Guía de Arquitectura del Proyecto

Stack: Astro + Tailwind + Vue (Islas Interactivas)

Fecha de generación: 20 de febrero de 2026

1. Objetivo de esta guía

Esta documentación explica cómo está montado tu proyecto actual y cómo trabajan juntas las tres piezas clave: Astro, Tailwind y Vue.

Está pensada para que puedas leer archivo por archivo y entender para qué sirve cada uno, cómo se conecta con los demás y dónde tocar cuando quieras cambiar algo.

2. Mapa general del proyecto

- Astro se encarga del enrutado, del render base y de la composición de páginas.
- Tailwind se usa para todo el estilo visual mediante clases utilitarias.
- Vue se usa para las islas interactivas (solo donde hay comportamiento dinámico real).
- La estructura principal vive dentro de src/ y se divide en layouts, pages, components y styles.

3. Flujo de render (de arriba a abajo)

Idea clave: todo lo que no necesita JavaScript en cliente se queda en Astro. Lo que sí necesita interacción viva se pasa a Vue.

- Cada página (src/pages/*.astro) importa un layout base y varios componentes.
- El layout base define HTML global, meta tags y estilos compartidos.
- Los componentes .astro renderizan contenido estático o semiestático.
- Los componentes .vue se hidratan en cliente con directivas client:* y funcionan como islas interactivas.

4. Configuración del stack

`astro.config.mjs`

Qué hace: Activa integraciones del proyecto. Aquí se registra @astrojs/vue y el plugin de Tailwind para Vite.

Cómo funciona: Astro usa este archivo al arrancar dev/build para saber cómo transformar archivos .astro y .vue.

`package.json`

Qué hace: Define scripts del proyecto y dependencias principales (astro, tailwindcss, @tailwindcss/vite, vue, @astrojs/vue).

Cómo funciona: npm run dev y npm run build usan estos scripts; las dependencias habilitan todo el entorno de trabajo.

5. Layout y estilos globales

`src/layouts/BaseSiteLayout.astro`
Qué hace: Plantilla base reutilizada por todas las páginas. Define estructura HTML global y metadatos SEO básicos.

Cómo funciona: Recibe title y description por props; si no se pasan, usa valores por defecto.

`src/styles/base-theme.css`
Qué hace: Estilos base del proyecto (import de fuentes, import de Tailwind, variables CSS y reglas globales).

Cómo funciona: Se importa desde el layout base para que toda la web comparta tipografías, fondo y comportamiento común.

6. Páginas y su responsabilidad

`src/pages/index.astro`
Qué hace: Página principal. Orquesta HomeHeroSection, HomeProcessSection y HomeTestimonialsSection.

Cómo funciona: Monta la experiencia de inicio usando el layout base y componentes reutilizables.

`src/pages/servicios.astro`
Qué hace: Página de servicios con scroll interno y snap vertical.

Cómo funciona: Renderiza ServicesBentoSections dentro de un main con altura controlada para snap por bloques.

`src/pages/contacto.astro`
Qué hace: Página de contacto con bloque informativo + formulario.

Cómo funciona: Integra la isla Vue ContactScheduleSelectorIsland para elegir horario dentro del formulario.

7. Componentes de estructura visual

`src/components/SiteHeader.astro`
Qué hace: Cabecera global. Incluye navegación desktop, menú móvil y marcado de ruta activa.

Cómo funciona: Compara Astro.url.pathname con cada enlace para aplicar estado activo.

`src/components/SiteFooter.astro`
Qué hace: Pie global con enlaces agrupados y copyright.

Cómo funciona: Render dinámico de grupos de links (navegación, legal y comunidad).

`src/components/HomeHeroSection.astro`
Qué hace: Primer bloque del inicio con propuesta de valor y CTAs principales.

Cómo funciona: Usa diseño estático optimizado sin JavaScript cliente.

`src/components/HomeProcessSection.astro`
Qué hace: Bloque que explica la metodología de trabajo en 4 pasos.

Cómo funciona: Crea tarjetas por mapeo de un array local processSteps.

`src/components/HomeTestimonialsSection.astro`
Qué hace: Prueba social con testimonios en tarjetas.

Cómo funciona: Mismo patrón de render por mapeo de datos con estructura limpia y reutilizable.

8. Servicios: Astro + isla Vue

Usar client:visible aquí es buena práctica: no cargas la lógica Vue de tarjetas hasta que esa parte entra en pantalla.

`src/components/ServicesBentoSections.astro`
Qué hace: Componente contenedor de secciones de servicio. Define contenido, tonos de fondo y estructura snap.

Cómo funciona: Por cada bloque de servicio pasa las cards a ServiceCardsBentoIsland con client:visible.

`src/components/ServiceCardsBentoIsland.vue`
Qué hace: Isla Vue que gestiona la interacción de las tarjetas bento.

Cómo funciona: Desktop: hover/focus por CSS. Móvil: click/tap para abrir/cerrar tarjeta. Incluye soporte Enter/Espacio/Escape y control de viewport.

9. Contacto: isla Vue del selector de horario

Aquí usas client:load porque la interacción debe estar disponible nada más entrar en la página de contacto.

`src/components/ContactScheduleSelectorIsland.vue`
Qué hace: Isla Vue para mostrar panel flotante de horarios y seleccionar uno.

Cómo funciona: Controla estado del panel, slot activo y valor final en un input hidden para que viaje con el submit del formulario.

10. Dónde tocar según lo que quieras cambiar

- Cambiar copy de inicio: HomeHeroSection.astro, HomeProcessSection.astro, HomeTestimonialsSection.astro.
- Cambiar estructura visual global: BaseSiteLayout.astro y base-theme.css.
- Cambiar navegación o enlaces globales: SiteHeader.astro y SiteFooter.astro.
- Cambiar servicios (bloques y contenido): ServicesBentoSections.astro.
- Cambiar comportamiento interactivo de cards: ServiceCardsBentoIsland.vue.
- Cambiar selector de horario: ContactScheduleSelectorIsland.vue.

11. Conceptos clave de Astro, Tailwind y Vue en este proyecto

- Astro te da páginas rápidas porque lo estático se renderiza sin enviar JS innecesario.
- Tailwind te permite construir UI consistente sin CSS tradicional por componente.
- Vue se usa como isla solo donde hay estado/interacción real en cliente.
- La combinación ideal: Astro para estructura, Tailwind para diseño, Vue para interactividad puntual.

12. Siguiente nivel recomendado

Con esto ya tienes una base profesional: estructura clara, diseño consistente y uso correcto de islas interactivas.

- Añadir TypeScript estricto en componentes Vue para reforzar tipos de props y eventos.
- Separar datos de servicios/testimonios en archivos data/ para escalabilidad.
- Añadir pruebas E2E básicas (por ejemplo navegación y selector de horario).

- Introducir validación del formulario de contacto y envío real a backend/API.

Fin de la guía. Puedes usar el .md generado para ampliarla fácilmente.