Programação Web I

M01T04 - Introduction to Vue.js (Components)

ESMAD | TSIW | 2025-26

M01 - Introduction to Vue.js

T04 - Components

1. Components
2. Props
3. Events
4. Slots

M01 - Introduction to Vue.js

T04 - Components

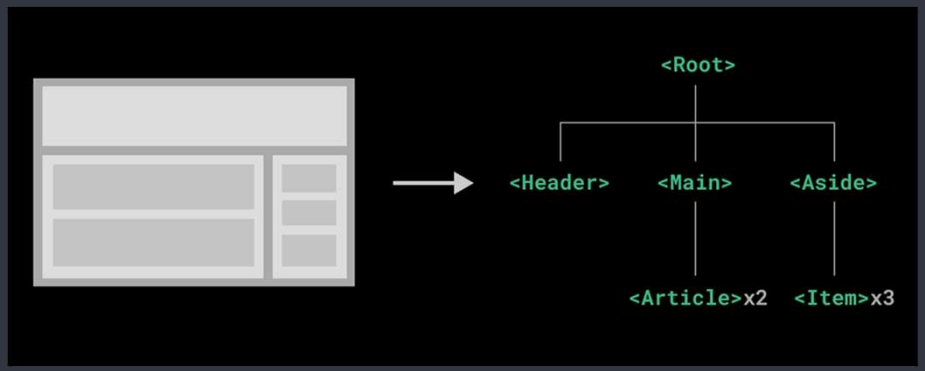
1. Components
2. Props
3. Events
4. Slots

M01 - Introduction to Vue.js

1. Components

- Components allow us to split the UI into independent and reusable pieces, and think about each piece in isolation

- It's common for an app to be organized into a tree of nested components



M01 - Introduction to Vue.js

1. Components

Defining a component

- We typically define each Vue component in a dedicated file using the .vue extension

- Known as a Single-File Component (SFC)

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">You clicked me {{ count }} times.</button>
</template>
```

M01 - Introduction to Vue.js

1. Components

Using a component

- To use a child component, we need to import it in the parent component
- To expose the imported component to our template, we need to register it with the components option
- The component will then be available as a tag using the key it is registered under
- Components can be reused as many times as you want

```
<script>
import ButtonCounter from './ButtonCounter.vue'

export default {
  components: {
    ButtonCounter
  }
}
</script>

<template>
  <h1>Here is a child component!</h1>
  <ButtonCounter />
</template>
```

```
<h1>Here are many child components!</h1>
<ButtonCounter />
<ButtonCounter />
<ButtonCounter />
```

M01 - Introduction to Vue.js

1. Components

Using a component

- In SFCs, it's recommended to use PascalCase tag names for child components to differentiate from native HTML elements.

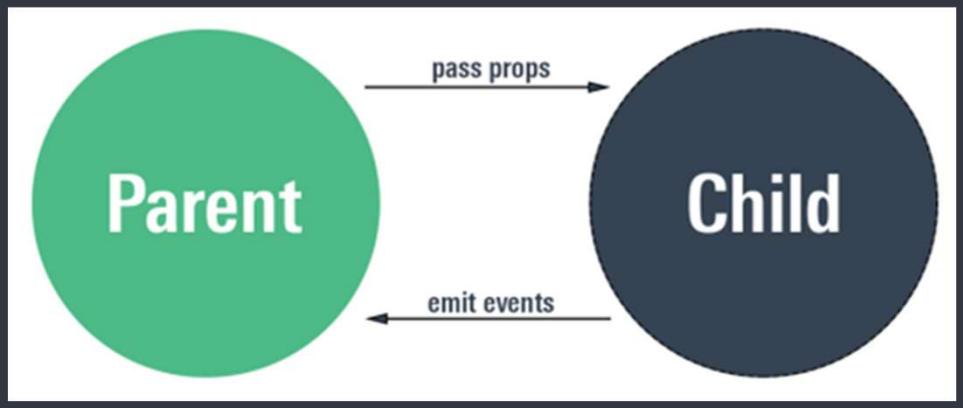
```
<h1>Here are many child components!</h1>
<ButtonCounter />
<ButtonCounter />
<ButtonCounter />
```

M01 - Introduction to Vue.js

1. Components

Communication between components

- Often, a Vue application will have several components
- How can we put them to communicate with each other?



M01 - Introduction to Vue.js

T04 - Components

1. Components
2. Props
3. Events
4. Slots

M01 - Introduction to Vue.js

2. Props

- Components are useful, but they really show their power when you pass data to them. This is done through the props!

- Props are custom attributes you can register in a component

- When a value is passed to a prop attribute, it becomes a property on that component

M01 - Introduction to Vue.js

2. Props

- For example, to pass a title to a blog posts component, we can include the post title in the list of props that this component accepts:

```
<script>
import BlogPost from "./components/BlogPost.vue";

export default {
  name: "App",
  components: {
    BlogPost,
  },
};
</script>
```

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

Call of child component assigning data using props

```
<template>
  <div>
    {{ title }}
  </div>
</template>

<script>
export default {
  name: "BlogPost",
  props: ["title"],
};
</script>
```

Use of props as a component property

Definition of props

M01 - Introduction to Vue.js

2. Props

- Instead of passing a array containing the names of the objects that the component can receive, it is possible to pass an object with other info

- Information about props:
  - Types of props
  - Obligatoriness
  - Default values
  - Custom validation functions

M01 - Introduction to Vue.js

2. Props

- Define props as an object, where the property names and values contain the prop names and types, respectively:

```
props: {
  title: String,
  likes: Number,
  isPublished: Boolean,
  commentIds: Array,
  author: Object,
},
```

Defines the type of a prop, with a native constructor such as Number, String, or Object, or a custom constructor function

- This not only documents the component, but also warns users in the browser console if they pass the wrong type
- A prop can be of several types, in which case it can pass all valid types in an array, for example: [Number, String]

M01 - Introduction to Vue.js

2. Props

- Mandatory and default values
  - Can specify if a prop is required or assign a default value if one is not defined
  - To do this, define an object instead of the constructor and pass the type with the object's type property

```
props: {
    price: {
      type: Number,
      required: true
    },
    unit: {
      type: String,
      dafault: '€'
    }
  }
```

Mandatory prop

Prop with default value

  - In this example:
    - price is mandatory and a warning will trigger if not specified
    - Unit is not mandatory, but has a default value of €, so if you don't assign any value, unit will be equal to € in the component

M01 - Introduction to Vue.js

2. Props

- Validation functions
  - you can pass a validator function that will receive the value of the prop and should:
    - Return true if prop is valid
    - Return false if prop is not valid

```
props: {
  price: {
    type: Number,
    required: true,
    validator(value) {
      return value >= 0;
    },
  },
},
```

Validation function

- This example validates that the number is above zero, so you cannot accidentally give negative prices

M01 - Introduction to Vue.js

2. Props

- Reactivity
  - When the value of the data object, method or calculated property is changed, the template is also updated, and this also works with props

  - The v-bind directive can be used when setting the prop in the parent to bind it to a value, then whenever that value is changed, wherever it is used in the component will also be updated

M01 - Introduction to Vue.js

2. Props

- Reactivity

```vue
<template>
  <div id="app">
    <display-number v-bind:number="number" />
  </div>
</template>

<script>
import DisplayNumber from "./components/DisplayNumber.vue";

export default {
  name: "App",
  data() {
    return {
      number: 0,
    };
  },
  created() {
    setInterval(() => {
      this.number++;
    }, 1000);
  },
  components: {
    DisplayNumber,
  },
};
</script>
```

```vue
<template>
  <div>
    {{ number }}
  </div>
</template>

<script>
export default {
  name: "DisplayNumber",

  props: {
    number: {
      type: Number,
      required: true,
    },
  },
};
</script>
```

DisplayNumber.vue

M01 - Introduction to Vue.js

2. Props

- Array of objects

```
<template>
  <div id="app">
    <display-castles v-bind:castles="castles" />
  </div>
</template>

<script>
import DisplayCastles from "./components/DisplayCastles.vue";

export default {
  name: "App",
  data() {
    return {
      castles: [],
    };
  },
  created() {
    this.castles = [{ id: 1, name: "Castelo ", link: "htt...", year: 1642 }];
  },
  components: {
    DisplayCastles,
  },
};
</script>
```

```
    <table>
      <tr>
        <th>id</th>
        <th>name</th>
        <th>link</th>
      </tr>
      <tr v-for="castle in castles" v-bind:key="castle.id">
        <td>{{ castle.id }}</td>
        <td>{{ castle.name }}</td>
        <td>{{ castle.link }}</td>
      </tr>
    </table>
    `

  </div>
</template>

<script>
export default {
  name: "DisplayCastles",

  props: {
    castles: {
      type: Array,
      required: true,
    },
  },
};
</script>
```

DisplayCastles.vue

M01 - Introduction to Vue.js

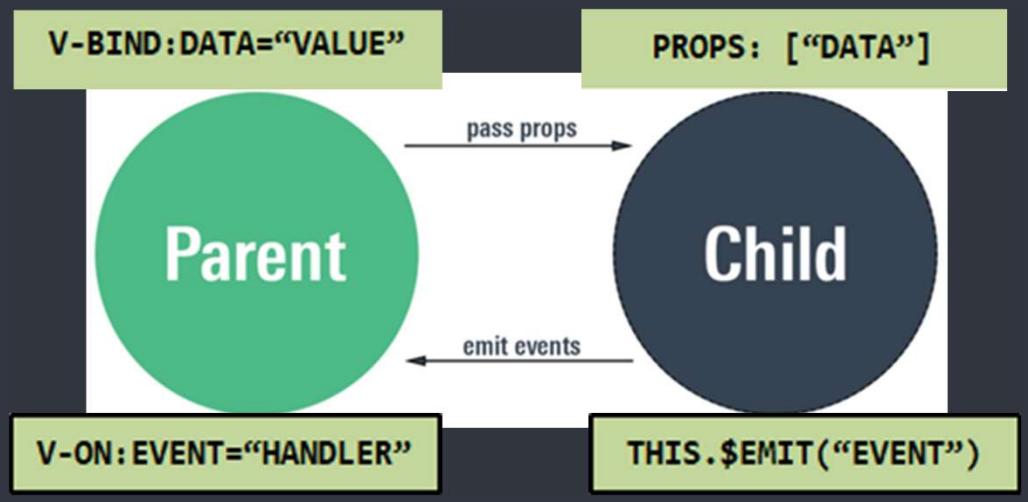
T04 - Components

1. Components
2. Props
3. Events
4. Slots

M01 - Introduction to Vue.js

3. Events

- To pass data from the child to the parent we must issue an event through this.$emit
- The event will have to be captured by the parent with the v-on directive



M01 - Introduction to Vue.js

3. Events

- Vue $emit lets emit (send) custom events from a child component to parent
- The best way to trigger certain events (like closing a popup, for example) or to send data from a child component (like making a custom input)
- How does it work?
  - Each Vue $emit call can pass two arguments:
    - An event name – name that we can listen to in our parent component
    - A payload object – data that we want to pass with the event (optional)

```
$emit('event-name', data)
```

M01 - Introduction to Vue.js

```vue
<template>
  <my-input @custom-change="logChange" />
</template>

<script>
import MyInput from "./components/MyInput.vue";
export default {
  components: {
    MyInput,
  },
  methods: {
    logChange(event) {
      console.log(event);
    },
  },
};
</script>
```

App.vue

```vue
<template>
  <div>
    <p>Wrapper for a text input</p>
    <input
      type="text"
      placeholder="Custom input!"
      @input="$emit('custom-change', $event.target.value)"
    /> <!-- INLINE EMIT -->
  </div>
</template>
```

MyInput.vue

M01 - Introduction to Vue.js

T04 - Components

1. Components
2. Props
3. Events
4. Slots

M01 - Introduction to Vue.js

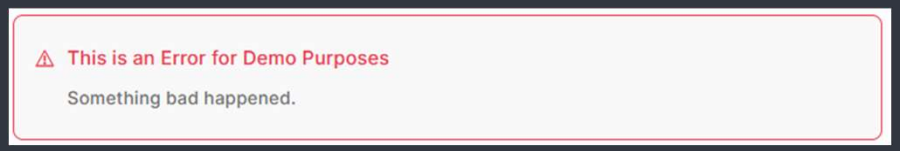
4. Slots

- Just like with HTML elements, it's often useful to be able to pass content to a component, like this:

```
<AlertBox>
  Something bad happened.
</AlertBox>
```

```
<template>
  <div class="alert-box">
    <strong>This is an Error for Demo Purposes</strong>
    <slot />
  </div>
</template>

<style scoped>
.alert-box {
  /* ... */
}
</style>
```



- As you'll see above, we use the <slot> as a placeholder where we want the content to go – and that's it.

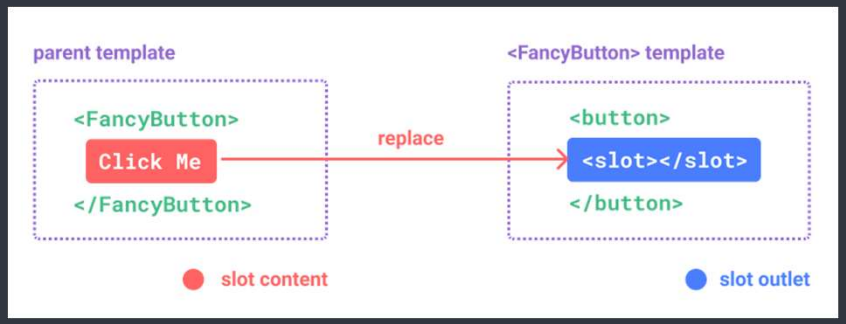
M01 - Introduction to Vue.js

4. Slots

- Normal slots

```
<FancyButton>
  Click me! <!-- slot content -->
</FancyButton>
```

```
<button class="fancy-btn">
  <slot></slot> <!-- slot outlet -->
</button>
```



M01 - Introduction to Vue.js

4. Slots

- Named slots

```
<BaseLayout>
  <template #header>
    <h1>Here might be a page title</h1>
  </template>

  <template #default>
    <p>A paragraph for the main content.</p>
    <p>And another one.</p>
  </template>

  <template #footer>
    <p>Here's some contact info</p>
  </template>
</BaseLayout>
```

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```