JESUÏTES educació

DAWM06UF3

ESDEVENIMENTS. MANEGAMENT DE FORMULARIS

UF3.1 VUE.JS

CFGS Desenvolupament d'Aplicacions Web
M06. Desenvolupament web en entorn client
Fundació Jesuïtes Educació - Escola del Clot
Sergi Grau sergi.grau@fje.edu

OBJECTIUS

- Aprendre les característiques del l'arquitectura del bastiment VUE.JS

- Desenvolupar aplicacions front-end SPA amb VUE.JS

QUÈ ÉS VUE

- Vue.js és un bastiment de JavaScript de codi obert creat per Evan You per crear interfícies d'usuari i aplicacions HTML5 de única pàgina

- Vue.js és un bastiment progressiu. Això es deu a que permet començar a construir l'aplicació amb un esforç mínim, ja que la biblioteca Vue.js només se centra en la capa de visualització. Amb el pas del temps i a mesura que creixen els requisits, podeu adaptar biblioteques addicionals per a la funcionalitat.

Vue.js

- versió developer <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

- versió producció <script src="https://cdn.jsdelivr.net/npm/vue"></script>

PUNT DE MUNTATGE I MUSTACHE

- Podríem afegir la referència de Vue.js <script> abans del punt de muntatge, però això genera que bloquegi la resta de la pàgina de la càrrega, fent que sembli més lent per a l'usuari.

- L'element <script> que contindrà la nostra aplicació ha de ser després del punt de muntatge perquè el DOM estigui preparat per carregar l'aplicació.

- El punt de muntatge és

- mustache syntax, consisteix en dos claus que envolten el nom de la propietat que volem injectar les dades, com ara {{nom}}

PUNT DE MUNTATGE I MUSTACHE

- Creem nova instància de Vue.js, amb new Vue()
- Per dir a la nostra instància de Vue.js on muntar l'objecte d'opcions, passem a una **propietat** anomenada **el**. El valor per a això serà el selector CSS. En el nostre cas, això és #app ja que vam donar la nostra id d'una aplicació.
- Si no es proporciona la plantilla i la representació, el codi HTML de l'element proporcionat com a punt de muntatge s'utilitzarà com a plantilla de Vue per representar el DOM. Així és com podem injectar la nostre propietat de data a l'HTML que es va representar sense una plantilla.

PRIMER SCRIPT VUE

fitxer

```
<html lang="ca">

<head>
    <title>Material VUE:JS Jesuïtes Educació El Clot</title>
    <meta name="author" content="sergi.grau@fje.edu">
    <meta charset="UTF-8">
</head>

<body>
    <div id="app">
        Benvingut a {{nom}}
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        let app = new Vue({
            el: '#app',
            data: {
                nom: 'Vue.js!'
            }
        });
    </script>
</body>

</html>
```

- Si Vue utilitza una plantilla significa que probablement hauríem de passar una plantilla com a part de l'objecte d'opcions. Quan creeu una nova instància de Veu, una de les opcions que podeu proporcionar com a propietat és la plantilla.

- La cadena de plantilla s'utilitza per Vue per generar el DOM que es col · locarà a la pàgina web en lloc de l'element seleccionat amb l'opció el.

- Ha de tenir un element arrel. Això reemplaçarà tot allò que hi hagi dins de l'element a on la muntatge de la instància de Vue.

- També és possible utilitzar la propietat de la plantilla per proporcionar un selector CSS per orientar un element HTML que tingui una ID. Ho fem iniciant la cadena de plantilla amb una etiqueta hash (#).

- Això es pot fer amb un element <script> si li doneu un tipus de x-template i no es renderitzarà a la pàgina fins que Vue la utilitzi com a plantilla.

- També es podria utilitzar l'etiqueta <template>https://caniuse.com/#search=template

fitxer

```
<div id="app">
    Benvingut a {{nom}}
</div>

<template id="plantilla">

    <div>
        Plantilla predefinida: {{ nom }}
    </div>
    </template>
    <script>
        let app = new Vue({
            el: '#app',
            data: {
                nom: 'Vue.js!'
            },
            template: '#plantilla'

        });
    </script>
</body>

</html>
```

DATA

- Utilitzem la propietat **data** per indicar a la nostra instància quina forma donarà forma a les nostres dades

- Si hi ha alguna cosa que volem vincular a la instància de Vue, hem d'incloure'l a les dades abans de crear la nostra instància Vue.

- Quan es crea una nova instància de Vue, s’afegeix totes les propietats de les dades a un sistema reactiu. El sistema reactiu de Vue controla les propietats de l'objecte de dades per als canvis i actualitza la vista per "reaccionar" a aquests canvis.

- Això significa que no podem afegir noves dades que cal controlar el sistema reactiu de Vue després d'iniciar l'aplicació.

+ Si, en el moment de crear la vostra instància de Vue, no sabem quins són els valors de les propietats de les dades, caldrà definir-los amb els noms i donar-los un valor d'una cadena “”, null o undefined. No utilitzeu l'objecte buit {}, ja que renderitzarà el JSON enregistrat de l'objecte buit.

```
var app = new Vue({
    el: '#app',
    data: {
        emptyObject: {},
        emptyString: '',
        nullProperty: null
    }
});
```

- Els noms de les propietats de l'objecte de dades no han de començar amb $ o _. Les propietats que comencin per $ o _ no s'afegiran al sistema reactiu, ja que poden provocar conflictes amb les propietats i mètodes interns de Vue.

- Si feu servir un valor que comença amb $ o _ a la plantilla, es produirà un error.

- Quan es crea una instància de Vue, l'objecte de dades originalment inclòs s'afegeix a la instància com una propietat amb el nom de **$data**. Per tant, si assigneu la vostra instància de Vue a una variable anomenada app, podeu accedir a l'objecte de dades original amb **app.$data.nomProp**. També podreu accedir-hi en mètodes amb **this** en lloc d'una referència desada a la instància de $data.nomPropQuan es crea una instància de Vue, l'objecte de dades originalment

MÈTODES

+ Mitjançant **mètodes**, podem crear un codi personalitzat que estarà lligat a la nostra instància de Vue. A continuació, podrem accedir a aquests mètodes des d'una referència a la nostra instància Vue. Quan creeu mètodes per a la nostra instància Vue, hem **d'evitar utilitzar la funció de fletxa** () => {}, ja que ens impedirà accedir al context adequat de **this**.

```
var app = new Vue({
  el: '#app',
  methods: {
      polsarBoto: function () {
          console.log('VUE!');
      }
  }
});
```

PROPIETATS COMPUTADES

+ Les propietats computables són molt similars als mètodes, amb una diferència important: els resultats es troben en memòria cau. Els valors només s'actualitzen quan canvien els valors en que la propietat calculada es basen.

```
computed: {
    passarMajuscules: function () {
        return this.nom.toUpperCase();
    }
}
```

DIRECTIVA V-BINDINGS

```
<div>{{ 1 + 1 }}</div>
<div>{{ falseValue === false ? yes : no }}</div>
<div>{{ 1 == 2 ? si : no }}</div>
<div>{{ 1 + 1 + 1 > 2 ? si : no }}</div>
```

La sintaxi de bigoti funciona bé per a les propietats d'enllaç que estan destinades a ser de text, però no es pot utilitzar per lligar valors als atributs d'elements HTML. Per enllaçar amb atributs, aprendrem sobre la nostra primera **directiva Vue**: **v-binding**.

```
<div v-bind:name="nom"></div>
```

DIRECTIVA V-HTML

+ Una altra directiva que podem utilitzar per enllaçar les dades a la plantilla és v-html. Amb v-html, el contingut de l'element que s'aplica es reemplaça amb el valor assignat i es tracta com a HTML. Això es pot utilitzar quan els requisits exigeixen l'addició d'HTML que provingui d'una font fora de la vostra aplicació Vue.

```
var app = new Vue({
    el: '#app',
    data: {
     codiHTML: '<h1 style="color:#41b883; background-
     color:#35495e;">Vue.js</h1>'
    },
    template: `
        <div>
            <div>{{ codiHTML }}</div>
            <div v-html="codiHTML"></div>
        </div>
        `
});
```

REPRESENTACIÓ CONDICIONAL

- De vegades, la vostra aplicació haurà de ser capaç de determinar si mostrar o no alguna cosa en funció de les interaccions de l'usuari.

- Vue proporciona dues directrius per mostrar el contingut de manera condicionada: v-if i v-show.

- Amb **v-show**, podem amagar i mostrar contingut mitjançant la propietat de visualització de CSS.

- Amb **v-if**, el contingut s'elimina del DOM. Es pot utilitzar amb les directrius **v-else i v-else-if**.

REPRESENTACIÓ CONDICIONAL

- Des d'una perspectiva de rendiment, v-show té un cost de renderització inicial superior, ja que es representa al DOM fins i tot si les condicions per mostrar-ho són falses.

- v-if no es representarà si el valor és fals.

- v-show té menys cost de renderització quan el valor canvia, ja que ja es troba al DOM i la propietat de visualització de CSS és l'únic canvi.

- D'altra banda, s'ha d'afegir v-if al DOM quan la condició per representar-la canvia de fals a veritable.

- Si es vol canviar només de tant en tant o mai després de la primera representació, és millor utilitzar v-if.

V-SHOW

- L’ús de **v-show** és similar a l’ús d’un atribut element HTML. La principal diferència és que el valor que assigneu és de la vostra instància Vue o una expressió que avalua a true o false.

- L'expressió pot comparar els valors de la vostra instància Vue amb els valors que heu establert en l'assignació o en altres valors de la vostra instància Vue.

- v-show=“mostra" No és una cadenes, és una propietat. Si volem avaluar una cadena cal posar-ho entre cometes simples

V-SHOW VS V-IF EN DOM

JESUÏTES educació

DAWM06UF3 sergi.grau@fje.edu

1: si

2:

3: si!

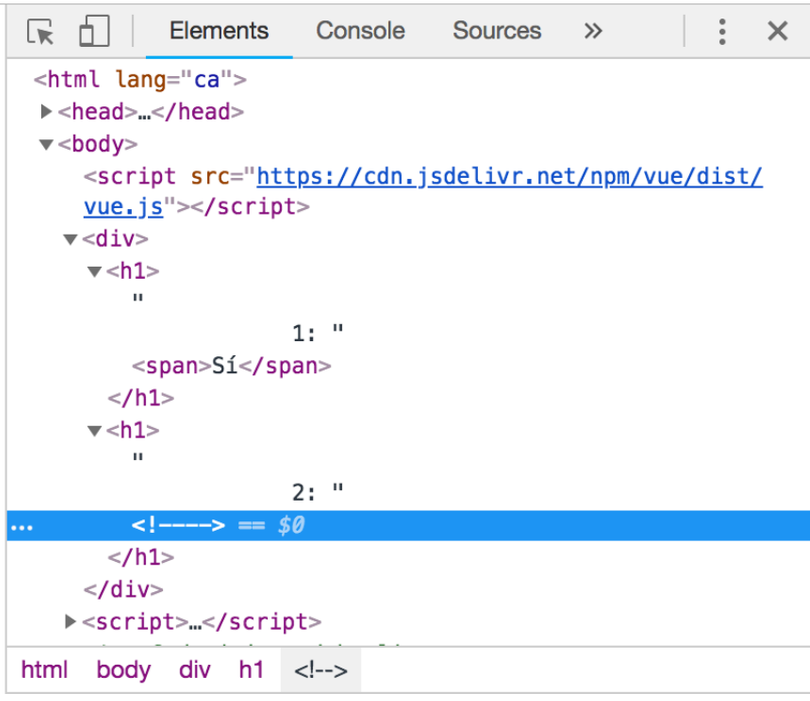
4:

5: si!

6: si!

1: Sí

2:



V-ELSE

```
var app = new Vue({
  el: '#app',
  data: {
    no: false
  },
  template: `
    <div>
        <h1 v-if=“no">No mostra</h1>
        <h1 v-else>Mostra!</h1>
    </div>
    `
});
```

V-ELSE-IF

- De vegades és necessari poder fer una selecció selectiva amb d'altres opcions. Per això, podem utilitzar v-else-if. Amb **v-else-if**, podem encadenar declaracions en conjunt, de manera similar a si es fan instruccions de si else en JavaScript.

```
<div>
  <h1>Show v-else-if</h1>
  <h2 v-if="no">If</h2>
  <h2 v-else-if="si">Else If</h2>
  <h2 v-else>Else</h2>
</div>
<div>
  <h1>Show v-else</h1>
  <h2 v-if="no">If</h2>
  <h2 v-else-if="no">Else If</h2>
  <h2 v-else>Else</h2>
</div>
```

TEMPLATE I V-IF

- Amb el grup de directives v-if, és possible agrupar els elements de manera que pugueu mostrar-los i amagar-los amb una acció en comptes d'utilitzar v-if cada vegada. Per fer-ho, envoliu els elements que es mostraran i s'amaguen en un element <template>. L'element <**template**> no es representarà si el valor **v-if** s'avalua com a true, però tots els elements fills ho faran.

```
<template v-if="no">
    <h1>no mostra</h1>
    <h2>aquest text</h2>
 </template>
 <template v-if="si">
    <h1>Mostra</h1>
    <h2>aquest text</h2>
 </template>
```

- Sovint, els desenvolupadors hem de tractar amb grups del mateix article, normalment per mostrar-los en una pàgina web. Vue proporciona una directiva per gestionar la visualització dels elements de la matriu, anomenats **v-for**.

```
let app = new Vue({
  el: '#app',
  data: {
    items: ['primer', ‘segon', '3']
  },
  template: `
      <ul>
        <li v-for="item in items">
          {{item}}
        </li>
      </ul>
    `
});
```

+ Com els objectes de la vostra col·lecció es compliquen, es recomana utilitzar l'atribut**: key**. Vue utilitza l'atribut: key per fer el seguiment de la identitat dels elements que s'han representat i actualitzar correctament el DOM.

- Si no esteu utilitzant : key, eviteu afegir o eliminar elements a la matriu excepte al final o quan ordeneu la matriu. Vue no pot fer un seguiment i actualitzar tots els elements secundaris que es repeteixen correctament.

- Vue ens ofereix l'opció d'afegir un segon paràmetre a v-for que ens proporciona l'índex.

```
<span v-for="(area, index) in llibres.area”>
    {{area}}<span v-if="index < llibre.area.length -1">,
 </span>
</span>
```

MANIPULAR ARRAYS

- Vue envolta observadors al voltant dels mètodes de mutació de la matriu següents: **push, pop, shift, unshift, splice, sort, reverse, empenta, pop, canvi, unshift**

- Això vol dir que sempre que canvieu la vostra matriu mitjançant aquests mètodes, Vue podrà detectar els canvis.

- Si utilitzeu un mètode que no muta ni canvia la matriu original, Vue no ho detectarà. Els mètodes que no canvien la matriu original són **filter, concat, i slice**.

- Per tal que Vue observi aquests canvis a la matriu, substituïu la matriu original pels resultats.

FILTRAR ARRAYS


```
methods: {
        filtrarAreaPerJS: function () {
          this.llibres = this.llibres.filter(llibre => {
            return (
              llibre.area.findIndex(area => {
                return area === 'JS';
              }) >= 0
            );
          });
        }
    },
});
```

- { "titol": "Vue", "preu": 6.99, "id": 2, "area": ["JS", "SPA"] }
- **Id:** 2

  Títol:

  Arees: JS , SPA

- Vue no pot detectar quan un element es substitueix en una matriu utilitzant l'índex de l'element i quan la matriu es canvia de mida assignant un valor nou a la propietat de longitud.

- Per evitar aquestes limitacions, podeu substituir elements d'una matriu mitjançant **Vue.set**

OBJECTES V-FOR

- També és possible utilitzar v-for per passar per les propietats d'un objecte. Com que els motors de JavaScript es comporten de manera diferent, no hi ha cap garantia sobre l'ordre de les propietats en diferents navegadors.

- La principal diferència en utilitzar v-for amb un objecte en lloc d'una matriu és que, amb un objecte, quan utilitzeu parèntesis per accedir al valor i l'índex, accepta tres paràmetres: **valor, clau i índex**. El valor i l'índex representen les mateixes coses que la matriu. La clau representa el nom de la propietat.

```
<ul>
  <li v-for="(prop, key, index) in llibre”>
    {{index}}) {{key}}: {{prop}}
  </li>
</ul>
```

PROPIETATS COMPUTADES I OBSERVADORS

- L'ús de mètodes en la vostra instància de Vue per obtenir dades formatades és excel·lent, però s'aconsegueix un elevat nombre de funcions cada vegada que la vista s'actualitza o es torna a representar.

- Podem evitar el pagament d'aquest rendiment utilitzant propietats computades.

- De vegades també hem de poder realitzar tasques de fons o asíncrones quan l'usuari interactua amb la pàgina, però no volem que l'usuari interaccioni. Vue ens ofereix l'opció d'utilitzar els observadors en aquests casos

- Les propietats computades funcionen de manera similar als mètodes de Vue. La principal diferència és que els resultats s'emmagatzemen per a un ús posterior o es desa en memòria cau fins que una de les dependències reactives de la propietat calculada canvia.

ESDEVENIMENTS

- La configuració d'oients aesdeveniments a Vue és bastant senzilla. A l'element des del qual voleu escoltar esdeveniments, afegiu un atribut de v-on: esdeveniment = "gestor", on esdeveniment és el nom de l'esdeveniment que us interessa i gestor és com voleu gestionar l'esdeveniment. $event ens permet saber el esdeveniment que ha ocorregut.

```
mostrar: function(nouValor, event) {
  if (event) {
    console.log(event.type);
  }
  this.mostra = nouValor;
}
  },
  template: `
<div>
    <h1 v-on:click="mostra = !mostra">Canvia</h1>
    <p v-show="mostra">Ocular i mostrar missatge</p>

    <h1 v-on:click="mostrar(true, $event)">
    Mostra
```

MODIFICADORS

+ Els modificadors d'esdeveniments ens permeten canviar declarativament el comportament d'un esdeveniment. Significa, de manera declarativa, quan volem modificar el comportament d'un esdeveniment, el declarem al marcatge. No assignem la modificació a algun lloc del JavaScript. Això ens permet veure les modificacions en les quals registrem els controladors d'esdeveniments al marcatge i deixem els mètodes de control només per a les parts del codi necessari per gestionar l'esdeveniment.

MODIFICADORS

- stop: Crida a event.stopPropagation () i deté la propagació de l'esdeveniment actual.

- prevent: Anomena event.preventDefault () i diu a l'agent d'usuari que no gestiona l'esdeveniment amb el controlador predeterminat.

- capture: afegeix el detector d'esdeveniments en mode de captura. L'ús del mode de captura per a l'esdeveniment permetrà que el nostre gestor s'enviï abans que l'objectiu de l'esdeveniment pugui gestionar-lo.

- self: truca al gestor només si l'esdeveniment comença a l'element on registrem el controlador. Això ens estalvia el treball addicional de comprovar l'esdeveniment.target per limitar el nostre controlador a només els esdeveniments que comencen amb l'element amb el qual registrem l'esdeveniment.

MODIFICADORS

- once: solament crida al controlador una sola vegada sense que haguem de treure el controlador de l'element quan gestionem l'esdeveniment.

- passive: defineix l'opció de gestor d'esdeveniments de passiu a true, de manera que el gestor no truca event.preventDefault () i, si ho fa, el navegador no ho farà. Es van introduir gestors passius d’esdeveniments per ajudar els navegadors a proporcionar un aspecte més coherent amb els esdeveniments durant el desplaçament.

```
<div v-on:click.stop="missatges.push('X prop’)">
```

js

```
<div v-on:click="missatges.push('A')">
            <h4>A</h4>
            <div v-on:click="missatges.push('X')">
                <h4>X</h4>
                <div v-on:click.stop="missatges.push('X prop')">
                    <h4>X prop</h4>
                </div>
                <div v-on:click="missatges.push('B')">
                    <h4>B</h4>
                </div>
            </div>
        </div>
        <p>
            Last clicked:
            <ol>
                <li v-for="missatge in missatges">
                    {{missatge}}
                </li>
            </ol>
            </p>
        <input type="button" v-on:click="missatges = []" value="Neteja" />
    </div>
```

MODIFICADORS TECLAT

- Enter: The Enter or Return key
  Tab: The Tab key
  Delete: The delete or backspace key
  Esc: The Escape key
  Space: The spacebar
  Up: The up arrow key
  Down: The down arrow key
  Left: The left arrow key
  Right: The right arrow key

ENLLAÇOS

+ Un dels principals motius per utilitzar un batiment com Vue és que facilita la resposta a l'entrada dels usuaris, com ara els formularis.

+ També per actualitzar l'aspecte quan els usuaris fan una entrada / selecció d'una opció, canviant els estils i les classes perquè l'usuari sàpiga que passa alguna cosa.

+ Vue ens proporciona la directiva del **v-model** per unir dades a les nostres entrades. Amb el v-model, tindrem una unió de dades bidireccional des de la variable de dades de suport a la IU. Amb l'enllaç de dades bidireccional si l'usuari fa un canvi al model de dades mitjançant un mètode, veurem que la interfície d'usuari mostrarà l'actualització. Si fem un canvi a la IU, s'actualitzarà el model de dades.

+ .lazy: utilitza l'esdeveniment de canvi en lloc de l'esdeveniment d'entrada per actualitzar el model de dades.Per exemple quan es canvia el focus.

+ .number: Intenta llançar el valor a un nombre en assignar-lo al model de dades.

+ .trim: elimina l'espai en blanc quan assigna al model de dades.

ESTILS EN LÍNIA

+ La vinculació als estils en línia ens permet assignar valors directament a les propietats CSS. Utilitzem la sintaxi similar a la vinculació amb altres atributs, però utilitzem un objecte JavaScript que definim a l'expressió.

```
<input type="number" v-model.number="fontSize" />
<p v-bind:style="{fontSize: fontSize + 'px'}">
   VUE.JS
```

OBJECTES I ESTILS

+ En lloc de definir l'objecte que volem utilitzar per a un estil de l'expressió, el podem definir com a propietat de dades. D'aquesta manera podem assignar tot l'objecte. No ens hem de preocupar de definir el nostre objecte com una cadena, ja que podem obtenir un ressaltat de sintaxi adequat a l'editor de JavaScript.

```
<input type="number" v-model.number="fontSize" />
<p v-bind:style="{fontSize: fontSize + 'px'}">
   VUE.JS
```

```
<input type="number" v-model.number="fontSize" />
<p v-bind:style="{fontSize: fontSize + 'px'}">
Vue.js
</p>
```

GESTIÓ D'ESTATS

- Cada aplicació té algunes dades per gestionar en forma de valors per fer un seguiment relacionat amb les opcions de l'usuari o la informació que es mostra. Per complicar encara més les coses, podríem estar utilitzant les mateixes dades a través de múltiples instàncies de Vue. La manipulació es denomina gestió d'estats.

- És pot fer comun simple objecte de dades, un magatzem de dades personalitzats o una biblioteca de gestió estatal anomenada Vuex.

- Podeu veure que tenim un objecte JavaScript separat que té una propietat, un valor. Aquest objecte s'utilitza llavors com a propietat de dades per a dues instàncies Vue diferents, app1 i app2.

- Tanmateix, la cosa és que, com que la vostra aplicació és més complexa, obtindreu cada vegada més dificultats per verificar que les dades compartides es canvien correctament, ja que cada instància de Vue amb què es comparteix pot canviar directament.

```
data: {compartides: dadesCompartides,private: {} },
```

MAGATZEM DE DADES

+ Utilitzem un objecte JavaScript per contenir les dades que volem compartir com l'objecte de dades simple, però afegim mètodes per canviar / actualitzar aquestes dades en lloc de canviar-lo directament. Això ens permet entendre millor quan i com es canvien les dades

```
data: {compartides: dadesCompartides,private: {} },
```

JESUÏTES educació
DAWM06UF3 sergi.grau@fje.edu

```
var dadesCompartides = {
  modeDesenvolupador: true,
  estat: {
    valor: 1
  },
  augmentar() {
    if (this.modeDesenvolupador) {
      console.log('augmentar() ');
    }
    this.estat.valor++;
  },
  disminuir() {
    if (this.modeDesenvolupador) {
      console.log('disminuir() ');
    }
    this.estat.valor--;
  },
  setValor(nouValor) {
    if (this.modeDesenvolupador) {
      console.log('setValor() amb nou valor: ', nouValor);
    }
    this.estat.valor = nouValor;
  }
};
+
```

- Vuex és una biblioteca mantinguda per l'equip Vue que proporciona gestió de l'estat juntament amb algunes trameses o característiques addicionals. El connector oficial de l'eina Vue dev permet a Vuex realitzar depuracions temporals juntament amb la importació i exportació de l'estat.

- Vuex està dissenyat per actuar com a estat d'aplicació per a tots els components Vue de la vostra aplicació.

- Vue necessita que el navegador suporti promeses

https://unpkg.com/vuex

o

```
npm install vuex -save
```

COMPONENTS

- Un component és un element personalitzat que podem definir i reutilitzar. Definirem els nostres components com a petites instàncies de Vue, però en comptes de cridar a Vue per a una instància completa de Vue, hem de registrar-los quan sigui necessari.

- Per tant, cada component tindrà la seva versió de la majoria de les coses que té una instància Vue, excepte una propietat el.

- Utilitzarem Vue.component per registrar el component. Penseu en registrar el component com a indicació a Vue sobre el que està disponible per utilitzar-lo. Passarem dos paràmetres per registrar el nostre component: un nom per al component i un objecte JavaScript que conté totes les opcions.

```
Vue.component('ComponentPropi', {
  data: function() {
    return {
      text: 'DAW2'
    };
  },
  template: `
    <h1 v-on:click="text =
text.split('').reverse().join('')">{{text}}</h1>
  `
});

let app = new Vue({
  el: '#app',
  template: `
      <div>
        <component-propi></component-propi>
        <ComponentPropi></ComponentPropi>
      </div>
      `
});
```

COMPONENTS

+ Tenir un component que conté les seves pròpies dades és genial, però probablement seria millor passar les dades al component del component pare, de manera que podríem reutilitzar-lo. Amb props, podem especificar valors que es poden passar al component.

```
Vue.component('ComponentDAW2', {
  props: ['text'],
  template: `
    <h1>{{text}}</h1>
  `

<ComponentDAW2 text=“DAW2" />


});
```

BIBLIOGRAFIA

+ https://vuejs.org/v2/guide/

+ https://vuejsdevelopers.com/

+ Brett Nelson (2018). Getting to Know Vue.js Apress, Berkeley, CA