTICE**

# When to Refactor Your Components

*Premature optimization is the root of all evil (or at least most of it) in programming.*

- Donald Knuth

**PRINCIPLE**

# Data Driven Refactoring

# PRINCIPLE

## Data Driven Refactoring

### Signs you need more components

- When your components are hard to understand
- You feel a fragment of a component could use its own state
- Hard to describe what what the component is actually responsible for

## PRINCIPLE

### Data Driven Refactoring

How to find reusable components?

- Look for v-for loops
- Look for large components
- Look for similar visual designs
- Look for repeating interface fragments
- Look for multiple/mixed responsibilities
- Look for complicated data paths



# Questions?

**PRO TIP**

**SFC Code Block Order**



## MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```



## MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```



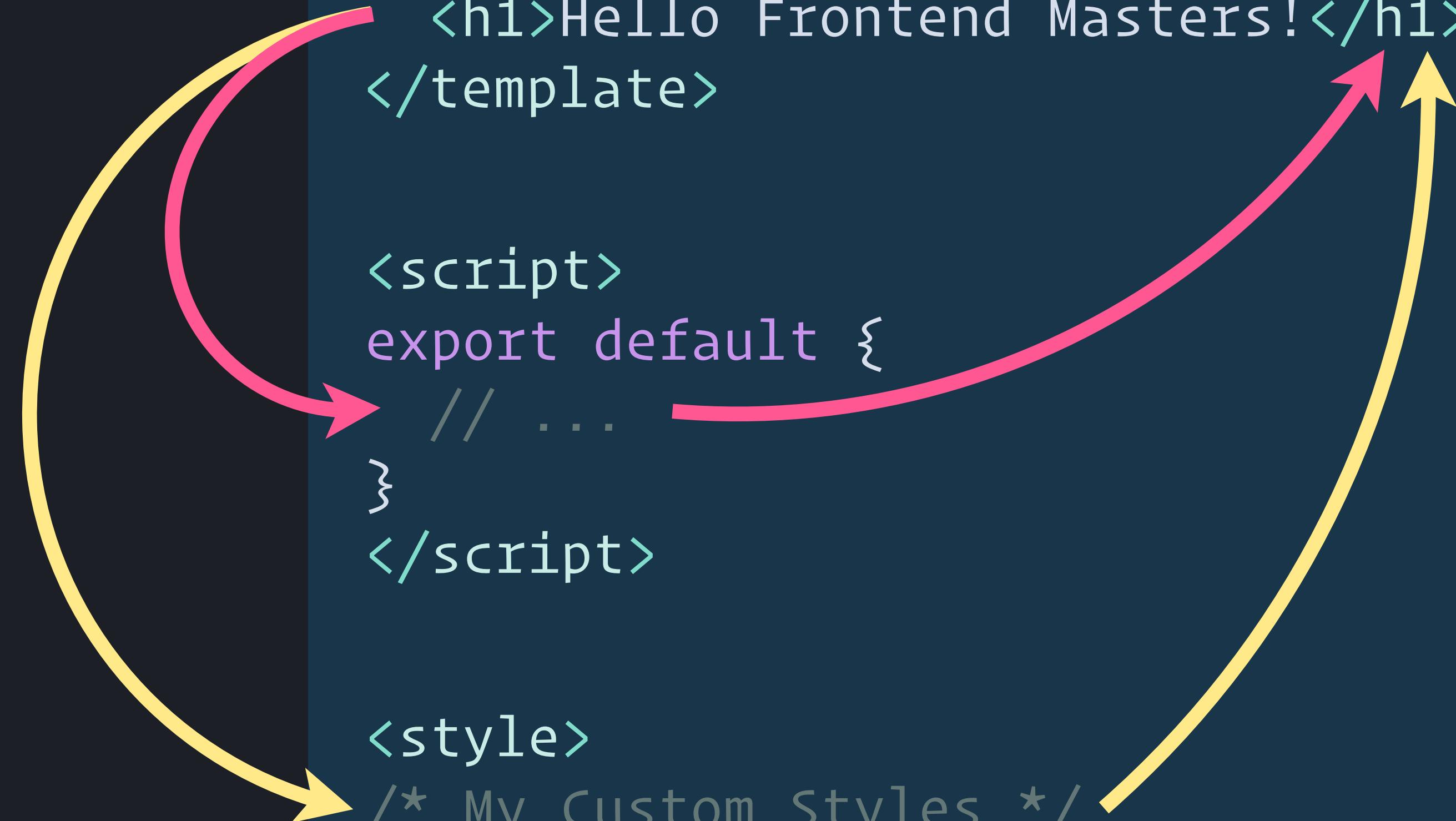


## MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```





## MyFirstComponent.vue

```
<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<script>
export default {
  // ...
}
</script>

<style>
/* My Custom Styles */
</style>
```



## MyFirstComponent.vue

```
<script>
export default {
  // ...
}
</script>

<template>
  <h1>Hello Frontend Masters!</h1>
</template>

<style>
/* My Custom Styles */
</style>
```



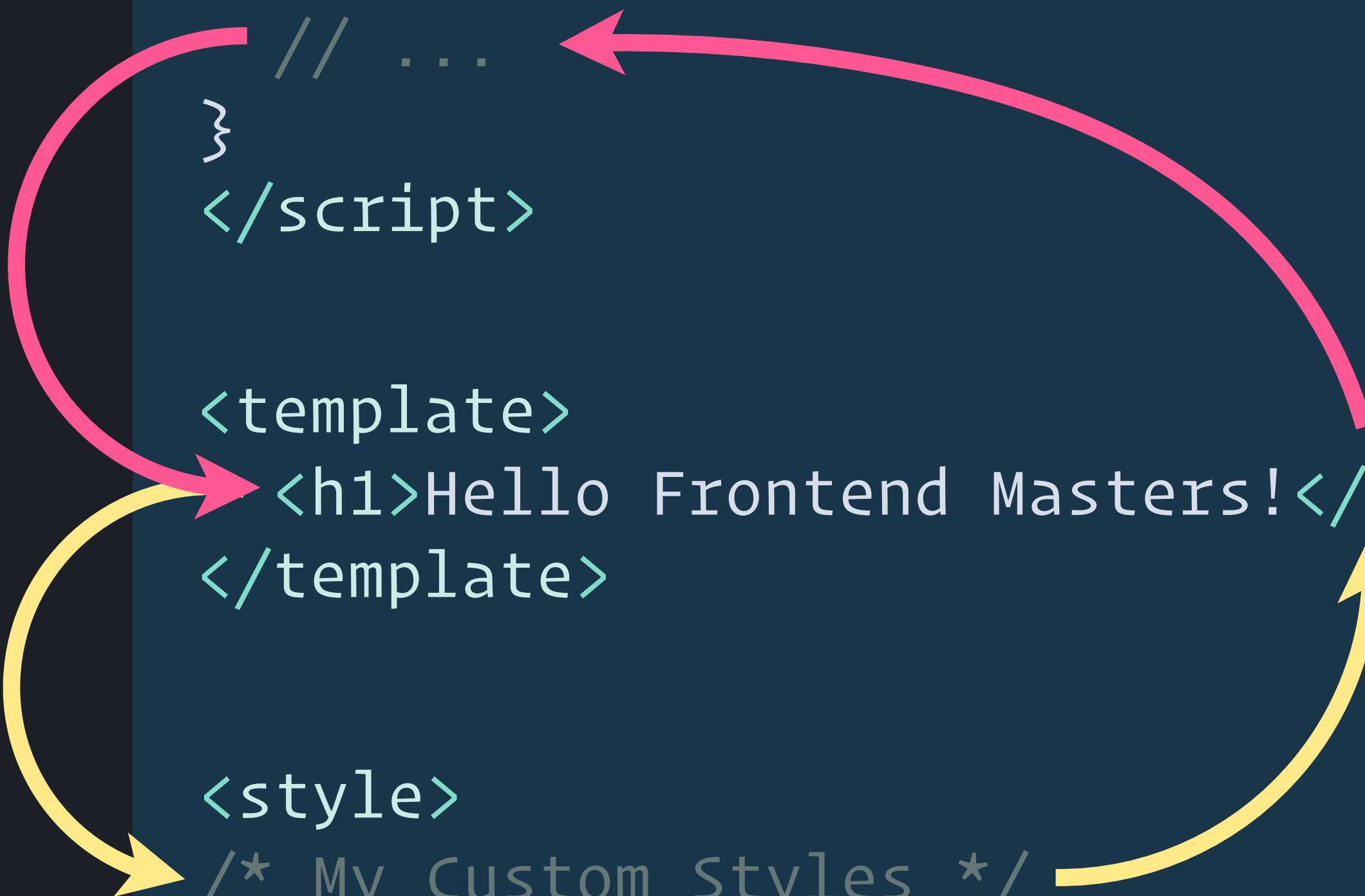
## MyFirstComponent.vue

```
<script>
export default {
  // ...
}

</script>

<template>
<h1>Hello Frontend Masters!</h1>
</template>

<style>
/* My Custom Styles */
</style>
```



**PRO TIP**

**Component File Organization**

# Nested Structure

```
▲ src
  ▲ components
    ▲ Dashboard
      ▶ tests
      ▼ Dashboard.vue
      ▼ Header.vue
    ▲ Login
      ▶ tests
      ▼ Header.vue
      ▼ Login.vue
      ▶ tests
      ▼ Header.vue
```

# Flat Structure

```
▲ src
  ▲ components
    ⚡ Dashboard.unit.js
    ▼ Dashboard.vue
    ⚡ DashboardHeader.unit.js
    ▼ DashboardHeader.vue
    ⚡ Header.unit.js
    ▼ Header.vue
    ⚡ Login.unit.js
    ▼ Login.vue
    ⚡ LoginHeader.unit.js
    ▼ LoginHeader.vue
```

```
▲ src
  ▲ components
    ▲ Dashboard
      ▶ tests
      ▼ Dashboard.vue
    ▼ Header.vue
  ▲ Login
    ▶ tests
    ▼ Header.vue
    ▼ Login.vue
    ▶ tests
  ▼ Header.vue
```

VS

```
▲ src
  ▲ components
    ⚡ Dashboard.unit.js
    ▼ Dashboard.vue
    ⚡ DashboardHeader.unit.js
    ▼ DashboardHeader.vue
    ⚡ Header.unit.js
    ▼ Header.vue
    ⚡ Login.unit.js
    ▼ Login.vue
    ⚡ LoginHeader.unit.js
    ▼ LoginHeader.vue
```

## PRO TIP

# Component File Organization

### **Flat makes refactoring easier**

No need to update imports if components move

### **Flat makes finding files easier**

Folders often leads to lazily named files

because they don't have to be unique

**PRO TIP**

Register base components globally

## PRO TIP

# Register base components globally

Instead of every component having:

```
import BaseButton from './components/BaseButton.vue'  
import BaseIcon from './components/BaseIcon.vue'  
import BaseInput from './components/BaseInput.vue'
```

Use this epic script by Chris Fritz:

<https://vuejs.org/v2/guide/components-registration.html#Automatic-Global-Registration-of-Base-Components>



# Questions?



# LUNCH BREAK

Be back at 1:00PM!



# COMPONENTS (PT. 2)

# TECHNIQUE

## Props

## NavItem.vue

```
<script>
export default {
  props: ['label']
}
</script>

<template>
  <li class="nav-item">
    <a :href="`/${label}`">
      {{ label }}
    </a>
  </li>
</template>
```

## NavItem.vue

```
<script>
export default {
  props: ['label']
}
</script>
<template>
  <li class="nav-item">
    <a :href="`/${label}`">
      {{ label }}
    </a>
  </li>
</template>
```

## Navbar.vue

```
<template>
<ul>
  <NavItem label="Home" />
  <NavItem label="About" />
  <NavItem label="Contact" />
</ul>
</template>
```

## NavItem.vue

```
<script>
export default {
  props: ['label'] 
}
</script>

<template>
  <li class="nav-item">
    <a :href="`/${label}`">
      {{ label }}
    </a>
  </li>
</template>
```

## Navbar.vue

```
<template>
  <ul>
    <NavItem label="Home" />
    <NavItem label="About" />
    <NavItem label="Contact" />
  </ul>
</template>
```

## NavItem.vue

```
<script>
export default {
  props: ['label']
}
</script>
```

## NavItem.vue

```
<script>
export default {
  props: {
    label: {
      type: String,
      required: true,
      default: 'Home'
    }
  }
}
</script>
```

## NavItem.vue

```
<script>
export default {
  props: {
    label: {
      type: String,
      default: 'Home'
    }
  }
}
</script>
```

## NavItem.vue

```
<script>
export default {
  props: {
    label: {
      type: String,
      default: 'Home',
      validator: (value) => {
        return ['Home', 'About'].indexOf(value) !== -1
      }
    }
  }
}
</script>
```

Let's do a  
Coding Experiment

# Task 1

*Create a button component that can display text specified in the parent component*

Submit

# Task 2

*Allow the button to display an icon of choice  
on the right side of the text*

Submit →

```
<AppIcon icon="arrow-right" class="ml-3"/>
```

*This is the code responsible for displaying an arrow.*

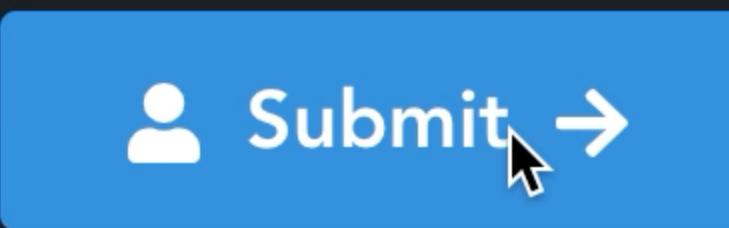
# Task 3

*Make it possible to have icons on either side or even both sides*

 Submit →

# Task 4

*Make it possible to replace the content with a loading spinner*

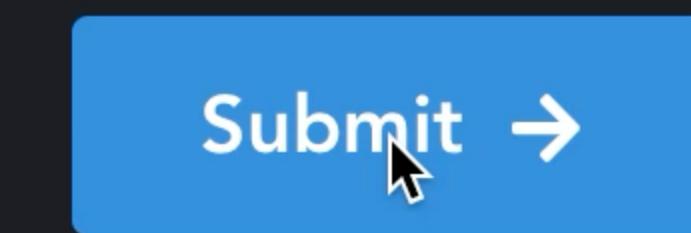


`<PulseLoader color="#fff" size="12px"/>`

*This is the code responsible for displaying a spinner.*

# Task 5

*Make it possible to replace an icon with a loading spinner*



# Possible solution

```
<template>
  <button type="button" class="nice-button">
    {{ text }}
  </button>
</template>
```

```
<script>
export default {
  props: ['text']
}
</script>
```

```
<template>
  <button type="button" class="nice-button">
    <PulseLoader v-if="isLoading" color="#fff" size="12px">
    <template v-else>
      <template v-if="iconLeftName">
        <PulseLoader v-if="isLoadingLeft" color="#fff"
size="6px">
          <AppIcon v-else :icon="iconLeftName"/>
        </template>
      {{ text }}
      <template v-if="iconRightName">
        <PulseLoader v-if="isLoadingRight" color="#fff"
size="6px">
          <AppIcon v-else :icon="iconRightName"/>
        </template>
      </template>
    </button>
</template>
```

```
<script>
export default {
  props: ['text', 'iconLeftName', 'iconRightName', 'isLoading',
  'isLoadingLeft', 'isLoadingRight']
}
</script>
```



OMG  
PROPS EVERYWHERE!

```
<template>
<button>
  <Pul...
<temp...
<t...
size="6p...
<...
{{
<t...
size="6p...
<...
</te...
</but...
</templa...
<script>
export d...
  props:...
  'isLoadi...
}
</script>
```

```
<template>
  <button type="button" class="nice-button">
    <PulseLoader v-if="isLoading" color="#fff" size="12px">
    <template v-else>
      <template v-if="iconLeftName">
        <PulseLoader v-if="isLoadingLeft" color="#fff"
size="6px">
          <AppIcon v-else :icon="iconLeftName"/>
        </template>
        {{ text }}
      <template v-if="iconRightName">
        <PulseLoader v-if="isLoadingRight" color="#fff"
size="6px">
          <AppIcon v-else :icon="iconRightName"/>
        </template>
      </template>
    </button>
</template>
```

**Let's call it the `props-based` solution**

```
<script>
export default {
  props: ['text', 'iconLeftName', 'iconRightName', 'isLoading',
  'isLoadingLeft', 'isLoadingRight']
}
</script>
```

props-based solution

Is it wrong?

props-based solution

Is it wrong?

No.

It does the job.

# props-based solution

Is it good, then?

props-based solution

Is it good, then?

Not exactly.

props-based solution

Problems

props-based solution

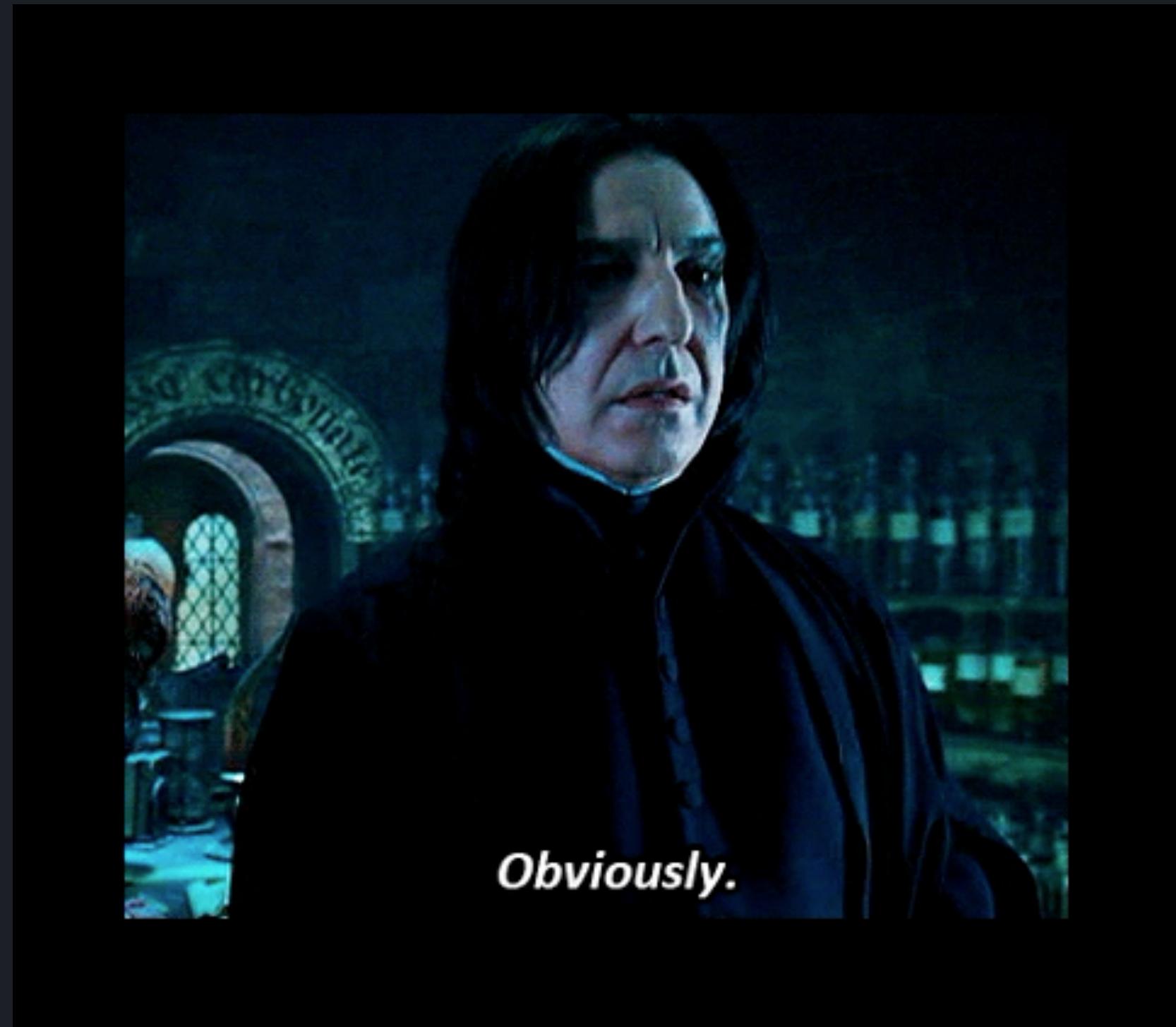
## Problems

- New requirements increase complexity
- Multiple responsibilities
- Lots of conditionals in the template
- Low flexibility
- Hard to maintain

Is it good, then?

Not exactly.

Is there a better another alternative?



Is there a better another alternative?

# Recommended solution

```
<template>
  <button type="button" class="nice-button">
    <slot />
  </button>
</template>
```

# PROBLEM

How to dynamically switch components  
based on data?

# TECHNIQUE

<Component :is="“name”">

```
<template>
  <div>
    <Component :is="clockType" v-bind="clockProps"/>
  </div>
</template>
```

```
<script>
export default {
  components: { DigitalClock },
  computed: {
    clockType () {
      if (this.selectedClock === 'analog') {
        this.clockProps = {
          ...analogProps
        }
        return () => import(`./components/${this.compName}`)
      } else {
        return 'DigitalClock'
      }
    }
  }
  // ...
}
</script>
```

## **<Component :is>**

Becomes the component specified by the **:is** prop.

# Pros

- Extremely powerful and flexible
- Easy to use
- Can accept props
- Can accept asynchronous components
- Can change into different components
- You can make a router-view out of it

# Cons

- Got to handle props carefully

```
<template>
  <div>
    <Component :is="clockType" v-bind="clockProps"/>
  </div>
</template>
```

```
<script>
export default {
  components: { DigitalClock },
  computed: {
    clockType () {
      if (this.selectedClock === 'analog') {
        this.clockProps = {
          ...analogProps
        }
        return () => import(`./components/${this.compName}`)
      } else {
        return 'DigitalClock'
      }
    }
  }
  // ...
}
</script>
```

# DESIGN PATTERN

## Vendor Components Wrapper

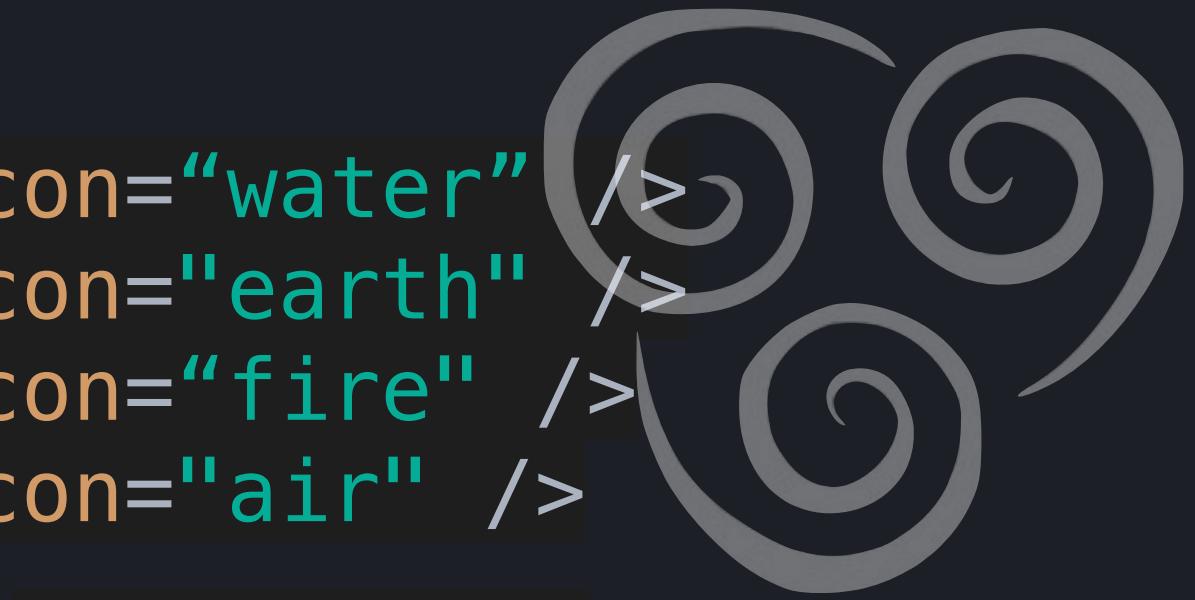
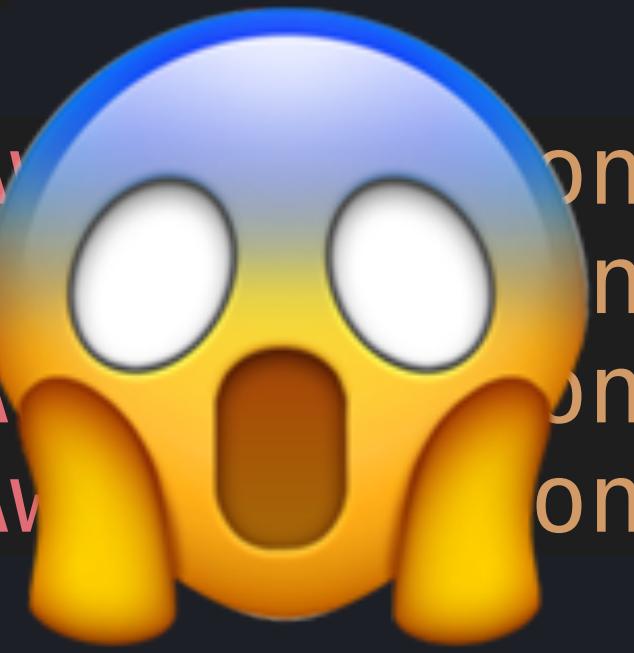
```
<template>
  <p>
    <FontAwesome icon="water" />
    <FontAwesome icon="earth" />
    <FontAwesome icon="fire" />
    <FontAwesome icon="air" />
  </p>
</template>
```



```
<template>
  <p>
    <FontAwesome icon="water" />
    <FontAwesome icon="earth" />
    <FontAwesome icon="fire" />
    <FontAwesome icon="air" />
  </p>
</template>
```



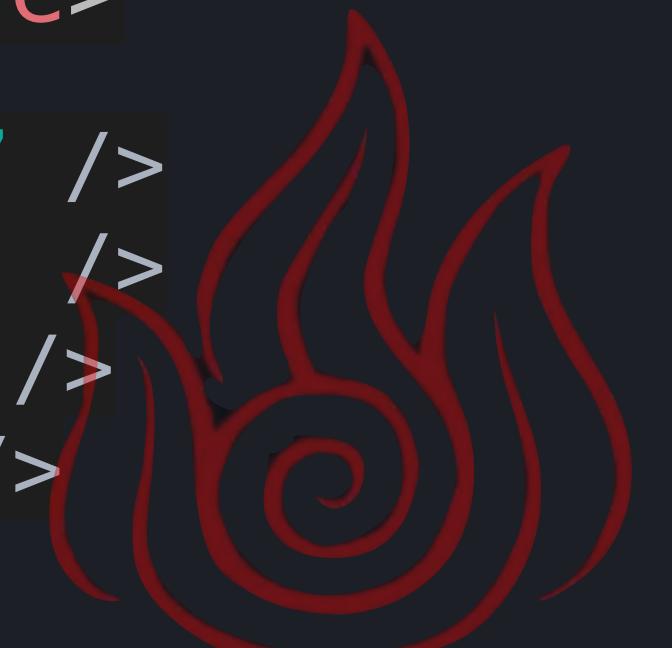
```
<template>
  <p>
    <FontAwesome icon="water" />
    <FontAwesome icon="earth" />
    <FontAwesome icon="fire" />
    <FontAwesome icon="air" />
  </p>
</template>
```



```
<template>
  <p>
    <FontAwesome icon="water" />
    <FontAwesome icon="earth" />
    <FontAwesome icon="fire" />
    <FontAwesome icon="air" />
  </p>
</template>
```



```
<template>
  <p>
    <FontAwesome icon="water" />
    <FontAwesome icon="earth" />
    <FontAwesome icon="fire" />
    <FontAwesome icon="air" />
  </p>
</template>
```



# BaseIcon .vue

```
<template>
  <FontAwesomeIcon
    v-if="source === 'font-awesome'"
    :icon="name"
  />
  <span
    v-else
    :class="customIconClass"
  />
</template>
```

```
<template>
  <p>
    <BaseIcon icon="earth" />
    <BaseIcon icon="fire" />
    <BaseIcon icon="water" />
    <BaseIcon icon="air" />
  </p>
</template>
```

# DESIGN PATTERN

## Transparent Components

# When passing props, listeners, and attributes...

```
// BaseInput.vue
<template>
  <div>
    <input
      type="text"
      v-bind="{ ...$attrs, ...$props }"
      v-on="$listeners"
    />
  </div>
</template>
```

# When passing props, listeners, and attributes...

```
// BaseInput.vue
<template>
  <div>
    <input
      type="text"
      />
  </div>
</template>

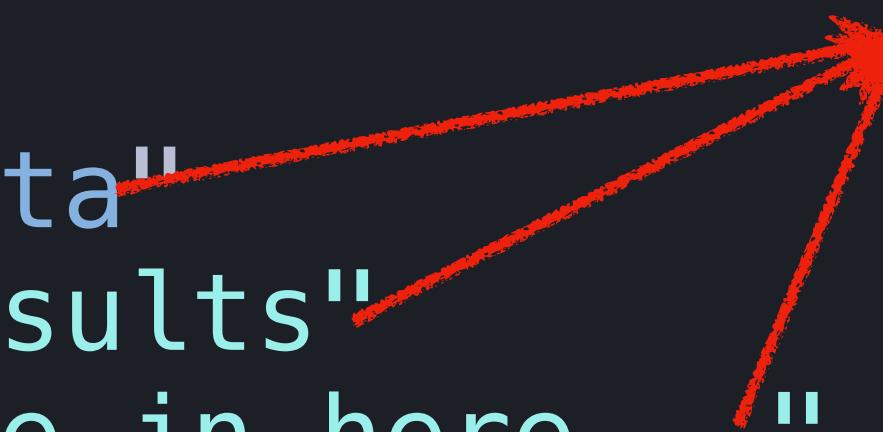
<BaseInput
  @input="filterData"
  label="Filter results"
  placeholder="Type in here...">
</BaseInput>
```



# When passing props, listeners, and attributes...

```
// BaseInput.vue
<template>
<div>
  <input
    type="text"
    />
</div>
</template>

<BaseInput
  @input="filterData"
  label="Filter results"
  placeholder="Type in here...">
</BaseInput>
```



# When passing props, listeners, and attributes...

```
<template>
  <div>
    <input
      type="text"
      v-bind="{ ...$attrs, ...$props }"
      v-on="$listeners"
    />
  </div>
</template>
<script>
export default {
  inheritAttrs: false,
  // ...
}
</script>
```

Both props and attributes, as well as all listeners will be passed to this element instead.

Prevent Vue from assigning attributes to top-level element

# When passing props, listeners, and attributes...

```
<template>
  <div>
    <input
      type="text"
      v-bind="{ ...$attrs, ...$props }"
      v-on="$listeners"
    />
  </div>
</template>
<script>
export default {
  inheritAttrs: false,
  // ...
}
</script>
```

# When passing props, listeners, and attributes in v3...

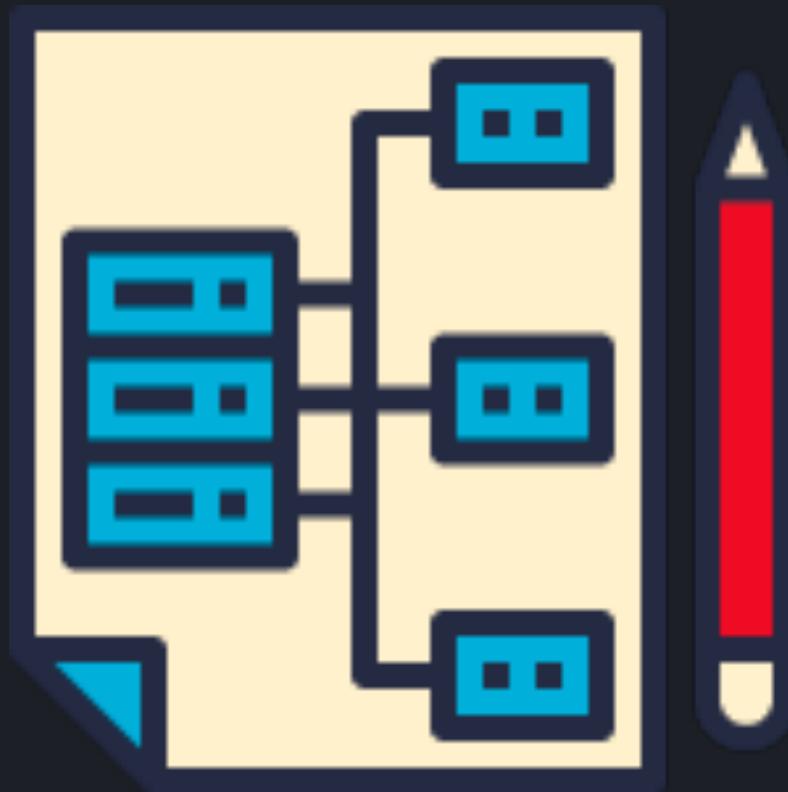
```
<template>
  <div>
    <input type="text" v-bind="$attrs" />
  </div>
</template>

<script>
export default {
  inheritAttrs: false,
  // ...
}
</script>
```





# Questions?



# REUSABILITY & COMPOSITION

# PROBLEM

How to share the functionality across  
different components?

# TECHNIQUE

## Mixins

<https://vuejs.org/v2/guide/mixins.html>

# A mixin

```
const myMixin = {  
  data () {  
    return {  
      foo: 'bar'  
    }  
  }  
}  
  
export default {  
  mixins: [myMixin],  
  // component code  
}
```

# Mixin as a function

```
const myMixin = (count) => ({  
  data () {  
    return {  
      currentCount: count  
    }  
  }  
})
```

```
export default {  
  mixins: [myMixin(10)],  
  // component code  
}
```

# Cons

- Possible properties name clashes.
- Can't share template fragments
- Gets harder to track where things are coming from once there are more mixins

Should you never use mixins?

No.

# Pros

- Relatively easy to use
- Good for refactoring

# TECHNIQUE

## Provide/Inject

<https://vuejs.org/v2/api/#provide-inject>

# Provide/Inject

```
export default {
  provide () {
    return {
      width: this.width,
      key: 'name',
      fetchMore: this.fetchMore
    }
  },
  data() {
    return {
      width: null,
    }
  },
  methods: {
    fetchMore () {
      // ...
    }
  }
}
```

# Provide/Inject

```
export default {  
  inject: ['width', 'key', 'fetchMore'],  
  props: {  
    optionKey: {  
      type: String,  
      default () {  
        return this.key  
      }  
    }  
  }  
}
```



Injected values can be used as  
default props and data values

# Pros

- Easy sharing data and methods with descendants
- Helps avoiding unnecessary props
- Components can choose which properties to inject
- Can be used to provide default props and data values

# Cons

- There are some reactivity caveats when it comes to usage in Vue 2
- Creates a tight relationship between two components that is not immediately apparent
- There is ambiguity when it comes to what is coming from where



Provide and inject are primarily useful for advanced plugin / component library use cases. It is NOT recommended to use them in generic application code.



# **NEW FEATURE**

# Composition API

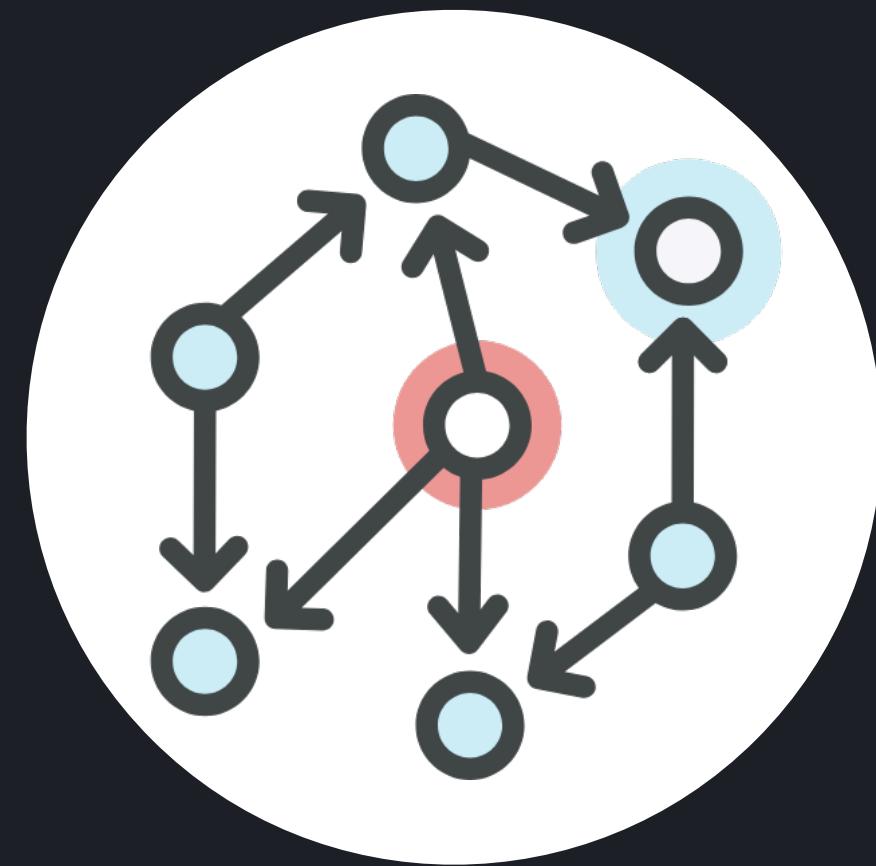
**Vue 3.0 is here!**

**Also available in Vue 2!**

<https://github.com/vuejs/composition-api>



# Questions?



# STATE MANAGEMENT

# **BEST PRACTICES**

## Vuex

# BEST PRACTICES

## Vuex

What data to put into Vuex?

- Data shared between components that might not be in direct parent-child relation
- Data that you want to keep between router views (for example lists of records fetched from the API)
- Route params are more important though (as a source of truth)

# BEST PRACTICES

## Vuex

What data to put into Vuex?

- Any kind of global state
- Examples: login status, user information, global notifications
- Anything if you feel it will make managing it simpler

# BEST PRACTICES

## Vuex

What data to **avoid** putting into Vuex?

- User Interface variables
  - Examples: `isDropdownOpen`, `isInputFocused`, `isModalVisible`
- Forms data
- Validation results
- Single records from the API

# BEST PRACTICES

## Vuex

Do I need to **always** use a **getter** to return a simple fragment of state?

Feel free to access state directly  
`this.$store.state.usersList`

**No.**

Use computed properties to return computed state

```
activeUsersList () {  
  return this.$store.state.usersList.filter(  
    user => user.isActive  
  )  
}
```

## BEST PRACTICES

### Vuex

If you need to share derived Vuex state between components, make it a getter.

*You should weigh the trade-offs and make decisions that fit the development needs of your app.*

## PRO TIP

Avoid calling mutations  
directly in components

## PRO TIP

Use built-in `map` helpers  
(except mutations)

# PRO TIP

Use built-in **map** helpers  
(except mutations)

```
computed: {  
  ...mapState({  
    userName: state => state.user.name  
}),  
  ...mapGetters([  
    'activeUsersList'  
]),  
},  
methods: {  
  ...actions([  
    'updateUserName'  
])  
}
```

## **BEST PRACTICES**

Always use namespace modules

# DISCUSSION

## With the Composition API, do we even need Vuex?



# ROUTING

# **BEST PRACTICES**

## Routing

# BEST PRACTICES

## Routing

- There are three main categories of routing:
- **View Components** - Definition for page level components (i.e., Home, About, Dashboard)
- **Layout Components** - Markup shared between pages (i.e., header, sidebar, etc.)
- **Routes** - Define how paths map to view components

## TECHNIQUE

Add a dynamic key  
to all router-view components

# TECHNIQUE

Add a dynamic key  
to all router-view components

```
<router-view :key="$route fullPath" />
```

Ensures that the page re-renders every time.

TOOL  
vue-meta

<https://github.com/nuxt/vue-meta>

# TOOL

## vue-meta

```
export default {
  metaInfo() {
    return {
      title: this.user.name,
      meta: [
        {
          name: 'description',
          content: `The user profile for ${this.user.name}.`,
        },
      ],
    },
  },
  // ...
}
```

# TECHNIQUE

## Lazy load routes

# TECHNIQUE

## Lazy load routes

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home
  },
  {
    path: "/about",
    name: "About",
    component: () =>
      import(/* webpackChunkName: "about" */ "../views/About.vue")
  }
];
```



# TESTING

**PRINCIPLE**

The Pareto Principle

**PRINCIPLE**

**The Pareto Principle**

**The 80-20 Rule**

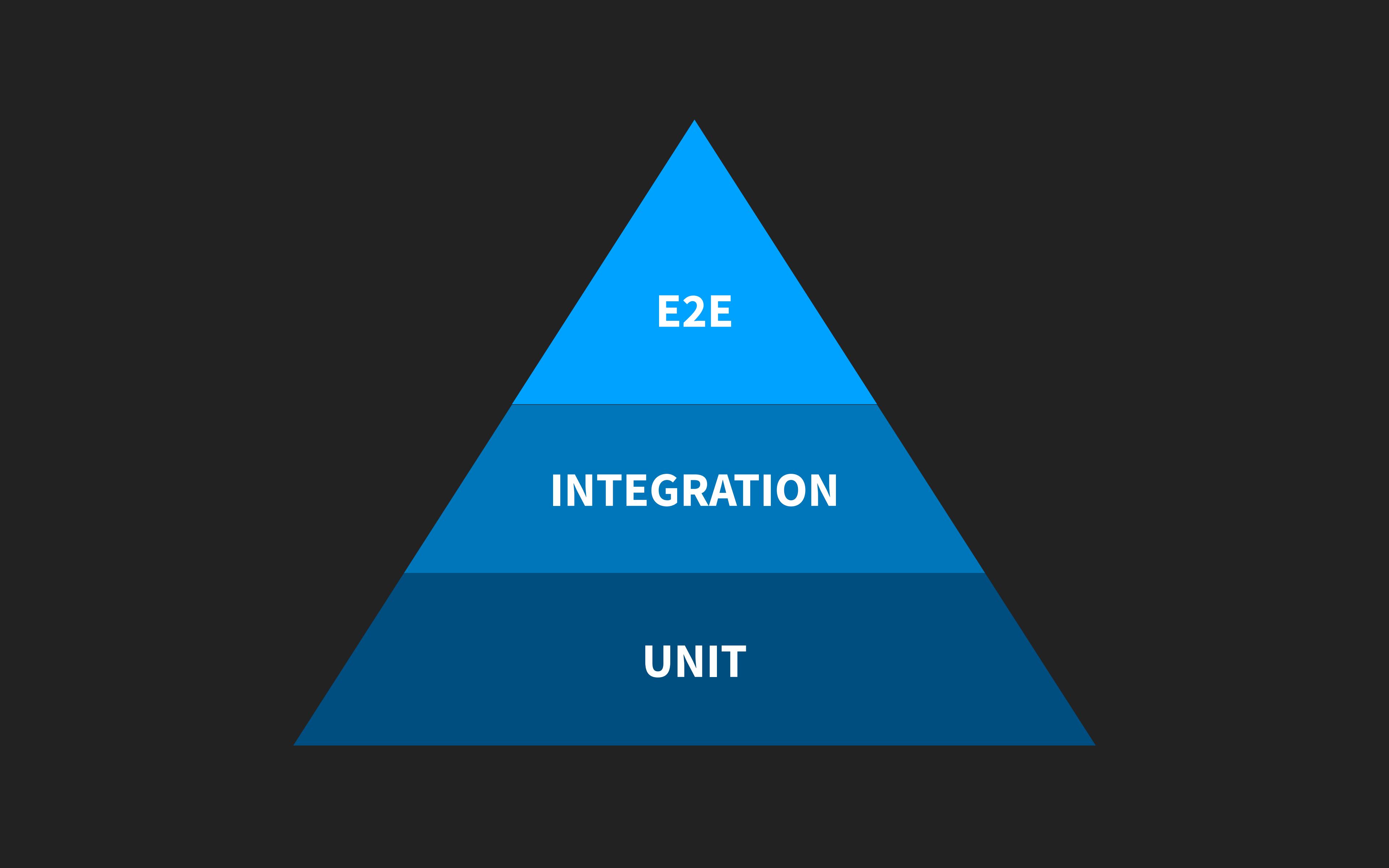


20% EFFORT



# **BEST PRACTICES**

## Testing



**E2E**

**INTEGRATION**

**UNIT**



**UNIT**

# UNIT



Jest



UNIT

What about Vue Test Utils?

# COMPONENT



Testing  
Library

<https://testing-library.com/>

# COMPONENT



Testing  
Library



Cypress

## BEST PRACTICE

# Writing unit tests

- **Don't test that Vue works**
  - Examples: Checking that a data, computed, etc property exists
- **Primarily stick with shallow rendering**
  - Otherwise a problem in a common component can break many tests
- **Build unit tests into generators**

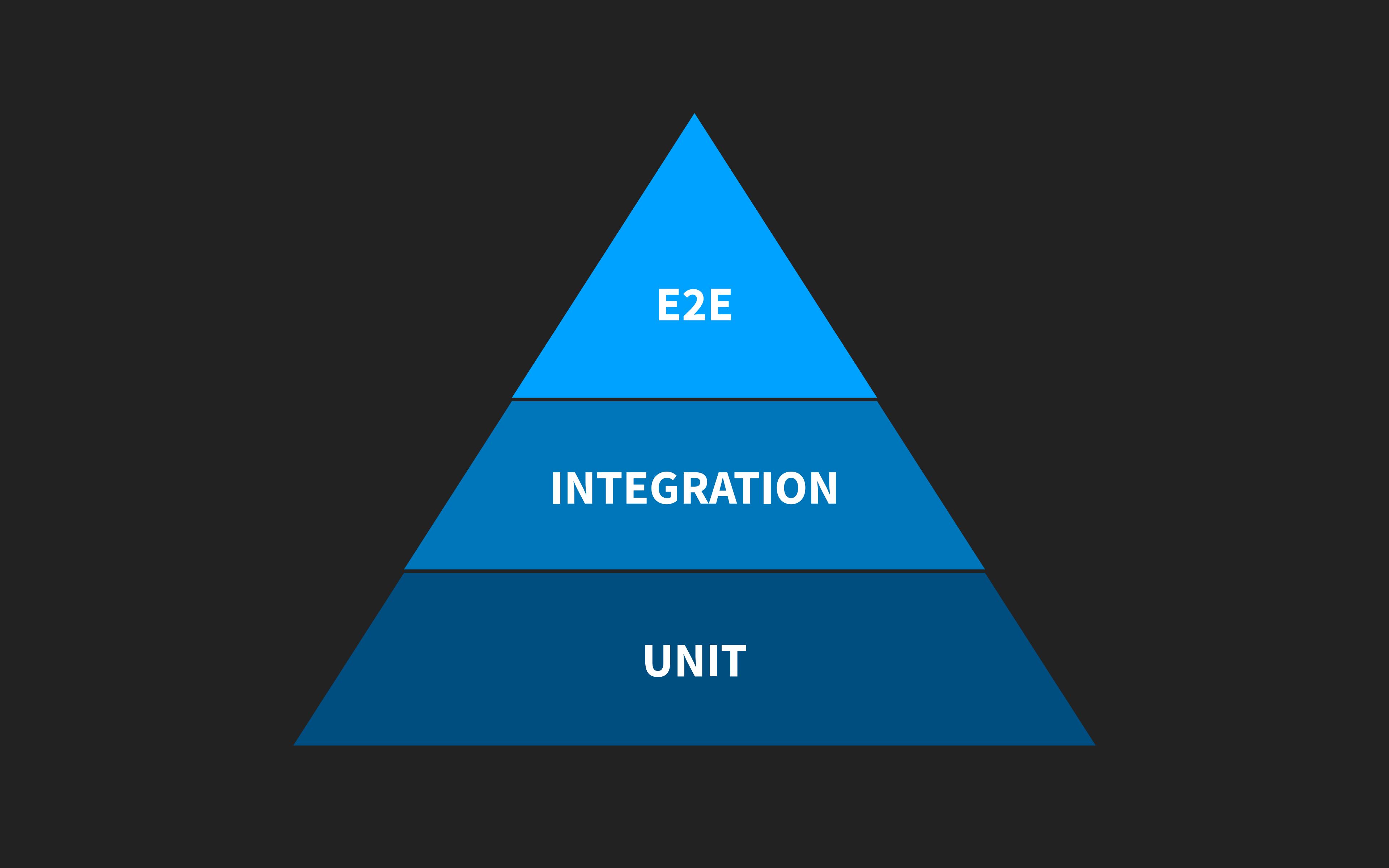
Unit tests are great, but they're  
often not the 20% we should  
often be focusing on

If you can only have two tests  
for your application...

If you can only have two tests  
for your application...

#1. Can the user login?

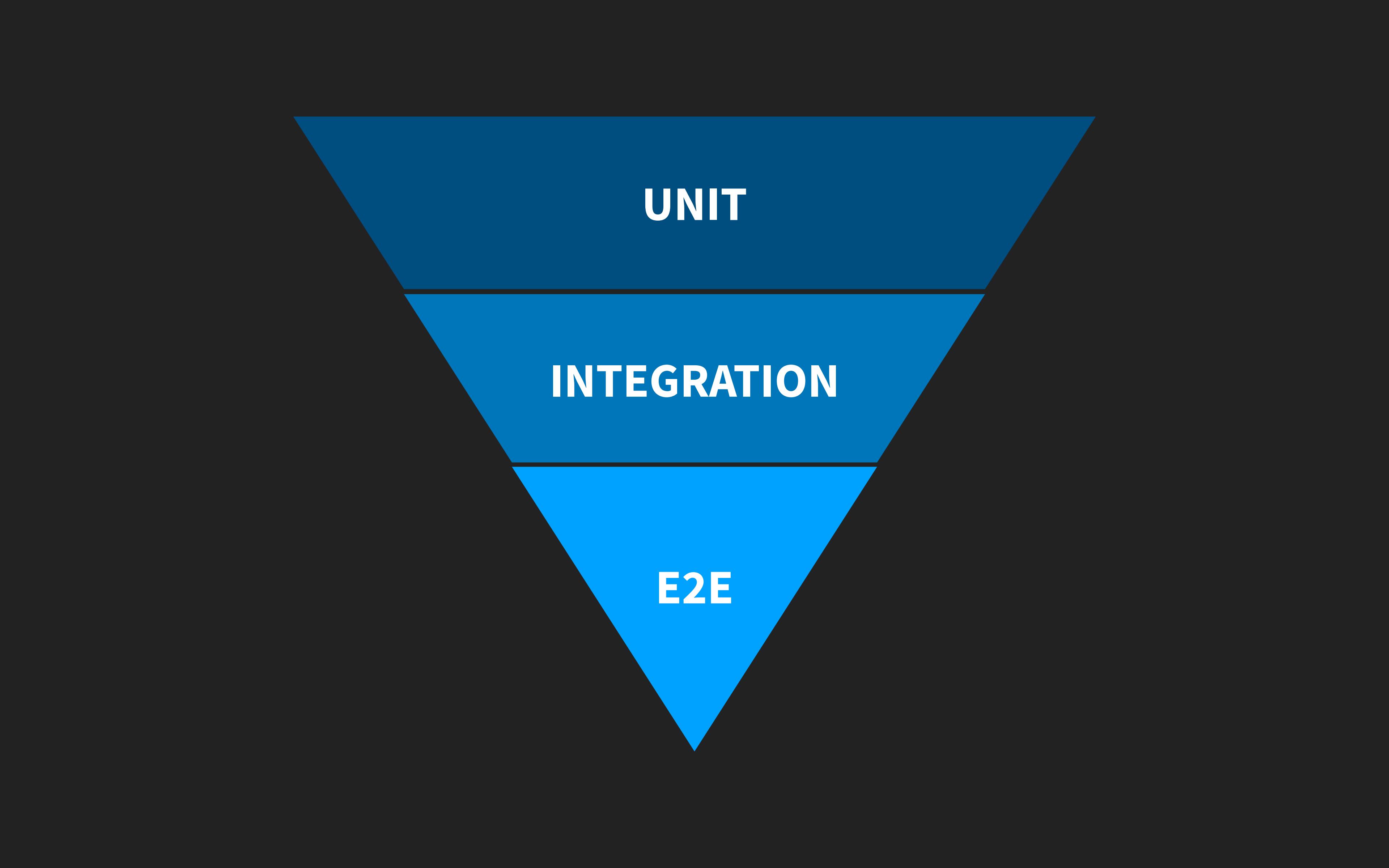
#2. Can the user pay us?



**E2E**

**INTEGRATION**

**UNIT**



UNIT

INTEGRATION

E2E



E2E

Most teams avoid E2E tests because...

- Take a long time to run
- "Flakey" (i.e., unreliable)

As a result, there are often no E2E tests at all...

E2E



E2E

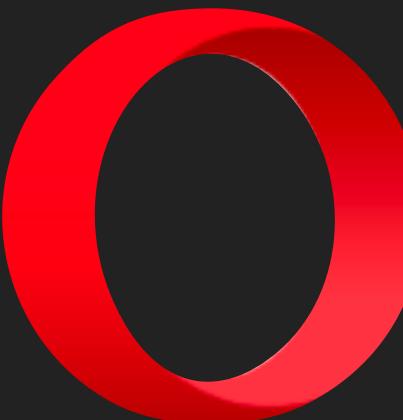


# E2E



## Project Coverage summary

Name	Classes	Conditionals	Files	Lines	Packages
Cobertura Coverage Report	45% 23/51	74% 469/630	45% 23/51	28% 1450/5222	88% 7/8
Coverage Breakdown by Package					
Name	Classes	Conditionals	Files	Lines	Packages
<a href="#">Stop-tabac</a>	0% 0/1	47% 8/17	0% 0/1	0% 0/5	
<a href="#">Stop-tabac.Classes</a>	100% 1/1	73% 22/30	19% 1/1	27% 58/213	
<a href="#">Stop-tabac.Classes.Controller</a>	19% 3/16	63% 20/32	100% 3/16	5% 120/2665	
<a href="#">Stop-tabac.Classes.Manager</a>	100% 3/3	81% 249/306	100% 3/3	65% 103/158	
<a href="#">Stop-tabac.Classes.Model</a>	75% 6/8	69% 163/235	75% 6/8	65% 669/869	
<a href="#">Stop-tabac.Classes.Service</a>	56% 5/9	N/A	56% 5/9	77% 388/830	
<a href="#">Stop-tabac.Classes.Utils</a>	60% 3/5	70% 7/10	60% 3/5	47% 74/126	
<a href="#">Stop-tabac.Classes.View</a>	25% 2/8		25% 2/8	59% 38/356	



# E2E



## Project Coverage summary

Name  
Cobertura Coverage Report

Classes  
23/51

Conditionals

Files  
23/51

Lines  
1450/5222

Packages  
7/8

## Coverage Breakdown by Package

Name

[Stop-tabac](#)

Classes  
0/1

Conditionals  
1/1

Files  
1/1

Lines  
0/5

Packages  
0/5

Stop-tabac.Classes  
1/1

Conditionals  
3/16

Files  
3/16

Lines  
58/213

Packages  
58/213

Stop-tabac.Classes.Controller  
3/3

Conditionals  
3/3

Files  
120/2665

Lines  
120/2665

Packages  
120/2665

Stop-tabac.Classes.Manager  
6/8

Conditionals  
6/8

Files  
103/158

Lines  
103/158

Packages  
103/158

Stop-tabac.Classes.Model  
5/9

Conditionals  
5/9

Files  
669/869

Lines  
669/869

Packages  
669/869

Stop-tabac.Classes.Service  
3/5

Conditionals  
3/5

Files  
388/830

Lines  
388/830

Packages  
388/830

Stop-tabac.Classes.Utils  
2/8

Conditionals  
2/8

Files  
74/126

Lines  
74/126

Packages  
74/126

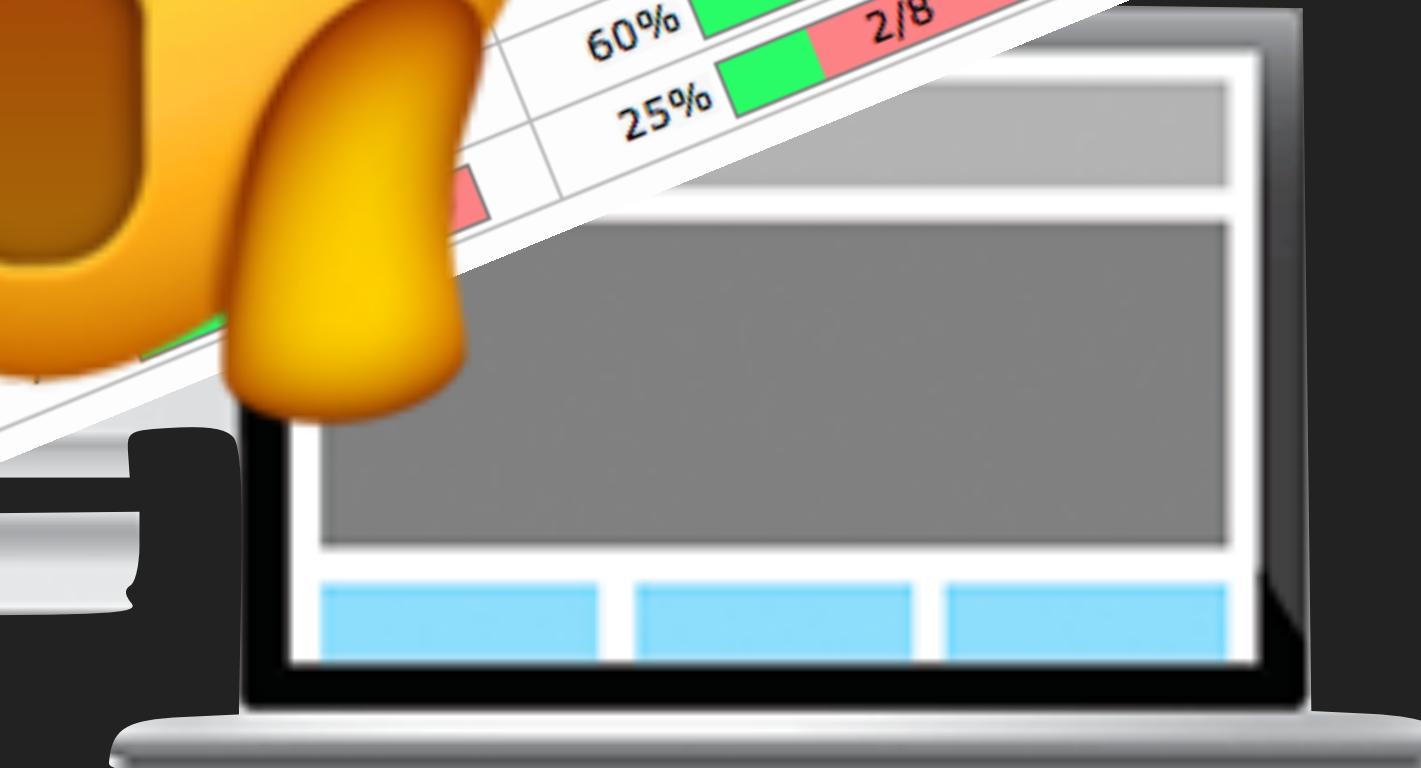
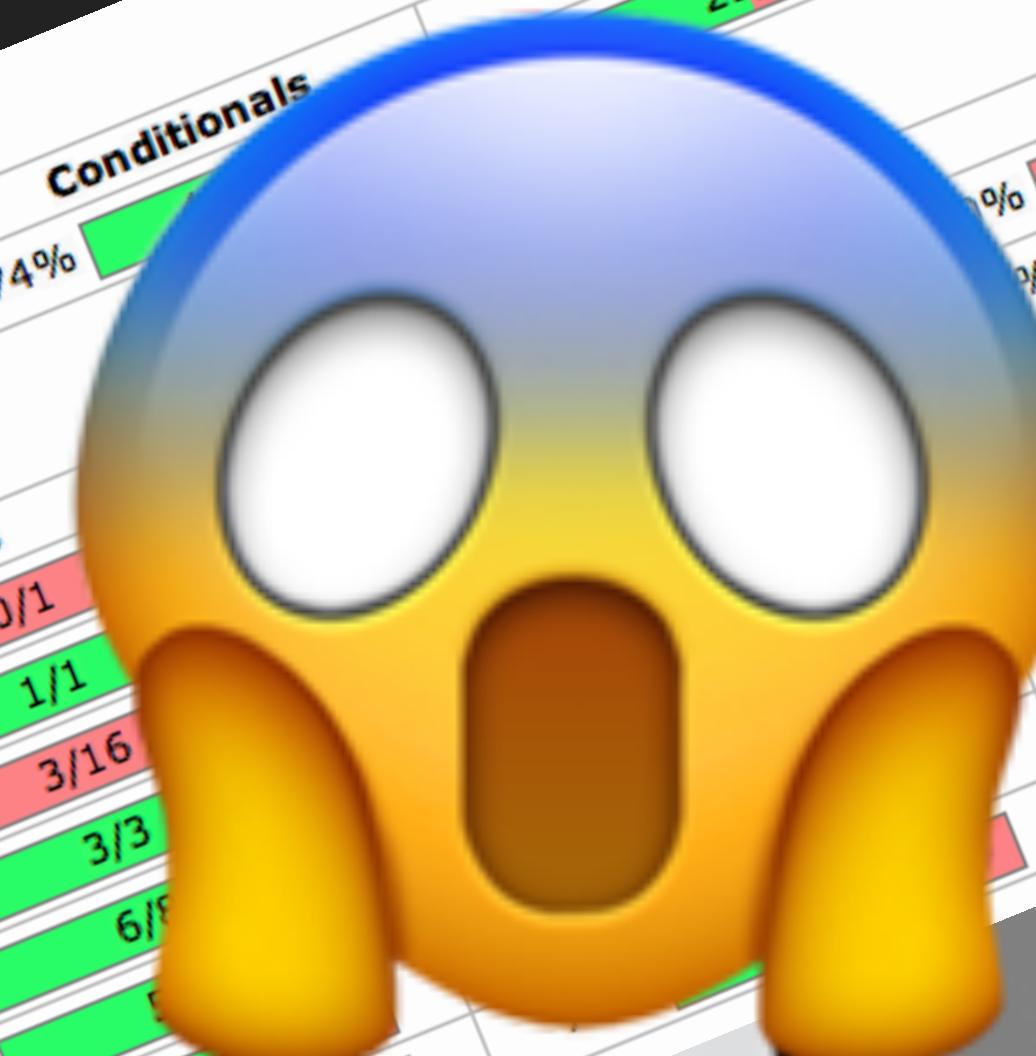
Stop-tabac.Classes.View  
25%

Conditionals  
25%

Files  
38/356

Lines  
38/356

Packages  
38/356



# E2E

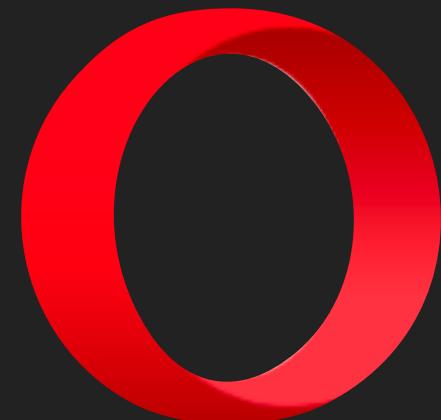


## Project Coverage summary

Name	Classes	Conditionals	Files	Lines	Packages
Cobertura Coverage Report	45% 23/51	74% 469/630	45% 23/51	28% 1450/5222	88% 7/8
Coverage Breakdown by Package					
Name	Classes	Conditionals	Files	Lines	Packages
<a href="#">Stop-tabac</a>	0% 0/1	47% 8/17	0% 0/1	0% 0/5	
<a href="#">Stop-tabac.Classes</a>	100% 1/1	73% 22/30	19% 1/1	27% 58/213	
<a href="#">Stop-tabac.Classes.Controller</a>	19% 3/16	63% 20/32	100% 3/16	5% 120/2665	
<a href="#">Stop-tabac.Classes.Manager</a>	100% 3/3	81% 249/306	100% 3/3	65% 103/158	
<a href="#">Stop-tabac.Classes.Model</a>	75% 6/8	69% 163/235	75% 6/8	65% 669/869	
<a href="#">Stop-tabac.Classes.Service</a>	56% 5/9	N/A	56% 5/9	77% 388/830	
<a href="#">Stop-tabac.Classes.Utils</a>	60% 3/5	70% 7/10	60% 3/5	47% 74/126	
<a href="#">Stop-tabac.Classes.View</a>	25% 2/8		25% 2/8	59% 38/356	



E2E





E2E



# E2E



E2E



Cypress



TestCafe

## BEST PRACTICE

# Writing e2e tests

- **Don't maintain state between tests**
  - Tests should be able to run independently of one another
- **Don't select elements with classes**
  - Think from the user's perspective, or select elements by their intent



# Questions?



**MAKING IT EASY  
TO FOLLOW BEST PRACTICES**

## DISCUSSION

Why are “best practices” important?

## DISCUSSION

# Why are “best practices” important?

- What do we all want at the end of the day?
  - Faster development
  - Fewer bugs
  - More opportunities for learning
- Instead of best practices, think of them as chosen conventions instead.

## DISCUSSION

What makes a convention “good”?

## DISCUSSION

# What makes a convention “good”?

- There are two main factors that contribute to whether a convention is good or not:
  - They enable developers to write great code with a low barrier of entry
  - They are easy to refactor and/or abandon

# DISCUSSION

## How to choose conventions

## DISCUSSION

# How to choose conventions

- There are three main stages to implementing chosen conventions:
  - Selection
  - Implementation
  - Maintenance

# DISCUSSION

## Phase: Selection

- Define the problem
  - Things are rarely objective and absolute
- Avoid bike-shedding
  - Time-constrained voting
  - Disagree and commit
  - 3 month discussion freezes

# DISCUSSION

## Phase: Implementation

- Automate everything
  - Linters (i.e., eslint, stylelint, markdownlint)
  - Formatters (i.e., prettier)
  - Image Optimization (i.e., image-min)
  - Generators (i.e., hygen/plop)
  - Code Snippets (i.e., VS Code)

# DISCUSSION

## Phase: Maintenance

- Build emotional safety and awareness
  - Don't blame individuals
  - Find systematic solutions
  - If you have power, protect your devs

# DISCUSSION

## Phase: Maintenance

- The Jidoka principle
  - Discover an abnormality
  - Stop the process
  - Fix the immediate process
  - Investigate and solve the root cause
- The Andon Cord



# Questions?



# CORE PRINCIPLES

# PRINCIPLE

## Impact over intent

## PRINCIPLE

Context is everything

## PRINCIPLE

All code is compromise



# FINAL THOUGHTS

## DISCUSSION

Migrating from Vue 2 to Vue 3

# DISCUSSION

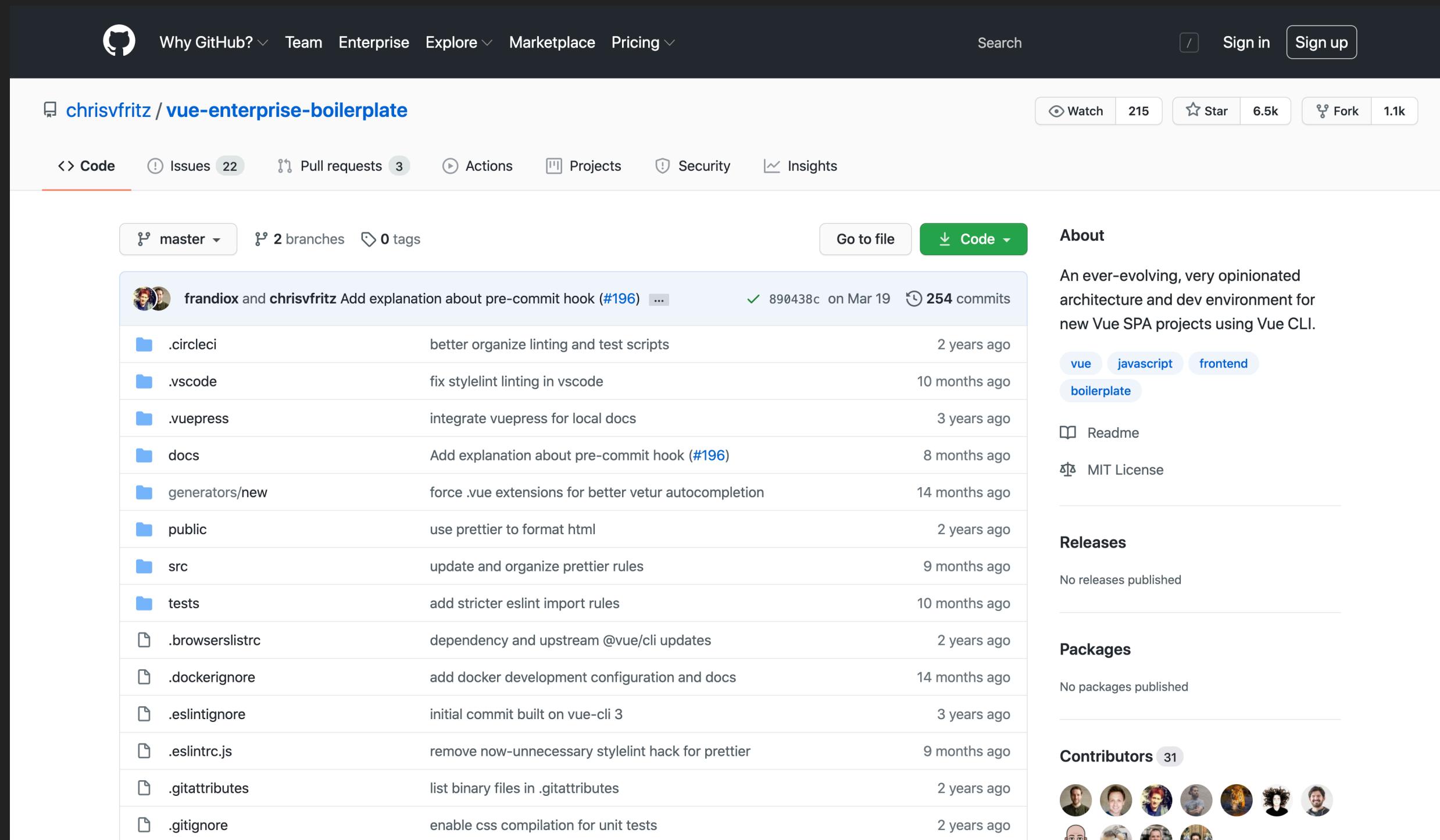
## Migrating from Vue 2 to Vue 3

- Vue 3 is ready for new projects that don't need IE11 support.
- If you have lots of dependencies, wait until those are officially migrated
- Migration tool and IE11 combat build should be ready by the end of the year



# OPEN Q&A

# Vue Enterprise Boilerplate



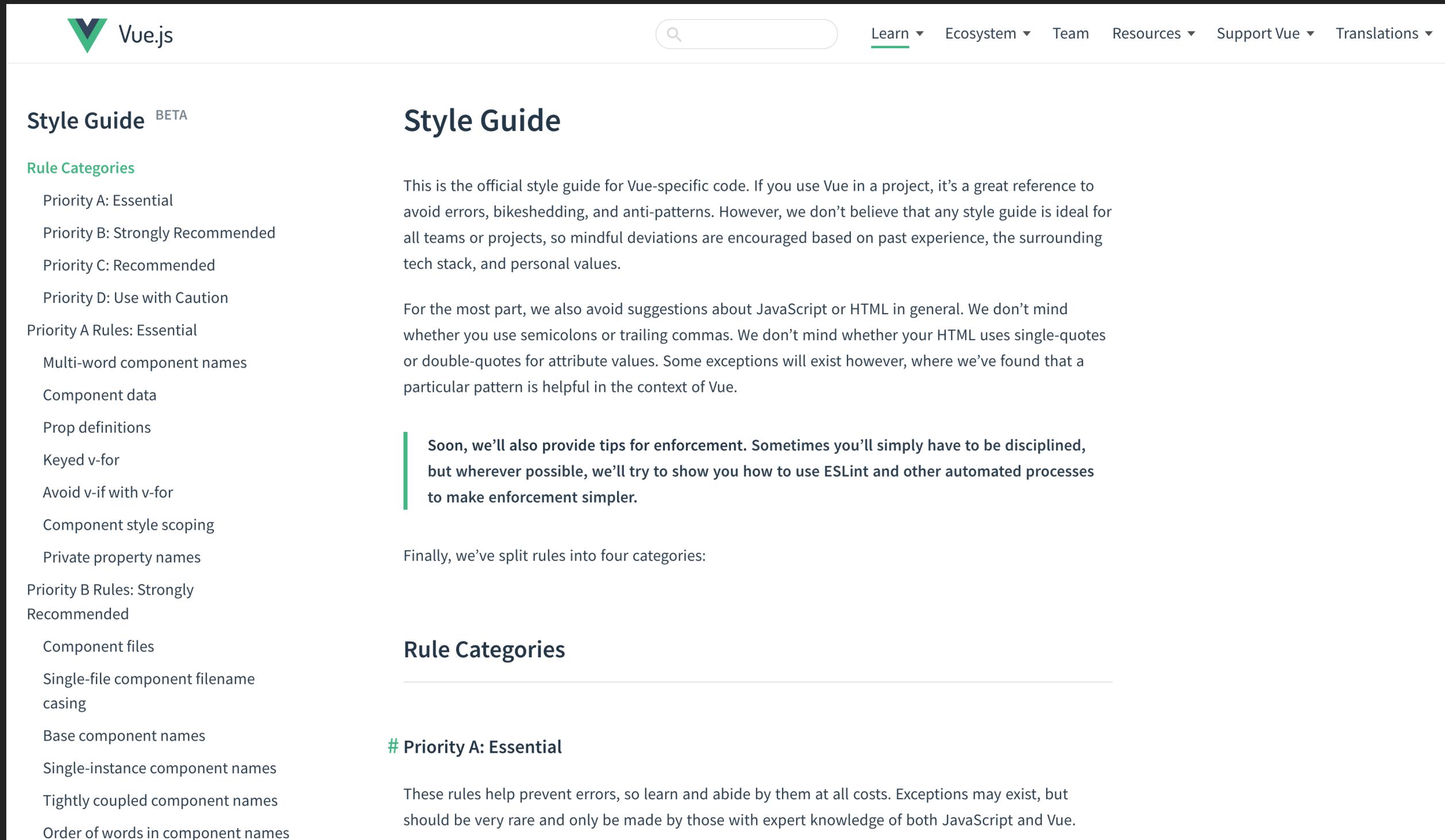
The screenshot shows the GitHub repository page for 'chrisvfritz / vue-enterprise-boilerplate'. The repository has 215 stars, 6.5k forks, and 1.1k issues. The code tab is selected, showing a list of 254 commits from the 'master' branch. The commits are organized into several folders: .circleci, .vscode, .vuepress, docs, generators/new, public, src, tests, .browserslistrc, .dockerignore, .eslintignore, .eslintrc.js, .gitattributes, and .gitignore. The commits are dated from 2 years ago to 10 months ago. The repository description is: 'An ever-evolving, very opinionated architecture and dev environment for new Vue SPA projects using Vue CLI.' It includes tags for vue, javascript, frontend, and boilerplate. It also links to the Readme and MIT License. The releases and packages sections show no activity, and the contributors section lists 31 contributors with their profile pictures.

Commit	Message	Date
.circleci	better organize linting and test scripts	2 years ago
.vscode	fix stylelint linting in vscode	10 months ago
.vuepress	integrate vuepress for local docs	3 years ago
docs	Add explanation about pre-commit hook (#196)	8 months ago
generators/new	force .vue extensions for better vetur autocompletion	14 months ago
public	use prettier to format html	2 years ago
src	update and organize prettier rules	9 months ago
tests	add stricter eslint import rules	10 months ago
.browserslistrc	dependency and upstream @vue/cli updates	2 years ago
.dockerignore	add docker development configuration and docs	14 months ago
.eslintignore	initial commit built on vue-cli 3	3 years ago
.eslintrc.js	remove now-unnecessary stylelint hack for prettier	9 months ago
.gitattributes	list binary files in .gitattributes	2 years ago
.gitignore	enable css compilation for unit tests	2 years ago



<https://github.com/chrisvfritz/vue-enterprise-boilerplate>

# Vue.js Style Guide



The screenshot shows the official Vue.js Style Guide website. The header features the Vue.js logo and navigation links for Learn, Ecosystem, Team, Resources, Support Vue, and Translations. The main content area has a dark background with white text. On the left, a sidebar titled "Style Guide BETA" lists "Rule Categories" including Priority A: Essential, Priority B: Strongly Recommended, Priority C: Recommended, Priority D: Use with Caution, and various sub-categories. On the right, the main content area has a title "Style Guide" and a paragraph explaining the purpose of the guide. It states that the guide is for Vue-specific code and that deviations are encouraged based on experience, the tech stack, and personal values. It also notes that suggestions about JavaScript or HTML in general are avoided. A callout box mentions future enforcement tips using ESLint. Below this, a section titled "Rule Categories" and "Priority A: Essential" provides details on rules for component names and files.

**Style Guide BETA**

**Rule Categories**

- Priority A: Essential
- Priority B: Strongly Recommended
- Priority C: Recommended
- Priority D: Use with Caution
- Priority A Rules: Essential
  - Multi-word component names
  - Component data
  - Prop definitions
  - Keyed v-for
  - Avoid v-if with v-for
  - Component style scoping
  - Private property names
- Priority B Rules: Strongly Recommended
  - Component files
  - Single-file component filename casing
  - Base component names
  - Single-instance component names
  - Tightly coupled component names
  - Order of words in component names

**Style Guide**

This is the official style guide for Vue-specific code. If you use Vue in a project, it's a great reference to avoid errors, bikeshedding, and anti-patterns. However, we don't believe that any style guide is ideal for all teams or projects, so mindful deviations are encouraged based on past experience, the surrounding tech stack, and personal values.

For the most part, we also avoid suggestions about JavaScript or HTML in general. We don't mind whether you use semicolons or trailing commas. We don't mind whether your HTML uses single-quotes or double-quotes for attribute values. Some exceptions will exist however, where we've found that a particular pattern is helpful in the context of Vue.

Soon, we'll also provide tips for enforcement. Sometimes you'll simply have to be disciplined, but wherever possible, we'll try to show you how to use ESLint and other automated processes to make enforcement simpler.

Finally, we've split rules into four categories:

## Rule Categories

### # Priority A: Essential

These rules help prevent errors, so learn and abide by them at all costs. Exceptions may exist, but should be very rare and only be made by those with expert knowledge of both JavaScript and Vue.

<https://vuejs.org/v2/style-guide/>

# Vue.js 3 Migration Guide

Vue.js

Docs API Reference Ecosystem Support Vue GitHub Search

**Introduction**

- Overview
- Quickstart
- Notable New Features
- Breaking Changes
- Global API
- Template Directives
- Components
- Render Function
- Custom Elements
- Other Minor Changes
- Removed APIs
- Supporting Libraries
- Vue CLI
- Vue Router
- Vuex
- Devtools Extension
- IDE Support
- Other Projects

**Details**

- v-for Array Refs
- Async Components
- Attribute Coercion Behavior
- \$attrs includes class & style
- \$children
- Custom Directives
- Custom Elements Interop
- Data Option

**Introduction**

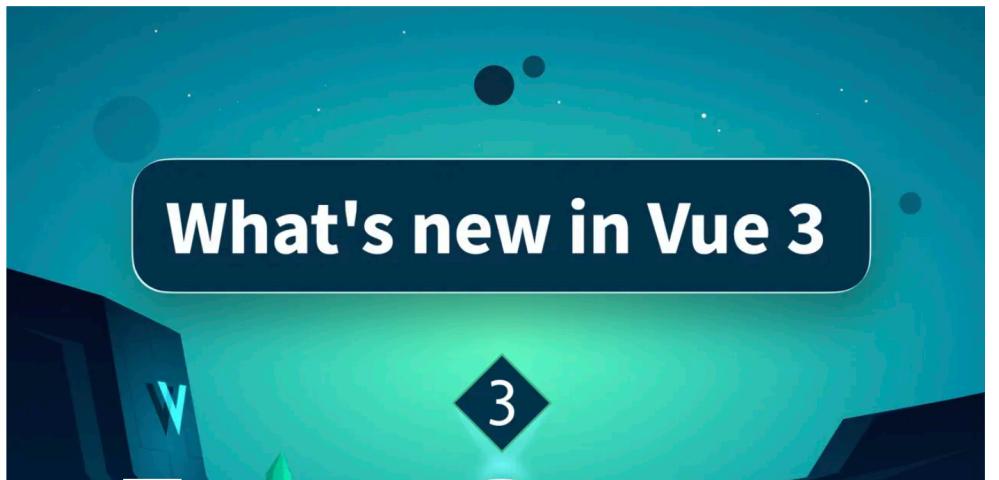
**INFO**

New to Vue.js? Check out our [Essentials Guide](#) to get started.

This guide is primarily for users with prior Vue 2 experience who want to learn about the new features and changes in Vue 3. This is not something you have to read from top to bottom before trying out Vue 3. While it looks like a lot has changed, a lot of what you know and love about Vue is still the same; but we wanted to be as thorough as possible and provide detailed explanations and examples for every documented change.

- [Quickstart](#)
- [Notable New Features](#)
- [Breaking Changes](#)
- [Supporting Libraries](#)

**Overview**



<https://v3.vuejs.org/guide/migration/introduction.html>



# CONGRATULATIONS!



**YOU OTTER BE PROUD  
OF YOURSELF!**

Thanks  
everyone!

