

Explorando Jamstack y Arquitectura de Islas con Astro

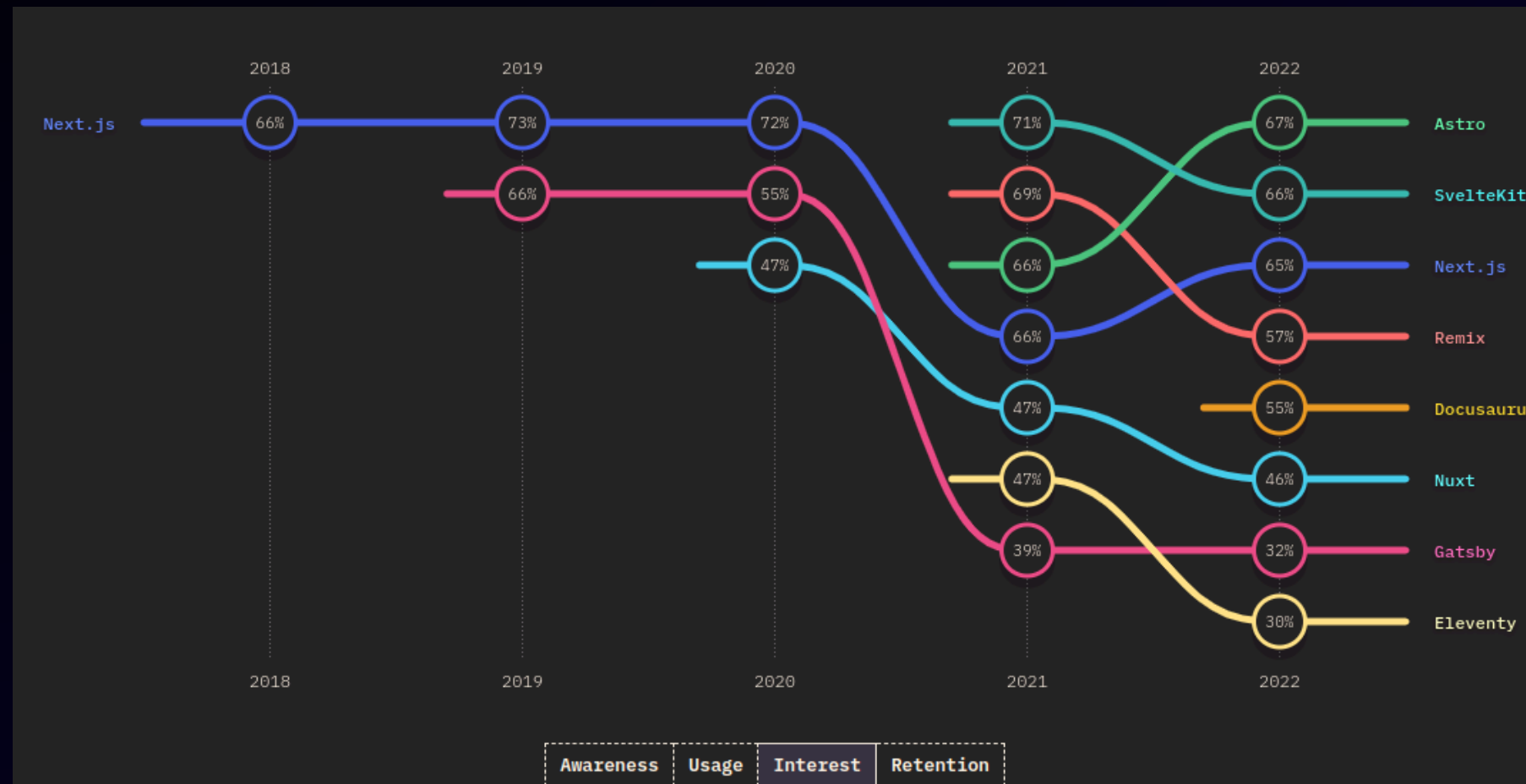
Manuel Sepúlveda – Daniel Toribio – Tomás Trincado



<https://t5-web.vercel.app>

Demo

¿Qué es Astro y la Arquitectura de Islas?



Astro = Cero JavaScript por defecto
= Sitios más rápidos.

Interactividad = "Islas" (Componentes de React/Vue/Svelte).

Botón de Favoritos (Isla de React)

- Es una acción principal. Queremos que sea interactivo de inmediato, sin demoras.

Reproductor de Audio (Isla de Vue)

- Optimiza el rendimiento. El componente solo se carga cuando el usuario se desplaza y lo ve en la pantalla.

Sección de Favoritos (Isla de React)

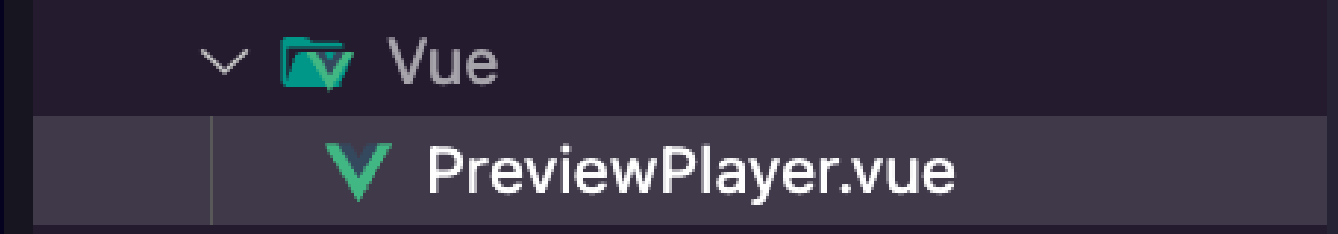
- Similar al reproductor, solo se carga cuando el usuario se desplaza

Nuestra Arquitectura Estático + Islas

```
<div class="controls-section">  
  <PreviewPlayer client:load previewUrl={preview_url} />  
  <FavoriteButton client:visible songId={id} />  
</div>
```

```
<FavoritesSection client:visible allSongs={canciones} />
```


Preview Player



```
<template>
  <div class="preview-player">
    <button
      @click="togglePlay"
      :disabled="!canPlay || isLoading"
      class="play-button"
      :class="{ playing: isPlaying, loading: isLoading }"
    >
      <div v-if="isLoading" class="loading-spinner"></div>
      <span v-else-if="isPlaying">⏸ Pausar</span>
      <span v-else-if="canPlay">▶ Reproducir</span>
      <span v-else>⛔ No disponible</span>
    </button>

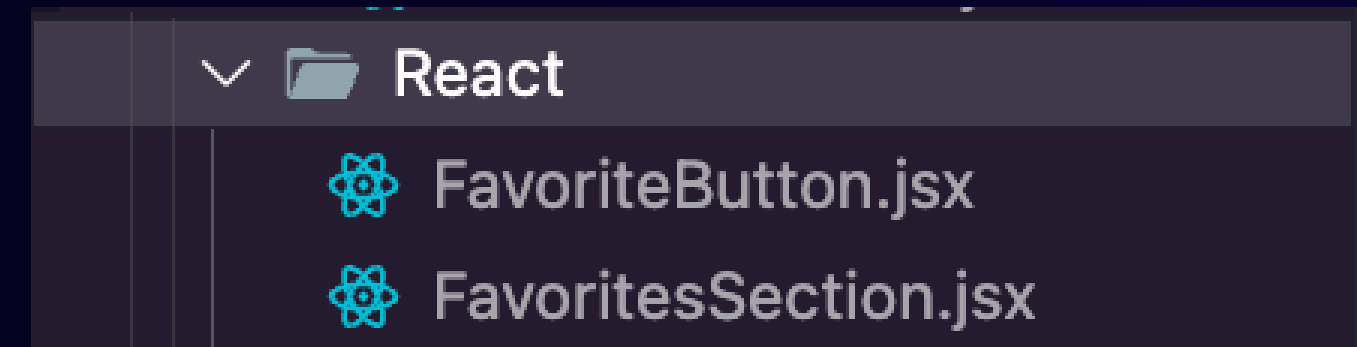
    <div v-if="error" class="error-message">{{ error }}</div>

    <audio ref="audio" preload="none" @ended="onAudioEnded" />
  </div>
</template>
```

- Maneja múltiples estados (cargando, reproduciendo, error) para dar un feedback claro al usuario.
- Implementa un sistema de 3 reintentos automáticos si la carga del audio falla, mejorando la experiencia de usuario.
- La UI responde en tiempo real a las acciones del usuario y a los cambios de estado del audio.

Favorite Button

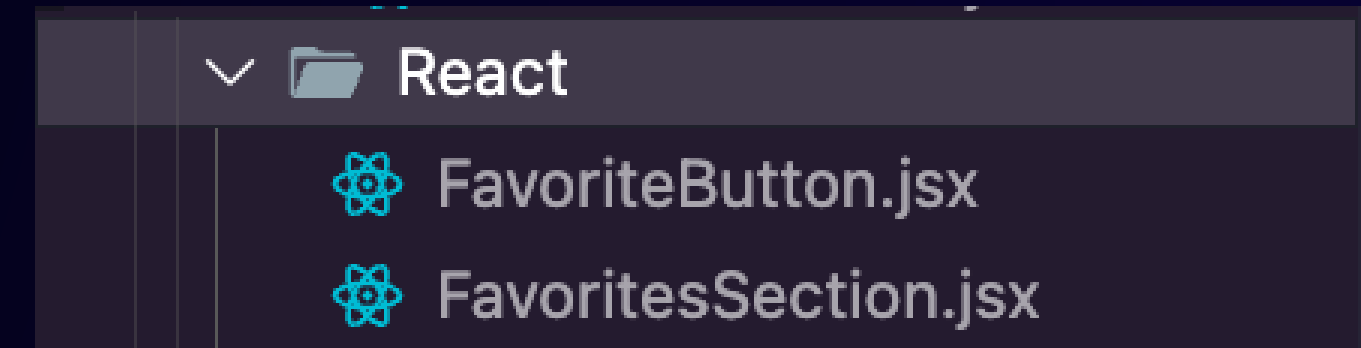
```
return (  
  <div className="favorite-container">  
    <button  
      onClick={toggleFav}  
      className={`favorite-button ${isFav ? "favorited" : ""} ${  
        isAnimating ? "animating" : ""  
      }}`  
      aria-label={isFav ? "Quitar de favoritos" : "Agregar a favoritos"}  
    >  
      <div className="heart-container">  
        <div className={`heart ${isFav ? "filled" : ""}`}>  
          {isFav ? "♥" : "♡"}  
        </div>  
        <div className="heart-bg"></div>  
      </div>  
      <span className="button-text">  
        {isFav ? "En favoritos" : "Agregar a favoritos"}  
      </span>  
    </button>  
  
    {showToast && <div className="toast-notification">{toastMessage}</div>}  
  </div>  
);
```



- Isla de React diseñada para capturar y procesar la interacción del usuario (el clic).
- Utiliza localStorage para escribir y guardar la elección del usuario, asegurando que los favoritos persistan entre visitas.
- Despacha un CustomEvent global para desacoplar la comunicación, permitiendo que otros componentes reaccionen al cambio de estado sin estar directamente conectados.

Favorite Section

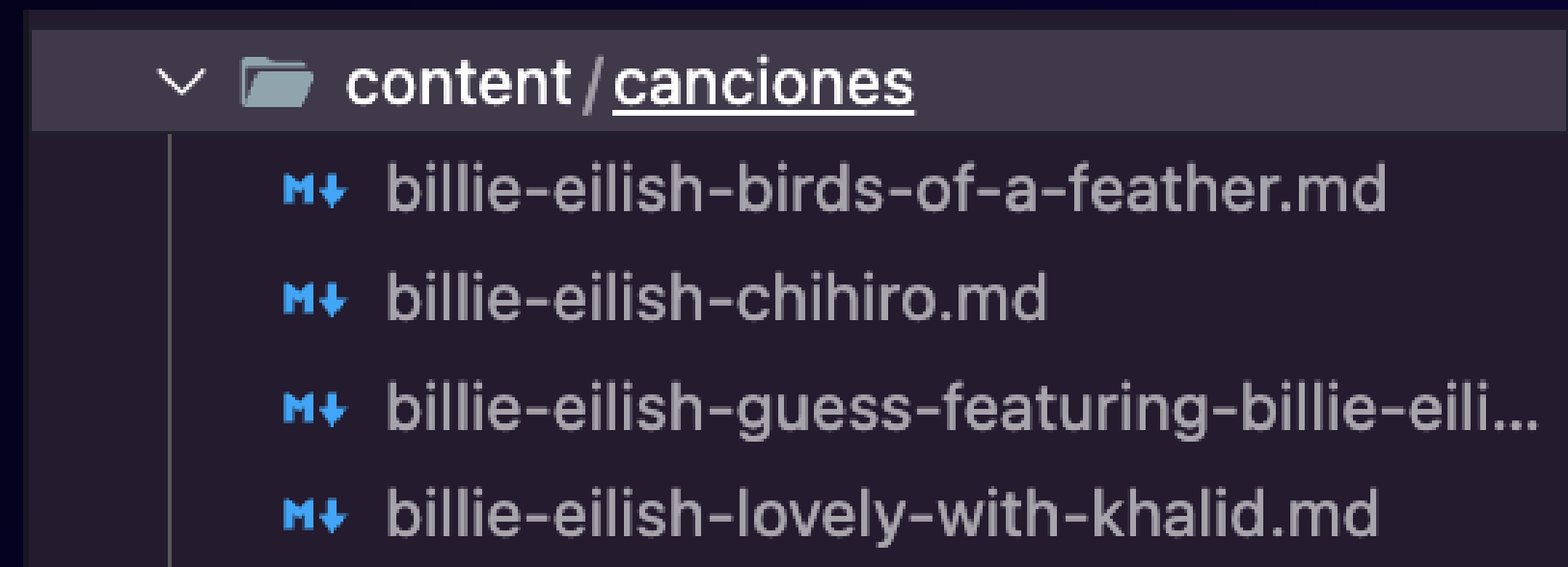
```
return (  
  <div className="favorites-section">  
    <h2 className="section-title">  
      📖 Mis Favoritos ({favoriteSongs.length})  
    </h2>  
    <div className="favorites-grid">  
      {favoriteSongs.map((song) => (  
        <div key={song.data.id} className="favorite-item">  
          <a href={` /cancion/${song.slug}`} className="favorite-link">  
            <img  
              src={song.data.image}  
              alt={song.data.name}  
              className="favorite-image"  
            />  
            <p className="favorite-name">{song.data.name}</p>  
          </a>  
          <button  
            onClick={() => removeFavorite(song.data.id)}  
            className="remove-favorite-btn"  
            aria-label="Remover de favoritos"  
          >  
            ✖  
          </button>  
        </div>  
      ) )}  
    </div>  
  </div>  
)
```



- Su única responsabilidad es mostrar el estado actual de los favoritos del usuario.
- Al montarse, utiliza useEffect para leer los datos desde localStorage y renderizar la lista inicial de favoritos.
- Escucha activamente el CustomEvent disparado por el FavoriteButton. Esto le permite actualizarse en tiempo real sin necesidad de recargar la página, creando una experiencia fluida.

- Datos de la API de Spotify
 - Archivos Markdown.
- Astro usa "Content Collections" para leer estos archivos.
- npm run build
 - Se generan +50 páginas HTML estáticas.
- Script para obtener los datos de Spotify
- Astro usa estos archivos para generar todo el sitio de forma estática.

Generación de Contenido (SSG)



```
id: "6d0tVTDdiauQNBQED0tLAB"  
name: "BIRDS OF A FEATHER"  
artist: "Billie Eilish"  
image: "https://i.scdn.co/image/ab67616d0000b27371d62ea7ea8a5be92d3c1f62"  
preview_url: "https://p.scdn.co/mp3-preview/2899f0275fc029d456d924a60ae6f747bda1ed80"
```


Generación de Contenido (SSG)

```
const client_id = process.env.SPOTIFY_CLIENT_ID!;
const client_secret = process.env.SPOTIFY_CLIENT_SECRET!;

export async function getSpotifyToken(): Promise<string> {
  const res = await axios.post(
    "https://accounts.spotify.com/api/token",
    "grant_type=client_credentials",
    {
      headers: {
        "Content-Type": "application/x-www-form-urlencoded",
        Authorization:
          "Basic " +
          Buffer.from(`${client_id}:${client_secret}`).toString("base64"),
      },
    },
  );

  return res.data.access_token;
}
```

```
export async function getTopTracksFromArtist(artistId: string): Promise<any[]> {
  const token = await getSpotifyToken();

  const res = await axios.get(
    `https://api.spotify.com/v1/artists/${artistId}/top-tracks?market=US`,
    {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    },
  );

  const tracks = res.data.tracks;

  const enriched = await Promise.all(
    tracks.map(async (track: any) => {
      if (!track.preview_url) {
        const fallback = await findPreviewUrl(
          track.name,
          track.artists[0]?.name ?? ""
        );
        return { ...track, preview_url: fallback };
      }
      return track;
    })
  );

  return enriched;
}
```

Robustez ante Datos Incompletos

Si Spotify no entrega preview, activamos un "Plan B" que obtiene el fragmento desde una fuente alternativa para no fallarle al usuario.

```
const enriched = await Promise.all(
  tracks.map(async (track: any) => {
    if (!track.preview_url) {
      const fallback = await findPreviewUrl(
        track.name,
        track.artists[0]?.name ?? ""
      );
      return { ...track, preview_url: fallback };
    }
    return track;
  })
);

return enriched;
```

Reproductor a Prueba de Fallos

Nuestro reproductor Vue es paciente: si la carga de audio falla, reintentará hasta 3 veces antes de mostrar un error.

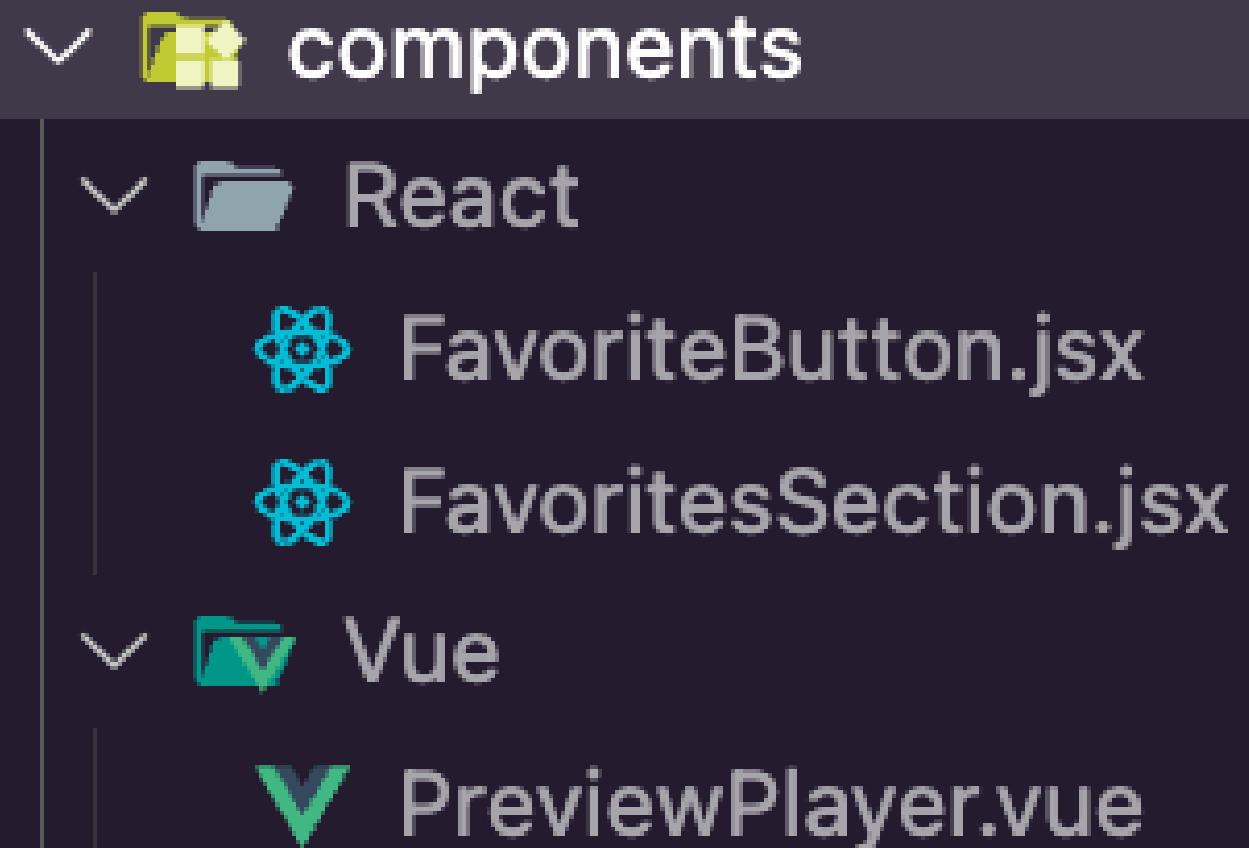
```
try {
  const resp = await fetch(props.previewUrl);
  if (!resp.ok) throw new Error('HTTP ' + resp.status);
  const blob = await resp.blob();
  const blobUrl = URL.createObjectURL(blob);

  audio.value.src = blobUrl;
  await new Promise<void>((resolve, reject) => {
    const onCP = () => {
      resolve();
    };
    const onErr = () => {
      reject();
    };
    audio.value!.addEventListener('canplaythrough', onCP, { once: true });
    audio.value!.addEventListener('error', onErr, { once: true });
    audio.value!.load();
  });

  canPlay.value = true;
} catch (e) {
  retryCount++;
  if (retryCount <= MAX_RETRIES) {
    setTimeout(prepareAudio, 500);
  } else {
    error.value = 'No se pudo cargar la vista previa.';
  }
} finally {
  isLoading.value = false;
}
```


Integración Multi-Framework

Con Astro combinamos React para el botón de favoritos y Vue para el reproductor en solo dos líneas de configuración, demostrando su gran flexibilidad.



```
components
├── React
│   ├── FavoriteButton.jsx
│   └── FavoritesSection.jsx
└── Vue
    └── PreviewPlayer.vue
```



Conclusiones

Para concluir, este proyecto nos permitió aplicar exitosamente los conceptos de Jamstack y la arquitectura de islas.

Demostramos que con Astro es posible construir un sitio de alto rendimiento, con páginas estáticas generadas desde Content Collections, y al mismo tiempo, enriquecerlo con islas interactivas de React y Vue.

Astro demostró ser una herramienta flexible y potente, ideal para el desarrollo web moderno.

Explorando Jamstack y Arquitectura de Islas con Astro

Manuel Sepúlveda – Daniel Toribio – Tomás Trincado