VueJS 3.x (Vue3)

Declarative Component-Based UI Framework

History of Vue.js

- Created by Evan You (ex-Googler)
- Oct 2015: version 1.0
- 2016-2012: vers 2.0-2.7
- 2020-2021: version 3.0-3.2

Which Front End Framework?

State of JS surveys:

2018 2019 2020 2021 2022

Resources

http://vuejs.org/guide (v1, v2, v3)

https://vuemastery.com

Online Playground

New in Vue 3.x: Composition API

- Vue 2.x
  - Use .vue component as building blocks
  - Difficult to share common code among these components
- New features in Vue 3.x
  - Composition API
    - Basic building blocks can be ***either .vue component*** or ***pure .TS code***
    - Easier to maintain shared logic across multiple components
  - Reactive references
  - Similar approach to functional React

Anatomy of .vue (SFC = Single File Component)

src/SampleComponent.vue

```
<template>
  <div>
     My number is {{count}}
  </div>
</template>

<script setup lang="ts">
import {ref} from "vue"
const count = ref(73)
</script>

<style scoped>
// CSS style rules applied to the HTML template above
</style>
```

.vue = .html + (.ts|.js) + (.css|.scss|.sass)

The class name has no significant meaning. You may name the class anything, but the common practice is to use the same name as the filename (without .vue)

*"**scope**" attribute implies the style will be applied only to this component and NOT to the child components of this one*

VueJS Playground

https://sfc.vuejs.org

1-way Data Binding
From class variables to UI

Data Binding: TS Expressions to UI

src//Hello.vue

```
<template>
  <h1>Hello {{who}} {{31 + 17}}</h1>
<template>

<script setup lang=”ts”>
import {ref} from “vue”
const who = ref(“VueJS”);
}
</script>
```

Begin
Hello VueJS 48
End

src//Hello.vue

```
<template>
  <h1>Hello {{who.toUpperCase()}}</h1>
<template>

<script setup lang=”ts”>
import {ref} from “vue”
const who = ref(“VueJS”)
</script>
```

Begin
Hello VUEJS
End

VueJS vs. JS & HTML

src//Hello.vue

```
<template>
  <h1>Hello {{who.toUpperCase()}}</h1>
<template>

<script setup lang=”ts”>
import {ref} from “vue”
const who = ref(“VueJS”)
</script>
```

hello.html

```
<html>
  <body>
    <h1>Hello <span id=”name”></span></h1>
    <script src=”mycode.js”></script>
  </body>
</html>
```

mycode.ts

```
const who = “VueJS”;
let sp = document.getElementById(“name”);
sp.innerText = who.toUpperCase();
```

Vue Data Binding Directives

HTML Attributes for Data Binding Directives

- `v-bind`: bind Vue data to HTML (native) attribute
- `v-for`: repeat data from arrays/lists
- `v-if`, `v-else`, `v-else-if`, `v-show`: conditional rendering
- `v-model`: 2-way data binding (data ⇌ UI)
  - Compare it to 1-way binding `{{my_data}}`
- And many more: `v-text`, `v-html`, `...`

Use Cases of Attribute Binding

```
<!-- How to control the paragraph alignment programmatically? -->
<p align="center">
    This a sample paragraph
</p>

<!-- How to control image size programmatically? -->
<img src="mypic.jpg" width="300" height="225" />
```

```
<template>
  <p v-bind:align="textDirection">This a sample paragraph</p>
  <img src="mypic.jpg" v-bind:width="imgWidth" v-bind:height="imgHeight" />
</template>
<script setup lang="ts">
const textDirection = ref("center")
const imgWidth = ref(300)
const imgHeight = ref(225)
</script>
```

v-bind: bind (Vue) data to HTML attributes

src//Hello.vue

```
<template>
  <div>
   <img src="http://bit.ly/10923f8d998.png">

   <!-- using a variable that holds the image URL -->
   <img v-bind:src="imgLocation">
   <img :src="imgLocation">
  </div>
<template>

<script setup lang="ts">
import {ref} from "vue"
const imgLocation = ref("https://bit.ly/10923f8d998.png");
</script>
```

```
// Compare to the following code snippet
const imgLocation = https://bit.ly/10923f8d998.png"
const imgEl = document.createElement("img");
imgEl.setAttribute(src, imgLocation);
```

`{{`*`data`*`}}` binds data/var to text nodes

`v-bind:`*<u>`attr`</u>*`="`*`data`*`"` binds data/var to HTML attrs

Playground Demo #2
data-binding & Vue DevTools

Iterate over arrays: v-for

src//Chemical.vue

```
<template>
  <div>
    <h1>Chemical Elements</h1>
    <ol>
      <li v-for="(a,arrIdx) in atoms"
          v-bind:key="arrIdx">{{a}}</li>
    </ol>
  </div>
<template>
<script setup lang="ts">
import {ref} from "vue"
const atoms = ref(["Argon", "Barium", "Carbon"])
</script>
```

```
<ol>
  <li :key="0">Argon</li>
  <li :key="1">Barium</li>
  <li :key="2">Carbon</li>
</ol>
```

Rendered Output

Chemical Elements

1. Argon
2. Barium
3. Carbon

Use one of the syntax options

```
<li v-for="(a,pos) in atoms" v-bind:key="pos">{{a}}</li>
<li v-for="(a,pos) in atoms"       :key="pos">{{a}}</li>
```

- *V-for must be used together with :key*
- *:key is required for improved VueJS rendering performance*
- *:key must be a unique value (of **any data type**) **among siblings***

More on v-for :key

```
const atoms = ref([
      {symbol: "Ar", name: "Argon"},
      {symbol: "C", name "Carbon"},
      {symbol: "Ne", name: "Neon"}
   ])
```

```
<ul>
  <li v-for="a in atoms"
      :key="a.symbol">{{a.name}}</li>
</ul>
```

```
<ul>
  <li :key="Ar">Argon</li>
  <li :key="C">Carbon</li>
  <li :key="Ne">Neon</li>
</ul>
```

```
<ul>
  <li v-for="(a,pos) in atoms"
      :key="pos">{{a.name}}</li>
</ul>
```

```
<ul>
  <li :key="0">Argon</li>
  <li :key="1">Carbon</li>
  <li :key="2">Neon</li>
</ul>
```

*Keys must be **unique among siblings** (like primary keys in DB)*

Playground Demo #3
forloop & forloop-object

Conditional: v-if, v-else-if, v-else

src//Random.vue

```
<template>
  <div>
    <h2>Random number: {{randVal}}</h2>
    <p v-if="randVal < 31">Below 31</p>
    <p v-else-if="randVal > 87">87 or more</p>
    <p v-else>Between 32-87</p>
  </div>
<template>
<script setup lang="ts">
const randVal = ref(Math.random() * 100);
</script>
```

Playground: Conditional

Rendered Output

Random number: 49

Between 32-87

```
<div>
  <h2>Random number: 49</h2>
  <p>Between 32-87</p>
</div>
```

Rendered Output

Random number: 14

Below 31

```
<div>
  <h2>Random number: 14</h2>
  <p>Below 31</p>
</div>
```

These directives automatically suppress elements whose condition evaluates to FALSE

v-if vs. v-show

src//Sample.vue

```
<template>
  <div>
    <p>Welcome</p>
    <p v-if=”debugMode”>Use STOP to kill the app</p>
    <p>Good bye!</p>
  </div>
</template>
<script setup lang=”ts”>

</script>
```

rendered HTML

```
<div>
  <p>Welcome</p>
  <p>Good bye!<p>
</div>
```

src//Sample.vue

```
<template>
  <div>
    <p>Welcome</p>
    <p v-show=”debugMode”>Use STOP to kill the app</p>
    <p>Good bye!</p>
  </div>
</template>
<script setup lang=”ts”>

</script>
```

rendered HTML

```
<div>
  <p>Welcome</p>
  <p style=”visibility:none”>Use STOP to kill the app</p>
  <p>Good bye!<p>
</div>
```

v-if vs. v-show

```
<ComponentOne v-if=”boolean_expression” >
<ComponentTwo v-show=”boolean_expressions” >
```

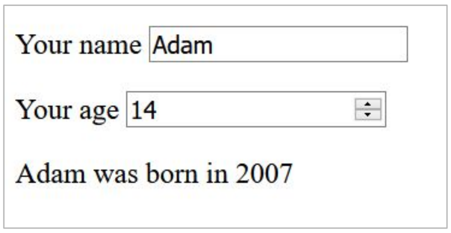
	Expressions is true	Expression is false
v-if	ComponentOne is created Setup function executed	ComponentOne is NOT created Setup function DID NOT execute
v-show	ComponentTwo is created Setup function executed	ComponentTwo is created but hidden Setup function executed

Playground Demo #4
conditional

Two-Way Data Binding (v-model)

src//Sample.vue

```
<template>
  <div>
    <p>Your name <input type=”text” v-model=”name”></p>
    <!--
    <p>Your name <input type=”text” v-model.lazy=”name”></p>
    -->
    <p>Your age <input type=”number” v-model.number=”age”></p>

    <p>{{name}} was born in {{thisYear - age}}</p>
  </div>
</template>

<script setup lang=”ts”>
import {ref} from “vue”
const thisYear: number = newDate().getFullYear();
const name = ref(“Adam”)
const age = ref(1)
</script>
```



Your name Adam

Your age 14

Adam was born in 2007

Vue Playground: Numeric & Lazy

- *Input type email, password, color, date is handled similarly to type=”text”*
- *Input type=”range” (a horizontal slider) is handled similarly to type=”number”*
- *Lazy: bind the value after input lost keyboard focus*

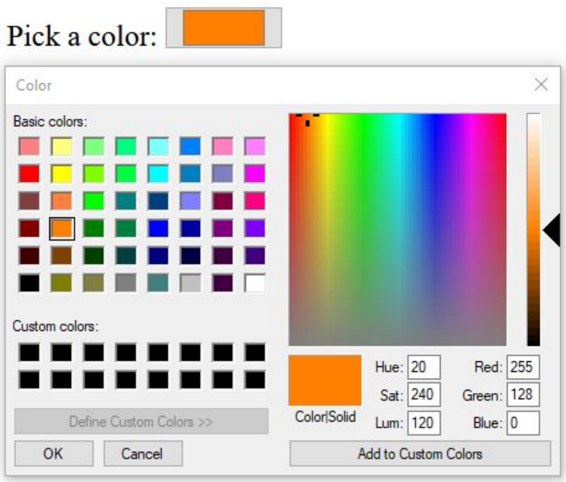
VueJS Data Binding

	One-way	One-way	Two-way
Syntax	`{{ varName }}`	`v-bind:attr=”varName”`	`v-model=”varName”`
Direction	Data → UI (Text)	Data to UI element attribute	Data ↔ UI
Effect	Updates to data are reflected on the UI	Updates to data are reflected on the HTML attribute	Updates to data are reflected on the UI *and vice versa*

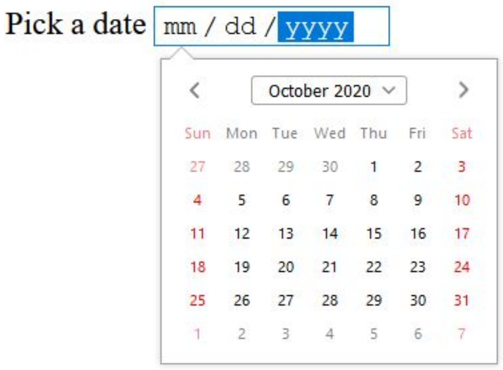
Playground Demo #5 simple input-binding

```
<input type="color" v-model="hexColorStr">
```

```
<input type="date" v-model="dateStr">
```



Hue: 20 Sat: 240 Lum: 120 Red: 255 Green: 128 Blue: 0



Pick a date mm / dd / yyyy

October 2020

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Playground Demo #6

input-binding

Event Handling v-on:_____

Handle Button Click

```
<script setup lang-=”ts”>
import { ref } from 'vue'

const count = ref(0)

function addOne() {
  count.value ++;
}
function subtractOne() {
  count.value --;
}
</script>

<template>
  <h1>Event Handling</h1>
  <p>
    Counter is {{count}}
  </p>
  <button v-on:click="addOne">More</button>
  <button @click="subtractOne">Less</button>
</template>
```

- *Use `.value` to access the current value of a reactive reference variable*
- *Use v-on:click=”__” of @click=”___”*

Online Playground: Button Click

Multiple Event Handlers on One Element

```
<template>
  <h1>Event Handling: Mouse Activity</h1>
  <p v-if="!mouseInside">
    Move mouse into the box
  </p>
  <p v-else>
    Move your mouse wheel
  </p>
  <div id="box" @wheel="wheelMoved"
                @mouseenter="mouseIn"
                @mouseleave="mouseOut">
    {{count}}
  </div>
</template>
```

```
<script setup lang="ts">
import { ref } from 'vue'

const count = ref(0)
const mouseInside = ref(false)

function wheelMoved(ev: WheelEvent) {
  count.value += Math.sign(ev.deltaY)
}

function mouseIn() {
  mouseInside.value = true
}
function mouseOut() {
  mouseInside.value = false
}
</script>
```

Event handling function may declare argument of the associated event type.

This can be used to obtain more details about the event.

Online Playground

Tons of Event Names & Event Classes

DOM Event Name	Event Handler Argument Type
keypress, keydown, keyup	KeyboardEvent
wheel	WheelEvent
click	MouseEvent
blue, focus	FocusEvent
mousedown, mouseenter, mousemove, mouseup	MouseEvent
and many more. . .	

These classes are native JavaScript classes, they are not defined by VueJS

Keyboard Events

```
<template>
  <h1>Event Handling: Keyboard Events</h1>
  <p>
    Place cursor in the input below.
    Press Alt-Up or Right Arrow
  </p>
  <p>
    Count is {{count}}
  </p>

  <input type="text" @keydown.alt.up="addTen"
                     @keydown.right="addOne"
                     @keydown.shift.esc.prevent="reset">
</template>
```

```
<script setup lang="ts">
import { ref } from 'vue'

const count = ref(0)

function addTen() {
  // Called only when Alt-Up arrow is pressed
  count.value += 10
}

function addOne() {
  // Called only when Right arrow is pressed
  count.value ++
}
function reset() {
  // Called only when Shift-Esc is pressed
  count.value = 0
}
</script>
```

Online Playground

Mouse/Keyboard Event Filters/Modifiers

Filters

.enter
.tab
.delete
.esc
.space
.up
.down
.left
.right

Modifiers

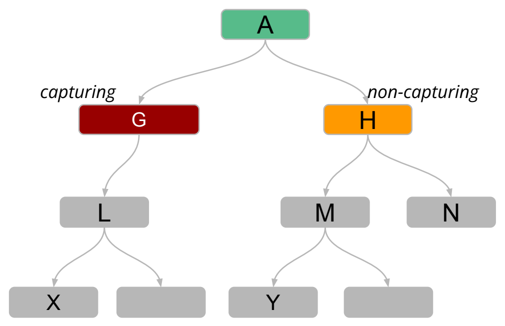
.alt
.ctrl
.meta
.shift

Mouse Modifiers

.left
.right
.middle

Mouse/Keyboard Event Filters/Modifiers

Event Modifier	Description
.prevent	Prevent browser default action of the event
.stop	Stop propagating event up (bubbling) to ancestor
.capture	Begin here, and propagate event down (capturing) to descendants
.self	Handle events only from self (neither from ancestors nor from descendants)



- *Events originating in X are handled G, then L, then A*
- *Events originating in Y are handled by M, then Hm, the A*

Online Playground: Event Bubble/Capturing

From index.html to App.vue

Hello World (Simplified)

index.html

```
<html>
  <body>
    <p>Begin</p>
    <div id="app"></div>
    <p>End</p>
    <script type="module"
            src="/src/main.ts" />
  </body>
</html>
```

src/main.ts

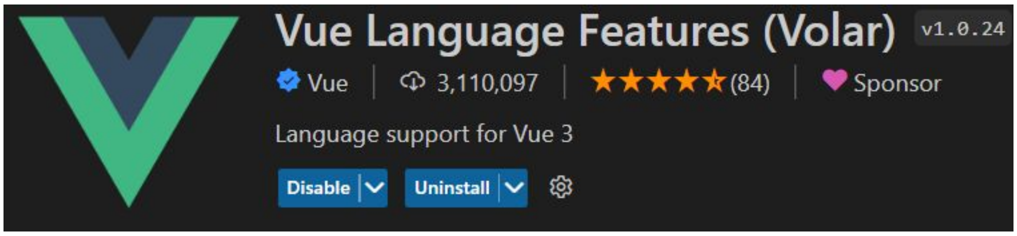
```
import {createApp} from "vue";
import App from "./App.vue";

const app = createApp(App)
app.mount("#app");
```

src/App.vue

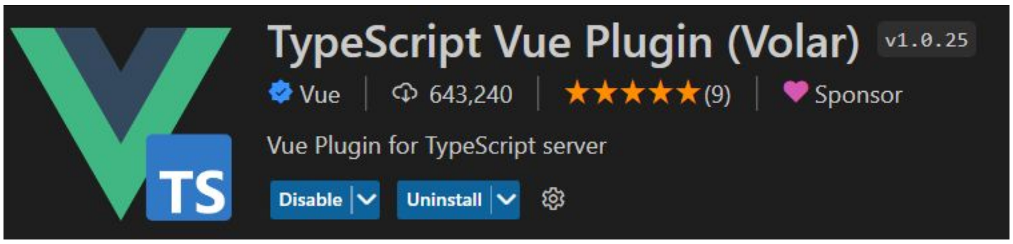
```
<template>
  <h1>Hello VueJS</h1>
<template>
```

Begin
Hello VueJS
End

Recommended VSCode Extensions

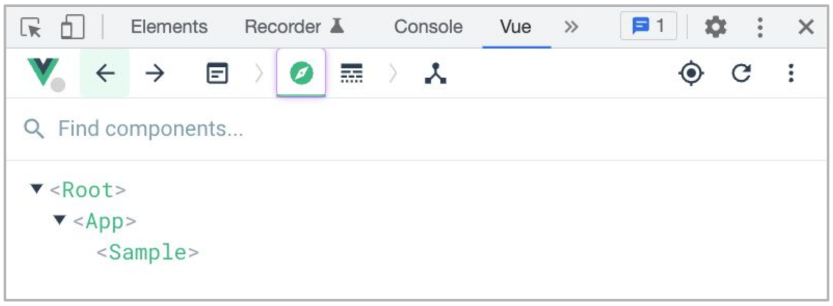


Vue Language Features (Volar) v1.0.24
Vue | 3,110,097 | ★★★★☆ (84) | Sponsor
Language support for Vue 3

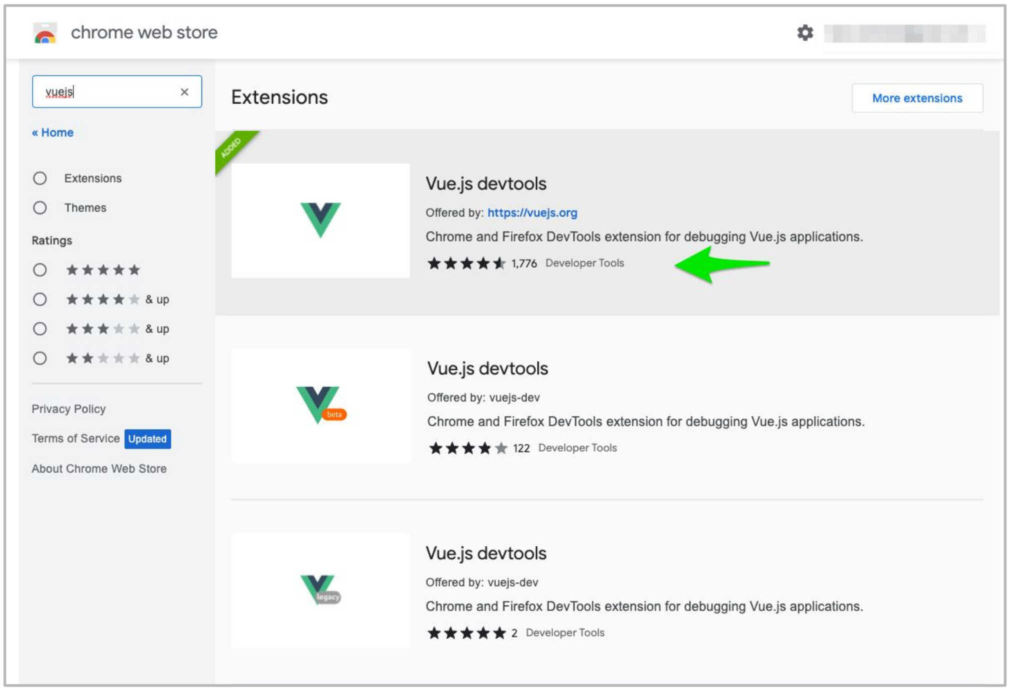


TypeScript Vue Plugin (Volar) v1.0.25
Vue | 643,240 | ★★★★★ (9) | Sponsor
Vue Plugin for TypeScript server

Recommended Browser Extensions



Vue Devtools Browser extension
(for FireFox and Chrome)


chrome web store

vuejs

« Home

Extensions

Themes

Ratings

★★★★★

★★★★ & up

★★★ & up

★★ & up

Privacy Policy

Terms of Service Updated

About Chrome Web Store

Extensions

More extensions

ADDED

Vue.js devtools
Offered by: https://vuejs.org
Chrome and Firefox DevTools extension for debugging Vue.js applications.
★★★★★ 1,776 Developer Tools

Vue.js devtools
Offered by: vuejs-dev
Chrome and Firefox DevTools extension for debugging Vue.js applications.
★★★★★ 122 Developer Tools

Vue.js devtools
Offered by: vuejs-dev
Chrome and Firefox DevTools extension for debugging Vue.js applications.
★★★★★ 2 Developer Tools

Using vite 4.x

- Prerequisite: Node.js already installed
  - LTS versions 12.22.x or 14.17.x or 16.x
- Install yarn (Optional)

```
npm --version   # check your NPM version

# Option 1
npm create vite@latest my-app    --template vue-ts     # NPM 6.x
npm create vite@latest my-app --  --template vue-ts    # NPM 7.x or later

# Option 2
yarn create vite my-app --template vue-ts
```

package.json

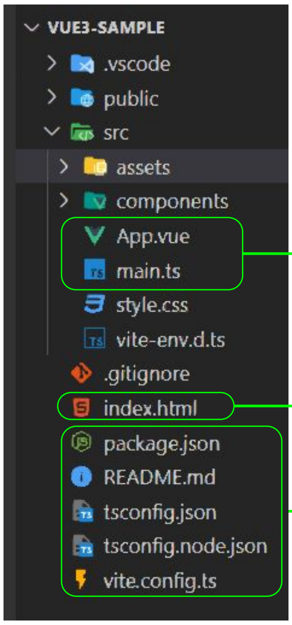
```
{
  “scripts”: {
    “dev”: “vite”,
    “build”: “vue-tsc && vite build”,
    “preview”: “vite preview”
  }
  “dependencies” {
     “vue”: “3.2.45”
  },
  “devDependencies”: {
    “@vitejs/plugin-vue”: “^4.0.0”,
    “typescript”: “^4.9.3”,
    “vite”: “^4.1.0”,
    “vue-tsc”: “^1.0.24”
}
```

```
npm run dev
# OR yarn dev

npm run build
# OR yarn build

npm run preview
# OR yarn preview
```

npm vs. yarn

NPM	Yarn
`npm init -y`	`yarn init -y`
`npm install`	`yarn install`
`npm install -g` *`package-name`*	`yarn global add` *`package-name`*
`npm install` *`package-name`*	`yarn add` *`package-name`*
`npm install -save` *`package-name`*	`yarn add` *`package-name`*
`npm install -save-dev` *`package-name`*	`yarn add -D` *`package-name`*
`npm run` *`name-of-script`*	`yarn` *`name-of-script`*

Files Generated by vite


```
VUE3-SAMPLE
  > .vscode
  > public
  v src
    > assets
    > components
      App.vue
      main.ts
      style.css
      vite-env.d.ts
    .gitignore
    index.html
    package.json
    README.md
    tsconfig.json
    tsconfig.node.json
    vite.config.ts
```

Main entry point and Vue components

Webapp landing page

Project settings

Live Demo:

(a) Vue DevTools

(b) Browser Debugger