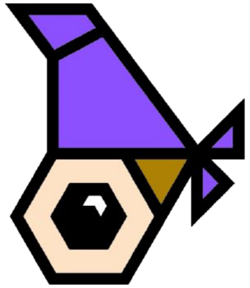
About me

Name:	Alexander Schranz
Workplace:	Sulu (sulu.io)
Tools:	PHP, Symfony, Twig, Elasticsearch, Redis ReactJS, MobX, React Native (Expo)
Experience:	Web Developer since 2012 Certified Symfony 5 Expert
OSS:	Symfony Redis Messenger (4.3) Broke Session Starts inside ESI (5.4) Symfony Stream JSON Response (6.3) Lock based Semaphore (maybe 7.3)



@alexander-schranz, @alex_s_

SEAL

Dive into the sea of search engines

What is SEAL?

SEAL stands for **Search Engine Abstraction Layer**

What flysystem is for file storage access and doctrine/dbal for databases, is SEAL for search engines / services.

SEAL provides a single interface, to communicate with different engines / services.

SEAL itself is framework agnostic, but provides integrations to your favorite frameworks.

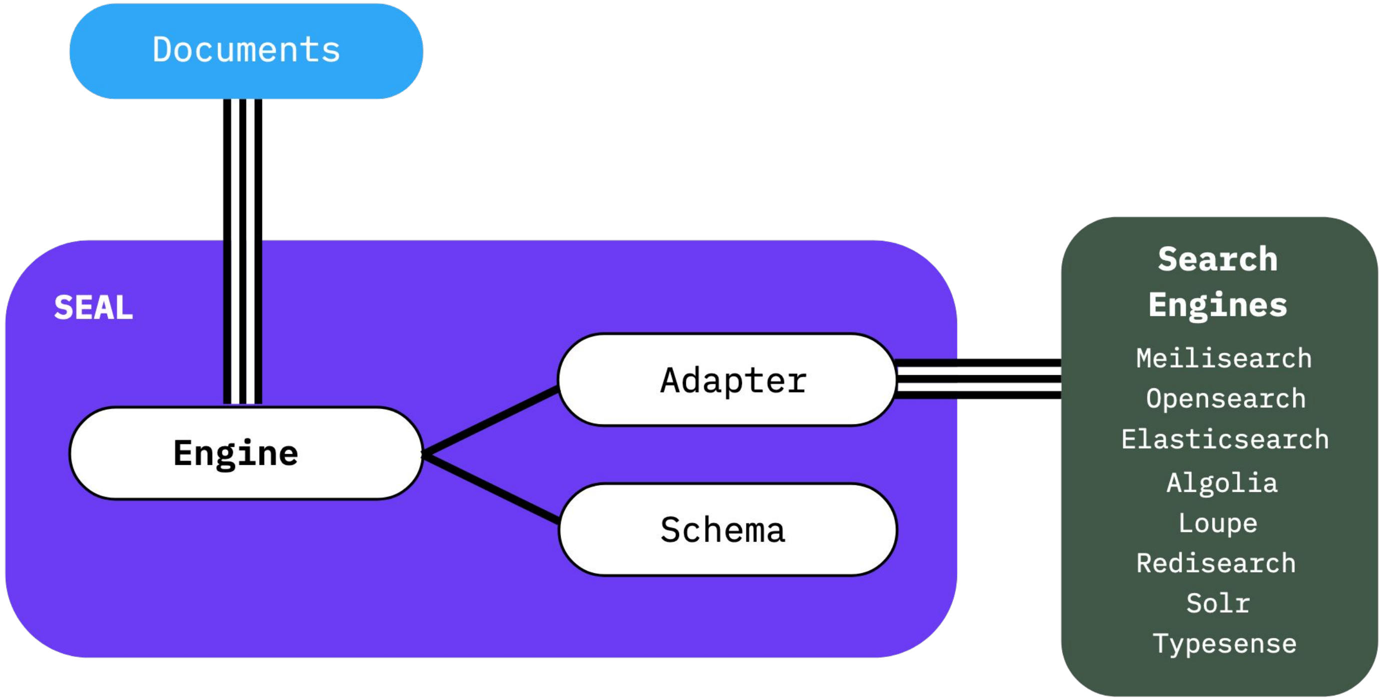
Which Search services?

- Elasticsearch
- Opensearch
- Meilisearch
- Algolia
- RediSearch
- Solr
- Typesense
- Loupe

Which Frameworks?

- Symfony
- Laravel
- Spiral
- Mezzio
- Yii

Quick Overview



Create Engine Instance (Standalone)

```php
<?php

use Meilisearch\Client;
use CmsIg\Seal\Adapter\Meilisearch\MeilisearchAdapter;
use CmsIg\Seal\Engine;

$client = new Client('http://127.0.0.1:7700');

$engine = new Engine(
    new MeilisearchAdapter($client),
    $schema,
);
```

Create Engine Instance (Framework integration)

```yaml
cmsig_seal:
    schemas:
        default:
            dir: '%kernel.project_dir%/config/schemas'
    engines:
        default:
            adapter: '%env(SEAL_DSN)%'
```

```
SEAL_DSN=algolia://%env(ALGOLIA_APPLICATION_ID)%:%env(ALGOLIA_ADMIN_API_KEY)%
SEAL_DSN=elasticsearch://127.0.0.1:9200
SEAL_DSN=meilisearch://127.0.0.1:7700
SEAL_DSN=opensearch://127.0.0.1:9200
SEAL_DSN=redis://supersecure@127.0.0.1:6379
SEAL_DSN=solr://127.0.0.1:8983
SEAL_DSN=typesense://S3CR3T@127.0.0.1:8108
SEAL_DSN=loupe://var/indexes/
SEAL_DSN=memory://
```

Defining the Schema

```php
new Schema(
    new Index(name: 'news', [
        'id' => new Field\IdentifierField('id'),
        'title' => new Field\TextField('title'),
        'description' => new Field\TextField('description'),
        'rating' => new Field\FloatField('rating'),
        'publishedAt' => new Field\DateTimeField('publishedAt'),
        'isAdvertising' => new Field\BooleanField('isAdvertising'),
        'tags' => new Field\TextField('tags', multiple: true),
    ]),
);
```

Strict Schema & PHP Type orientated

Defining the Schema

```
new Field\TextField('title',
    searchable: true,
),

new Field\TextField('tags',
    filterable: true,
    searchable: true,
    multiple: true,
),

new Field\FloatField('rating',
    sortable: true,
),
```

Instead of telling how to store the data:

types + ~~storage attributes~~

Tell what we want to do with the stored data:

types + usages

No search engine jargons:

~~index: true, doc_values: true, keyword~~

Understandable configurations:

searchable, filterable, sortable

Create Index

```php
$engine->createIndex(index: 'news');
```

Drop Index

```php
$engine->dropIndex(index: 'news');
```

Exist Index

```php
$engine->existIndex(index: 'news');
```

Create Index

```
# Symfony
bin/console cmsig:seal:index-create --index=news
# Laravel
php artisan cmsig:seal:index-create --index=news
# Spiral
php app.php cmsig:seal:index-create --index=news
# Mezzio
vendor/bin/laminas cmsig:seal:index-create --index=news
# Yii
./yii cmsig:seal:index-create --index=news
```

Drop Index

```
# Symfony
bin/console cmsig:seal:index-drop --index=news
# Laravel
php artisan cmsig:seal:index-drop --index=news
# Spiral
php app.php cmsig:seal:index-drop --index=news
# Mezzio
vendor/bin/laminas cmsig:seal:index-drop --index=news
# Yii
./yii cmsig:seal:index-drop --index=news
```

Add or Update Document

```php
$document = [
    'id' => 1,
    'title' => 'SEAL',
    'description' => 'Dive into the sea of search engines ',
];

$engine->saveDocument(index: 'news', document: $document);
```

Get Document

```php
$document = $engine->getDocument(index: 'news', identifier: 1);
```

Remove Document

```php
$engine->deleteDocument(index: 'news', identifier: 1);
```

Search Documents

```php
use CmsIg\Seal\Search\Condition;

$engine→createSearchBuilder('blog')
    →addFilter(new Condition\SearchCondition('first'))
    →getResult();

echo 'Found documents: ' . $result→total() . PHP_EOL;

foreach ($result as $document) {
    echo $document['title'] . PHP_EOL;
}
```

Fluent search builder interface.

Filter Documents

```php
use CmsIg\Seal\Search\Condition;

$engine->createSearchBuilder('blog')
    ->addFilter(new Condition\EqualCondition('tag', 'Tag A'))
    ->getResult();

$engine->createSearchBuilder('blog')
    ->addFilter(new Condition\SearchCondition('My Search ...'))
    ->addFilter(new Condition\EqualCondition('tag', 'Tag A'))
    ->getResult();
```

More Filters

```php
use CmsIg\Seal\Search\Condition;

// id == '1'
new Condition\IdentifierCondition('1')
// tag == 'Tag A'
new Condition\EqualCondition('tag', 'Tag A')
// tag ≠ 'Tag A'
new Condition\NotEqualCondition('tag', 'Tag A')
// rating > 2.5
new Condition\GreaterThanCondition('rating', 2.5)
// rating ⩾ 2.5
new Condition\GreaterThanEqualCondition('rating', 2.5)
// rating < 2.5
new Condition\LowerThanCondition('rating', 2.5)
// rating ⩽ 2.5
new Condition\LowerThanEqualCondition('rating', 2.5)
```

Nested Filters

```php
use CmsIg\Seal\Search\Condition;

$result = $this->engine->createSearchBuilder('blog')
    ->addFilter(new Condition\AndCondition(
        new Condition\EqualCondition('tags', 'Tech'),
        new Condition\OrCondition(
            new Condition\EqualCondition('tags', 'UX'),
            new Condition\EqualCondition('isSpecial', true),
        ),
    ))
    ->getResult();
```

Geo Filters

```php
<?php

use CmsIg\Seal\Search\Condition;

$result = $this→engine→createSearchBuilder('restaurants')
    →addFilter(new Condition\GeoDistanceCondition('location',
        45.472735, 9.184019, 2000.
    ))
    →getResult();

$result = $this→engine→createSearchBuilder('restaurants')
    →addFilter(new Condition\GeoBoundingBoxCondition('location',
        45.494181, 9.214024, 45.449484, 9.179175,
    ))
    →getResult();
```

The Engine Interface

```php
interface EngineInterface
{
    public function saveDocument(string $index, array $document): void;
    public function deleteDocument(string $index, string $identifier): void;
    public function getDocument(string $index, string $identifier): array;
    public function dropIndex(string $index): void;
    public function createIndex(string $index): void;
    public function existIndex(string $index): bool;
    public function createSearchBuilder(string $index): SearchBuilder;
}
```

Reindex Providers

```php
use Schranz\Search\SEAL\Reindex\ReindexProviderInterface;

class NewsReindexProvider implements ReindexProviderInterface
{
    public function total(): ?int
    {
        return 2;
    }

    public function provide(): \Generator
    {
        yield [
            'id' ⇒ 1,
            'title' ⇒ 'Title 1',
        ];

        yield [
            'id' ⇒ 2,
            'title' ⇒ 'Title 2',
        ];
    }

    public static function getIndex(): string
    {
        return 'news';
    }
}
```

```
total(): ?int

provide(): \Generator

getIndex(): string
```

Reindexing

```
# Symfony
bin/console schranz:search:reindex --index=news --drop
# Laravel
php artisan schranz:search:reindex --index=news --drop
# Spiral
php app.php schranz:search:reindex --index=news --drop
# Mezzio
vendor/bin/laminas schranz:search:reindex --index=news --drop
# Yii
./yii schranz:search:reindex --index=blog --drop
```

```
$engine->reindex($reindexProviders, 'news', dropIndex: true);
```

Bulk actions

```
$engine→bulk(
    index: 'blog',
    saveDocuments: [
        ['id' ⇒ 1, 'title' ⇒ '...', 'description' ⇒ '...'],
        ['id' ⇒ 2, 'title' ⇒ '...', 'description' ⇒ '...'],
        ['id' ⇒ 3, 'title' ⇒ '...', 'description' ⇒ '...'],
    ],
    deleteDocumentIdentifiers: [4, 5],
    bulkSize: 100,
);
```

Bulk actions support any iterable for *saveDocuments* and *deleteDocumentIdentifier* recommended is usage of Generators.

The Engine Interface

```php
interface EngineInterface
{
    public function bulk(
        string $index,
        iterable $saveDocuments,
        iterable $deleteDocumentIdentifiers,
        int $bulkSize = 100
    ): void;

    public function reindex(
        iterable $reindexProviders,
        ReindexConfig $reindexConfig,
        callable|null $progressCallback = null,
    ): void;

    public function saveDocument(string $index, array $document): void;
    public function deleteDocument(string $index, string $identifier): void;
    public function getDocument(string $index, string $identifier): array;
    public function dropIndex(string $index): void;
    public function createIndex(string $index): void;
    public function existIndex(string $index): bool;
    public function createSearchBuilder(string $index): SearchBuilder;
}
```

Complex Objects Possible

```php
<?php

use CmsIg\Seal\Schema\Field;
use CmsIg\Seal\Schema\Index;

$index = new Index('blog', [
    'uuid' => new Field\IdentifierField('uuid'),
    'title' => new Field\TextField('title'),
    'header' => new Field\TypedField('header', 'type', [
        'image' => [
            'media' => new Field\IntegerField('media'),
        ],
        'video' => [
            'media' => new Field\TextField('media', searchable: false),
        ],
    ]),
    'article' => new Field\TextField('article'),
    'blocks' => new Field\TypedField('blocks', 'type', [
        'text' => [
            'title' => new Field\TextField('title'),
            'description' => new Field\TextField('description'),
            'media' => new Field\IntegerField('media', multiple: true),
        ],
        'embed' => [
            'title' => new Field\TextField('title'),
            'media' => new Field\TextField('media', searchable: false),
        ],
    ], multiple: true),
    'footer' => new Field\ObjectField('footer', [
        'title' => new Field\TextField('title'),
    ]),
    'created' => new Field\DateTimeField('created', filterable: true, sortable: true),
    'commentsCount' => new Field\IntegerField('commentsCount', filterable: true, sortable: true),
    'rating' => new Field\FloatField('rating', filterable: true, sortable: true),
    'comments' => new Field\ObjectField('comments', [
        'email' => new Field\TextField('email', searchable: false),
        'text' => new Field\TextField('text'),
    ], multiple: true),
    'tags' => new Field\TextField('tags', multiple: true, filterable: true),
    'categoryIds' => new Field\IntegerField('categoryIds', multiple: true, filterable: true),
]);
```

Complex objects possible.

But …

Best practices

The best practices are to keep your document also when it index complex model as simple as possible. This means that you concat data from different sources to one field. And create additional fields only for things which need to be searchable or filterable a special way.

```php
<?php

use CmsIg\Seal\Schema\Field;
use CmsIg\Seal\Schema\Index;

$index = new Index('blog', [
    'uuid' ⇒ new Field\IdentifierField('uuid'),
    'title' ⇒ new Field\TextField('title'),
    'description' ⇒ new Field\TextField('description'),
    'url' ⇒ new Field\TextField('url'),
    'image' ⇒ new Field\IntegerField('image'),
    'content' ⇒ new Field\TextField('content', multiple: true),
]);
```

Packages

cmsig/seal-elasticsearch-adapter

cmsig/seal-opensearch-adapter

cmsig/seal-meilisearch-adapter

cmsig/seal-algolia-adapter

cmsig/seal-redisearch-adapter

cmsig/seal-solr-adapter

cmsig/seal-typesense-adapter

cmsig/seal-loupe-adapter

cmsig/symfony-bundle

cmsig/laravel-package

cmsig/spiral-bridge

cmsig/mezzio-module

cmsig/yii-module

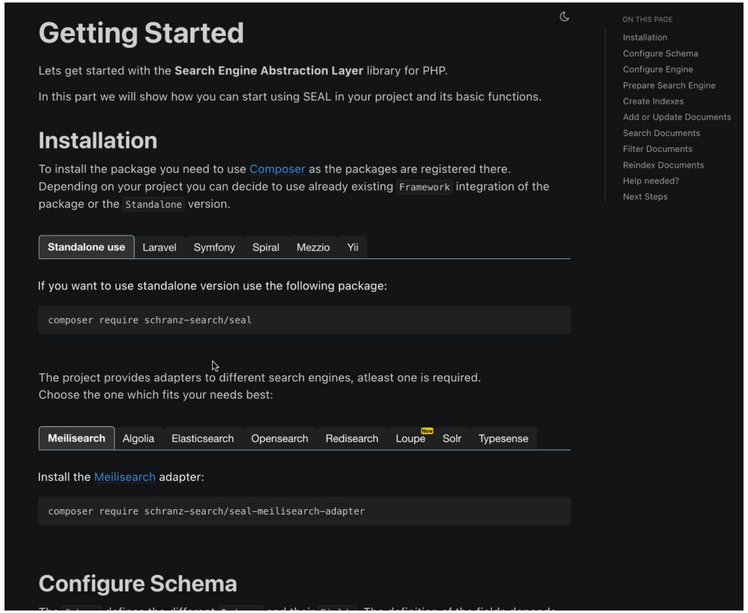
cmsig/seal

cmsig/seal-memory-adapter

cmsig/seal-read-write-adapter

cmsig/seal-multi-adapter

The Documentation



Getting Started

Lets get started with the **Search Engine Abstraction Layer** library for PHP.

In this part we will show how you can start using SEAL in your project and its basic functions.

Installation

To install the package you need to use Composer as the packages are registered there. Depending on your project you can decide to use already existing `Framework` integration of the package or the `Standalone` version.

If you want to use standalone version use the following package:

```
composer require schranz-search/seal
```

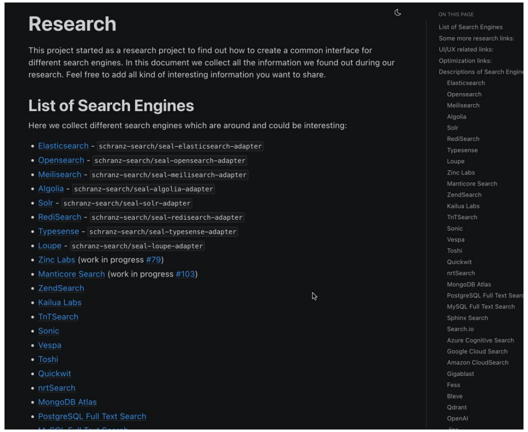
The project provides adapters to different search engines, atleast one is required. Choose the one which fits your needs best:

Install the Meilisearch adapter:

```
composer require schranz-search/seal-meilisearch-adapter
```

Configure Schema

https://php-cmsig.github.io/search/



Research

This project started as a research project to find out how to create a common interface for different search engines. In this document we collect all the information we found out during our research. Feel free to add all kind of interesting information you want to share.

List of Search Engines

Here we collect different search engines which are around and could be interesting:

- Elasticsearch - `schranz-search/seal-elasticsearch-adapter`
- Opensearch - `schranz-search/seal-opensearch-adapter`
- Meilisearch - `schranz-search/seal-meilisearch-adapter`
- Algolia - `schranz-search/seal-algolia-adapter`
- Solr - `schranz-search/seal-solr-adapter`
- RediSearch - `schranz-search/seal-redisearch-adapter`
- Typesense - `schranz-search/seal-typesense-adapter`
- Loupe - `schranz-search/seal-loupe-adapter`
- Zinc Labs (work in progress #79)
- Manticore Search (work in progress #103)
- ZendSearch
- Kailua Labs
- TnTSearch
- Sonic
- Vespa
- Toshi
- Quickwit
- nrtSearch
- MongoDB Atlas
- PostgreSQL Full Text Search

Milestones

2022. Dec	Research Project (schranz-search)
2023. May	First Release 0.1 — 7 Search Engines / 5 Frameworks
2023. Sep	**Loupe Support** with 0.2
2024. Jan	PHP 8.3 and Symfony 7 with 0.3
2024. Mar	Laravel 11 Support with 0.4
2024. Sep	**GeoDistance** and **GeoBoundingBox** 0.5
2024. Dec	**Bulk** support and move repository to **PHP-CMSIG**

Roadmap / What is coming next?

- 💡 Highlighting
- ☀️ Summarize / Matched context snippet
- 🏗️ Laravel 12 Support

Any ideas?

Join the community on Github:
https://github.com/php-CMSIG/search/discussions

- 💈 Faceting / Aggregations
- ⏰ Zero Downtime Reindexing
- ↔️ StartWith and EndWith Conditions
- 🌿 MongoDB Atlas Search Adapter
- 🏢 Static Factory for Conditions

What about ODM? – Classes instead of arrays

The SEAL package will provide the fundamentals to communicate with different search engines, like doctrine/dbal.

The ODM implementation will so be its own package like doctrine/orm.

An ODM package is planned after SEAL is stabilized.

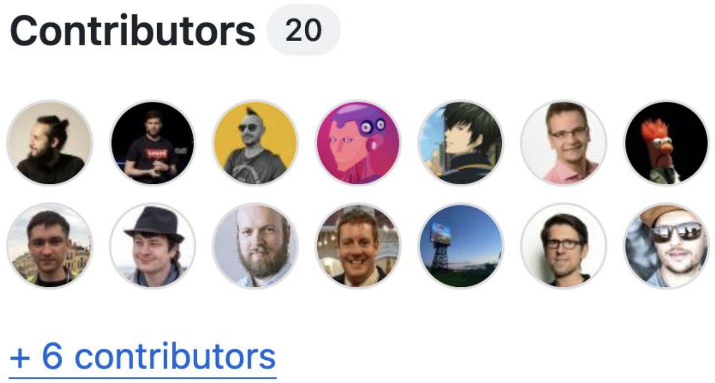
Workaround currently use `symfony/serializer` normalizers.

Maybe future implementation of a ODM for SEAL:

```php
#[ODM\Index('news')]
class News {
    #[ODM\Identifier()]
    private string $id;

    #[ODM\Text(searchable: true)]
    private string $title;
}
```

Thx to all contributors
and people who did already give SEAL a try!



Contributors 20

+ 6 contributors

Why SEAL?

- No Vendor lock-in
- Framework agnostic with integration into different Frameworks
- Easy to tryout other Search Engines or change for different usages
- Strict Schema
- Backed by the PHP CMS-IG (Contao, Sulu, Typo3)
- No rewrites for basics (reindex, add, update, remove documents)

Hint:

Use what you need, for special queries you still can use the client of your search engine. Add, Remove, Reindex can do SEAL, So if search engine need to be changed only your special query need be migrated.

Time for your questions

https://github.com/php-cmsig/search

Github:	@alexander-schranz
Bluesky:	@alexander-schranz.bsky.social
X / Twitter:	@alex_s_
Mastodon:	alex_s@mastodon.social