CIS 371 Web Application Programming

VueJS 3.x (Vue3) II

Declarative Component-Based UI Framework

Lecturer: Dr. Yong Zhuang

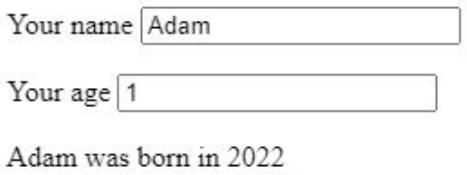
Based on the original version by Professor Hans Dulimarta.

Two-Way Data Binding (v-model)

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 name = ref("Adam");
const thisYear = new Date().getFullYear();
const age: number = 13;
</script>
```

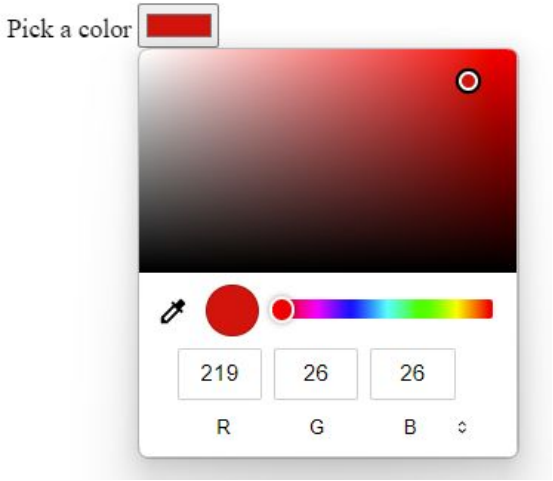


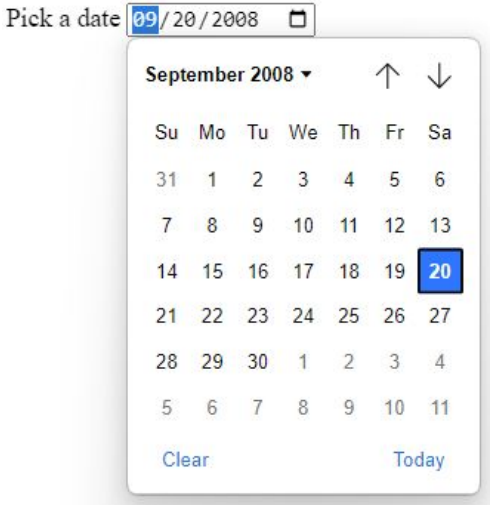
- *Input type email, password, color, date is handled similarly to type=”text”*
- *Input type=”range” (a horizontal slider) is handled similarly to type=”number”*
- *Lazy: bind the value after input lost keyboard focus*

Demo

Color and Date

```
<p>Pick a date <input type="date" v-model="dateStr" /></p>
```



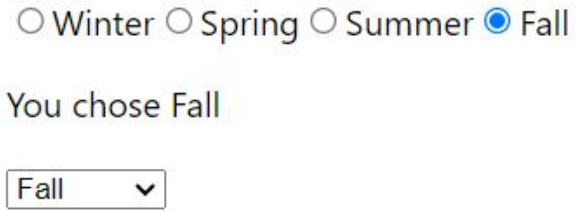



```
<p>Pick a color <input type="color" v-model="hexColorStr" /></p>
```

Demo

Radio buttons & Dropdown List

```
<template>
  <div>
    <input type="radio" id="t0" value="Winter" v-model="season" />
    <label for="t0">Winter</label>
    <input type="radio" id="t1" value="Spring" v-model="season" />
    <label for="t1">Spring</label>
    <input type="radio" id="t2" value="Summer" v-model="season" />
    <label for="t2">Summer</label>
    <input type="radio" id="t3" value="Fall" v-model="season" />
    <label for="t3">Fall</label>
    <p>You chose {{ season }}</p>
    <select v-model="season">
      <option value="Winter">Winter</option>
      <option value="Spring">Spring</option>
      <option value="Summer">Summer</option>
      <option value="Fall">Fall</option>
    </select>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
const season = ref("Fall");
</script>
```



Dropdown menus are handled similar to a radio button

Demo

Radio buttons & data array

```
<template>
  <div>
    <input type="radio" id="t0" value="Winter" v-model="season" />
    <label for="t0">Winter</label>
    <input type="radio" id="t1" value="Spring" v-model="season" />
    <label for="t1">Spring</label>
    <input type="radio" id="t2" value="Summer" v-model="season" />
    <label for="t2">Summer</label>
    <input type="radio" id="t3" value="Fall" v-model="season" />
    <label for="t3">Fall</label>
    <p>You chose {{ season }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
const season = ref("Fall");
</script>
```

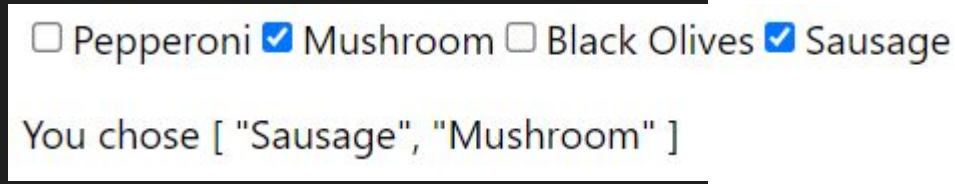
v-for and array source

```
<template>
  <div>
    <template v-for="(s, idx) in allSeasons" :key="idx">
      <input type="radio" :id="`r${idx}`" :value="s" v-model="season" />
      <label :for="`r${idx}`">{{ s }}</label>
    </template>
    <p>You chose {{ season }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
const allSeasons = ref(["Winter", "Spring", "Summer", "Fall"]);
const season = ref("Fall");
</script>
```

Demo

Checkbox

```
<template>
  <div>
    <input type="checkbox" id="c0" value="Pepperoni" v-model="toppings" />
    <label for="c0">Pepperoni</label>
    <input type="checkbox" id="c1" value="Mushroom" v-model="toppings" />
    <label for="c1">Mushroom</label>
    <input type="checkbox" id="c2" value="Black Olive" v-model="toppings" />
    <label for="c2">Black Olives</label>
    <input type="checkbox" id="c3" value="Sausage" v-model="toppings" />
    <label for="c3">Sausage</label>
    <p>You chose {{ toppings }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
const toppings = ref([]);
</script>
```



Demo

Checkbox & data array)

```vue
<template>
  <div>
    <input type="checkbox" id="c0" value="Pepperoni" v-model="toppings" />
    <label for="c0">Pepperoni</label>
    <input type="checkbox" id="c1" value="Mushroom" v-model="toppings" />
    <label for="c1">Mushroom</label>
    <input type="checkbox" id="c2" value="Black Olive" v-model="toppings" />
    <label for="c2">Black Olives</label>
    <input type="checkbox" id="c3" value="Sausage" v-model="toppings" />
    <label for="c3">Sausage</label>
    <p>You chose {{ toppings }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
const toppings = ref([]);
</script>
```

v-for and array source

```vue
<template>
  <div>
    <template v-for="(t, idx) in allToppings" :key="idx">
      <input type="checkbox" :id="`c${idx}`" :value="t" v-model="toppings" />
      <label :for="`c${idx}`">{{ t }}</label>
    </template>
    <p>You chose {{ toppings }}</p>
  </div>
</template>
<script setup lang="ts">
import { ref } from "vue";
const allToppings = ref(["Pepperoni", "Mushroom", "Black Olive", "Sausage"]);
const toppings = ref([]);
</script>
```

Demo

Two-way Data Binding (v-model)

- textbox
- colorpicker
- datepicker
- radio button list
- checkbox list

Demo

```
<script setup lang="ts">
import { ref } from "vue";
const name = ref("Adam");
const age = ref(21);
const hexColorStr = ref("#000");
const dateStr = ref("2018-10-12");
const season = ref("Fall");
const toppings = ref([]);
</script>
```

```
<template>
  <div>
    <p>Your name <input type="text" v-model="name" /></p>
    <p>Pick a date <input type="date" v-model="dateStr" /></p>
    <p>{{ name }} was born in {{ dateStr }}</p>
    <p>
      Pick a number
      <input type="range" v-model.number="age" min="1" max="100" step="2" />
    </p>
    <p>Your age <input type="number" v-model.number="age" /></p>
    <p>Pick a color <input type="color" v-model="hexColorStr" /></p>
    <p>Color <input type="text" v-model.lazy="hexColorStr" /></p>
    <input type="radio" id="t0" value="0" v-model="season" />
    <label for="t0">Winter</label>
    <input type="radio" id="t1" value="1" v-model="season" />
    <label for="t1">Spring</label>
    <input type="radio" id="t2" value="2" v-model="season" />
    <label for="t2">Summer</label>
    <input type="radio" id="t3" value="3" v-model="season" />
    <label for="t3">Fall</label>
    <p>You chose {{ season }}</p>
    <input type="checkbox" id="c0" value="0" v-model="toppings" />
    <label for="c0">Pepperoni</label>
    <input type="checkbox" id="c1" value="1" v-model="toppings" />
    <label for="c1">Mushroom</label>
    <input type="checkbox" id="c2" value="2" v-model="toppings" />
    <label for="c2">Black Olives</label>
    <input type="checkbox" id="c3" value="3" v-model="toppings" />
    <label for="c3">Sausage</label>
    <p>You chose {{ toppings }}</p>
  </div>
</template>
```

Event Handling

Event Handling Directives

<someHTMLTag **v-on:**domEvents=”yourEventHandlingFunction” ...>

<someHTMLTag **@**domEvents=”yourEventHandlingFunction” ...>

```
<div @mouseenter="yourFunctionHere">_____</div>
<img src="bird.png" @click="yourFunctionHere" @wheel="yourOtherFunction">
```

Tons of Event Names

Function Argument Type	Event Names
KeyboardEvent	`keypress, keydown, keyup, key`
`WheelEvent`	`wheel`
`MouseEvent`	`click`
`FocusEvent`	`blur, focus`
`MouseEvent`	`mousedown, mouseenter, mousemove, mouseup`

More Event names

Mouse/Keyboard Events: Filters/Modifiers

```
<template>
  <div>
    <input type="text"
    @keydown.right="showNextPage"
    @keydown.left.alt="showFirstPage" />
    <button @click.shift="goFirst">Start Over</button>
  </div>
</template>

<script setup lang="ts">
function showNextPage() {
  alert('Showing next page');
}

function showFirstPage() {
  alert('Showing first page');
}

function goFirst() {
  alert('Going to the first page');
}
</script>
```

*when **right-arrow** key is pressed*

*when both the **alt** key and the **left-arrow** key are pressed*

*when the **shift** key is held down during the **click***

Filters:

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

Modifiers:

- .alt
- .ctrl
- .meta
- .shift

Demo

Event Handling

- Handle Button Click
- Multiple Event Handlers on One Element

[More Event names]

[Demo]

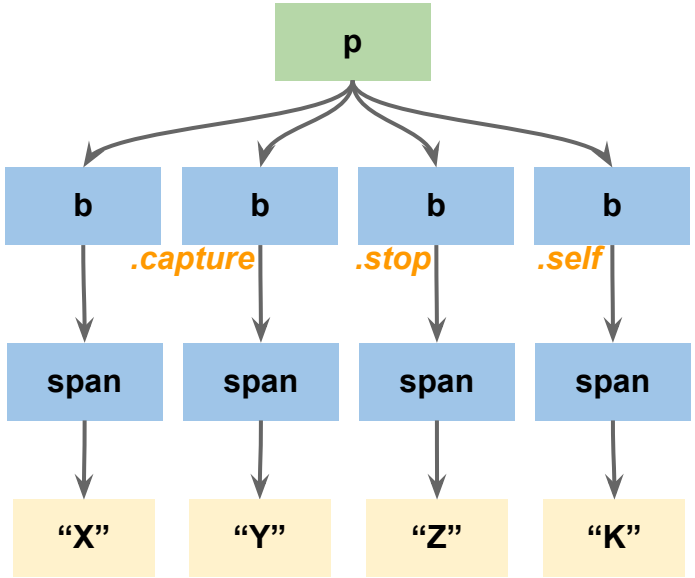
```
<template>
  <h1>Event Handling: Button Click and Mouse Activity</h1>
  <p>Counter is {{ count }}</p>
  <button @click="addOne">More</button>
  <button @click="subtractOne">Less</button>

  <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"
  >
    {{ wheelCount }}
  </div>
</template>
```

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

const count = ref(0);

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

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)

[More Modifiers]



- *Events originating in “X” are handled by span > b > p*
- *Events originating in “Y” are handled by b > span > p*
- *Events originating in “Z” are handled by span > b*
- *Events originating in “K” are handled by span > p*

[Demo]

Passing Arguments to Event Handler

```vue
<template>
  <ul>
    <li v-for="(p, index) in planets" :key="p.name">
      {{ p.name }}
      <button @click="deletePlanet(index)">Delete</button>
      <button @click="show(p.name)">View</button>
      <button @click="showDetails">Details</button>
    </li>
  </ul>
</template>
```

Demo

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

const planets = ref([
  { name: "Mercury", revolution: 87.97 },
  { name: "Earth", revolution: 365.26 },
  { name: "Mars", revolution: 686.68 },
]);

function deletePlanet(index: number) {
  planets.value.splice(index, 1);
}

function show(name: string) {
  alert(`Showing ${name}`);
}

function showDetails(event: MouseEvent) {
  alert(`Showing details of ${event.target}`);
}
</script>
```

Template Refs

```vue
<template>
  <div>
    <h1 ref="bar"></h1>
    <button @click="incrementH1Counter">Plus 1</button>
  </div>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue';
const bar = ref(null);
function incrementH1Counter() {
  bar.value.textContent++;

}

onMounted(() => {
  bar.value.textContent = "3";
});
</script>
```

Online doc

Demo 1

Demo 2

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

In VueJS

One Component ⇒ One .vue File

SFC = Single File Component

Defining and Using Components

ThumbsUp.vue

```
<template>
  <span>👍</span>
</template>
<style>
span {
  font-size: 200%;
}
</style>
```

Sample.vue

```
<script setup lang="ts">
import { ref } from "vue";
import ThumsUp from "./ThumbsUp.vue";
const msg = ref("Hello World!");
</script>

<template>
  <h1>Component Demo</h1>
  <ThumsUp></ThumsUp>
  <h2>{msg}</h2>
</template>
```

Demo

Defining and Passing "Argument"

ThumbsUp.vue

```vue
<template>
  Can you show {{ props.repeat }} thumbs?
  <span v-for="k in props.repeat">👍</span>
</template>
<script setup lang="ts">
type ThumbProp = {
  repeat: number;
};
const props = defineProps<ThumbProp>();
</script>
<style>
span {
  font-size: 200%;
}
</style>
```

```vue
<script setup lang="ts">
import { ref } from "vue";
import ThumbsUp from "./ThumbsUp.vue";
const numThumbsUp = ref(1);
</script>

<template>
  <h1>Component Demo</h1>
  <p>
    Pick a number
    <input type="range" v-model.number="numThumbsUp" min="1" max="20" />
  </p>
  <ThumbsUp :repeat="numThumbsUp"></ThumbsUp>
</template>
```

Sample.vue

Demo

Simple Timer

```
<script setup lang="ts">
import { ref } from "vue";
const seconds = ref(0);
const minutes = ref(0);
let timerInterval: number | null = null; // This variable will hold the interval ID

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() {
  if (!timerInterval) {
    // Check if the timer isn't already running
    timerInterval = setInterval(updateTime, 1000);
  }
}
function stopTimer() {
  if (timerInterval) {
    clearInterval(timerInterval);
    timerInterval = null;
    minutes.value = 0;
    seconds.value = 0;
  }
}
</script>
```

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

Demo

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>
```

set default value

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

Customization of Components Via Props

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

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
```

```
const name: string = ref("");
```

Is this correct?

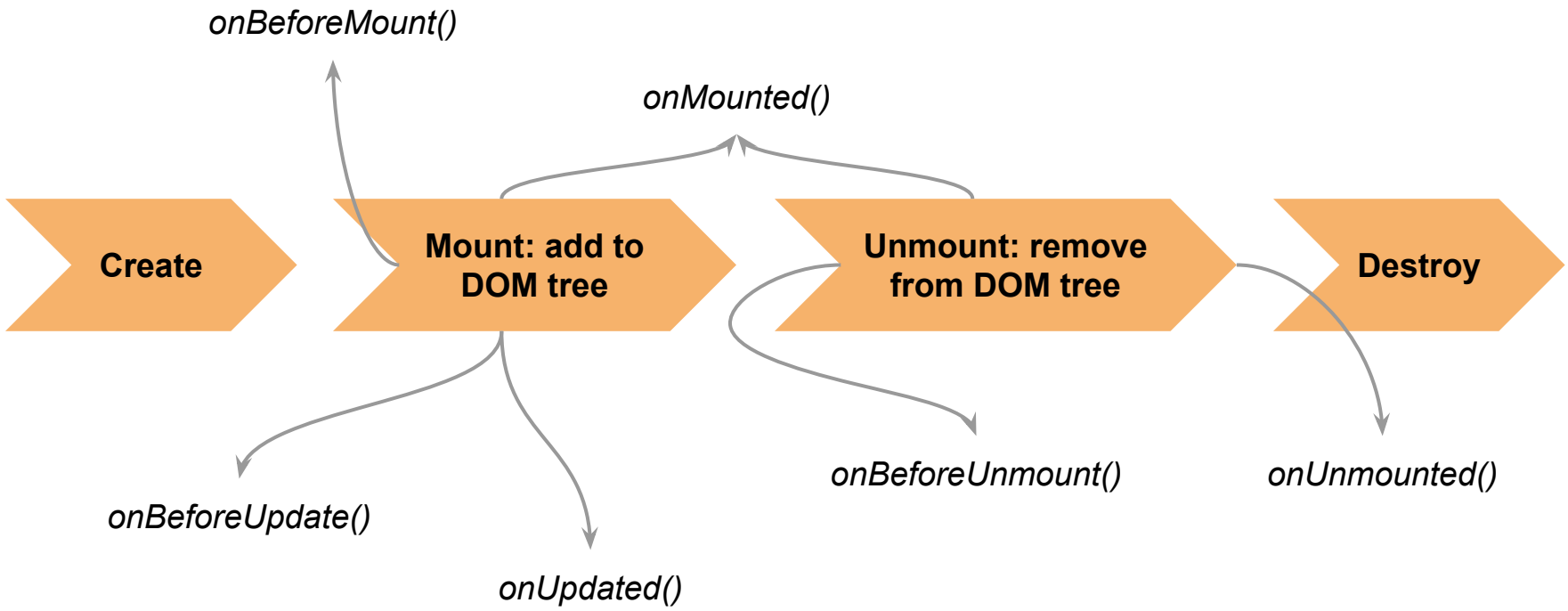
VueJS Reactive Reference + TypeScript Typing

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
```

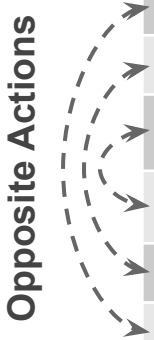
```
import { ref, Ref } from "vue";
const name: Ref<string> = ref("");
const name1 = ref<string>("");
const year: Ref<number> = ref(2001);
const year1 = ref<number>(2001);
const names: Ref<string[]> = ref([]);
const names1 = ref<string[]>([]);
```

Vue3 Lifecycle Functions



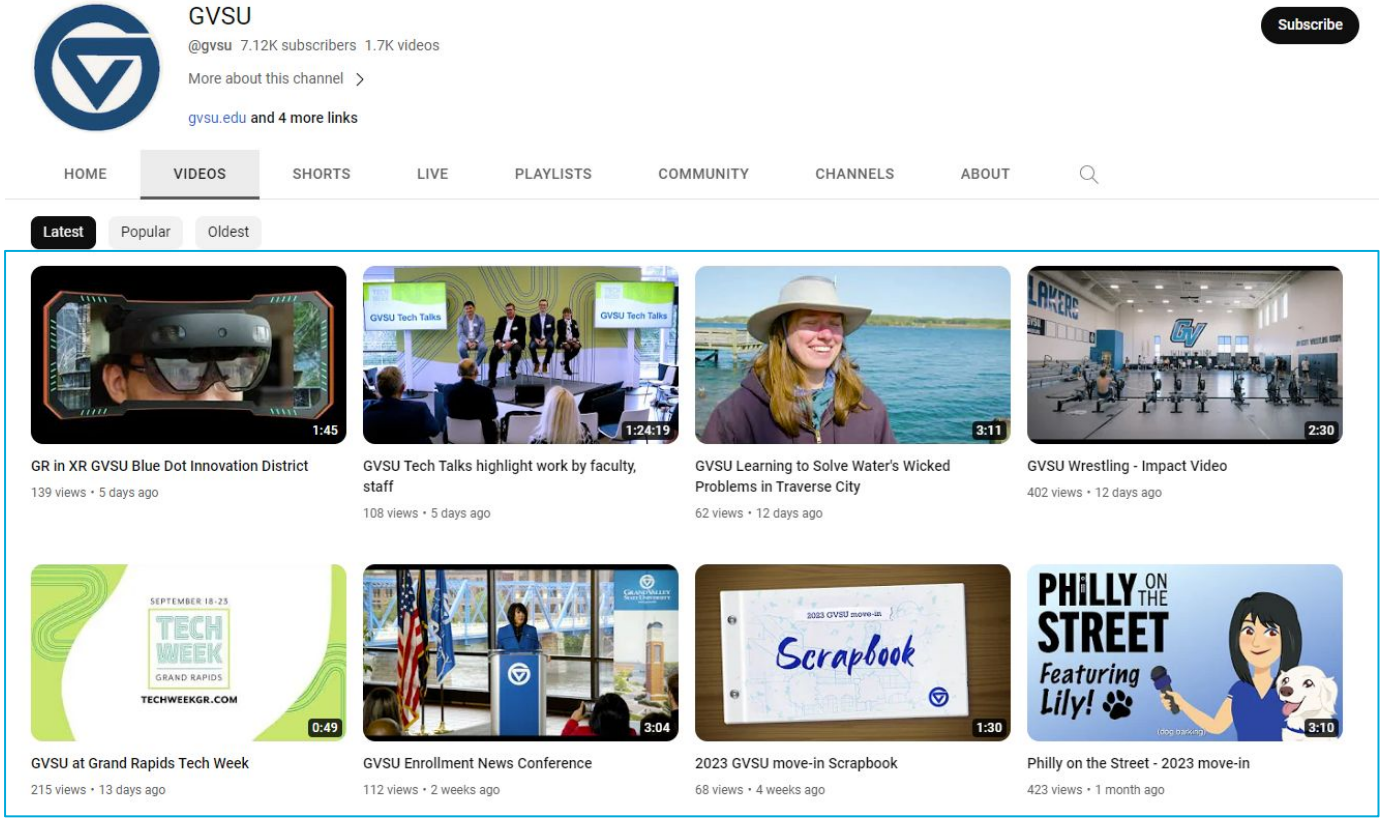
Demo

Practical Use of Lifecycle Hooks



Function	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 preferences

Using Multiple Vue Components


GVSU
@gvsu 7.12K subscribers 1.7K videos
More about this channel
gvsu.edu and 4 more links
Subscribe
HOME
VIDEOS
SHORTS
LIVE
PLAYLISTS
COMMUNITY
CHANNELS
ABOUT
Latest
Popular
Oldest
1:45
GR in XR GVSU Blue Dot Innovation District
139 views • 5 days ago
GVSU Tech Talks
GVSU Tech Talks
1:24:19
GVSU Tech Talks highlight work by faculty, staff
108 views • 5 days ago
3:11
GVSU Learning to Solve Water's Wicked Problems in Traverse City
62 views • 12 days ago
LAKERS
GV
2:30
GVSU Wrestling - Impact Video
402 views • 12 days ago
SEPTEMBER 18-23
TECH WEEK
GRAND RAPIDS
TECHWEEKGR.COM
0:49
GVSU at Grand Rapids Tech Week
215 views • 13 days ago
3:04
GVSU Enrollment News Conference
112 views • 2 weeks ago
2023 GVSU move-in
Scrapbook
1:30
2023 GVSU move-in Scrapbook
68 views • 4 weeks ago
PHILLY ON THE STREET
Featuring Lily!
(dog barking)
3:10
Philly on the Street - 2023 move-in
423 views • 1 month ago

video cover

coverImage

videoDuration

Philly on the Street - 2023 move-in

videoTitle

423 views • 1 month ago

releaseDate

numberOfViews

GR in XR GVSU Blue Dot Innovation District

139 views • 5 days ago

GVSU Tech Talks highlight work by faculty, staff

108 views • 5 days ago

GVSU Learning to Solve Water's Wicked Problems in Traverse City

62 views • 12 days ago

GVSU Wrestling - Impact Video

402 views • 12 days ago

GVSU at Grand Rapids Tech Week

215 views • 13 days ago

GVSU Enrollment News Conference

112 views • 2 weeks ago

2023 GVSU move-in Scrapbook

68 views • 4 weeks ago

Philly on the Street - 2023 move-in

423 views • 1 month ago

src/YouTubeApp.vue

```vue
<template>
  <div>
    <YouTubeCover
      v-for="z in availableVideos"
      :cover-image="z.imageURL"
      :title="z.videoTitle"
      :duration="z.videoDuration"
      :views="z.numberOfViews"
      :release="z.release"
    />
  </div>
</template>
<script lang="ts">
import YouTubeCover from "./components/YTCover.vue";
@Component({ components: { YouTubeCover } })
export default class YouTubeApp extends Vue {
  private availableVideos = [
    /* vide data here */
  ];
}
</script>
```

src/YouTubeApp.vue

```vue
<template>
  <div>
    <YouTubeCover v-for="z in availableVideos" :key="z.id"
    :coverImage="z.imgURL"
    :title="z.videoTitle"
    :duration="z.videoDuration"
    :views="z.numberOfViews"
    :release="z.releaseDate" />
  </div>
</template>

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

const availableVideos = ref([
  {
    id: 1,
    imgURL: 'http://img1',
    videoTitle: 'First Video',
    videoDuration: '12:34',
    numberOfViews: 123456,
    releaseDate: '2021-01-01',
  },
  {
    id: 2,
    imgURL: 'http://img2',
    videoTitle: 'Second Video',
    videoDuration: '5:67',
    numberOfViews: 78910,
    releaseDate: '2021-02-02',
  },
  // more video objects
]);
</script>
```

Parent Component

src/components/YTCover.vue

```vue
<template>
    <div>
        <!-- UI design goes here -->
        <img :src="coverImage" alt="Video cover">
        <h1>{{ title }}</h1>
        <p>Duration: {{ duration }} minutes</p>
        <p>Views: {{ views }}</p>
        <p>Released: {{ release }}</p>
    </div>
</template>
<script setup lang="ts">
type VideoBlock = {
    coverImage: string;
    title: string;
    duration: string;
    views: number;
    release: string;
}
defineProps<VideoBlock>()
</script>
```

Child Component(s)

kebab-case vs. camelCase

kebab-case (in HTML)	camelCase (in TypeScript)
image	`image`
`cover-image`	`coverImage`
`cover-image-url`	`coverImageUrl`