You signed in with another tab or window. Reload to refresh your session.

FriendsOfCake / **search**

Watch 19 Star 30 Fork 21

Code Issues 8 Pull requests 6 Wiki Pulse Graphs

No description or website provided.

139 commits **3** branches **5** releases **15** contributors

Branch: master New pull request New file Find file HTTPS https://github.com/Friend Download ZIP

dereuromark Merge pull request #78 from FriendsOfCake/josegonzalez-patch-1 … Latest commit 376cf04 on Mar 1

src	Fix typo in implementedMethods	a month ago
tests	Add support for ILIKE.	2 months ago
.editorconfig	Add tests	9 months ago
.gitattributes	Add tests	9 months ago
.gitignore	Add gitignore	9 months ago
.travis.yml	Bye bye coveralls, say hello to codecov.	2 months ago
LICENSE.txt	Add license	9 months ago
README.md	Bye bye coveralls, say hello to codecov.	2 months ago
composer.json	Add tests	9 months ago
phpunit.xml.dist	Bye bye coveralls, say hello to codecov.	2 months ago

README.md

CakePHP Search

build passing | coverage 75% | downloads 18k | license MIT

Search provides a search module for CakePHP applications.

Requirements

The master branch has the following requirements:

- CakePHP 3.0.0 or greater.

Installation

- Install the plugin with composer from your CakePHP Project's ROOT directory (where composer.json file is located)

```
php composer.phar require friendsofcake/search
```

- Load the plugin by adding following to your `config/bootstrap.php` or running command

```
./bin/cake plugin load Search
```

Usage

The plugin has three main parts which you will need to configure and include in your application.

Table class

There are three tasks during setup in your table class. Firstly you must add a `use` statement for the `Search\Manager`. Next you need to attach the `Search` behaviour to your table class. Then you have two options to work with the search filters:

The first way is the prefered way as it works the same as many core classes as well. In your table classes `initialize()` method call the `searchManager()` method, it will return a search manager instance. You can now add filters to the manager by chaining them. The first arg of the `add()` method is the field, the second the filter using the dot notation of cake to load filters from plugins. The third one is an array of filter specific options.

```
use Search\Manager;

class ExampleTable extends Table {

    public function initialize(array $config)
    {
        parent::initialize();
        // Add the behaviour to your table
        $this->addBehavior('Search.Search');

        $this->searchManager()
            ->add('author_id', 'Search.Value')
            // Here we will alias the 'q' query param to search the `Articles.title`
            // field and the `Articles.content` field, using a LIKE match, with `%`
            // both before and after.
            ->add('q', 'Search.Like', [
                'before' => true,
                'after' => true,
                'field' => [$this->aliasField('title'), $this->aliasField('content')]
            ])
            ->add('foo', 'Search.Callback', [
                'callback' => function ($query, $args, $manager) {
                    // Modify $query as required
                }
            ]);
    }
```

The old way is to add a `searchConfiguration()` method to the class. The behavior will look if such a method exists and if yes use it to get the search manager instance from it. This method **must** return a search manager instance.

If you want to change the name of the method, or have multiple methods and switch between them, you can configure the name of the method by setting the behaviors option `searchConfigMethod` to the name of the method you want.

```
use Search\Manager;

class ExampleTable extends Table {

    public function initialize(array $config)
```

You signed in with another tab or window. Reload to refresh your session.

```
    }

    // Configure how you want the search plugin to work with this table class
    public function searchConfiguration()
    {
        $search = new Manager($this)
            ->value('author_id', [
                'field' => $this->aliasField('author_id')
            ])
            // Here we will alias the 'q' query param to search the `Articles.title`
            // field and the `Articles.content` field, using a LIKE match, with `%`
            // both before and after.
            ->like('q', [
                'before' => true,
                'after' => true,
                'field' => [$this->aliasField('title'), $this->aliasField('content')]
            ])
            ->callback('foo', [
                'callback' => function ($query, $args, $manager) {
                    // Modify $query as required
                }
            ]);

        return $search;
    }
```

Controller class

In order for the Search plugin to work it will need to process the query params which are passed in your url. So you will need to edit your `index` method to accomodate this.

```
public function index()
{
    $query = $this->Articles
        // Use the plugins 'search' custom finder and pass in the
        // processed query params
        ->find('search', $this->Articles->filterParams($this->request->query))
        // You can add extra things to the query if you need to
        ->contain(['Comments'])
        ->where(['title IS NOT' => null]);

    $this->set('articles', $this->paginate($query));
}
```

The `search` finder and the `filterParams()` method are dynamically provided by the `Search` behavior.

Component

Then add the Search Prg component to the necessary methods in your controller.

⚠️ Make sure,

- That you add this in the controller's `initialize()` method.
- That you only add the methods which are using search, such as your `index()` method.

```
public function initialize()
{
    parent::initialize();
    $this->loadComponent('Search.Prg', [
```

The `Search.Prg` component will allow your filtering forms to be populated using the data in the query params. It uses the Post, redirect, get pattern.

Filtering your data

Once you have completed all the setup you can now filter your data by passing query params in your index method. Using the `Article` example given above, you could filter your articles using the following.

```
example.com/articles?q=cakephp
```

Would filter your list of articles to any article with "cakephp" in the `title` or `content` field. You might choose to make a `get` form which posts the filter directly to the url, but if you're using the `Search.Prg` component, you'll want to use `POST` .

Creating your form

In most cases you'll want to add a form to your index view which will search your data.

```
    echo $this->Form->create();
    // You'll need to populate $authors in the template from your controller
    echo $this->Form->input('author_id');
    // Match the search param in your table configuration
    echo $this->Form->input('q');
    echo $this->Form->button('Filter', ['type' => 'submit']);
    echo $this->Html->link('Reset', ['action' => 'index']);
    echo $this->Form->end();
```

If you are using the `Search.Prg` component the forms current values will be populated from the query params.

Filters

The Search plugin comes with a set of predefined search filters that allow you to easily create the search results you need. Use:

- `value` to limit results to exact matches
- `like` to produce results containing the search query (`LIKE` or `ILIKE`)
- `finder` to produce results using a (custom) finder
- `compare` to produce results requiring operator comparison (`>` , `<` , `>=` and `<=`)
- `callback` to produce results using your own custom callable function

Optional fields

Sometimes you might want to search your data based on two of three inputs in your form. You can use the `filterEmpty` search option to ignore any empty fields.

```
// ExampleTable.php
// Inside your searchConfiguration() method
    $search->value('author_id', [
        'filterEmpty' => true
    ]);
```

Be sure to allow empty in your search form, if you're using one.

⚠ You signed in with another tab or window. Reload to refresh your session.