



Proven Patterns for Building Vue Apps

Ben Hong

<https://www.bencodezen.io>

VueConfUS 2024

WiFi: Renaissance_GUEST
PASSWORD: arts2020



Introductions

Who am I?



November 12, 2017

TM & © Universal Studios and Amblin





Ben Hong

Independent Consultant

@bencodezen

The screenshot shows the official Vue.js website. At the top, the word "Vue.js" is displayed next to a magnifying glass icon. A navigation bar includes links for "Learn", "Ecosystem", "Resources", "Extended LTS NEW", and "Translations". Below the header is the large Vue.js logo, followed by the text "The Progressive JavaScript Framework". Three buttons are present: "WHY VUE.JS?", "GET STARTED", and "GITHUB". The main content area features a large, bold title "The Progressive JavaScript Framework" in green and blue. Below it is a subtitle: "An approachable, performant and versatile framework for building web user interfaces." Three calls-to-action buttons are shown: "Why Vue", "Get Started →", and "Install". At the bottom, there's a "Special Sponsor" section featuring the Appwrite logo and the tagline "Build Fast. Scale Big. All in One Place." To the right, three columns describe the framework's qualities: "Approachable", "Performant", and "Versatile".

Vue.js

Learn ▾ Ecosystem ▾ Resources ▾ Extended LTS NEW Translations ▾

The Progressive JavaScript Framework

WHY VUE.JS? GET STARTED GITHUB

Vue.js Search K

Docs ▾ API Playground Ecosystem ▾ About ▾ Sponsor Partners

The Progressive JavaScript Framework

An approachable, performant and versatile framework for building web user interfaces.

Why Vue Get Started → Install

Special Sponsor appwrite Build Fast. Scale Big. All in One Place.

Approachable
Builds on top of standard HTML, CSS and JavaScript with intuitive API and world-class documentation.

Performant
Truly reactive, compiler-optimized rendering system that rarely requires manual optimization.

Versatile
A rich, incrementally adoptable ecosystem that scales between a library and a full-featured framework.



A dark blue background featuring a complex network of thin, light blue lines forming various geometric shapes, including hexagons and triangles. The lines are slightly curved and intersect at different angles, creating a sense of depth and perspective.

Who are you?

Who are you?

Fill out the following survey:

<https://form.typeform.com/to/VpX2LvR4>

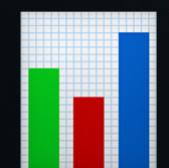
Get to know your neighbors

Name

Job / Title

What is their top goal for this workshop?

Get to know your neighbors



Survey results!





Setup



Schedule



Time Slot (CDT)	Event
9:00AM - 10:30AM	Part 1
10:30AM - 10:40AM	☕ Short Break ☕
10:40AM - 12:00PM	Part 2
12:00PM - 1:00PM	🥪 🥗 Lunch 🍟🥤
1:00PM - 2:00PM	Part 3
2:00PM - 2:15PM	🍪 Short Break 🍫
2:15PM - 3:45PM	Part 4
3:45PM - 4:00PM	🧁 Short Break 🍰
4:00PM - 5:00PM	Part 5

Content

Content

- Languages
- Components
- Routing
- State Management
- Reusability & Composition
- Testing
- Making It Easy to Follow Best Practices

Workshop Format

Workshop Format

1. Learn
2. Question
3. Apply

Workshop Format

1. Learn

Explanations

Examples

Stories

2. Question

Clarifications

What-ifs

3. Apply

Code

Experiment

Pairing

Resources

Resources

These slides

<https://slides.com/bencodezen/proven-patterns-vueconfus2024>

Please open it in a tab and bookmark it for today

Resources

Vue Enterprise Boilerplate

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

Proven Patterns Workshop

<https://github.com/bencodezen/proven-patterns-workshop>

Please star / bookmark / clone them now

Resources

Your Projects

I encourage you to pull from your experience
to make the most of this workshop.

Resources

Your Projects

I encourage you to pull from your experience
to make the most of this workshop.

Me

I'm here to talk about your codebase
and any ideas and questions you're having.

PARTICIPATION TIPS

PARTICIPATION TIPS

This is a **safe place** to learn from one another

<https://vuejs.org/about/coc.html>

PARTICIPATION TIPS

Questions are **welcome!**

Discussions are how we can make the most of our time together.

PARTICIPATION TIPS

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

Discussions are how we can make the most of our time together.

PARTICIPATION TIPS

Please **no recording.**

Out of respect for you and your participants' privacy

PARTICIPATION TIPS

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

Or if you need to leave for any other reason, feel free to.

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.

PARTICIPATION TIPS

Raise your hand for **questions** at any time.

Discussions are how we can make the most of our time together.



Languages



HelloVueConfUS.vue

```
<template>
  <h1>Hello VueConfUS 2024!</h1>
</template>
```

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

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

LANGUAGE

HTML

BEST PRACTICE

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

BEST PRACTICE

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

- A benefit of doing this is that Vue has an opportunity to parse it before the browser does

BEST PRACTICE

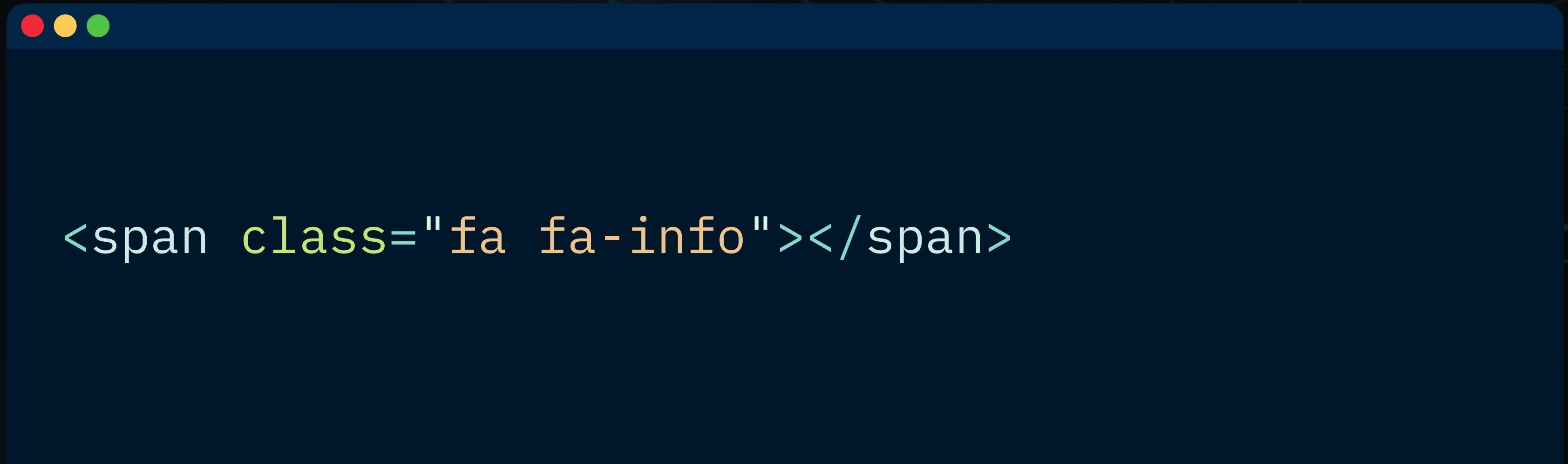
All HTML should exist in .vue files as a <template> or render function.

- 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 as a <template> or render function.

- Self-closing tags

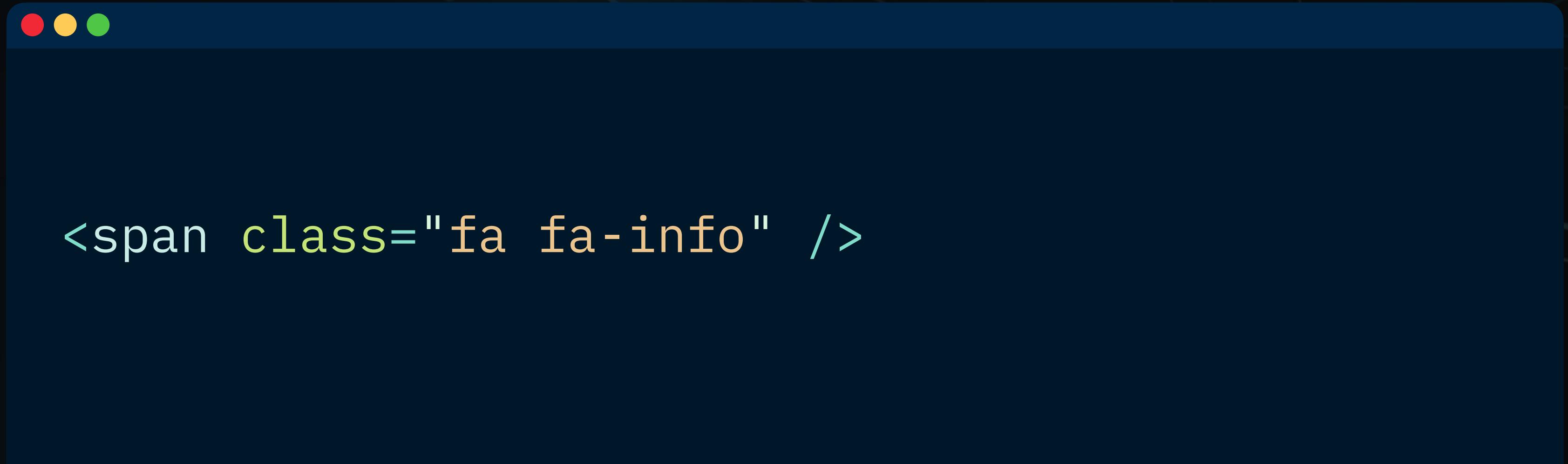


```
<span class="fa fa-info"></span>
```

BEST PRACTICE

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

- Self-closing tags

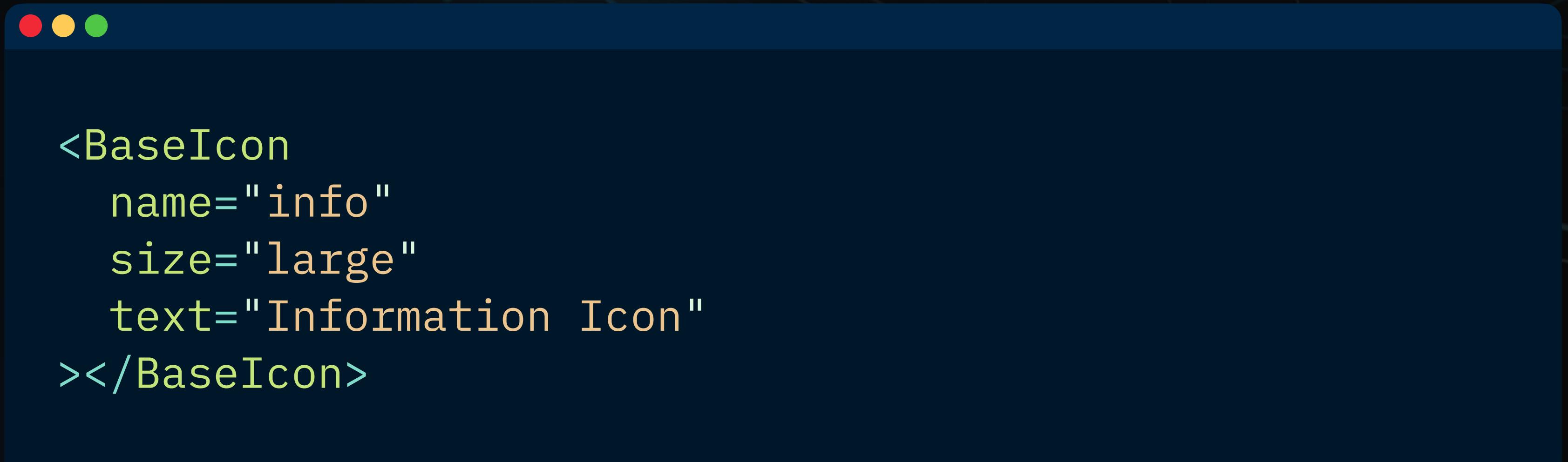


```
<span class="fa fa-info" />
```

BEST PRACTICE

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

- Self-closing tags



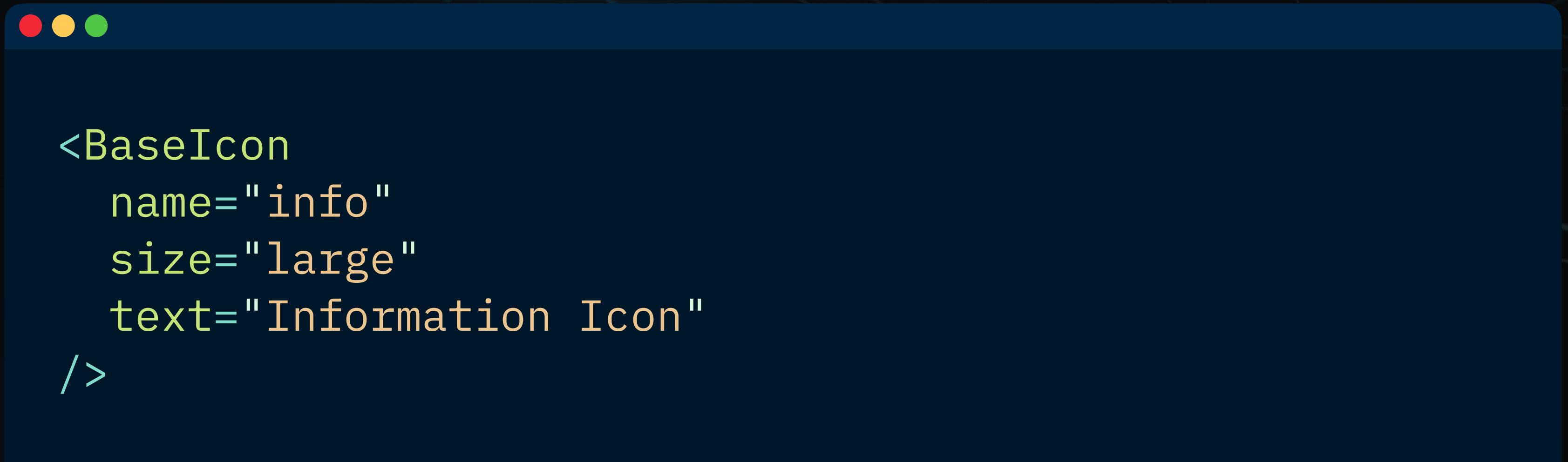
A screenshot of a code editor window showing a Vue component template. The window has a dark blue header bar with three circular icons (red, yellow, green) in the top-left corner. The main area displays the following code:

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

BEST PRACTICE

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

- Self-closing tags



A screenshot of a code editor window showing a Vue component template. The window has a dark blue header bar with three circular icons (red, yellow, green) in the top-left corner. The main area contains the following code:

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

BEST PRACTICE

All HTML should exist in .vue files as 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 vs Render Function

TECHNIQUE

Template



MyFirstComponent.vue

```
<template>
  <h1 class="title">Hello VueConfUS!</h1>
</template>
```

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

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

TECHNIQUE

Render Function



MyFirstComponent.vue

```
<template>
  <h1 class="title">Hello VueConfUS!</h1>
</template>
```

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

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

TECHNIQUE

Render Function



MyFirstComponent.vue

```
<template>
  <h1 class="title">Hello VueConfUS!</h1>
</template>
```

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

```
export default {
  // ...
}
```

```
</script>
```

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

TECHNIQUE

Render Function



MyFirstComponent.vue

```
<template>
  <h1 class="title">Hello VueConfUS!</h1>
</template>

<script>
import { h } from 'vue'

export default {
  render() {
    return h()
  }
}
</script>
```

TECHNIQUE

Render Function



MyFirstComponent.vue

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

export default {
  render() {
    return h('h1', { class: 'title' }, 'Hello VueConfUS!')
  }
}
</script>

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

TECHNIQUE

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?



A dark blue background featuring a complex network of thin, light blue lines forming various geometric shapes, including hexagons and triangles, creating a sense of depth and motion.

LANGUAGE
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>
```

TECHNIQUE

State Driven CSS

TECHNIQUE

State Driven CSS



```
<template>
  <h1 class="title">Make my color dynamic!</h1>
</template>
```



```
<style scoped>
.title {
  color: red;
}
</style>
```

TECHNIQUE

State Driven CSS



```
<script>
export default {
  data: () => ({
    textColor: 'papayawhip'
  })
}
</script>

<template>
  <h1 class="title">Color me whatever you want!</h1>
  <input v-model="textColor" type="color" />
</template>

<style scoped>
.title {
  color: v-bind(textColor);
}
</style>
```



Questions?

LANGUAGE

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?



CODE / EXPERIMENT



In the boilerplate

- Create a component using the render function
- Experiment with different CSS techniques (scoped, modules, state driven, etc.)
- Try writing some TypeScript!

In your app

- Refactor a template to use the render function instead.
- Refactor a component to use CSS modules
- Setup TypeScript in your project and add it to a component.



SHORT BREAK

Be back at 10:45AM!



COMPONENTS



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

AppPrefixedName.vue / **Base**PrefixedName.vue

Reusable, globally registered UI components.

AppButton, AppModal, BaseDropdown, BaseInput

ThePrefixedName.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)  
  }  
}
```

✗ Not recommended

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)  
  }  
}
```



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 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 VueConfUS 2024!</h1>
</template>

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

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



MyFirstComponent.vue

```
<template>
  <h1>Hello VueConfUS 2024!</h1>
</template>

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

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



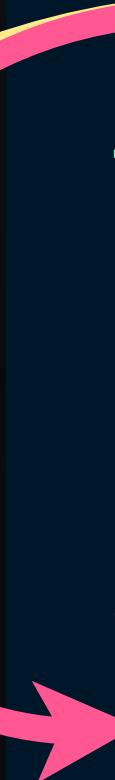


MyFirstComponent.vue

```
<template>
  <h1>Hello VueConfUS 2024!</h1>
</template>

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

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





MyFirstComponent.vue

```
<template>
  <h1>Hello VueConfUS 2024!</h1>
</template>

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

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



MyFirstComponent.vue

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

<template>
  <h1>Hello VueConfUS 2024!</h1>
</template>

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

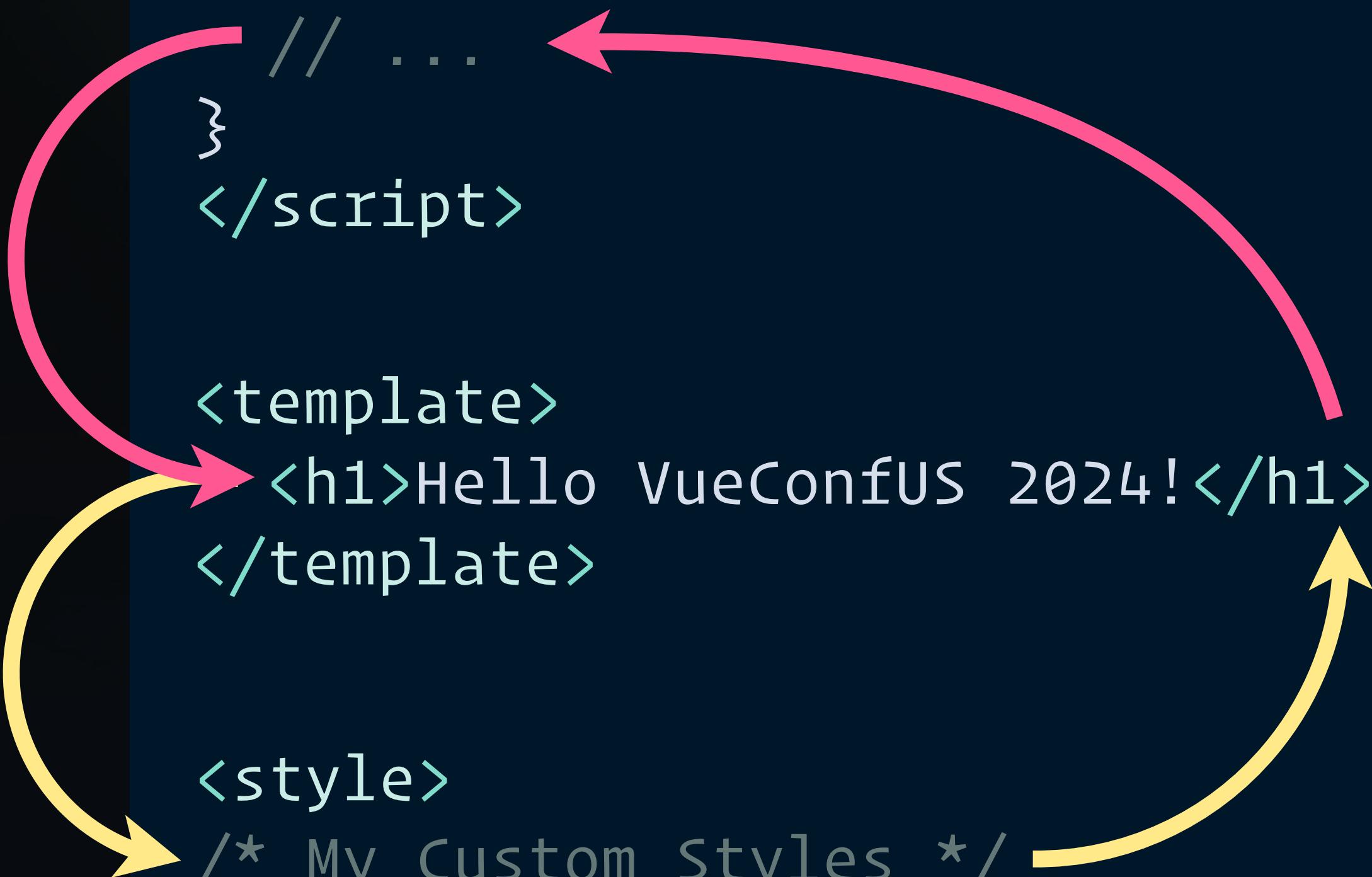


MyFirstComponent.vue

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

<template>
<h1>Hello VueConfUS 2024!</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

TECHNIQUE

Namespaced Components



```
// @/components/index.ts
export { default as List } from './TodoList.vue'
export { default as Item } from './TodoItem.vue'

// @/App.vue
<script setup lang="ts">
import * as Todo from '@/components/todo'
</script>

<template>
  <div>
    <Todo.List>
      <Todo.Item v-for="i in 5" :key="i"> Todo {{ i }} </Todo.Item>
    </Todo.List>
  </div>
</template>
```

PRO TIP

Auto import components

PRO TIP

Auto import components

No more multi-line component import statements...

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

Do this in a performative way with:

<https://github.com/antfu/unplugin-vue-components>



Questions?

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

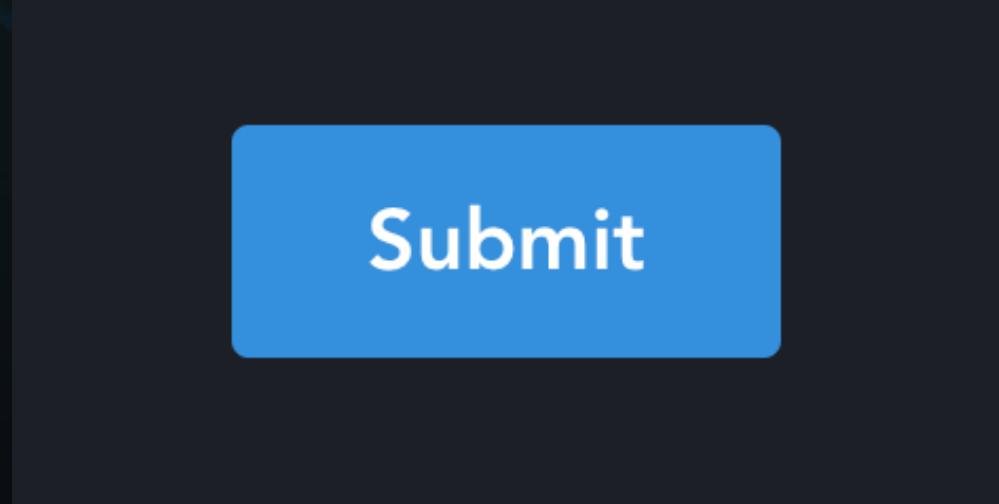
```
<script>
export default {
  props: {
    label: {
      type: String,
      default: 'Home',
      validator: (value) => {
        return ['Home', 'About'].indexOf(value) !== -1
      }
    }
  }
}
</script>
```



Let's do a
Coding Experiment

Task 1

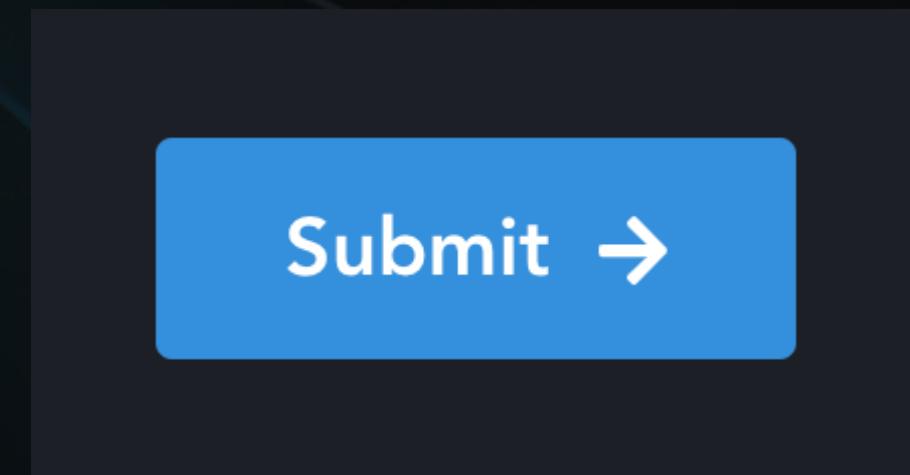
Create a button component that can display text specified in the parent component



Submit

Task 2

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



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

This is the code responsible for displaying an arrow.

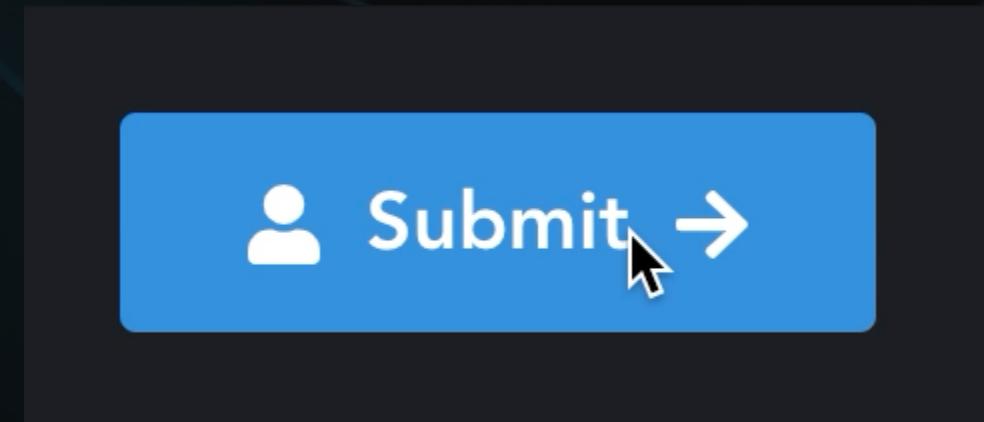
Task 3

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

Submit →

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 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.

The background of the slide features a dark, abstract geometric design. It consists of numerous thin, light-blue lines that form a complex network of triangles and hexagons. Some of these shapes are filled with a very light shade of blue. Small, isolated dots of varying sizes are scattered across the dark background, some aligned with the grid lines.

Is there a better another alternative?



Obviously.

Is there a better another alternative?

Recommended solution

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

TECHNIQUE

Slots

```
<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>
```

BaseButton.vue

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

BaseButton.vue

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

App.vue

```
<template>
  <BaseButton>
    Submit
  </BaseButton>
</template>
```

BaseButton.vue

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

App.vue

```
<template>
  <BaseButton>
    Submit
    <PulseLoader v-if="isLoading" color="#fff" size="6px"/>
    <AppIcon v-else icon="arrow-right"/>
  </BaseButton>
</template>
```

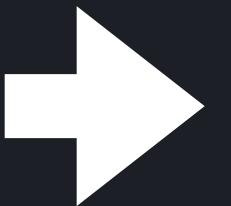
PROBLEM

What if you want to define multiple slots?

Default Slot

NavigationLink.vue

```
<a  
  :href="url"  
  class="nav-link"  
>  
  <slot></slot>  
</a>
```

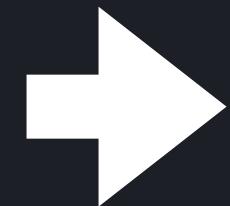


```
<navigation-link url="/profile">  
  <span class="fa fa-user"/>  
  Your Profile  
</navigation-link>
```

Named Slots

BaseLayout.vue

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```



```
<base-layout>
  <template v-slot:header>
    <h1>Here might be a page title</h1>
  </template>

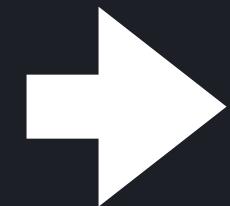
  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <template v-slot:footer>
    Here's some contact info
  </template>
</base-layout>
```

Named Slots

BaseLayout.vue

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```



```
<base-layout>
  <template #header>
    <h1>Here might be a page title</h1>
  </template>

  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <template #footer>
    Here's some contact info
  </template>
</base-layout>
```

Dynamic Slot Names

```
<base-layout>
  <template v-slot:[dynamicSlotName]>
    ...
  </template>
</base-layout>
```

PROBLEM

How do you access component data from the slot?

Scoped Slots

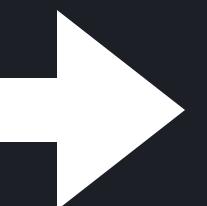
```
// todo-list.vue
<ul>
  <li v-for="todo in todos" :key="todo.id">
    <slot name="header" :header="todo.header" >
      <!-- Fallback content -->
      {{ todo.text }}
    </slot>
    <slot :todo="todo" :newData="newData" >
      <!-- Fallback content -->
      {{ todo.text }}
    </slot>
  </li>
</ul>
```



```
<todo-list :todos="todos">
  <template v-slot="slotProps">
    {{ slotProps.todo }}
  </template>
  <template v-slot:header="slotProps">
    {{ slotProps.header }}
  </template>
</todo-list>
```

Destructuring slot-scope

```
<todo-list :todos="todos">  
  <template v-slot="scope">  
    <AppIcon  
      v-if="scope.todo.completed"  
      icon="checked"  
    />  
    {{ scope.todo.text }}  
  </template>  
</todo-list>
```



```
<todo-list :todos="todos">  
  <template v-slot="{ todo }">  
    <AppIcon  
      v-if="todo.completed"  
      icon="checked"  
    />  
    {{ todo.text }}  
  </template>  
</todo-list>
```

Use slots for:

- Content distribution (like layouts)
- Creating larger components by combining smaller components
- Default content in Multi-page Apps
- Providing a wrapper for other components
- Replace default component fragments

Use scoped slots for:

- Applying custom formatting/template to fragments of a component
- Creating wrapper components
- Exposing its own data and methods to child components

Pros

- Great for creating reusable and *composable components*
- Receiving properties from slot-scope is explicit

Cons

- Properties received through slot-scope can't be easily used in component script
 - However, you can pass those to methods inside the template as arguments



Questions?



A dark blue background featuring a complex network of thin, light blue lines forming various geometric shapes like hexagons and triangles. Small, semi-transparent white dots are scattered across the surface, particularly concentrated along the lines.

Slots > Props

Composition > Configuration

With composition, you're less restricted by what you were building at first.

With configuration, you have to document everything and new requirements means new configuration.

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'
      }
    }
  }
}
// ...
```

<Component :is>

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



```
<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'
      }
    }
  }
}
// ...
}
```

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

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" />
```

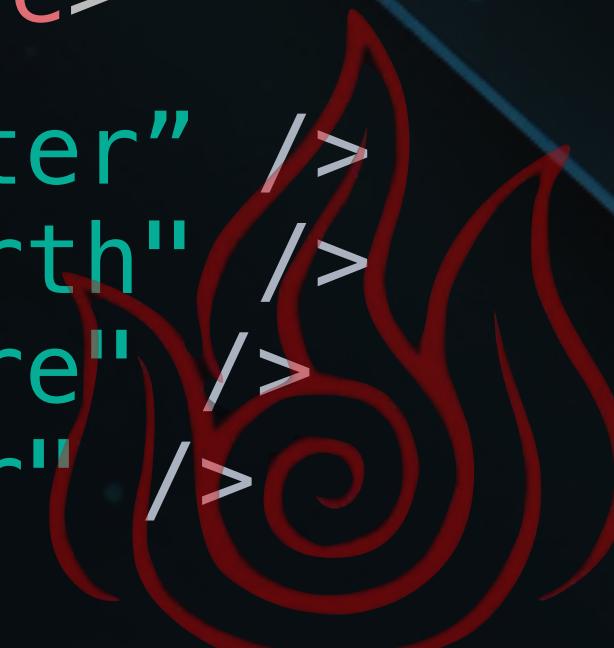
```
   <FontAwesome icon="earth" />
```

```
   <FontAwesome icon="fire" />
```

```
   <FontAwesome icon="air" />
```

```
 </p>
```

```
</template>
```



```
<template>
```

```
 <p>
```

```
   <FontAwesome icon="water" />
```

```
   <FontAwesome icon="earth" />
```

```
   <FontAwesome icon="fire" />
```

```
   <FontAwesome icon="air" />
```

```
 </p>
```

```
</template>
```

```
<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...">
```



The diagram illustrates the flow of props and attributes from the parent component to the child template. It shows three green arrows originating from the prop and attribute declarations in the parent component's code and pointing to their corresponding elements in the child template. One arrow points from '@input="filterData"' to the input element. Another arrow points from 'label="Filter results"' to the div element. A third arrow points from 'placeholder="Type in here..."' to the input element.

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>
```



The diagram illustrates the flow of data from the parent component's props and attributes to the child component's template. Three red arrows originate from the 'filterData' prop, the 'label' attribute, and the 'placeholder' attribute in the first code block, and point to their respective counterparts in the second code block: the input element, the div element, and the input element respectively.

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 Vue 3...

```
<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 Vue 3...

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

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



Questions?



CODE / EXPERIMENT



In the boilerplate

- Create a `BaseInputText` component using the transparent wrapper pattern
- Use dynamic components to toggle between different components

In your app

- Look for opportunities to rename your components
- See if you can optimize any props, emits or slots usage
- Refactor a component using the vendor wrapper pattern



LUNCH BREAK

Be back at 1:00PM!