



innusual

together in technology

25-04-24

Astro

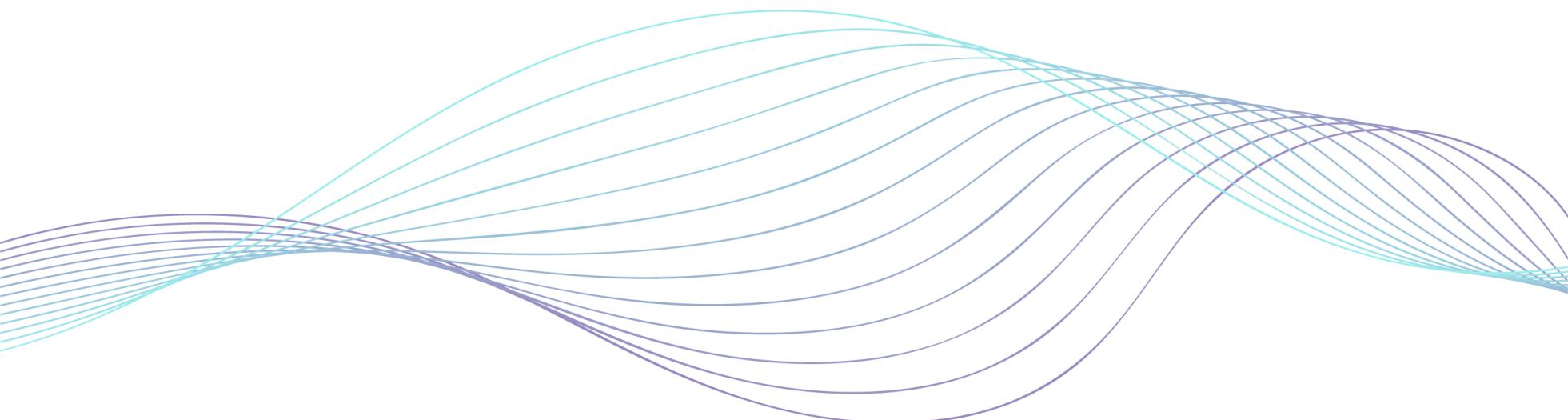
GCD JAVASCRIPT



Índice



- [**<1> Introducción**](#)
- [**<2> Fundamentos de Astro**](#)
- [**<3> Funcionalidades avanzadas**](#)
- [**<4> Calidad y rendimiento**](#)



<1> Introducción

Objetivos del Webinar

- **🚀 CONTROL GENERAL.** Controla las bases para crear sitios web modernos y dinámicos.
- **🌙 ENFOQUES POSIBLES** para construir diferentes proyectos.
- **🐎 COMPARATIVA** con otros Frameworks populares.
Astro no es solo una opción más.
- **帷 DEMOSTRACIÓN** de uso con SSG, Arquitectura insular y API Rest.



¿Qué es Astro?

- **Metaframework** para la crear sitios y aplicaciones web.
- Enfocado en la velocidad y el **rendimiento**.
- SSG, SSR o ambos mundos.
- **MPA** - **multi page application**

Características

Development

RENDIMIENTO: sitios web estáticos por defecto (**SSG**); las páginas se cargan instantáneamente. Utilizando **hidratación de componentes** para optimizar el rendimiento de las partes interactivas.

FLEXIBILIDAD Y FACILIDAD DE USO: desde blogs, portfolios, ecommerce, hasta aplicaciones web complejas, API REST. **Curva de aprendizaje sencilla**, empezando con HTML, CSS y Javascript.

EXPERIENCIA DE DESARROLLO: basado en Vite. Plugins y DevTools que hacen que el desarrollo sea amigable. Compatible con como **React**, **Svelte** o **Vue**. Y completísima **documentación**.

COMUNIDAD: es Open Source y cuenta con una comunidad activa y comprometida, en Discord y Github, y meetup semanal comunitaria “Talking and Doc’ing”. Que además cuenta con sus propios Premios “Astro Community Awards”

Características

UX/UI

EXPERIENCIA DE USUARIO FLUIDA: Astro.js ofrece una experiencia de usuario fluida similar a las aplicaciones web SPA. Podemos utilizar **API Web Transitions** entre páginas y sin tiempos de carga.

ORIENTADO A CONTENIDO Y SEO: es una gran opción, para sitios que necesitan entregar contenido de forma crítica. Genera **HTML limpio y semántico**, lo que lo hace ideal para el SEO. Además, ofrece soporte para meta datos y otras características de SEO importantes.



Historia y antecedentes

ASTRO es un framework relativamente nuevo (lanzamiento oficial en 2021), que sin embargo tiene sus raíces en proyectos y tecnologías como **Snowpack** o **Svelte**

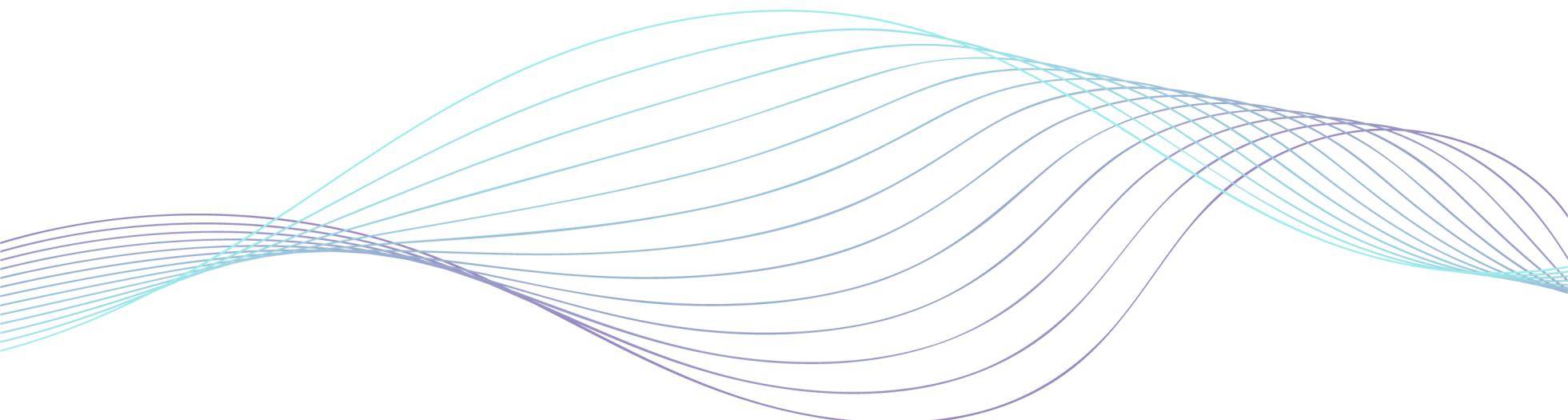
En 2020, **FRED K. SCHOTT**, preocupado por la creciente complejidad y la carga de los frameworks de frontend, creó un generador de sitios estáticos.

El objetivo era colaborativo, esto llevó al concepto **BYOF** (**BRING YOUR OWN FRAMEWORK**), que permite a los desarrolladores integrar código de **React**, **Svelte** o **Vue** beneficiándose de las *ventajas de rendimiento* de un sitio estático.



Casos de uso

-  Aplicaciones y sitios web estáticos y dinámicos
-  Blogs, Ecommerce, Landing Pages
-  API Rest
-  Documentación



<2> Fundamentos de Astro

Fundamentos de Astro



Arquitectura

<1> Arquitectura insular

<2> Rendimiento: Cero Javascript por defecto

<3> Beneficios de la arquitectura

Estructura y elementos esenciales

<1> Dependencias necesarias

<2> Componentes

<3> Páginas

<4> Templates

An aerial photograph of a small, densely forested island in a large body of water. The island is covered in a mix of green and yellow autumn foliage. In the center of the island, there is a white graphic element consisting of a large letter 'A' with a smaller, irregular shape attached to its bottom right side, resembling a puzzle piece.

Arquitectura



Arquitectura insular

Descripción por **Katie Sylor-Miller**, este patrón consiste en renderizar en el servidor HTML e injectar "slots" o piezas en regiones interactivas, hidratar en el cliente widgets autónomos.

Astro utiliza una arquitectura basada en islas, donde la interfaz de usuario permanece **agnostic**a, de nosotros depende integrar otros frameworks.

Un beneficio es que el navegador puede cargar varios componentes **en paralelo** e hidratarlos de **forma aislada**, por lo que el tiempo de carga de una página mejorará significativamente.

Las regiones individuales de la página se vuelven interactivas **sin que sea necesario** cargar primero ningún otro elemento de la página.



Esto se traduce en

Cero JS por defecto

Astro solo envía el Javascript necesario para cada isla, los tiempos de carga de la página en conexiones lentas o inestables seguirán siendo óptimos.

[Si no utilizas js, no mandas js.](#)

Beneficios

-  Diseño modular
-  Desarrollo independiente, equipos pueden trabajar en paralelo.
-  Carga en paralelo e hidratación aislada.
-  Páginas livianas, mejor indexación y mejor SEO

One more thing...

BRING YOUR OWN FRAMEWORK!!!

Astro te permite utilizar el framework que mejor se adapte a las necesidades de tu proyecto.

- Nos da mayor **libertad de elección**.
- **Menor curva de aprendizaje**, puedes usar astro sin tener que aprender nuevas librerías para la UI.
- **Reutilización de código**, si ya has desarrollado o tienes conocimientos de otros frameworks.
- Preparación para el **futuro**, salen frameworks nuevos cada 20 minutos, probablemente Astro te permita integrarte con novedades futuras.

Todo esto se logra con las [HTML templates](#), la [interoperabilidad](#) con otros frameworks y su arquitectura de [islas](#).

Estructura y elementos esenciales



Estructura de un proyecto en Astro



- **ASSETS**: recursos activos procesados por astro, como imágenes, esto no es obligatorio.
- **COMPONENTS**: componentes reutilizables, en Astro, React, Vue..., esta convención tampoco es obligatoria.
- **CONTENT**: esto esta reservado para colecciones de contenido y solo se permiten archivos de tipo, ` `.md, .mdx, .yaml, .json` .
- **LAYOUTS**: componentes de UI compartida, normalmente temas o plantillas, que suelen agrupar componentes comunes a todas las vistas. Esta carpeta es una convención no necesaria.
- **PAGES**: componentes especiales para crear las páginas del proyecto, esto si es necesario, ya que crea las páginas o rutas de tu aplicación.
- **STYLES**: convención para almacenar css.
- **PUBLIC**: esta carpeta se reserva para archivos que no necesitan o no vayan a ser procesados por Astro.
- **PACKAGE.JSON**: archivo para gestionar paquetes en javascript
- **ASTRO.CONFIG.MJS**: archivo que se utiliza para configurar las integraciones con Astro, como la compilación o la configuración del servidor
- **TSCONFIG.JSON**: para configurar typescript si lo utilizas

Components

```
1  ---
2 // Component script
3 import Banner from '../components/Banner.astro';
4 import ReactFuturamaComponent from '../components/ReactFuturamaComponent.jsx';
5 const myCharacterFavoriteFuturama = /* ... */;
6 const { title } = Astro.props;
7 ---
8
9 <!-- HTML comments supported! -->
10 {/* JS comment syntax is also valid! */}
11
12 <Banner />
13 <h1>Hello, world!</h1>
14
15 <p>{title}</p>
16
17 <ReactFuturamaComponent client:visible />
18
19 <ul>
20   {myCharacterFavoriteFuturama.map((data) => <li>{data.name}</li>)}
21 </ul>
22
23 <p class:list={["add", "dynamic", {classNames: true}]}>
```

Componentes - Props y Slots

■ PROPS

Un componente puede definir y aceptar props.

Estarán disponibles para ser utilizadas en el renderizado del maquetado HTML y disponibles en el script del componente de manera global dentro del objeto **Astro.props**.

■ SLOTS

El elemento slot es un espacio reservado para contenido HTML externo, permitiéndote injectar elementos hijos.

Un componente de Astro también puede tener slots con nombre.

Esto te permite compartir elementos HTML únicamente con el nombre correspondiente al slot.

```
1  ---
2  import Header from './Header.astro';
3  import Logo from './Logo.astro';
4  import Footer from './Footer.astro';
5
6  const { titulo } = Astro.props;
7  ---
8  <div id="content-wrapper">
9    <Header />
10   <slot name="after-header" /> <!-- hijos con el atributo `slot="after-header"` van aquí -->
11   <Logo />
12   <h1>{titulo}</h1>
13   <slot /> <!-- hijos sin `slot`, o con el atributo `slot="default"` van aquí -->
14   <Footer />
15   <slot name="after-footer" /> <!-- hijos con el atributo `slot="after-footer"` van aquí -->
16 </div>
17
```

```
1  ---
2  import Wrapper from '../components/Wrapper.astro';
3  ---
4
5  <Wrapper titulo="Página de Fred">
6    
7    <h2>Todo sobre Fred</h2>
8    <p>Aquí veremos cosas sobre Fred.</p>
9    <p slot="after-footer"> Copyright 2022</p>
10 </Wrapper>
11
```

Páginas

- Las **PÁGINAS** son componentes que se encuentran en la subcarpeta `src/pages/`. Ellas son las responsables de manejar el enrutamiento, la carga de datos y el diseño general de cada página HTML de tu proyecto.
- Podemos usar archivos: `.astro`, `.md`, `.mdx`, `.html`, `.js/.ts` (como endpoints).
- Para enlazar entre páginas usamos `<a/>`
- En `.md` / `.mdx`, usamos un template de `.yaml`, y el resto como contenido plano en markdown o podemos importar componentes `jsx` si se trata de un fichero `.mdx`.

```
1  ---
2  import Wrapper from '../components/Wrapper.astro';
3  ---
4
5  <Wrapper titulo="Página de Fred">
6    
7    <h2>Todo sobre Fred</h2>
8    <p>Aquí veremos cosas sobre Fred.</p>
9    <p slot="after-footer"> Copyright 2022</p>
10   </Wrapper>
11
```

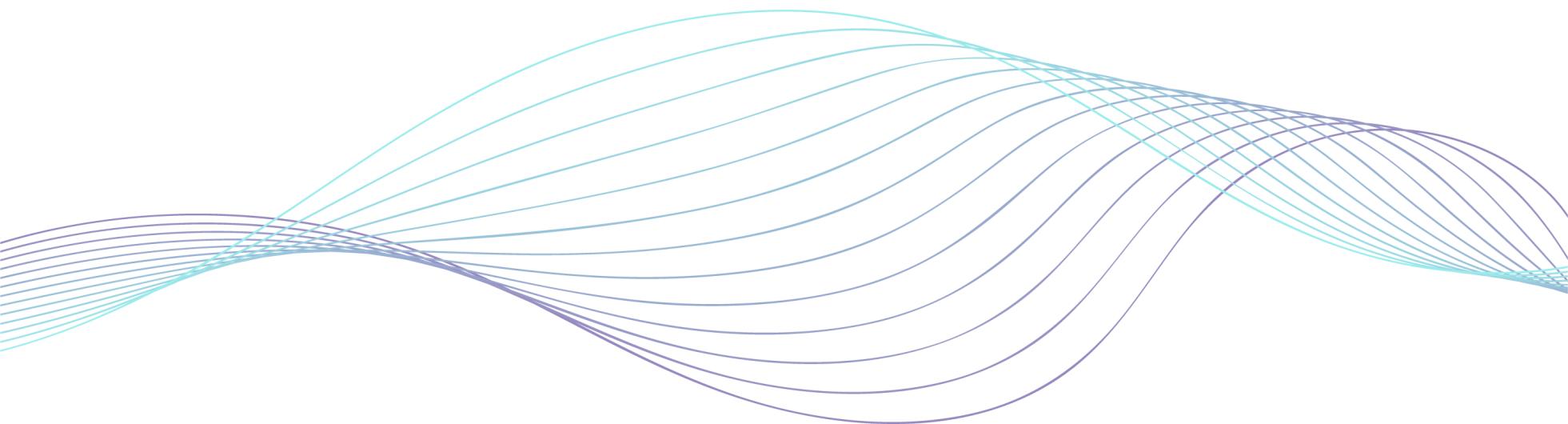
Templates

Las plantillas son componentes de Astro para proporcionar una estructura reutilizable, como una plantilla de página.

Convencionalmente usamos el término “plantilla” para proporcionar elementos compartidos en la interfaz del usuario por medio de páginas, como encabezados, barras de navegación y pies de página. Una plantilla proporciona a Astro, Markdown o páginas MDX con:

- una **PÁGINA SHELL** (con estiquetas `<html>`, `<head>` y `<body>`)
- un `<SLOT/>` para especificar dónde colocar el contenido de la página.

```
1   ---
2   import BaseHead from '../components/BaseHead.astro';
3   import Footer from '../components/Footer.astro';
4   const { title } = Astro.props;
5   ---
6   <html lang="es">
7     <head>
8       <meta charset="utf-8">
9       <meta name="viewport" content="width=device-width, initial-sc
10      <BaseHead title={title} />
11    </head>
12    <body>
13      <nav>
14        <a href="#">Inicio</a>
15        <a href="#">Publicaciones</a>
16        <a href="#">Contacto</a>
17      </nav>
18      <h1>{title}</h1>
19      <article>
20        <slot /> <!-- tu contenido es injectado aquí -->
21      </article>
22      <Footer />
23    </body>
24  </html>
25
```



<3> **Funcionalidades
avanzadas**

Índice



- <1> Contenido y Frameworks UI
- <2> Fetching, Auth y i18n
- <3> SSR y API Rest
- <4> Integraciones a medida

Contenido

El contenido en Astro se maneja mediante colecciones, y una colección de contenido es cualquier directorio dentro del directorio reservado `src/content`. Una entrada de colección se escribe en `.md/.mdx`, con una cabecera `.yaml`, podemos obviar la construcción de entradas con el prefijo `_md`.

Definición de colecciones

Podemos definir las colecciones siempre y cuando lo hagamos en `typescript`, esto se hará mediante un archivo `config.ts`, en la carpeta `content`. Aquí mediante `zod`, definiremos nuestros esquemas de validación.

```
1 import { z, defineCollection } from 'astro:content';
2
3 const blogCollection = defineCollection({
4   type: 'content', // v2.5.0 y posteriores
5   schema: z.object({
6     title: z.string(),
7     tags: z.array(z.string()),
8     image: z.string().optional(),
9   }),
10 });
11
12 export const collections = {
13   'blog': blogCollection,
14 };
15
```

Markdown y gestión de contenido

```
1  ---
2  layout: ../layouts/BaseLayout.astro
3  title: "¡Hola, mundo!"
4  author: "Matthew Phillips"
5  date: "09 Aug 2022"
6  ---
7  Todas las propiedades frontmatter están disponibles como props en un componente plantilla de Astro.
8  La propiedad `layout` es la única especial proporcionada por Astro.
9  Tu puedes usarla tanto en archivos Markdown y MDX localizada dentro de `src/pages/` .
10

1  ---
2  const { frontmatter } = Astro.props;
3  ---
4  <html>
5    <head>
6      <title>{frontmatter.title}</title>
7    </head>
8    <body>
9      <h1>{frontmatter.title} por {frontmatter.author}</h1>
10     <slot />
11     <p>Escrito en: {frontmatter.date}</p>
12   </body>
13 </html>
14
```

Static Site Generation (SSG) - Demo

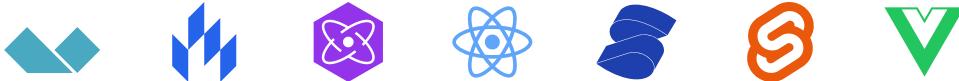


Integraciones con Frameworks UI

Con solo unas pocas líneas de código, las integraciones de Astro te permiten agregar nuevas funciones y comportamientos a un proyecto de Astro. Sin embargo, para escribir integraciones de Astro de manera efectiva, debemos comprender el ciclo de vida de los hooks de Astro.

Las integraciones amplían un proyecto de Astro con nuevas funcionalidades y comportamientos.

<https://astro.build/integrations/>



Integraciones con Frameworks UI

```
1  ---
2 // Ejemplo: hidratando los componentes de framework en el navegador.
3 import InteractiveButton from '../components/InteractiveButton.jsx';
4 import InteractiveCounter from '../components/InteractiveCounter.jsx';
5 import InteractiveModal from '../components/InteractiveModal.svelte';
6 ---
7 <!-- Este componente de JavaScript comenzará a importarse cuando se cargue la página-->
8 <InteractiveButton client:load />
9
10 <!-- El JavaScript de este componente no se enviará al cliente hasta que
11 el usuario se desplace hacia abajo y el componente sea visible en la página -->
12 <InteractiveCounter client:visible />
13
14 <!-- Este componente no se renderizará en el servidor, pero se renderizará en el
15 cliente cuando la página cargue -->
16 <InteractiveModal client:only="svelte" />
17
```

Solo los componentes de Astro (.astro) pueden contener componentes de múltiples frameworks.

Arquitectura insular - Demo



Fetching

Los componentes de Astro tienen acceso a la función global `fetch()` en su script de componente para hacer peticiones HTTP a APIs usando la URL completa.

Esta llamada a `fetch` será ejecutada **EN EL MOMENTO DE LA COMPILACIÓN**, y los datos estarán disponibles para la plantilla del componente para generar HTML dinámico. Si el modo `SSR` está habilitado, cualquier llamada a `fetch` será ejecutada **EN TIEMPO DE EJECUCIÓN**.

```
1  ---
2  import Contact from '../components/Contact.jsx';
3  import Location from '../components/Location.astro';
4
5  const response = await fetch('https://randomuser.me/api/');
6  const data = await response.json();
7  ---
8
9  <h1>Usuario</h1>
10 <h2>{data.randomUser.name.first} {data.randomUser.name.last}</h2>
11
12 <Contact client:load email={data.randomUser.email} />
13 <Location city={data.randomUser.location.city} />
14
```

Auth

La autenticación es un aspecto importante en el desarrollo de aplicaciones web modernas.

Normalmente en Astro utilizaremos librerías de autenticación como por ejemplo, Lucia Auth, Auth.js, que proporcionan utilidades para múltiples métodos de autenticación, como el inicio de sesión con correo electrónico y los proveedores de OAuth.

- [Ejemplo de proyecto Astro + Lucia OAuth](#)
- [Ejemplo de proyecto Astro + Lucia usuario y contraseña](#)
- [auth-astro en GitHub](#)
- [Documentación de Auth.js](#)

Rendering

Static - SSG

Por defecto es **static**, en el momento de compilación Astro crea el html de todas las rutas. Por defecto todos los proyectos en Astro están configurados para ser pre-renderizados en el momento de compilación para así dar una experiencia de usuario ligera.

Pueden incluir también islas de componentes interactivos escritos en el framework o librería elegido. Y View Transitions API está disponible.

Server / Hybrid

El renderizado bajo demanda en el servidor en el momento de la solicitud también se conoce como renderizado del lado del servidor **SSR**.

Normalmente usaremos estos modos cuando, utilicemos Astro para crear **Endpoints de API**, **Rutas protegidas**, **Contenido con cambio frecuente**.

Cuanto tenemos un sitio generalmente estático pero tenemos algunas rutas bajo demanda, entonces podremos configurarlo como **hybrid**. En ambos modos podremos utilizar la islas de Astro.

Integraciones SSR



API Endpoints - Demo

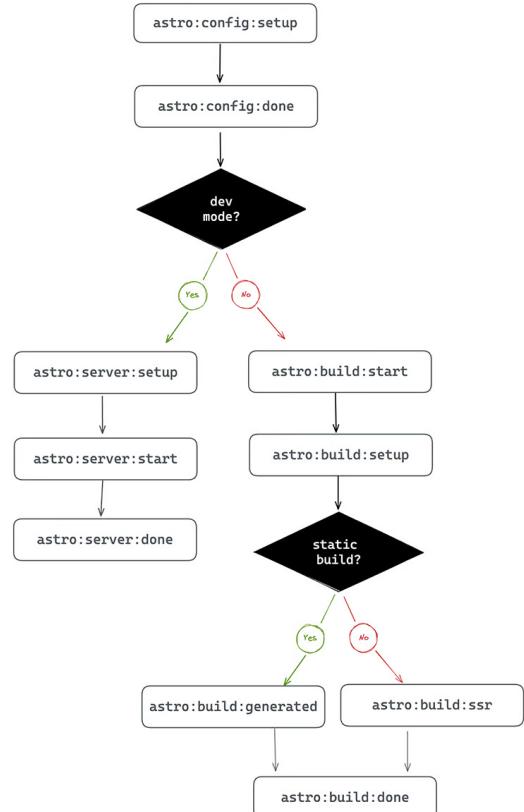


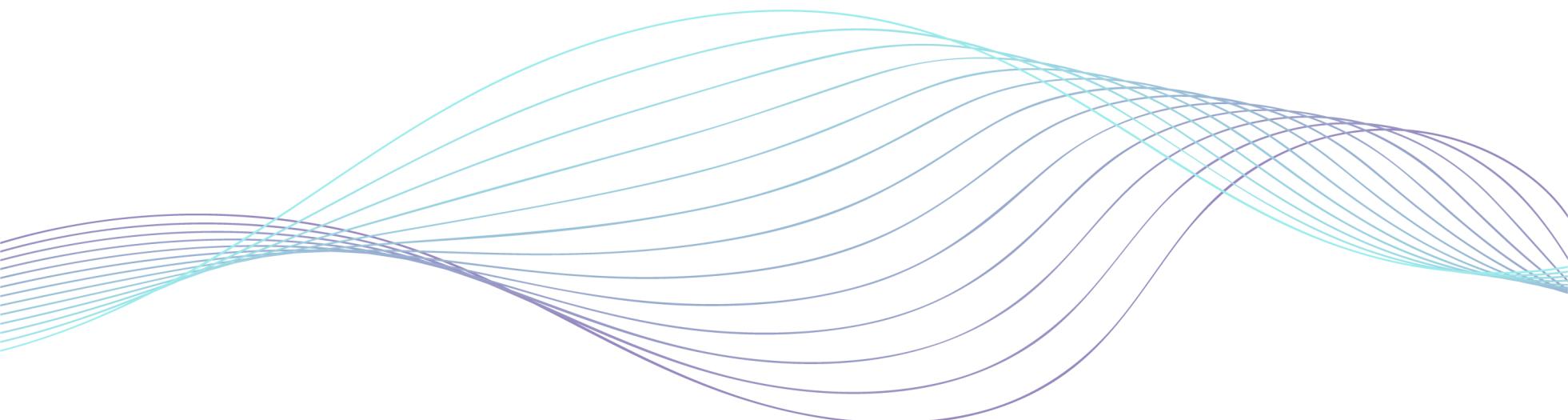
Integraciones a medida

Podemos desarrollar integraciones a medida para utilizar con Astro e interactuar con sus procesos, así es como lo hacen el resto de las librerías y frameworks.

Astro posee un ciclo de vida con el cual podemos integrar funcionalidades extra a nuestro proyecto.

Para desarrollar integraciones personalizadas de manera productiva, necesitaremos saber en qué parte del ciclo de vida queremos efectuar un cambio o reacción. Los hooks son funciones que se llaman en varias etapas de la construcción; Para interactuar con el proceso de construcción, podemos usar los siguientes diez hooks:





<4> **Calidad y
rendimiento**

Índice



- <1> Pruebas unitarias y de integración
- <2> Estudio del rendimiento

Pruebas unitarias y de integración

Astro soporta muchas herramientas populares para testing unitario, testing de componentes y testing end-to-end incluyendo **Jest**, **Mocha**, **Jasmine**, **Cypress** y **Playwright**.

Puedes *instalar librerías de testing específicas* para frameworks como React Testing Library para testear tus componentes de UI.

Astro recomienda las siguientes librerías para testing: <https://docs.astro.build/en/guides/testing/>

- **VITEST**: <https://github.com/withastro/astro/tree/latest/examples/with-vitest>
- **CYPRESS**
- **NIGHTWATCHJS**
- **PLAYWRIGHT**

Estudio del rendimiento

Es una prueba que analiza datos de medición de usuarios reales en tres métricas (**FIRST INPUT DELAY, CUMULATIVE LAYOUT SHIFT, LARGEST CONTENTFUL PAINT**) para determinar una calificación general de aprobado.

Para que un sitio web pase, debe alcanzar el umbral asociado de "bueno" en las tres métricas. Si alguna métrica no supera el umbral, el sitio web no pasa la evaluación.

La Evaluación CWV es un reflejo más preciso de cómo los usuarios realmente experimentan un sitio web, especialmente durante sesiones más largas.

LIGHTHOUSE y otras herramientas de prueba de laboratorio solo pueden medir la carga de la primera página, lo que no logra capturar la experiencia completa de usar un sitio web.

Core Web Vitals

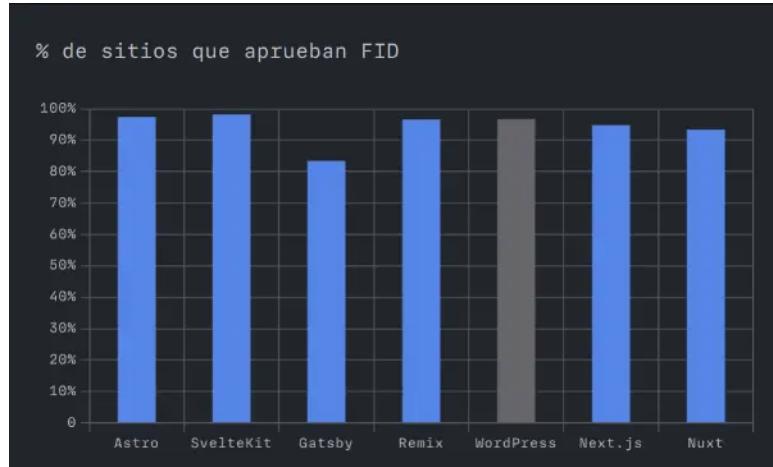


Al observar todos los sitios creados con un determinado framework, **ASTRO Y SVELTEKIT** superaron la tasa de aprobación promedio de todos los sitios web probados (40,5%).

Astro fue el único marco que llegó a más del 50 % de los sitios web que aprobaron la evaluación CWV de Google.

Next.js y Nuxt quedaron al final del grupo con aproximadamente 1 de cada 4 y 1 de cada 5 sitios web que aprobaron la evaluación, respectivamente.

First Input Delay



Mide el tiempo desde que un usuario interactúa por primera vez con una página hasta el momento en que el navegador puede responder a esa interacción. La evaluación CWV de Google busca un FID de 100 milisegundos o menos. Cualquier cosa más lenta se considera que necesita mejoras y no pasa la evaluación.

Ningún fwk cae por debajo de una tasa de aprobación del 80% en esta prueba. Esto significa que la mayoría de los sitios web probados responden a la primera interacción del usuario.

Cumulative Layout Shift



Mide la estabilidad visual en la página. Para aprobar esta evaluación, debe reducir los cambios inesperados en el diseño a casi cero para brindarles a sus usuarios una experiencia visual confiable.

Es una métrica que no está estrictamente relacionada con la velocidad o la capacidad de respuesta. Su inclusión subraya la importancia de considerar algo más que el rendimiento cuando se trata de medir la calidad general de las experiencias de los usuarios en la web.

Todos los frameworks obtuvieron una puntuación del 50% o más en esta métrica. Sin embargo, son los más jóvenes (Astro, SvelteKit y Remix) los que obtienen la puntuación más alta en esta métrica, superior al 75 %.

Largest Contentful Paint



Es el último de los tres Core Web Vitals y el más importante en lo que respecta al rendimiento percibido. Mide el momento en el que es probable que se haya cargado el contenido principal de la página. Se requiere un LCP de 2,5 segundos o menos para aprobar la evaluación CWV de Google. Cualquier cosa más lenta se considera que necesita mejoras y no pasa la evaluación.

Sólo el 52% de todos los sitios web analizados superan esta métrica. De los seis frameworks probados, sólo Astro y SvelteKit superaron este promedio.

Lighthouse

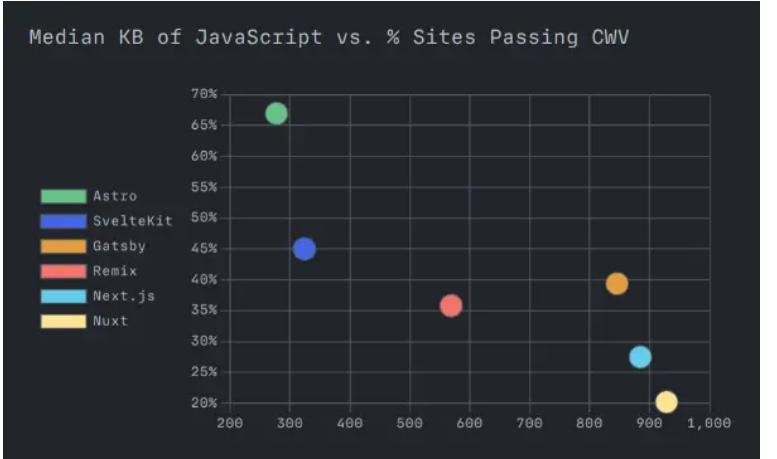


En lugar de analizar grandes umbrales y categorías "buenos" versus "malos", Lighthouse brinda una puntuación de rendimiento más detallada medida sobre 100.

Desafortunadamente, la puntuación media de rendimiento de Lighthouse es baja en todos los ámbitos. La mitad de los frameworks evaluados tuvieron un desempeño medio considerado "pobre" (49 o menos).

Solo la mitad de los frameworks probados (Astro, SvelteKit y Remix) estuvieron por encima del promedio de Internet.

El coste de Javascript



La tendencia en los datos es clara: los sitios que incluyen menos Javascript tienden a tener un mejor rendimiento.



Bibliografía

- <https://astro.build/>
- <https://docs.astro.build/>
- <https://github.com/understanding-astro/understanding-astro-book>
- <https://prismic.io/blog/javascript-meta-frameworks-ecosystem>
- <https://blog.logrocket.com/understanding-astro-islands-architecture/>
- <https://blog.logrocket.com/understanding-astro-integrations-hooks-lifecycle/>
- <https://www.freecodecamp.org/news/how-to-use-the-astro-ui-framework/>
- <https://jsonformat.com/islands-architecture/>
- <https://frontendmastery.com/posts/understanding-micro-frontends/>
- <https://betterprogramming.pub/microfrontends-island-architecture-ef94d95f2214>
- <https://betterprogramming.pub/astro-js-the-new-kid-on-the-block-33fa295104b8>
- <https://astro.build/blog/2023-web-framework-performance-report/>

IVK innusual
together in technology