Astro
Guia de atajos

{dev/talles}

Crear aplicación

Pueden cambiar npm por, pnpm, yarn, bun.

```
npm create astro@latest
```

Sintaxis

Similar a JSX, separado en dos partes, el HTML y el código de JavaScript / TypeScript llamado “FrontMatter”, “component script” o Metadatos, el cual está separado por dos bloques de código (- - -) (**Code fences**).

```
---
const name = 'Astro';
---

<div>
  <h1>Hello {name}!</h1>
</div>
```

Importante: Todo el frontmatter es código que se ejecuta únicamente en el servidor o cuando se hace se construye la aplicación en “build time”

Iteraciones de elementos

```
---
const items = ["Dog", "Cat", "Platypus"];
---
<ul>
  {items.map((item) => (
    <li>{item}</li>
  ))}
</ul>
```

Nota: Como notarás, no hay un “key” en la iteración del arreglo, esto debido a que el contenido se genera de forma estática en tiempo de construcción.

Iteraciones de elementos

Usamos ternarios y/o expresiones lógicas.

```
---
const visible = true;
---
{visible && <p>Show me!</p>}

{visible ? <p>Show me!</p> : <p>Else show me!</p>}
```

Nota: La variable visible, es estática (no reactiva) por lo que aunque pudieras cambiarla, no reaccionaría tu DOM.

Elementos HTML Dinámicos

Puedes crear elementos HTML dinámicos.

```
---
const Element = 'div'
---
<Element>Hello!</Element>
<!-- <div>Hello!</div> -->
```

Importar otros componentes

Pedes mandar properties y elementos HTML usando Props y Slots.

```
---
import MyComponent from "./MyComponent.astro";
---
<Component /> <!-- renders as <MyComponent /> -->
<Component title="Hola" isActive />
<Component title={'Hola'} />

<Component>
  <h1>Hi</h1>
</Component>
```

Nota: Valores booleanos, pueden especificarse únicamente con el nombre lo que equivale a isActive={true}

Fragmentos

Son útiles cuando quieres para agrupar elementos sin necesidad de colocar un DIV adicional.

```
---
const htmlString = '<p>Raw HTML content</p>';
---
<Fragment set:html={htmlString} />
```

Nota: Los archivos .astro, no necesitan fragmentos cuando se quiere regresar elementos en el root, a diferencia del JSX tradicional.

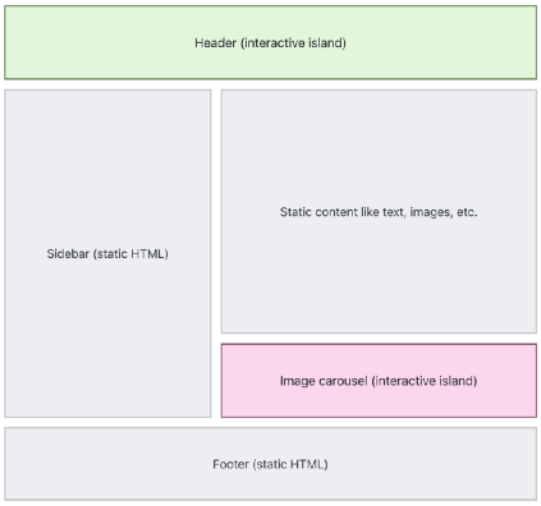
Componentes de Astro

- Permiten importar otros componentes.
- Importar componente de otros Frameworks (**islas**)
- Importar archivos de JS/TS
- Importar JSON
- Realizar peticiones http
- Hablar al backend
- Hijo asíncrono por defecto (**await directo**)
- Código de JavaScript o TypeScript
- Pueden recibir otros componentes mediante **slots**

Astro
Guia de atajos

{dev/talles}

Islas
Es un componente interactivo que permite que permite JavaScript en el lado del cliente, si es necesario. Puedes usar muchos UI Frameworks y librerías como React, Preact, Vue, Solid, HTMX y mucho más.



Properties
Puedes enviar y recibir properties de esta forma e inclusive Astro infiere las properties si usas TypeScript definiendo una interfaz opcional llamada **Props.**

```
---
// Uso: <GreetingHeadline greeting="Howdy" name="Partner" />
const { greeting, name } = Astro.props;
---
<h2>{greeting}, {name}!</h2>
```

TypeScript

```
---
interface Props {
  greeting: string;
  name: string;
}

const { greeting, name } = Astro.props;
---
<h2>{greeting}, {name}!</h2>
```

Nota: Observen que no es necesario usar la interfaz props en un elemento genérico. Astro las infiere.

Slots
Es un espacio destinado a un elemento HTML externo, permitiendo inyectar el mismo elemento dentro del slot.

MainLayout.astro

```
---
import Header from './Header.astro';
import Footer from './Footer.astro';

const { title } = Astro.props;
---
<div id="content-wrapper">
  <Header />
  <h1>{title}</h1>
  <slot />  <!-- Los hijos irán aquí -->
  <Footer />
</div>
```

Fred.astro

```
---
import MainLayout from '../layouts/MainLayout.astro';
---
<MainLayout title="Fred's Page">
  <h2>Todo sobre Fred</h2>
  <p>Aquí cosas sobre Fred</p>
</MainLayout>
```

Slots con Nombre
En caso de ocupar enviar hijos y alojarlos en lugares específicos, podemos usar los nombres de los slots. Ten presente que el “**<slot />**” es el lugar por defecto.

```
---
import Header from './Header.astro';
import Footer from './Footer.astro';

const { title } = Astro.props;
---
<div id="content-wrapper">
  <Header />
  <slot name="after-header" />
  <h1>{title}</h1>
  <slot />
  <Footer />
  <slot name="after-footer" />
</div>
```

Astro
Guia de atajos

{dev/talles}

Uso del slot con nombre

Ten presente que el "**<slot />**" es el lugar por defecto.

```
---
import MainLayout from '../layouts/
MainLayout.astro';
---
<MainLayout title="Fred's Page">

  <img src="/red.jpg" slot="after-header" />

  <h2>All about Fred</h2>
  <p>Here is some stuff about Fred.</p>

  <p slot="after-footer">Copyright 2022</p>
</MainLayout>
```

Router - File based routing

Cualquier archivo del siguiente listado que sea colocado en la carpeta **"src/pages"**, se transformará en una ruta.

.astro, .md, .html	
.mdx	Requiere integración
.js, .ts	**Como API endpoints**

Nota: Componentes islas, no pueden ser páginas, y para navegar entre pantallas usamos el simple anchor tag que ya conocemos

Páginas de error 404

Simplemente crea un 404.astro, 404.md en **"src/pages"** y esto creará el 404.html

Páginas parciales

Permite regresar páginas HTML parciales, lo que permite trabajar con **HTMX**

Importante:
Siempre debe de existir una carpeta "**src/pages**" dentro del proyecto.

Crea rutas:
- blog/index.astro
- about.astro
- index.astro

rss.xml.js: es un endpoint que retorna un xml

Layouts - Plantillas

No son mas que componentes de Astro con un slot que permite la reutilización de código, pero usualmente estos tienen etiquetas html, head y body

```
---
import BaseHead from '../components/BaseHead.astro';
import Footer from '../components/Footer.astro';
const { title } = Astro.props;
---
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <BaseHead title={title}/>
  </head>
  <body>
    <nav>
      <a href="#">Home</a>
      <a href="#">Posts</a>
      <a href="#">Contact</a>
    </nav>
    <h1>{title}</h1>
    <article>
      <slot /> <!-- your content is injected here -->
    </article>
    <Footer />
  </body>
</html>
```

Layouts en Markdown o MDX

Agrega una propiedad en el frontmatter del MD llamada layout con el path relativo al archivo Layout

```
---
layout: ../layouts/BaseLayout.astro
title: "Hello, World!"
author: "Matthew Phillips"
date: "09 Aug 2022"
---
All frontmatter properties are available as props
to an Astro layout component.
```

Todas las properties del frontmatter, están disponibles en las props

```
---
const { frontmatter } = Astro.props;
---
```

Astro
Guia de atajos

{dev/talles}

Estilos
Astro puede manejar estilos globales, por componente, por sitio web.

Este código crea un estilo con ámbito del componente, el cual es el recomendado por Astro.

```
<style>
  h1 { color: red; }
</style>
```

El siguiente código genera un estilo global

```
<style is:global>
  /* Astro no aplica scope aquí */
  /* Todos los h1 serán rojos en el sitio */
  h1 { color: red; }
</style>
```

Combinar clases de CSS
Puedes combinar clases de CSS con la directiva class:list y mandar un arreglo, objetos o una mezcla.

```
---
const { isRed } = Astro.props;
---
<div class:list={['box', { red: isRed }]}><slot /></div>

<style>
  .box { border: 1px solid blue; }
  .red { border-color: red; }
</style>
```

Variables de CSS
Es posible mandar desde variables desde el script del componente directo al css y se transforman en variables de CSS

```
---
const foregroundColor = "rgb(221 243 228)";
const backgroundColor = "rgb(24 121 78)";
---
<style define:vars={{ foregroundColor, backgroundColor }}>
  h1 {
    background-color: var(--backgroundColor);
    color: var(--foregroundColor);
  }
</style>
```

Estilo en línea - Inline style
Puedes definir tu estilo en línea de dos formas

```
<p style={{ color: "brown", textDecoration: "underline" }}>My text</p>
<p style="color: brown; text-decoration: underline;">My text</p>
```

Estilos externos
Puedes colocar estilos externos en los layouts, esos estilos pueden ser paths completos a CDNs o bien a tu propia carpeta **public**

Importar estilos localmente
Podemos importar estilos usando la sintaxis de módulos de ECMAScript - ESM

```
---
// AStro va a empaquetar y optimizar este CSS automáticamente para ti
// Esto también funciona para archivos como .scss, .styl, etc.

import '../styles/utils.css';
---
<html><!-- Tu página --></html>
```

Integraciones
Las integraciones de Astro añaden y expanden funcionalidad de una forma simple y eficiente.

Con integraciones puedes:
- Añadir React, Vue, Svelte, Solid, etc
- Añadir herramientas como Tailwind y Partytown
- Añadir nuevas funcionalidades como RSS y Mapas
- Añadir código personalizado, server side y más.

```
npx astro add react
npx astro add tailwind party town node
```

Variables de entorno
Podemos definir 2 tipos de variables de entorno, las privadas usadas únicamente en el lado del servidor y las publicas, que serán enviadas al cliente.

```
PUBLIC_WEBSITE=http://localhost:4321
AUTH_SECRET=EstoNoSeVeraEnElCliente
```

La diferencia es que la pública debe de empezar con la palabras **PUBLIC_** esto le permite a Vite / Astro, enviarla al cliente.

Astro
Guia de atajos

{dev/talles}

Component Frameworks

Hay una lista de integraciones oficiales con Astro, como también integraciones de terceros.

Para usar un framework / librería

```
---
import MyReactComponent from '../components/
MyReactComponent.jsx';
---
<html>
  <body>
    <h1>Use React components directly in Astro!</h1>
    <MyReactComponent />
  </body>
</html>
```

Nota: Por defecto, los componentes usados de esta forma, generan su HTML en el servidor y no envían **nada** de JavaScript al cliente.

Pero si queremos la interactividad producto de estos frameworks/librerías, debemos de añadir directivas:

```
---
// Example: hidratar componentes interactivos en el navegador.

import InteractiveButton from '../components/InteractiveButton.jsx';
import InteractiveCounter from '../components/InteractiveCounter.jsx';
import InteractiveModal from '../components/InteractiveModal.svelte';
---
 <!-- Inicia su importación cuando se carga la página -->
<InteractiveButton client:load />

<!-- Cuando el componente es visible en la página -->
<InteractiveCounter client:visible />

<!-- No renderiza nada en el servidor -->
 <!-- Pero hay que especificar el framework -->
<InteractiveModal client:only="svelte" />
```

Nota: Cuando estamos en el "**client:only**" hay que especificar la tecnología usada porque Astro en ese punto no sabe qué tecnología fue usada para su creación.

Directivas disponibles:

client:idle	Cuando el navegador está inactivo (no esta cargando nada)
client:load	Empieza su importación cuando la página empieza a cargar
client:media	Mediante un media query, podemos decir cuándo queremos que empiece a cargar
client:only	Delega la construcción del componente directamente al cliente
client:visible	Cuando el componente entra en el punto de vista de la pantalla

Astro Glob

Es una función que permite importar muchos archivos de una sola vez

```
---
// importa todos los archivos que terminan con MD en el
directorio `./src/pages/post/`
const posts = await Astro.glob('../pages/post/*.md');
---
<!-- Muestra 5 artículos -->
<div>
{posts.slice(0, 4).map((post) => (
  <article>
    <h2>{post.frontmatter.title}</h2>
    <p>{post.frontmatter.description}</p>
    <a href={post.url}>Read more</a>
  </article>
))}
```

Content Collections

Son una excelente forma de organizar tu contenido con tipado estricto y organizarlo para su consumo.

Tienen que estar en una carpeta llamada **content,** y estar definido dentro del **config.ts**

Luego por cada colección, debe de haber una carpeta independiente, pero puedes tener sub-carpetas también.

```
import { defineCollection, z } from 'astro:content';

const blog = defineCollection({
  type: 'content',
  // Type-check frontmatter usando el esquema
  schema: z.object({
    title: z.string(),
    description: z.string(),
    // Transforma el string a fecha
    pubDate: z.coerce.date(),
    updatedDate: z.coerce.date().optional(),
    heroImage: z.string().optional(),
  }),
});

export const collections = { blog };
```

Astro
Guia de atajos

{dev/talles}

Peticiones HTTP
Pueden crear RESTful API endpoints estáticos o dinámicos.

GET Request: Es opcional poner el status si la petición retornará de forma exitosa.

```
import type { APIRoute } from 'astro';

export const GET: APIRoute = async ({ params, request }) => {
  return new Response('Ok!', { status: 200 });
};
```

POST Request: De forma, se crea una función con el nombre del método que quieras invocar.

```
import type { APIRoute } from 'astro';

export const POST: APIRoute = async ({ request, redirect }) => {

  const formData = await request.formData();
  const email = formData.get('email')?.toString();
  const password = formData.get('password')?.toString();

  if (!email || !password) {
    return new Response(
       'Email and password are required',
       { status: 400 }
     );
  }

   // Revisar las credenciales
  if (!hasValidCredentials) {
    return new Response(error.message, { status: 401 });
  }

  return redirect('/dashboard');
```

Esto aplica también para peticiones **PUT, PATCH y Delete.**

Nota:
Recuerden que el comportamiento de POST, PATCH y Delete, requieren generación del lado del servidor.

Si haces SST (static site generation) las peticiones **GET** pueden ser dinámicas si se especifican los **getStaticPaths.**

Configurar Server Side Rendering
En el archivo **astro.config.mjs**

```
import { defineConfig } from 'astro/config';
import node from "@astrojs/node";

export default defineConfig({
  output: 'server', // hybrid
  adapter: node({
    mode: "standalone"
  })
});
```

Hybrid:
Significa que todo el contenido será generado de forma estática con la excepción de páginas que tengan el **prerender = false;**

```
export const prerender = false;
```

Sever:
Significa que todo el contenido será generado del lado del servidor en momento de la solicitudcon la excepción de páginas que tengan el **prerender = true;**

```
export const prerender = true;
```

Nota:
Esto aplica para endpoints y archivos MDX.

Cookies
Cuando se trabaja en SSR, Hybrid o Server,

```
---
let counter = 0

if (Astro.cookies.has("counter")) {
  const cookie = Astro.cookies.get("counter")
  counter = cookie.number() + 1
}

Astro.cookies.set("counter",counter)
---
<html>
  <h1>Counter = {counter}</h1>
</html>
```

Response
Podemos cambiar el tipo de respuesta de esta forma o utilizando el objeto response.

```
---
import { getProduct } from '../api';

const product = await getProduct(Astro.params.id);

// No product found
if (!product) {
  Astro.response.status = 404;
  Astro.response.statusText = 'Not found';
}
---
<html>
  <!-- Page here... -->
</html>
```