



Guía del alumno: Proyecto "La Piazza"

¡Bienvenido chef del código! En esta práctica vamos a trabajar con **Astro**, una de las herramientas más modernas para crear sitios web ultra rápidos. He preparado esta guía para que puedas descargar el proyecto y entender cómo funciona por dentro.

🚀 1. ¿Cómo poner en marcha el proyecto?

1. **Clonar:** `git clone https://github.com/guillermofoix/la-piazza-astro.git`
2. **Entrar:** `cd la-piazza-astro`
3. **Instalar:** `npm install`
4. **Ejecutar:** `npm run dev` → Abre `http://localhost:4321`

Ἑ 2. ¿Cómo se organiza Astro? (Teoría esencial)

Astro organiza la web como si fuera una pizzería:

- **src/pages/**: Las páginas (el Menú). Cada `.astro` aquí es una URL.
- **src/layouts/**: La estructura base (plato, servilletas, cubiertos).
- **src/components/**: Los ingredientes (botones, fotos, textos).

🛒 3. El Corazón de la Tienda: El Carrito (Explicación Exhaustiva)

¿Cómo es posible que hagas clic en un botón en la mitad de la página y el numerito de arriba se actualice al instante? Vamos a ver los 3 pilares que hacen que el carrito de "La Piazza" funcione:

A. El Cerebro: NanoStores (`src/cartStore.ts`)

Imagina que en la cocina hay una **pizarra central** donde todos los camareros escriben los pedidos.

- Usamos una librería llamada **NanoStores**.
- Esta librería crea un "espacio de memoria" compartido. Los componentes no se hablan entre ellos, todos hablan con la pizarra.
- Cuando el botón "Añadir" escribe una nueva pizza en la pizarra, el icono del carrito (que está mirando la pizarra) lo ve y se actualiza solo.

B. La Memoria: `localStorage`

Si refrescas la página, tus pizzas siguen en el carrito. Esto es porque el archivo del Store tiene una orden especial: "*Cada vez que la pizarra cambie, guarda una copia en el disco duro del navegador*".

C. La Interfaz: React + Astro

Astro es estático (no tiene "vida" por defecto para que la web vole). Para el carrito necesitamos "vida" (interactividad).

- Por eso usamos **React**.
 - Al llamar al componente en un archivo `.astro`, verás que usamos `client:load`. Esto le dice a Astro: "Oye, carga el motor de React para este trozo, porque tiene que moverse".
-

🛠 4. ¿Cómo llevar este carrito a tu proyecto `Guia_practica_astro`?

Si quieras que tu proyecto anterior tenga esta misma potencia, estos son los pasos exactos que tendrías que seguir:

1. **Instalar las herramientas:** Abre tu terminal en la carpeta de tu proyecto antiguo e instala NanoStores:

```
npm install nanostores @nanostores/react
```

2. **Crear la "Pizarra" (El Store):** Crea un archivo `src/cartStore.js` y define una variable `cartItems` usando `atom` o `map` de nanostores. Este archivo será el que importe cualquier botón para añadir productos.

3. **El Botón Mágico:** Crea tu componente `BotonAñadir` en React. Dentro de la función de clic, haz que llame a `cartItems.set(...)`.

4. **Conectar con la Base de Datos (Opcional):** Si en tu otro proyecto tienes una base de datos (con Node.js o Firebase), el proceso sería:

- En el botón de "Añadir", antes de escribir en la pizarra de NanoStores, haces una petición `fetch` a tu servidor para guardar el pedido en la DB.
 - Así el carrito no solo existe en el navegador del alumno, sino también en tu base de datos central.
-

🌐 5. Origen del proyecto: Astro + Shopify

Este proyecto se basa en el template profesional **Storeplate** de [Zeon Studio](#). El repositorio original es: <https://github.com/zeon-studio/storeplate>.

¿Cómo funciona en el mundo real (sin adaptaciones)?

Nosotros hemos "trucado" el proyecto para que funcione sin conexión a internet usando datos simulados (Mock Data). Sin embargo, este template está diseñado para conectarse directamente a **Shopify**.

Si tú tuvieras una tienda en Shopify:

1. **Conexión Directa:** No tendrías que programar los productos. Solo tendrías que poner tu `Domain` y tu `Access Token` en el archivo `.env`.
2. **Sincronización Total:** Si cambias el precio de una pizza en el panel de control de Shopify, la web de Astro se actualizaría automáticamente sin tocar una sola línea de código.
3. **Pagos Reales:** El carrito enviaría al cliente directamente a la pasarela de pago segura de Shopify.

Esto es lo que se llama **Headless Commerce**: Usar una plataforma potente para los datos (Shopify) y una tecnología ultra rápida para el diseño (Astro).

📝 6. Práctica: Añadir una Insignia (Badge) de Shadcn UI

Vamos a usar **Shadcn UI**. El objetivo es añadir una etiqueta de "OFERTA" encima del nombre de nuestras pizzas.

🔍 ¿Por qué no usamos la etiqueta <Badge> de la web?

Si vas a la web de Shadcn ([Badge Docs](#)) verás que usan <Badge>. En Shadcn, **tú eres el dueño del código**, así que hoy vamos a usar el atajo profesional: **copiar las clases de estilo de Tailwind**.

🌐 Elige tu Estilo (Copia uno de estos códigos)

Opción 1: Rojo "Oferta" 💧

```
<div className="inline-flex items-center rounded-full bg-red-500 px-2.5 py-0.5 text-xs font-semibold text-white mb-2 uppercase tracking-wider">
  ¡Oferta!
</div>
```

Opción 2: Negro "Nuevo" ★

```
<div className="inline-flex items-center rounded-full bg-zinc-900 px-2.5 py-0.5 text-xs font-semibold text-white mb-2">
  ¡Nuevo!
</div>
```

👤 7. Instrucciones de instalación en la web

1. Abre: [src/layouts/functional-components/ProductGrid.tsx](#).
2. Ve a la **Línea 179**: donde pone `{product?.title}`.
3. Justo **encima** del título, pega el código de tu Badge.

Debería quedarte así:

```
<div className="py-2 md:py-4 text-center z-20">

  /* TU CÓDIGO AQUÍ */
  <div className="inline-flex items-center ...">¡OFERTA! 💧 </div>

  <h2 className="font-medium text-base ...">
    <a ...>{product?.title}</a>
```

```
</h2>
/* ... */
```