Production-Grade Vue

Frontend Masters

Ben Hong
https://www.bencodezen.io


UNIVERSAL'S
ISLANDS OF ADVENTURE
JURASSIC PARK
RIVER ADVENTURE
UNIVERSAL'S
ISLANDS OF ADVENTURE
November 12, 2017
TM & © Universal Studios and Amblin


JURASSIC PARK
RIVER ADVENTURE
ISLANDS OF ADVENTURE
November 12, 2017
TM & © Universal Studios and Amblin

BEN HONG

Vue.js

Docs API Reference Ecosystem Support Vue GitHub Search ⌘ K

Introduction

NOTE

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

What is Vue.js?

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

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

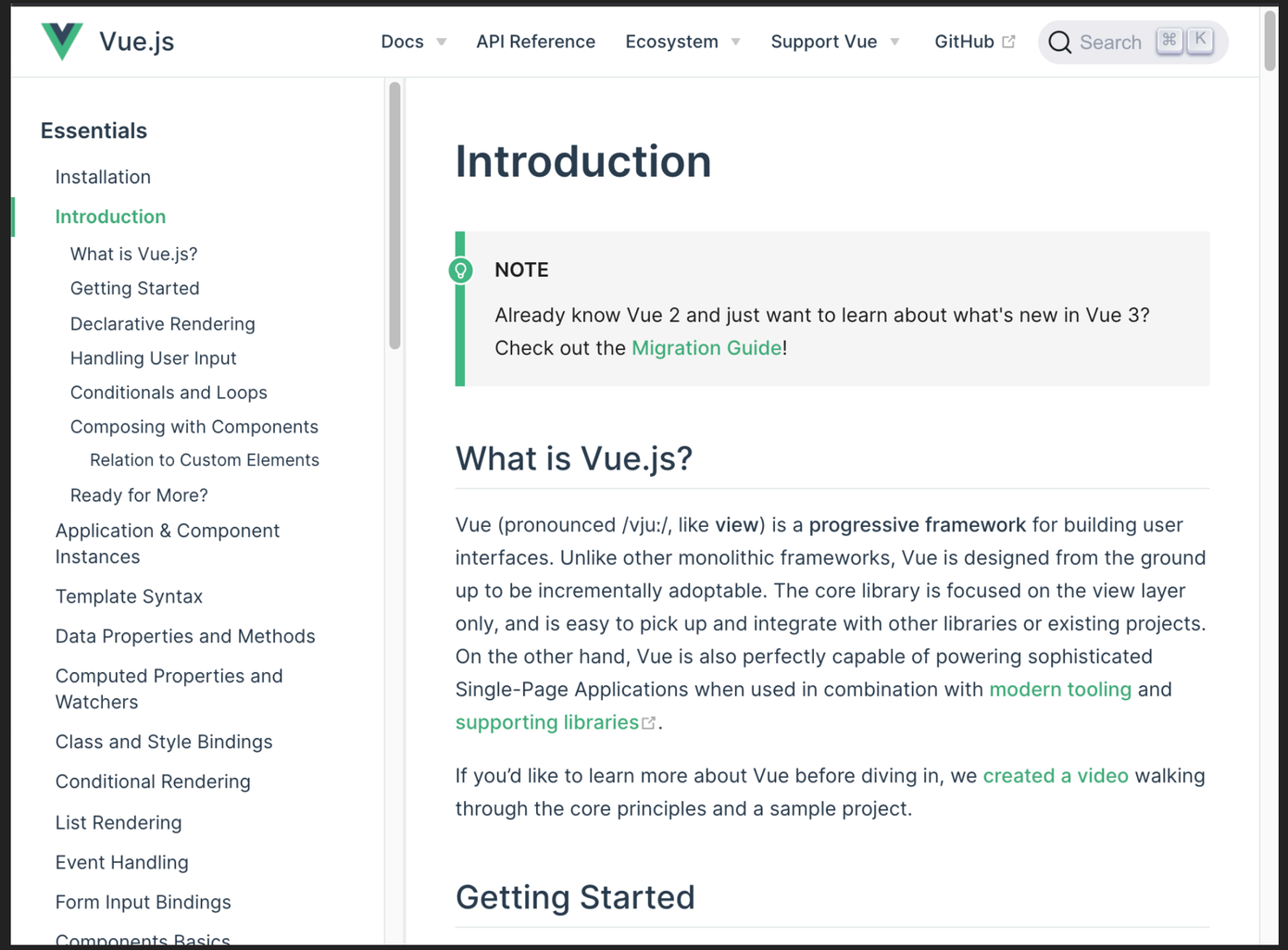
Getting Started

BEN HONG

Senior DX Engineer

Netlify

@bencodezen



Vue.js

Docs API Reference Ecosystem Support Vue GitHub Search ⌘ K

Essentials

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

Introduction

NOTE

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

What is Vue.js?

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

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

Getting Started

📅 SCHEDULE 📅

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

Before we get started...

PARTICIPATION TIPS

Questions are **welcome**!

PARTICIPATION TIPS

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

No need to hurry and copy down examples.

PARTICIPATION TIPS

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

PARTICIPATION TIPS

Most things are applicable to Vue 2 and 3.

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

PARTICIPATION TIPS

All code is **compromise**.

I encourage you to question and/or disagree.

Your opinion and experience matter.

Choose what works best for you and your team.

Questions?

LANGUAGES

TOOL

Single File Components (SFCs)

TOOL

Single File Components (SFCs)

MyFirstComponent.vue

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

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

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

BEST PRACTICES

JavaScript

DISCUSSION

JavaScript vs TypeScript

DISCUSSION

JavaScript vs TypeScript

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

DISCUSSION

JavaScript vs TypeScript

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

Questions?

BEST PRACTICES

HTML

BEST PRACTICE

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

BEST PRACTICE

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

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

BEST PRACTICE

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

- Self-closing tags

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

BEST PRACTICE

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

- Self-closing tags

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

BEST PRACTICE

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

- Self-closing tags

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

BEST PRACTICE

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

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

TECHNIQUE

Template

MyFirstComponent.vue

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

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

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

TECHNIQUE

Render Function

MyFirstComponent.vue

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

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

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

TECHNIQUE

Render Function

MyFirstComponent.vue

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

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

TECHNIQUE

Render Function

MyFirstComponent.vue

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

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

TECHNIQUE

Render Function

MyFirstComponent.vue

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

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

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

Vue.js 3

TECHNIQUE

Render Function

MyFirstComponent.vue

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

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

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

Vue.js 3

DISCUSSION

Template vs Render Function

DISCUSSION

Template vs Render Function

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

TECHNIQUE

```
v-bind="{ … }"
v-on="{ … }"
```

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

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

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

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

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

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

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

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

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

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

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

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

Questions?

BEST PRACTICES

CSS

BEST PRACTICES

CSS

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

MyRedText.vue

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

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

TECHNIQUE

Scoped Styles

MyRedText.vue

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

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

MyRedText.vue

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

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

MyRedText.vue

```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

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

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

MyRedText.vue

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

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

MyRedText.vue

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

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

MyRedText.vue

```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

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

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

TECHNIQUE

CSS Modules Exports

TECHNIQUE

CSS Modules Exports

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

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

Questions?

Practice

In the repo

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

In your app

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

VUE CLI

TOOL

Vue CLI

TOOL

Vue CLI - GUI Mode

PRO TIP

Vue CLI Modern Mode

PRO TIP

Vue CLI - Modern Mode

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

PRO TIP

Vue CLI - Modern Mode

```
vue-cli-service build --modern
```

- 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

Q&A

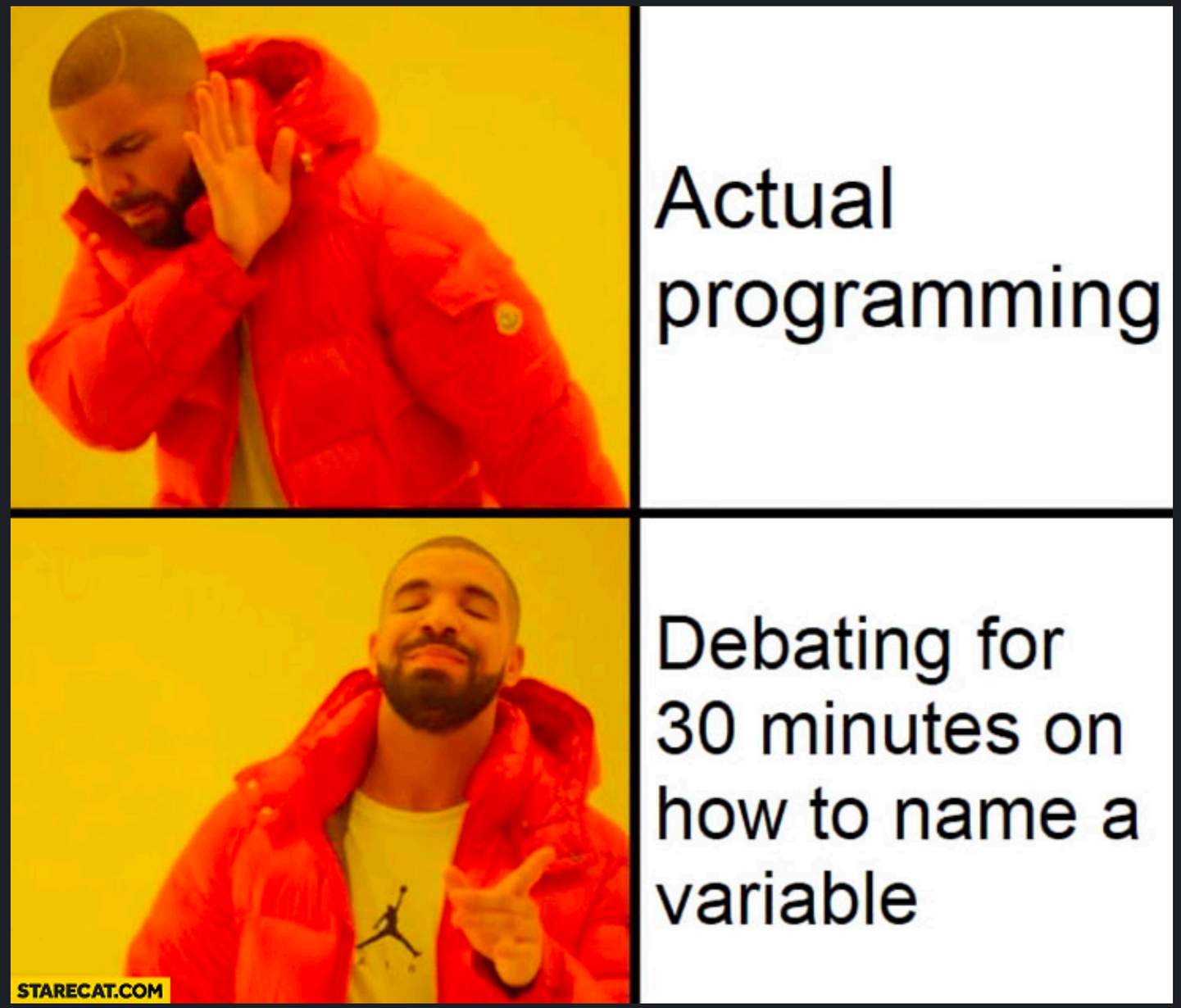
SHORT BREAK

Be back at 10:45AM!

COMPONENTS (PT. 1)

BEST PRACTICE

Naming Components


Actual programming
Debating for 30 minutes on how to name a variable
STARECAT.COM

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)
  }
}
```

BEST PRACTICE

Naming Component. Methods

Prefer destructuring over multiple arguments

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

❌ Not recommended

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

✅ Recommended

BEST PRACTICE

When to Refactor Your Components

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

- Donald Knuth

PRINCIPLE

Data Driven Refactoring

PRINCIPLE

Data Driven Refactoring

Signs you need more components

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

PRINCIPLE

Data Driven Refactoring

How to find reusable components?

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

Questions?

PRO TIP

SFC Code Block Order

MyFirstComponent.vue

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


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


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

MyFirstComponent.vue

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


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


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

MyFirstComponent.vue

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


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


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

MyFirstComponent.vue

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


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


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

MyFirstComponent.vue

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

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


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

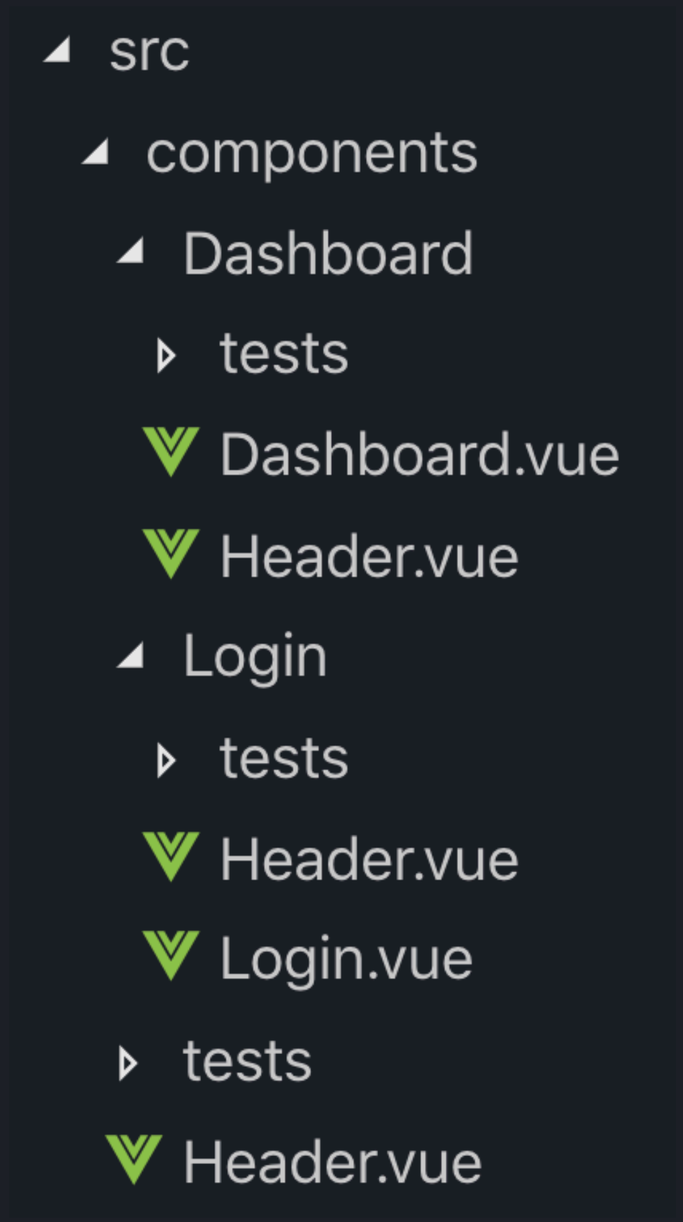
MyFirstComponent.vue

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

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

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

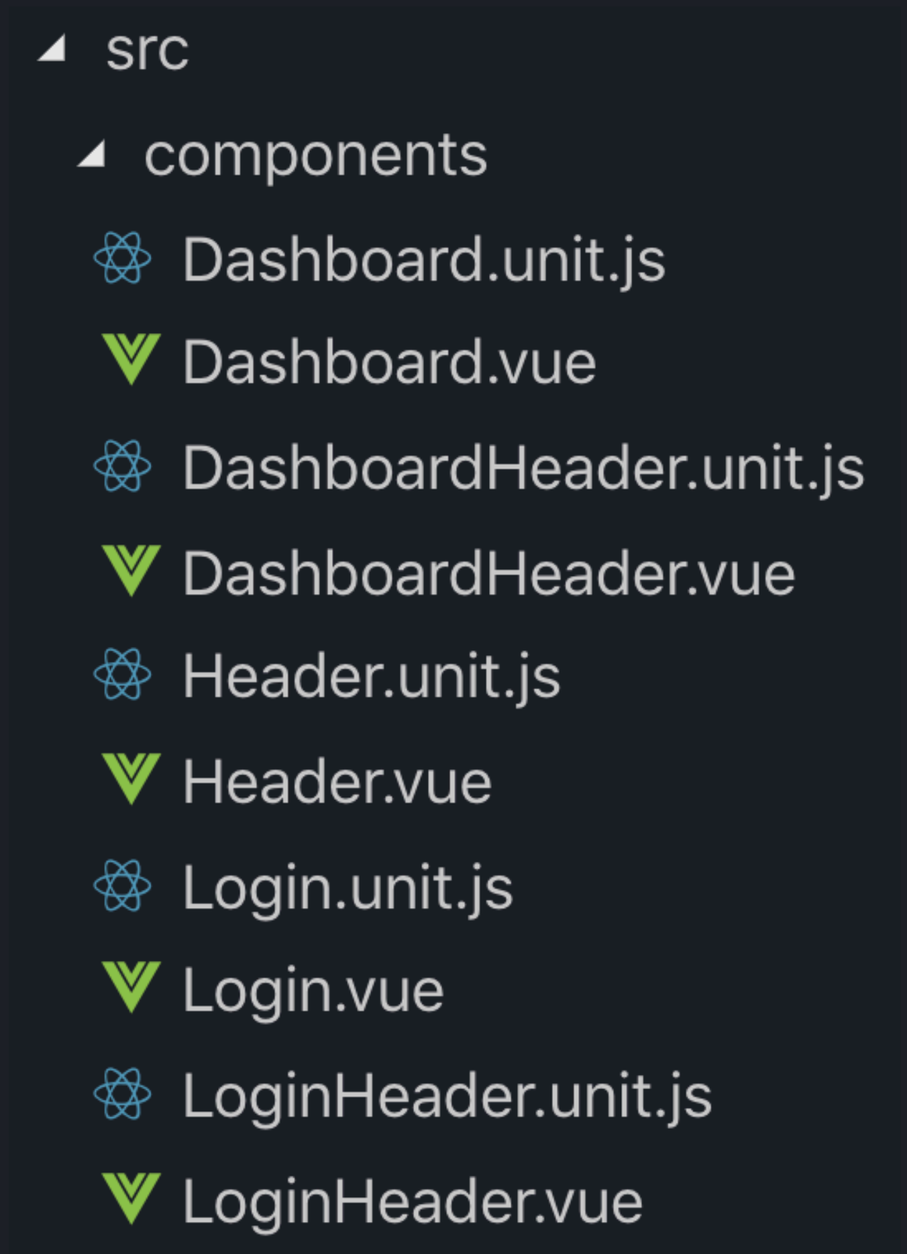
PRO TIP

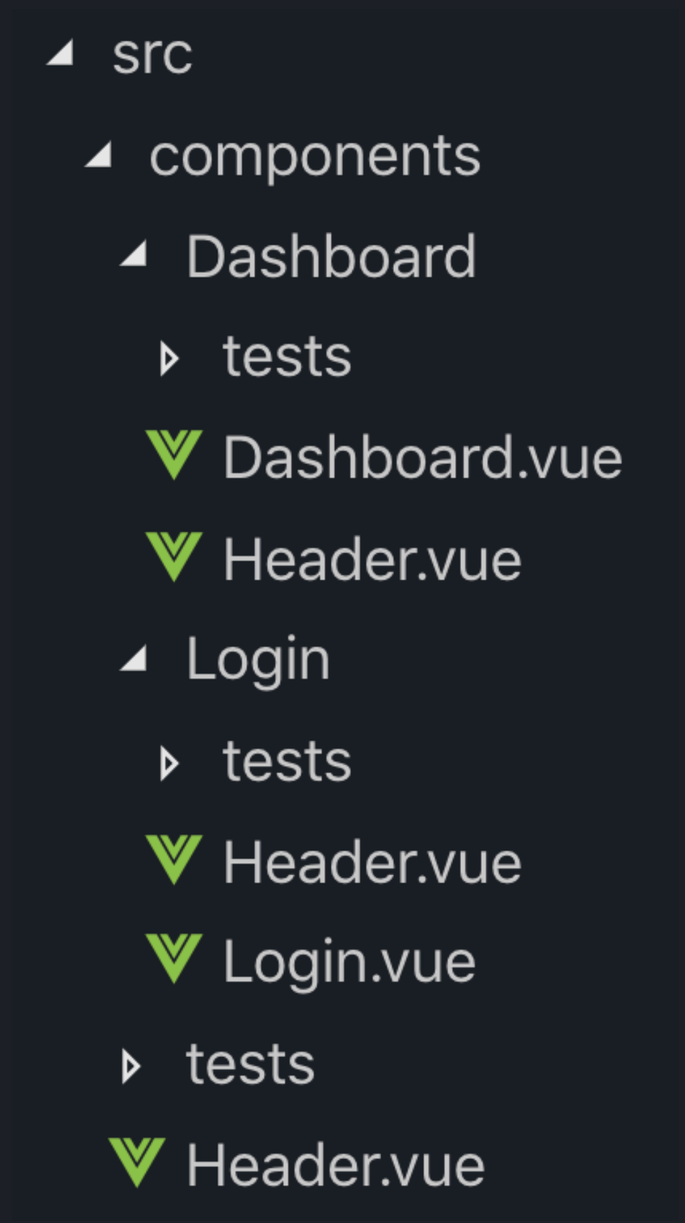
Component File Organization

Nested Structure



Flat Structure




src
components
Dashboard
tests
Dashboard.vue
Header.vue
Login
tests
Header.vue
Login.vue
tests
Header.vue

VS


src
components
Dashboard.unit.js
Dashboard.vue
DashboardHeader.unit.js
DashboardHeader.vue
Header.unit.js
Header.vue
Login.unit.js
Login.vue
LoginHeader.unit.js
LoginHeader.vue

PRO TIP

Component File Organization

Flat makes refactoring easier

No need to update imports if components move

Flat makes finding files easier

Folders often leads to lazily named files

because they don’t have to be unique

PRO TIP

Register base components globally

PRO TIP

Register base components globally

Instead of every component having:

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

Use this epic script by Chris Fritz:

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

Questions?