Database: Pagination

# Introduction

In other frameworks, pagination can be very painful. Laravel's paginator is integrated with the query builder and Eloquent ORM and provides convenient, easy-to-use pagination of database results out of the box. The HTML generated by the paginator is compatible with the Bootstrap CSS framework.

Basic Usage

Paginating Query Builder Results

There are several ways to paginate items. The simplest is by using the `paginate` method on the query builder or an Eloquent query. The `paginate` method automatically takes care of setting the proper limit and offset based on the current page being viewed by the user. By default, the current page is detected by the value of the `page` query string argument on the HTTP request. This value is automatically detected by Laravel, and is also automatically inserted into links generated by the paginator.

In this example, the only argument passed to the `paginate` method is the number of items you would like displayed "per page". In this case, let's specify that we would like to display `15` items per page:

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    /**
     * Show all of the users for the application.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::table('users')->paginate(15);

        return view('user.index', ['users' => $users]);
    }
}
```

!

Currently, pagination operations that use a `groupBy` statement cannot be executed efficiently by Laravel. If you need to use a `groupBy` with a paginated result set, it is recommended that you query the database and create a paginator manually.

"Simple Pagination"

If you only need to display simple "Next" and "Previous" links in your pagination view, you may use the `simplePaginate` method to perform a more efficient query. This is very useful for large datasets when you do not need to display a link for each page number when rendering your view:

```
$users = DB::table('users')->simplePaginate(15);
```

Paginating Eloquent Results

You may also paginate Eloquent queries. In this example, we will paginate the `User` model with `15` items per page. As you can see, the syntax is nearly identical to paginating query builder results:

```
$users = App\User::paginate(15);
```

You may call `paginate` after setting other constraints on the query, such as `where` clauses:

```
$users = User::where('votes', '>', 100)->paginate(15);
```

You may also use the `simplePaginate` method when paginating Eloquent models:

```
$users = User::where('votes', '>', 100)->simplePaginate(15);
```

Manually Creating A Paginator

Sometimes you may wish to create a pagination instance manually, passing it an array of items. You may do so by creating either an `Illuminate\Pagination\Paginator` or `Illuminate\Pagination\LengthAwarePaginator` instance, depending on your needs.

The `Paginator` class does not need to know the total number of items in the result set; however, because of this, the class does not have methods for retrieving the index of the last page. The `LengthAwarePaginator` accepts almost the same arguments as the `Paginator`; however, it does require a count of the total number of items in the result set.

In other words, the `Paginator` corresponds to the `simplePaginate` method on the query builder and Eloquent, while the `LengthAwarePaginator` corresponds to the `paginate` method.

! When manually creating a paginator instance, you should manually "slice" the array of results you pass to the paginator. If you're unsure how to do this, check out the array_slice PHP function.

Displaying Pagination Results

When calling the `paginate` method, you will receive an instance of `Illuminate\Pagination\LengthAwarePaginator`. When calling the `simplePaginate` method, you will receive an instance of `Illuminate\Pagination\Paginator`. These objects provide several methods that describe the result set. In addition to these helpers methods, the paginator instances are iterators and may be looped as an array. So, once you have retrieved the results, you may display the results and render the page links using Blade:

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

{{ $users->links() }}
```

The `links` method will render the links to the rest of the pages in the result set. Each of these links will already contain the proper `page` query string variable. Remember, the HTML generated by the `links` method is compatible with the Bootstrap CSS framework.

Customizing The Paginator URI

The `withPath` method allows you to customize the URI used by the paginator when generating links. For example, if you want the paginator to generate links like `http://example.com/custom/url?page=N`, you should pass `custom/url` to the `withPath` method:

```
Route::get('users', function () {
    $users = App\User::paginate(15);

    $users->withPath('custom/url');

    //
});
```

Appending To Pagination Links

You may append to the query string of pagination links using the `appends` method. For example, to append `sort=votes` to each pagination link, you should make the following call to `appends`:

```
{{ $users->appends(['sort' => 'votes'])->links() }}
```

If you wish to append a "hash fragment" to the paginator's URLs, you may use the `fragment` method. For example, to append `#foo` to the end of each pagination link, make the following call to the `fragment` method:

```
{{ $users->fragment('foo')->links() }}
```

Adjusting The Pagination Link Window

You may control how many additional links are displayed on each side of the paginator's URL "window". By default, three links are displayed on each side of the primary paginator links. However, you may control this number using the `onEachSide` method:

```
{{ $users->onEachSide(5)->links() }}
```

Converting Results To JSON

The Laravel paginator result classes implement the `Illuminate\Contracts\Support\Jsonable` Interface contract and expose the `toJson` method, so it's very easy to convert your pagination results to JSON. You may also convert a paginator instance to JSON by returning it from a route or controller action:

```
Route::get('users', function () {
    return App\User::paginate();
});
```

The JSON from the paginator will include meta information such as `total`, `current_page`, `last_page`, and more. The actual result objects will be available via the `data` key in the JSON array. Here is an example of the JSON created by returning a paginator instance from a route:

```
{
   "total": 50,
   "per_page": 15,
   "current_page": 1,
   "last_page": 4,
   "first_page_url": "http://laravel.app?page=1",
   "last_page_url": "http://laravel.app?page=4",
   "next_page_url": "http://laravel.app?page=2",
   "prev_page_url": null,
   "path": "http://laravel.app",
   "from": 1,
   "to": 15,
   "data":[
        {
            // Result Object
        },
        {
            // Result Object
        }
   ]
}
```

Customizing The Pagination View

By default, the views rendered to display the pagination links are compatible with the Bootstrap CSS framework. However, if you are not using Bootstrap, you are free to define your own views to render these links. When calling the `links` method on a paginator instance, pass the view name as the first argument to the method:

```
{{ $paginator->links('view.name') }}

// Passing data to the view...
{{ $paginator->links('view.name', ['foo' => 'bar']) }}
```

However, the easiest way to customize the pagination views is by exporting them to your `resources/views/vendor` directory using the `vendor:publish` command:

```
php artisan vendor:publish --tag=laravel-pagination
```

This command will place the views in the `resources/views/vendor/pagination` directory. The `bootstrap-4.blade.php` file within this directory corresponds to the default pagination view. You may edit this file to modify the pagination HTML.

If you would like to designate a different file as the default pagination view, you may use the paginator's `defaultView` and `defaultSimpleView` methods within your `AppServiceProvider`:

```
use Illuminate\Pagination\Paginator;

public function boot()
{
    Paginator::defaultView('view-name');

    Paginator::defaultSimpleView('view-name');
}
```

Paginator Instance Methods

Each paginator instance provides additional pagination information via the following methods:

Method	**Description**
`$results->count()`	Get the number of items for the current page.
`$results->currentPage()`	Get the current page number.
`$results->firstItem()`	Get the result number of the first item in the results.
`$results->getOptions()`	Get the paginator options.
`$results->getUrlRange($start, $end)`	Create a range of pagination URLs.
`$results->hasMorePages()`	Determine if there are enough items to split into multiple pages.
`$results->items()`	Get the items for the current page.
`$results->lastItem()`	Get the result number of the last item in the results.
`$results->lastPage()`	Get the page number of the last available page. (Not available when using `simplePaginate`).
`$results->nextPageUrl()`	Get the URL for the next page.
`$results->onFirstPage()`	Determine if the paginator is on the first page.
`$results->perPage()`	The number of items to be shown per page.
`$results->previousPageUrl()`	Get the URL for the previous page.
`$results->total()`	Determine the total number of matching items in the data store. (Not available when using `simplePaginate`).
`$results->url($page)`	Get the URL for a given page number.
`$results->getPageName()`	Get the query string variable used to store the page.
`$results->setPageName($name)`	Set the query string variable used to store the page.

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport
- Scout
- Socialite

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.