Sistemes i Tecnologies Web
Name: Joan Mata Pàrraga
NIU: 1533089

Problemes Vue

Introducció:

Tots els exercicis estan en dos documents, un primer anomenat **index.html** on nomes cambiem el nom del archiu en el que treballarem en cada exercici.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Vuen</title>
        <script src="vue.js"></script>
    </head>
    <body>
        <h1>Exercici XX</h1>
        <div id="app"></div>
        <script type="text/javascript" src="exXX.js"></script>
    </body>
</html>
```

Exercici 1:

Create web page that counts from 0 to infinity. Create a vue instance **vm** with
- a **counter** property in its data option, which is initially set to 0,
- a template that interpolates the **counter**, and
- use the following piece of code to increase the counter.

```
setInterval(() => vm.counter++, 100);
```

```
let options = {
    data: function(){
        return{
            counter : 0,
        }
    },
    created: function () {
        setInterval(() => this.counter++, 100);
    },
    template: '<span> {{counter}} </span>'
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 2:

Create a vue instance that has a template with
- two <**input**> controls bound to variables **a** and **b**,
respectively, and - after these two controls, an interpolation of the addition of **a** and **b**.

Note: the term bound, as employed here, means that there is a data bindings between the variable and the form control. It may either be an one-way or a two-way binding (with **v-bind** or **v-model**, respectively). Hint: use **parseFloat**.

Option 1:

```
let options = {
    data: function() {
        var variable = {
            a : 0,
            b : 0
        }
        return variable
    },
    methods: {
        sum: function(a, b) {
          return a+b;
        }
    },
    template: '<input type="number" v-model="a">'
            + '<input type="number" v-model="b">'
            + '<span> {{ sum(a,b) }} <\span>'

}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Option 2:

```
let options = {
    data: function() {
        var variable = {
            a : 0,
            b : 0
        }
        return variable
    },
    methods: {
        sum: function(a, b) {
          return parseFloat(a, 10) + parseFloat(b, 10);
        }
    },
    template: '<input type="text" v-model="a">'
            + '<input type="text" v-model="b">'
            + '<span> {{ sum(a,b) }} <\span>'

}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 3:

Create a vue instance with a single <**button**> that disappears when clicked.

```
let options = {
    data: function() {
        return {
            button: 'Click me!'
        }
    },
    methods: {
        del: function() {
            if (this.button) {
                this.button = this.button.slice(0,0);
            }
        }
    },
    template: `<button v-if="button!=''" v-on:click="del()">{{button}}</button>`
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 4:

Create a vue instance with an empty <**input**> text box. The text box clears itself when its text length reaches 5 characters (or surpasses that number).

Option 1:

```
let options = {
    data: function() {
        return {
            box: ''
        }
    },
    watch: {
        box: function() {
            if (this.box.length > 4) {
                this.box = '';
            }
        }
    },
    template: '<input type="text" v-model="box">'
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Option 2:

```
let options = {
    data: function() {
        return {
            box: 0
        }
    },
    watch: {
        box: function() {
            if (this.box > 9999) {
                this.box = 0;
            }
        }
    },
    template: '<input type="number" v-model="box">'
```

```
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 5:

Create a vue instance with an empty <**input**> text box. The text box turns red when keys are pressed, and restores its original color upon key release.
Hint: v-on:keydown.

```
let options = {
    data: function() {
        return {
            box: {
                'background-color' : 'white'
            }
        }
    },
    methods: {
        red: function() {
            this.box = {
                'background-color' : 'red'
            }
        },
        white: function() {
            this.box = {
                'background-color' : 'white'
            }
        }
    },
    template: '<input type="text" v-bind:style="box" v-on:keydown="red()" ...
        v-on:keyup="white()">'
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 6:

Using the following template, create an instance that changes the 'redness' of the **AM I RED**? text according to the value in the range slider. Hide the **YES**! text when redness is under 70

```
<div>
      <div style="color: hsl(0,??%,50%)">AM I RED?</div>
      <input type="range" min="0" max="100">
      <div>YES!</div>
</div>
```

```
let options = {
    data: function() {
        return {
            flag : false,
            value : 0,
            object : {'color' : 'hsl(0,0%,50%)', }
        }
    },
    watch: {
        value: function(value) {
```

```
            this.object = {
                'color' : 'hsl(0,'+value+'%,50%)',
            }
            if (value < 70) {
                this.flag = false;
            }
            else {
                this.flag = true;
            }
        }
    },
    template: `<div>
        <div v-bind:style="object"> AM I RED? </div>
        <input type="range" min="0" max="100" v-model="value">
        <div v-if="flag">YES!</div>
        </div>`
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 7:

Create a vue instance with:
- the properties **a**, **b**, **c** and **d** in its data option (initially set to **false**), and
- a template with an <**input type=checkbox**> bound to **a**, followed by the interpolation of the four vari- ables (**a, b, c and d**).

Create a watch function for the variable **a** that sets **b** equal to **a**. Similarly, create a watch function for the variable **b** that sets **c** equal to **b**, and a watch function for the variable **c** that sets **d** equal to **c**.

```
let options = {
    data: function() {
        return {
            a: false,
            b: false,
            c: false,
            d: false,
        }
    },
    watch: {
        a:function() {
            this.b = this.a
        },
        b:function() {
            this.c = this.b
        },
        c:function() {
            this.d = this.c
        },
    },
    template: '<input type=checkbox v-model="a"> {{a}} {{b}} {{c}} {{d}}'
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 8:

Create a vue instance that displays the following phone book as shown in the accompanying figure.

```
[
  { name: 'Jaime Sommers', phone: '311-555-2368' },
  { name: 'Ghostbusters', phone: '555-2368' },
  { name: 'Mr. Plow', phone: '636-555-3226' },
  { name: 'Gene Parmesan: Private Eye', phone: '555-0113' },
  { name: 'The A-Team', phone: '555-6162' },
]
```

Employ the following CCS style.

```
table { border-collapse: collapse; }
table th,td { border: 1px solid black; }
```

Hints: **v-for** and the following html code.

```
<table><tr><th>Name</th><th>Phone number</th></tr>
<tr><td>{{item.name}}</td><td></td></tr>
</table>
```

```
let options = {
    data: function() {
        return {
            phone: [
                { name: 'Jaime Sommers', phone: '311-555-2368' },
                { name: 'Ghostbusters', phone: '555-2368' },
                { name: 'Mr. Plow', phone: '636-555-3226' },
                { name: 'Gene Parmesan: Private Eye', phone: '555-0113' },
                { name: 'The A-Team', phone: '555-6162' },
            ],
            tableClass: { 'border -collapse' : 'collapse'},
            trThClass: {'border': '1px solid black'}
        }
    },
    template: `
    <table v-bind:style="tableClass"><tr v-bind:style="trThClass"><th ...
        v-bind:style="trThClass">Name</th><th v-bind:style="trThClass">Phone ...
        number</th></tr>
    <tr v-for='item in phone' v-bind:style="trThClass"><td ...
        v-bind:style="trThClass">{{item.name}}</td><td ...
        v-bind:style="trThClass">{{item.phone}}</td></tr>
    </table>`
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 9:

Suppose this is a 'sempahore':

```
<div style="display: inline-block; width:30px;">
  <div style="height: 30px; background-color: indianRed"></div>
  <div style="height: 30px; background-color: khaki"></div>
  <div style="height: 30px; background-color: seagreen"></div>
</div>
```

Create a web page that:
- renders the semaphore in a vue template,
- has a **state** variable, which is an integer representing which light is on, and
- has a <**button**> that switches the semaphore **state**.

A value of **0** for **state** denotes a green light, a value of **1** denotes a yellow light, and a value of **2** denotes a red light. The initial state is **0**. Use the following css colors to represent when lights are on: **red**, **yellow**, and **lawngreen**; and use the follow- ing to represent when lights are off: **indianRed**, **khaki**, and **seagreen**.

```
let options = {
    data: function() {
        return {
            redLight: {
                'height': '30px',
                'background-color': 'indianRed',
            },
            yellowLight: {
                'height': '30px',
                'background-color': 'khaki',
            },
            greenLight: {
                'height': '30px',
                'background-color': 'green',
            },
            state: 0,
        }
    },
    methods: {
        change:function() {
            this.state  =this.state + 1;
            if (this.state > 2) {
                this.state = 0;
            }
        }
    },
    watch: {
        state: function(state) {
            if (this.state == 0) {
                this.redLight= {
                    'height': '30px',
                    'background-color': 'indianRed',
                };
                this.greenLight= {
                    'height': '30px',
                    'background-color': 'green',
                };
            }
            else if (this.state == 1) {

                this.yellowLight= {
                    'height': '30px',
                    'background-color': 'yellow',
                };
                this.greenLight= {
                    'height': '30px',
                    'background-color': 'seagreen',
                };
            }
            else {

                this.redLight= {
                    'height': '30px',
                    'background-color': 'red',
                };
                this.yellowLight= {
                    'height': '30px',
                    'background-color': 'khaki',
```

```
                };
            }
        }
    },
    template:
    `<div style="display: inline-block; width:30px;">
          <div v-bind:style="redLight"></div>
          <div v-bind:style="yellowLight"></div>
          <div v-bind:style="greenLight"></div>
        <button v-on:click="change()">switch</button>
    </div>`,
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 10:

Extend the previous 'phone book' exercise by adding delete buttons. Add a third column with an individual delete button for each entry.

Hints:

- `list.splice`,
- `v-for="(item,index) in list"`.

```
let options = {
    data: function() {
        return {
            phone: [
                { name: 'Jaime Sommers', phone: '311-555-2368' },
                { name: 'Ghostbusters', phone: '555-2368' },
                { name: 'Mr. Plow', phone: '636-555-3226' },
                { name: 'Gene Parmesan: Private Eye', phone: '555-0113' },
                { name: 'The A-Team', phone: '555-6162' },
            ],
            tableClass: { 'border -collapse' : 'collapse'},
            trThClass: {'border': '1px solid black'}
        }
    },
    methods: {
        del: function(index) {
            this.phone.splice(index, 1);
        }
    },
    template: `
    <table v-bind:style="tableClass"><tr v-bind:style="trThClass"><th ...
        v-bind:style="trThClass">Name</th><th v-bind:style="trThClass">Phone ...
        number</th><th v-bind:style="trThClass"></th></tr>
    <tr v-for='(item, index) in phone' v-bind:style="trThClass"><td ...
        v-bind:style="trThClass">{{item.name}}</td><td ...
        v-bind:style="trThClass">{{item.phone}}</td><td ...
        v-bind:style="trThClass"><button v-on:click="del(index)">delete</button></td></tr>
    </table>`
}

var app = Vue.createApp(options);
const vm = app.mount('#app')
```

Exercici 11:

Create the component **¡words-to-list¿** that transforms words into list items. Words given through the **words** attribute are transformed into <**li**> elements inside an <**ul**>. For example,<**words-to-list words=”w1 w2 w3”**></**words-to-list**> is transformed into <**ul**><**li**>**w1**</**li**><**li**>**w2**</**li**><**li**>**w3**</**li**></**ul**>.

Hint: **string.split(’ ’)**.

```
let options = { template:
    `<words-to-list words="Lorem ipsum dolor sit amet"></words-to-list>`
}
var app = Vue.createApp(options);

app.component(`words-to-list`,{
    data: function() {
        return {
            str : "",
        }
    },
    props: ['words'],
    template: `<li v-for='item in this.words.split(" ")'>{{item}}</li>`
})

const vm = app.mount('#app')
```

Exercici 12:

Create the component <**card**>, which is used to render user information. It is used as follows:

```
Vue.createApp({
    data: function() {
      return {
        person: {
            name: ’My Name’,
            picture: ‘
            fFcSJAAAADUlEQVR42mM82Mz1HwAFqgJP3gasfwAAAABJRU5ErkJggg==‘,
            email: ’me@somerandomdomain.com’,
            phone: ’+00 00 000 0000’,
        },
    template:
    ‘<div style="display:flex;">
        <card v-bind:personal-data="person"></card>
    </div>‘,
}).mount(’#app’);
```

For the previous data, the component template has to yield this final result:

```
<div class="card">
    <div>
        <img src="
        fFcSJAAAADUlEQVR42mM82Mz1HwAFqgJP3gasfwAAAABJRU5ErkJggg==">
    </div>
    <div><h1>My Name</h1></div>
    <div>me@somerandomdomain.com</div>
    <div>+00 00 000 0000</div>
</div>
```

Employ the following css style:

```
.card { font-family: Roboto; text-align: center; background: #ffbcbc; box-shadow: 6px 6px 8px
.card div {padding: 10px; }
.card img { width: 100px; }
```

```
let options = {
    data: function() {
        return {
          person: {
            name: 'My Name',
            picture: ` ...
                AfFcSJAAAADUlEQVR42mM82Mz1HwAFqgJP3gasfwAAAABJRU5ErkJggg==`,
            email: 'me@somerandomdomain.com',
            phone: '+00 00 000 0000',
          },
        }
    },
    template: `<div style="display:flex;">
                <card v-bind:personalData="person"></card>
              </div>`,
}
var app = Vue.createApp(options);

app.component(`card`,{
    data: function() {
        return {
            ClassCard : {
                'font-family': 'Roboto',
                'text-align': 'center',
                'background': '#ffbcbc',
                'box-shadow': '6px 6px 8px #888',
                'margin': '15px',
            },
            cardDiv : {
                'padding' : '10px',
            },
            cardImg : {
                'width' : '100px',
            }
        }
    },
    props: ['personalData'],
    template:
    `<div v-bind:style="ClassCard">
        <div>
            <img v-bind:style="cardImg" src={{this.personalData.picture}}>
        </div>
          <div v-bind:style="cardDiv"><h1>{{this.personalData.name}}</h1></div>
          <div v-bind:style="cardDiv">{{this.personalData.email}}</div>
          <div v-bind:style="cardDiv">{{this.personalData.phone}}</div>
    </div>`
})
const vm = app.mount('#app')
```

Exercici 13:

Create a <**switch-button**> component as follows:
- The rendered component has to resemble the following html snippet.

```
<div style="border:solid;display:inline-block">
    <button>ON</button>
    <button disabled>OFF</button>
</div>
```

- When the 'ON' button is clicked, the **on** event is dispatched, the 'ON' button is disabled, and the 'OFF' button is enabled.

- Similarly, when the 'OFF' button is clicked, the **off** event is dispatched, the 'OFF' button is disabled, and the 'ON' button is enabled.

```
let options = {
    data: () => ({ state: null }),
        template: `<div><switch-button
        v-on:on="state='just turned on'"
        v-on:off="state='just turned off'">
        </switch-button>{{state}}</div>`,
}
var app = Vue.createApp(options);

app.component(`switch-button`, {
    data: function() {
        return {
            state : '',
            styleClass : {
                'border' : 'solid',
                'display' : 'inline-block',
            },
            flagOFF: false,
            flagON: false,
        }
    },
    methods: {
        on: function() {
            this.$emit('on');
            this.flagON = true;
            this.flagOFF = false;
        },
        off: function() {
            this.$emit('off');
            this.flagOFF = true;
            this.flagON = false;
        }
    },
    template:
    `<div v-bind:style=styleClass>
        <button v-on:click="on()" v-if="flagON" disabled>ON</button>
        <button v-on:click="on()" v-if="!flagON">ON</button>
        <button v-on:click="off()" v-if="flagOFF" disabled>OFF</button>
        <button v-on:click="off()" v-if="!flagOFF">OFF</button>

    </div>`
})
const vm = app.mount('#app')
```

Exercici 14:

Create a <**color-selector**> component as follows:
- The rendered component has to resemble the following html snippet.

```
<div style="border:solid; display:flex;">
    <div style="background-color:#000; width:110px; height:110px;"></div>
    <div style="display:flex; flex-direction:column; padding:10px;">
        <div>R: <input type="range" min=0 max=255> red value</div>
        <div>G: <input type="range" min=0 max=255> green value</div>
        <div>B: <input type="range" min=0 max=255> blue value</div>
    </div>
</div>
```

- When a new color is selected, the component has to emit a **color** event with the selected color value (in css format).

```
let options = {
    data: () => ({ color: null }),
    template:
    `<div style="border:solid red; display:flex;">
        <color-selector v-on:color="color = $event"></color-selector>
        <div v-bind:style="'color:' + color">TEXT</div>
    </div>`,
}
var app = Vue.createApp(options);

app.component(`color-selector`, {
    data: function() {
        return {
            redValue : 0,
            greenValue : 0,
            blueValue : 0,
            styleColor :  0,
            StyleDiv1 : {
                'background-color' : '#000',
                'width' : '110px',
                'height' : '110px',
            },
            StyleDiv2 : {
                'display' : 'flex',
                'flex-direction' : 'column',
                'padding' : '10px',
            },
        }
    },
    watch: {
        redValue : function(){
           this.styleColor= ...
                "rgb("+this.redValue+","+this.greenValue+","+this.blueValue+")";
           this.StyleDiv1 = {
                'background-color': this.styleColor,
                'width':'110px',
                'height':'110px',
           };
           this.$emit('color',this.styleColor);
        },
        greenValue : function(){
           this.styleColor= ...
                "rgb("+this.redValue+","+this.greenValue+","+this.blueValue+")";
           this.StyleDiv1 = {
                'background-color': this.styleColor,
                'width':'110px',
                'height':'110px',
           };
           this.$emit('color',this.styleColor);
        },
        blueValue : function(){
           this.styleColor= ...
                "rgb("+this.redValue+","+this.greenValue+","+this.blueValue+")";
           this.StyleDiv1 = {
                'background-color': this.styleColor,
                'width':'110px',
                'height':'110px',
           };
           this.$emit('color',this.styleColor);
        },
    },
    template:
    `<div style="border:solid; display:flex;">
        <div v-bind:style="StyleDiv1"></div>
        <div v-bind:style="StyleDiv2">
            <div>R: <input type="range" min=0 max=255 v-model="redValue"> {{redValue}} ...
                </div>
            <div>G: <input type="range" min=0 max=255 v-model="greenValue"> ...
                {{greenValue}} </div>
            <div>B: <input type="range" min=0 max=255 v-model="blueValue"> ...
```

```
                    {{blueValue}} </div>
        </div>
    </div>`
})
const vm = app.mount('#app')
```

Exercici 15:

Create a <**magic-input**> component that works like a regular input text box (<**input type=text**>), except that it turns upper case letters into lower case letters and viceversa. The component has to support **v-model**, and should be used as follows:

```
<magic-input v-model="modelVariable"></magic-input>
```

Note that the <**magic-input**> has to display exactly what the user writes, and only change the text case in its model variable. For example, after typing the text "Hola" in <**magic-input v-model="modelVariable"**>, the text box has to display the text "Hola" and the value of **modelVariable** has to be "hOLA".

Similarly, when **modelVariable** is set to "Test", the text box has to display "tEST", as in the following example.

```
const vm = Vue.createApp({
  data: { modelVariable: "" },
  template: `<magic-input v-model="modelVariable"></magic-input>`,
}).mount('#app');
setInterval(() => vm.modelVariable = "Test", 1000);
```

Hints:
- Use the following case-switching snippet.

```
text.replace(/./g,
    x => x.toUpperCase() == x ? x.toLowerCase() : x.toUpperCase())
```

- Use the **created** option to initially set a data variable to the the value of the **value** prop.
- Watch for changes both in the prop variable and your data variable.

```
let options = {
    data: () => ({ modelVariable: "" }),
    template: `<magic-input v-model="modelVariable"></magic-input>`,
}
var app = Vue.createApp(options);

app.component(`magic-input`, {
    data: function() {
        return {
            box: '',
            propValue : '',
        }
    },
    watch: {
        box : function() {
            this.propValue = this.box.replace(/./g, x => x.toUpperCase() == x ? ...
                x.toLowerCase() : x.toUpperCase())
        },
    },
    template:
    `<div>
        <input v-model="box">
        <input v-model="propValue">
        {{propValue}}
```

```
24      </div>`
25  })
26  const vm = app.mount('#app')
```