Creating Vue3 Components

Why Components?

- Reusable
- Organize application into modular units
- Deploy services to your client by paid subscription
  - Disable/enable services by adding/removing components
- Easier testing: module by module
- What else?

In VueJS

One Component ⇒ One .vue File

SFC = Single File Component

Defining and Using Components

Define here

ThumbsUp.vue

```
<template>
  <p>👍</p>
</template>
<script lang=”ts” setup>
import {ref} from “vue”

// More code here
</script>
```

Playground Demo

Use the component here

Sample.vue

```
<template>
  <p>Use the component here</p>
  <ThumbsUp></ThumbsUp>
</template>
<script lang=”ts” setup>
import {ref} from “vue”
import ThumbsUp from “./ThumbsUp.vue”

// More code here
</script>
```

Customizing Components via Properties

Defining and Passing "Argument"

```
<template>
  Can you show {{props.repeat}} thumbs?
  <span v-for="k in props.repeat">👍</span>
</template>
<script setup lang="ts">
import {defineProps} from "vue"

type ThumbProp = {
  repeat: number
}
const props = defineProps<ThumbProp>()
</script>
```

Sample.vue

```
<template>
  <p>Use the component here</p>
  <ThumbsUp :repeat="7"></ThumbsUp>
</template>
<script lang="ts" setup>

import ThumbsUp from "./ThumbsUp.vue"

// More code here
</script>
```

Playground

Simple Timer

```
<template>
  <div id="timer">
    <div id="timedisplay">
      {{minutes}}:{{twoDigitSeconds()}}
    </div>
    <button @click="runTimer">
      Start
    </button>
  </div>
</template>
```

```
<script setup lang="ts">
  import {ref} from "vue"
  const seconds = ref(0)
  const minutes = ref(0)

  function twoDigitSeconds() {
    return seconds.value
           .toLocaleString('en-US', { minimumIntegerDigits: 2})
  }
  function updateTime() {
    seconds.value++;
    if (seconds.value === 60) {
      minutes.value++
      seconds.value = 0
    }
  }
  function runTimer() {
    // Update the time once every second (1000 milliseconds)
    setInterval(updateTime, 1000)
  }
</script>
```

VueJS Reactive Reference + TypeScript Typing

The TS compiler infers the type from the surrounding context

```
import {ref} from “vue”
const name = ref(“”)      // name.value is implicitly a string
const year = ref(2001)    // year.value is implicitly a number
const names = ref([])     // names.value is an array of UNKNOWN type
```

```
import {ref, Ref} from “vue”
const name: Ref<string> = ref(“”)        // name.value is a string
const year: Ref<number> = ref(2001)      // year.value is a number
const names: Ref<string[]> = ref([])     // names.value is an array of string
```

Playground Demo

Simple Timer

Customization of Components Via Props

- Injection into component variable(s):
  - Timer update speed
  - Timer update speed with default value
- Injection into UI `<template> & <style>`: Stylish Timers
- *what else?*

Setting Default Value on Properties

```
<script setup lang="ts">
  import {defineProps} from "vue"
  type TimerProp = {
    updateInterval: number
  }
  const props = defineProps<TimerProp>()

  // more code here
</script>
```

```
<script setup lang="ts">
  import {defineProps, withDefaults} from "vue"
  type TimerProp = {
    updateInterval: number
  }
  const props = withDefaults(
    defineProps<TimerProp>(),
    {
       updateInterval: 1000
    }
  )

  // more code here
</script>
```

Passing Values to Properties

Timer.vue

```
<script setup lang="ts">
  import {defineProps} from "vue"
  type TimerProp = {
    updateInterval: number,
    borderColor: string
  }
  const props = defineProps<TimerProp>()

  // more code here
</script>
```

App.vue

```
<template>
  <!-- other elements here -->

  <Timer v-bind:update-interval=”100”
         :border-color=”#530A44”/>
  <!-- other elements here -->
</template>

<script setup lang=”ts”>
import Timer from “./Timer.vue”

</script>
```

camelCase in TypeScript

kebab-case in HTML

(DOM) Element Lifecycle

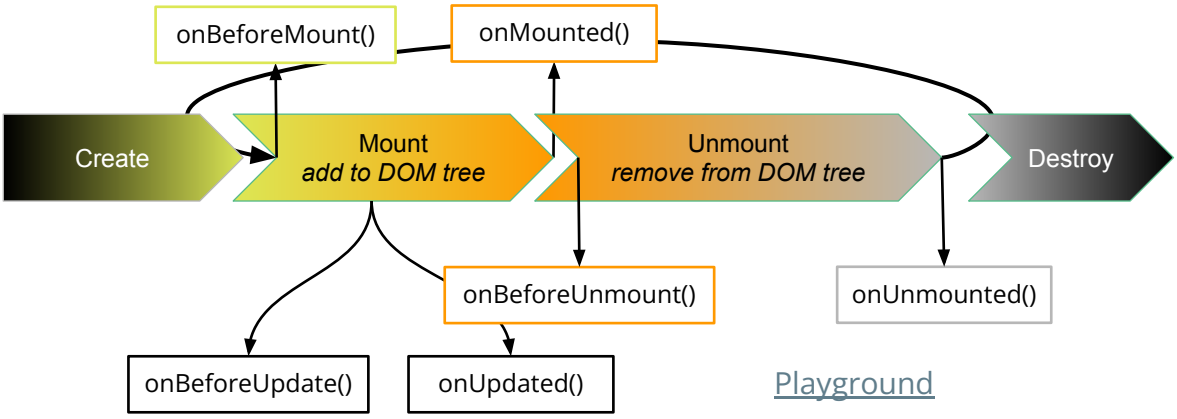
Creating DOM Tree Programmatically

```
const top = document.getElementById(“foo”)

const par = document.createElement(“p”);                  // <p>created
top.appendChild(par);                                     // <p> inserted to DOM tree

const text = document.createTextNode(“Hello world”)       // text created
par.appendChild(text);                                    // text inserted to DOM tree
```

Vue3 Lifecycle Functions



Playground

Practical Use of Lifecycle Hooks

Function	General Description	Sample Usage
`onBeforeMount()`	Component will appear	Restore UI from persistent storage (user prefs)
`onMounted()`	Component appeared	Start timer to monitor user engagement
`onBeforeUpdate()`	Properties will be updated	Any necessary logic needed • to save any data related to the old props • to restore data related to the new props
`onUpdated()`	Properties updated	
`onBeforeUnmount()`	Component will disappear	Stop timer
`onUnmounted()`	Component disappeared	Save UI details to user prefereces