Class 3: Events & Reactive Variables

1. Review: App Prototypes
2. Event Handling
3. Vue.js Troubleshooting
4. Reactive Variables
5. Event Handling (Part II)

Review: Prototype

> A user interface prototype is a hypothesis — a candidate design solution that you consider for a specific design problem.

- NN/Group

Review: Prototyping Methods

- Low-Fidelity (e.g. Paper Prototypes, Sketches, Wireframes)
  - lower cost
  - less interactive (often static)
- Medium-Fidelity (e.g. Digital Wireframes, Clickable Prototypes)
  - moderate cost
  - somewhat interactive
- High-Fidelity (e.g. Interactive/Functional Prototypes)
  - higher cost
  - more interactive (often dynamic)

Review: Vue.js

> Vue.js is an approachable, performant and versatile framework for building interactive **web-based** (HTML, CSS, JavaScript) user interfaces.

- https://vuejs.org/

Demo: Development Environment

1. Launch Codespace
2. Launch Development Server
3. Open Website (pop-up blockers; Ports view)
4. Stop Development Server (or **control** + **C** in terminal)

Vue.js

Discussion: What did you learn about Vue.js?

For today's class preparation, you independently explored Vue.js using your Homework 1 repository.

Share some of your observations and discoveries.

Single Page Applications (SPAs)

A Vue.js app is a **Single Page Application (SPA)**.

> An SPA (Single-page application) is a web app implementation that loads only a single web document (i.e. HTML), and then updates the body content of that single document via JavaScript [...]

Source: https://developer.mozilla.org/en-US/docs/Glossary/SPA

Components – `.vue` Files

- Each `.vue` file is a **component**.
- Each component has:
  - `<script>`: JavaScript logic
  - `<template>`: HTML structure
  - `<style>`: CSS styling

`App.vue`:

```
<script setup>
</script>

<template>
  <button>Click Me!</button>
</template>

<style scoped>
button {
  font-size: 1.5em;
  padding: 0.5em 1em;
}
</style>
```

Events

Review: React Events

```
function App() {
  return (
    <button onClick={() => console.log('Button clicked!')}>
      Click Me!
    </button>
  )
}
```

Vue.js: Event Handling

The `v-on` directive listens for DOM events and runs JavaScript code when they're triggered.

`v-on` Directive

```
<script setup>
function buttonClicked() {
  console.log('Button clicked!')
}
</script>

<template>
  <button v-on:click="buttonClicked">Click Me!</button>
</template>
```

`@` Shorthand

```
<script setup>
function buttonClicked() {
  console.log('Button clicked!')
}
</script>

<template>
  <button @click="buttonClicked">Click Me!</button>
</template>
```

Activity: Event Practice

1. Open your Homework 1 repository (if you submitted already) **or** today's activity repository.
2. Add a click event handler to **a** button in your app.

```
<button v-on:click="buttonClicked">Click Me!</button>
```

3. The handler should play a sound (e.g. `rain.mp3`) when the button is clicked.

```
const audio = new Audio()
audio.src = '/sounds/rain.mp3'
audio.play()
```

Tip: Public domain sounds - https://www.wnoise.org/white-noise-sounds.html

Troubleshooting Vue.js Apps

Syntax Errors

Syntax errors are shown in the **Debug Console** (or Terminal if you launched via `npm run dev`).

1. Vite compiles Vue.js to an SPA.
2. Vite runs on the local development server (e.g., `http://localhost:5173/`).
3. If there is a syntax error, Vite shows the error in the console.

Runtime Errors

Single Page Applications (SPAs) run in the **browser**.

Where should you check for errors in client-side code?

Browser DevTools Console!

All I see is a blank page!

You have an error in your code.

1. Check your Codespace's **Debug Console** for syntax errors.
2. Check your browser's **DevTools Console** for runtime errors.

Reactive Variables

Review: React State + Expression Placeholders

```
import { useState } from 'react'

function App() {
  const [count, setCount] = useState(0)

  function incrementCount() {
    setCount(count + 1)
  }

  return (
    <>
      <button onClick={incrementCount}>
        Increment
      </button>
      <span>{count}</span>
    </>
  )
}
```

Vue.js: Reactive Variables

Reactive variables in Vue.js are variables that are automatically tracked for changes. When a reactive variable is updated, the SPA is automatically re-rendered to reflect the new value.

Create a reactive variable using the `ref()` function and initialize it with a value.

```
<script setup>
import { ref } from 'vue'
const name = ref('Alice')
</script>
```

Vue.js: Reactive Variables (cont.)

Change the value of a reactive variable by assigning a new value to its `.value` property.

```
name.value = 'Bob'
```

Access the value of a reactive variable by reading its `.value` property.

```
console.log(name.value) // 'Bob'
```

Example: Reactive Variable + Interpolation

```
<script setup>
import { ref } from 'vue'
const count = ref(0)

function incrementCount() {
  count.value++
}
</script>

<template>
  <button @click="incrementCount">
    Increment
  </button>
  <span>{{ count }}</span>
</template>
```

The `{{` and `}}` are **interpolation operators** that display the value of a reactive variable in the SPA.

Gotcha: You must use `.value` to read or write a reactive variable in the `<script>` section, but you do **not** use `.value` in the `<template>` section.

Activity: Tracing Reactive Variables

Working with your **peer**, complete the **handout**.

Minimum Working Example

A minimum working example is the smallest amount of code that demonstrates a concept or problem.

Examples:

- A Vue.js component
- Event handling
- Reactive variables

Activity: Minimum Working Example

Working with your **peer**, create a **minimum working example** that demonstrates a reactive variable and an event handler.

1. In your Homework 1 repository (if you submitted already) **or** today's activity repository, add a reactive variable to your app that tracks the current sound.
2. Display the current sound in your app using an interpolation operator.

Event Handling (Part II)

DOM Events

Any event that occurs in the DOM can be listened for using Vue.js event handling.

- `v-on:click` / `@click` - Mouse click
- `v-on:mouseover` / `@mouseover` - Mouse hover
- `v-on:keyup` / `@keyup` - Key press
- `v-on:input` / `@input` - Input value change
- `v-on:change` / `@change` - Input value change

Reference Documentation: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Events

Gotcha: Event Handlers

Handler

```
v-on:click="incrementCount"
```

Inline Code

```
v-on:click="count++"
```

Not a Handler:

```
v-on:click="incrementCount()"
```

What's Next

- Due Wednesday, 9/2/2026, 11:69PM: Class 4, Preparation