Components in Vue

Components are reusable building blocks in Vue apps. They are like custom HTML elements you define and reuse across your app.

we need Watchers

- Reuse code
- Organize big projects
- Separate logic into meaningful chunks
- Easy to maintain and scale

That's where watchers shine. Unlike computed, which is for deriving values, watchers are for doing something when values change.

Structure of a Vue Component

A Vue component usually has three main parts: template, script and style.

```
<template> Block this is where you write the HTML (the user interface).

<script setup> Block
This is where you write your JavaScript logic: variables, functions, props,
events, etc.

<style> Block
This is where you write CSS to style your component.
```

```
<template>
<h2>Hello World!</h2>
</template>

<script setup>
// you can write logic here
</script>

<style scoped>
/* styles go here */
</style>
```

Example :

```
<template>
  <div>
    <h1>Hello, {{ name }}</h1>
```

```
    <button @click="changeName">Change Name</button>
  </div>
</template>

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

const name = ref('Vue User')

function changeName() {
  name.value = 'Vue Master'
}
</script>

<style scoped>
h1 {
  color: green;
}
button {
  margin-top: 10px;
  padding: 6px 12px;
}
</style>
```

Types of Components in Vue

1. Root Component This is the main component from which the entire app starts. It's usually defined in App.vue. Mounted in main.js.

```
// main.js
import { createApp } from 'vue'
import App from './App.vue'

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

<!-- App.vue -->
<template>
  <h1>This is the Root Component</h1>
</template>
```

2. Child Components These are components nested inside other components. They help break the UI into reusable pieces.

```
<!-- HelloWorld.vue -->
<template>
  <p>Hello from Child Component!</p>
</template>

<!-- In App.vue -->
```

```
<template>
  <HelloWorld />
</template>

<script setup>
import HelloWorld from './components/HelloWorld.vue'
</script>
```

3. Global Components Registered once and used anywhere in the app without importing. Not recommended for large apps (can clutter global space).

```
// main.js
import HelloWorld from './components/HelloWorld.vue'
app.component('HelloWorld', HelloWorld)
Now you can use <HelloWorld /> in any component.
```

4. Local Components Registered and used only inside a specific component. Keeps code organized and scoped.

```
<script>
import HelloWorld from './HelloWorld.vue'

export default {
  components: {
    HelloWorld
  }
}
</script>
```

5. Functional Components Stateless, no reactive data. Lightweight, faster to render. Often used for simple UI rendering.

```
<script setup>
defineProps(['title'])
</script>

<template>
  <h2>{{ title }}</h2>
</template>
```

6. Dynamic Components Loaded or switched dynamically using .

```
<component :is="currentComponent"></component>

<script setup>
```

```
import CompA from './CompA.vue'
import CompB from './CompB.vue'

const currentComponent = ref('CompA') // or 'CompB'
</script>
```

7. Async Components Loaded only when needed (lazy loading). Useful for improving performance.

```
const AsyncComponent = defineAsyncComponent(() =>
  import('./MyBigComponent.vue')
)
```

Component Registration

Before using a component inside another, Vue needs to know about it. Registration is how we tell Vue, "Hey, this component exists, and I want to use it."

1. Local Component Registration

Local means the component is only available in the component where it's registered.

Create your child component (e.g., MyButton.vue)

```
<!-- MyButton.vue -->
<template>
  <button class="my-button">Click Me</button>
</template>

<script>
export default {
  name: 'MyButton'
}
</script>
```

Import and register it locally in a parent component (e.g., App.vue)

```
<!-- App.vue -->
<template>
  <div>
    <MyButton />
  </div>
</template>

<script>
import MyButton from './components/MyButton.vue'
```

```
export default {
  components: {
    MyButton  // Local registration
  }
}
</script>
```

2. Global Component Registration

Global means the component is available in all components without importing every time.

Create global component (e.g., BaseCard.vue)

```
<!-- BaseCard.vue -->
<template>
  <div class="base-card"><slot /></div>
</template>

<script>
export default {
  name: 'BaseCard'
}
</script>
```

Register it globally in main.js or main.ts

```
// main.js
import { createApp } from 'vue'
import App from './App.vue'
import BaseCard from './components/BaseCard.vue'

const app = createApp(App)

app.component('BaseCard', BaseCard)  // Global registration

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

Example : Global Registration of a LoadingSpinner Component

Create the LoadingSpinner.vue component

```
<template>
  <div class="spinner">Loading...</div>
</template>
```

```
<script setup>
</script>

<style scoped>
.spinner {
  font-size: 1.2rem;
  color: teal;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
</style>
```

Register LoadingSpinner globally in main.js

```
import { createApp } from 'vue'
import App from './App.vue'
import LoadingSpinner from './components/LoadingSpinner.vue'

const app = createApp(App)

//Global registration
app.component('LoadingSpinner', LoadingSpinner)

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

Use LoadingSpinner in any component (Home.vue) without importing

```
<template>
  <h2>Home Page</h2>
  <LoadingSpinner />
</template>

<script setup>
// No import needed for globally registered components!
</script>
```