CIS 658 Web Architectures

Vue.js II

Declarative Component-Based UI Framework

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Review

- Vue Fundamentals
  - Why do we need Vue
  - Options API vs. Composition API
  - Create a new Vue App Using vite
  - .vue (SFC = Single File Component)
  - One-Way Data Binding:
  - {{data}}: bind data/var to text nodes
  - 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

Practice

Answer

Topics

- Two-Way Data Binding (v-model) :
  - .lazy, .number,
  - color, and date picker
  - radio buttons & dropdown list & checkbox list
- Event Handling:
  - v-on:domEvents=”function”; @domEvents=”function”
  - Event Names: keypress, keydown, keyup, wheel, click, blur, focus mousedown, mouseenter, mousemove, mouseup, …
- Event Modifiers
  - Key Modifiers: .enter, .tab, .delete, .esc, .space, .up, .down, .left, .right, …
  - System Modifiers: .alt, .ctrl, .meta, .shift,
  - .prevent, .stop, .capture, .self, …
  - .exact modifier
- computed
- watch

Two-Way Data Binding (v-model)

```html
<input type="text" id="nameInput" placeholder="Enter your name" />
<p id="output"></p>

<script type="module" src="./main.js"></script>
```

index.html

If we want an <input> to update a JavaScript variable when the user types, what do we need?

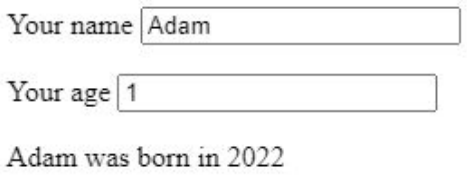
```javascript
// Get references to DOM elements
const nameInput = document.getElementById("nameInput");
const output = document.getElementById("output");

let name = "";
// Listen for user typing
nameInput.addEventListener("input", () => {
  // Update variable from input
  name = nameInput.value;
  // Update UI from variable
  output.textContent = `Hello, ${name}`;
});
```

main.js

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>
```

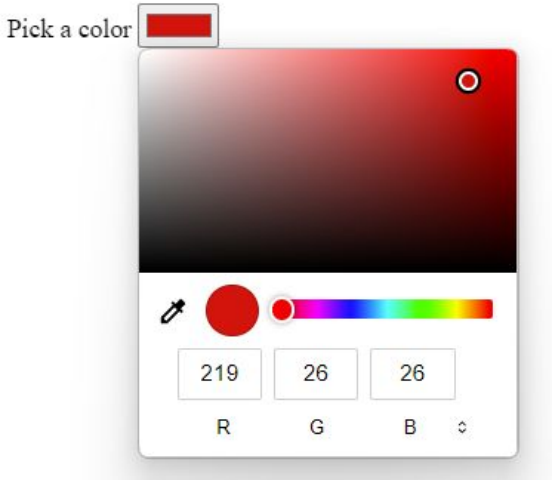


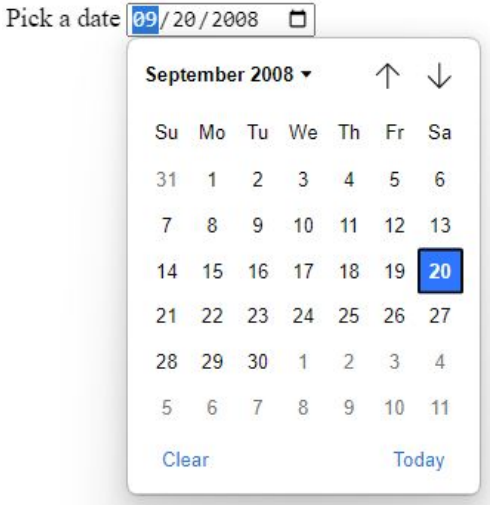
- *Input type email, password, color, date is handled similarly to type=”text”*
- *Input type=”range” (a horizontal slider) is handled similarly to type=”number”*
- *Lazy: bind the value after input lost keyboard focus*

Demo

Color and Date

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



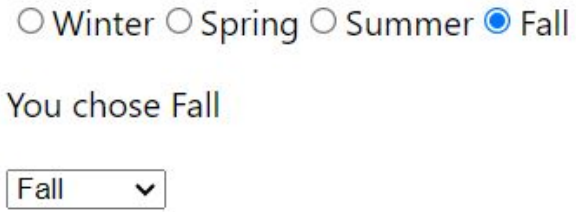



```
<p>Pick a color <input type="color" v-model="hexColorStr" /></p>
```

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>
```

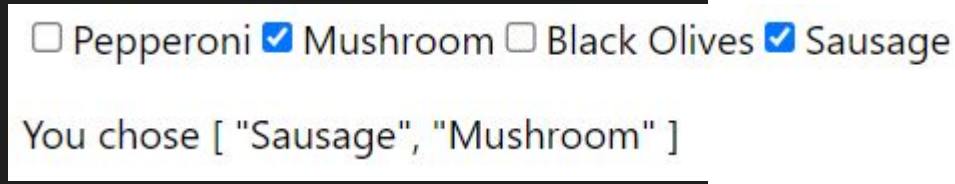
v-for and array source

```
<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

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 Events: 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*

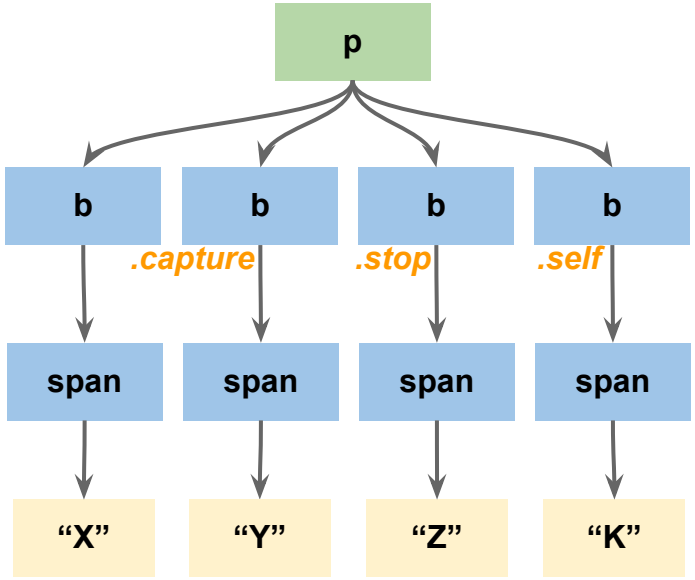
*when both the **alt** key and the **left-arrow** key are pressed*

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

Demo

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*
- *Events originating in “Z” are handled by*
- *Events originating in “K” are handled by*

Demo

More Modifiers

More Modifiers

Key Modifiers

- .enter
- .tab
- .delete: ("Delete" & "Backspace")
- .esc
- .space
- .up
- .down
- .left
- .right

Mouse Modifiers

- .left
- .right
- middle

System Modifiers

- .alt
- .ctrl
- .shift
- .meta: command key (⌘) or Windows key (⊞) or solid diamond (◆)

```
<!-- this will fire even if Alt or Shift is also pressed -->
<button @click.ctrl="onClick">A</button>

<!-- this will only fire when Ctrl and no other keys are pressed -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- this will only fire when no system modifiers are pressed -->
<button @click.exact="onClick">A</button>
```

.exact modifier

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

```vue
<template>
  <div>
    <p>First name <input type="text" v-model="first" /></p>
    <p>Last name <input type="text" v-model="last" /></p>
    <p>My name is {{ first }} {{last}}</p>
  </div>
</template>
<script setup lang="ts">
  import { ref } from "vue";
  const first = ref("John");
  const last = ref("Smith");
</script>
```

If we change the first name in the text input, will fullName update automatically?

Demo

```vue
<template>
  <div>
    <p>First name <input type="text" v-model="first" /></p>
    <p>Last name <input type="text" v-model="last" /></p>
    <p>My name is {{ fullName }}</p>
  </div>
</template>
<script setup lang="ts">
  import { ref } from "vue";
  const first = ref("John");
  const last = ref("Smith");
  const fullName = ref(first.value + " " + last.value);
</script>
```

computed

- computed is read-only derived state
- It automatically tracks dependencies
- It recalculates only when needed

computed is for pure calculation, no side effects.

Demo

```
<template>
  <div>
    <p>First name <input type="text" v-model="first" /></p>
    <p>Last name <input type="text" v-model="last" /></p>
    <p>My name is {{ first }} {{last}} </p>
    <p>My name is {{ fullName }} </p>
  </div>
</template>
<script setup lang="ts">
import { ref, computed} from "vue";
const first = ref("John");
const last = ref("Smith");
const fullName = computed(() => {
  return first.value + ' ' + last.value
})
</script>
```

watch

If computed is for pure derived state, how do we run code when something changes?

Demo

Practice: Two-Way Data Binding (v-model)