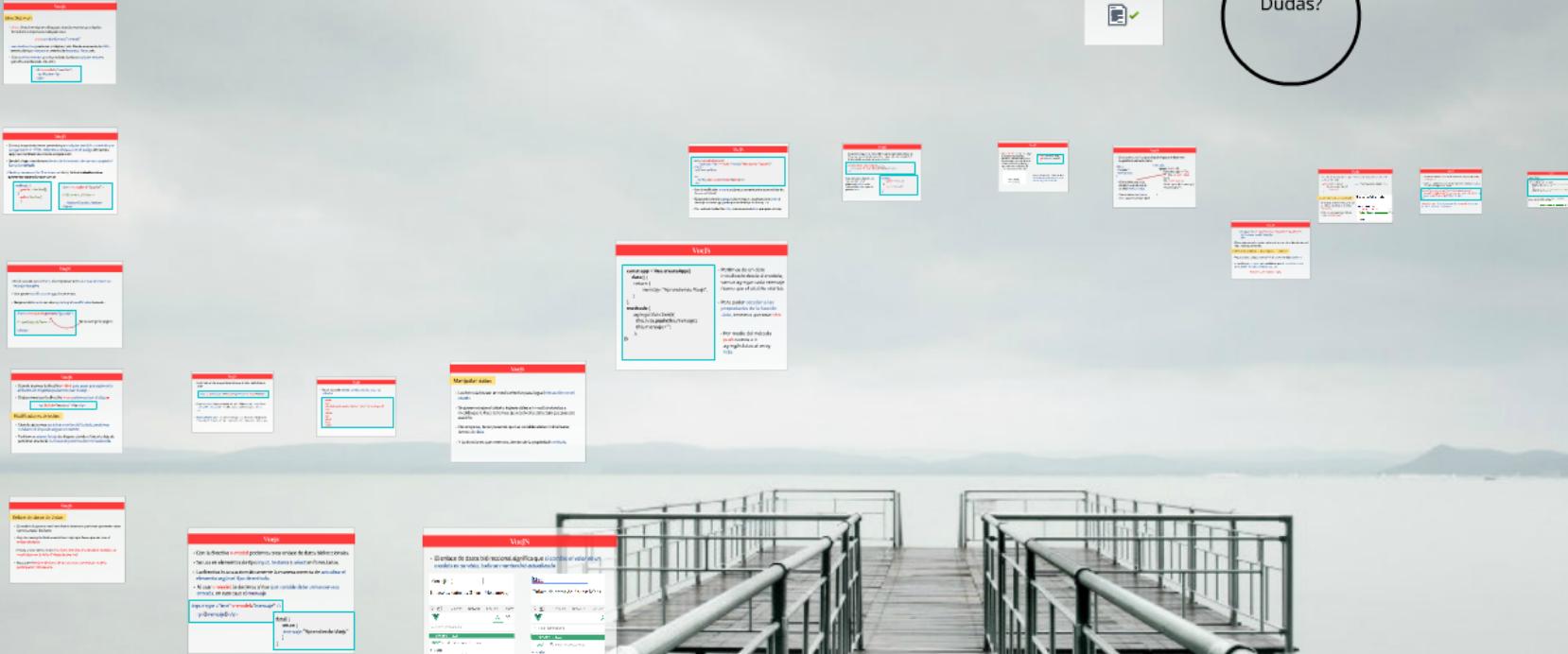


ADM

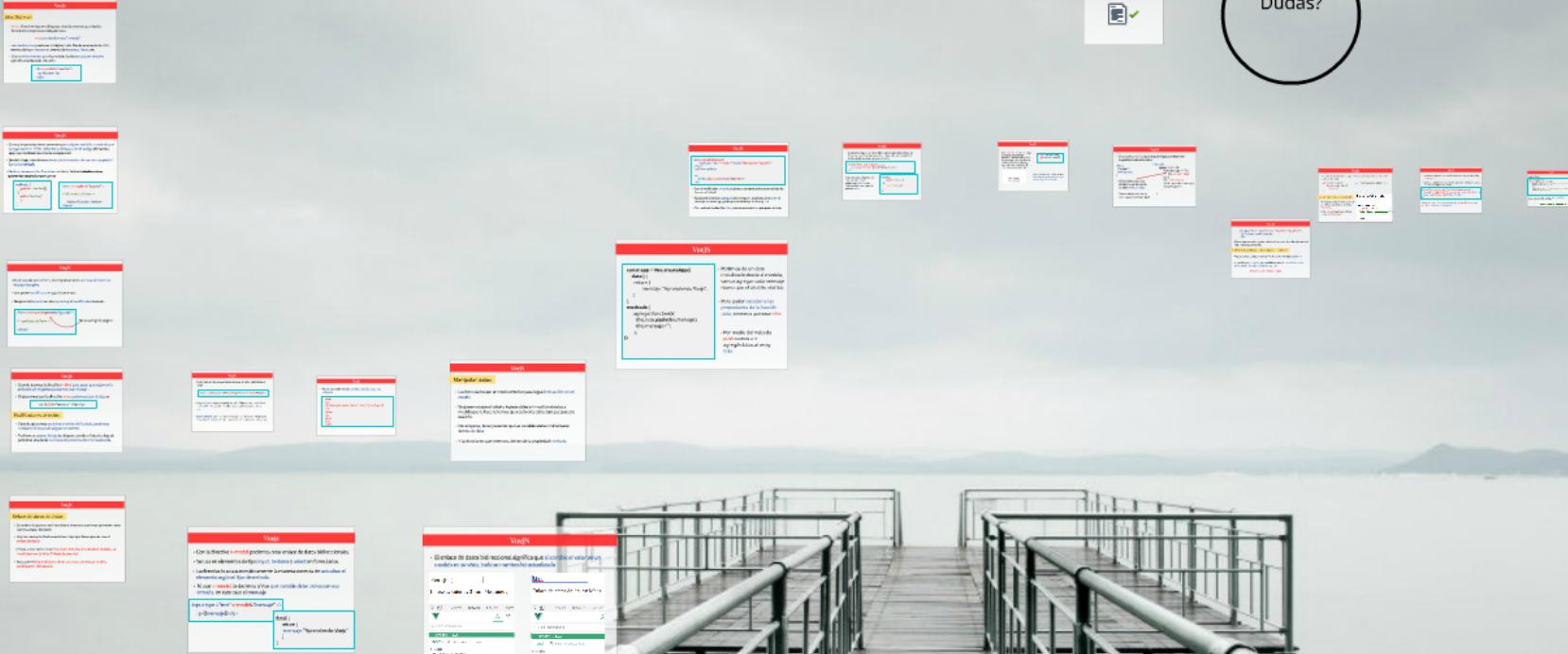
Dudas?



Prezi

ADM

Dudas?



Prezi

Directiva v-on

- **V-on** : Esta directiva se utiliza para vincular eventos a un botón, formulario o input o en cualquier cosa.

v-on:nombreEvento="metodo"

- **nombreEvento=** puede ser cualquier cosa. Puede ser evento de **click**, evento de tipo **mouseover**, evento de **keypress**, **focus**, etc.
- Esto **une los eventos** a un formulario, botón o **cualquier etiqueta** (párrafo, encabezado, div, etc.)

```
<div v-on:click="mostrar">  
  <p>Mostrar</p>  
</div>
```

VueJS

- Es muy importante tener presente que cualquier variable, o método que pongamos en el HTML, debe estar declarado en el código de nuestra app, caso contrario la consola arrojará error.
- Desde la lógica tendremos dentro de la instancia de vue una propiedad llamada **methods**
- Dentro, crearemos las funciones, es decir, tantos métodos como queramos separados por comas.

```
methods : {  
    guardar : function(){  
    },  
    editar: function(){  
    }  
}
```

```
<form v-on:submit="guardar">  
  
    <!--Elementos del form-->  
  
    <button>Guardar</button>  
</form>
```

VueJS

- En el caso de ser un **form**, el comportamiento al enviar el mismo es **recargar la página**.
- Vue posee **modificadores** para los eventos.
- Después del **evento** se coloca **punto** y el **modificador** deseado.

```
<form v-on:submit.prevent="guardar">  
  <!--controles del form-->  
</form>
```

No se recarga la página

VueJS

- Cuando usamos la directiva **v-bind** para pasar por argumento atributos de etiquetas podíamos usar el atajo :
- Si queremos usar la directiva **v-on** podemos usar el atajo **@**

```
<p @click="mostrar">Ver</p>
```

Modificadores de teclas

- Cuando queremos escuchar eventos del teclado, podemos mediante el **keycode** asignar un evento.
- Podríamos asignar **keyup** (se dispara cuando el usuario deja de presionar una tecla) si el usuario presiona determinada tecla.

VueJS

- La función enviar se ejecutará cuando el usuario suelta la tecla enter

```
<input v-on:keyup.enter="enviar" type="text" v-model="dato"/>
```

- Se puede usar cualquier nombre de tecla válido expuesto a través de **KeyboardEvent.key** como modificadores convirtiéndolos a kebab-case.
- **KeyboardEvent.key** es una propiedad que se utiliza en JavaScript para representar la clave (tecla) que fue presionada en un evento de teclado.



VueJS

- Vue ofrece poder trabajar con alias para las teclas más utilizadas:

.enter

.tab

.delete (captura ambas teclas “Delete” y “Backspace”)

.esc

.space

.up

.down

.left

.right

Enlace de datos de 2 vías

- Cuando trabajamos con formularios tenemos que tener presente como vamos a tratar los datos.
- Hay un concepto fundamental en vuejs que tiene que ver con el **enlace de datos**.
- Hasta ahora vimos como **los datos inicializados desde el modelo, se mostraban en la vista. (Enlace de una vía.)**
- Para que el **enlace de datos de 2 vías** ocurra necesitamos de la **participación del usuario**.

Vuejs

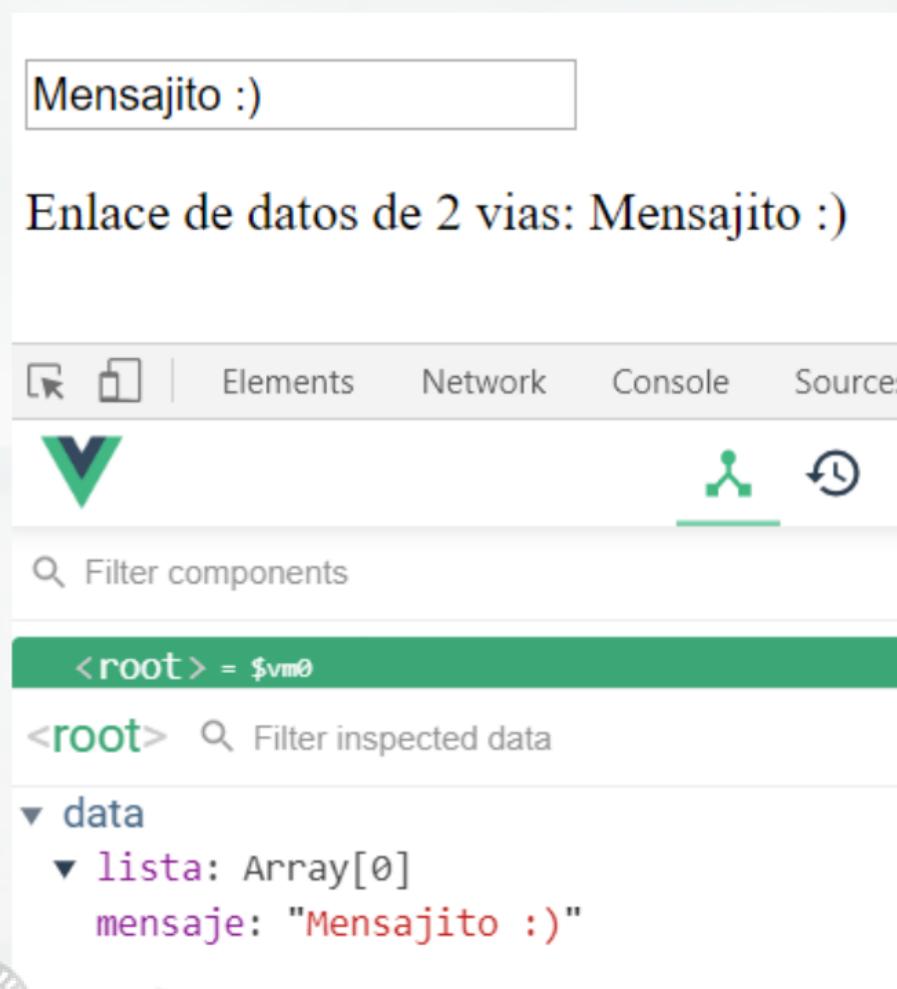
- Con la directiva **v-model** podemos crear enlace de datos bidireccionales.
- Se usa en elementos de tipo **input**, **textarea** y **select** en formularios.
- La directiva busca automáticamente la manera correcta de **actualizar el elemento** según el tipo de entrada.
- Al usar **v-model**, le decimos a Vue **qué variable** debe unirse con esa entrada, en este caso el **mensaje**

```
<input type ="text" v-model="mensaje" />  
<p>{{mensaje}}</p>
```

```
data() {  
    return {  
        mensaje: "Aprendiendo Vuejs"  
    }  
}
```

VueJS

- El enlace de datos bidireccional significa que *si cambia el valor de un modelo en su vista, todo se mantendrá actualizado*



Mensajito :)

Enlace de datos de 2 vias: Mensajito :)

Elements Network Console Source

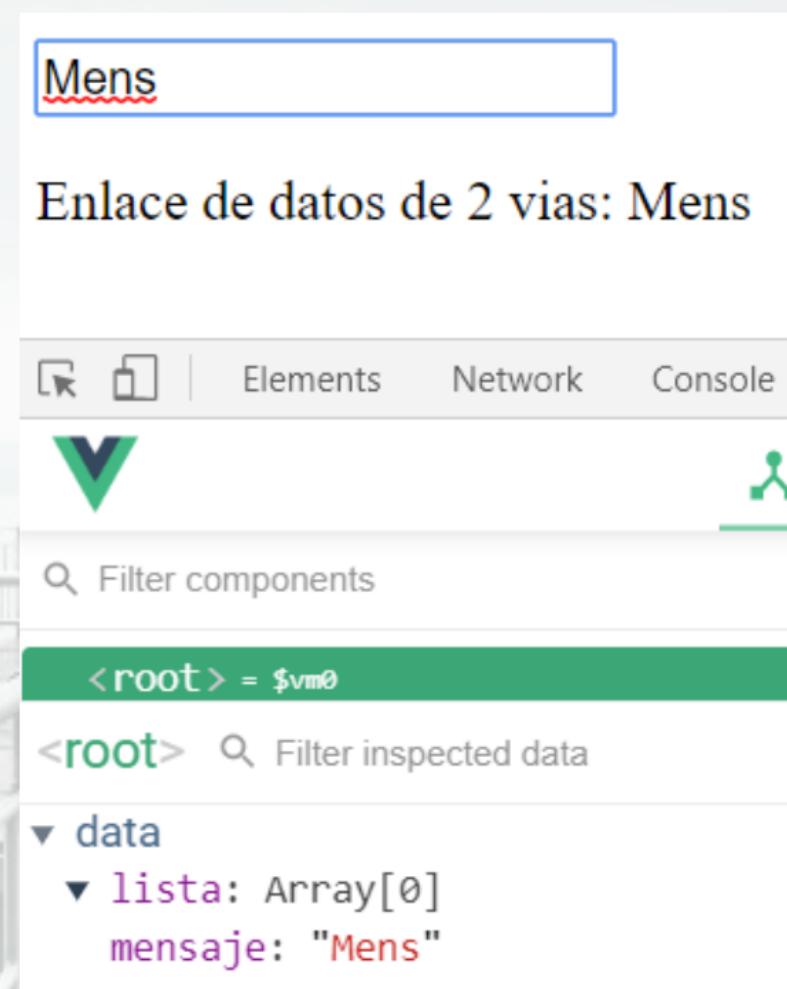
<root> = \$vm0

<root> Filter inspected data

▼ data

 ▼ lista: Array[0]

 mensaje: "Mensajito :)"



Mens

Enlace de datos de 2 vias: Mens

Elements Network Console

<root> = \$vm0

<root> Filter inspected data

▼ data

 ▼ lista: Array[0]

 mensaje: "Mens"

Manipular datos

- Los formularios son un medio efectivo para lograr interacción con el usuario
- Si queremos que el usuario ingrese datos e ir mostrándoselos a medida que lo hace tenemos que resolver la estructura que poseerá esa info.
- Por empezar, tener presente que las variables deben inicializarse dentro de data.
- Y las funciones que creermos, dentro de la propiedad methods.



VueJS

```
const app = Vue.createApp({  
  data() {  
    return {  
      mensaje: "Aprendiendo Vuejs",  
    }  
  },  
  methods:{  
    agregar:function(){  
      this.lista.push(this.mensaje);  
      this.mensaje="";  
    },  
  }  
})
```

- Partimos de un dato inicializado desde el modelo, vamos agregar cada mensaje nuevo que el usuario escriba.
- Para poder acceder a las propiedades de la función `data`, tenemos que usar `this`
- Por medio del método `push` vamos a ir agregándolos al array `lista`.

VueJS

```
<form v-on:submit.prevent>
  <input type ="text" v-model="mensaje" @keyup.enter="agregar"/>
</form>
<p> {{mensaje}}</p>

<ul>
  <li v-for="(item, index) in lista">{{item}}</li>
</ul>
```

- Con el modificador `.prevent` anulamos el comportamiento por defecto del `form` en el `submit`.
- Ejecutando la función `agregar`, cada vez que el usuario presione `enter` el mensaje lo vamos agregando por medio del `push` al array `lista`
- Por medio de la directiva `v-for`, recorremos cada `item` que posea el array

VueJS

- Si queremos lograr más interacción y que el usuario pueda borrar los datos que fue ingresando, necesitamos crear otra función y desde la vista un botón asociado a la misma función.

```
<li v-for="(item, index) in lista">{{item}}  
    <button @click="borrar(index)">Borrar</button>  
</li>
```

Cada vez que escribamos una función, dentro de la propiedad **methods** cada función debe estar separada por una coma.

```
methods:{  
    agregar : function(){  
    },  
    borrar : function(){  
    }  
}
```

VueJS

- El **método splice** nos permite **elegir el elemento del array que queramos eliminar, (index)** como segundo argumento le indicamos cuántos elementos a eliminar a partir del número del index. (En este caso, ese elemento solo.)

```
borrar : function(index){  
    this.lista.splice(index,1)  
}
```

A screenshot of a Vue.js application. It displays a list of items: "Uno" and "Tres". Each item has a "Borrar" button next to it. The "Borrar" button for "Uno" is highlighted with a red border.

- Uno Borrar
- Tres Borrar

Una vez eliminado el elemento Dos, los elementos que quedan aún en el array se seguirán mostrando.

VueJS

- Si quisieramos verificar que el usuario ingresa un dato y no guarda un elemento vacío:

```
data: {  
  mensaje:"",  
  verificar:true  
}
```

- Si la variable esta vacía, cambiamos el valor de la variable **verificar** a **false**
- Caso contrario **verificar** es **true**, y podemos hacer push

```
methods:{  
  agregar: function(){  
    if (this.mensaje == "") {  
      this.verificar = false  
    }else{  
      this.verificar = true  
      this.lista.push(this.mensaje);  
      this.mensaje="";  
    }  
  }  
}
```

VueJS

```
<div class="error" :class="verificar ? 'classBien' : 'classError'">  
  <p>No puede quedar vacío</p>  
</div>
```

- El mensaje solo se le mostrará al usuario en caso de soltar el enter y el input no tenga contenido.

Cuántos elementos fue ingresando el usuario?

- Desde el objeto data podemos inicializar una variable total en 0
- A medida que se haga el push establecer que el valor de total va a ser el .length de lo que tenga el array lista

```
this.total= this.lista.length;
```

VueJS

- Cuando el usuario borre algún elemento deberíamos ir restando del valor de total

```
borrar: function(index){  
    this.lista.splice(index,1)  
    this.total --  
},
```

Lista de elementos seleccionable

- Por defecto el usuario ingresa tareas pendientes, asociadas a una clase "pendientes"
- Y si selecciona alguna cambiamos la clase a "terminadas"

<p> Datos Ingresados: {{total}}</p>

Lista Pendientes:

Enlace de datos de 2 vias:

Concirnar tacos, indice: 0

Ordenar escritorio, indice: 1

Regar plantas, indice: 2

Borrar

Borrar

Borrar

Datos Ingresados: 3

Terminadas : 1

VueJS

- En este caso vamos a crear un objeto que contenga 2 propiedades, **mensaje** y **css**
- Cuando el usuario vaya cargando las nuevas tareas iremos haciendo push de estos objetos al array lista

```
<li v-for="(item, index) in lista" :key="index" :class="item.css" >
  <span @click="toggle(index)">{{item.mensaje}}, indice: {{index}}</span>
  <button @click="borrar(index)">Borrar</button>
</li>
```

:class="item.css --> lo definiremos con la clase **pendientes** cada vez que el usuario agregue un elemento

VueJS

- Desde la lógica:

```
toggle: function(index){  
  if (this.lista[index].css == "pendientes"){  
    this.sumarTerminadas ++; //sumará y su css pasará a terminadas  
    this.lista[index].css = 'terminadas'  
  }else{  
    this.sumarTerminadas --; //restará y el su css pasará a pendientes  
    this.lista[index].css = 'pendientes'  
  }  
}
```

this.lista[index] --> Nos va a permitir afectar a la posición de ese elemento que estamos haciendo click

Ir a la ferreteria, indice: 0
Limpiar estantes, indice: 1

Borrar

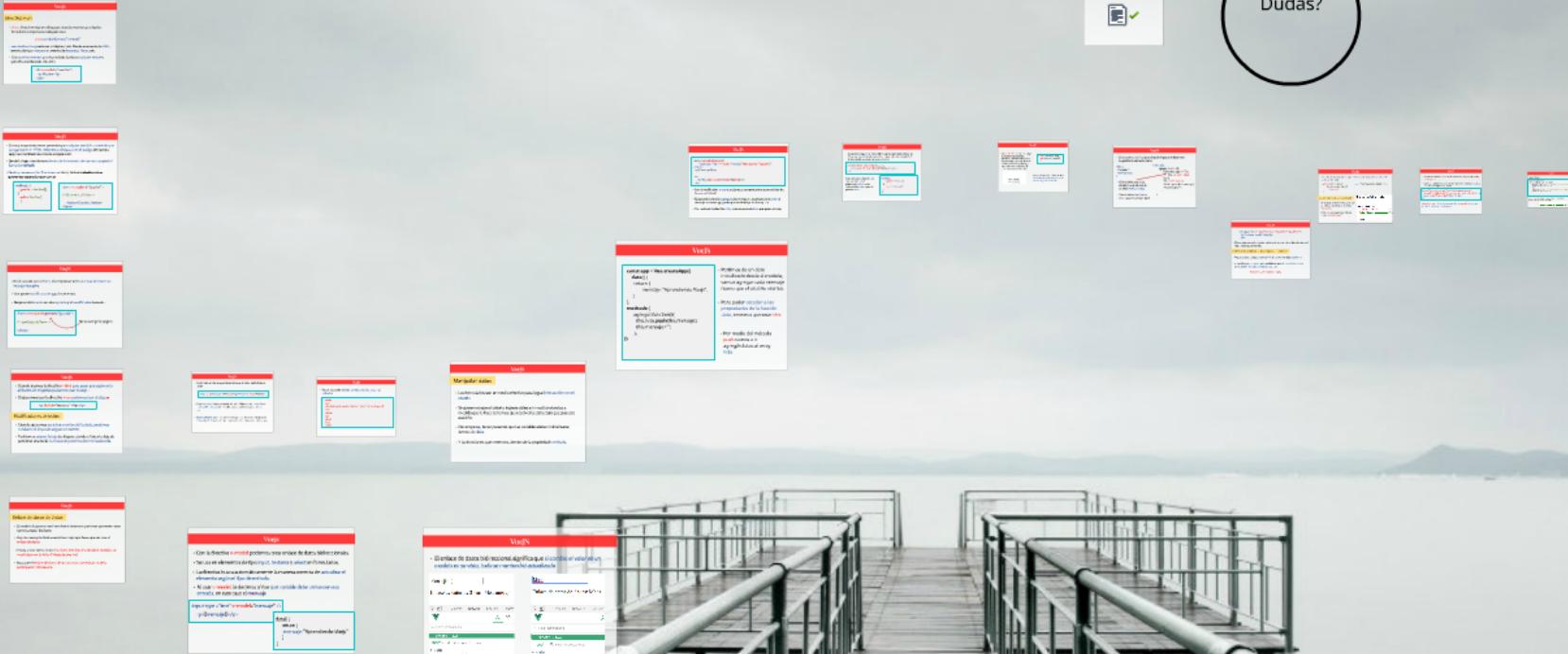
Borrar

Dudas?



ADM

Dudas?



Prezi