



Proven Patterns for Building Vue Apps

Ben Hong

<https://www.bencodezen.io>

VueConfUS 2024

WiFi: Renaissance_GUEST
PASSWORD: arts2020



Introductions

The background features a dark blue gradient with a complex network of thin, light blue lines forming a hexagonal lattice. Small white dots are scattered across the surface, particularly concentrated along the edges of the hexagons.

Who am I?



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. To the right are navigation links: "Learn", "Ecosystem", "Resources", "Extended LTS NEW", and "Translations". Below this is the large Vue.js logo, followed by the tagline "The Progressive JavaScript Framework". Underneath are three buttons: "WHY VUE.JS?", "GET STARTED", and "GITHUB". A secondary navigation bar at the top of the main content area includes links for "Docs", "API", "Playground", "Ecosystem", "About", "Sponsor", and "Partners". The main content 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." There are also three descriptive sections: "Approachable", "Performant", and "Versatile", each with a brief description.

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>

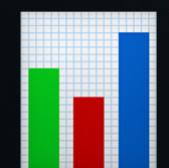
Get to know your neighbors

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

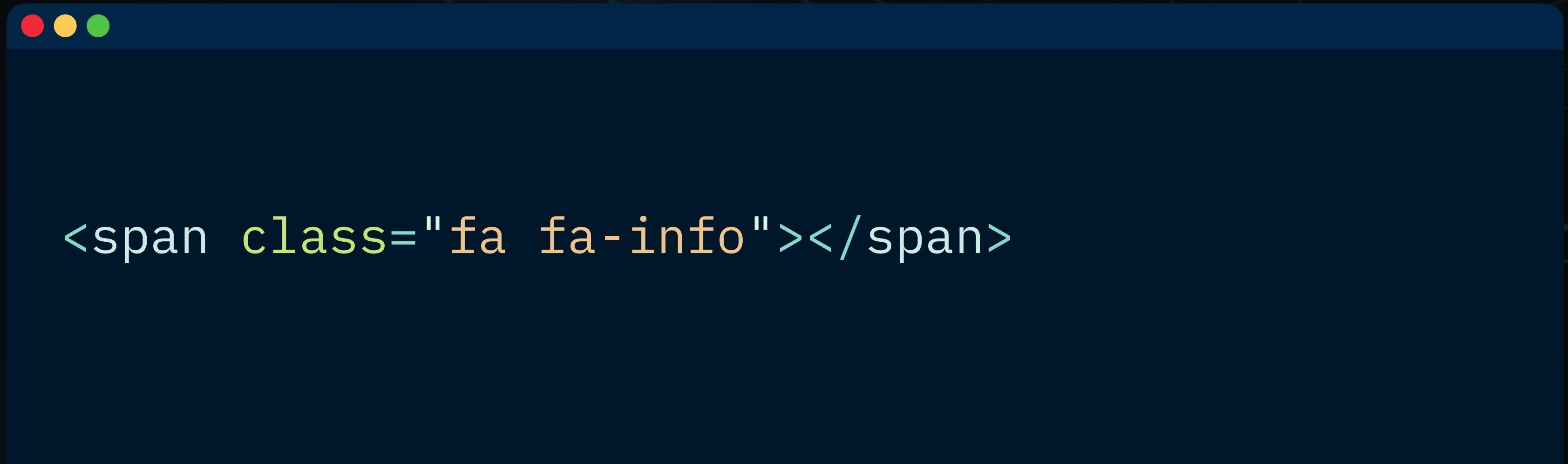
All HTML should exist in .vue files as a <template> or render function.

- 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

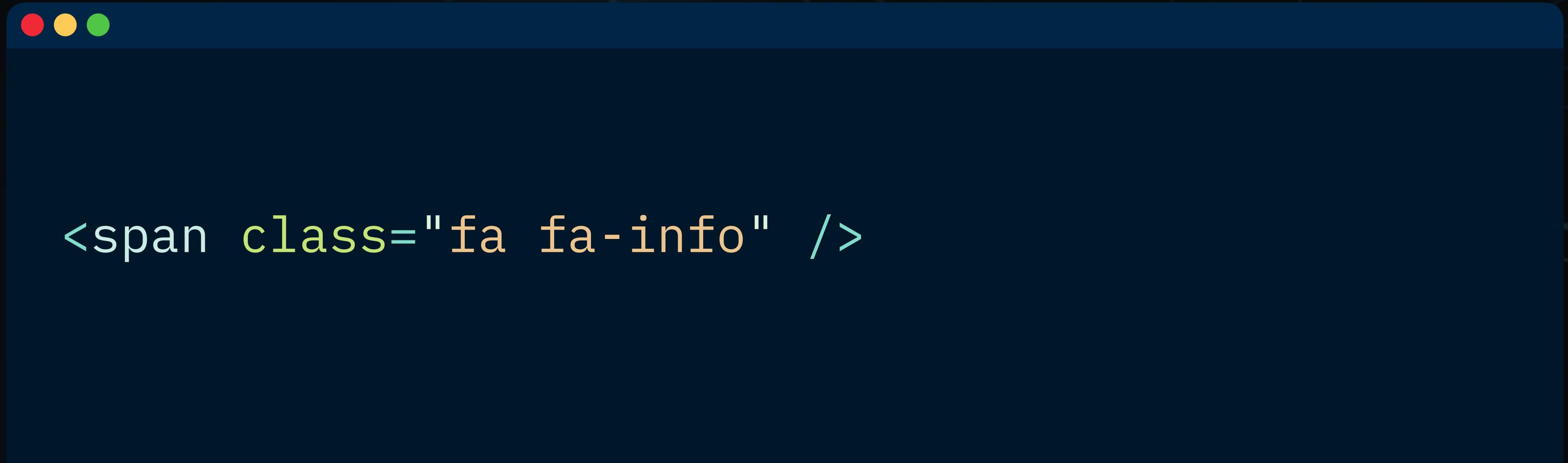


```
<span class="fa fa-info"></span>
```

BEST PRACTICE

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

- Self-closing tags

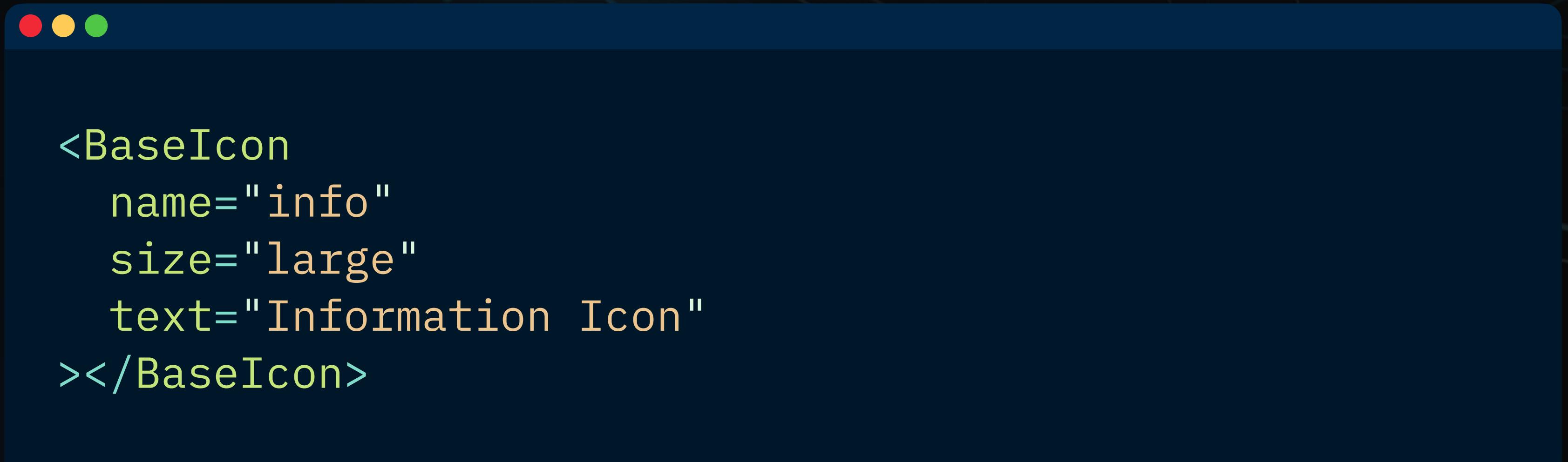


```
<span class="fa fa-info" />
```

BEST PRACTICE

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

- Self-closing tags



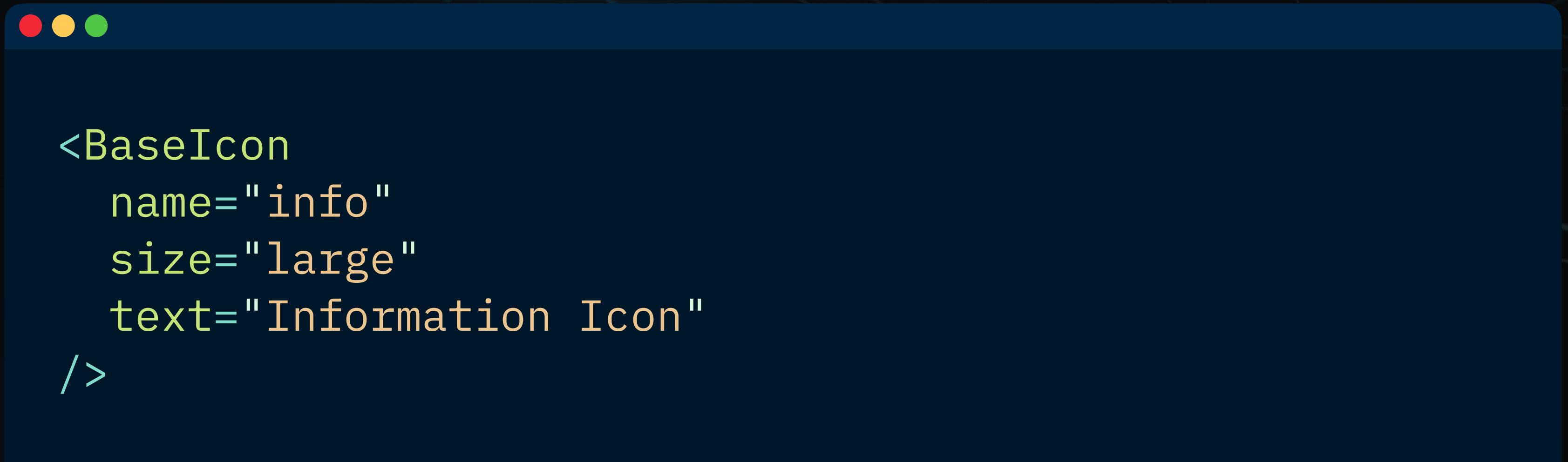
A screenshot of a code editor window showing a Vue component template. The window has a dark blue header bar with three circular icons (red, yellow, green) in the top-left corner. The main area displays the following code:

```
<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 background featuring a complex network of thin blue lines forming various geometric shapes, including hexagons and triangles, creating a sense of depth and perspective.

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 playground

- 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>
```



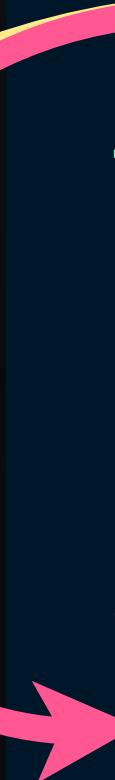


MyFirstComponent.vue

```
<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>
```



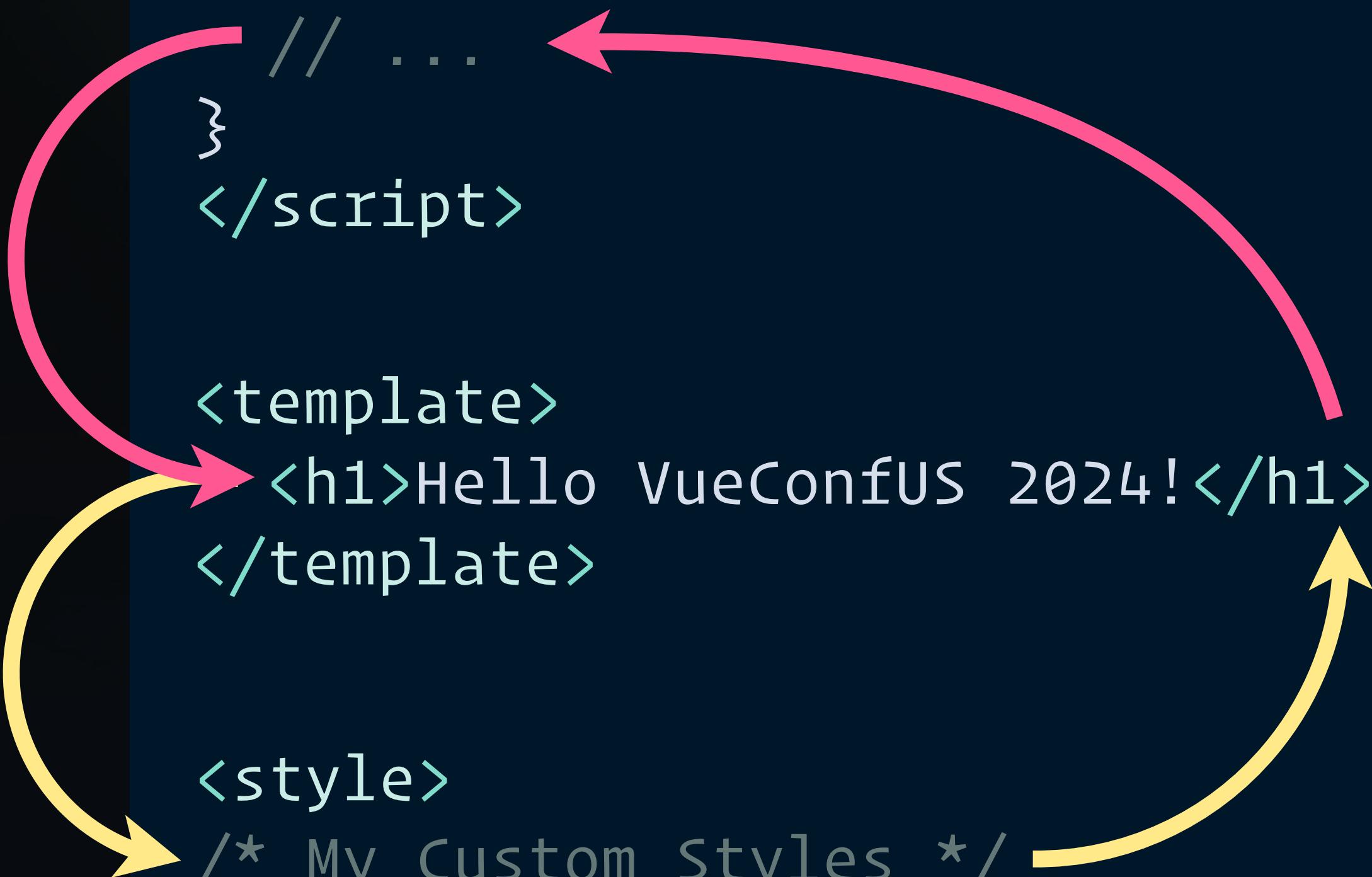
MyFirstComponent.vue

```
<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

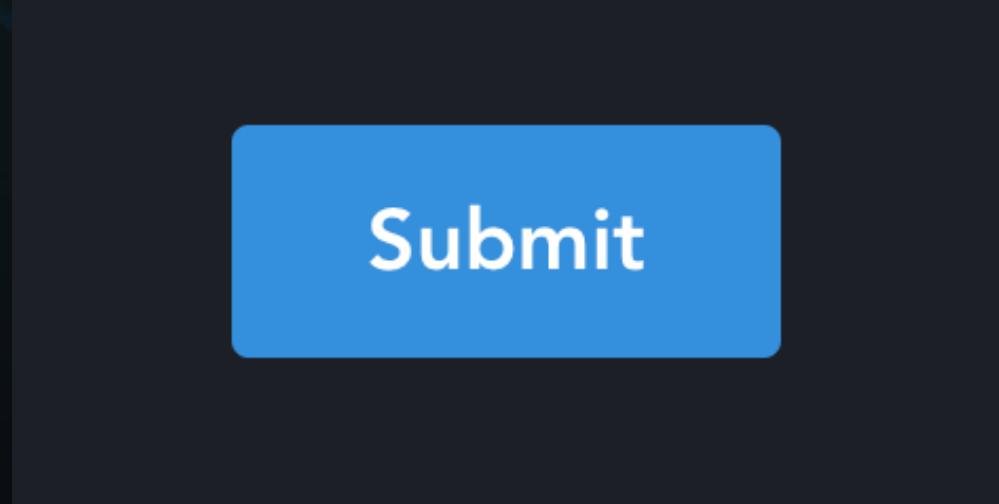
```
<script>
export default {
  props: {
    label: {
      type: String,
      default: 'Home',
      validator: (value) => {
        return ['Home', 'About'].indexOf(value) !== -1
      }
    }
  }
}
</script>
```



Let's do a
Coding Experiment

Task 1

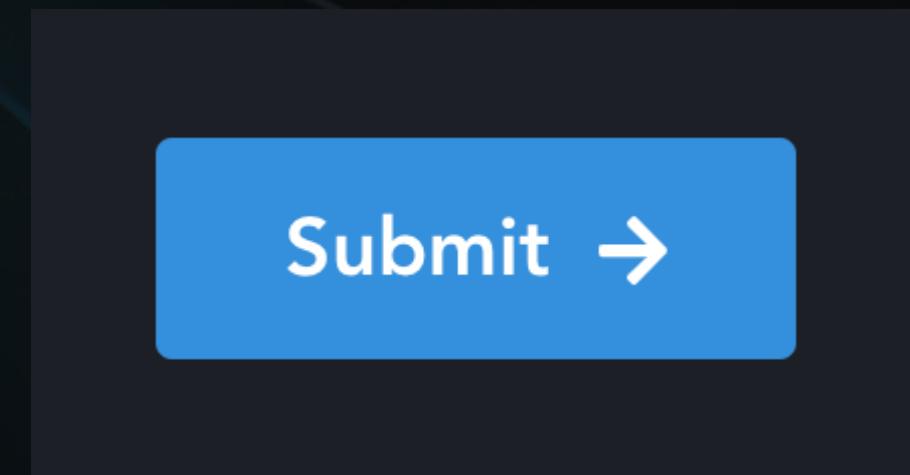
Create a button component that can display text specified in the parent component



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.

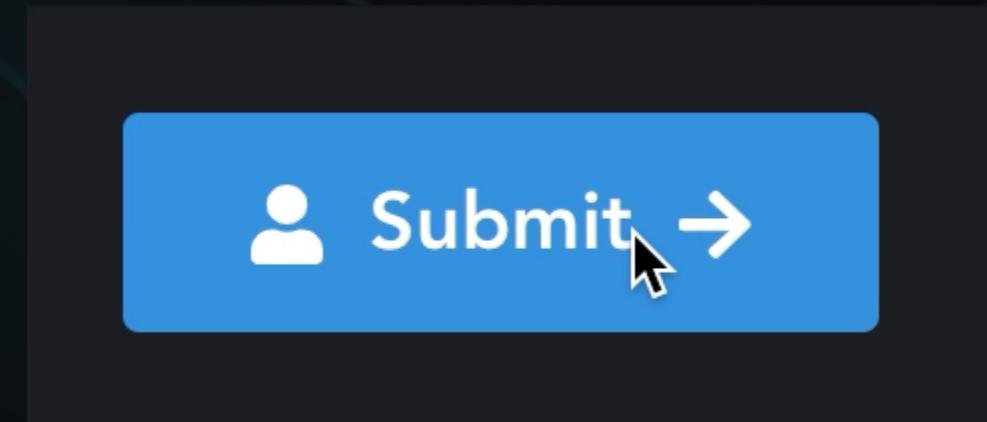
Task 3

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>
    </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 of varying lengths and orientations, some forming small triangles or rectangles. Scattered throughout the space are small, semi-transparent blue dots of different sizes.

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>
```

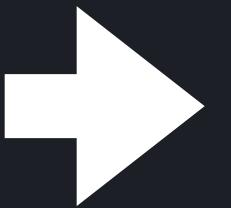
PROBLEM

What if you want to define multiple slots?

Default Slot

NavigationLink.vue

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

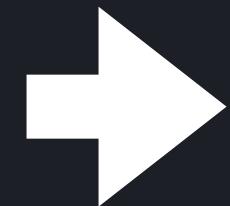


```
<navigation-link url="/profile">  
  <span class="fa fa-user"/>  
  Your Profile  
</navigation-link>
```

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>

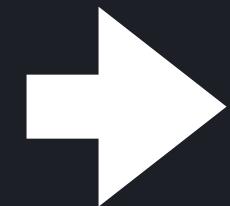
  <p>A paragraph for the main content.</p>
  <p>And another one.</p>

  <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

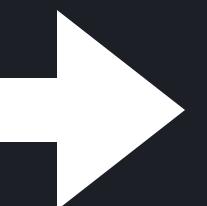
```
// todo-list.vue
<ul>
  <li v-for="todo in todos" :key="todo.id">
    <slot name="header" :header="todo.header" >
      <!-- Fallback content -->
      {{ todo.text }}
    </slot>
    <slot :todo="todo" :newData="newData" >
      <!-- Fallback content -->
      {{ todo.text }}
    </slot>
  </li>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

The background features a dark blue gradient with a subtle grid pattern. Overlaid on this are several thin, translucent blue lines forming a hexagonal prism-like structure. Small white dots are scattered across the surface of these 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



LUNCH BREAK

Be back at 1:00PM!

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>
```

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

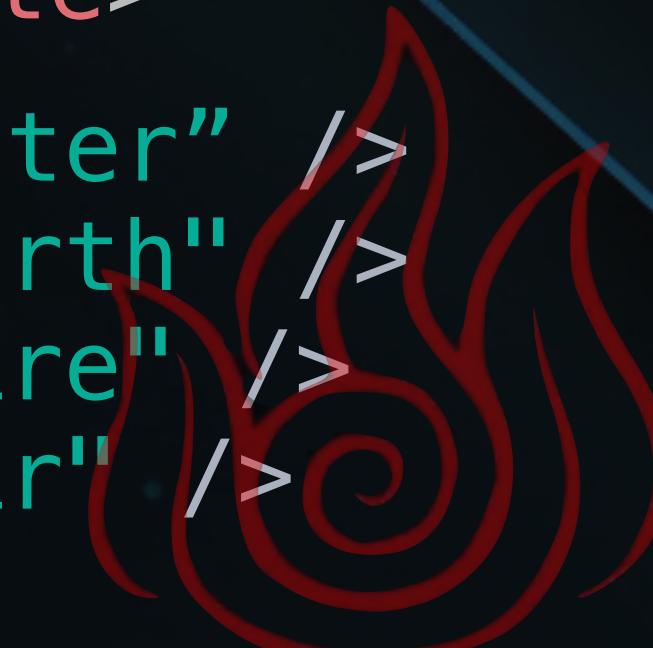
```
<FontAwesome icon="earth" />
```

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

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

```
 </p>
```

```
</template>
```



```
<template>
```

```
 <p>
```

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

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

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

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

```
 </p>
```

```
</template>
```

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

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

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

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

```
<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 component. It shows three green arrows originating from the prop and attribute declarations in the parent component's template and pointing to their corresponding elements in the child component's template.

- A top arrow points from the `@input="filterData"` declaration to the `<input` element in the child component.
- A middle arrow points from the `label="Filter results"` declaration to the `<div>` element in the child component.
- A bottom arrow points from the `placeholder="Type in here..."` declaration to the `<input>` element in the child component.

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



ROUTING

BEST PRACTICES

Routing

BEST PRACTICES

Routing

- There are three main categories of routing:
 - **View (Page) 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

unhead/vue

<https://unhead.unjs.io/setup/vue/installation>

TOOL

unhead/vue

```
<script setup>
import { useHead } from '@unhead/vue'

const username = ref('bencodezen')

useHead({
  title: `${username.value} - VueConfUS 2024`,
  meta: [
    {
      name: 'description',
      content: `This is the profile page for ${username.value}`
    }
  ]
})
</script>
```

TECHNIQUE

Lazy load routes

TECHNIQUE

Lazy load routes

```
const routes = [
  {
    path: '/',
    name: 'home',
    component: () => import('@/pages/index.vue')
  },
  {
    path: '/about',
    name: 'about',
    // route level code-splitting
    // this generates a separate chunk for this route
    // which is lazy-loaded when the route is visited.
    component: () => import('@/pages/about.vue')
  }
]
```

TECHNIQUE

<keep-alive>

BEST PRACTICE

Keep params in route

TOOLS

unplugin-vue-route

<https://github.com/posva/unplugin-vue-router>



CODE / EXPERIMENT

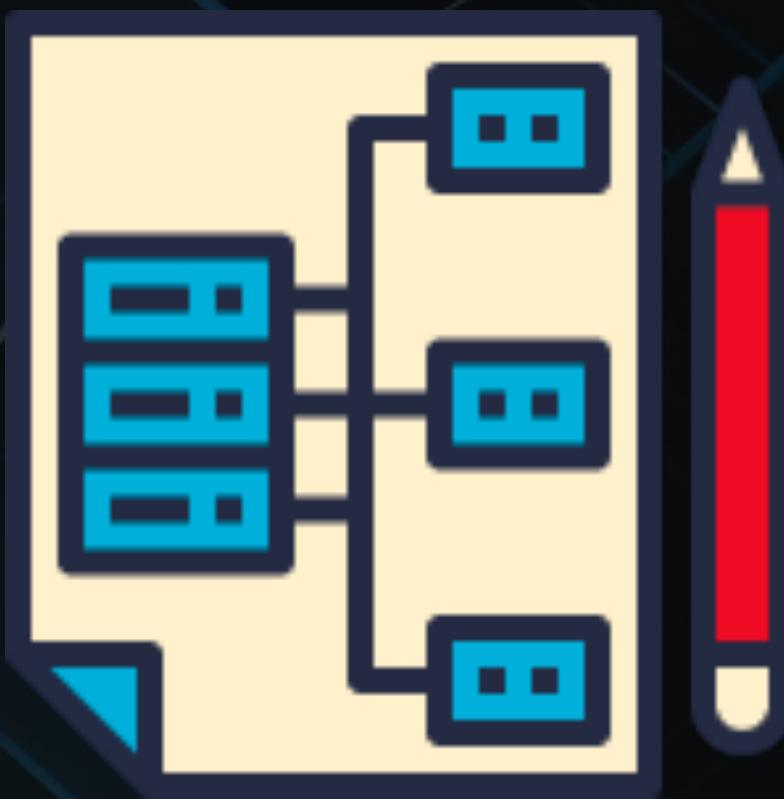


In the boilerplate

- Create a new lazy route
- Add a new layout and use it inside a page component
- Use unhead/vue to manipulate tags in the head

In your app

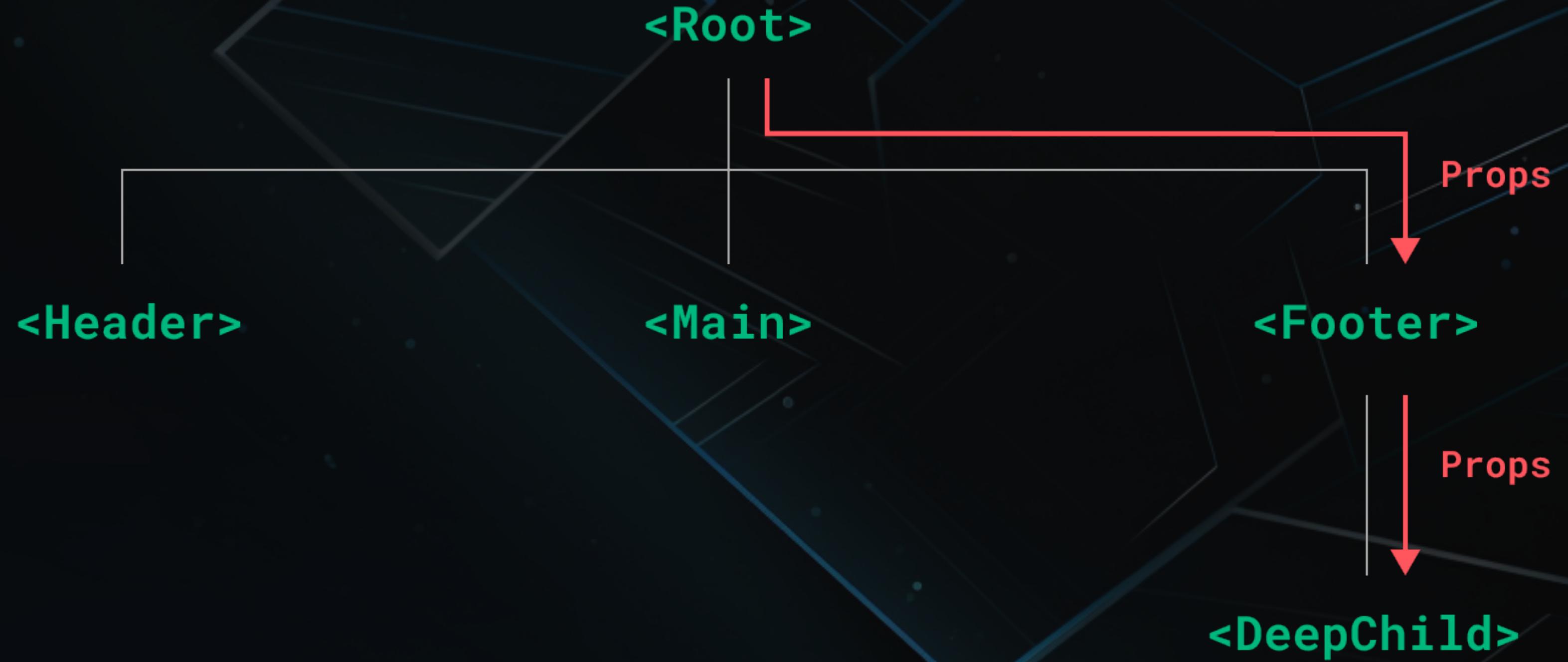
- Create layouts for your app
- Use router params to enable deep linking
- Add unhead/vue to manage head tag info



REUSABILITY & COMPOSITION

PROBLEM

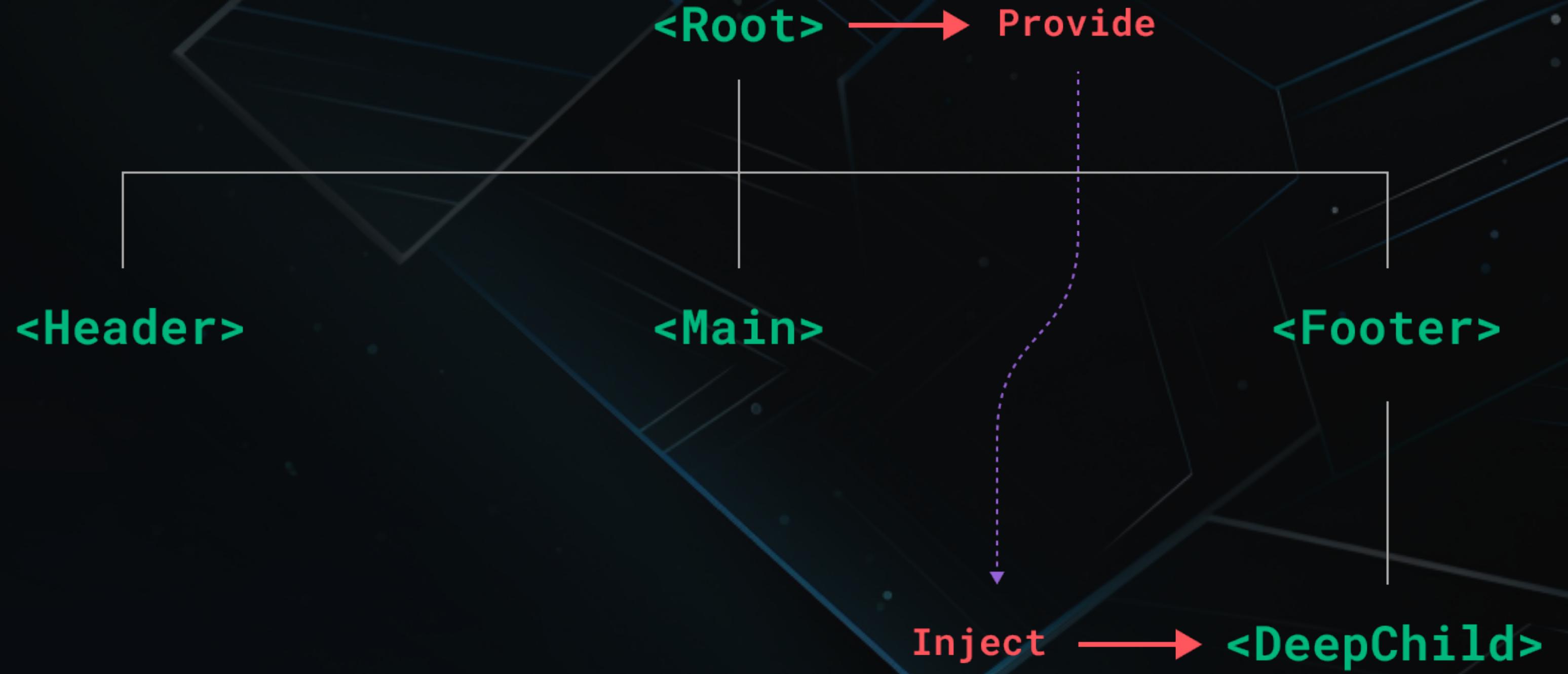
How can we avoid prop drilling?



TECHNIQUE

Provide/Inject

<https://vuejs.org/guide/components/provide-inject.html>





```
// Grandparent Component
<script setup>
import { ref, provide } from 'vue'

const count = ref(0)

provide('count', count)
</script>
```

```
<template>
  <ParentComponent />
</template>
```

```
// Parent Component
<template>
  <ChildComponent />
</template>
```

```
// Child Component
<script setup>
import { inject } from 'vue'

const message = inject('count')
</script>

<template>
  <p>{{ count }}</p>
</template>
```

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.

PROBLEM

How to share the functionality
across different components?

TECHNIQUE

Mixins

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

Mixin (as an Object)

```
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.
- Unclear source of truth
- Can't share template fragments
- Increased risk of unintended side effects

Should you never use mixins?

No.

Pros

- Relatively easy to use
- Good for refactoring

PROBLEM

How do we share code across
components in a better way?



NEW PARADIGM

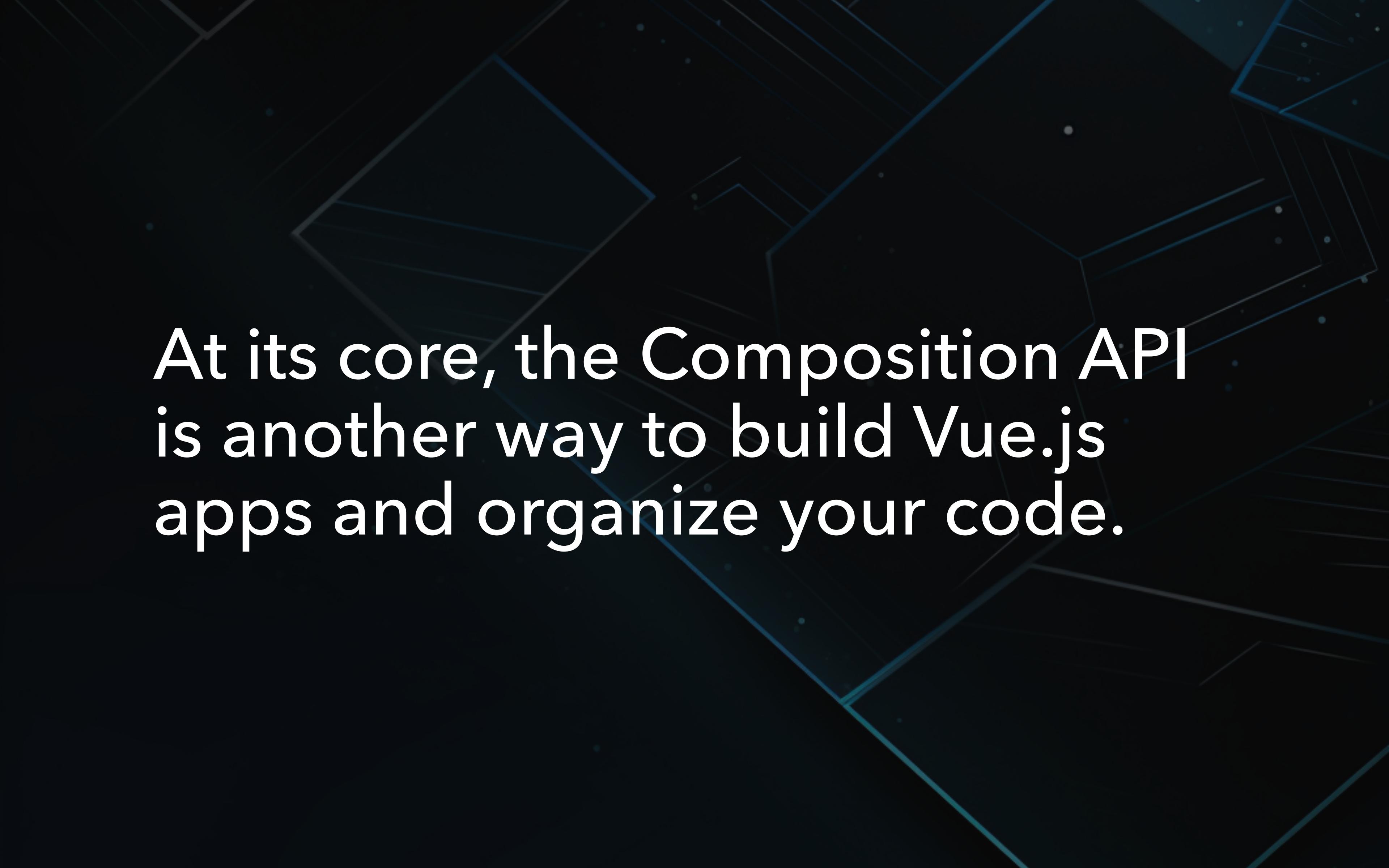
Composition API

Vue 3.0 is here!

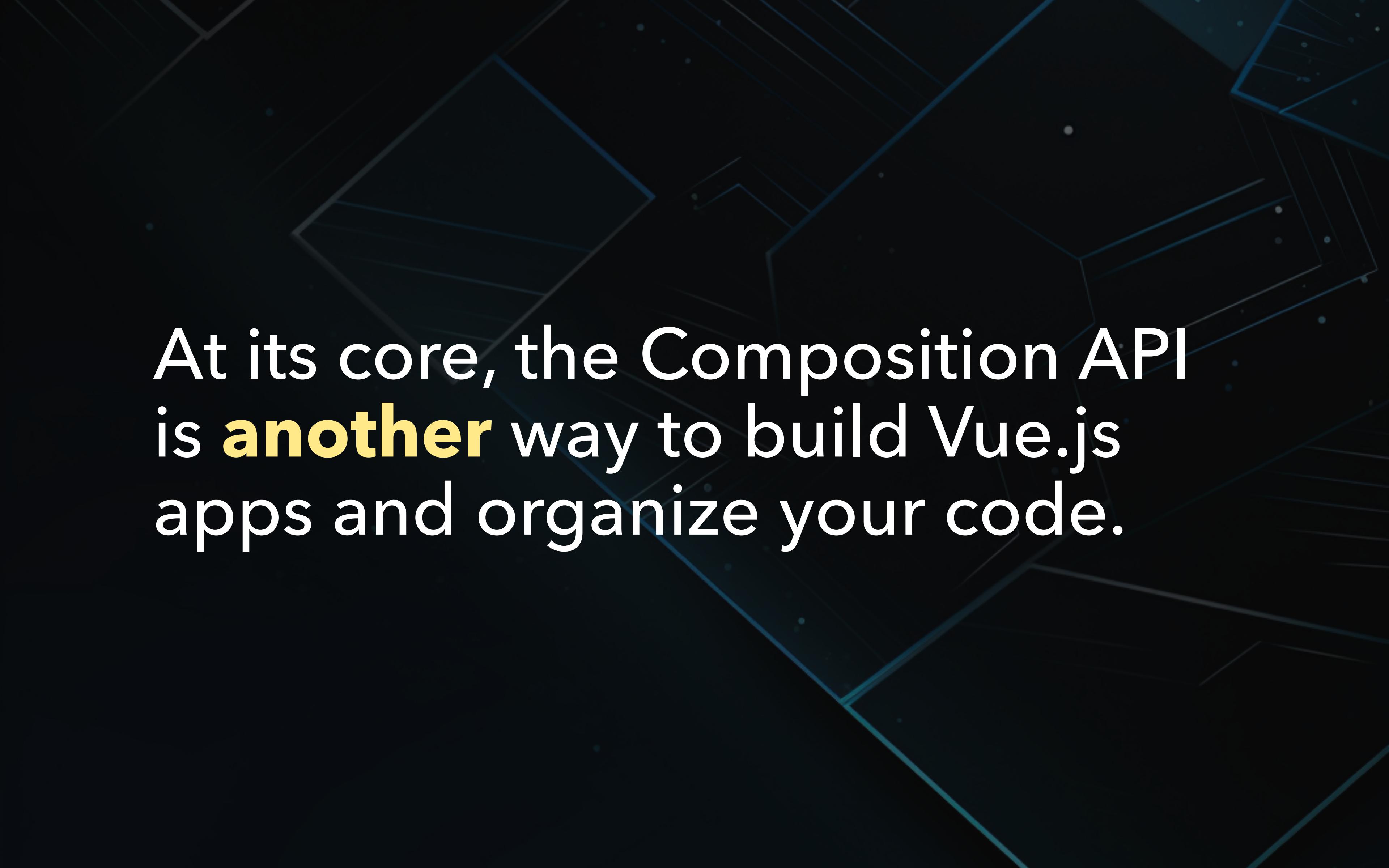
Also available in Vue 2!

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

What is the Composition API?

The background features a dark blue hexagonal grid pattern with thin white lines connecting the centers of the hexagons. Some lines extend beyond the grid boundaries. Small white dots are scattered across the dark background.

At its core, the Composition API
is another way to build Vue.js
apps and organize your code.



At its core, the Composition API
is **another** way to build Vue.js
apps and organize your code.



The Composition API
does not replace
the Options API.

Let's ReVue

**The Options API
Composition API
(Side by Side)**



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const doubleCount = computed(() => {
      return count.value * 2
    })

    const incrementCount = () => {
      count.value += 1
    }

    return {
      count,
      doubleCount,
      incrementCount
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const doubleCount = computed(() => {
      return count.value * 2
    })

    const incrementCount = () => {
      count.value += 1
    }

    return {
      count,
      doubleCount,
      incrementCount
    }
  }
}
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const doubleCount = computed(() => {
      return count.value * 2
    })

    const incrementCount = () => {
      count.value += 1
    }

    return {
      count,
      doubleCount,
      incrementCount
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  }
}.
```

```
methods: {
  incrementCount() {
    this.count++
  }
}
```

```
}
```

```
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const doubleCount = computed(() => {
      return count.value * 2
    })
  }
}
```

```
const incrementCount = () => {
  count.value += 1
}
```

```
return {
  count,
  doubleCount,
  incrementCount
}
```

```
}
```

```
}
```

```
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const doubleCount = computed(() => {
      return count.value * 2
    })

    const incrementCount = () => {
      count.value += 1
    }

    return {
      count,
      doubleCount,
      incrementCount
    }
  }
}
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-option-api.vue

```
<script>
export default {
  data: () => ({
    count: 0
  }),
  computed: {
    doubleCount() {
      return this.count * 2
    }
  },
  methods: {
    incrementCount() {
      this.count++
    }
  }
}
</script>
```

```
<template>
  <p>{{ count }}</p>
</template>
```



simple-composition-api.vue

```
from 'vue'

computed(() => {
  * 2
  = () => {
```

```
<template>
  <p>{{ count }}</p>
</template>
```

Reactive Data in Composition API

ref

● ● ● reactive-in-composition-api.vue

```
<script>
export default {
  setup() {
    const count = 0

    return {
      count
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = 0

    return {
      count
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    return {
      count
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    ★ const count = ref(0)

    return {
      count
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {

    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {
      count += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {
      count += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {
      count += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {
      this.count += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {
      this.count += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    const incrementCount = () => {
      count += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```



reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = ref(0)

    ★ const incrementCount = () => {
      count.value += 1
    }

    return {
      count
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

reactive

● ● ● reactive-in-composition-api.vue

```
<script>
export default {
  setup() {
    const count = 0

    return {
      count
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = 0

    return {
      count
    }
  }
}
</script>

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = 0

    const state = reactive({})

    return {
      count
    }
  }
}
</script>
<p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = 0

    const state = reactive({})

    return {
      count,
      state
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```



reactive-in-composition-api.vue

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

export default {
  setup() {
    const count = 0

    const state = reactive({})

    return {
      count,
      state
    }
  }
}

<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      count,
      state
    }
  }
}

</script>
<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      state
    }
  }
}
</script>
<template>
  <p>{{ count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      state
    }
  }
}
</script>
<template>
  <p>{{ state.count }}</p>
</template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    const incrementCount = () => {
      state.count += 1
    }

    return {
      state,
      incrementCount
    }
  }
}
</script>
<template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    const incrementCount = () => {
      count += 1
    }

    return {
      state
    }
  }
}
</script>
<template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    const incrementCount = () => {
      state.count += 1
    }

    return {
      state
    }
  }
}
</script>
<template>
```

● ● ● reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      state
    }
  }
}
</script>
<template>
  <p>{{ state.count }}</p>
</template>
```

ref vs reactive

toRefs



reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      state
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      ...state
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      ...state
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })
    return {
      ...state
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

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

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      ...state
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

```
<script>
import { reactive, toRefs } from 'vue'

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      ...state
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

```
<script>
import { reactive, toRefs } from 'vue'

export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      ...toRefs(state)
    }
  }
}
</script>

<template>
  <p>{{ state.count }}</p>
</template>
```



reactive-in-composition-api.vue

```
<script>
import { reactive, toRefs } from 'vue'

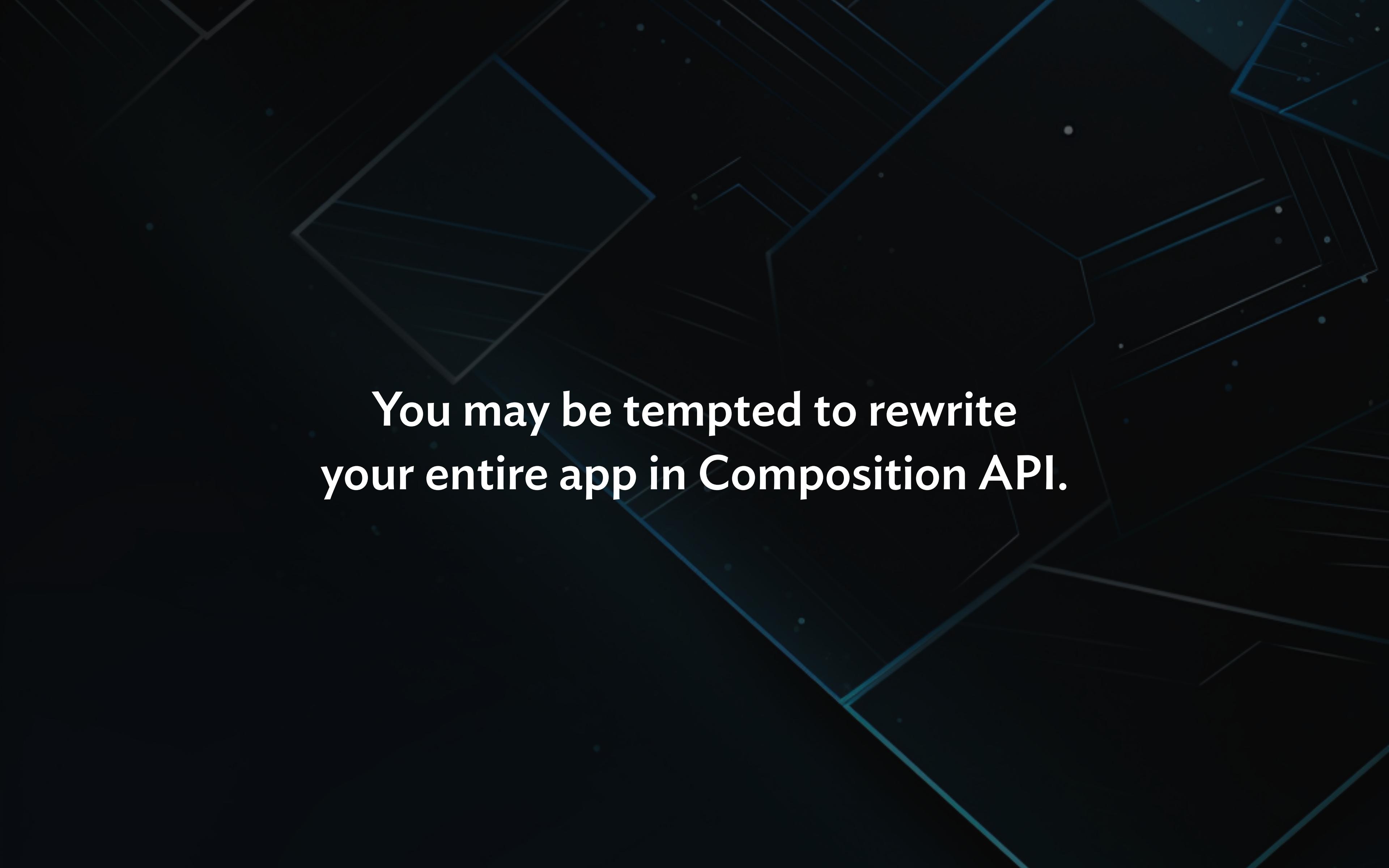
export default {
  setup() {
    const state = reactive({
      count: 0
    })

    return {
      ...toRefs(state) ★
    }
  }
}
</script>

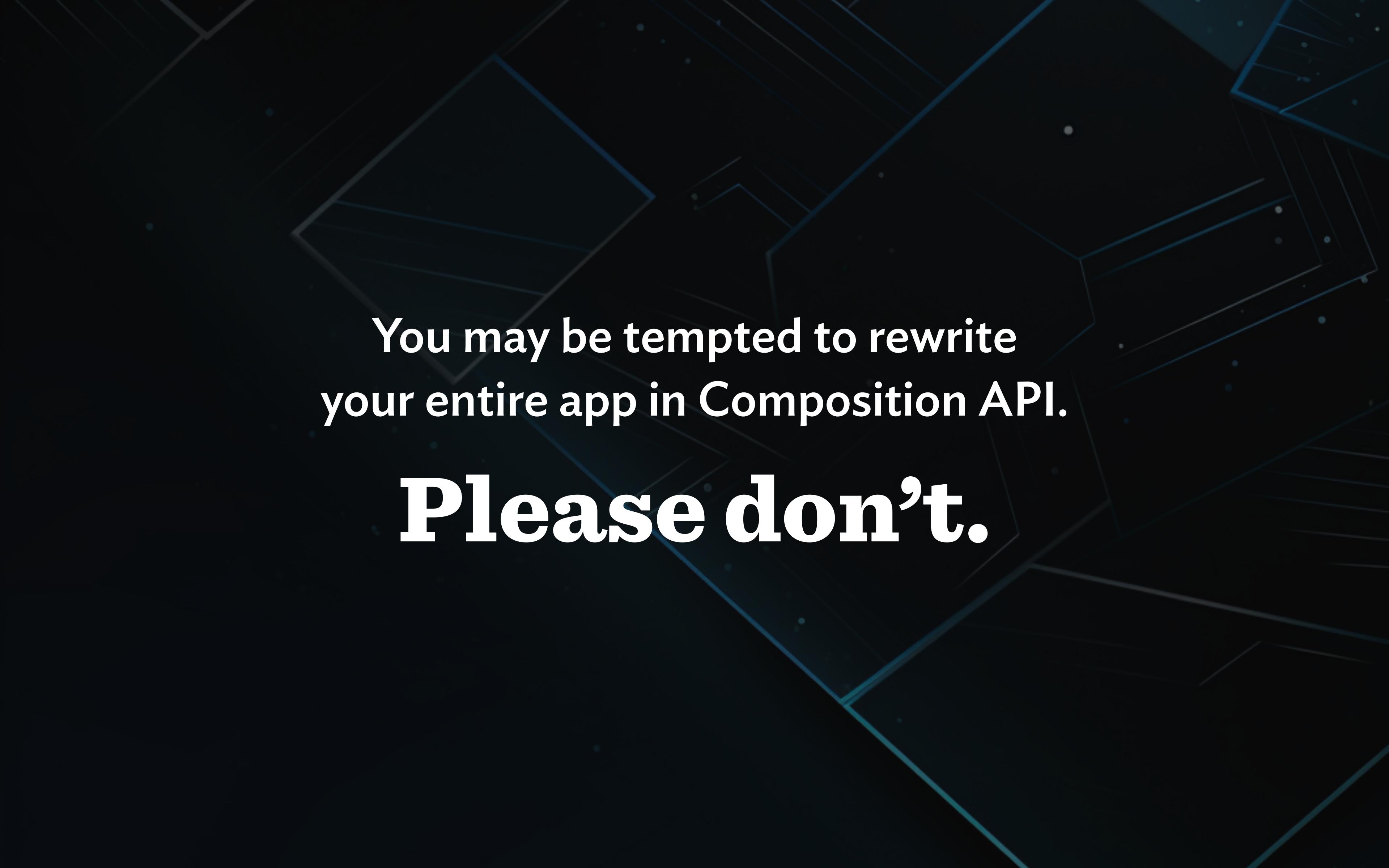
<template>
  <p>{{ state.count }}</p>
</template>
```

Trade-offs

- Can be more verbose and have less structure compared to Options API
- More JavaScript knowledge is needed
- Deeper understanding of Vue.js is required to use it
- Architectural decisions are up to you

The background of the slide features a dark, abstract geometric design. It consists of numerous thin, light-blue lines of varying lengths that intersect to form a complex network of shapes. Small, semi-transparent blue dots are scattered across the surface, particularly concentrated along the lines and at the vertices of the geometric figures. The overall effect is one of depth and a digital or futuristic aesthetic.

You may be tempted to rewrite
your entire app in Composition API.



You may be tempted to rewrite
your entire app in Composition API.
Please don't.

TECHNIQUE

Composables

<https://vuejs.org/guide/reusability/composables.html>

TECHNIQUE

defineModel()

TECHNIQUE

toValue() and toRef()



CODE / EXPERIMENT



In the playground

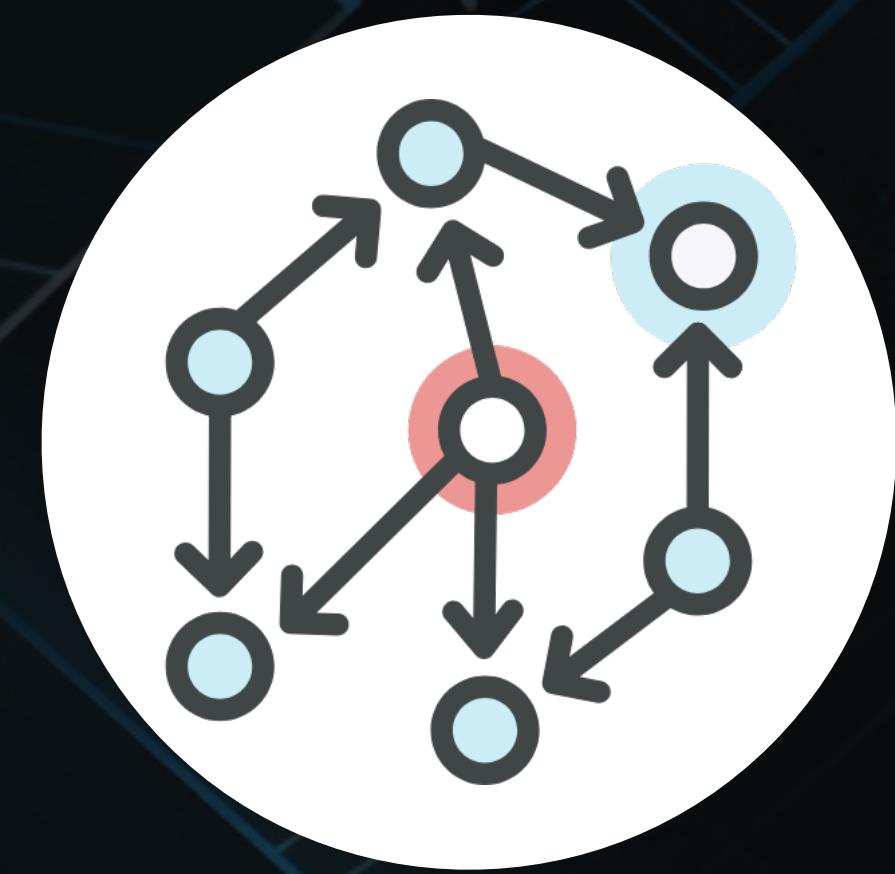
- Practice using Composition API in a component
- Use composables to organize code between components
- Try the new defineModel macro

In your app

- Identify opportunities for Composition API
- Try refactoring code using Composition API



Questions?



STATE MANAGEMENT

BEST PRACTICES

Vuex & Pinia

BEST PRACTICES

Vuex & Pinia

What data to put into Vuex & Pinia?

- 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 & Pinia

What data to put into Vuex & Pinia?

- 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 & Pinia

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 & Pinia

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 Vuex 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?

DISCUSSION

With the Composition API, do we even need Vuex?

Yes

TOOLING

Pinia



CODE / EXPERIMENT



In the playground

- Create a new Pinia store using Options or Composition style stores

In your app

- Identify opportunities to improve state management



TESTING

PRINCIPLE

The Pareto Principle



PRINCIPLE

The Pareto Principle

The 80-20 Rule

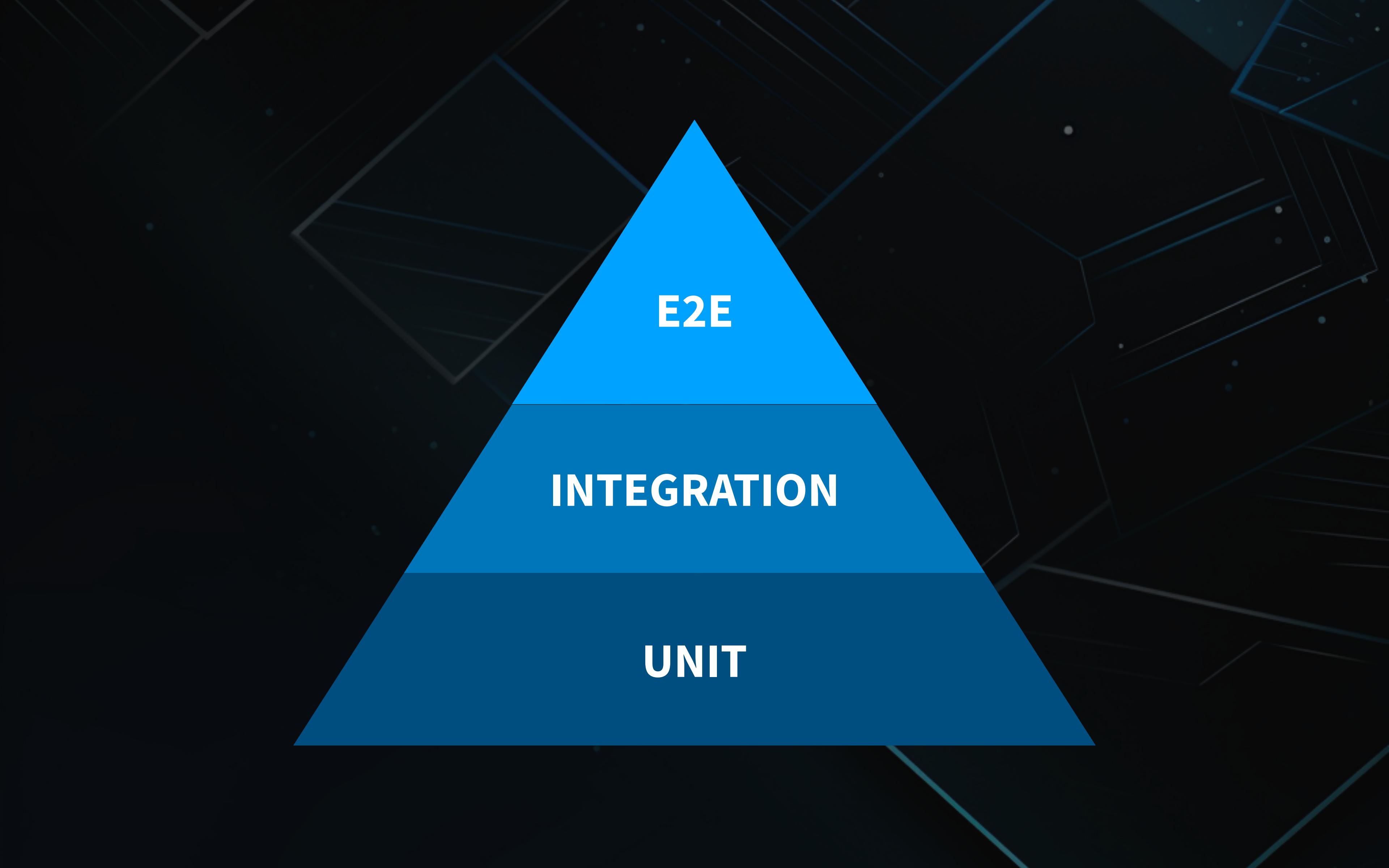


20% EFFORT



BEST PRACTICES

Testing



E2E

INTEGRATION

UNIT



The background features a dark blue hexagonal grid pattern. Overlaid on this are several large, semi-transparent blue hexagons of varying sizes, some rotated diagonally. A network of thin, light blue lines connects the vertices of these hexagons, creating a complex web-like structure. Small, scattered white dots are also present throughout the background.
UNIT

UNIT



Vitest

UNIT

What about Vue Test Utils?

COMPONENT

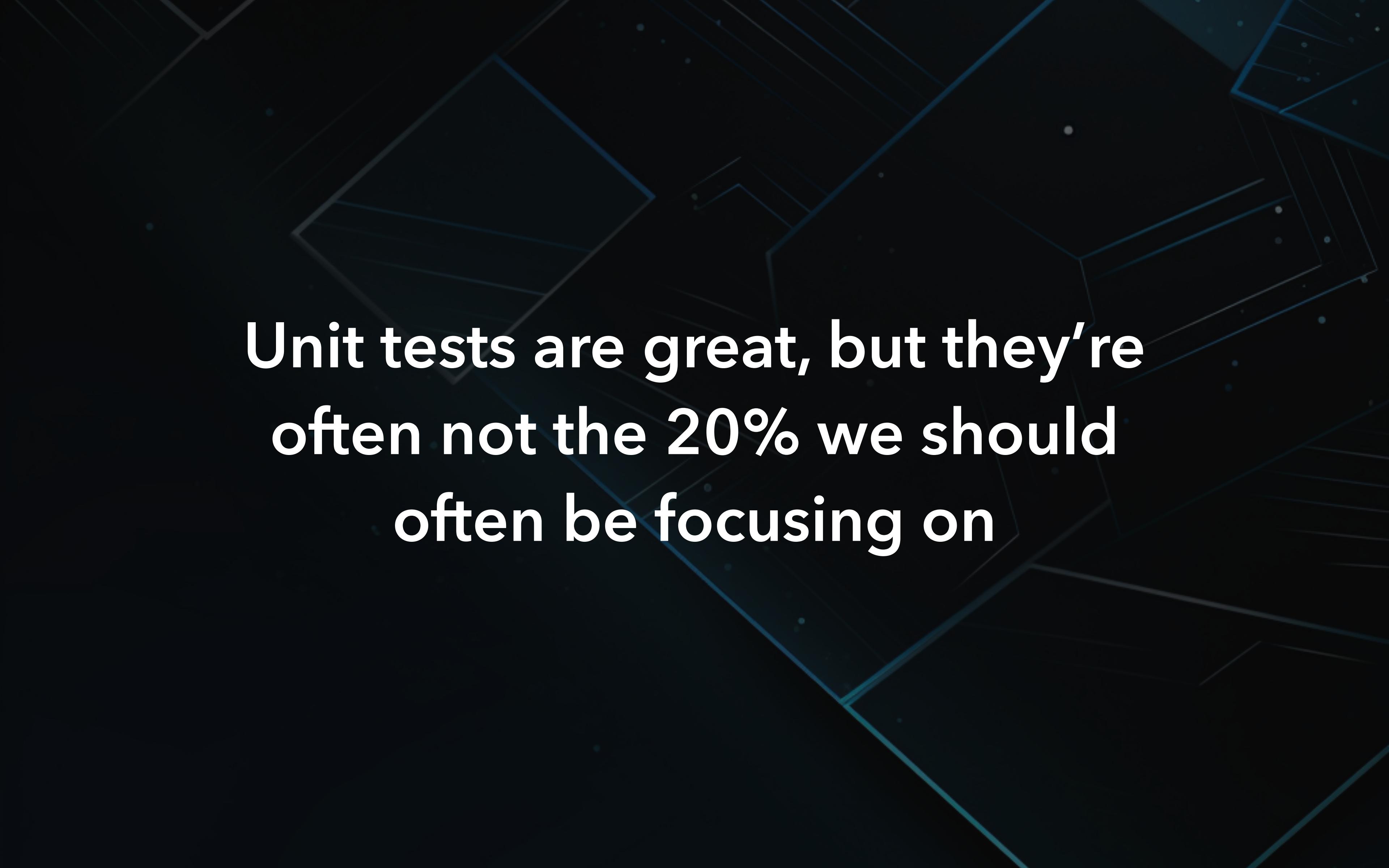


Cypress

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**



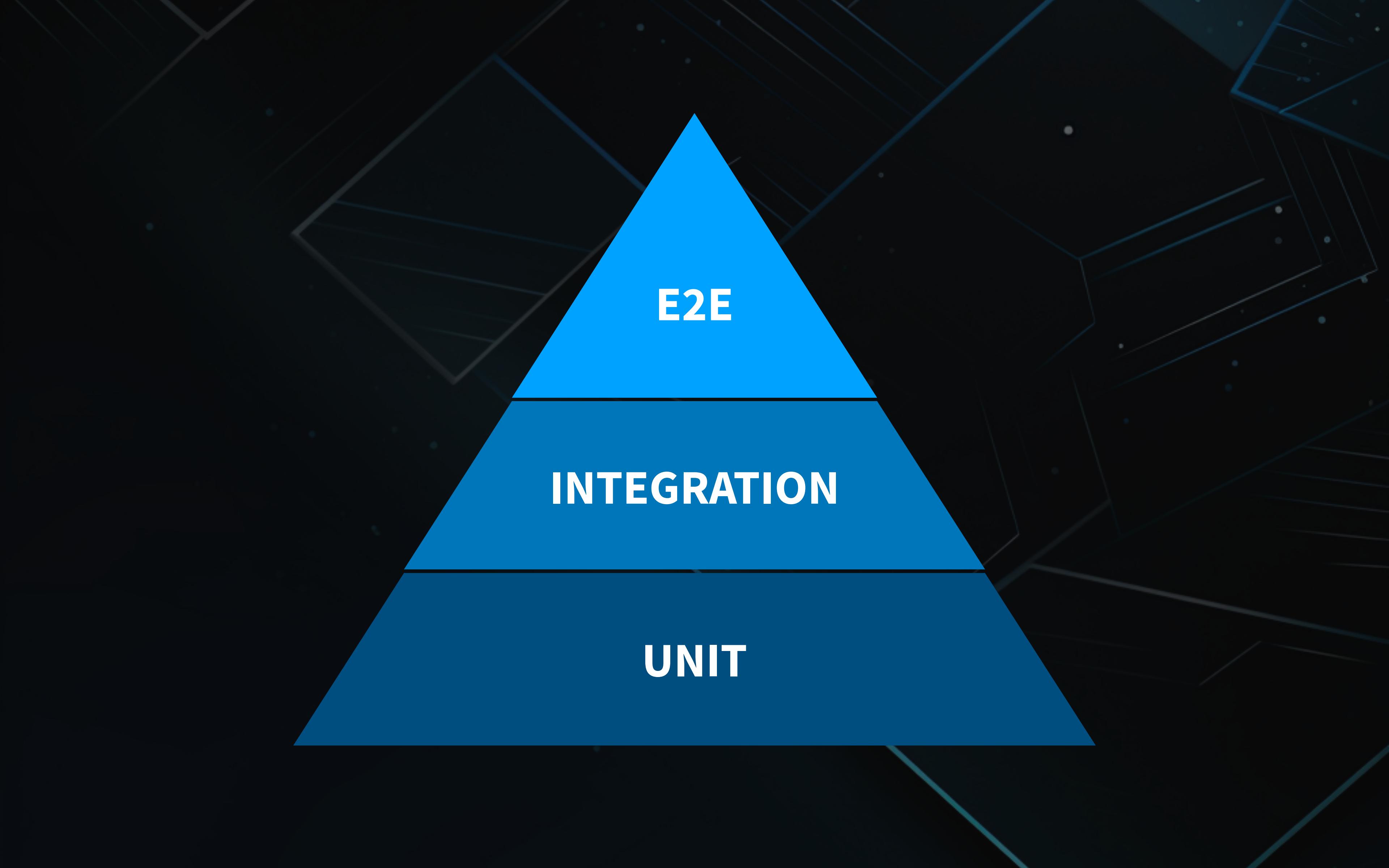
Unit tests are great, but they're
often not the 20% we should
often be focusing on

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?

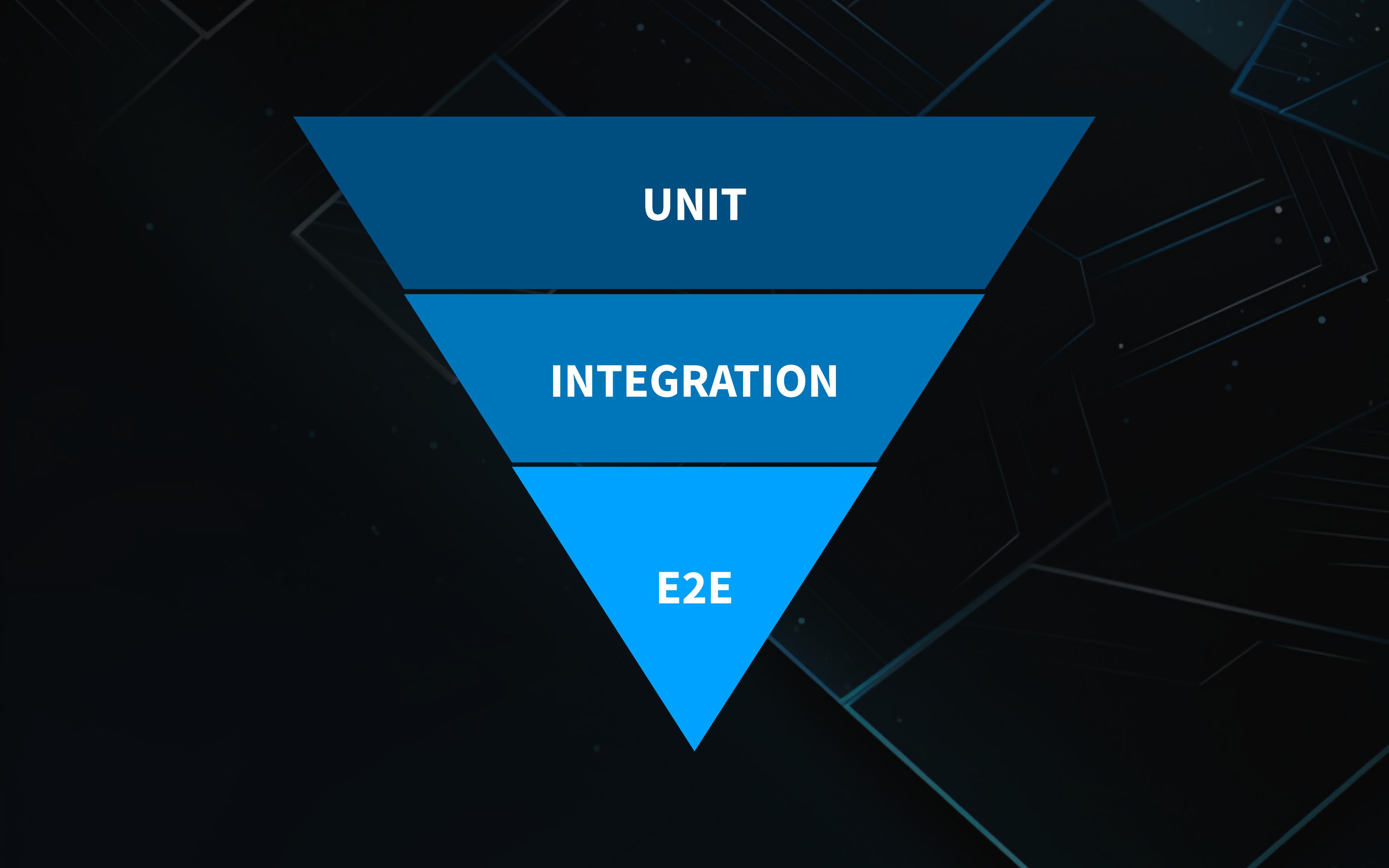
#2. Can the user pay us?



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

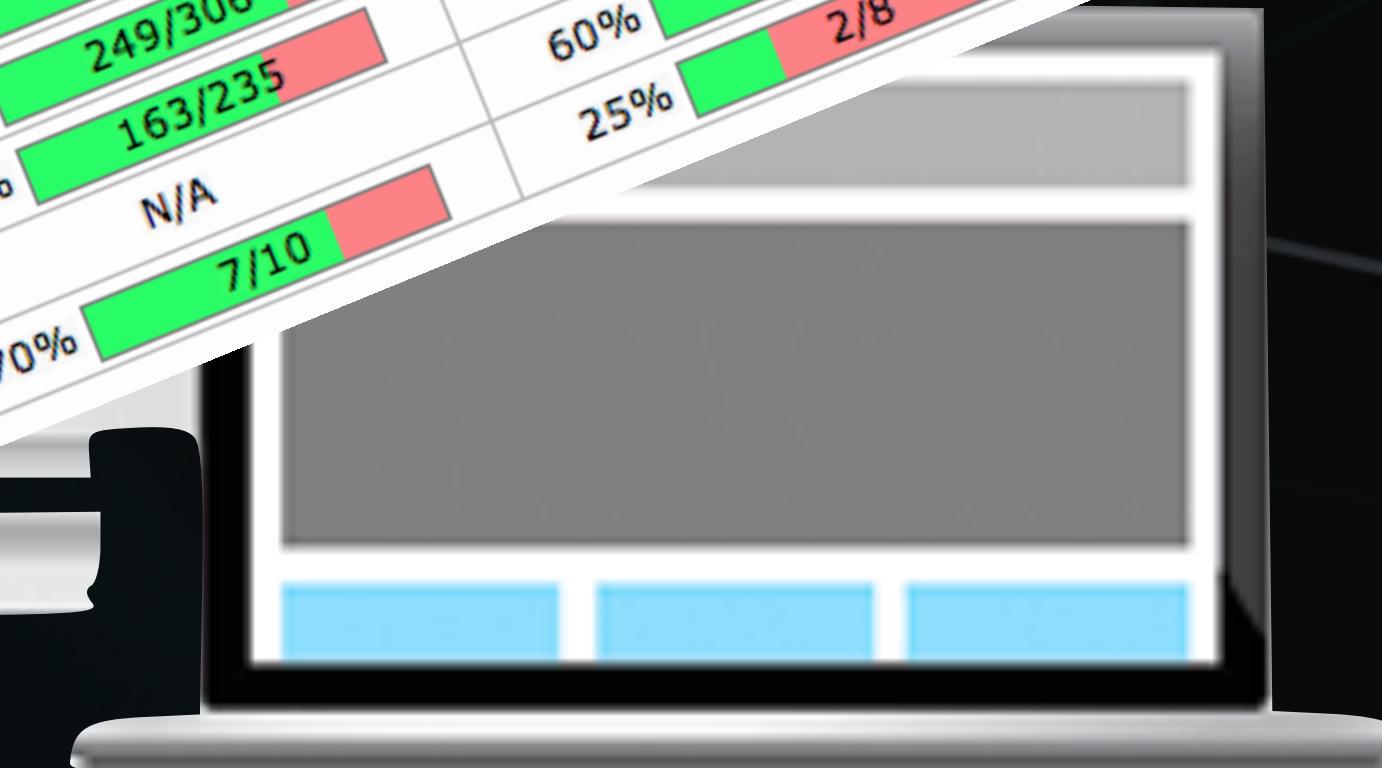


E2E



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
Stop-tabac	0% 0/1	47% 8/17	0% 0/1	0% 0/5	
Stop-tabac.Classes	100% 1/1	73% 22/30	19% 1/1	27% 58/213	
Stop-tabac.Classes.Controller	19% 3/16	63% 20/32	100% 3/16	5% 120/2665	
Stop-tabac.Classes.Manager	100% 3/3	81% 249/306	100% 3/3	65% 103/158	
Stop-tabac.Classes.Model	75% 6/8	69% 163/235	75% 6/8	65% 669/869	
Stop-tabac.Classes.Service	56% 5/9	N/A	56% 5/9	77% 388/830	
Stop-tabac.Classes.Utils	60% 3/5	70% 7/10	60% 3/5	47% 74/126	
Stop-tabac.Classes.View	25% 2/8		25% 2/8	59% 38/356	



E2E



Project Coverage summary

Name	Classes	Conditionals
Cobertura Coverage Report	45% 23/51	74% 1450/5222

Coverage Breakdown by Package

Name	Classes
Stop-tabac	0% 0/1
Stop-tabac.Classes	100% 1/1
Stop-tabac.Classes.Controller	19% 3/16
Stop-tabac.Classes.Manager	100% 3/3
Stop-tabac.Classes.Model	75% 6/8
Stop-tabac.Classes.Service	56% 3/5
Stop-tabac.Classes.Utils	60% 2/8
Stop-tabac.Classes.View	25% 2/8

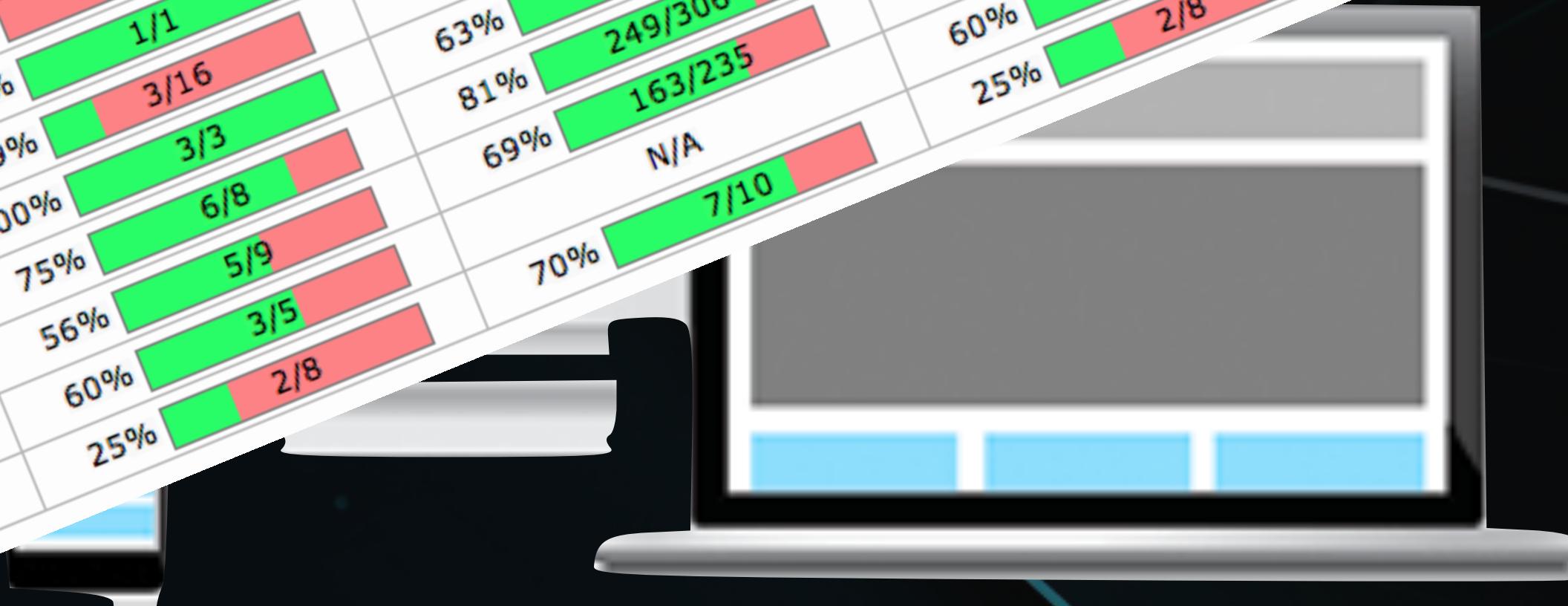


E2E



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
Stop-tabac	0% 0/1	N/A	0% 0/1	0% 0/5	0% 0/5
Stop-tabac.Classes	100% 1/1	47% 8/17	100% 1/1	27% 58/213	100% 1/1
Stop-tabac.Classes.Controller	19% 3/16	73% 22/30	19% 3/16	5% 120/2665	63% 3/3
Stop-tabac.Classes.Manager	100% 3/3	63% 20/32	100% 3/3	65% 103/158	100% 3/3
Stop-tabac.Classes.Model	75% 6/8	81% 249/306	75% 6/8	77% 669/869	56% 5/9
Stop-tabac.Classes.Service	56% 5/9	69% 163/235	60% 3/5	47% 388/830	60% 3/5
Stop-tabac.Classes.Utils	60% 3/5	N/A	25% 2/8	59% 74/126	25% 2/8
Stop-tabac.Classes.View	25% 2/8	70% 7/10	70% 7/10	11% 38/356	70% 7/10



E2E





E2E



E2E



E2E



Playwright

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?



CODE / EXPERIMENT



In the playground

- Write a unit test using Vitest and Vue Test Utils
- Write an e2e test using Playwright

In your app

- Identify opportunities for how to enhance your testing strategy



**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

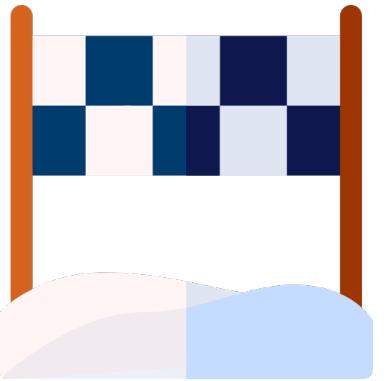
- **The Jidoka Principle**
 - Discover an abnormality
 - Stop the process
 - Fix the immediate process
 - Investigate and solve the root cause
- **The Andon Cord** - Empower your team to speak up



Questions?



Open Practice + Q&A



Closing Thoughts

Content

-  Languages
-  Components
-  Routing
-  State Management
-  Reusability
-  Testing
-  Fostering Change

RESOURCES

Next steps...

Vue Enterprise Boilerplate

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search Sign in Sign up

chrisvfritz / vue-enterprise-boilerplate

Code Issues 22 Pull requests 3 Actions Projects Security Insights

master 2 branches 0 tags Go to file Code

frandiox and chrisvfritz Add explanation about pre-commit hook (#196) ... 890438c on Mar 19 254 commits

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 autocomplete	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

About

An ever-evolving, very opinionated architecture and dev environment for new Vue SPA projects using Vue CLI.

vue javascript frontend
boilerplate

Readme MIT License

Releases

No releases published

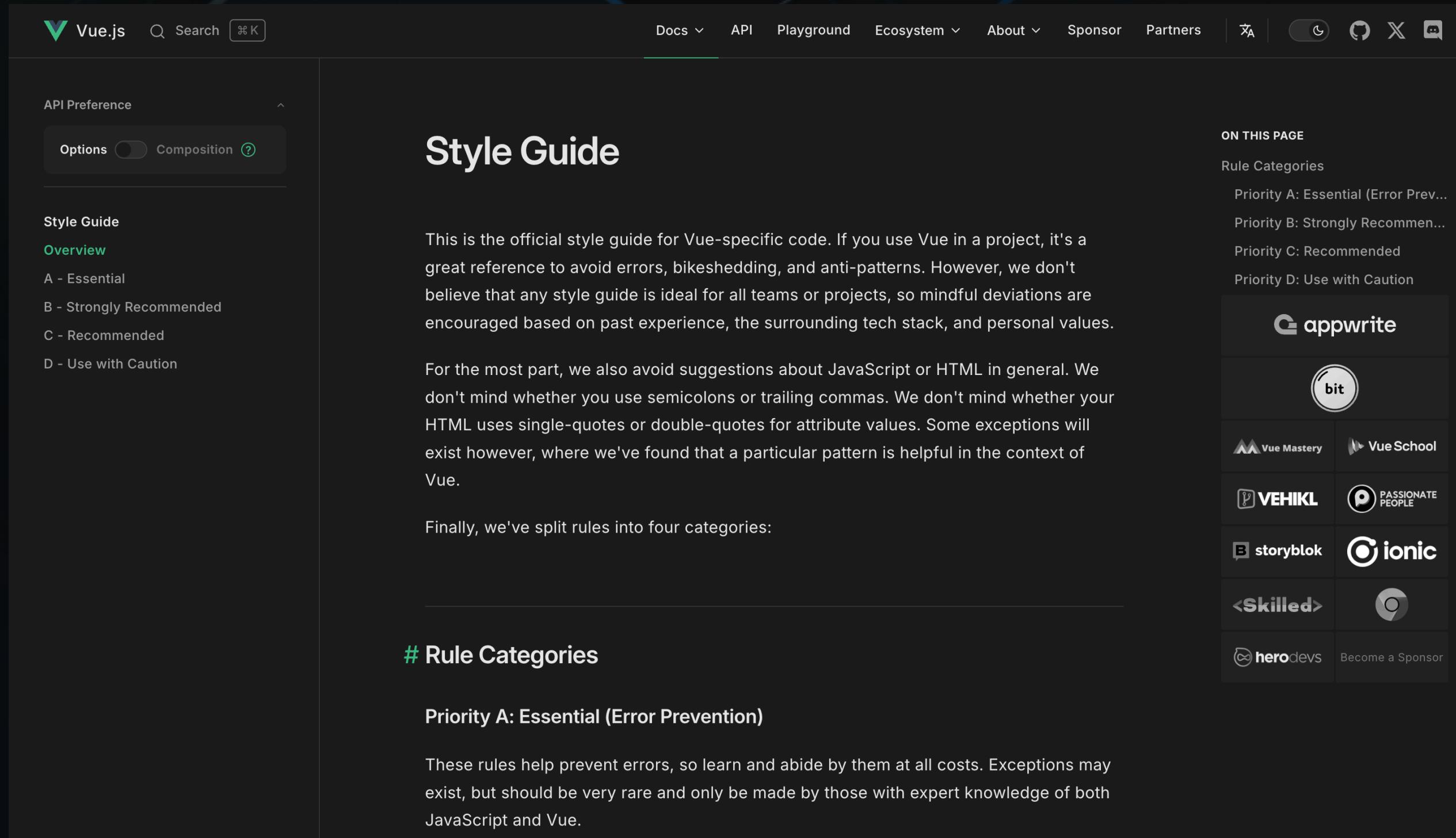
Packages

No packages published

Contributors 31

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

Vue.js Style Guide



The screenshot shows the official Vue.js Style Guide page. At the top, there's a navigation bar with links for Docs, API, Playground, Ecosystem, About, Sponsor, and Partners. Below the navigation is a search bar and some UI controls. The main content area has a dark background with white text. On the left, there's a sidebar titled "API Preference" with a "Style Guide" section containing categories A, B, C, and D. The main content starts with a large heading "Style Guide". It explains the purpose of the style guide and categorizes rules into four priority levels. It also mentions avoiding suggestions about JavaScript or HTML in general. Below this, there's a section for "Rule Categories" with a heading "# Rule Categories" and a link to "Priority A: Essential (Error Prevention)". The right side of the page features a sidebar titled "ON THIS PAGE" with sections for Rule Categories (Priority A-D) and a "Sponsors" section listing various companies like appwrite, bit, Vue Mastery, Vue School, VEHIKL, PASSIONATE PEOPLE, storyblok, ionic, <Skilled>, and hero.devs, along with a "Become a Sponsor" button.

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.

Finally, we've split rules into four categories:

Rule Categories

Priority A: Essential (Error Prevention)

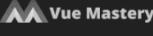
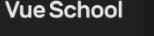
These rules help prevent errors, so learn and abide by them at all costs. Exceptions may exist, but should be very rare and only be made by those with expert knowledge of both JavaScript and Vue.

ON THIS PAGE

Rule Categories

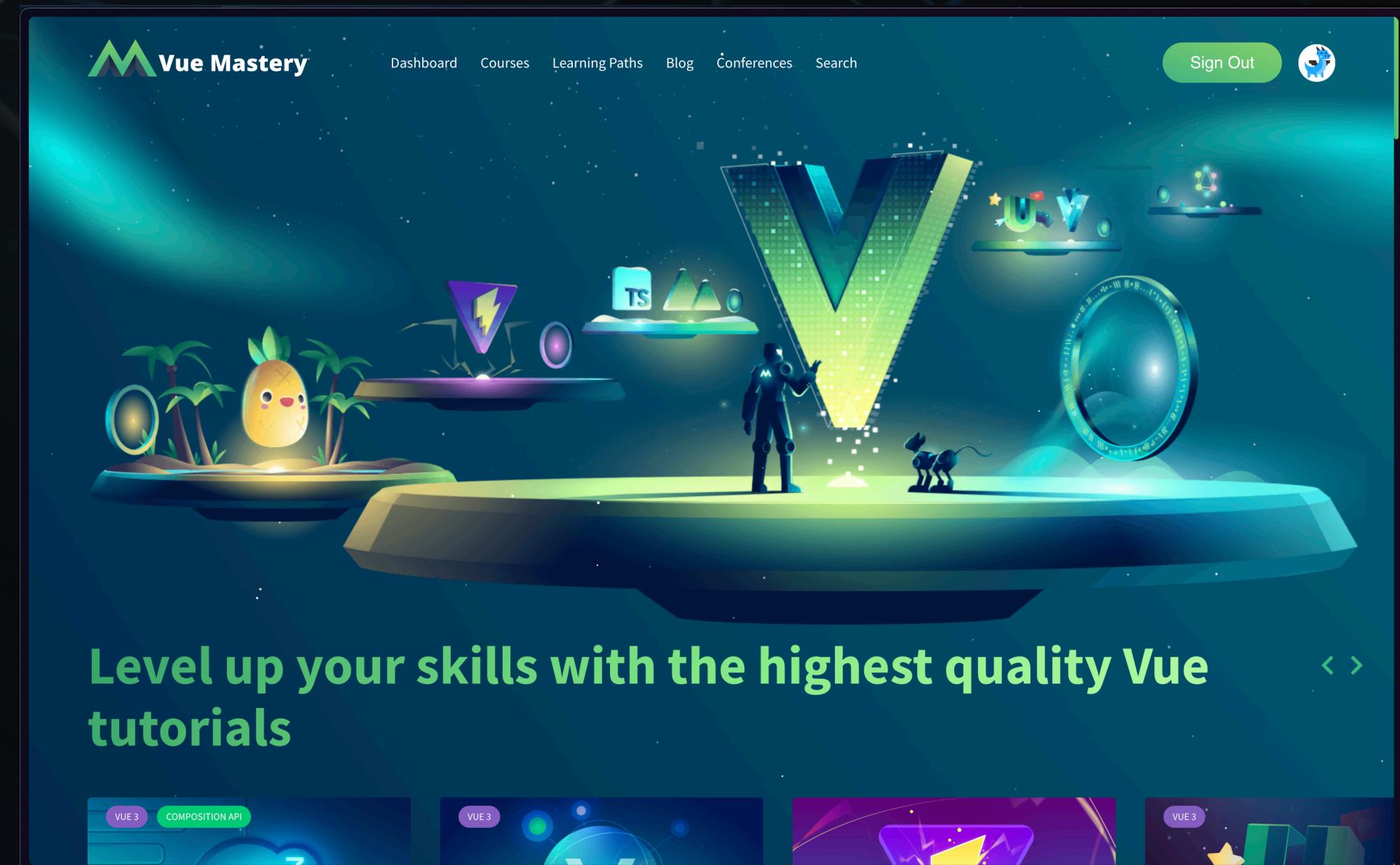
- Priority A: Essential (Error Prev...)
- Priority B: Strongly Recommend...
- Priority C: Recommended
- Priority D: Use with Caution

Sponsors

 appwrite	
 Vue Mastery	 Vue School
 VEHIKL	 PASSIONATE PEOPLE
 storyblok	 ionic
 <Skilled>	
 hero.devs	Become a Sponsor

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

Vue Mastery



<https://www.vuemastery.com/>

Frontend Masters

The screenshot shows the homepage of the Frontend Masters website. At the top, there's a dark header bar with the "Frontend Masters" logo on the left and navigation links for "Courses", "Learn", "Pricing", "Login", and a red "Join Now" button on the right. Below the header is a large, semi-transparent image of a person working at a desk with multiple monitors displaying code and web development tools like Studio and Frontend Masters. Overlaid on this image is the main headline: "Advance Your Skills with In-Depth, Modern Front-End Engineering Courses". Below the headline are two buttons: a red one labeled "Browse Our Courses" and an orange one labeled "View Learning Paths". At the bottom of the page, there's a blue footer bar containing several technology logos: JS (yellow square), React (atom icon), TS (blue square), node.js (green V), Vue.js (teal V), Angular (red A), CSS (blue E), MongoDB (orange B), and Docker (blue cube).

Frontend **Masters**

Courses Learn Pricing Login Join Now

Advance Your Skills
with In-Depth, Modern
Front-End Engineering Courses

Browse Our Courses View Learning Paths

JS React TS node.js Vue.js Angular CSS MongoDB Docker

What They're Saying About Us (@FrontendMasters)

<https://frontendmasters.com/>

Mastering Pinia



The screenshot shows the landing page for the "Mastering Pinia" course. At the top, there's a navigation bar with links for Pricing, Course Outline, Blog, Live Talk, Get Course Preview, Buy Now (in a yellow button), and Sign in. The main title "The complete guide to Mastering Pinia" is prominently displayed in white and yellow text. Below the title, a subtitle reads "Become an expert in the official state management solution for Vue.js". There are two buttons at the bottom left: "Get Course Preview" and "Buy Now >". A small yellow graduation cap emoji is positioned next to the "Buy Now" button. At the bottom center, it says "Course created by the **author of Pinia**, powered by **Vue School**". To the right of the text, there's a cartoon illustration of a yellow emoji face wearing a black graduation cap and dark sunglasses, with green leaves on its head.

Mastering Pinia

Pricing Course Outline Blog Live Talk Get Course Preview Buy Now Sign in

The complete guide to Mastering Pinia

Become an expert in the official state management solution for Vue.js

Get Course Preview Buy Now >

Course created by the **author of Pinia**, powered by **Vue School**.

<https://masteringpinia.com/>

Michael Thiessen

 Michael Thiessen

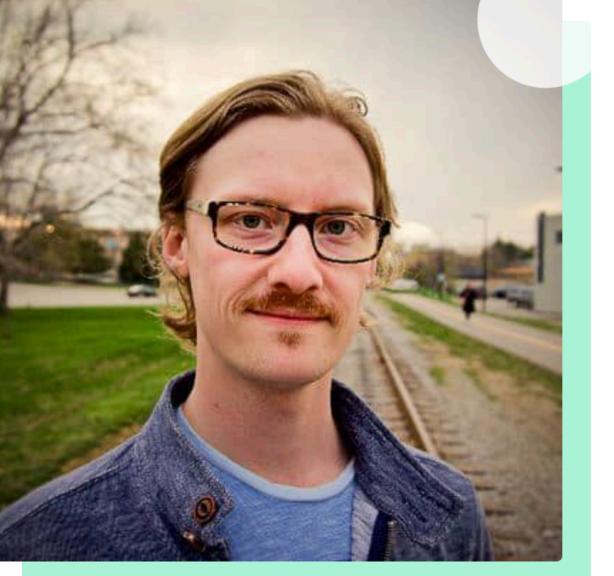
Home Articles [Courses](#)

Want to level up your Vue skills?

With over two million reads and 11,067 subscribers, you've come to the right place.

Subscribe now to get exclusive insights and tips every week.

[Subscribe](#)

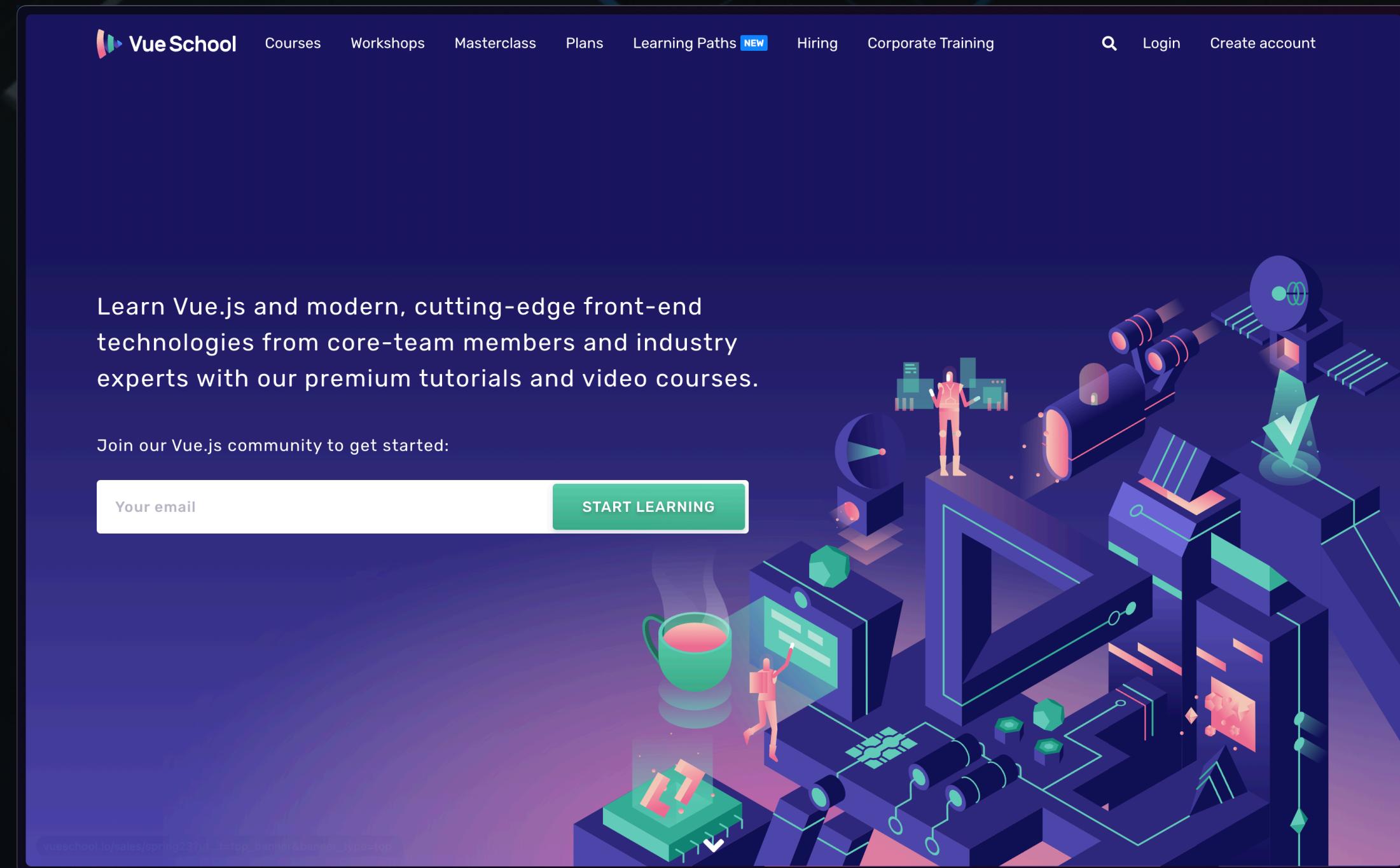


👋 Hey there! Welcome to my site. Take a look around and make yourself at home.

— Michael

<https://michaelnthiessen.com/#courses>

Vue School



<https://vueschool.io/>

 Congratulations! 



**YOU OTTER BE PROUD
OF YOURSELF!**

Thanks everyone!

Please fill out the feedback survey:

[https://form.typeform.com/to/
C2PTiSLE](https://form.typeform.com/to/C2PTiSLE)



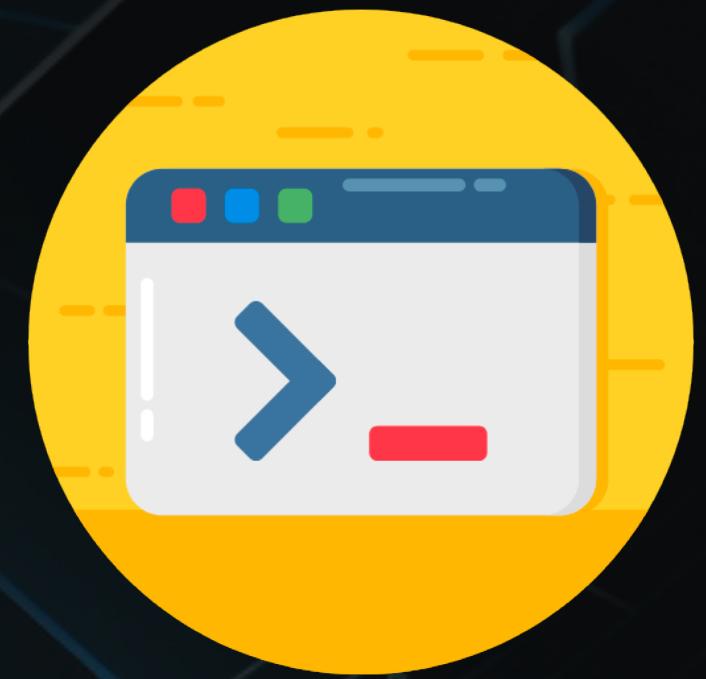


Templates



Additional Content





TOOLING



TOOL

VSCode + Volar



TOOL

VS Code Snippets



TOOL

Vue CLI



TOOL

Vue CLI - GUI Mode

PRO TIP

Vue CLI Modern Mode

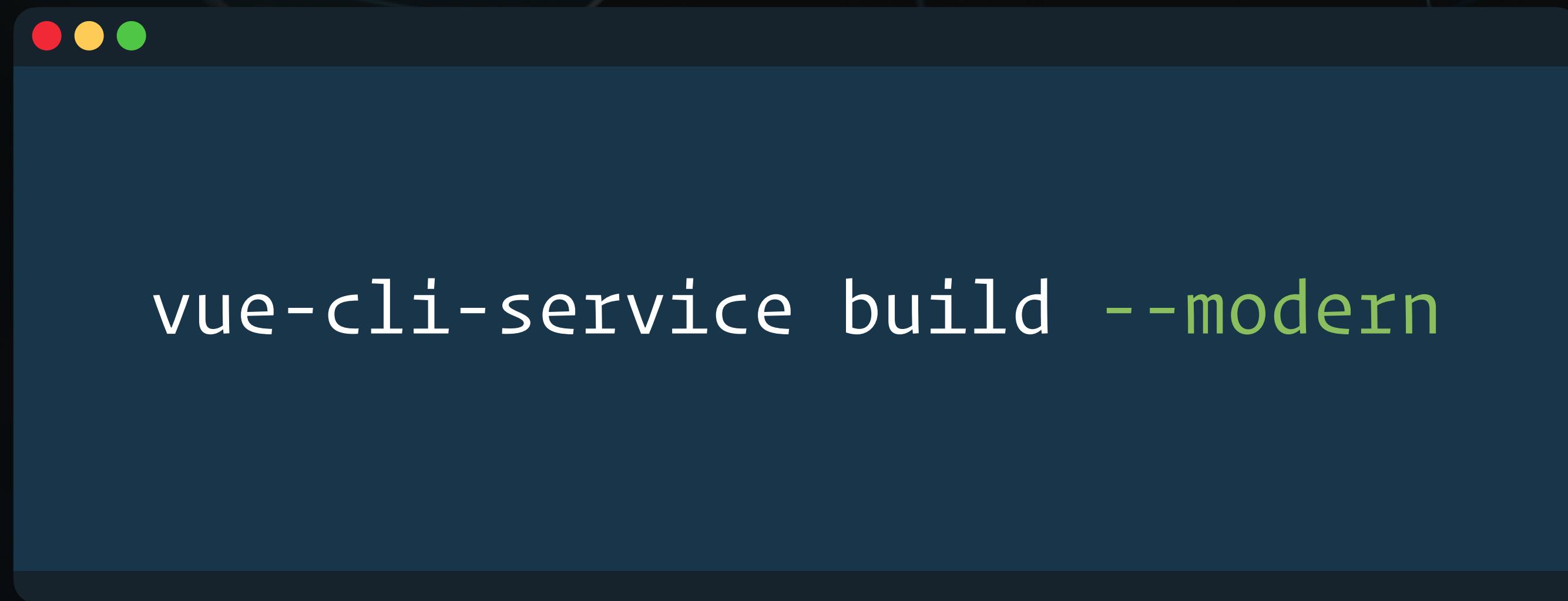
PRO TIP

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



TOOL

create-vue



TOOL

Vue as a CDN



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

TOOL

Nuxt 3



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

TOOL

CapacitorJS



TOOL

Electron



TOOL

Better deployment pipelines



Questions?