Web Programming

Vue.js II. (components)

Leander Jehl | University of Stavanger

Vue.js
installation

- https://v3.vuejs.org/guide
- Simply include the vue library.

```
<!-- Import Vue ->
<!-- development version, includes helpful console warnings -->
<script src="https://unpkg.com/vue@3.0.5/dist/vue.global.js"></script>
```

Vue.js

Recap

- Create element to contain vue app:

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

- Create vue app in JS:

```
<script>
    var app = Vue.createApp({
        // specify a template
        template: "<strong>{{ message }}</strong>",
        // the data used in this app
        data() {return { … }},
        // methods used to change state
        methods: {… },
        // computed properties
        computed: {… }
    });
    // mount the app on the element #app
    app.mount("#app");
</script>
```

Vue.js
Recap - Templates

- Content between **{{ }}** is replaced with element from data

```
{{ message }}
```

```
data: function(){
    return {
        message: "Hello Vue!"
    }
}
```

v-on and methods

- Specify methods of our app (component):

```
data(){
    return { message: "Hello Vue!" }
},
methods: {
    toggleMessage: function(){
        if (this.message == "Hello Vue!")
            this.message = "Can I help you?";
        else this.message = "Hello Vue!";
    }
}
```

Access **message** in **data** as **this.message**

Can pass arguments here, e.g. **method(argument)**

- Add event handler to template **v-on:event="method"**

```
template: /*html*/`
    <div
      v-on:mouseover="toggleMessage"
      v-on:mouseout="toggleMessage"
    >{{ message }}</div>`,
```

or

```
template: /*html*/`
    <div
      v-on:mouseover="toggleMessage()"
      v-on:mouseout="toggleMessage()"
    >{{ message }}</div>`,
```

Render lists: v-for

- Use **`v-for="item in array"`** in the template
  - Displays one element for each item in array

```
template: /*html*/`
    <ul>
        <li v-for="fruit in fruits">{{ fruit }}</li>
    </ul>`,
data(){
    return { fruits: ["apple", "pear", "banana", "orange"] }
}
```

  - Use **`v-for="item, index in array"`** to get both item and index

```
template: `
    <ul>
        <li v-for="(fruit, index) in shortFruits">#{{ index }} {{ fruit }}</li>
    </ul>`,
```

Exercise #0

github.com/dat310-2024/info/tree/master/
exercises/js/vue2

Forms

- **`v-bind`** only creates a one way binding
  - i.e. changes in input below are not reflected in JS

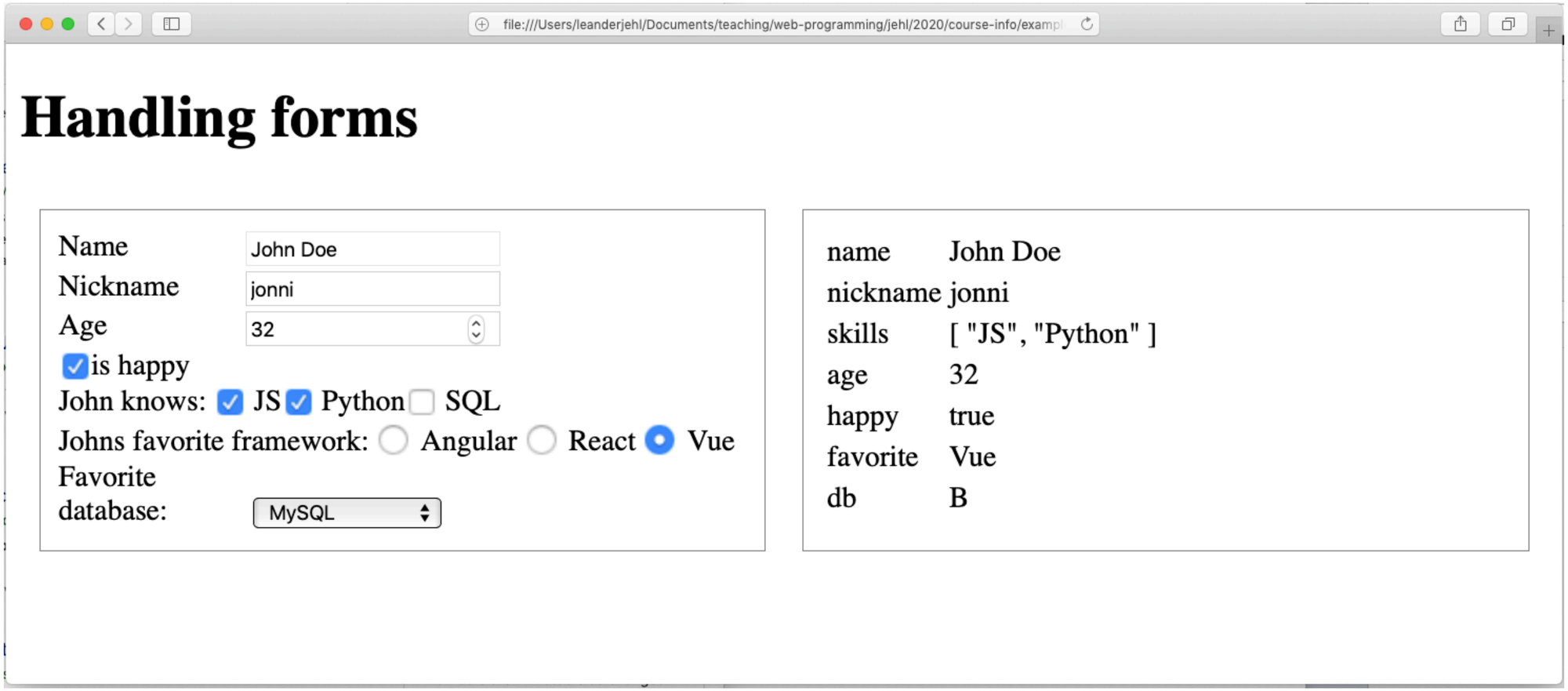
```
<input id="name" v-bind:value="john.name"/>
```

- use **`v-model`** to create two way binding

```
<input type="text" id="nickname" v-model="john.nickname"/>
```

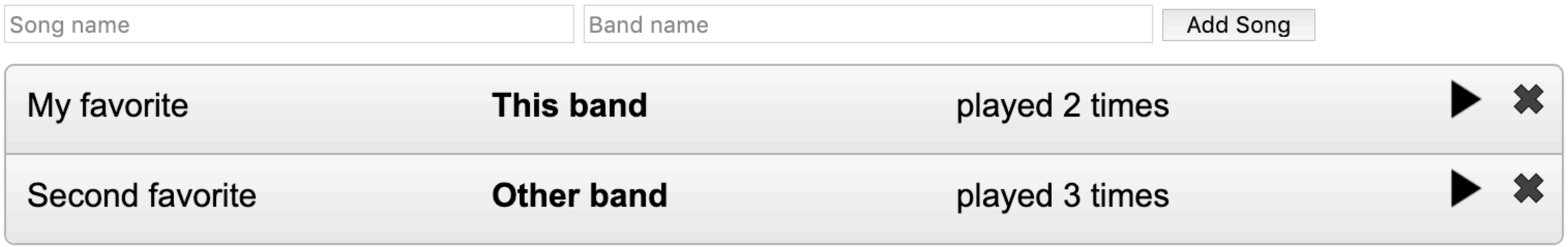
Example #2

examples/js/vue/vue6_model/index.html



Example #1

examples/js/vue2/list/index.html



v-bind:class

- Object syntax for **`v-bind:class="{ class: doApply }"`**
  - Only applied if **`doApply == true`**

```
<div v-bind:class="{ border: hasBorder }" > {{ message }} </div>
```

```
data() {
    return {
        message: "Hello Vue!",
        hasBorder: true,
    }
}
```

  - Can be combined with plain class:

```
<div v-bind:class="{ border: hasBorder }" class="box" >
    {{ message }}
</div>
```

v-bind:class

- Array syntax for **`v-bind:class="[ class1, class2 ]"`**
  - Easy to add multiple classes:

```
<div v-bind:class="[ class1, class2 ]" > {{ message }} </div>
```

```
data() {
    return {
        message: "Hello Vue!",
        class1: "box",
        class2: "border",
    }
}
```

  - Can be combined with object syntax:

```
<div v-bind:class="[{ border: hasBorder }, 'box']" >
    {{ message }}
</div>
```

v-bind:style

- Object syntax for **v-bind:style="{ prop: value }"**

```
<div v-bind:style="{ backgroundColor: bcolour, color: textcolor }" >
    {{ message }} </div>
```

In JS use camelCase for CSS properties.
Alternative use string **'background-color'**.

```
data() {
    return {
        message: "Hello Vue!",
        bcolor: "lightblue",
        textcolor: "white"
    }
}
```

Exercise #1

github.com/dat310-2024/info/tree/master/
exercises/js/vue2

Shortcuts

- Instead of **`v-bind:src="img"`** you can use **`:src="img"`**

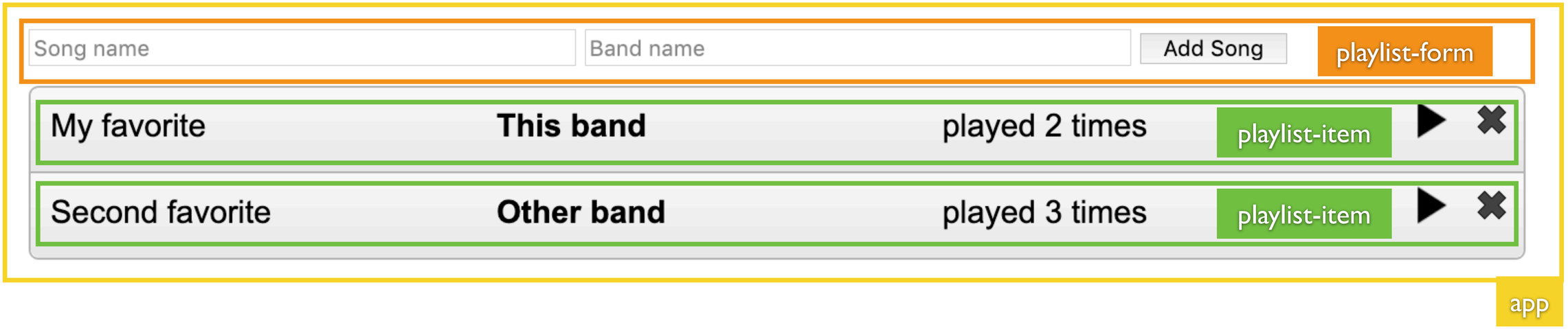
```
<img width="100px"
 :src='image'
 :alt='desc'>
```

- Instead of **`v-on:click="x"`** you can use **`@click="x"`**

```
<div
    @mouseover="toggleMessage()"
    @mouseout="toggleMessage()"
>{{ message }}</div>
```

Components

- If a Vue application gets too big/bloated we can separate it in multiple components.



Use **components,** do **not** use **multiple app instances.**

Components

- A component is like a Vue instance, that you can reuse several times in your app.
  - Define Component in JavaScript:

Use **kebab-case**.
Name must not conflict with html tag, i.e. **table**.

```
app.component('my-box', {
    template: `<div class="box"> HW! </div>`
});
```

  - Use several times in your app:

```
    <div id="app">
        <div style="float: left;">
            <my-box></my-box>
       </div>
        <div style="float: right;">
            <my-box></my-box>
        </div>
    </div>
```

Components

- The object passed to createApp is also a component
  - Define components (possibly in separate file:

```
let mainC = {
            data() { return { clicked: "" }},
            methods: { ... }
        }

let myboxC = { template: `<div class="box"> HW! </div>` }
```

  - Create app, register components and mount:

```
let app = Vue.createApp(mainC);
app.component('my-box', myboxC);
app.mount("#app");
```

Components

- Can have **`state`**, **`methods`**, and **`computed`** properties:

```
app.component('my-counter',{
    template: `
        <div class="counter">
            <span id="count" class="count">{{ count }}</span>
            <button v-on:click="increment">Add</button>
        </div>`,
    data: function(){
        return {count: 0};
    },
    methods: {
        increment: function(){
            this.count++;
        }
    }
})
```

Props: passing values to components

- A component can state properties (props),
  i.e. values it receives from parent component.

```
app.component('my-box', {
    // specify properties received
    props: [“color", “title”,”nr”],

    // use properties in template
    template: `
    <div class="box"
        v-bind:style="{ backgroundColor: color}">
        #{{ nr }}. {{ title }}
    </div>`
});
```

```
<my-box color=“green” title=“Hello” nr=“1” ></my-box>
```

Dynamic props

- Use v-bind on your property to:
  - Define state props based on JS and parent state
  - Reactively update props

```
<counter v-bind:init-count="count + 1"></counter>
```

```
<my-box v-bind:nr=“2 + 1”></my-box>
```

Using props

Do not reassign to a prop!

- In the template

```
props: ["initCount"],
template: `<div>{{ initCount }}</div>`,
```

- To assign initial state

```
props: ["initCount"],
data: function(){
        return {
            count: this.initCount,
        }
    },
```

Not reactive: Will not reflect changes in parent.

- To define computed property

```
computed: {
        totalCount: function(){
            return this.initCount + this.count;
        }
    },
data: function(){ return {count: 0}; },
```

Using props

- Property has camleCase

```
props: ["initCount"],
template: `<div>{{ initCount }}</div>`,
```

- In template use Kebab-Case

```
<counter v-bind:init-count="count + 1"></counter>
```

Property validation

- You can define properties in more detail:
  - **`type`**: e.g. String, Number, Object, Array, Boolean
  - **`required: true`** raises error if property is not given
  - **`default: defaultValue`**
  - **`validator: function(value)`** … if returns false, raise error

```
    props: {
        // color property is a string
        myColor: String,
        title: {
            // if not specified, use default
            default: "TBA",
        },
        nr: {
            type: Number,
            // raise error if not given
            required: true,
        },
    },
```

Property validation

- You can define properties in more detail:

```
    props: {
        // color property is a string
        myColor: String,
        title: {
            // if not specified, use default
            default: "TBA",
        },
        nr: {
            type: Number,
            // raise error if not given
            required: true,
        },
    },
```

```
<my-box my-color="green" v-bind:nr="1"></my-box>
```

Use **my-colo**r in html for **myColor** in JS,
i.e. kebab-case for camelCase.

To pass an object or number, use **v-bind**,
e.g. **v-bind:nr="3"**

(read the docs)

Events: communicating to the parent

- A component can emit events to inform its parent
  - Use **`$emit('event-name')`** in component

```
app.component('my-box', {
    // emit inside template
    template: `
    <div class="box"
        v-on:click="$emit('hello')">
        #{{ nr }}. {{ title }}
    </div>`,
    // emit inside method
    methods: {
        clicked: function(){
            this.$emit('hello');
        }
    }
```

  - Use **`v-on:event-name`** in parent

```
<my-box
  v-on:hello="clicked('the green one')"
  color="green"
  v-bind:nr="1"
></my-box>
```

Events: communicating to the parent

- Use kebab-case for composed event names

```
this.$emit('my-click');
```

```
<my-box
  v-on:my-click="handle()"
></my-box>
```

- Emit an event with value, by additional parameters to **`$emit`**

```
this.$emit('my-click', value);
```

1. Access value explicitly as **`$event`**

```
v-on:my-click="handle($event)"
```

2. Value passed as first argument to handler

```
<my-box
  v-on:my-click="handle"
></my-box>
```

```
// value as first argument
methods: {
    handle: function(value){ … }
}
```

Events: communicating to the parent

- You can declare what a component **`emits:`**

```
app.component('my-box', {
    // declare events this component emits:
    emits: [“my-click”],
    template: `…`,
```

- This allows type checking like in props.
- If you want to overwrite default events, e.g. click you need to declare them.

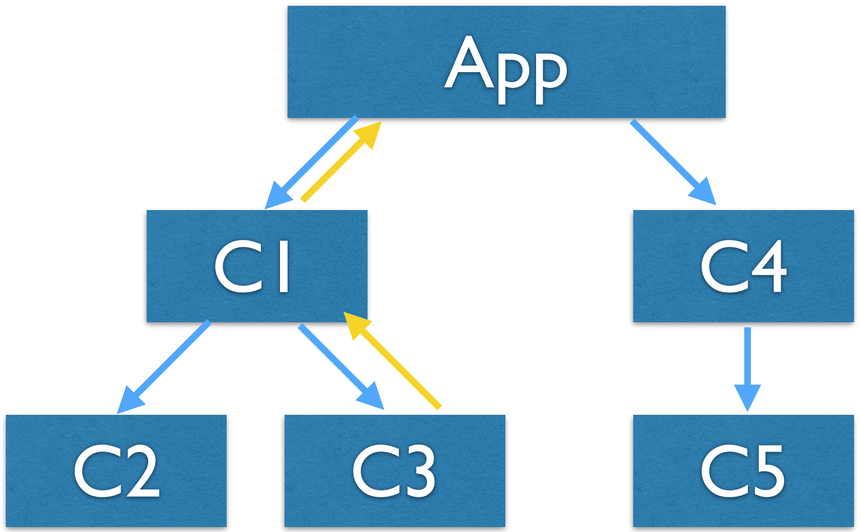
```
app.component('my-box', {
    // declare events this component emits:
    emits: [“click”],
    template: `…`,
```

Exercise #2, #3

github.com/dat310-2024/info/tree/master/
exercises/js/vue2

State management

- If multiple components access the same state, it needs to be passed down using props and changed using events.
  - State shared by C3 and C5 must be located in App.
  - If shared state is changed in C3, change is propagated using events and props



A different pattern: External store

- Outside of your app, define a store:

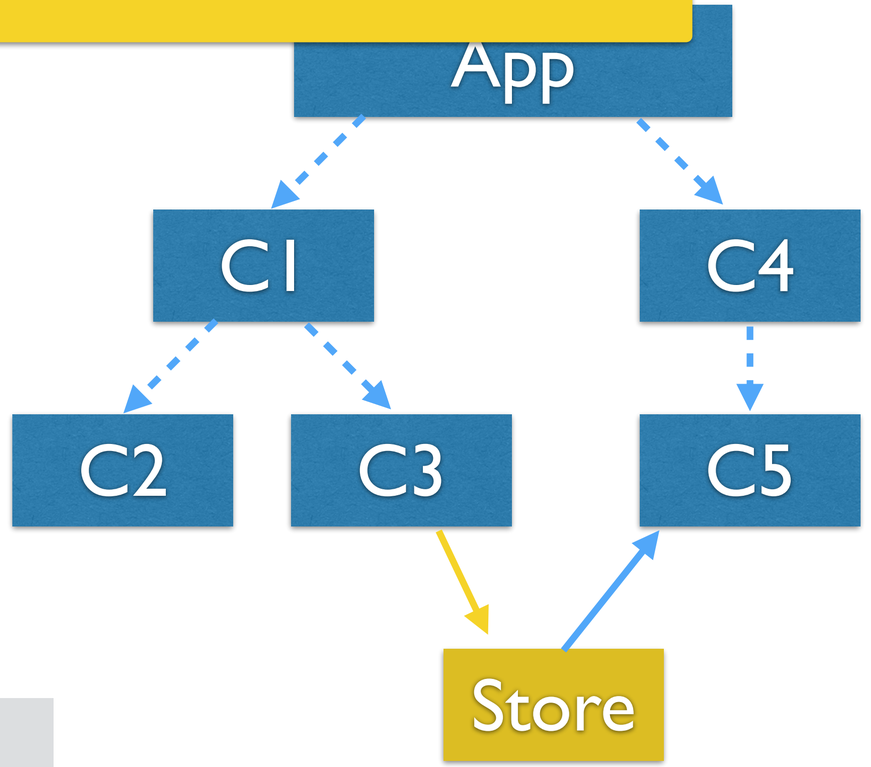
```
function DataStore(data){
    this.data = Vue.reactive(data);
    this.getter = function(){}
    this.setter = function(){}
}

let store = new DataStore(data);
```

Vue.reactive( ) allows Vue to react to changes.

- Retrieve data from store,
  e.g. on component creation

```
   data() {
        return store.data;
    }
```



(read the docs)

Example #2

examples/js/vue2/global-store-playlist

gstate.js

```
class GState { … }

const gState = new GState();

let app = Vue.createApp({
    data() {
        return gState.state;
    }
});
```

songListItem.js

```
Vue.component("song-list-item",{
    props: ['song'],
    template: ...
    methods: {
        remove: function(){
            gState.remove(this.song);
        }
    },
}
```

songForm.js

```
Vue.component("song-form",{
    template: ...
    methods: {
        addSong: function() {
            gState.add(new Song(this.song, this.band));
        }
    },
}
```

Update global state instead of emitting event.

Exercise #4, #4b

github.com/dat310-2024/info/tree/master/
exercises/js/vue2