Panduan Pembuatan Sitemap XML di CodeIgniter 4

> Panduan ini menjelaskan cara membuat sitemap XML dinamis menggunakan CodeIgniter 4 dengan dukungan multibahasa (Indonesia & Inggris).

Daftar Isi

Gambaran Umum

Sitemap XML adalah file yang memberi tahu mesin pencari (Google, Bing, dll.) tentang halaman-halaman yang ada di website kita. Implementasi ini mencakup:

- **Homepage** untuk tiap bahasa (`/id` dan `/en`)
- **Halaman kategori** dalam dua bahasa
- **Halaman artikel** dengan struktur URL `/{lang}/{slug-kategori}/{slug-artikel}`

Struktur File

```
app/
├── Controllers/
│   └── Sitemap.php          ← Controller yang menangani request sitemap
├── Models/
│   └── ArtikelModel.php     ← Model yang mengumpulkan semua URL
└── Views/
    └── sitemap_xml.php      ← Template output XML
```

Model: ArtikelModel

```php
<?php
namespace App\Models;
use CodeIgniter\Model;
use App\Models\KategoriModel;
```

```php
class ArtikelModel extends Model
{
    protected $table = 'tb_artikel';
    protected $primaryKey = 'id_artikel';
    protected $allowedFields = [
        'id_user', 'id_kategori', 'judul_id', 'judul_en',
        'slug_id', 'slug_en', 'konten_id', 'konten_en',
        'thumbnail', 'tags_id', 'tags_en', 'views',
        'meta_title_id', 'meta_title_en',
        'meta_description_id', 'meta_description_en',
        'published_at', 'created_at', 'updated_at'
    ];
    protected $useTimestamps = false;
    // ...
}
```

Penjelasan Property Model

Property	Nilai	Penjelasan
`$table`	`tb_artikel`	Nama tabel di database
`$primaryKey`	`id_artikel`	Primary key tabel, berbeda dari default `id`
`$allowedFields`	*(daftar kolom)*	Kolom yang boleh diisi saat insert/update
`$useTimestamps`	`false`	Menonaktifkan auto-manage `created_at`/`updated_at` oleh CI4 karena kita kelola sendiri

Method: `getForSitemap()`

Method ini adalah inti dari sitemap. Ia mengumpulkan semua URL yang akan dimuat ke dalam sitemap, lalu mengembalikannya sebagai array.

```php
public function getForSitemap()
{
    $urls = [];
```

Variabel `$urls` adalah array kosong yang akan diisi satu per satu dengan setiap URL beserta tanggal modifikasinya (`lastmod`).

Bagian 1 — Homepage

```php
    // Homepage Indonesia
    $urls[] = [
        'loc'     => base_url('id'),
```

```
        'lastmod' => date('Y-m-d')
    ];

    // Homepage English
    $urls[] = [
        'loc'     => base_url('en'),
        'lastmod' => date('Y-m-d')
    ];
```

- `base_url('id')` → menghasilkan URL seperti `https://example.com/id`
- `base_url('en')` → menghasilkan URL seperti `https://example.com/en`
- `date('Y-m-d')` → tanggal hari ini dalam format `2025-01-15` (format yang disyaratkan sitemap)
- Kedua homepage ini selalu dimasukkan karena merupakan halaman utama website

---

Bagian 2 — Ambil Data Artikel dan Kategori

```
    // Ambil semua artikel (hanya kolom yang diperlukan)
    $select = $this->select('id_kategori, slug_id, slug_en, published_at')-
>findAll();

    // Ambil semua kategori
    $kategoriModel = new KategoriModel();
    $kategori = $kategoriModel->select('id_kategori, slug_id, slug_en,
created_at')->findAll();
```

- `$this->select(...)` → query builder CI4 untuk memilih kolom tertentu saja, agar lebih efisien daripada mengambil semua kolom (`SELECT *`)
- `->findAll()` → mengeksekusi query dan mengembalikan semua baris sebagai array asosiatif
- `new KategoriModel()` → membuat instance model lain di dalam model ini untuk mengambil data kategori

---

Bagian 3 — URL Halaman Kategori

```
    foreach ($kategori as $s) {
        if (!empty($s['slug_id'])) {
            $urls[] = [
                'loc'     => base_url('id/' . $s['slug_id']),
                'lastmod' => date('Y-m-d', strtotime($s['created_at']))
            ];
        }

        if (!empty($s['slug_en'])) {
            $urls[] = [
                'loc'     => base_url('en/' . $s['slug_en']),
                'lastmod' => date('Y-m-d', strtotime($s['created_at']))
            ];
```

```
        }
    }
```

- Iterasi setiap kategori dari database
- `!empty($s['slug_id'])` → pengecekan defensive: hanya tambahkan URL jika slug tidak kosong/null (menghindari URL rusak seperti `https://example.com/id/`)
- `base_url('id/' . $s['slug_id'])` → membangun URL kategori, misal `https://example.com/id/teknologi`
- `strtotime($s['created_at'])` → mengubah string tanggal dari database menjadi Unix timestamp
- `date('Y-m-d', ...)` → mengformat timestamp tadi ke format sitemap yang valid

---

Bagian 4 — Buat Category Map (Lookup Table)

```
    $categoryMap = [];
    foreach ($kategori as $c) {
        $categoryMap[$c['id_kategori']] = $c;
    }
```

Ini adalah teknik optimasi penting. Daripada melakukan pencarian loop di dalam loop ($O(n^2)$), kita membuat **lookup table** berbasis `id_kategori` sebagai key.

Tanpa category map (lambat):

```
// Harus loop semua kategori untuk setiap artikel — tidak efisien
foreach ($artikel as $a) {
    foreach ($kategori as $k) {
        if ($k['id_kategori'] === $a['id_kategori']) {
            // ketemu
        }
    }
}
```

Dengan category map (cepat):

```
// Akses langsung O(1) — jauh lebih efisien
$segment = $categoryMap[$s['id_kategori']];
```

Bagian 5 — URL Halaman Artikel

```
    foreach ($select as $s) {
        if (!isset($categoryMap[$s['id_kategori']])) {
```

```
            continue; // skip jika kategori tidak valid
        }

        $segment = $categoryMap[$s['id_kategori']];
        $lastmod = date('Y-m-d', strtotime($s['published_at']));

        // Bahasa ID
        if (!empty($s['slug_id']) && !empty($segment['slug_id'])) {
            $urls[] = [
                'loc'     => site_url('id/' . $segment['slug_id'] . '/' .
$s['slug_id']),
                'lastmod' => $lastmod
            ];
        }

        // Bahasa EN
        if (!empty($s['slug_en']) && !empty($segment['slug_en'])) {
            $urls[] = [
                'loc'     => site_url('en/' . $segment['slug_en'] . '/' .
$s['slug_en']),
                'lastmod' => $lastmod
            ];
        }
    }
```

- `!isset($categoryMap[$s['id_kategori']])` → jika artikel punya `id_kategori` yang tidak ada di tabel kategori (data orphan), artikel tersebut di-skip dengan `continue`
- `$segment` → data kategori milik artikel tersebut, didapat langsung dari lookup table
- `$lastmod` → menggunakan `published_at` artikel, bukan tanggal hari ini — lebih akurat untuk mesin pencari
- `site_url()` vs `base_url()` → keduanya menghasilkan URL lengkap, `site_url()` secara otomatis menambahkan `index.php` jika diperlukan oleh konfigurasi CI4, sedangkan `base_url()` tidak
- Struktur URL artikel: `/{lang}/{slug-kategori}/{slug-artikel}`, misal `/id/teknologi/cara-install-linux`
- Pengecekan `!empty()` pada **kedua** slug (kategori dan artikel) memastikan URL yang dihasilkan selalu valid

---

Bagian 6 — Return

```
    return $urls;
}
```

Mengembalikan array berisi semua URL yang sudah terkumpul ke controller.

Controller: Sitemap

```php
<?php
namespace App\Controllers;
use App\Controllers\BaseController;
use App\Models\ArtikelModel;
use App\Models\KategoriModel;
use CodeIgniter\HTTP\ResponseInterface;

class Sitemap extends BaseController
{
    public function sitemap()
    {
        session()->close();

        $artikelModel = new ArtikelModel();
        $urls = $artikelModel->getForSitemap();

        if ($this->request->getMethod() === 'head') {
            return $this->response
                ->setStatusCode(200)
                ->setHeader('Content-Type', 'application/xml');
        }

        return $this->response
            ->setHeader('Content-Type', 'application/xml; charset=UTF-8')
            ->setHeader('Cache-Control', 'public, max-age=3600')
            ->setBody(view('sitemap_xml', ['urls' => $urls]));
    }
}
```

Penjelasan Baris per Baris

`session()->close()`

```php
session()->close();
```

Menutup session sebelum memproses request. Ini penting untuk:

- **Performa** — sitemap bisa memakan waktu; menutup session membebaskan lock session lebih awal sehingga tab browser lain tidak menunggu
- **Resource** — bot mesin pencari sering mengakses sitemap, tidak butuh session

---

Inisialisasi Model dan Ambil Data

```php
$artikelModel = new ArtikelModel();
$urls = $artikelModel->getForSitemap();
```

Membuat instance model dan memanggil method `getForSitemap()` yang sudah kita buat. Hasilnya disimpan di `$urls`.

Menangani Request HEAD

```
if ($this->request->getMethod() === 'head') {
    return $this->response
        ->setStatusCode(200)
        ->setHeader('Content-Type', 'application/xml');
}
```

Request `HEAD` adalah request HTTP yang hanya meminta **header** tanpa body. Google Search Console dan beberapa crawler menggunakan HEAD request untuk mengecek apakah sitemap ada dan valid sebelum mengunduh isinya. Dengan menangani ini secara khusus:

- Tidak perlu memproses query database dan membangun XML jika hanya dicek keberadaannya
- Respons lebih cepat untuk bot

Mengirim Response XML

```
return $this->response
    ->setHeader('Content-Type', 'application/xml; charset=UTF-8')
    ->setHeader('Cache-Control', 'public, max-age=3600')
    ->setBody(view('sitemap_xml', ['urls' => $urls]));
```

- `Content-Type: application/xml; charset=UTF-8` → memberitahu browser dan bot bahwa response ini adalah XML, bukan HTML
- `Cache-Control: public, max-age=3600` → mengizinkan CDN dan browser meng-cache sitemap selama **1 jam** (3600 detik). Ini mengurangi beban server saat bot sering mengakses sitemap
- `view('sitemap_xml', ['urls' => $urls])` → merender view template dan mengirimkan hasilnya sebagai body response
- Tidak menggunakan `return view(...)` biasa karena kita perlu mengatur header custom — CI4's `$this->response` memberikan kontrol penuh atas response HTTP

View: sitemap_xml.php

```
<?php
echo '<?xml version="1.0" encoding="UTF-8"?>';
?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9
```

```
http://www.sitemaps.org/schemas/sitemap/0.9/sitemap.xsd">
<?php foreach ($urls as $u): ?>
  <url>
    <loc><?= esc($u['loc']) ?></loc>
    <lastmod><?= esc($u['lastmod']) ?></lastmod>
  </url>
<?php endforeach; ?>
</urlset>
```

Penjelasan Detail

Deklarasi XML via `echo`

```
<?php echo '<?xml version="1.0" encoding="UTF-8"?>'; ?>
```

Baris ini **tidak bisa** ditulis langsung sebagai `<?xml ... ?>` karena PHP akan mengira itu adalah PHP opening tag (`<?`). Solusinya adalah meng-echo string tersebut. Deklarasi ini wajib ada di baris pertama file XML.

Namespace dan Schema pada `<urlset>`

```
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="...">
```

- `xmlns` → mendefinisikan namespace default dokumen sesuai standar Sitemaps Protocol
- `xmlns:xsi` → namespace untuk XML Schema Instance
- `xsi:schemaLocation` → lokasi schema XSD untuk validasi. Google tidak mewajibkan ini, tapi membantu validator XML

Loop dan Fungsi `esc()`

```
<?php foreach ($urls as $u): ?>
  <url>
    <loc><?= esc($u['loc']) ?></loc>
    <lastmod><?= esc($u['lastmod']) ?></lastmod>
  </url>
<?php endforeach; ?>
```

- `foreach ... endforeach` → sintaks alternatif PHP yang lebih mudah dibaca dalam template
- `esc($u['loc'])` → fungsi bawaan CI4 untuk **escape output**. Ini mengonversi karakter seperti `&`, `<`, `>`, `"` menjadi entitas XML yang aman. Sangat penting untuk mencegah karakter dalam URL merusak struktur XML

- `<loc>` → elemen wajib sitemap yang berisi URL halaman
- `<lastmod>` → elemen opsional yang menyatakan kapan halaman terakhir dimodifikasi (format: `YYYY-MM-DD`)

Routing

```
$routes->get('sitemap.xml', 'Sitemap::sitemap');
$routes->head('sitemap.xml', 'Sitemap::sitemap');
```

Penjelasan

- `$routes->get(...)` → menangani request `GET` ke URL `https://example.com/sitemap.xml`, mengarahkannya ke method `sitemap()` di class `Sitemap`
- `$routes->head(...)` → menangani request `HEAD` ke URL yang sama. Tanpa baris ini, CI4 tidak akan merespons HEAD request karena secara default ia tidak otomatis memetakan HEAD ke handler GET yang sama
- `'sitemap.xml'` → route tanpa slash awal, CI4 akan otomatis menanganinya relatif terhadap base URL
- `'Sitemap::sitemap'` → format `NamaController::namaMethod`

> **Catatan:** Route ini harus ditempatkan di luar grup route yang memerlukan autentikasi/filter, agar bot mesin pencari bisa mengaksesnya tanpa login.

Alur Kerja Keseluruhan

```
Bot/Browser
    │
    │  GET /sitemap.xml
    ▼
Router CI4
    │
    │  Cocokkan route → Sitemap::sitemap()
    ▼
Controller: Sitemap::sitemap()
    │
    ├─ session()->close()
    │
    ├─ Cek: HEAD request? → Kembalikan header saja
    │
    ├─ ArtikelModel::getForSitemap()
    │       │
    │       ├─ Tambah URL homepage (ID & EN)
    │       ├─ Query semua kategori
    │       ├─ Tambah URL tiap kategori (ID & EN)
    │       ├─ Buat categoryMap untuk lookup cepat
    │       ├─ Query semua artikel
    │       └─ Tambah URL tiap artikel (ID & EN)
```

```
    │
    ├─ Set header: Content-Type, Cache-Control
    │
    └─ Render view 'sitemap_xml' → Output XML
```

Tips & Catatan Penting

1. Pastikan `published_at` Tidak Null

Jika ada artikel dengan `published_at = NULL`, `strtotime(NULL)` akan mengembalikan `false`, dan `date('Y-m-d', false)` akan menghasilkan tanggal yang salah (`1970-01-01`). Tambahkan pengecekan:

```php
$lastmod = !empty($s['published_at'])
    ? date('Y-m-d', strtotime($s['published_at']))
    : date('Y-m-d');
```

2. Filter Artikel yang Sudah Dipublikasikan

Sebaiknya hanya masukkan artikel yang sudah dipublikasikan ke sitemap:

```php
$select = $this->select('id_kategori, slug_id, slug_en, published_at')
               ->where('published_at <=', date('Y-m-d H:i:s'))
               ->findAll();
```

3. Batasi Jumlah URL

Google merekomendasikan maksimal **50.000 URL per file sitemap**. Jika artikel banyak, pertimbangkan menggunakan **Sitemap Index** yang memecah sitemap menjadi beberapa file.

4. Daftarkan ke Google Search Console

Setelah sitemap dibuat, daftarkan di Google Search Console → Sitemaps → masukkan URL `https://example.com/sitemap.xml`.

5. Verifikasi XML Valid

Cek output sitemap dengan membuka URL-nya di browser dan validasi menggunakan XML Sitemap Validator.