

Architecture of the Aigrama Technical Web Blog

Author: Pankaj Sonani from aigrama.net

Publication Date: 2026-05-18

This tutorial explains how **aigrama.net** is structured as a static technical blog built with **Astro 6.x**, **MDX**, **vanilla CSS**, and a small amount of client-side JavaScript. It also explains how the site integrates with **Google Analytics 4**, **Giscus (GitHub Discussions)**, **Mermaid.js**, and **GoDaddy shared hosting**.

Tech Stack

| Layer | Technology |

|---|---|

| Framework | Astro 6.x (<https://astro.build>) — SSG, TypeScript strict mode |

| Styling | Vanilla CSS + CSS custom properties |

| Diagrams | Mermaid.js (<https://mermaid.js.org>) — client-side, via `<MermaidDiagram>` component |

| Syntax highlighting | Shiki (<https://shiki.style>) (built into Astro) — github-dark-dimmed / github-light |

| Fonts | Inter (body/headings), JetBrains Mono (code) via [@fontsource](#) |

| Comments | Giscus (<https://giscus.app>) — GitHub Discussions |

| RSS | [@astrojs/rss](#) |

| Sitemap | [@astrojs/sitemap](#) |

| MDX support | [@astrojs/mdx](#) — for posts with inline Astro components |

| Hosting | GoDaddy Shared Hosting — dist/ uploaded manually after CI build |

Project Structure

```
aigrama/
├── .github/
│   └── workflows/build.yml # CI: builds site, zips dist/, uploads artifact
```

```

■   ■■■ copilot-instructions.md    # Copilot workspace context
■   ■■■ CHANGELOG.md
■■■ docs/                          # ← this folder: project documentation
■■■ public/
■   ■■■ images/                   # Static images (hero images, og images)
■■■ src/
■   ■■■ components/              # Reusable Astro components
■   ■   ■■■ Breadcrumb.astro
■   ■   ■■■ Giscus.astro
■   ■   ■■■ Header.astro
■   ■   ■■■ Footer.astro
■   ■   ■■■ LikeButton.astro
■   ■   ■■■ MermaidDiagram.astro
■   ■   ■■■ ShareButtons.astro
■   ■   ■■■ ThemeToggle.astro
■   ■   ■■■ TOC.astro
■   ■■■ layouts/
■   ■   ■■■ BaseLayout.astro      # HTML shell, OG meta, theme init script
■   ■   ■■■ PostLayout.astro     # Two-column post layout (TOC + content)
■   ■■■ pages/
■   ■   ■■■ index.astro           # Home page with post cards
■   ■   ■■■ blog/[...slug].astro # Individual post pages
■   ■   ■■■ category/[category].astro
■   ■   ■■■ tags/[tag].astro
■   ■   ■■■ search.astro          # Full-site static search
■   ■   ■■■ rss.xml.ts
■   ■■■ content/
■   ■   ■■■ blog/                 # Markdown / MDX posts go here
■   ■■■ content.config.ts         # Zod schema for frontmatter validation
■   ■■■ styles/
■       ■■■ global.css            # CSS variables, reset, typography
■       ■■■ post.css              # Article/prose styles, code blocks
■       ■■■ components.css        # Header, footer, TOC, share buttons
■■■ astro.config.mjs
■■■ package.json
■■■ tsconfig.json

```

What this repository is made of

This repository is primarily organized around a few important areas:

- ? src/content/blog/ stores posts, grouped into category folders like architecture/, java/, and python/
- ? src/layouts/ contains page layouts such as BaseLayout.astro and PostLayout.astro
- ? src/components/ contains reusable UI pieces such as Mermaid, Giscus, TOC, and sharing-related components
- ? src/pages/ defines Astro routes like the home page, blog slug pages, category pages, tag pages, profile page, and RSS feed
- ? src/styles/ contains global, post, and component CSS
- ? public/ contains static assets, including the production Giscus theme files

The project is configured as a **static site** in `astro.config.mjs`, with the site URL set to `https://aigrama.net`, and it uses the Astro integrations for **MDX** and **sitemap generation**.

High-level architecture

<MermaidDiagram code={flowchart TD

A[Author writes .md or .mdx post in `src/content/blog/category/`] --> B[Astro content collection loader]\n B --> C[Zod schema validation in `src/content.config.ts`]\n C --> D[Layouts and components render the article]\n D --> E[Astro static build]\n E --> F[`dist/` output]\n F --> G[Upload to GoDaddy shared hosting]\n G --> H[Readers load `aigrama.net` in browser]\n H --> I[Google Analytics 4]\n H --> J[Giscus widget]\n J --> K[GitHub Discussions in `sonani-pankaj/aigrama-comments`]\n H --> L[Static assets from `public/` and generated CSS/JS]

} />

How content moves through the system

1. Authoring

Posts live under `src/content/blog/**` and the repository expects them to be placed under category-specific subfolders such as:

? `src/content/blog/architecture/`

? `src/content/blog/java/`

? `src/content/blog/python/`

Each post must include frontmatter validated by `src/content.config.ts`. Required fields include:

? title

? description

? pubDate

? draft

? tags

? category

The allowed category values are:

? Java

? Python

? JavaScript

- ? SQL
- ? DevOps
- ? Architecture
- ? AI/ML

If a post needs Mermaid diagrams or Astro component imports, it should use the .mdx extension rather than .md.

2. Content collection loading and validation

The file `src/content.config.ts` defines the blog collection using Astro content collections with a `glob()` loader for `**/*.md,mdx` under `./src/content/blog`.

That means Astro automatically:

- ? discovers all blog post files
- ? validates their frontmatter using Zod
- ? makes the entries queryable in pages and layouts

This is useful because it prevents malformed posts from quietly entering production.

3. Layout composition

After Astro loads a post, the site renders it using shared layouts.

According to the repo instructions:

- ? `BaseLayout.astro` provides the HTML shell, Open Graph metadata, Google Analytics script, and theme initialization
- ? `PostLayout.astro` provides the two-column article layout with TOC and content

This separation keeps shared responsibilities centralized:

- ? SEO metadata lives in one place
- ? analytics lives in one place
- ? theme setup lives in one place
- ? article presentation stays consistent across every post

4. Styling and code rendering

The styling system is intentionally simple:

- ? src/styles/global.css defines CSS variables, reset styles, and typography
- ? src/styles/post.css handles prose and code block styling
- ? src/styles/components.css handles shared component styling

The site uses **vanilla CSS with custom properties**, not Tailwind or Bootstrap.

For code blocks, Astro is configured to use **Shiki** syntax highlighting with:

- ? github-dark-dimmed for dark mode
- ? github-light for light mode

Supported languages in astro.config.mjs include Java, Python, JavaScript, TypeScript, SQL, Markdown, Bash, JSON, and YAML.

5. Static build and deployment

The build is fully static:

- ? npm run build runs astro build
- ? Astro outputs the final site into dist/
- ? GitHub Actions builds the site and produces an artifact ZIP
- ? the dist/ contents are then uploaded to **GoDaddy shared hosting** under public_html/

This means the site does not depend on a Node.js server at runtime. Once built, it is just static HTML, CSS, and JavaScript served by hosting.

How the site talks to other tools

Google Analytics 4

Google Analytics 4 is wired into BaseLayout.astro according to the repository instructions.

What it does

The site tracks:

- ? page_view
- ? share

? like_post

The share event includes a method parameter, and like_post includes the post slug and page path.

How it works

The browser loads a page from the static site first. After that, the GA4 script sends analytics events to Google.

So the integration is **client-side and non-blocking** relative to the article itself.

```
<MermaidDiagram code={sequenceDiagram
```

```
participant U as Reader Browser
```

```
participant S as aigrama.net static site
```

```
participant GA as Google Analytics 4
```

```
U->>S: Request article page
```

```
S-->>U: Return HTML, CSS, JS
```

```
U->>GA: Send page_view and later interaction events
```

```
U->>GA: Send share event on share click
```

```
U->>GA: Send like_post event on like click
```

```
} />
```

Why this matters architecturally

Google Analytics is not needed to render the post, so if analytics fails, the content still works. That is a good separation of concerns.

Giscus and GitHub Discussions

The comment system is implemented with **Giscus** and backed by **GitHub Discussions**.

The repository instructions state that the discussion repo is:

? sonani-pankaj/aigrama-comments

What Giscus does

Giscus embeds a discussion-driven comment widget into each post page. Instead of maintaining a custom comment backend, the site reuses GitHub Discussions for storage, identity, and

moderation.

How it works

- ? 1. A reader opens an article page
- ? 2. The static article renders first
- ? 3. The Giscus component loads client-side
- ? 4. Giscus maps the page to a discussion thread
- ? 5. Comments are fetched from GitHub Discussions
- ? 6. Logged-in GitHub users can reply through the embedded widget

```
<MermaidDiagram code={sequenceDiagram
```

```
participant U as Reader Browser
```

```
participant P as Article Page
```

```
participant G as Giscus Widget
```

```
participant GH as GitHub Discussions
```

```
U->>P: Open article
```

```
P-->>U: Render page and comments container
```

```
U->>G: Initialize Giscus script
```

```
G->>GH: Resolve linked discussion thread
```

```
GH-->>G: Return comments
```

```
G-->>U: Render discussion UI
```

```
} />
```

Theme behavior

The repo has two custom Giscus CSS files in public/:

? /giscus-dark.css

? /giscus-light.css

In development, Giscus uses built-in presets because localhost cannot serve those custom styles to giscus.app due to CORS limitations. In production, it loads the custom CSS themes.

Architectural benefit

This allows the site to remain static while still supporting dynamic comments.

GoDaddy shared hosting and domain responsibilities

GoDaddy plays two roles in this project:

- ? 1. **hosting destination** for the built dist/ files on shared hosting
- ? 2. likely **domain/DNS management** for <https://aigrama.net>

How the hosting flow works

- ? 1. GitHub Actions builds the Astro site
- ? 2. A ZIP artifact is created
- ? 3. The built files are extracted
- ? 4. Files are uploaded to GoDaddy public_html/
- ? 5. GoDaddy serves the static files to visitors

DNS and custom domain flow

When a reader types aigrama.net into a browser:

- ? 1. DNS resolves the domain
- ? 2. The browser reaches the GoDaddy-hosted site
- ? 3. Static assets are returned
- ? 4. Additional client-side services like GA4 and Giscus initialize after page load

```
<MermaidDiagram code={sequenceDiagram
```

```
participant U as Reader Browser
```

```
participant DNS as Domain/DNS
```

```
participant GD as GoDaddy Hosting
```

```
participant SITE as Static Astro build
```

```
U->>DNS: Resolve aigrama.net
```

```
DNS-->>U: Return hosting target
```

```
U->>GD: Request page
```

```
GD-->>U: Return static files from deployed dist/
```

```
} />
```

Why this matters

GoDaddy is not part of the Astro rendering process. It is part of the ****delivery layer****. Astro builds the site; GoDaddy serves it.

Other important built-in subsystems

Mermaid diagrams

The repo includes Mermaid support through a `<MermaidDiagram>` Astro component and the mermaid package.

This allows architecture and flow diagrams to be embedded directly inside .mdx posts. The diagrams are rendered client-side and are theme-aware.

This is especially valuable for a technical blog because system behavior can be documented visually instead of only in prose.

Table of contents

The repository uses a TOC component that receives headings from Astro's content rendering pipeline.

This creates a structured reading experience for longer posts and makes multi-section tutorials easier to navigate.

Share buttons and like button

The project also includes:

- ? a custom share button component
- ? a localStorage-backed like button

These features are interesting because they show a clean separation between:

- ? persistent discussion content stored externally in GitHub Discussions
- ? lightweight engagement interactions stored locally in the browser
- ? analytics events forwarded to GA4

A simple mental model

You can think about this blog in layers:

- ? **MD/MDX files** are the source of content
- ? **Astro content collections** validate and load that content
- ? **layouts and components** transform the content into full pages
- ? **Astro build** produces static output
- ? **GoDaddy hosting** serves the output
- ? **Google Analytics** observes behavior
- ? **Giscus** adds a dynamic discussion layer from GitHub Discussions

Why this architecture works well

This repo uses a strong architecture for a technical blog because it is:

- ? **fast**: most pages are static
- ? **maintainable**: posts live in version control
- ? **structured**: frontmatter is schema-validated
- ? **extensible**: features like Mermaid and MDX can be added to posts
- ? **low-ops**: no backend server or CMS database is required

Common troubleshooting map

| Problem | Likely layer |

|---|---|

| Post does not appear | content file, frontmatter, or collection validation |

| Comments do not load | Giscus configuration or GitHub Discussions mapping |

| Analytics data is missing | GA4 measurement/configuration in BaseLayout |

| Domain works inconsistently | DNS or hosting configuration |

| Styling looks wrong | global.css, post.css, components.css, or theme tokens |

| Mermaid does not render | file must be .mdx and import path must be correct |

Final summary

The **Aigrama web blog** is a well-structured Astro static site.

- ? Content lives in categorized Markdown and MDX files
- ? Astro validates, renders, and builds the site
- ? Layouts and components provide consistency
- ? GoDaddy delivers the final static output
- ? Google Analytics measures usage
- ? Giscus adds comments through GitHub Discussions
- ? Mermaid enables architecture visuals inside posts

That combination makes the repository a practical and modern setup for publishing technical tutorials with strong performance and manageable operational complexity.

PLAN.MD

Code (markdown)

```
# aigrama — Copilot Workspace Instructions
```

```
You are assisting with aigrama.net, a static technical blog for developers built with Astro
```

```
## Project Overview
```

- **Framework**: Astro 6.x, SSG (no server runtime), TypeScript strict mode
- **Hosting**: GoDaddy Shared Hosting — `dist/` folder contents are uploaded manually after CI build
- **Site URL**: <https://aigrama.net>
- **Styling**: Vanilla CSS + CSS custom properties (no Tailwind, no Bootstrap)
- **Fonts**: Inter (body/headings), JetBrains Mono (code) via `@fontsource`
- **Diagrams**: Mermaid.js (client-side, `` component)
- **Comments**: Giscus (GitHub Discussions) — custom CSS themes in `public/`, reactions disabled
- **Syntax Highlighting**: Shiki built into Astro (`github-dark-dimmed` dark, `github-light` light)
- **Analytics**: Google Analytics 4 — page views, share clicks, and like clicks tracked automatically
- **MDX**: `@astrojs/mdx` installed — posts can be `.md` or `.mdx` (MDX required for ``)

```
---
```

```
## File Structure
```

```
src/
```

```
■■■■ components/ # Reusable Astro components
```

```
■■■■ layouts/
```

```
■ ■■■■ BaseLayout.astro # HTML shell, OG meta, GA4 script, theme init
```

```
■ ■■■■ PostLayout.astro # Two-column post layout (TOC + content)
```

```
■■■■ pages/ # File-based routing
```

```
■ ■■■■ index.astro
```

```
■ ■■■■ blog/[...slug].astro
```

- ■■■■ category/[category].astro
- ■■■■ tags/[tag].astro
- ■■■■ profile.astro # Developer profile + contact form (Web3Forms)
- ■■■■ rss.xml.ts
- content/
- ■■■■ blog/ # Posts organised into category subfolders
- ■■■■ java/
- ■■■■ python/
- ■■■■ architecture/
- ■■■■ ... # Add a subfolder per category
- content.config.ts # Zod schema — glob pattern is `**/*.md,mdx`
- styles/
- ■■■■ global.css # CSS variables, reset, typography
- ■■■■ post.css # Article/prose styles, code blocks
- ■■■■ components.css # Header, footer, TOC, share/like buttons
- utils/toc.ts # TOC heading type helpers
- public/
- giscus-dark.css # Custom Giscus dark theme (production only)
- giscus-light.css # Custom Giscus light theme (production only)
-
- ## Content Collection Schema
- Every post lives in a `**category subfolder**` under ``src/content/blog/<category>/your-slug.md``.
- All required frontmatter fields:
-
- title: "Your Post Title" # required — string
- description: "One sentence." # required — string (used for OG + card)
- pubDate: YYYY-MM-DD # required — ISO date
- updatedAt: YYYY-MM-DD # optional
- draft: false # required — set true to exclude from build
- tags: ["tag1", "tag2"] # required — lowercase kebab-case array
- category: Java # required — one of the enum values below
- author: aigrama # optional — defaults to "aigrama"

image: # optional

src: /images/your-image.png

alt: "Descriptive alt text"

****Valid `category` values** (exact casing required):**
`Java` | `Python` | `JavaScript` | `SQL` | `DevOps` | `Architecture` | `AI/ML`

CSS Design Tokens

All theme colors are CSS custom properties in `src/styles/global.css`.
Dark mode is the default; light mode uses `[data-theme="light"]` selector.

Token	Dark	Light
--bg-primary	#0f172a	ffffff
--bg-secondary	#1e293b	#f1f5f9
--text-primary	#f8faff	#0f172a
--text-secondary	#94a3b8	#475569
--accent	#38bdf8	#0284c7
--code-bg	#0d1117	#f6f8fa

Always use these tokens in new CSS – never hardcode hex colors.

Component APIs

`Renders a Mermaid diagram string client-side. Theme-aware. **\*\*Only works in `.mdx` files.****

Import path depends on how deep the post file is:

{/* From src/content/blog/<category>/post.mdx — one category subfolder deep */}

`Icon-only heart button with per-post like count stored in `localStorage`. Fires a `like_post` GA4 event

<LikeButton slug={post.id} />

`Icon-only social share buttons: LinkedIn, Facebook, YouTube, Copy Link. Fires a `share` GA4 event

`GitHub Discussions comments. Repo: `sonani-pankaj/aigrama-comments`. Reactions are disabled (site policy)
- ****Dev (localhost)**:** uses built-in `dark\_dimmed` / `light` presets (CORS prevents loading local CSS)
- ****Production**:** loads custom CSS from `/giscus-dark.css` or `/giscus-light.css`

`Sticky sidebar TOC. Receives `TocHeading[]` from `render(entry).headings` filtered via `filterHeadings`

`

items: Array<{ label: string; href?: string }>

```
// Last item has no href (current page)
```

```
---
```

```
## Astro 6.x Patterns to Use
```

```
// Content collection — render API
```

```
const posts = await getCollection('blog', ({ data }) => !data.draft);
```

```
const { Content, headings } = await render(post);
```

```
// Content config uses glob() loader
```

```
---
```

```
## Coding Conventions
```

- **TypeScript**: strict mode – no `any`, no non-null assertions without comment
- **CSS**: BEM-inspired class names (`.component__element--modifier`)
- **Components**: Props interface named `Props` in each `.astro` file
- **Scripts**: All client-side scripts inside `<script>` tags in `.astro` files (bundled by Vite)
- **No frameworks**: No React, Vue, Svelte – Astro components only
- **Accessibility**: Every interactive element needs `aria-label` or visible label
- **Security**: No `innerHTML` with unsanitized content; Mermaid uses `securityLevel: 'strict'`

```
---
```

```
## Authoring a New Post
```

1. Create `src/content/blog/<category>/your-post-slug.md` (use the matching category subfolder, e.g. `astro`)
2. Use `.mdx` extension instead of `.md` if the post needs `<MermaidDiagram>` or other component in `src/components`
3. Fill in all required frontmatter fields (see schema above)
4. Set `draft: true` while writing, flip to `false` when ready
5. Use fenced code blocks with language tags: ````java`, ````python`, ````sql`
6. For Mermaid diagrams in `.mdx` posts, import with three `../` levels:
7. Update `CHANGELOG.md` under `[Unreleased]` when publishing

```
---
```

```
## Analytics (Google Analytics 4)
```

GA4 is wired into `BaseLayout.astro`. Replace the `G-XXXXXXXXXX` placeholder with the real Measurement ID.

Events fired automatically:

Event name	Trigger	Key params
<code>page_view</code>	Every page load (built-in GA4)	<code>page_path</code>
<code>share</code>	Any share button click	<code>method</code> (<code>linkedin</code> / <code>facebook</code> / <code>youtube</code> / <code>copy_link</code>), <code>item_title</code>
<code>like_post</code>	Like button clicked (like only, not unlike)	<code>post_slug</code> , <code>page_path</code>

Like counts are stored in `localStorage` – they are per-browser only and not aggregated server-side.

```
---
```

```
## Deployment
```

1. Push to `main` – GitHub Actions builds the site and creates `aigrama-dist-{sha}.zip`
2. Download the ZIP from the Actions artifacts tab
3. Extract and upload contents to GoDaddy `public\_html/` via FTP/cPanel File Manager

Citation

Pankaj Sonani from aigrama.net. "Architecture of the Aigrama Technical Web Blog." aigrama.net, 2026-05-18.

URL: <https://aigrama.net/blog/architecture/aigrama-webblog-architecture-tutorial/>

PDF: <https://aigrama.net/papers/architecture-of-the-aigrama-technical-web-blog.pdf>