Implementing Full-Text Search with Laravel Scout and Algolia

Here's a comprehensive guide to setting up powerful full-text search in your Laravel application using Laravel Scout with Algolia.

1. Installation & Setup

Install required packages:

```
composer require laravel/scout
composer require algolia/algoliasearch-client-php
```

Publish Scout configuration:

```
php artisan vendor:publish --
provider="Laravel\Scout\ScoutServiceProvider"
```

Configure environment variables (.env):

```
SCOUT_DRIVER=algolia
ALGOLIA_APP_ID=your_app_id
ALGOLIA_SECRET=your_admin_api_key
ALGOLIA_SEARCH=your_search_only_api_key
```

PROF

2. Model Configuration

Prepare your model for indexing:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Laravel\Scout\Searchable;

class Post extends Model
{
    use Searchable;

    /**
```

```
     * Get the indexable data array for the model.
     */
    public function toSearchableArray()
    {
        $array = $this->toArray();

        // Customize the data array...
        $array['author_name'] = $this->author->name;
        $array['category_name'] = $this->category->name;

        return $array;
    }

    /**
     * Determine if the model should be searchable.
     */
    public function shouldBeSearchable()
    {
        return $this->is_published;
    }
}
```

3. Algolia Configuration

Create a custom index configuration:

```
// In your model
public function searchableAs()
{
    return 'posts_index';
}
```

PROF

Configure index settings:

```
// Typically in a service provider
use Algolia\AlgoliaSearch\SearchClient;
use Laravel\Scout\EngineManager;

public function boot()
{
    resolve(EngineManager::class)->extend('algolia', function () {
        $client = SearchClient::create(
            config('scout.algolia.id'),
            config('scout.algolia.secret')
        );

        // Custom index settings
        $index = $client->initIndex('posts_index');
```

```
            $index->setSettings([
                'searchableAttributes' => [
                    'title',
                    'content',
                    'author_name',
                    'category_name'
                ],
                'customRanking' => [
                    'desc(published_at)',
                    'desc(views_count)'
                ],
                'attributesForFaceting' => [
                    'category_name',
                    'author_name'
                ],
                'replicas' => [
                    'posts_index_price_asc',
                    'posts_index_price_desc'
                ],
            ]);

            return new AlgoliaEngine($client);
        });
    }
```

4. Indexing Data

Import existing records:

```
php artisan scout:import "App\Models\Post"
```

Queueing imports (for large datasets):

PROF

```
php artisan queue:work
php artisan scout:import "App\Models\Post" --queue
```

Controlling chunk size:

```
php artisan scout:import "App\Models\Post" --chunk=100
```

5. Searching Implementation

Basic search in controller:

```
public function search(Request $request)
{
    return Post::search($request->input('query'))->get();
}
```

Advanced search with filters:

```
$results = Post::search($query)
    ->where('category_id', $categoryId)
    ->where('status', 'published')
    ->orderBy('published_at', 'desc')
    ->paginate(15);
```

Faceted search:

```
$results = Post::search($query)
    ->with([
        'filters' => 'category_name:Technology OR
category_name:Science',
        'facets' => ['category_name', 'author_name']
    ])
    ->paginate(15);
```

6. Frontend Integration

Install Algolia JavaScript client:

PROF

```
npm install algoliasearch instantsearch.js
```

Create search component:

```
import algoliasearch from 'algoliasearch/lite';
import instantsearch from 'instantsearch.js';

const searchClient = algoliasearch(
  process.env.MIX_ALGOLIA_APP_ID,
  process.env.MIX_ALGOLIA_SEARCH
);

const search = instantsearch({
  indexName: 'posts_index',
  searchClient,
});

search.addWidgets([
  instantsearch.widgets.searchBox({
    container: '#searchbox',
  }),

  instantsearch.widgets.hits({
    container: '#hits',
    templates: {
      item: `
        <div>
          <h3>{{#helpers.highlight}}{ "attribute": "title" }
{{/helpers.highlight}}</h3>
          <p>{{#helpers.highlight}}{ "attribute": "content" }
{{/helpers.highlight}}</p>
          <p>By: {{author_name}}</p>
        </div>
      `,
    },
  }),

  instantsearch.widgets.refinementList({
    container: '#category-list',
    attribute: 'category_name',
  }),
]);

search.start();
```

7. Real-Time Syncing

Handle model events:

PROF

```
// In your model
protected static function boot()
{
    parent::boot();

    static::created(function ($model) {
        $model->searchable();
    });

    static::updated(function ($model) {
        $model->searchable();
    });

    static::deleted(function ($model) {
        $model->unsearchable();
    });
}
```

Queue sync operations:

```
// In config/scout.php
'queue' => [
    'connection' => 'redis',
    'queue' => 'scout',
],
```

8. Performance Optimization

Selective indexing:

```
public function toSearchableArray()
{
    // Only index necessary fields
    return [
        'title' => $this->title,
        'content' => $this->content,
        'author' => $this->author->name,
        'published_at' => $this->published_at->timestamp,
    ];
}
```

Conditional indexing:

PROF

```
public function shouldBeSearchable()
{
    return $this->is_published && $this->approved;
}
```

9. Advanced Features

Synonyms:

```
$index->setSettings([
    'synonyms' => [
        ['objectID' => 'tech', 'type' => 'synonym', 'synonyms' =>
['tech', 'technology']],
        ['objectID' => 'mobile', 'type' => 'synonym', 'synonyms' =>
['mobile', 'cellphone', 'smartphone']]
```

```
    ]
]);
```

Typo tolerance:

```
$index->setSettings([
    'typoTolerance' => [
        'minWordSizeForTypos' => [
            'oneTypo' => 4,
            'twoTypos' => 8
        ]
    ]
]);
```

Geo-search:

```
// Add to your model
public function toSearchableArray()
{
    return [
        // ... other fields
        '_geoloc' => [
            'lat' => $this->latitude,
            'lng' => $this->longitude,
        ],
    ];
}

// Search usage
$results = Post::search($query)
    ->aroundLatLng($lat, $lng)
    ->within('5km')
    ->get();
```

PROF

10. Monitoring & Maintenance

Set up webhook for monitoring:

```
// In a service provider
Algolia::setSettings('posts_index', [
    'forwardToReplicas' => true,
    'attributesForFaceting' => ['filterOnly(category)'],
    'webhook' => [
        'url' => route('algolia.webhook'),
        'events' => ['settingsUpdate']
    ]
]);
```

Handle Algolia webhooks:

```
Route::post('/algolia/webhook', function (Request $request) {
    Log::info('Algolia settings updated', $request->all());
    return response()->json(['status' => 'success']);
})->middleware('auth:api');
```

11. Docker Integration

Update your `docker-compose.yml`:

```
services:
  # ... existing services ...

  scout:
    build:
      context: .
      dockerfile: docker/php/Dockerfile
    container_name: laravel_scout
    command: php artisan scout:flush "App\Models\Post" && php artisan
scout:import "App\Models\Post"
    volumes:
      - ./src:/var/www
    depends_on:
      - mysql
      - redis
```

PROF

12. Testing

Mock Scout in tests:

```
// In your test
Post::shouldReceive('search')
    ->once()
    ->with('test query')
    ->andReturnSelf();

Post::shouldReceive('get')
    ->once()
    ->andReturn(collect([Post::factory()->make()]));
```

```
$response = $this->get('/search?query=test+query');
$response->assertStatus(200);
```

Integration test:

```
public function test_search_returns_results()
{
    $post = Post::factory()->create(['title' => 'Test Post']);

    // Manually sync with Algolia
    $post->searchable();

    // Wait for indexing to complete
    sleep(1);

    $response = $this->get('/search?query=Test');

    $response->assertStatus(200)
        ->assertSee('Test Post');
}
```

This implementation provides a complete full-text search solution with Laravel Scout and Algolia, covering installation, configuration, advanced features, and maintenance. The system will provide fast, relevant search results with features like typo-tolerance, faceting, and geo-search.