

CIS 658 Web Architectures

Vue.js I

Fundamentals with the Composition API



GRAND VALLEY
STATE UNIVERSITY®

Lecturer: **Dr. Yong Zhuang**

What is Vue?

- Vue (pronounced /vju:/, like view) is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be they simple or complex.



History of Vue.js

- Created by Evan You (ex-Googler)
- Version 0.6: Dec 2013 (first version on GitHub)
- Oct 2015: version 1.0
- 2016-2018: vers 2.0-2.5
- 2019-2021: vers 2.6.0-2.6.14
- 2020-2022: version 3.0-3.2
- Latest version: 3.2.31



Related Background:

**Custom (HTML) Elements/
W3C Web Components**

Why Web Components?

```

<head>
  <style>
    .fancy-button {
      background-color: #6200ea;
      color: white;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
    }

    .fancy-button:hover {
      background-color: #3700b3;
    }
  </style>
</head>
<body>
  <button class="fancy-button">Click Me</button>
  <button class="fancy-button">Submit</button>
  <script>
    document.querySelectorAll('.fancy-button').forEach(button => {
      button.addEventListener('click', () => {
        alert('Button clicked!');
      });
    });
  </script>
</body>

```

```

<head>
  <title>Web Components Button Example</title>
  <script src="fancy-button.js" defer></script>
</head>
<body>
  <h2>Web Components</h2>
  <fancy-button label="Click Me"></fancy-button>
  <fancy-button label="Submit"></fancy-button>
</body>

```

- The logic for the Web Component is in a separate file, making it reusable across different HTML pages.

- The HTML file focuses on the structure and layout, while the JavaScript file handles the Web Component's behavior and styling.

Resources

<https://vuejs.org/guide> (v1, v2, v3)

<https://www.vuemastery.com>

Which Vue version?

- Vue 2.6.x
 - Many options for UI third-party libraries
 - These UI libraries are being ported to support Vue 3.x
- New features in Vue 3.x
 - Separate configuration settings per instance: easier to create multiple instances of VueJS within one web app
 - Composition API
 - Easier to maintain large components
 - Easier to maintain shared logic across multiple components
 - Reactive references

Options API vs. Composition API

Vue 3 Composition API

```
<script setup>
import { ref, onMounted } from 'vue';

const task = ref("Complete the Vue.js project"); // Reactive state for the task
const logTask = () => console.log(`Your current task is: ${task.value}`);

onMounted(() => {
  logTask();
});
</script>
```

Vue 2 Options API

```
<script>
export default {
  data() {
    return {
      task: "complete the Vue.js project",
    };
  },
  methods: {
    logTask() {
      console.log(`Your current task is: ${this.task}`);
    },
  },
  mounted() {
    this.logTask();
  },
};
</script>
```

Vue Single File Component Playground

<https://sfc.vuejs.org/>

Vue DevTools

chrome web store zhuangyo@mail.gvsu.edu

Help test the new Chrome Web Store in Preview
Try out the new experience →

x

« Home


- Extensions
- Themes

Ratings

- ★★★★★

Extensions ⓘ

[More extensions](#)



V

Vue.js devtools

vuejs.org

Browser DevTools extension for debugging Vue.js applications.

★★★★★ 1,893 Developer Tools

Vue.js devtools

Manage Extensions

Visit Chrome Web Store

New tab Ctrl+T

New window Ctrl+N

New Incognito window Ctrl+Shift+N

History ▶

Downloads Ctrl+J

Bookmarks ▶

Google Password Manager New

Extensions ▶

Zoom - 100% + [Full Screen]

Print... Ctrl+P

Cast...

Find... Ctrl+F

More tools ▶

Edit Cut Copy Paste

Settings

Help ▶

Exit

Create a new Vue App Using vite 4.x

- Install Node.js
 - Using Docker
 - Direct Installation [Node.js](#)
- [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
```

Run Vue App

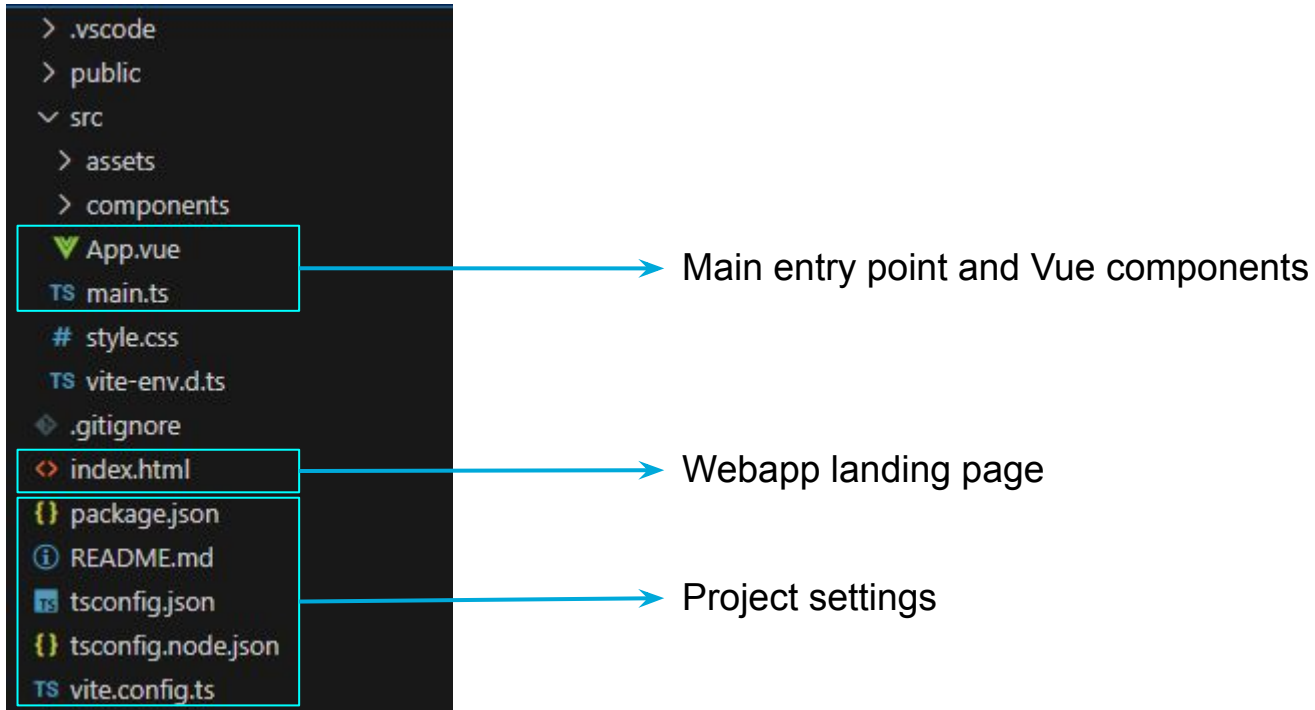
package.json

```
"scripts": {  
  "dev": "vite",  
  "build": "vue-tsc && vite build",  
  "preview": "vite preview"  
},  
"dependencies": {  
  "vue": "^3.3.4"  
},  
"devDependencies": {  
  "@vitejs/plugin-vue": "^4.2.3",  
  "typescript": "^5.0.2",  
  "vite": "^4.4.5",  
  "vue-tsc": "^1.8.5"  
}
```



```
npm install  
  
npm run dev  
# OR  
yarn dev  
  
npm run build  
# OR  
yarn build  
  
npm run preview  
# OR  
yarn preview
```

Files Generated by vite



The image shows a file explorer window with a dark background. The file tree is as follows:

- > .vscode
- > public
- ▼ src
 - > assets
 - > components
 - ▼ App.vue (highlighted with a red box and an arrow pointing to the text "Main entry point and Vue components")
 - TS main.ts
 - # style.css
 - TS vite-env.d.ts
- ◆ .gitignore
- <> index.html (highlighted with a red box and an arrow pointing to the text "Webapp landing page")
- { } package.json (highlighted with a red box and an arrow pointing to the text "Project settings")
- ① README.md
- ts tsconfig.json
- { } tsconfig.node.json
- TS vite.config.ts

Anatomy of a .vue (SFC = Single File Component)

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

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>
```

Hello World Project (default)

public/index.html

```
<body>
  <div id="app"></div>
  <!-- built files will be auto injected -->
</body>
```

src/main.ts

```
import { createApp }
import './style.css'
import App from './App.vue'

createApp(App).mount('#app')
```

```
<template>
  <div>
    <a href="https://vitejs.dev" target="_blank">
      
    </a>
    <a href="https://vuejs.org/" target="_blank">
      
    </a>
  </div>
  <HelloWorld msg="Vite + Vue" />
</template>
```

src/App.vue

Typical project organization:

- One index.html
- One main.ts
- Many xxxx.vue files

Anatomy of a .vue (SFC = Single File Component)

src/SampleComponent.vue

```
<template>
```

```
  <!-- The UI design in HTML goes here -->
```

```
</template>
```

<template> contains the HTML elements

```
<script setup lang="ts">
```

```
import { ref } from 'vue'
```

```
// Your data and methods/functions
```

```
</script>
```

The script includes the data and logical code for this component.

```
<style scoped>
```

```
/* CSS style rules applied to HTML template above. */
```

```
</style>
```

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

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 \rightleftharpoons UI)
 - Compare it to 1-way binding `{{my_data}}`
- And many more: v-text, v-html, ...

1-way Data Binding

From variables to UI

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

src/Hello.vue

```
<template>
  <div>
    
    
    
  </div>
</template>
<script setup lang="ts">
import { ref } from 'vue'
const imgLocation = ref("https://avatars.githubusercontent.com/u/6128107?s=280&v=4");
</script>
```

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

double moustache syntax: `{{data}}`
binds data/var to text nodes

`v-bind:attr="data"` binds data/var to HTML attrs

1-way Data Binding

```
<template>
  <h1>Hello {{ who }}</h1>
</template>

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

  const who = ref("VueJS 3.x");
</script>

<style scoped>
  h1 {
    color: #888;
  }
</style>
```



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

Only evaluate one expression at a time



```
<h1>Hello {{ who.toLocaleUpperCase(); who = "World" }}</h1>
```



1-way Data Binding

```
<template>  
  <h1>Hello {{ who }}</h1>  
</template>
```

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

```
const who = ref("VueJS 3.x");  
</script>
```

```
<style scoped>  
h1 {  
  color: #888;  
}  
</style>
```



```
<h1>Hello {{ if(!who) {return "World"} }}</h1>
```



```
<h1>Hello {{ who? who: "World" }}</h1>
```



```
<h1>Hello {{ who || "World" }}</h1>
```

Iterate over arrays: v-for

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

src/Hello.vue

```
<ol>
  <li>Argon</li>
  <li>Barium</li>
  <li>Carbon</li>
</ol>
```

Chemical Elements

1. Argon
2. Barium
3. Carbon

Rendered Output

- *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

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

```
<script setup lang="ts">
import { ref } from 'vue';
const atoms = ref([
  { symbol: "Ar", name: "Argon" },
  { symbol: "C", name: "Barium" },
  { symbol: "Ne", name: "Carbon" }
]);
</script>
```

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

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

Using key with v-for in Vue.js 3

- Include key:
 - Whenever you've got something unique
 - You're iterating over more than a simple hard coded array
 - and even when there is nothing unique but it's dynamic data (in which case you need to generate random unique id's)
- Without key:
 - You have nothing unique AND
 - When you're quickly testing something or demonstrating basic functionality with v-for
 - If you're iterating over a simple hard-coded array that doesn't change and doesn't need complex updates (not dynamic data from a database, etc)

Repeating a group of elements (the wrong way)

```
src/Movie.vue
<template>
  <div>
    <h2 v-for="(movie, idx) in movies" :key="idx">{{ movie.title }}</h2>
    <p v-for="(movie, idx) in movies" :key="idx">Release year: {{ movie.year }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from 'vue';
const movies = ref([
  { title: "Batman Begins", year: 2005 },
  { title: "The Upside", year: 2017 },
]);
</script>
```

```
<div>
  <h2>Batman Begins</h2>
  <h2>The Upside</h2>
  <p>Release year: 2005</p>
  <p>Release year: 2017</p>
</div>
```

Rendered Output

Batman Begins
The Upside

Release year: 2005
Release year: 2017

We need to put the title and year of each movie in a block.

Repeating a group of elements

```
<template>
  <div v-for="(movie, idx) in movies" :key="idx">
    <h2>{{ movie.title }}</h2>
    <p>Release year: {{ movie.year }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from 'vue';

const movies = ref([
  { title: "Batman Begins", year: 2005 },
  { title: "The Upside", year: 2017 },
]);
</script>
```

src/Movie.vue

```
<div>
  <h2>Batman Begins</h2>
  <p>Release yer: 2005</p>
</div>
<div>
  <h2>The Upside</h2>
  <p>Release yer: 2017</p>
</div>
```

Batman Begins

Release year: 2005

The Upside

Release year: 2017

Rendered Output

Repeat N times: v-for

src/Hello.vue

```
<template>
  <div>
    <h1>Count Down</h1>
    <p v-for="n in 5" v-bind:key="n">Downto {{ 6 - n }}...</p>
  </div>
</template>
```

- The loop variable (*n* in the above snippet) starts at **ONE** (not zero)
- Variables can be used in place of the constant (number 5)

Count Down

Downto 5...
Downto 4...
Downto 3...
Downto 2...
Downto 1...

Rendered Output

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">
  import { ref } from "vue";
  const randVal = ref(Math.floor(Math.random() * 100));
</script>
```

These directives automatically suppress elements whose condition evaluates to FALSE

Random number: 49

Between 32-87

Rendered Output

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

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

Random number: 14

Below 31

Rendered Output

Group Conditional

Poor

```
<template>
  <div>
    <h2>Introduction</h2>
    <p>Lorem ipsum ...blah...blah...</p>
    <h2 v-if="!loggedIn">Logging in to your Account</h2>
    <p v-if="!loggedIn">Please do the following__</p>

    <p>Thank you!</p>
  </div>
</template>
```

Better

```
<template>
  <div>
    <h2>Introduction</h2>
    <p>Lorem ipsum ...blah...blah...</p>
    <template v-if="!loggedIn">
      <h2>Logging in to your Account</h2>
      <p>Please do the following__</p>
    </template>
    <p>Thank you!</p>
  </div>
</template>
```

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">
import { ref } from "vue";
const debugMode = ref(false);
</script>
```

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">
import { ref } from "vue";
const debugMode = ref(false);
</script>
```

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

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

v-show and v-if on Custom Vue Elements

```
<template>  
  <DeliveryMessage order-id="FY4328ZX" v-if="itemDelivered"></DeliveryMessage>  
</template>
```

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

```
const itemDelivered = ref(false)  
</script>
```

Since DeliveryMessage is NOT rendered, any initialization logic in DeliveryMessage.vue did NOT run

```
<template>  
  <DeliveryMessage order-id="FY4328ZX" v-show="itemDelivered"></DeliveryMessage>  
</template>
```

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

```
const itemDelivered = ref(false);  
</script>
```

Since DeliveryMessage is rendered (but hidden), any initialization logic in DeliveryMessage.vue would have RUN

1-way Data Binding (from variables to UI)

```
<template>
  <h1>Hello {{ who }}</h1>
  <div>
    >
    <ol>
      <li v-for="(a, idx) in atoms" v-bind:key="idx">{{ a }}</li>
    </ol>
  </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>
</template>
<script setup lang="ts">
import { ref } from "vue";
const who = ref("VueJS 3.x");
const imgLocation = ref("https://bit.ly/10923f8d998.png");
const atoms = ref(["Argon", "Barium", "Carbon"]);
const randVal = ref(Math.random() * 100);
</script>
```

Demo

Practice