1v0 Ship v1.0 Faster

How to Build an XML Sitemap Generator in Laravel

How to Build an XML Sitemap Generator in Laravel

An XML sitemap tells search engines which pages to index and when they were last updated. Laravel makes it simple to **generate a sitemap dynamically** from your database. In this guide, we'll build a **Laravel sitemap dynamic generate** feature that covers posts, pages, and categories, outputs valid XML, supports toggling inclusion, and also shows how to export a `sitemap.xml` file and automatically regenerate it when new content is added.

Create the Sitemap Route and Controller

```
// routes/web.php
use App\Http\Controllers\SitemapController;

Route::get('/sitemap-preview.xml', [SitemapController::class,
'index']);Code language: PHP (php)
```

Here we expose the sitemap at `/sitemap-preview.xml`. This controller-based version is ideal for testing or development, but in production you will serve a static `sitemap.xml` file instead (explained below).

```
// app/Http/Controllers/SitemapController.php
namespace App\Http\Controllers;

use App\Models\Post;
use App\Models\Page;
use App\Models\Category;
```

1v0 Ship v1.0 Faster

```
class SitemapController extends Controller
{
    public function index()
    {
        $posts = Post::where('published', true)->get();
        $pages = Page::where('published', true)->get();
        $categories = Category::all();

        return response()
            ->view('sitemap.index', compact('posts', 'pages',
'categories'))
            ->header('Content-Type', 'application/xml');
    }
}Code language: PHP (php)
```

The controller collects resources and renders XML via a Blade view. This is the core of the *Laravel sitemap dynamic generate* flow, but for production Google indexing we'll rely on a static file.

Blade View for Sitemap XML

```
<!-- resources/views/sitemap/index.blade.php -->
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  @foreach($posts as $post)
    <url>
      <loc>{{ url('/posts/'.$post->slug) }}</loc>
      <lastmod>{{ $post->updated_at->toAtomString() }}</lastmod>
      <changefreq>weekly</changefreq>
      <priority>0.8</priority>
    </url>
  @endforeach
```

1v0 Ship v1.0 Faster

```
  @foreach($pages as $page)
    <url>
      <loc>{{ url('/'.$page->slug) }}</loc>
      <lastmod>{{ $page->updated_at->toAtomString() }}</lastmod>
      <changefreq>monthly</changefreq>
      <priority>0.6</priority>
    </url>
  @endforeach

  @foreach($categories as $category)
    <url>
      <loc>{{ url('/categories/'.$category->slug) }}</loc>
      <lastmod>{{ $category->updated_at->toAtomString() }}</lastmod>
      <changefreq>weekly</changefreq>
      <priority>0.5</priority>
    </url>
  @endforeach
</urlset>Code language: PHP (php)
```

The Blade view outputs valid XML with `loc`, `lastmod`, `changefreq`, and `priority` for each URL.

Export a Static `sitemap.xml` File

The best practice is to generate a static `public/sitemap.xml` file. It's served directly by your web server or CDN and is the file you'll submit to Google Search Console. Let's create an Artisan command that renders the XML and saves it:

```
// app/Console/Commands/GenerateSitemap.php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use App\Models\Post;
```

```
use App\Models\Page;
use App\Models\Category;
use Illuminate\Support\Facades\File;

class GenerateSitemap extends Command
{
    protected $signature = 'sitemap:generate';
    protected $description = 'Generate static sitemap.xml in public/';

    public function handle(): int
    {
        $posts = Post::where('published',
true)->where('include_in_sitemap', true)->get();
        $pages = Page::where('published', true)->get();
        $categories = Category::all();

        $xml = view('sitemap.index', compact('posts', 'pages',
'categories'))->render();

        File::put(public_path('sitemap.xml'), $xml);
        $this->info('sitemap.xml generated at public/sitemap.xml');

        return self::SUCCESS;
    }
}Code language: PHP (php)
```

This generates `public/sitemap.xml`, which is now a static file. **This is the file you should submit to Google Search Console**. The dynamic version (`/sitemap-preview.xml`) is helpful for debugging or previewing changes before exporting.

Auto-Regenerate on Content Changes

Whenever a post, page, or category changes, you can dispatch a job to regenerate the sitemap so the static file stays fresh.

```
// app/Jobs/RegenerateSitemap.php
namespace App\Jobs;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Artisan;

class RegenerateSitemap implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public function handle(): void
    {
        Artisan::call('sitemap:generate');
    }
}Code language: PHP (php)
```

Attach this job to model events like `saved` and `deleted` to ensure your sitemap is always current. For high traffic, debounce by adding a small delay.

Wrapping Up

You built a complete sitemap system in Laravel: dynamic preview via a controller, a static `sitemap.xml` exporter, and automatic regeneration with jobs and scheduling. The **Laravel**

sitemap dynamic generate preview is useful for testing, but the static `sitemap.xml` is what you should serve to crawlers and submit to Google Search Console.

What’s Next

Continue strengthening your Laravel SEO setup with these guides:

- Laravel SEO Guide: Optimizing Meta, Slugs, and Sitemaps
- How to Generate SEO-Friendly URLs and Slugs in Laravel
- Creating JSON-LD Structured Data in Laravel for SEO