
!! UNMAINTAINED !! This package is no longer maintained

Please see Issue #117

Here are some links to alternatives that you may be able to use (I do not guarantee the suitability or the quality of these projects, please ensure you do your own due diligence in ensuring they meet your requirements):

- *TODO*

Laravel 5+ Searchy

Database Searching Made Easy

Searchy is an; easy-to-use, light-weight, MySQL only, Laravel package that makes running user driven searches on data in your models simple and effective. It uses pseudo fuzzy searching and other weighted mechanics depending on the search driver that you have enabled. It requires no other software installed on your server (so can be a little slower than dedicated search programs) but can be set up and ready to go in minutes.

!! Laravel 4 !! Looking for Laravel 4 compatible Searchy? Checkout the 1.0 branch :)

<https://github.com/TomLingham/Laravel-Searchy/tree/1.0>

Installation

Add `"tom-lingham/searchy": "2.*"` to your composer.json file under `require`:

```
1 "require": {  
2     "laravel/framework": "5.*",  
3     "tom-lingham/searchy" : "2.*"  
4 }
```

Run `composer update` in your terminal to pull down the package into your vendors folder.

Add the service provider to the `providers` array in Laravel's `./config/app.php` file:

```
1 TomLingham\Searchy\SearchyServiceProvider::class
```

Add the Alias to the `aliases` array in Laravel's `./config/app.php` file if you want to have quick access to it in your application:

```
1 'Searchy' => TomLingham\Searchy\Facades\Searchy::class
```

Usage

To use Searchy, you can take advantage of magic methods.

If you are searching the name and email column/field of users in a `users` table you would, for example run:

```
1 $users = Searchy::users('name', 'email')->query('John Smith')->get();
```

You can also write this as:

```
1 $users = Searchy::search('users')->fields('name', 'email')->query('John Smith')->get();
```

In this case, pass the columns you want to search through to the `fields()` method.

These examples both return an array of Objects containing your search results. You can use `getQuery()` instead of `get()` to return an instance of the Database Query Object in case you want to do further manipulation to the results:

```
1 $users = Searchy::search('users')->fields('name', 'email')->query('John Smith')
2     ->getQuery()->having('relevance', '>', 20)->get();
```

Limit on results returned

To limit your results, you can use Laravel's built in DatabaseQuery Builder method and chain further methods to narrow your results.

```
1 // Only get the top 10 results
2 $users = Searchy::search('users')->fields('name', 'email')->query('John Smith')
3     ->getQuery()->limit(10)->get();
```

Searching multiple Columns

You can also add multiple arguments to the list of fields/columns to search by.

For example, if you want to search the name, email address and username of a user, you might run:

```
1 $users = Searchy::users('name', 'email', 'username')->query('John Smith')->get();
```

If you need to build your table list dynamically, you can also pass an array of fields instead as the first argument (All other following arguments will be ignored):

```
1 $users = Searchy::users(['name', 'email', 'username'])->query('John Smith')->get();
```

Searching Joined/Concatenated Columns

Sometimes you may want to leverage searches on concatenated column. For example, on a `first_name` and `last_name` field but you only want to run the one query. To do this can separate columns with a double colon:

```
1 $users = Searchy::users('first_name::last_name')->query('John Smith')->get();
```

Soft Deleted Records

By default soft deletes will not be included in your results. However, if you wish to include soft deleted records you can do so by adding the `withTrashed()` after specifying your table and fields;

```
1 Searchy::users('name')->withTrashed()->query('Batman')->get();
```

Return only specific columns

You can specify which columns to return in your search:

```
1 $users = Searchy::users('first_name::last_name')->query('John Smith')->select('first_name')->get();
2
3 // Or you can swap those around...
4 $users = Searchy::users('first_name::last_name')->select('first_name')->query('John Smith')->get();
```

This will, however, also return the `relevance` aliased column regardless of what is entered here.

How to get a Laravel Eloquent Collection

Transforming the search results into a collection of Laravel Eloquent models is outside the scope of this project. However, an easy way to achieve this without hitting your database more than necessary is to use the Eloquent `hydrate()` method.

```
1 \App\User::hydrate(Searchy::users('name', 'email')->query('Andrew')->get());
```

This method creates a collection of models from a plain arrays. This is just our case because Searchy results are provided as arrays, and using `hydrate` we will converted them to instances of `User` model.

Unicode Characters Support

If you are having issues with the returned results because you have unicode characters in your search data, you can use the `FuzzySearchUnicodeDriver`.

PLEASE NOTE: There is no sanitization of strings passed through to the `FuzzySearchUnicodeDriver` prior to inserting into raw MySQL statements. You will have to sanitize the string yourself first or risk opening up your application to SQL injection attacks. You have been warned.

To use, first follow the instructions to publish your configuration file (`php artisan vendor:publish`) and change your default driver from `fuzzy` to `ufuzzy`.

```
1
2 return [
3
4     'default' => 'ufuzzy',
5
6     ...
7 ]
```

Configuration

You can publish the configuration file to your `app` directory and override the settings by running `php artisan vendor:publish` to copy the configuration to your config folder as `searchy.php`

You can set the default driver to use for searches in the configuration file. Your options (at this stage) are: `fuzzy`, `simple` and `levenshtein`.

You can also override these methods using the following syntax when running a search:

```
1 Searchy::driver('fuzzy')->users('name')->query('Batman')->get();
```

Drivers

Searchy takes advantage of 'Drivers' to handle matching various conditions of the fields you specify.

Drivers are simply a specified group of 'Matchers' which match strings based on specific conditions.

Currently there are only three drivers: Simple, Fuzzy and Levenshtein (Experimental).

Simple Search Driver

The Simple search driver only uses 3 matchers each with the relevant multipliers that best suited my testing environments.

```
1 protected $matchers = [  
2     'TomLingham\Searchy\Matchers\ExactMatcher'           => 100,  
3     'TomLingham\Searchy\Matchers\StartOfStringMatcher'    => 50,  
4     'TomLingham\Searchy\Matchers\InStringMatcher'         => 30,  
5 ];
```

Fuzzy Search Driver

The Fuzzy Search Driver is simply another group of matchers setup as follows. The multipliers are what I have used, but feel free to change these or roll your own driver with the same matchers and change the multipliers to suit.

```
1 protected $matchers = [  
2     'TomLingham\Searchy\Matchers\ExactMatcher'           => 100,  
3     'TomLingham\Searchy\Matchers\StartOfStringMatcher'    => 50,  
4     'TomLingham\Searchy\Matchers\AcronymMatcher'          => 42,  
5     'TomLingham\Searchy\Matchers\ConsecutiveCharactersMatcher' => 40,  
6     'TomLingham\Searchy\Matchers\StartOfWordsMatcher'     => 35,  
7     'TomLingham\Searchy\Matchers\StudlyCaseMatcher'       => 32,  
8     'TomLingham\Searchy\Matchers\InStringMatcher'         => 30,  
9     'TomLingham\Searchy\Matchers\TimesInStringMatcher'    => 8,  
10 ];
```

Levenshtein Search Driver (Experimental)

The Levenshtein Search Driver uses the Levenshtein Distance to calculate the 'distance' between strings. It requires that you have a stored procedure in MySQL similar to the following [levenshtein](#) (*string1*, *string2*). There is an SQL file with a suitable function in the [res](#) folder - feel free to use this one.

```
1 protected $matchers = [  
2     'TomLingham\Searchy\Matchers\LevenshteinMatcher' => 100  
3 ];
```

Matchers

ExactMatcher

Matches an exact string and applies a high multiplier to bring any exact matches to the top.

StartOfStringMatcher

Matches Strings that begin with the search string. For example, a search for 'hel' would match; 'Hello World' or 'helping hand'

AcronymMatcher

Matches strings for Acronym 'like' matches but does NOT return Studly Case Matches For example, a search for 'fb' would match; 'foo bar' or 'Fred Brown' but not 'FreeBeer'.

ConsecutiveCharactersMatcher

Matches strings that include all the characters in the search relatively positioned within the string. It also calculates the percentage of characters in the string that are matched and applies the multiplier accordingly.

For Example, a search for 'fba' would match; 'Foo Bar' or 'Afraid of bats', but not 'fabulous'

StartOfWordsMatcher

Matches the start of each word against each word in a search.

For example, a search for 'jo ta' would match; 'John Taylor' or 'Joshua B. Takeshi'

StudlyCaseMatcher

Matches Studly Case strings using the first letters of the words only

For example a search for 'hp' would match; 'HtmlServiceProvider' or 'HashParser' but not 'hasProvider'

InStringMatcher

Matches against any occurrences of a string within a string and is case-insensitive.

For example, a search for 'smi' would match; 'John Smith' or 'Smiley Face'

TimesInStringMatcher

Matches a string based on how many times the search string appears inside the string it then applies the multiplier for each occurrence. For example, a search for 'tha' would match; 'I hope that that cat has caught that mouse' (3 x multiplier) or 'Thanks, it was great!' (1 x multiplier)

LevenshteinMatcher

See *Levenshtein Driver*

Extending

Drivers

It's really easy to roll your own search drivers. Simply create a class that extends [TomLingham\Searchy\SearchDrivers\BaseSearchDriver](#) and add a property called `$matchers` with an array of matcher classes as the key and the multiplier for each matcher as the values. You can pick from the classes that are already included with Searchy or you can create your own.

Matchers

To create your own matchers, you can create your own class that extends [TomLingham\Searchy\Matchers\BaseMatcher](#) and (for simple Matchers) override the `formatQuery` method to return a string formatted with % wildcards in required locations. For more advanced extensions you may need to override the `buildQuery` method and others as well.

Contributing & Reporting Bugs

If you would like to improve on the code that is here, feel free to submit a pull request.

If you find any bugs, submit them here and I will respond as soon as possible. Please make sure to include as much information as possible.