How to Generate SEO-Friendly URLs and Slugs in Laravel

Clean and descriptive URLs are essential for SEO. Instead of numeric IDs like `/posts/123`, you should use slugs like `/posts/my-first-laravel-app`. In this article, we'll explore how to generate slugs automatically, prevent duplicates, allow manual editing in forms, and integrate slugs into routes and controllers.

Auto-Generating Slugs with `booted()`

Laravel's model events make it easy to generate slugs whenever a record is created. You can use `booted()` with the `creating` event.

```
// app/Models/Post.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Str;

class Post extends Model
{
    protected static function booted()
    {
        static::creating(function ($post) {
            $post->slug = Str::slug($post->title);
        });
    }
}Code language: PHP (php)
```

Whenever a new post is created, its title is converted to a slug. For example: "My First Laravel App" → `my-first-laravel-app`.

Performance Note on `booted()`

The `booted()` method is called **once per request**, registering model event listeners. The slug logic itself only runs on the `creating` event (when saving new records). Simply retrieving or initializing a model does not regenerate the slug — this keeps performance safe.

Preventing Duplicate Slugs

Duplicate slugs cause routing conflicts and SEO issues. For example, two posts titled “Hello World” would both generate `hello-world`. A better approach is to check existing slugs and append a number if needed.

```
// app/Models/Post.php
protected static function booted()
{
    static::creating(function ($post) {
        $base = Str::slug($post->title);

        // If slug is free, use it directly
        if (! static::where('slug', $base)->exists()) {
            $post->slug = $base;
            return;
        }

        // Otherwise compute next suffix
        $pattern = '^' . preg_quote($base, '/') . '(-[0-9]+)?$';

        $maxSuffix = static::whereRaw('slug REGEXP ?', [$pattern])
            ->selectRaw("
                MAX(
                    CASE
```

```
                        WHEN slug = ? THEN 0
                        ELSE CAST(SUBSTRING_INDEX(slug, '-', -1) AS
UNSIGNED)
                    END
                ) AS max_suffix
            ", [$base])
            ->value('max_suffix');

        $post->slug = $base . '-' . (((int) $maxSuffix) + 1);
    });
}Code language: PHP (php)
```

With this strategy, your slugs remain unique:

- `hello-world`
- `hello-world-1`
- `hello-world-2`

For extra safety, add a unique index on the `slug` column at the database level. This prevents race conditions under high concurrency.

```
$table->string('slug')->unique();Code language: PHP (php)
```

Using Slugs in Routes and Controllers

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

Route::get('/posts/{post:slug}', [PostController::class, 'show']);Code
language: PHP (php)
```

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

```
use App\Models\Post;

class PostController extends Controller
{
    public function show(Post $post)
    {
        return view('posts.show', compact('post'));
    }
}Code language: PHP (php)
```

By using `{post:slug}`, Laravel automatically looks up posts by their slug instead of their ID.

Allowing Manual Slug Editing in Forms

Sometimes you want editors to override the auto-generated slug. Add a `slug` field in your form.

```
<form action="{{ route('posts.store') }}" method="POST">
  @csrf
  <label>Title</label>
  <input type="text" name="title" id="title">

  <label>Slug (optional)</label>
  <input type="text" name="slug" id="slug">

  <button type="submit">Save</button>
</form>Code language: PHP (php)
```

In your model's `creating` event, only auto-generate the slug if it wasn't provided:

```
if (empty($post->slug)) {
    $post->slug = Str::slug($post->title);
```

```
}Code language: PHP (php)
```

UI Enhancement: Slug Preview with JavaScript

For better UX, you can show a live preview of the slug as the user types the title.

```
<script>
document.getElementById('title').addEventListener('input', function ()
{
    let slug = this.value.toLowerCase()
        .replace(/[^a-z0-9]+/g, '-')
        .replace(/(^-|-$)/g, '');
    document.getElementById('slug').value = slug;
});
</script>Code language: JavaScript (javascript)
```

This script converts the title into a slug format on the fly and updates the slug input field.

Best Practices for Slugs

- **Keep them short:** Ideally under 60 characters.
- **Use lowercase only:** Prevents duplicate variations.
- **Avoid stop words:** Remove words like “and,” “the,” “of” unless needed.
- **Hyphens not underscores:** Use `-` as Google prefers it over `_`.
- **Unique per resource:** Always enforce uniqueness in DB with a unique index.
- **Match content keywords:** Include relevant keywords for better SEO ranking.

Good vs Bad Slug Examples

Bad Slug	Good Slug	Why
`Post_123!!!`	`my-first-laravel-app`	Readable, keyword-rich, clean format.
`My First Laravel App`	`my-first-laravel-app`	Lowercase + hyphens instead of spaces.
`laravel---guide`	`laravel-guide`	No unnecessary dashes.
`the-best-guide-to-laravel-php-framework`	`laravel-guide`	Shorter and more focused on keywords.

This table shows how simple adjustments make slugs much cleaner for both SEO and users.

Wrapping Up

SEO-friendly slugs make your URLs readable, keyword-rich, and unique. You learned how to auto-generate slugs, prevent duplicates with an efficient strategy, allow manual overrides in forms, preview slugs in the UI, follow best practices, and identify good vs bad slug patterns. Combined with clean routes, this gives your Laravel app a strong SEO foundation.

What’s Next

Continue improving your Laravel SEO setup with these related guides:

- Laravel SEO Guide: Optimizing Meta, Slugs, and Sitemaps
- Adding Meta Tags and Open Graph Data Dynamically in Laravel
- How to Build an XML Sitemap Generator in Laravel