Integrating GraphQL with Laravel using Lighthouse

Here's a comprehensive guide to setting up GraphQL in your Laravel application using the Lighthouse package, which provides a powerful and flexible GraphQL implementation.

1. Installation

Install required packages:

```
composer require nuwave/lighthouse
```

Publish the Lighthouse configuration:

```
php artisan vendor:publish --
provider="Nuwave\Lighthouse\LighthouseServiceProvider" --tag="config"
```

2. Configuration

Update your `.env` file:

```
LIGHTHOUSE_CACHE_ENABLE=false # Set to true in production
```

Configure `config/lighthouse.php` (optional):

PROF

```
return [
    'schema' => [
        'register' => base_path('graphql/schema.graphql'),
    ],
    // Other configuration options...
];
```

3. Set Up GraphQL Schema

Create a `graphql/schema.graphql` file in your project root:

```
type Query {
    users: [User!]! @all
    user(id: ID @eq): User @find
    posts: [Post!]! @all
```

```
    post(id: ID @eq): Post @find
}

type Mutation {
    createUser(name: String!, email: String! @rules(apply: ["email",
"unique:users"]), password: String! @hash): User @create
    updateUser(id: ID!, name: String, email: String): User @update
    deleteUser(id: ID!): User @delete
}

type User {
    id: ID!
    name: String!
    email: String!
    posts: [Post!]! @hasMany
    created_at: DateTime!
    updated_at: DateTime!
}

type Post {
    id: ID!
    title: String!
    content: String!
    author: User! @belongsTo
    created_at: DateTime!
    updated_at: DateTime!
}
```

4. Create Models and Migrations

User model (app/Models/User.php):

PROF

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'email',
        'password',
    ];
```

```
    protected $hidden = [
        'password',
        'remember_token',
    ];

    public function posts()
    {
        return $this->hasMany(Post::class);
    }
}
```

Post model (app/Models/Post.php):

```
php artisan make:model Post -m
```

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'title',
        'content',
        'user_id'
    ];

    public function author()
    {
        return $this->belongsTo(User::class, 'user_id');
    }
}
```

PROF

Migration for posts table:

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
```

```
            $table->text('content');
            $table->foreignId('user_id')->constrained();
            $table->timestamps();
        });
    }
```

Run migrations:

```
php artisan migrate
```

5. Set Up GraphQL Controller (Optional)

Create a controller to handle GraphQL requests:

```
php artisan make:controller GraphQLController
```

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class GraphQLController extends Controller
{
    public function __invoke(Request $request)
    {
        return response()->json(
            app('graphql')->executeQuery(
                $request->input('query'),
                null,
                [
                    'request' => $request,
                ],
                $request->input('variables')
            )->toArray()
        );
    }
}
```

Add route in `routes/api.php`:

```
use App\Http\Controllers\GraphQLController;

Route::post('/graphql', GraphQLController::class);
```

PROF

6. Testing GraphQL API

Using GraphQL Playground

Install GraphQL Playground:

```
composer require mll-lab/laravel-graphql-playground
```

Access it at `/graphql-playground` in your browser.

Example Queries:

Query all users with their posts:

```
query {
  users {
    id
    name
    email
    posts {
      id
      title
    }
  }
}
```

Create a new user:

```
mutation {
  createUser(
    name: "John Doe"
    email: "john@example.com"
    password: "secret"
  ) {
    id
    name
    email
  }
}
```

Update a user:

```
```

```
mutation {
  updateUser(
    id: 1
    name: "Updated Name"
    email: "updated@example.com"
  ) {
    id
    name
    email
  }
}
```

7. Advanced Features

Custom Resolvers

Create a custom resolver for complex queries:

```
php artisan make:graphql:query CustomUserQuery
```

Update the generated file:

```
<?php

namespace App\GraphQL\Queries;

use App\Models\User;

class CustomUserQuery
{
    public function __invoke($rootValue, array $args, $context,
$resolveInfo)
    {
        return User::where('name', 'LIKE', "%{$args['search']}%")
            ->with('posts')
            ->get();
    }
}
```

Add to your schema:

```
type Query {
    searchUsers(search: String!): [User!]! @field(resolver:
"App\\GraphQL\\Queries\\CustomUserQuery")
}
```

Authentication

Add authentication middleware in `config/lighthouse.php`:

```
'middleware' => [
    \Illuminate\Cookie\Middleware\EncryptCookies::class,
    \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
    \Illuminate\Session\Middleware\StartSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,

\Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::clas
s,
    'api',
],
```

Create a login mutation:

```
type Mutation {
    login(email: String!, password: String!): String!
}
```

Create a resolver:

```
php artisan make:graphql:mutation Login
```

```
<?php

namespace App\GraphQL\Mutations;

use Illuminate\Support\Facades\Auth;

class Login
{
    public function __invoke($rootValue, array $args, $context,
$resolveInfo)
    {
        if (Auth::attempt($args)) {
            $user = Auth::user();
            return $user->createToken('graphql')->plainTextToken;
        }

        throw new \Exception('Invalid credentials');
```

```
    }
}
```

Authorization

Use directives for authorization:

```
type Mutation {
    deletePost(id: ID!): Post @delete @can(ability: "delete", find:
"id")
}
```

Create a policy:

```
php artisan make:policy PostPolicy --model=Post
```

8. Performance Optimization

Enable Schema Caching (Production)

```
php artisan lighthouse:cache
```

Pagination

```
type Query {
    postsPaginated(page: Int = 1, count: Int = 15): [Post!]!
@paginate(type: "paginator" model: "App\\Models\\Post")
}
```

PROF

DataLoader for N+1 Problem

Lighthouse automatically batches queries to avoid N+1 problems.

9. Docker Integration

Update your `docker-compose.yml` to ensure GraphQL works:

```
services:
  app:
    # ... existing config ...
    ports:
```

```
      - "8000:8000"
      - "8001:8001" # For GraphQL subscriptions if needed
```

10. Testing with PHPUnit

Create a test:

```
php artisan make:test GraphQLTest
```

```php
<?php

namespace Tests\Feature;

use Tests\TestCase;
use App\Models\User;

class GraphQLTest extends TestCase
{
    public function testUserQuery()
    {
        $user = User::factory()->create();

        $response = $this->postJson('/graphql', [
            'query' => '{
                user(id: '.$user->id.') {
                    id
                    name
                    email
                }
            }'
        ]);

        $response->assertStatus(200)
            ->assertJson([
                'data' => [
                    'user' => [
                        'id' => (string) $user->id,
                        'name' => $user->name,
                        'email' => $user->email,
                    ]
                ]
            ]);
    }
}
```

This implementation provides a complete GraphQL API with Laravel using Lighthouse, including queries, mutations, authentication, authorization, and testing. You can extend it further based on your

PROF

application's specific requirements.