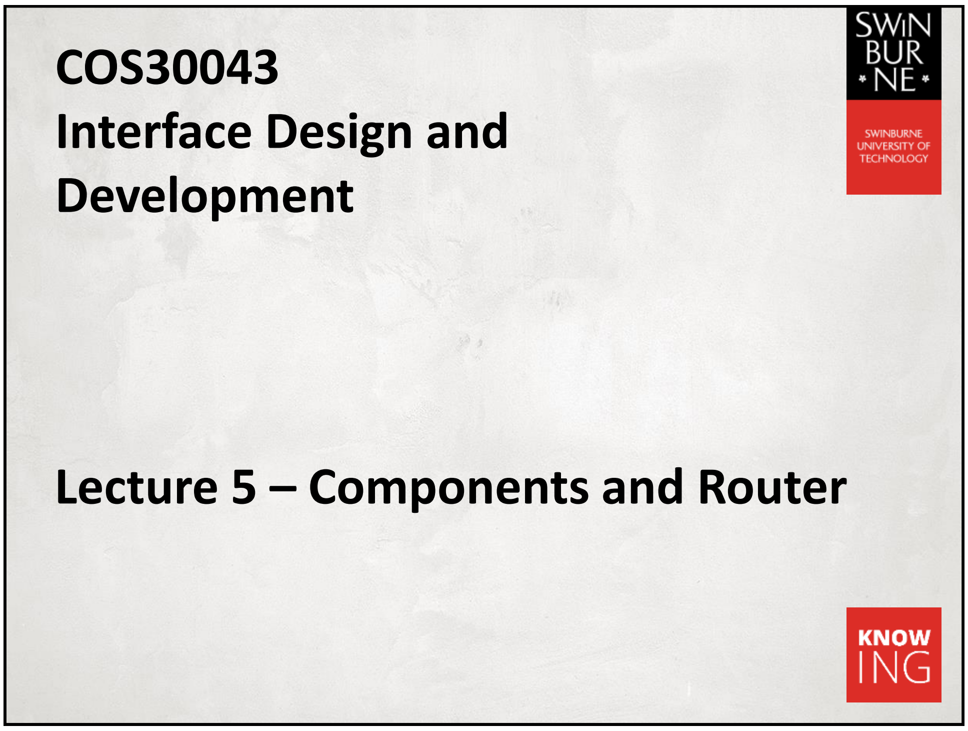
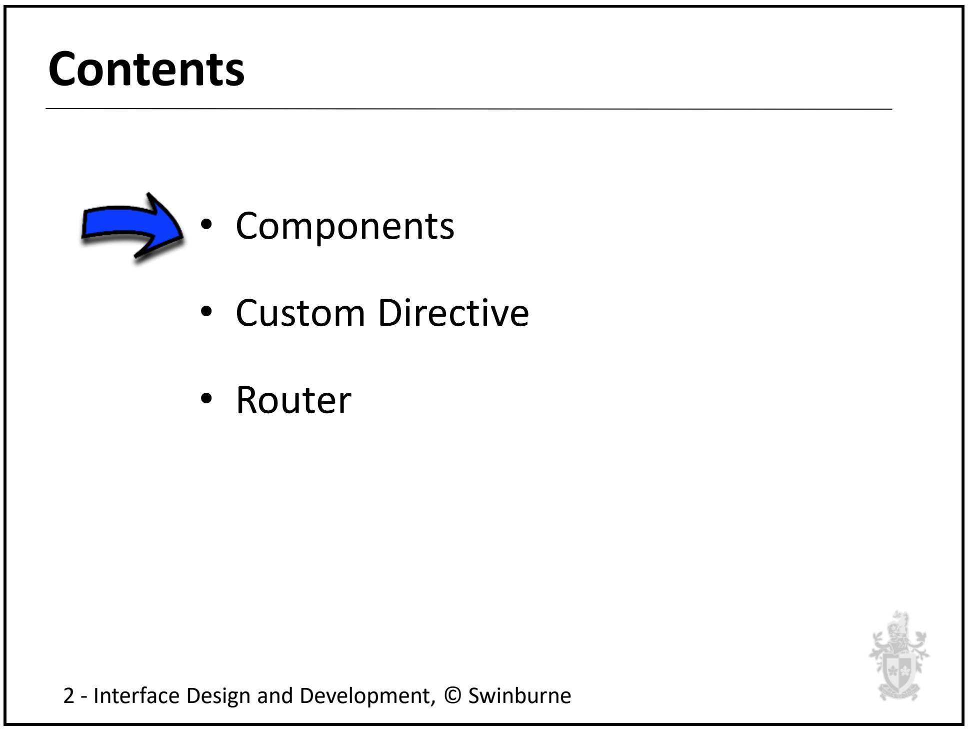


COS30043 Interface Design and Development

Lecture 5 – Components and Router

KNOW ING

1

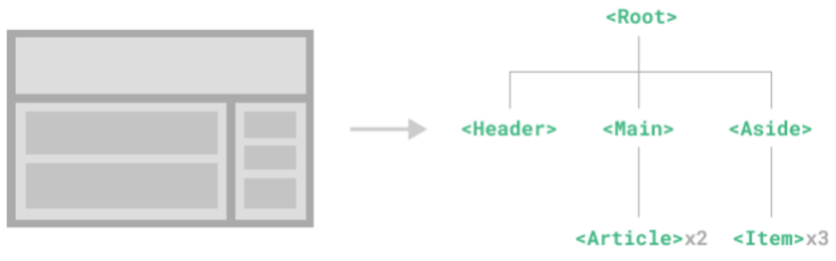


Contents

- Components
- Custom Directive
- Router

2

Components



Component (Use)

- Components are one of the most powerful features of Vue.js.
- They help to extend basic HTML elements to encapsulate reusable code.
- Components are custom elements that Vue.js' compiler would attach specified behavior to.

```
<div id="app">
    <example_component>
        </example_component>
</div>
```

Component (Use)

- Components are reusable Vue instances with a name
- Components can be reused as many times as you want, for example

```
<div id="app">
      <example_component></example_component>
      <example_component></example_component>
      <example_component></example_component>
</div>
```

Component (Declare)

```
const app = Vue.createApp({   })
app.component('example_component’,
{
      data (){// shorthand
            return {
                  // declare properties
            }
      },

      template:
      // declare methods


})
app.mount('#app')
```

Component name accessed in the view (HTML) as an HTML element

Component (Declare)

```
app.component('example_component’,
{
      data:function(){
            return {
                  // declare properties
            }
      },
      template:
      // declare methods
});
```

/* a component’s data option must be a function, so that each instance can maintain an independent copy of the returned data object
*/

7

Component Example

JavaScript:

```
const app = Vue.createApp({   })
app.component('example_component',
{
      data: function(){
            return  {msg: 'Hello Mr. Chua'}
      },
      template: '<p>{{ msg }} </p>'
});
app.mount('#app’)
```

HTML:

```
<div id="app" >
 <example_component> </example_component>
</div>
```

8

Passing Data to Child Components with Props

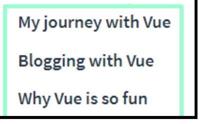
Props are custom attributes you can register on a component. When a value is passed to a prop attribute, it becomes a property on that component instance.

```
app.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

Once a prop is registered, you can pass data to it as a custom attribute:

```
<blog-post title="My journey with Vue"></blog-post>
<blog-post title="Blogging with Vue"></blog-post>
<blog-post title="Why Vue is so fun"></blog-post>
```

Here we pass text, do not use “v-bind:title” or “:title”
To pass a javascript expression, use “v-bind:” or “:”


My journey with Vue

Blogging with Vue

Why Vue is so fun

Passing Data to Child Components with Props

If you want to pass a javascript expression to the child compoment, you can use “v-bind:” or simply “:” before the attribute.

Component declare (in JavaScript):

```
app.component('myComp', {
 props: ['fruits'],
 template: '<ul><li v-for="f in fruits">{{ f }}</li></ul>'
})
```

Component use (in HTML):

```
<myComp v-bind:fruits="['apple','orange','grape']"></myComp>
Or
<myComp :fruits="['apple','orange','grape']"></myComp>
```

To pass a javascript expression, use “v-bind:” or “:”
To pass text, do not use “v-bind:” or “:”

Contents

- Components
- Custom Directive
- Router

Ways to Declare and Use a Directive

- Use directive as attribute

  `<span v-mydirective></span>`
- Declare it globally using

  ```
  app.directive('mydirective',
  { {
        // code
        } })
  ```
- Declare it locally using component. Components accept a “directives” option

Parts of a Directive

- Directive name
- Hook functions(optional)

  created - called before bound element's attributes or event listeners are applied

  beforeMount - called right before the element is inserted into the DOM.

  mounted - called when the bound element's parent component and all its children are mounted.

  beforeUpdate - called before the parent component is updated

  updated - called after the parent component and all of its children have updated

  beforeUnmount - called before the parent component is unmounted

  unmounted - called when the parent component is unmounted
- Hook arguments

13

Creating a Directive Object

- Creating a directive

```
const app = Vue.createApp({})
app.directive('highlight', {
      created(el, binding, vnode) {
      el.style.backgroundColor = 'lightgreen'
      }
})
app.mount('#app’)
```

N.B. directives can also be registered locally by defining them inside the components option. This is an example where a directive is registered globally.

14

Using a directive

```
<html>
:
<body id="app">
<p v-highlight> Colour changed to green</p>

</body>
</html>
```

15 - Interface Design and Development, © Swinburne

Custom Directive 1 (Use)

Passing value to a custom directive

```
<div v-highlight="'red'">
</div>
```

- This enables you to create a library of directives that you can use across different web apps.

16 - Interface Design and Development, © Swinburne

Custom Directive 1 (Declare)

```
app.directive('highlight', {
       created(el, binding, vnode) {
       el.style.backgroundColor = binding.value
       }
})
```

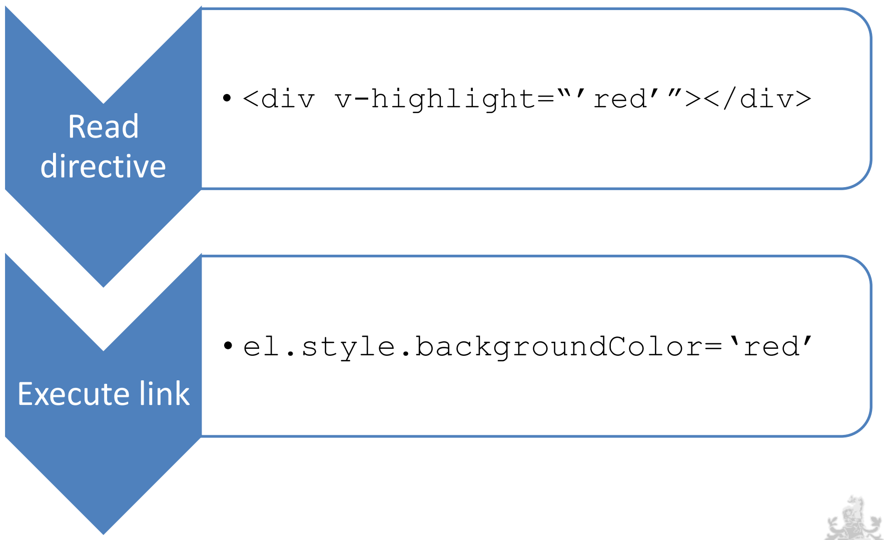
el: The element. This can be used to directly manipulate the DOM.

vnode: A virtual DOM tree produced by Vue's compiler. It is a JavaScript data structure that describes a DOM tree.

binding: An object containing the following properties.
- **name**: The name of the directive, without the v- prefix.
- **value**: The value passed to the directive. For example in v-my-directive="1 + 1", the value would be 2.
- **expression**: The expression of the binding as a string. For example in v-my-directive="1 + 1", the expression would be "1 + 1".
- … …

17

Execution of a Directive



- <div v-highlight="'red'"></div>
- el.style.backgroundColor='red'

18

Custom Directive 2 (Use)

Passing Objects as values

```
<div
  v-highlight = “{colour: ‘red’,
font: ‘italic’}”>
</div>
```

- method evaluates the string as an expression to create the array.

Custom Directive 2 (Declare)

```
app.directive('highlight', {
   created(el, binding, vnode) {
      el.style.backgroundColor = binding.value.color
      el.style.border = binding.value.border
      el.style.fontStyle = binding.value.font
   }
})
```

Execution of a Directive

Read directive

- `<div v-highlight="{colour: 'red', font: 'italic'}"></div>`

Execute link

- `el.style.backgroundColor = binding.value.colour;`
- `el.style.fontStyle = binding.value.font;`

Contents

- Components
- Custom Directive
- Router

Link to library

Link to vue and vue-router:

```
<script src="js/vue.min.js"> </script>
<script src="js/vue-router.js"></script>
```

You can download Vue Router from
https://router.vuejs.org/installation.html

Router (Use)

In HTML:

```
<div id=“app”>
     <!-- links to template -->
     <nav>
          <router-link to="<destination>">
          Menu Option</router-link>
     </nav>

     <!-- div to display the component -->

     <router-view></router-view>
</div>
```

Router (Use) Example

In HTML:

```
<div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- use the router-link for navigation. -->
    <router-link to="/">Home</router-link>
    <router-link to="/about">About</router-link>
  </p>

  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```

Router (Declare)

In JavaScript:

```
1. Define route components
const Home = { … }
// 2. Define routes. Each route should map to a
component.
const routes = [  …  ]
// 3. Create the router instance and pass the
`routes` option
const router = VueRouter.createRouter({
  history: VueRouter.createWebHashHistory(),
  routes, // short for `routes: routes`
})
// 4. Create and mount the root instance.
const app = Vue.createApp({})
app.use(router)
app.mount('#app')
```

Router (Declare) Example

In JavaScript:

```
// 1. Define route components
const Home = { template: '<div>Home</div>' }
const About = { template: '<div>About</div>' }
// 2. Define routes. Each route should map to a component.
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
]
// 3. Create the router instance and pass the `routes` option
const router = VueRouter.createRouter({
  history: VueRouter.createWebHashHistory(),
  routes, // short for `routes: routes`
})
// 4. Create and mount the root instance.
const app = Vue.createApp({})
app.use(router)
app.mount('#app')
```

27

Dynamic Route with Param (example 1)

router-link:

```
<ul v-for="c in cities">
       <!-- use the router-link to create navigation. -->
       <li><router-link :to="{ path:'/city/'+c }">
                     {{c}}</router-link>  </li>
</ul>
```

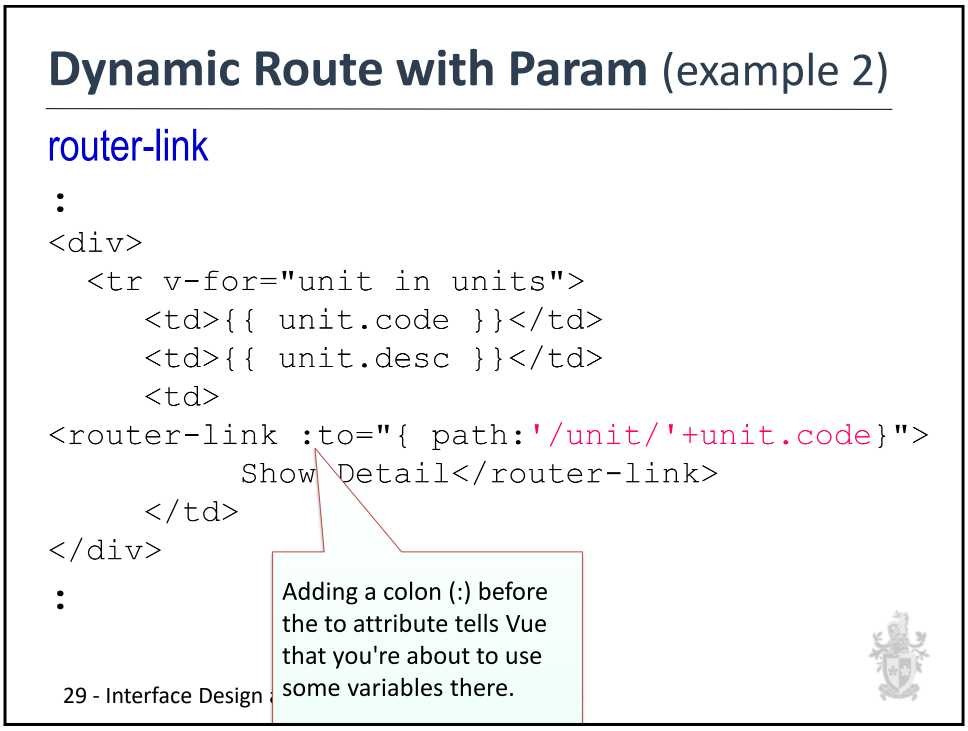
component (When a route is matched, the value of the dynamic segments will be exposed as $route.params)

```
const City = {
   template:
    '<div>{{$route.params.id}} is a great city. </div>'
}
```

routes (A dynamic segment is denoted by a colon :)

```
const routes = [
       { path: '/city/:id', component: City },
]
```

28



Dynamic Route with Param (example 2)

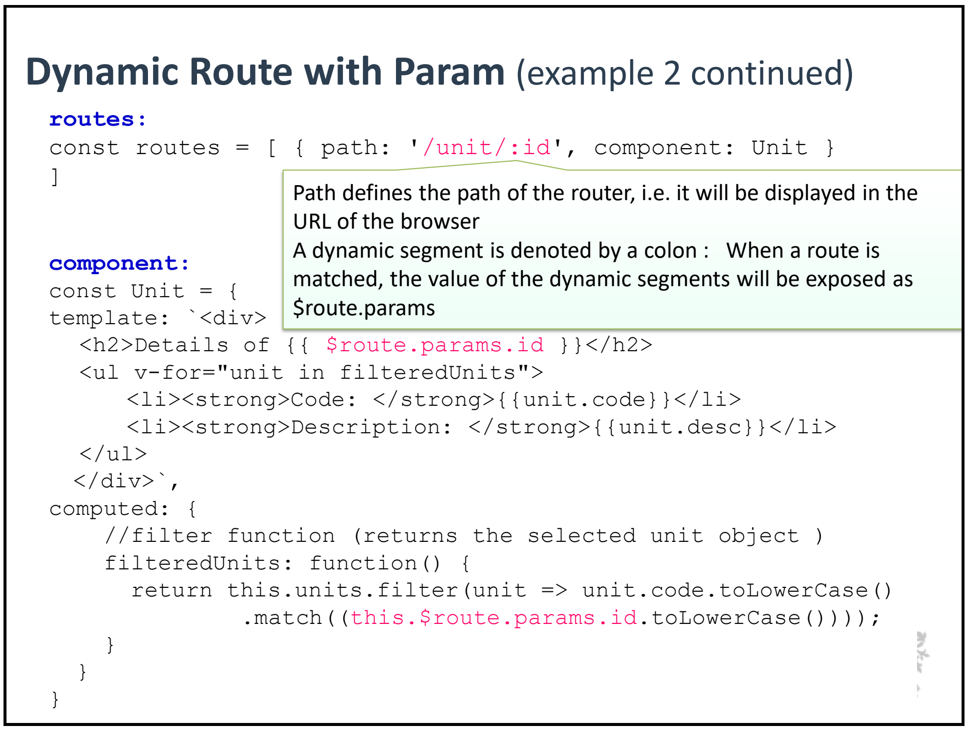
router-link

```
:
<div>
  <tr v-for="unit in units">
     <td>{{ unit.code }}</td>
     <td>{{ unit.desc }}</td>
     <td>
<router-link :to="{ path:'/unit/'+unit.code}">
          Show Detail</router-link>
     </td>
</div>
:
```

Adding a colon (:) before the to attribute tells Vue that you're about to use some variables there.

29 - Interface Design

29



Dynamic Route with Param (example 2 continued)

routes:

```
const routes = [ { path: '/unit/:id', component: Unit }
]
```

Path defines the path of the router, i.e. it will be displayed in the URL of the browser
A dynamic segment is denoted by a colon : When a route is matched, the value of the dynamic segments will be exposed as $route.params

component:

```
const Unit = {
template: `<div>
  <h2>Details of {{ $route.params.id }}</h2>
  <ul v-for="unit in filteredUnits">
     <li><strong>Code: </strong>{{unit.code}}</li>
     <li><strong>Description: </strong>{{unit.desc}}</li>
  </ul>
  </div>`,
computed: {
    //filter function (returns the selected unit object )
    filteredUnits: function() {
      return this.units.filter(unit => unit.code.toLowerCase()
              .match((this.$route.params.id.toLowerCase())));
    }
  }
}
```

30

WHAT'S NEXT?
- FORMS AND VALIDATION

31