# Chapter 4 – Class and Style Binding

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

# Scoped Styles

```
<template>
   …
</template>
<script>
   …
</script>
<style>
   style affect every template in every files
</style>
```

// global scope

```
<style>
#app {
  padding: 32px;
  margin: 0 auto;
  width: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
}
</style>
```

```
<style scoped>
 scoped style to apply only this component
defined it and affect only this template
</style>
```

// local scope

```
<style scoped>
.other {
  font-size: 3rem;
  color: purple;
}
</style>
```

# Class and Style Bindings

- A common need for data binding is manipulating an element's class list and its inline styles.
- Since they are both attributes, we can use `v-bind` to handle them: we only need to calculate a final string with our expressions.
- However, meddling with string concatenation is annoying and error-prone.
- For this reason, Vue provides special enhancements when `v-bind` is used with class and style.
- In addition to strings, the expressions can also evaluate to *objects* or *arrays*.

# Styles

# Binding HTML Style

- `:style` supports binding to JavaScript object values - it corresponds to an HTML element's style property:

```
<script setup>
const dark = { 'background-color': 'black', color: 'white' }
</script>
```

```
 <div :style="dark">
    …
 </div>
```

# Binding HTML Style (Object Syntax)

- `:style` supports binding to JavaScript object values - it corresponds to an HTML element's style property:

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

```
<div
      :style="
      darkTheme
        ? { 'background-color': 'black', color: 'white' }
        : { 'background-color': 'white', color: 'black' }"
  >
    …
</div>
```

Although camelCase keys are recommended, :style also supports kebab-case (use quotes with kebab-case) for the CSS property names

# Binding HTML Style (Array Syntax)

The array syntax for `:style` allows you to apply multiple style objects to the same element:

```
<div
    :style="
    darkTheme
      ? [{ 'background-color': 'black' }, { color: 'white' }]
      : [{ 'background-color': 'white' }, { color: 'black' }]
  "
>
    …
</div>
```

# Binding HTML Style (Array Syntax)

It can be a bit verbose if you have multiple conditional classes. That's why it's also possible to use the object syntax inside array syntax:

```
<div
    :style="[
      darkTheme
        ? { 'background-color': 'black', color: 'white' }
        : { 'background-color': 'white', color: 'black' },
      {'font-style':'italic' }
    ]"
  >
    …
  </div>
```

# Classes

# Binding HTML Class

- We can bind :`class` to apply a list of classes:

```
<style scoped>
.dark {
  background-color: black;
  color: white;
}
.light {
  background-color: white;
  color: black;
}
</style>
```

```
  <div :class="darkTheme ? 'dark' : 'light'">
        …
  </div>
```

```
  <div :class="darkTheme ? 'dark tracking-widest' : 'light'">
        …
  </div>
```

# Binding HTML Class (Object Syntax)

- You can have multiple classes toggled by having more fields in the object. In addition,

```
<style scoped>
.dark {
  background-color: black;
  color: white;
}
.light {
  background-color: white;
  color: black;
}
</style>
```

```
  <div
     :class="darkTheme ? { dark: true } : { light: true }"
  >
     …
  </div>
```

# Binding HTML Class (Array Syntax)

- The `:class` directive can also co-exist with the plain class attribute.
- We can bind `:class` to an array to apply a list of classes:

```
<style scoped>
.dark {
  background-color: black;
  color: white;
}
.light {
  background-color: white;
  color: black;
}
</style>
```

```
<div
        class="text-sm font-semibold tracking-wider p-2"
        :class="darkTheme ? ['dark', 'text-white'] : ['light', 'text-black']"
>
    …
</div>
```

# v-bind() in CSS

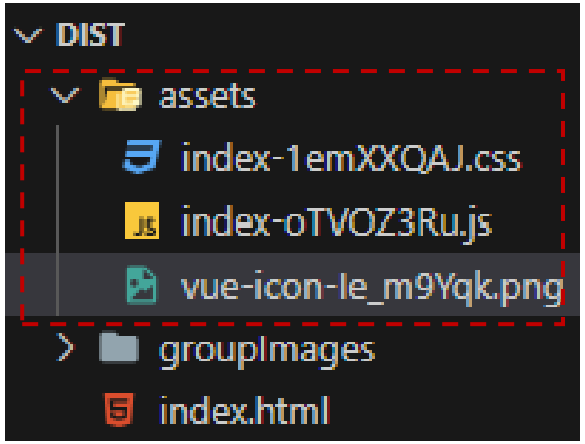
- SFC `<style>` tags support linking CSS values to dynamic component state using the `v-bind` CSS function

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

<template>
  Your Color: <input type=“color" v-model= "yourColor" />
  <p class="specialText">Special Text</p>
</template>

<style>
.specialText {
  color: v-bind(yourColor);
}
</style>
```

# Assets Handling

# Static Assets

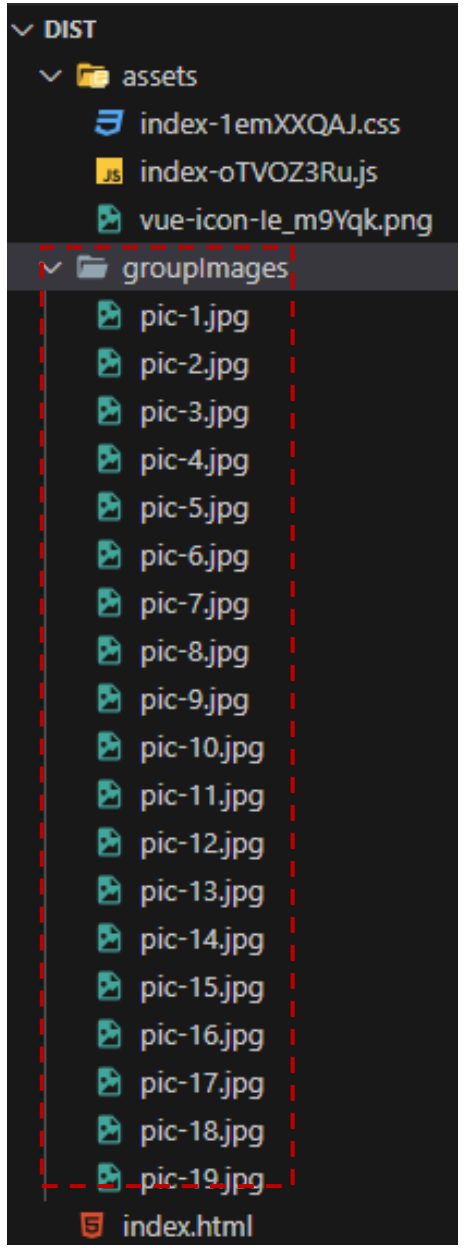
- If you have assets that refer  in source code (e.g. the file `vue-icon.png`)
- put your assets under `assets` folder, for example, `./assets/vue-icon.png`
- during development the file `./assets/vue-icon.png` will become `"/assets/vue-icon-Ie_m9Yqk.png"` in the production build.




https://vitejs.dev/guide/assets

# Dynamic Assets

- If you have assets that are:
  - Never referenced in source code (`e.g. robots.txt`) then you can place the asset in a special `public` directory under your project root.
  - Assets in `public` directory will be served at root path / during dev, and copied to the root of the **dist** directory as-is.
- Note that:
  - you should always reference public assets using root absolute path - for example, `public/icon.png` should be referenced in source code as `/icon.png` or `public/groupImages/pic-1.jpg` should be referenced in source code as `/groupImages/pic-1.jpg`
  - Assets in `public` cannot be imported from JavaScript.




https://vitejs.dev/guide/assets#the-public-directory

# Template Ref

https://vuejs.org/guide/essentials/template-refs#template-refs

# Template Refs

- While Vue's declarative rendering model abstracts away most of the direct DOM operations for you, there may still be cases where we need direct access to the underlying DOM elements. To achieve this, we can use the special `ref` attribute:

```
<input ref="input">
```

- To obtain the reference with Composition API, we need to declare a `ref` with the same name. It allows us to obtain a direct reference to a specific DOM element or child component instance after it's mounted.
- If you try to access input before mounted, it will be `null` on the first render.

# The <audio> element

- <audio> HTML element used to play an audio file on a web page.

```
<audio controls>
  <source src="sample.mp3" type="audio/mp3" />
  <p>
    Your browser doesn't support this audio file. Here is a
    <a href="sample.mp3">link to the audio</a> instead.
  </p>
</audio>
```

- The controls attribute adds audio controls, like play, pause, and volume.


- The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element.

https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content#the_audio_element

# Build Custom Audio Player

```
<script setup>
const isPlaying = ref(false)
const player = ref('')
const musicControl = () => {
  isPlaying.value = !isPlaying.value
  if (isPlaying.value) player.value.play()
  else player.value.pause()
}
</script>
```

```
<template>
 <audio controls class="hidden" ref="player">
    <source src="./assets/sample.mp3" type="audio/mp3" />
 </audio>
</template>
```