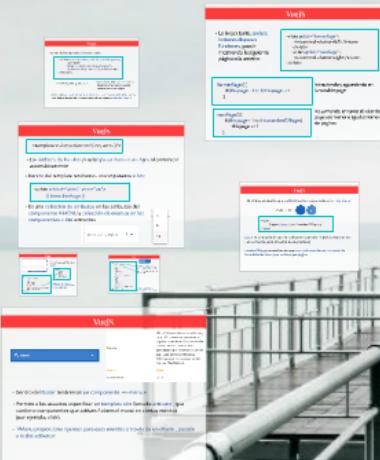
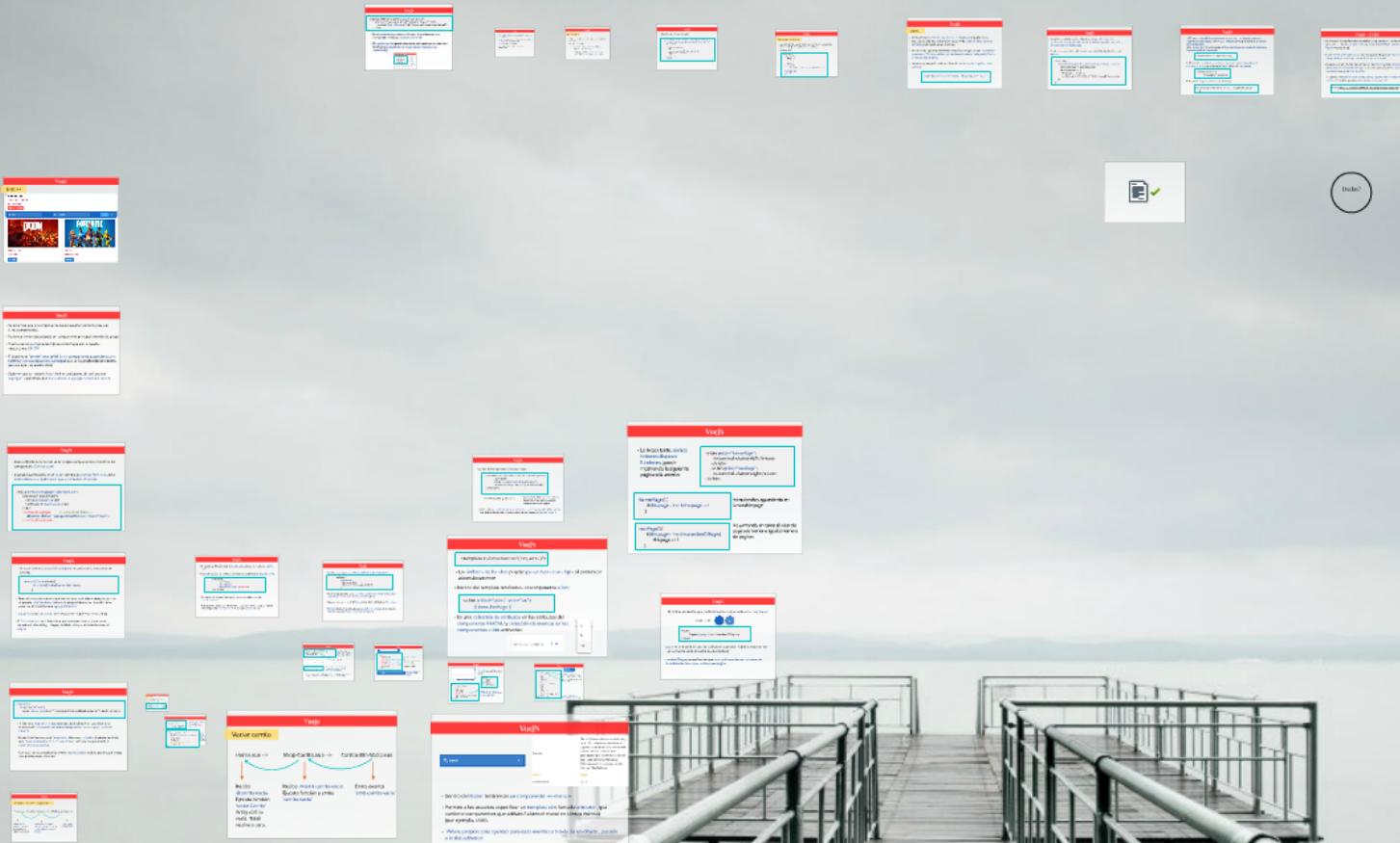


ADM



ADM



Fetch - CORS

- AJAX se puede implementar en cualquier aplicación de JavaScript mediante el uso de **API web nativas**, incluida **XMLHttpRequest** o la **API Fetch** (más reciente)
 - La API fetch utiliza **promesas**, es una operación que aún no se ha completado pero se espera que lo haga en el futuro.
 - Devuelve un objeto con dos métodos, uno **then()** y otro **catch()** a los que **pasaremos una función** que será invocada cuando se **obtenga la respuesta** o se **produzca un error**.
- 1 - Llamamos a **fetch()** con la URL a la que queremos acceder como **parámetro**. Esta llamada nos devuelve una promesa.

```
fetch("http://vueprueba.000webhostapp.com/api/api.php")
```

VueJS

2- El método **then()** de esa promesa nos entrega un **objeto response**.

A partir de este objeto llamamos al **método json()** para obtener el **cuerpo de la respuesta**.

Este **método json()** nos devuelve **otra promesa que se resolverá cuando se haya obtenido el contenido**.

```
.then(response => response.json())
```

3 - El método **then()** de la 2º promesa recibe el cuerpo devuelto por la promesa anterior y guardamos el resultado en el **array juegos**.

```
.then(response => {
    this.juegos = response;
```

4 - Finalmente guardamos en LocalStorage

```
localStorage.setItem("dato",JSON.stringify(this.juegos))
})
```

VueJS

- Nuestro archivo php estará alojado en un servidor, con datos inicializados para que desde nuestro js podamos pedirlos por fetch y almacenarlos en localStorage.
- En el ciclo de vida mounted haremos esta petición, desde al archivo App.vue

```
mounted(){
    fetch("http://vueprueba.000webhostapp.com/api/api.php")
        .then(response => response.json())
        .then(response => {
            this.juegos = response;
            localStorage.setItem("dato",JSON.stringify(this.juegos))
        })
}
```

VueJS

CORS

- Es el acrónimo de *Cross-origin resource sharing* (compartición de recursos de distintos orígenes) y nos permite obtener datos de otros dominios para nuestras aplicaciones.
- Es una tecnología implementada a nivel de navegador que *intercambia cabeceras HTTP con el servidor de destino para saber si debe permitir o no el intercambio de datos*
- La sintaxis para poder solicitar datos de archivos php alojados en un servidor:

```
<?php header(" Access-Control-Allow-Origin: * " ); ?>
```

Componente HelloWorld

- Este componente será el responsable de mostrar al usuario la información que se obtuvo desde el servidor.
- Lo importante :

```
export default {
  name: 'HelloWorld',
  data: () => ({
    local:[],
  }),
  methods:{},
  ver: function(){
    this.local=JSON.parse(localStorage.getItem("dato"))
  },
  mounted:function(){
    this.ver();
  },
}
```

VueJS

- Dentro de su template la iteración

```
<v-card width="600" v-for="item in local" :key="item.id" class="centrado">
  <img :src="" http://vueprueba.000webhostapp.com/api/ + item.img
  " :alt="item.alt" />

  <v-card-title primary-title>
    <div>
      <h3 class="headline mb-0">{{item.nombre}}</h3>
      <div> {{item.detalle}} </div>
    </div>
  </v-card-title>
</v-card>
```

Api File HTML5

- Permite seleccionar archivos locales y luego leer el contenido de esos archivos
- Se puede utilizar esta Api:
 - Crear una vista previa en **miniatura** de imágenes mientras se envían al servidor
 - Permitir que una aplicación **guarde una referencia** de un archivo mientras el usuario está sin conexión.

VueJS

- Hay **tres atributos del objeto File** que contienen información útil sobre el archivo.
 - **name**: nombre del archivo (String). Este es solo el nombre del archivo y no incluye ninguna información de ruta.
 - **size**: tamaño del archivo en bytes.
 - **type**: tipo MIME del archivo (string) o "" si no se pudo determinar el tipo.

VueJS

```
<input type="file" id="imagen" @change="readFileInput">
  <div class="imagen-preview" v-if="form_data.imagen !== null">
    
  </div>
```

- En el **evento change** vamos a llamar a la function que será responsable de darnos **los datos de ese file**
- El **input de tipo file** puede seleccionar varios archivos, en este caso tenemos un **solo archivo** por lo que solo es necesario usar **input.files[0]**.

VueJS	
<pre>readFileInput: function(ev) { const file = ev.target.files[0]; const fr = new FileReader(); fr.addEventListener('load', () => [this.form_data.imagen = fr.result;]); fr.readAsDataURL(file); },</pre>	<p>Obtenemos el file que seleccionó el usuario</p> <p>Creamos un nuevo FileObject:</p> <p>Agregamos un controlador de eventos para manejar el load evento del obj FileReader</p> <p>Método de lectura para leer el contenido de un archivo.</p> <p>• El método <code>readAsDataURL()</code> devuelve un objeto con la propiedad <code>result</code> que contiene los datos como data: URL. El data:URL representa los datos del archivo como una cadena codificada en base64.</p>

VueJS

```
readFileInput: function(ev) {  
    const file = ev.target.files[0];  
  
    const fr = new FileReader();  
    fr.addEventListener('load', () => {  
        this.form_data.imagen = fr.result;  
    });  
    fr.readAsDataURL(file);  
},
```

Obtenemos el file que seleccionó el usuario

Creamos un nuevo FileObject:

Agregamos un controlador de eventos para manejar el load evento del obj FileReader

Método de lectura para leer el contenido de un archivo.

- El método `readAsDataURL()` devuelve un objeto con la propiedad `result` que contiene los datos como `data: URL`. El `data:URL` representa los datos del archivo como una cadena codificada en base64.

Emit ++

PRODUCTOS

CANTIDAD: 2 JUEGOS

TOTAL: \$5500

VACIAR CARRITO

Buscar

Ordenar por

↑ ↓



Doom Exodus

Precio: \$500



Fortnite

Precio: \$5000

AGREGAR

VueJS

- Ya sabemos que un componente suele necesitar comunicarse con otros componentes.
- Podemos enviar datos desde un componente principal a través de props
- Y para que un componente hijo se comunique con su padre, necesitamos **EMITIR**
- El objetivo es "**emitir**" una señal de un **componente secundario** para **notificar a un componente principal** que se ha **producido un evento** (por ejemplo, un evento click).
- Cada vez que un usuario hace click en cualquiera de los botones "Aregar" , queremos que ese artículo se agregue a nuestro carrito.

VueJS

- Cada artículo de la tienda es su propio componente (el nombre del componente **Carrito-Item**)
- Cuando hacemos clic en el **botón** dentro de **Carrito-Item.vue**, debe emitir datos a su padre para que se actualice el carrito.

```

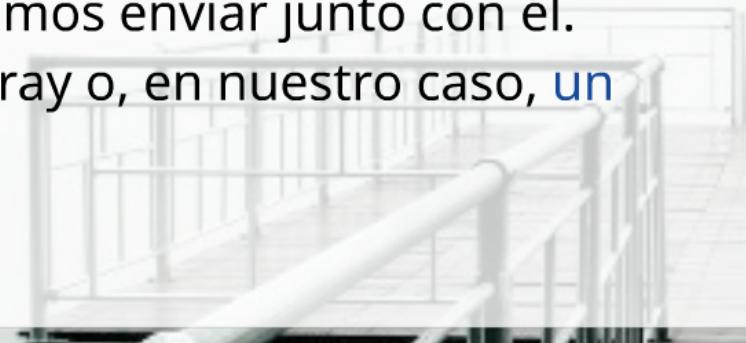
  <div class="item_detalle">
    <p>{{item.nombre}}</p>
    <p>Precio: ${{item.precio}}</p>
  </div>
  <carrito-btn-agregar      <!-- componente boton -->
    @button-clicked="agregarAlCarrito(item)" :item="item">
  </carrito-btn-agregar>
```

VueJS

- Al hacer click en `agregarAlCarrito(item)` se activa nuestro detector de eventos.

```
agregarAlCarrito(item) {  
    this.$emit('actualizar-carrito', item)  
},
```

- Toma el elemento como un parámetro (este es el objeto completo que se ha pasado `<Carrito-Item>` como una props). Cuando se hace click en el botón, se activa la función `agregarAlCarrito`:
- `this.$emit` tiene un `string` como 1º parámetro (nombre de la señal).
- El `2º parámetro`, será algún dato que queramos enviar junto con él. (puede ser otro string, integer, variable, array o, en nuestro caso, `un object`.



VueJS

- En nuestro archivo principal `Home.vue` agregamos el componente `<carrito-item>`.
- Posee un detector de eventos personalizado escuchando a `actualizar-carrito`.

```
<carrito-item  
    :item="item"  
    :key="item.id"  
    @actualizar-carrito="actualizar">  
</carrito-item>
```

- El detector de eventos está esperando que se active el evento `actualizar-carrito`
- Esto pasará cuando se emita el string '`actualizar-carrito`' desde el interior del componente `Carrito-Item`, en el archivo `Carrito-item.vue`.

VueJS

- Una vez que detecte el evento se activará función "actualizar" :

```
methods: {  
    actualizar(e) {  
        this.cart.push(e);  
        this.total = this.shoppingCartTotal;  
    },
```

- El evento (e) que toma es el elemento que pusimos inicialmente como segundo parámetro cuando llamamos this.\$emit.
- Luego lo pushea en el array this.cart (que fue inicializado en Home.vue)
- this.total también se actualiza para devolver el resultado de la función this.shoppingCartTotal (propiedad computada)

VueJS

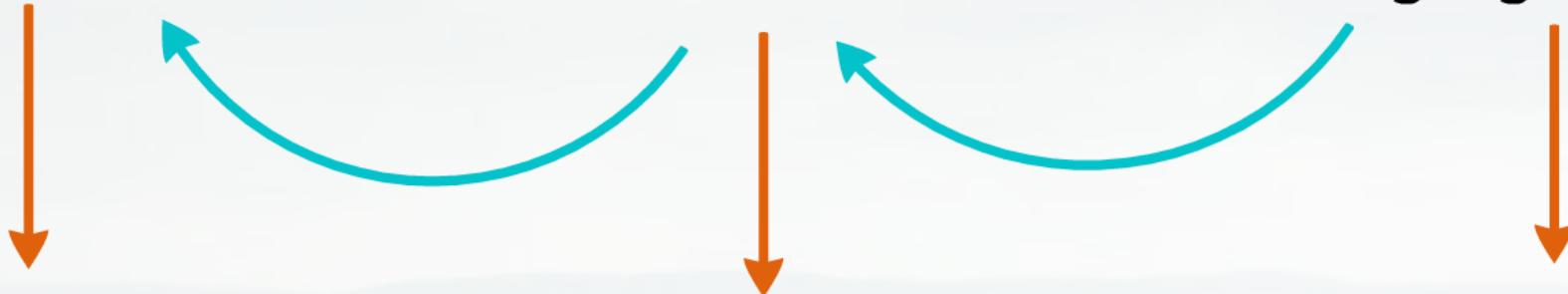
```
computed: {  
    shoppingCartTotal() {  
        return this.cart.map(item => item.precio).reduce((total, amount) => total + amount);  
    },
```

- Al llamar a `map` en un `array`, este ejecuta el `callback` en cada elemento dentro de él, **retornando un nuevo array** con los valores que el `callback` retorna.
- Es decir de `item` nos da el `item.precio`. Hacemos un `reduce` (método de `array`) que es **un acumulador en el 2º argumento** - `amount`- ira guardando la acumulación de totales.
- Con esto iremos actualizando el valor de `this.total` a medida que el usuario vaya agregando juegos al carrito



Comunicación entre componentes

Home.vue -->Carrito-Item.vue -->Carrito-Btn-Agregar.vue



recibe **evento**
actualizar-carrito.
hace push y va
sumando el precio

recibe **evento button-
clicked**, ejecuta función y
emite actualizar-carrito
(item)

Emite evento
button-clicked

Vaciar carrito

Home.vue -->



Recibe
@carrito-vacio.
Ejecuta función
'vaciarCarrito'
Array cart se
vacía. Total
vuelve a cero.

Shop-Carrito.vue -->



Recibe @emit-carrito-vacio.
Ejecuta función y emite
'carrito-vacio'

Carrito-Btn-Vacio.vue



Emite evento
'emit-carrito-vacio'

v-data-iterator

- Se usa para mostrar datos y comparte la mayor parte de su funcionalidad con el componente `v-data-table`.
- Las características incluyen *clasificación, búsqueda, paginación y selección*.

```
<v-data-iterator  
  :items="items"  
  :items-per-page.sync="itemsPerPage"  
  :page.sync="page"  
  :search="search"
```

los juegos
cantidad de elementos a
mostrar por pagina
desde qué página empieza
nombre del v-model
para buscar

Vuejs

```
:sort-by="sortBy.toLowerCase()"  
:sort-desc="sortDesc"  
hide-default-footer>
```

sortBy (nombre de v-model) lo que ingrese el usuario en minúsculas

nombre de v-model

Items per page: 5 ▾ 1-4 of 6 < >

```
<template v-slot:header
```

slots son un mecanismo de Vue JS que sirve para insertar contenido HTML dentro de los componentes.

- Este contendrá la parte superior para la búsqueda, el select para buscar por keys que hayamos declarado y para cambiar el orden (asc o desc)

Vuejs



```
<v-text-field  
  v-model="search"  
  clearable  
  flat  
  solo-inverted  
  hide-details  
  prepend-inner-icon="mdi-magnify"  
  label="Buscar">  
</v-text-field>
```

El campo se puede borrar

Estilo plano

diseño individual alternativo. (en
foco queda blanco)

Oculta mensajes de error
antepone icono dentro

Ordenar por

Dentro podremos elegir
'keys' para ordenar los
juegos

Vuejs

Nombre

Detalle

Precio

Lanzamiento

Inicializado acorde a las propiedades de cada object para ordenar según estos criterios

```
<v-select  
    v-model="sortBy"  
    flat  
    solo-inverted  
    hide-details  
    :items="keys"  
    prepend-inner-icon="mdi-magnify"  
    label="Ordenar por">  
</v-select>
```

keys: [
 'Nombre',
 'Detalle',
 'Precio',
 'Lanzamiento'
],

El objeto puede tener más propiedades, pero no todas serán keys de ordenamiento.

Vuejs

```
<v-btn-toggle  
    v-model="sortDesc"  
    mandatory>  
    <v-btn  
        large  
        depressed  
        color="blue"  
        :value="false">  
        <v-icon>mdi-arrow-up</v-icon>  
    </v-btn>  
    <v-btn  
        large  
        depressed  
        color="blue"  
        :value="true">  
        <v-icon>mdi-arrow-down</v-icon>  
    </v-btn>  
</v-btn-toggle>
```



Inicializado en **false** el **v-model** = "sortDesc" nos mostrará los juegos ordenados en **orden ascendente (A-Z)**

prop **mandatory** (obligatorio) siempre tendrá un value.

Agrupa v-btn, permite alternar entre botones con un solo v-model

Vuejs

```
<template v-slot:default="props">  
  <v-col  
    v-for="item in props.items"  
    :key="item.nombre">
```

- Se puede acceder directamente a las propiedades de los subcomponentes mediante props.

```
<v-list-item  
  v-for="(key, index) in filteredKeys"  
  :key="index">  
  <v-list-item-content :class="{ 'orange--text': sortBy === key }">  
    {{ key }}:  
  </v-list-item-content>  
  <v-list-item-content  
    class="align-end"  
    :class="{ 'orange--text': sortBy === key }">  
    {{ item[key.toLowerCase()] }}  
  </v-list-item-content>  
</v-list-item>
```

Según el key elegido, le pondrá una clase y quedará resaltado el key y el item



VueJS



Detalle:

Marvel's Spider-Man es un videojuego de acción y aventura basado en el popular superhéroe homónimo de la editorial Marvel Comics. Fue desarrollado por Insomniac Games y publicado por Sony Interactive Entertainment en exclusiva para la consola PlayStation 4.

Precio:

1000

Lanzamiento:

2018

- Dentro del **footer** tendremos **un componente <v-menu>**
- Permite a los usuarios especificar un **template slot** llamado **activator**, que contiene componentes que activan / abren el menú en ciertos eventos (por ejemplo, click).
- **VMenu** proporciona oyentes para esos eventos a través de un objeto , pasado a la slot activator:

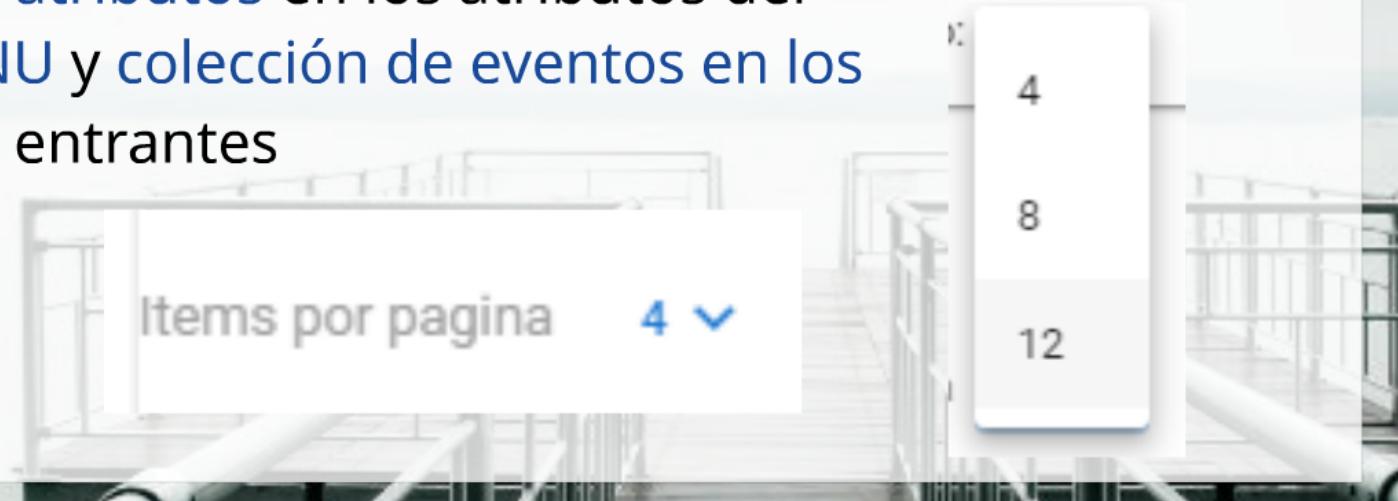
VueJS

```
<template v-slot:activator="{ on, attrs }">
```

- Los **ámbitos de los slots** pueden **pasar datos a sus hijos** al comunicar automáticamente
- Dentro del template tendremos un componente **v-btn**:

```
<v-btn v-bind="attrs" v-on="on">  
    {{ itemsPerPage }}
```

- Es una **colección de atributos** en los atributos del componente V-MENU y **colección de eventos** en los componentes v-btn entrantes



VueJS

- Dentro del componente `v-list` tendremos:

```
<v-list-item v-for="(number, index) in itemsPerPageArray"  
    :key="index"  
    @click="updateItemsPerPage(number)">  
    <v-list-item-title>{{ number }}</v-list-item-title>  
</v-list-item>
```

itemsPerPageArray: [4, 8, 12],

- Lo tenemos inicializado, nosotros determinamos cuantos juegos mostraremos por página

- La función recibe el `número` y actualizará el `valor` que nosotros inicializamos por defecto dentro de la función `data` del componente `itemsPerPage: 4,`

VueJS

- El último elemento que conforma este componente es el [paginador](#)

Página 1 de 1



```
<span>  
    Página {{ page }} de {{ numberOfPages }}  
</span>
```

- [page](#) es una variable que inicializamos con valor 1 (debe arrancar en 1
 - el comienzo de la muestra de elementos)
- [numberOfPages](#) es una función que [retorna el redondeo de la division de la cantidad de items y de los items por página](#)

VueJS

- Lo importante, ambos botones disparan funciones, para ir mostrando la siguiente página o la anterior

```
<v-btn @click="formerPage">  
  <v-icon>mdi-chevron-left</v-icon>  
</v-btn>  
<v-btn @click="nextPage" >  
  <v-icon>mdi-chevron-right</v-icon>  
</v-btn>
```

```
formerPage () {  
  if (this.page - 1 >= 1) this.page -= 1  
},
```

Irá restando y aguardando en la variable page

```
nextPage () {  
  if (this.page + 1 <= this.numberOfPages)  
    this.page += 1  
},
```

Irá sumando, en tanto el valor de page sea menor o igual al número de páginas

Dudas?



ADM

