Template

```
<div v-if="this.type == 'default'">
        <div class="btn loadmore"
             v-if="(errorsCount > 15) || (!infiniteLoading && !loading &&
this.currentPage < this.lastPage)"
             @click="fetchData">
            {{ this.loadBtnValue }}
        </div>

        <div class="loader" v-if="loading && this.currentPage <
this.lastPage">
            <span v-if="errorsCount < 1 ">{{ this.loadingText }}</span>
            <span v-if="errorsCount > 0">{{ this.fetchErrMsg }}</span>
            <div class="loader-line">
                <div class="line"></div>
            </div>
        </div>
</div>
```

Родительский div рендерится только если передаваемое компоненту свойство props равно 'default'.

Первый дочерний div с кнопкой загрузки монтируется в DOM при выполнении двух условий:

```
(errorsCount > 15) || (!infiniteLoading && !loading && this.currentPage
< this.lastPage)"
```

Если значение свойства errorsCount больше 15 или infiniteLoading и loading возвращают false, а номер текущей страницы пагинации меньше номера последней.

По клику на этот div вызывается метод **fetchData**.

Второй дочерний div, являющийся прелоадером, рендерится при условии, что свойство loading возвращают true и номер текущей страницы пагинации меньше номера последней.

Data

```
loading: false
```

Флаг состояния процесса загрузки данных. Используется для регулировки отображения кнопки «Загрузить еще» и прелоадера, а также как одно из условий вызова функции **fetchData** в методе load, который вызывается обработчиком события click в хуке mounted.

По умолчанию принимает значение false. Значение true присваивается при вызове метода **fetchData**.

```
infiniteLoading: true
```

Флаг, регулирующий каким образом происходит пагинация. По умолчанию принимает значение true, что позволяет осуществлять так называемую «бесконечную загрузку», при которой содержимое каждой следующей страницы подгружается при скролле. При достижении определенной страницы пагинации (вычисляется это в теле метода **fetchData**) свойству присваивается значения false, после чего пагинация выполняется не при скролле, а при клике на кнопку «Загрузить еще».

```
fetchPath: this.apiPath + '/' + this.model
```

Свойство, в котором собирается url, на который отправляется запрос данных в методе **fetchData**.

```
isReseted: true
```

Флаг, использующийся в свойстве **page()** в computed properties. В зависимости от значения, которое он возвращает – формируется вид get-параметра page. По умолчанию принимает значение true и изменить его может во время

прослушивания пользовательского события filtered, которое имитирует плагин threeStageFilter.

```
startPage: this.start,
currentPage: this.start,
lastPage: this.last
```

Свойства, в которых записываются номера стартовой (с которой начинается пагинация), текущей (т.е. последней полученной страницы с данными от сервера) и последней страниц пагинации соответственно. Изначально принимают соответствующие значения свойств props, которые передаются компоненту сервером.

```
errorsCount: 0
```

Свойство, хранящее количество вернувших ошибку запросов подряд

Props

```
apiPath: String
```

Ссылка на главную страницу api

```
start: String
last: String
```

Номера стартовой и последней страниц пагинации контента, с которым взаимодействует компонент.

```
container: String
```

id контейнера (элемента DOM), в который будут помещаться полученные от сервера данные

```
type: String
```

тип пагинации

```
model: String,
```

Название модели данных Laravel, контент которой пагинируется данным экземпляром компонента.

```
loadBtnValue: String,
loadingText: String
```

Текст, который отображается на кнопке «Загрузить еще» и текст, использующийся в прелоадере. Данные свойства необходимы для поддержки мультиязычности компонента.

```
fetchErrMsg: String
```

Текст сообщения об ошибке, возникающей при неудачно обработанной запросе данных.

Computed

```
nextPage() {
   return (+this.currentPage + 1)
}
```

Функция возвращает номер следующей страницы (инкриминируя номер текущей), на которую будет послан запрос данных.

```
page() {
    return ( ( this.isReseted ) ? '?page='+this.nextPage :
'&page='+this.nextPage )
}
```

Функция, вычисляющая в каком виде в url, на который отправляется зарос данных, передается параметр page – как единичный параметр (?page=) либо как добавочный (&page=). Необходимость в этом нужна ввиду взаимодействия с компонентом фильтрации theeStageFilter. Если он имитирует

пользовательское событие 'filtered', то данный компонент 'слушает' его (происходит это в хуке created) и в качестве аргумента принимает параметр isReseted, который сообщает о том, отфильтровано ли данные по неким параметрам или фильтр был сброшен и на странице отображен исходный контент. Если контент отфильтрован – параметр isReseted отдает false, в следствии чего к url запроса, содержащему параметры фильтрации, добавляется дополнительный в виде '&page='. Если же фильтр сброшен, то отдаваемое параметром значение равно true и к url добавляется единственный get-параметр '?page='.

Methods

Функция, отправляющая запрос серверу на получение контента:

```
fetchData() {
                this.loading = true;
                if(this.errorsCount > 15) {
                    this.errorsCount = 0
                };
                this.$http.get( this.fetchPath + this.page ).then(response =>
{
                        document.getElementById(this.container)
                                .insertAdjacentHTML('beforeEnd',
response.data.content);
                        this.currentPage++;
                        this.nextPage++;
                        this.errorsCount = 0;
                        this.loading = false;
                        this.infiniteLoading = ( (this.currentPage -
this.startPage ) > 1 ) ? false : true
                }, error => {
                    console.log(error);
                    this.errorsCount++;
                    if(this.errorsCount <=15 ){
```

```
                setTimeout( () => { this.fetchData() }, 2000 )
                } else {
                    this.loading = false;
                }
            })
        },
```

В первую очередь перед отправкой запроса присваиваем свойству loading значение true и сбрасываем счетчик количества неудачных запросов, если на момент вызова метода он возвращает значение больше 15 (это позволяет пользователю в ручную повторить попытку загрузки данных, если автоматическое кол-во запросов превысило норму в 15 попыток).

Затем отправляется get запрос на url, состоящий из значения свойства data **fetchPath** и свойства computed properties **page**.

Если запрос успешно обработан сервером, то мы обращаемся к элементу DOM с id, переданным свойством props **container** и помещаем полученные данные в конец этого элемента (т.е. после содержимого, которое в нем уже имеется). Так же мы инкриминируем свойства **currentPage** и **nextPage**, сбрасываем счетчик ошибок до нуля и устанавливаем свойству **loading** значение false, свидетельствующее об удачном окончании загрузки данных и демонтирующее прелоадер. Затем мы проверяем разницу между между текущей страницей пагинации и стартовой и, если она больше 10, то мы устанавливаем флагу **infiniteLoading** значение false, отключая автоподгрузку данных при скролле странице и монтируя в DOM кнопку «Загрузить еще».

Если запрос вернул ошибку, то выбрасываем в консоль тело этой ошибки, записываем в свойство **loading** значение false и инкриминируем **errorsCounter**. Если количество неудачно отправленных/обработанных запросов не достигло 15 – рекурсивно вызываем функцию каждые 2 секунды. В противном случае – прерываем рекурсию и присваиваем свойству loading значение false. В то же время в DOM монтируется кнопка загрузки материала,

позволяющая пользователю вручную перезапустить функцию, т.к. в контексте подобного вызова значение **errorsCounter** сбросится до нуля.

Функция **load**, которая вызывается при регистрации обработчика события 'scroll' в хуке created, вычисляет в какой момент времени вызывать метод **fetchData**.

```
load() {
                let scrollTop = window.pageYOffset ||
document.documentElement.scrollTop,
                    wrapperHeight =
document.getElementById(this.container).offsetHeight,
                    diffHeight = wrapperHeight - 1000;
                if(diffHeight <= scrollTop && !this.loading
                                           && this.errorsCount < 1
                                           && this.infiniteLoading
                                           && (this.lastPage >
this.currentPage)) {
                    return this.fetchData()
                }
}
```

В переменную **scrollTop** мы записываем значение количества пикселей, прокрученных от верха страницы.

В переменную **wrapperHeight** записываем высоту контейнера с контентом.

В **diffHeight** мы записываем выражение, суть которого ясна из следующей строчки – условного выражения, в котором оно используется.

Если разница пикселей, записанная в выражение **diffHeight** меньше, либо равна количеству пикселей, на которое прокручена от верха страница (т.е. как только до конца контейнера с контентом остается 1000 пикселей либо меньше), свойства **loading** и **infiniteLoading** возвращают значения false и true соответственно, количество вернувшихся с ошибкой запросов меньше 1 и

номер последней страницы пагинации больше номера текущей, то вызывается метод **fetchData**.

Server-side declaration

Рендер компонента вызывается добавлением в шаблон экземпляра компонента <infinite-pagination></infinite-pagination>

```
<infinite-pagination api-path="{{ URL::route('api') }}"
                     start="{{ $articles->currentPage() }}"
                     last="{{ $articles->lastPage() }}"
                     container="content"
                     model=" название модели, с содержимым которого
взаимодействует компонент "
                     load-btn-value="{{ $loadBtnValue }}"
                     loading-text="{{ $loadingText }}"
                     type="default"
                      fetch-err-msg="{{ $fetchErrMsg }}">
</infinite-pagination>
```

Все аттрибуты экземпляра соответствуют свойствам props. Те, которые написаны в позвоночном-регистре, соответствуют одноименным свойствам props в верблюжемРегистре.

Значение аттрибута **api-path** автоматически возвращает ссылку на страницу api, основываясь на описание роутинга под названием 'api' в файле App/routes/web.php.

Значением аттрибута **model**, как указано выше, является название модели, с содержимым которое взаимодействует компонент.

Значения всех остальных аттрибутов являются переменными, которые при формировании страницы на сервере предоставляются композером infinitePaginationComposer, который вызывается при компоновке определенных представлений Laravel, перечень которых объявляется в методе **composeInfinitePagination** сервис-провайдера ViewComposerServiceProvider.