SvelteKit SEO 2026 Implementation Guide

> *A comprehensive, code-driven guide to making SvelteKit applications rank, perform, and convert in 2026.*

Table of Contents

Chapter 1: SEO Fundamentals

What is SEO in 2026?

Search Engine Optimization in 2026 is about three pillars:

1. **Technical Excellence** — Fast, crawlable, structured
2. **Content Quality** — Authoritative, helpful, experience-driven
3. **User Experience** — Core Web Vitals, mobile-first, accessible

How Search Engines Work

```
Crawling → Indexing → Ranking → Serving Results
   │           │          │           │
   │           │          │           └─ SERP features, snippets
   │           │          └─ Relevance, authority, UX signals
   │           └─ Parse content, understand relationships
   └─ Discover pages via links, sitemaps, APIs
```

The SEO Ranking Factors That Matter

Factor	Weight	SvelteKit Advantage
Content quality	Very High	Markdown + dynamic rendering
Page experience (CWV)	High	SSR + streaming
Mobile usability	High	Responsive by default
HTTPS	Baseline	Vercel/Cloudflare auto-SSL
Internal linking	Medium	Programmatic link generation
Structured data	Medium	Server-side JSON-LD
Page speed	High	Code splitting + preloading
Crawlability	Baseline	SSR = full HTML on first load

SEO Mindset for Developers

```
Traditional: "Build it, then SEO it"
Modern:      "SEO is architecture. Build it in from day one."
```

Key principles:

- Every page has one purpose (one primary keyword intent)
- Every page is a landing page (search can find any page)
- Every page tells search engines what it is (structured data)
- Every page loads fast (performance is a ranking signal)

Chapter 2: Technical SEO Basics

Crawlability

Search engines must be able to discover and read your pages.

robots.txt

```
# static/robots.txt
User-agent: *
Allow: /
Disallow: /api/
Disallow: /admin/
Disallow: /private/

Sitemap: https://yoursite.com/sitemap.xml
```

Dynamic robots.txt with SvelteKit

```
// src/routes/robots.txt/+server.js
export function GET() {
  const body = `User-agent: *
Allow: /
Disallow: /api/
Disallow: /admin/

Sitemap: https://yoursite.com/sitemap.xml`;

  return new Response(body, {
    headers: {
      'Content-Type': 'text/plain'
    }
  });
}
```

XML Sitemap

```
// src/routes/sitemap.xml/+server.js
import { SITE_URL } from '$lib/config';

const pages = [
  { path: '/', priority: 1.0, changefreq: 'weekly' },
  { path: '/about', priority: 0.8, changefreq: 'monthly' },
  { path: '/features', priority: 0.8, changefreq: 'monthly' },
  { path: '/pricing', priority: 0.9, changefreq: 'weekly' },
  { path: '/blog', priority: 0.9, changefreq: 'daily' },
  { path: '/contact', priority: 0.6, changefreq: 'yearly' }
];

export async function GET() {
  const sitemap = `<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
${pages
  .map(
    (page) => `  <url>
    <loc>${SITE_URL}${page.path}</loc>
    <lastmod>${new Date().toISOString().split('T')[0]}</lastmod>
    <changefreq>${page.changefreq}</changefreq>
    <priority>${page.priority}</priority>
  </url>`
  )
  .join('\n')}
</urlset>`;

  return new Response(sitemap, {
    headers: {
      'Content-Type': 'application/xml',
      'Cache-Control': 'max-age=3600'
    }
  });
}
```

Dynamic Sitemap with Blog Posts

```
// src/routes/sitemap.xml/+server.js
import { SITE_URL } from '$lib/config';

async function getBlogPosts() {
  // Fetch from CMS, database, or filesystem
  const modules = import.meta.glob('/src/content/blog/*.md', { eager: true });
  return Object.entries(modules).map(([path, module]) => ({
    slug: path.split('/').pop().replace('.md', ''),
    lastmod: module.metadata?.date || new Date().toISOString()
  }));
}

export async function GET() {
  const posts = await getBlogPosts();

  const staticPages = [
    { loc: '/', priority: 1.0 },
    { loc: '/about', priority: 0.8 },
    { loc: '/pricing', priority: 0.9 },
    { loc: '/blog', priority: 0.9 }
  ];
  const blogPages = posts.map((post) => ({
    loc: `/blog/${post.slug}`,
    lastmod: post.lastmod,
    priority: 0.7
  }));

  const allPages = [...staticPages, ...blogPages];

  const sitemap = `<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
${allPages
  .map(
    (page) => `  <url>
    <loc>${SITE_URL}${page.loc}</loc>
    ${page.lastmod ? `<lastmod>${page.lastmod}</lastmod>` : ''}
    <priority>${page.priority}</priority>
  </url>`
  )
  .join('\n')}
</urlset>`;

  return new Response(sitemap, {
    headers: {
      'Content-Type': 'application/xml',
      'Cache-Control': 'max-age=3600'
    }
  });
}
```

Canonical URLs

```
<!-- src/routes/+layout.svelte -->
<script>
  import { page } from '$app/stores';
  import { SITE_URL } from '$lib/config';
</script>

<svelte:head>
  <link rel="canonical" href="{SITE_URL}{$page.url.pathname}" />
</svelte:head>
```

HTTP Status Codes

```
// src/routes/old-page/+page.server.js
import { redirect } from '@sveltejs/kit';

export function load() {
  // 301 Permanent Redirect
  redirect(301, '/new-page');
}
```

```
// src/routes/blog/[slug]/+page.server.js
import { error } from '@sveltejs/kit';
```

```
export async function load({ params }) {
  const post = await getPost(params.slug);

  if (!post) {
    error(404, {
      message: 'Post not found'
    });
  }

  return { post };
}
```

Chapter 3: SvelteKit for SEO

Why SvelteKit Excels at SEO

1. **Server-Side Rendering (SSR)** — Full HTML delivered to crawlers
2. **Static Generation (SSG)** — Pre-rendered pages at build time
3. **Streaming** — Fast Time to First Byte (TTFB)
4. **Code Splitting** — Only load what each page needs
5. **Preloading** — Prefetch on hover/viewport
6. **Head Management** — `<svelte:head>` for meta tags

Rendering Strategies

```
// src/routes/blog/[slug]/+page.server.js

// SSR (default) — Best for dynamic content
export async function load({ params }) {
  const post = await fetchPost(params.slug);
  return { post };
}
```

```
// src/routes/about/+page.js

// SSG — Best for static content
export const prerender = true;

export function load() {
  return {
    title: 'About Us',
    description: 'Learn about our mission and team.'
  };
}
```

```
// src/routes/dashboard/+page.js

// CSR Only — NOT good for SEO (use only for app-like pages)
export const ssr = false;
```

Prerendering for SEO

```
// svelte.config.js
const config = {
  kit: {
    prerender: {
      entries: ['*'],               // Crawl all discoverable pages
      handleHttpError: 'warn',      // Don't fail build on 404s
      handleMissingId: 'warn'
    }
  }
};

export default config;
```

```
// src/routes/+layout.js
// Prerender all pages by default
export const prerender = true;
```

```
// src/routes/api/contact/+server.js
// Override for dynamic routes
export const prerender = false;
```

SvelteKit Head Management

```
<!-- src/routes/pricing/+page.svelte -->
<script>
  let { data } = $props();
</script>

<svelte:head>
  <title>{data.title} | ShipForge</title>
  <meta name="description" content={data.description} />
  <meta name="robots" content="index, follow" />
  <link rel="canonical" href="https://shipforge.dev/pricing" />
</svelte:head>

<h1>{data.title}</h1>
```

Error Pages for SEO

```
<!-- src/routes/+error.svelte -->
<script>
  import { page } from '$app/stores';
</script>

<svelte:head>
  <title>{$page.status} | ShipForge</title>
  <meta name="robots" content="noindex" />
</svelte:head>

{#if $page.status === 404}
  <h1>Page Not Found</h1>
  <p>The page you're looking for doesn't exist.</p>
  <a href="/">Go back home</a>
{:else}
  <h1>Something Went Wrong</h1>
  <p>{$page.error?.message}</p>
{/if}
```

Chapter 4: Meta Tags Implementation

SEO Meta Tag Component

```
<!-- src/lib/components/SEO.svelte -->
<script>
  import { page } from '$app/stores';

  let {
    title = '',
    description = '',
    keywords = '',
    image = '',
    type = 'website',
    noindex = false,
    nofollow = false,
    canonical = ''
  } = $props();

  const SITE_NAME = 'ShipForge';
  const SITE_URL = 'https://shipforge.dev';
  const DEFAULT_IMAGE = `${SITE_URL}/og-default.png`;

  const fullTitle = title ? `${title} | ${SITE_NAME}` : SITE_NAME;
  const fullImage = image || DEFAULT_IMAGE;
  const fullCanonical = canonical || `${SITE_URL}${$page.url.pathname}`;
  const robotsContent = [
    noindex ? 'noindex' : 'index',
    nofollow ? 'nofollow' : 'follow'
  ].join(', ');
</script>

<svelte:head>
  <!-- Primary Meta Tags -->
  <title>{fullTitle}</title>
  <meta name="description" content={description} />
  <meta name="keywords" content={keywords} />
  <meta name="robots" content={robotsContent} />
  <link rel="canonical" href={fullCanonical} />

  <!-- Open Graph -->
  <meta property="og:type" content={type} />
  <meta property="og:title" content={fullTitle} />
  <meta property="og:description" content={description} />
  <meta property="og:image" content={fullImage} />
  <meta property="og:url" content={fullCanonical} />
  <meta property="og:site_name" content={SITE_NAME} />
  <!-- Twitter Card -->
  <meta name="twitter:card" content="summary_large_image" />
  <meta name="twitter:title" content={fullTitle} />
  <meta name="twitter:description" content={description} />
  <meta name="twitter:image" content={fullImage} />

  <!-- Additional -->
  <meta name="author" content={SITE_NAME} />
  <meta name="theme-color" content="#6366f1" />
</svelte:head>
```

Using the SEO Component

```
<!-- src/routes/+page.svelte -->
<script>
  import SEO from '$lib/components/SEO.svelte';
</script>

<SEO
  title="Build & Ship Faster"
  description="ShipForge is the modern SaaS boilerplate for SvelteKit. Launch your product in
days, not months."
  keywords="sveltekit, saas, boilerplate, web development"
  image="https://shipforge.dev/og-home.png"
/>

<main>
  <h1>Build & Ship Faster</h1>
</main>
```

Dynamic Meta Tags from Data

```
<!-- src/routes/blog/[slug]/+page.svelte -->
<script>
  import SEO from '$lib/components/SEO.svelte';

  let { data } = $props();
</script>

<SEO
  title={data.post.title}
  description={data.post.excerpt}
  image={data.post.coverImage}
  type="article"
  keywords={data.post.tags.join(', ')}
/>

<article>
  <h1>{data.post.title}</h1>
  <time datetime={data.post.date}>{data.post.formattedDate}</time>
  {@html data.post.content}
</article>
```

Essential Meta Tags Checklist

```html
<!-- Must-have for every page -->
<title>Page Title | Brand (50-60 chars)</title>
<meta name="description" content="155 character max description" />
<link rel="canonical" href="https://yoursite.com/page" />
<meta name="robots" content="index, follow" />

<!-- Must-have for crawlers -->
<html lang="en">
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />

<!-- Must-have for social sharing -->
<meta property="og:title" content="Title" />
<meta property="og:description" content="Description" />
<meta property="og:image" content="https://yoursite.com/og.png" />
<meta property="og:url" content="https://yoursite.com/page" />
<meta name="twitter:card" content="summary_large_image" />
```

Title Tag Best Practices

```javascript
// src/lib/utils/seo.js

export function generateTitle(pageTitle, options = {}) {
  const {
    siteName = 'ShipForge',
    separator = '|',
    maxLength = 60
  } = options;

  if (!pageTitle) return siteName;

  const full = `${pageTitle} ${separator} ${siteName}`;

  if (full.length > maxLength) {
    const available = maxLength - siteName.length - separator.length - 2;
    return `${pageTitle.substring(0, available)}... ${separator} ${siteName}`;
  }

  return full;
}

export function generateDescription(text, maxLength = 155) {
  if (!text) return '';
  if (text.length <= maxLength) return text;
  return text.substring(0, maxLength - 3).trim() + '...';
}
```

Chapter 5: Open Graph & Social Sharing

Open Graph Protocol

Open Graph meta tags control how your pages appear when shared on social media.

```
<!-- Full Open Graph implementation -->
<svelte:head>
  <!-- Basic OG -->
  <meta property="og:type" content="website" />
  <meta property="og:title" content="ShipForge — Build & Ship Faster" />
  <meta property="og:description" content="The modern SaaS boilerplate for SvelteKit." />
  <meta property="og:image" content="https://shipforge.dev/og-home.png" />
  <meta property="og:image:width" content="1200" />
  <meta property="og:image:height" content="630" />
  <meta property="og:image:alt" content="ShipForge landing page preview" />
  <meta property="og:url" content="https://shipforge.dev" />
  <meta property="og:site_name" content="ShipForge" />
  <meta property="og:locale" content="en_US" />
</svelte:head>
```

Article Open Graph

```
<!-- For blog posts -->
<svelte:head>
  <meta property="og:type" content="article" />
  <meta property="article:published_time" content={post.date} />
  <meta property="article:modified_time" content={post.updated} />
  <meta property="article:author" content={post.author} />
  <meta property="article:section" content={post.category} />
  {#each post.tags as tag}
    <meta property="article:tag" content={tag} />
  {/each}
</svelte:head>
```

Twitter Card Tags

```
<svelte:head>
  <!-- Summary with large image (recommended) -->
  <meta name="twitter:card" content="summary_large_image" />
  <meta name="twitter:site" content="@shipforge" />
  <meta name="twitter:creator" content="@johndoe" />
  <meta name="twitter:title" content="ShipForge — Build & Ship Faster" />
  <meta name="twitter:description" content="The modern SaaS boilerplate for SvelteKit." />
  <meta name="twitter:image" content="https://shipforge.dev/twitter-card.png" />
  <meta name="twitter:image:alt" content="ShipForge preview" />
</svelte:head>
```

Dynamic OG Image Generation

```
// src/routes/og/[slug].png/+server.js
import { ImageResponse } from '@vercel/og';
import { SITE_URL } from '$lib/config';

export async function GET({ params }) {
  const { slug } = params;
  // Fetch page data
  const pageData = await getPageData(slug);

  const html = `
    <div style="
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      width: 1200px;
      height: 630px;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      color: white;
      font-family: Inter, sans-serif;
      padding: 60px;
    ">
      <div style="font-size: 64px; font-weight: 800; text-align: center; margin-bottom: 20px;">
        ${pageData.title}
      </div>
      <div style="font-size: 28px; opacity: 0.9; text-align: center;">
        ${pageData.description}
      </div>
      <div style="
        position: absolute;
        bottom: 40px;
        font-size: 24px;
        opacity: 0.7;
      ">
        shipforge.dev
      </div>
    </div>
  `;

  return new ImageResponse(html, {
    width: 1200,
    height: 630
  });
}
```

OG Image Best Practices

```
Dimensions:
  - OG Image: 1200 x 630 px (1.91:1 ratio)
  - Twitter: 1200 x 628 px (same as OG, practically)
  - Square fallback: 400 x 400 px minimum

File requirements:
  - Format: PNG or JPEG
  - Max size: < 5MB (aim for < 500KB)
  - Resolution: 2x for retina

Design tips:
  - Keep text readable at small sizes
  - Include your logo/brand
```

```
 - Use contrasting colors
 - Leave safe margins (avoid edges)
 - Test on dark and light backgrounds
```

Social Sharing Preview Testing

```
// Tools for testing social sharing previews:
// 1. Facebook Sharing Debugger: https://developers.facebook.com/tools/debug/
// 2. Twitter Card Validator: https://cards-dev.twitter.com/validator
// 3. LinkedIn Post Inspector: https://www.linkedin.com/post-inspector/
// 4. Open Graph Check: https://opengraph.xyz
```

Chapter 6: Structured Data & Rich Results

What is Structured Data?

Structured data is JSON-LD markup that helps search engines understand your content and display rich results (stars, prices, FAQs, breadcrumbs, etc.).

JSON-LD Component

```
<!-- src/lib/components/JsonLd.svelte -->
<script>
  let { schema } = $props();
</script>

<svelte:head>
  {@html `<script type="application/ld+json">${JSON.stringify(schema)}</script>`}
</svelte:head>
```

Organization Schema

```
<script>
  import JsonLd from '$lib/components/JsonLd.svelte';

  const orgSchema = {
    '@context': 'https://schema.org',
    '@type': 'Organization',
    name: 'ShipForge',
    url: 'https://shipforge.dev',
    logo: 'https://shipforge.dev/logo.png',
    description: 'Modern SaaS boilerplate for SvelteKit',
    sameAs: [
      'https://twitter.com/shipforge',
      'https://github.com/shipforge',
      'https://linkedin.com/company/shipforge'
    ],
    contactPoint: {
      '@type': 'ContactPoint',
      email: 'hello@shipforge.dev',
      contactType: 'customer service'
    }
  };
```

```
</script>

<JsonLd schema={orgSchema} />
```

Website Schema with Search Action

```
<script>
  import JsonLd from '$lib/components/JsonLd.svelte';

  const websiteSchema = {
    '@context': 'https://schema.org',
    '@type': 'WebSite',
    name: 'ShipForge',
    url: 'https://shipforge.dev',
    potentialAction: {
      '@type': 'SearchAction',
      target: {
        '@type': 'EntryPoint',
        urlTemplate: 'https://shipforge.dev/search?q={search_term_string}'
      },
      'query-input': 'required name=search_term_string'
    }
  };
</script>

<JsonLd schema={websiteSchema} />
```

Article/Blog Post Schema

```
<!-- src/routes/blog/[slug]/+page.svelte -->
<script>
  import JsonLd from '$lib/components/JsonLd.svelte';

  let { data } = $props();
  const { post } = data;

  const articleSchema = {
    '@context': 'https://schema.org',
    '@type': 'Article',
    headline: post.title,
    description: post.excerpt,
    image: post.coverImage,
    datePublished: post.date,
    dateModified: post.updated || post.date,
    author: {
      '@type': 'Person',
      name: post.author.name,
      url: post.author.url
    },
    publisher: {
      '@type': 'Organization',
      name: 'ShipForge',
      logo: {
        '@type': 'ImageObject',
        url: 'https://shipforge.dev/logo.png'
      }
    },
    mainEntityOfPage: {
      '@type': 'WebPage',
      '@id': `https://shipforge.dev/blog/${post.slug}`
    }
  };
</script>

<JsonLd schema={articleSchema} />
```

FAQ Schema

```
<script>
  import JsonLd from '$lib/components/JsonLd.svelte';

  const faqs = [
    {
      question: 'What is ShipForge?',
      answer: 'ShipForge is a production-ready SvelteKit boilerplate for building SaaS
applications.'
    },
    {
      question: 'How much does it cost?',
      answer: 'ShipForge starts at $49 for a single project license.'
    },
    {
      question: 'Do I need SvelteKit experience?',
      answer: 'Basic knowledge of Svelte and JavaScript is recommended.'
    }
  ];

  const faqSchema = {
    '@context': 'https://schema.org',
    '@type': 'FAQPage',
    mainEntity: faqs.map((faq) => ({
      '@type': 'Question',
      name: faq.question,
      acceptedAnswer: {
        '@type': 'Answer',
        text: faq.answer
      }
    }))
  };
</script>

<JsonLd schema={faqSchema} />

<section class="faq">
  {#each faqs as faq}
    <details>
      <summary>{faq.question}</summary>
      <p>{faq.answer}</p>
    </details>
  {/each}
</section>
```

Product Schema

```
<script>
  import JsonLd from '$lib/components/JsonLd.svelte';

  const productSchema = {
    '@context': 'https://schema.org',
    '@type': 'Product',
    name: 'ShipForge Pro',
    description: 'Production-ready SvelteKit SaaS boilerplate',
    image: 'https://shipforge.dev/product-image.png',
    brand: {
      '@type': 'Brand',
      name: 'ShipForge'
    },
    offers: {
      '@type': 'Offer',
      price: '99',
      priceCurrency: 'USD',
      availability: 'https://schema.org/InStock',
      url: 'https://shipforge.dev/pricing'
    },
    aggregateRating: {
      '@type': 'AggregateRating',
      ratingValue: '4.8',
      reviewCount: '127'
    }
  };
</script>

<JsonLd schema={productSchema} />
```

Breadcrumb Schema

```
<script>
  import JsonLd from '$lib/components/JsonLd.svelte';
  import { page } from '$app/stores';

  function generateBreadcrumbs(pathname) {
    const segments = pathname.split('/').filter(Boolean);
    const items = [
      { name: 'Home', url: 'https://shipforge.dev/' }
    ];

    let path = '';
    for (const segment of segments) {
      path += `/${segment}`;
      items.push({
        name: segment.charAt(0).toUpperCase() + segment.slice(1).replace(/-/g, ' '),
        url: `https://shipforge.dev${path}`
      });
    }

    return items;
  }

  $effect(() => {
    breadcrumbs = generateBreadcrumbs($page.url.pathname);
  });

  let breadcrumbs = $state([]);

  const breadcrumbSchema = $derived({
    '@context': 'https://schema.org',
    '@type': 'BreadcrumbList',
    itemListElement: breadcrumbs.map((item, index) => ({
      '@type': 'ListItem',
      position: index + 1,
      name: item.name,
      item: item.url
    }))
  });
</script>

<JsonLd schema={breadcrumbSchema} />

<nav aria-label="Breadcrumb">
  <ol>
    {#each breadcrumbs as crumb, i}
      <li>
        {#if i < breadcrumbs.length - 1}
          <a href={crumb.url}>{crumb.name}</a>
        {:else}
          <span aria-current="page">{crumb.name}</span>
        {/if}
      </li>
    {/each}
  </ol>
</nav>
```

Testing Structured Data

```
# Tools for validating structured data:
# 1. Google Rich Results Test: https://search.google.com/test/rich-results
# 2. Schema.org Validator: https://validator.schema.org/
# 3. Google Search Console → Enhancements

# Quick validation in dev: check the <script type="application/ld+json"> in page source
```

Chapter 7: Core Web Vitals

The Three Core Web Vitals (2026)

Metric	What It Measures	Good	Needs Improvement	Poor

LCP (Largest Contentful Paint)	Loading speed	< 2.5s	2.5s - 4.0s	> 4.0s
INP (Interaction to Next Paint)	Responsiveness	< 200ms	200ms - 500ms	> 500ms
CLS (Cumulative Layout Shift)	Visual stability	< 0.1	0.1 - 0.25	> 0.25

LCP Optimization

```
<!-- Optimize the largest element on the page (usually hero image or heading) -->

<!-- 1. Preload critical images -->
<svelte:head>
  <link
    rel="preload"
    as="image"
    href="/hero-image.webp"
    type="image/webp"
    fetchpriority="high"
  />
</svelte:head>

<!-- 2. Use fetchpriority on the LCP image -->
<img
  src="/hero-image.webp"
  alt="Hero"
  width="1200"
  height="630"
  fetchpriority="high"
  decoding="async"
/>

<!-- 3. Inline critical CSS -->
<style>
  /* Critical above-the-fold styles should be in the component */
  .hero {
    min-height: 100vh;
    display: flex;
    align-items: center;
  }
</style>
```

INP Optimization

```
<script>
  // 1. Defer non-critical work
  function handleClick(event) {
    // Immediate visual feedback
    event.target.classList.add('active');

    // Defer heavy computation
    requestAnimationFrame(() => {
      setTimeout(() => {
        performHeavyComputation();
      }, 0);
    });
  }

  // 2. Use web workers for CPU-intensive tasks
  async function processData(data) {
    const worker = new Worker(
      new URL('$lib/workers/processor.js', import.meta.url),
      { type: 'module' }
    );

    return new Promise((resolve) => {
      worker.onmessage = (e) => {
        resolve(e.data);
        worker.terminate();
      };
      worker.postMessage(data);
    });
  }

  // 3. Debounce input handlers
  let searchTimeout;
  function handleSearch(event) {
    clearTimeout(searchTimeout);
    searchTimeout = setTimeout(() => {
      performSearch(event.target.value);
    }, 300);
  }
</script>

<!-- 4. Use CSS containment -->
<style>
  .card-grid {
    contain: layout style;
  }
  .card {
    contain: content;
  }
</style>
```

CLS Optimization

```
<!-- 1. Always set explicit dimensions on images -->
<img
  src="/photo.webp"
  alt="Description"
  width="800"
  height="450"
  loading="lazy"
  decoding="async"
/>

<!-- 2. Use aspect-ratio for responsive containers -->
<style>
  .video-container {
    aspect-ratio: 16 / 9;
    width: 100%;
    background: #f0f0f0; /* Placeholder color */
  }

  .image-container {
    aspect-ratio: 4 / 3;
    width: 100%;
    overflow: hidden;
  }
</style>

<!-- 3. Reserve space for dynamic content -->
<style>
  .ad-slot {
    min-height: 250px; /* Reserve space for ad */
    width: 100%;
  }

  .skeleton {
    min-height: 200px;
    background: linear-gradient(90deg, #f0f0f0 25%, #e0e0e0 50%, #f0f0f0 75%);
    background-size: 200% 100%;
    animation: shimmer 1.5s infinite;
  }

  @keyframes shimmer {
    0% { background-position: 200% 0; }
    100% { background-position: -200% 0; }
  }
</style>

<!-- 4. Avoid inserting content above existing content -->
<!-- BAD: Banner that pushes content down -->
<!-- GOOD: Fixed/overlay banner that doesn't affect layout -->
```

Font Loading Strategy (Prevents CLS)

```
<!-- src/app.html -->
<head>
  <!-- Preload critical fonts -->
  <link
    rel="preload"
    href="/fonts/inter-var.woff2"
    as="font"
    type="font/woff2"
    crossorigin="anonymous"
  />
</head>
```

```
/* src/app.css */
@font-face {
  font-family: 'Inter';
  src: url('/fonts/inter-var.woff2') format('woff2');
  font-weight: 100 900;
  font-display: swap; /* Use swap to prevent invisible text */
  unicode-range: U+0000-00FF; /* Only Latin characters */
}

/* Size-adjusted fallback to prevent CLS */
@font-face {
  font-family: 'Inter Fallback';
  src: local('Arial');
  ascent-override: 90%;
  descent-override: 22%;
  line-gap-override: 0%;
  size-adjust: 107%;
}

body {
  font-family: 'Inter', 'Inter Fallback', system-ui, sans-serif;
}
```

Measuring Core Web Vitals

```
// src/lib/utils/vitals.js
export function measureWebVitals() {
  if (typeof window === 'undefined') return;

  import('web-vitals').then(({ onLCP, onINP, onCLS }) => {
    onLCP(sendToAnalytics);
    onINP(sendToAnalytics);
    onCLS(sendToAnalytics);
  });
}

function sendToAnalytics(metric) {
  console.log(`${metric.name}: ${metric.value}`);

  // Send to your analytics endpoint
  if (navigator.sendBeacon) {
    navigator.sendBeacon('/api/vitals', JSON.stringify({
      name: metric.name,
      value: metric.value,
      rating: metric.rating,
      delta: metric.delta,
      id: metric.id,
      page: window.location.pathname
    }));
  }
}
```

```
<!-- src/routes/+layout.svelte -->
<script>
  import { browser } from '$app/environment';
  import { measureWebVitals } from '$lib/utils/vitals';

  $effect(() => {
    if (browser) {
      measureWebVitals();
    }
  });
</script>
```

Chapter 8: Page Speed Optimization

Image Optimization

```
<!-- src/lib/components/OptimizedImage.svelte -->
<script>
  let {
    src,
    alt,
    width,
    height,
    loading = 'lazy',
    fetchpriority = 'auto',
    sizes = '100vw',
    class: className = ''
  } = $props();

  // Generate srcset for responsive images
  const widths = [320, 640, 768, 1024, 1280, 1536];
  const srcset = widths
    .filter(w => w <= width * 2)
    .map(w => `${src}?w=${w}&format=webp ${w}w`)
    .join(', ');
</script>

<picture>
  <source {srcset} {sizes} type="image/webp" />
  <img
    {src}
    {alt}
    {width}
    {height}
    {loading}
    {fetchpriority}
    decoding="async"
    class={className}
  />
</picture>

<style>
  img {
    max-width: 100%;
    height: auto;
    display: block;
  }
</style>
```

Critical CSS Inlining

```
// vite.config.js
import { sveltekit } from '@sveltejs/kit/vite';

export default {
  plugins: [sveltekit()],
  css: {
    devSourcemap: true
  },
  build: {
    cssMinify: 'lightningcss',
    rollupOptions: {
      output: {
        // Ensure CSS is split per route for optimal loading
        assetFileNames: 'assets/[name]-[hash][extname]'
      }
    }
  }
};
```

JavaScript Bundle Optimization

```
// Dynamic imports for code splitting
// Only load heavy libraries when needed

// BAD: Always loaded
import Chart from 'chart.js';

// GOOD: Loaded on demand
const loadChart = () => import('chart.js');
```

```
<script>
  let showChart = $state(false);
  let Chart;

  async function initChart() {
    const module = await import('chart.js/auto');
    Chart = module.default;
    showChart = true;
  }
</script>

<button onclick={initChart}>Show Analytics</button>

{#if showChart}
  <canvas id="chart"></canvas>
{/if}
```

Preloading and Prefetching

```
<!-- SvelteKit auto-preloads links on hover -->
<!-- You can customize this behavior -->
```

```
<!-- Preload on viewport (more aggressive) -->
<a href="/pricing" data-sveltekit-preload-data="hover">
  View Pricing
</a>

<!-- Disable preloading for external links -->
<a href="https://external.com" data-sveltekit-preload-data="off">
  External Link
</a>
```

```
// src/routes/+layout.js
// Preload data for linked pages
export const prerender = false;
export const trailingSlash = 'never';
```

Caching Strategy

```
// src/hooks.server.js
export async function handle({ event, resolve }) {
  const response = await resolve(event);

  // Cache static assets aggressively
  if (event.url.pathname.startsWith('/assets/')) {
    response.headers.set(
      'Cache-Control',
      'public, max-age=31536000, immutable'
    );
  }

  // Cache HTML pages with stale-while-revalidate
  if (response.headers.get('content-type')?.includes('text/html')) {
    response.headers.set(
      'Cache-Control',
      'public, max-age=60, stale-while-revalidate=600'
    );
  }

  return response;
}
```

Lazy Loading Components

```
<script>
  import { browser } from '$app/environment';

  let visible = $state(false);
  let observer;
  let sentinel = $state(null);

  $effect(() => {
    if (!browser || !sentinel) return;

    observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          visible = true;
          observer.disconnect();
        }
      },
      { rootMargin: '200px' }
    );

    observer.observe(sentinel);

    return () => observer?.disconnect();
  });
</script>

<div bind:this={sentinel}>
  {#if visible}
    {#await import('$lib/components/HeavyComponent.svelte') then module}
      <module.default />
    {/await}
  {:else}
    <div class="skeleton" style="min-height: 400px;"></div>
  {/if}
</div>
```

Performance Budget

```
// Target performance metrics
const PERFORMANCE_BUDGET = {
  // Bundle sizes (gzipped)
  js: '150KB',          // Total JS
  css: '50KB',          // Total CSS
  fonts: '100KB',       // Total fonts
  images_per_page: '500KB',

  // Timing
  ttfb: '200ms',        // Time to First Byte
  fcp: '1.5s',          // First Contentful Paint
  lcp: '2.5s',          // Largest Contentful Paint
  tti: '3.5s',          // Time to Interactive

  // Core Web Vitals
  lcp_target: '2.5s',
  inp_target: '200ms',
  cls_target: '0.1'
};
```

Chapter 9: Internal Linking Strategy

Why Internal Links Matter for SEO

Internal links:

- Help search engines discover pages
- Distribute page authority (PageRank)

- Establish content hierarchy
- Improve user navigation
- Reduce bounce rate

Navigation Component with SEO

```
<!-- src/lib/components/Navigation.svelte -->
<script>
  import { page } from '$app/stores';

  const navLinks = [
    { href: '/', label: 'Home' },
    { href: '/features', label: 'Features' },
    { href: '/pricing', label: 'Pricing' },
    { href: '/blog', label: 'Blog' },
    { href: '/about', label: 'About' },
    { href: '/contact', label: 'Contact' }
  ];
</script>

<nav aria-label="Main navigation">
  <ul>
    {#each navLinks as link}
      <li>
        <a
          href={link.href}
          aria-current={$page.url.pathname === link.href ? 'page' : undefined}
        >
          {link.label}
        </a>
      </li>
    {/each}
  </ul>
</nav>
```

Breadcrumb Navigation

```
<!-- src/lib/components/Breadcrumbs.svelte -->
<script>
  import { page } from '$app/stores';

  const routeLabels = {
    '': 'Home',
    'blog': 'Blog',
    'features': 'Features',
    'pricing': 'Pricing',
    'about': 'About',
    'contact': 'Contact'
  };

  const crumbs = $derived(() => {
    const segments = $page.url.pathname.split('/').filter(Boolean);
    let path = '';

    const items = [{ label: 'Home', href: '/' }];
    for (const segment of segments) {
      path += `/${segment}`;
      items.push({
        label: routeLabels[segment] || segment.replace(/-/g, ' '),
        href: path
      });
    }

    return items;
  });
</script>

<nav aria-label="Breadcrumb">
  <ol class="breadcrumbs">
    {#each crumbs() as crumb, i}
      <li>
        {#if i < crumbs().length - 1}
          <a href={crumb.href}>{crumb.label}</a>
          <span aria-hidden="true">/</span>
        {:else}
          <span aria-current="page">{crumb.label}</span>
        {/if}
      </li>
    {/each}
  </ol>
</nav>

<style>
  .breadcrumbs {
    display: flex;
    gap: 0.5rem;
    list-style: none;
    padding: 0;
    font-size: 0.875rem;
  }
</style>
```

Related Content Links

```
<!-- src/lib/components/RelatedPosts.svelte -->
<script>
  let { currentSlug, posts, maxItems = 3 } = $props();

  const related = $derived(
    posts
      .filter((p) => p.slug !== currentSlug)
      .slice(0, maxItems)
  );
</script>

{#if related.length > 0}
  <aside aria-label="Related articles">
    <h2>Related Articles</h2>
    <ul>
```

```
      {#each related as post}
        <li>
          <a href="/blog/{post.slug}">
            <article>
              <h3>{post.title}</h3>
              <p>{post.excerpt}</p>
              <time datetime={post.date}>{post.formattedDate}</time>
            </article>
          </a>
        </li>
      {/each}
    </ul>
  </aside>
{/if}
```

Footer Link Architecture

```
<!-- src/lib/components/Footer.svelte -->
<script>
  const footerSections = [
    {
      title: 'Product',
      links: [
        { href: '/features', label: 'Features' },
        { href: '/pricing', label: 'Pricing' },
        { href: '/changelog', label: 'Changelog' },
        { href: '/roadmap', label: 'Roadmap' }
      ]
    },
    {
      title: 'Resources',
      links: [
        { href: '/blog', label: 'Blog' },
        { href: '/docs', label: 'Documentation' },
        { href: '/tutorials', label: 'Tutorials' },
        { href: '/support', label: 'Support' }
      ]
    },
    {
      title: 'Company',
      links: [
        { href: '/about', label: 'About' },
        { href: '/contact', label: 'Contact' },
        { href: '/careers', label: 'Careers' },
        { href: '/press', label: 'Press' }
      ]
    },
    {
      title: 'Legal',
      links: [
        { href: '/privacy', label: 'Privacy Policy' },
        { href: '/terms', label: 'Terms of Service' },
        { href: '/cookies', label: 'Cookie Policy' }
      ]
    }
  ];
</script>

<footer>
  <nav aria-label="Footer navigation">
    {#each footerSections as section}
      <div>
        <h3>{section.title}</h3>
        <ul>
          {#each section.links as link}
            <li><a href={link.href}>{link.label}</a></li>
          {/each}
        </ul>
      </div>
    {/each}
  </nav>
</footer>
```

Programmatic Internal Link Utility

```
// src/lib/utils/links.js

// Ensure all internal links use consistent format
export function internalLink(path) {
  // Remove trailing slash (except for root)
  if (path !== '/' && path.endsWith('/')) {
    return path.slice(0, -1);
  }
  return path;
}

// Generate anchor link for heading
export function slugify(text) {
  return text
    .toLowerCase()
    .replace(/[^a-z0-9]+/g, '-')
    .replace(/(^-|-$)/g, '');
}

// Table of contents generator
export function generateTOC(headings) {
  return headings.map((h) => ({
    text: h.text,
    id: slugify(h.text),
    level: h.level,
    href: `#${slugify(h.text)}`
  }));
}
```

Chapter 10: Content SEO

Heading Hierarchy

```html
<!-- CORRECT heading hierarchy -->
<h1>Main Page Title (only one per page)</h1>

<section>
  <h2>First Major Section</h2>
  <p>Content...</p>

  <h3>Subsection</h3>
  <p>Content...</p>

  <h3>Another Subsection</h3>
  <p>Content...</p>
</section>

<section>
  <h2>Second Major Section</h2>
  <p>Content...</p>
</section>

<!-- WRONG - skipping levels -->
<h1>Title</h1>
<h3>Subsection (skipped h2!)</h3>
<h5>Deep section (skipped h4!)</h5>
```

Semantic HTML for SEO

```html
<article>
  <header>
    <h1>{post.title}</h1>
    <p class="meta">
      By <a href="/author/{post.author.slug}" rel="author">{post.author.name}</a>
      on <time datetime={post.date}>{post.formattedDate}</time>
    </p>
  </header>

  <section class="content">
    {@html post.content}
  </section>

  <footer>
    <nav aria-label="Post tags">
      <ul>
        {#each post.tags as tag}
          <li>
            <a href="/blog/tag/{tag}" rel="tag">{tag}</a>
          </li>
        {/each}
      </ul>
    </nav>
  </footer>
</article>

<aside aria-label="Related articles">
  <h2>You Might Also Like</h2>
  <!-- Related posts -->
</aside>
```

Image SEO

```
<!-- Descriptive alt text is critical -->
<!-- BAD -->
<img src="/photo.jpg" alt="image" />
<img src="/photo.jpg" alt="" />
<img src="/photo.jpg" />

<!-- GOOD -->
<img
  src="/team-meeting-2026.webp"
  alt="ShipForge team collaborating on product design in our Stockholm office"
  width="800"
  height="450"
  loading="lazy"
/>

<!-- Decorative images: empty alt is correct -->
<img src="/decorative-wave.svg" alt="" role="presentation" />
```

URL Structure Best Practices

```
GOOD URLs:
  /blog/sveltekit-seo-guide
  /pricing
  /features/analytics
  /blog/tag/sveltekit

BAD URLs:
  /blog/post?id=123
  /p/12345
  /features/analytics#section-2?ref=nav
  /Blog/SvelteKit-SEO-Guide  (inconsistent casing)
```

```
// src/routes/blog/[slug]/+page.server.js
import { redirect } from '@sveltejs/kit';

export function load({ params }) {
  // Enforce lowercase URLs
  if (params.slug !== params.slug.toLowerCase()) {
    redirect(301, `/blog/${params.slug.toLowerCase()}`);
  }

  // Fetch and return post data
  return { post: getPost(params.slug) };
}
```

Content Freshness Signals

```
<!-- Show last updated date for content freshness -->
<script>
  let { data } = $props();
</script>

<svelte:head>
  <meta property="article:published_time" content={data.post.publishedAt} />
  <meta property="article:modified_time" content={data.post.updatedAt} />
</svelte:head>

<article>
  <h1>{data.post.title}</h1>
  <div class="dates">
    <time datetime={data.post.publishedAt}>
      Published: {data.post.publishedFormatted}
    </time>
    {#if data.post.updatedAt !== data.post.publishedAt}
      <time datetime={data.post.updatedAt}>
        Updated: {data.post.updatedFormatted}
      </time>
    {/if}
  </div>
</article>
```

Keyword Optimization (Without Stuffing)

```
Page: /blog/sveltekit-seo-guide

Primary keyword: "SvelteKit SEO"
Secondary keywords: "SvelteKit search optimization", "Svelte meta tags"

Placement checklist:
  [x] Title tag: "SvelteKit SEO Guide 2026 | ShipForge"
  [x] H1: "The Complete SvelteKit SEO Guide for 2026"
  [x] First paragraph: mention naturally
  [x] URL: /blog/sveltekit-seo-guide
  [x] Meta description: include primary keyword
  [x] Image alt text: where relevant
  [x] Subheadings: use variations naturally
```

Chapter 11: Mobile SEO

Mobile-First Indexing

Google indexes the mobile version of your site first. Your mobile experience IS your SEO.

Viewport Configuration

```html
<!-- src/app.html -->
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
```

Responsive Design Patterns

```css
/* Mobile-first CSS approach */

/* Base: Mobile (< 640px) */
.grid {
  display: grid;
  grid-template-columns: 1fr;
  gap: 1rem;
  padding: 1rem;
}

/* Tablet (>= 640px) */
@media (min-width: 640px) {
  .grid {
    grid-template-columns: repeat(2, 1fr);
    gap: 1.5rem;
    padding: 1.5rem;
  }
}

/* Desktop (>= 1024px) */
@media (min-width: 1024px) {
  .grid {
    grid-template-columns: repeat(3, 1fr);
    gap: 2rem;
    padding: 2rem;
  }
}

/* Large desktop (>= 1280px) */
@media (min-width: 1280px) {
  .grid {
    max-width: 1200px;
    margin: 0 auto;
  }
}
```

Touch Target Sizing

```css
/* Minimum touch target: 44x44px (WCAG) / 48x48px (Google recommendation) */
.button,
.nav-link,
.interactive-element {
  min-height: 48px;
  min-width: 48px;
  padding: 12px 24px;
}

/* Ensure adequate spacing between touch targets */
.nav-list {
  display: flex;
  gap: 8px; /* Minimum 8px between targets */
}
```

```
/* Links within text need larger tap areas */
.content a {
  padding: 4px 0;
  display: inline-block;
}
```

Mobile Performance

```
<script>
  import { browser } from '$app/environment';

  // Detect connection quality
  let isSlowConnection = $state(false);

  $effect(() => {
    if (!browser) return;

    const connection = navigator.connection || navigator.mozConnection;
    if (connection) {
      isSlowConnection = connection.effectiveType === '2g'
        || connection.effectiveType === 'slow-2g'
        || connection.saveData;
    }
  });
</script>

<!-- Serve lighter content on slow connections -->
{#if isSlowConnection}
  <img src="/hero-mobile-low.webp" alt="Hero" width="400" height="225" />
{:else}
  <img src="/hero-mobile.webp" alt="Hero" width="800" height="450" />
{/if}
```

Mobile-Specific Meta Tags

```
<svelte:head>
  <!-- Prevent phone number auto-detection -->
  <meta name="format-detection" content="telephone=no" />

  <!-- iOS web app capable -->
  <meta name="apple-mobile-web-app-capable" content="yes" />
  <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent" />
  <meta name="apple-mobile-web-app-title" content="ShipForge" />

  <!-- Theme color for browser chrome -->
  <meta name="theme-color" content="#6366f1" media="(prefers-color-scheme: light)" />
  <meta name="theme-color" content="#1e1b4b" media="(prefers-color-scheme: dark)" />
</svelte:head>
```

Mobile Usability Checklist

```
Content:
  [ ] Text readable without zooming (16px+ base font)
  [ ] Content fits viewport width (no horizontal scroll)
  [ ] Line length < 80 characters on mobile
  [ ] Adequate contrast ratios (4.5:1 for text)

Interaction:
  [ ] Touch targets >= 48x48px
  [ ] Touch targets spaced >= 8px apart
  [ ] No hover-only interactions
  [ ] Forms have appropriate input types

Performance:
  [ ] Images are responsive (srcset/sizes)
  [ ] Fonts are subset and swap-loaded
  [ ] JavaScript is code-split per route
  [ ] Critical CSS is inlined

UX:
  [ ] No intrusive interstitials
  [ ] Navigation is accessible on mobile
  [ ] Forms are mobile-friendly
  [ ] Back button works as expected
```

Chapter 12: Monitoring & Analytics

Google Search Console Integration

```
// src/routes/+layout.svelte or src/app.html
// Add verification meta tag
```

```html
<!-- src/app.html -->
<head>
  <meta name="google-site-verification" content="YOUR_VERIFICATION_CODE" />
</head>
```

SEO Health Check API

```js
// src/routes/api/seo-check/+server.js
import { SITE_URL } from '$lib/config';

export async function GET() {
  const checks = {
    timestamp: new Date().toISOString(),
    site: SITE_URL,
    checks: {}
  };

  // Check robots.txt
  try {
    const robots = await fetch(`${SITE_URL}/robots.txt`);
    checks.checks.robotsTxt = {
      status: robots.ok ? 'pass' : 'fail',
      statusCode: robots.status
    };
  } catch {
    checks.checks.robotsTxt = { status: 'fail', error: 'unreachable' };
  }

  // Check sitemap
  try {
    const sitemap = await fetch(`${SITE_URL}/sitemap.xml`);
    const text = await sitemap.text();
    const urlCount = (text.match(/<url>/g) || []).length;
    checks.checks.sitemap = {
      status: sitemap.ok ? 'pass' : 'fail',
      urlCount
    };
  } catch {
    checks.checks.sitemap = { status: 'fail', error: 'unreachable' };
  }

  // Check key pages for meta tags
  const keyPages = ['/', '/about', '/pricing', '/blog'];
  checks.checks.pages = {};

  for (const pagePath of keyPages) {
    try {
      const res = await fetch(`${SITE_URL}${pagePath}`);
      const html = await res.text();

      checks.checks.pages[pagePath] = {
        status: res.ok ? 'pass' : 'fail',
        hasTitle: html.includes('<title>'),
        hasDescription: html.includes('name="description"'),
        hasOgTitle: html.includes('property="og:title"'),
        hasCanonical: html.includes('rel="canonical"'),
        hasH1: html.includes('<h1')
      };
    } catch {
      checks.checks.pages[pagePath] = { status: 'fail', error: 'unreachable' };
    }
  }

  return new Response(JSON.stringify(checks, null, 2), {
    headers: { 'Content-Type': 'application/json' }
  });
}
```

Analytics Setup (Privacy-First)

```
<!-- src/lib/components/Analytics.svelte -->
<script>
  import { browser } from '$app/environment';
  import { page } from '$app/stores';
  import { afterNavigate } from '$app/navigation';
  let { measurementId = '' } = $props();

  $effect(() => {
    if (!browser || !measurementId) return;

    // Load analytics script
    const script = document.createElement('script');
    script.src = `https://www.googletagmanager.com/gtag/js?id=${measurementId}`;
    script.async = true;
    document.head.appendChild(script);

    window.dataLayer = window.dataLayer || [];
    function gtag() { window.dataLayer.push(arguments); }
    gtag('js', new Date());
    gtag('config', measurementId, {
      send_page_view: false // We'll track manually for SPA
    });

    window.gtag = gtag;
  });

  // Track page views on SPA navigation
  afterNavigate(() => {
    if (browser && window.gtag) {
      window.gtag('event', 'page_view', {
        page_path: $page.url.pathname,
        page_title: document.title
      });
    }
  });
</script>
```

SEO Monitoring Checklist

```
## Weekly SEO Monitoring

### Google Search Console
- [ ] Check for crawl errors
- [ ] Review indexing status
- [ ] Check Core Web Vitals report
- [ ] Review search performance (clicks, impressions, CTR, position)
- [ ] Check for manual actions or security issues

### Technical Health
- [ ] Run Lighthouse audit on key pages
- [ ] Verify sitemap is up to date
- [ ] Check for broken links (404s)
- [ ] Verify structured data validity
- [ ] Test mobile usability

### Content Performance
- [ ] Track top-performing pages
- [ ] Identify pages with declining traffic
- [ ] Review new keywords ranking
- [ ] Check content freshness (update stale content)
```

```
### Monthly Deep Dive
- [ ] Full site crawl audit
- [ ] Competitor analysis
- [ ] Backlink profile review
- [ ] Content gap analysis
- [ ] Page speed regression test
```

Automated SEO Testing in CI

```
// tests/seo.test.js
import { expect, test } from '@playwright/test';

const PAGES = ['/', '/about', '/pricing', '/blog', '/features'];

for (const pagePath of PAGES) {
  test(`SEO: ${pagePath} has required meta tags`, async ({ page }) => {
    await page.goto(pagePath);

    // Title exists and is reasonable length
    const title = await page.title();
    expect(title.length).toBeGreaterThan(10);
    expect(title.length).toBeLessThan(70);

    // Meta description exists
    const description = await page.getAttribute('meta[name="description"]', 'content');
    expect(description).toBeTruthy();
    expect(description.length).toBeGreaterThan(50);
    expect(description.length).toBeLessThan(160);

    // Canonical URL exists
    const canonical = await page.getAttribute('link[rel="canonical"]', 'href');
    expect(canonical).toBeTruthy();

    // OG tags exist
    const ogTitle = await page.getAttribute('meta[property="og:title"]', 'content');
    expect(ogTitle).toBeTruthy();

    const ogDescription = await page.getAttribute('meta[property="og:description"]', 'content');
    expect(ogDescription).toBeTruthy();

    const ogImage = await page.getAttribute('meta[property="og:image"]', 'content');
    expect(ogImage).toBeTruthy();

    // Only one H1
    const h1Count = await page.locator('h1').count();
    expect(h1Count).toBe(1);

    // No broken images
    const images = await page.locator('img').all();
    for (const img of images) {
      const src = await img.getAttribute('src');
      if (src && !src.startsWith('data:')) {
        const alt = await img.getAttribute('alt');
        expect(alt).not.toBeNull(); // All images must have alt
      }
    }
  });

  test(`SEO: ${pagePath} passes Core Web Vitals thresholds`, async ({ page }) => {
    await page.goto(pagePath);

    // Check for CLS-causing issues
    const imagesWithoutDimensions = await
page.locator('img:not([width]):not([style*="width"])').count();
    expect(imagesWithoutDimensions).toBe(0);

    // Check viewport meta
    const viewport = await page.getAttribute('meta[name="viewport"]', 'content');
    expect(viewport).toContain('width=device-width');
  });
}

test('SEO: sitemap.xml is valid', async ({ request }) => {
  const response = await request.get('/sitemap.xml');
  expect(response.ok()).toBeTruthy();
  expect(response.headers()['content-type']).toContain('xml');

  const body = await response.text();
  expect(body).toContain('<urlset');
  expect(body).toContain('<url>');
  expect(body).toContain('<loc>');
});

test('SEO: robots.txt is valid', async ({ request }) => {
  const response = await request.get('/robots.txt');
  expect(response.ok()).toBeTruthy();

  const body = await response.text();
  expect(body).toContain('User-agent');
  expect(body).toContain('Sitemap');
});
```

Summary

Chapter	Key Implementation
1. Fundamentals	SEO is architecture, not an afterthought
2. Technical Basics	robots.txt, sitemap.xml, canonicals
3. SvelteKit SEO	SSR + prerendering = full crawlability
4. Meta Tags	Reusable SEO component for every page
5. Open Graph	OG + Twitter cards for social sharing
6. Structured Data	JSON-LD for rich results in SERPs
7. Core Web Vitals	LCP, INP, CLS optimization
8. Page Speed	Images, code splitting, caching

9. Internal Links	Navigation, breadcrumbs, related content
10. Content SEO	Semantic HTML, heading hierarchy, URLs
11. Mobile SEO	Responsive, touch-friendly, fast
12. Monitoring	Search Console, automated testing, CI

End of SvelteKit SEO 2026 Implementation Guide