El Ecosistema Conectado

Comunicación con APIs, Asincronía y Flujo de Datos en React

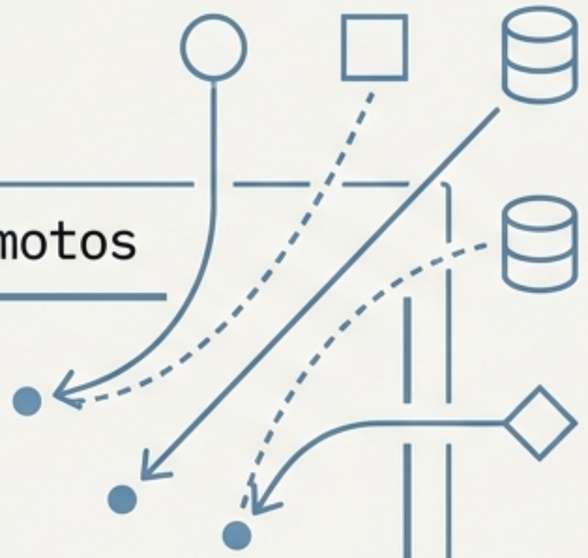
Unidad 4 – Programación Visual

El Límite de las Aplicaciones Aisladas

Datos Locales

- Contenido estático escrito en el código base.
- Sin actualizaciones en tiempo real.
- Inviable para tiendas, redes sociales o bancos.

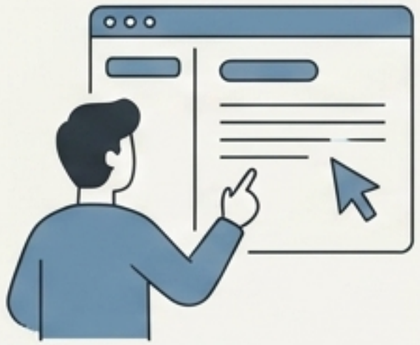
Datos Remotos



- Datos provenientes de servidores externos.
- Actualización constante y dinámica.
- La base del desarrollo web moderno.

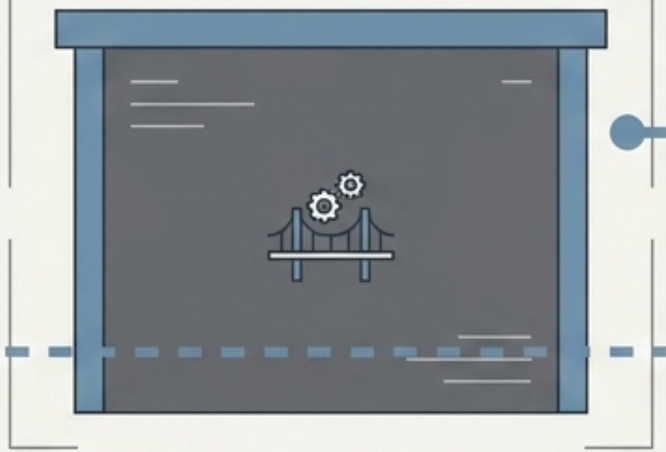
La API: El Puente de Abstracción

Cliente (React)



El usuario solicita información simple.

La API (El Intermediario)



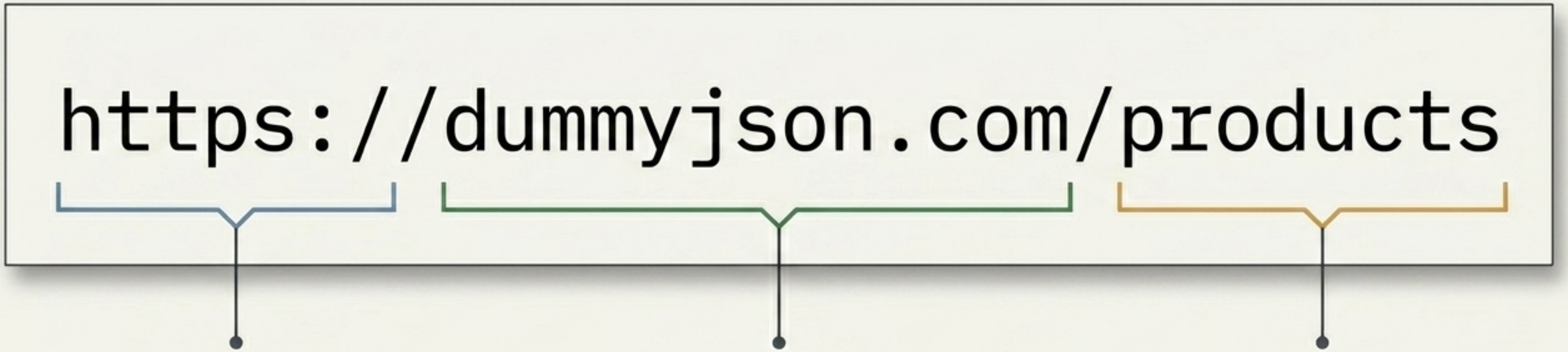
Recibe el pedido y oculta la complejidad. No revela cómo se preparan los datos.

Servidor / Base de Datos

Procesa, consulta y devuelve el resultado.

Una API permite que dos sistemas intercambien información sin conocer cómo están implementados internamente.

Anatomía de un Endpoint REST



Protocolo

El medio de transporte (HTTP Seguro).

Servidor

El dominio físico donde reside la información.

Recurso (Endpoint)

El conjunto de datos específico que deseamos consultar.

Las APIs REST organizan la información en URLs predecibles.

El Vocabulario de la Red: Métodos HTTP

Método	Acción (CRUD)	¿Incluye Cuerpo (Body)?
GET	Leer / Obtener Datos	No
POST	Crear Nueva Información	Sí
PUT	Actualizar Información Existente	Sí
DELETE	Eliminar Información	No

En esta clase nos enfocaremos exclusivaments exclusivamente en GET para consumir información de la API.

JSON: El Formato de Intercambio Universal

```
{
  "id": 1,
  "title": "Notebook",
  "price": 1200
}
```

response.json()

Beneficios en React

- Liviano y fácil de leer
- Compatible con múltiples lenguajes
- JavaScript lo convierte a objetos nativos al instante

JSON (JavaScript Object Notation) es el estándar moderno para transportar datos entre Cliente y Servidor.

Fetch y el Poder de la Asincronía

Síncrono - Bloqueante

El navegador se congela esperando la respuesta.

Asíncrono - React

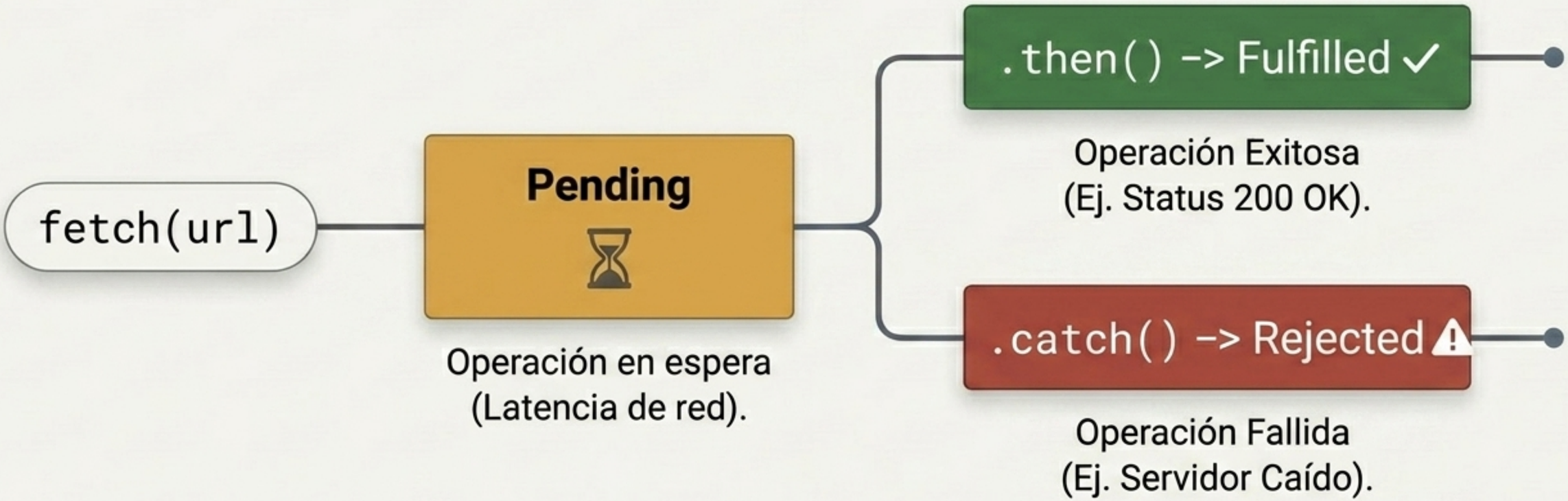
`fetch()`

La interfaz sigue interactiva.

Espera de red en segundo plano.

La función nativa `fetch()` realiza peticiones HTTP de forma asíncrona, evitando que la aplicación se bloquee por la latencia de la red.

La Anatomía de una Promesa



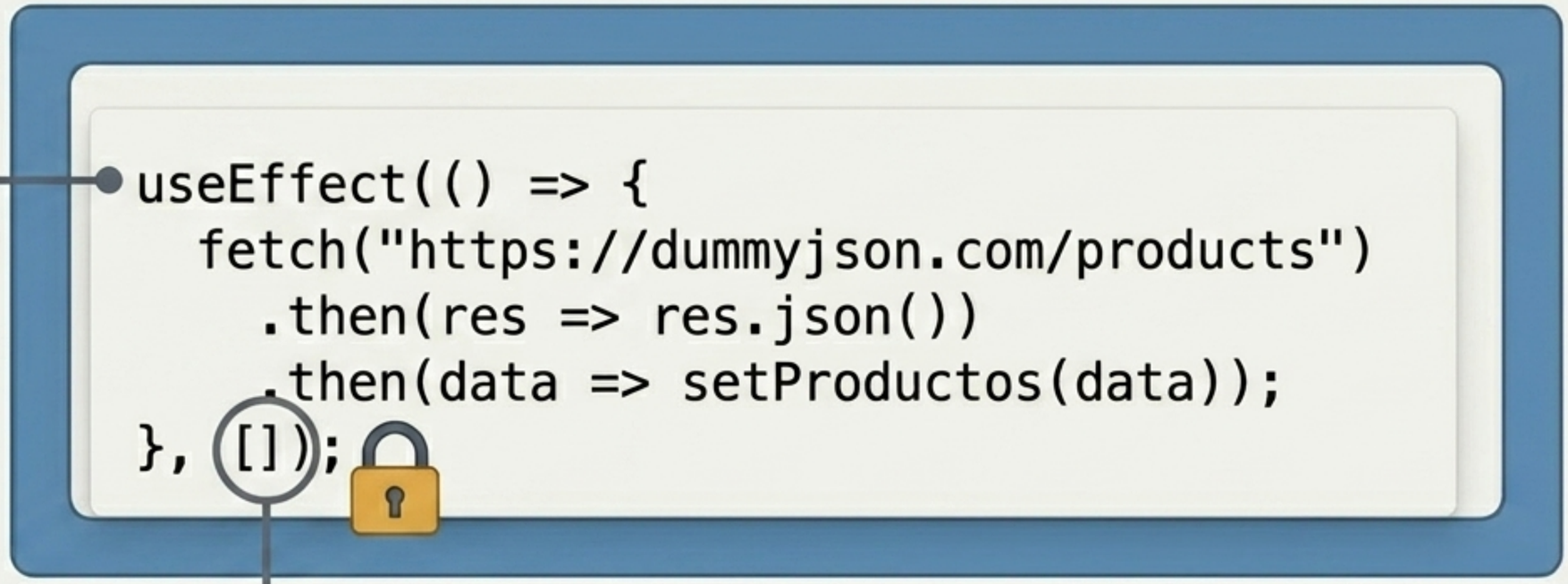
Una Promesa en JavaScript es el compromiso de que una operación futura tendrá un resultado.

El Peligro del Renderizado Infinito


```
function App() {
  fetch('https://api.com/data');
  return <Component />;
}
```

Ejecutar peticiones externas sin control satura la red y destruye el rendimiento.

useEffect: El Controlador de Ciclo



Aísla el efecto secundario del renderizado principal.

El arreglo de dependencias vacío asegura que el código se ejecute una sola vez al montar el componente en pantalla.

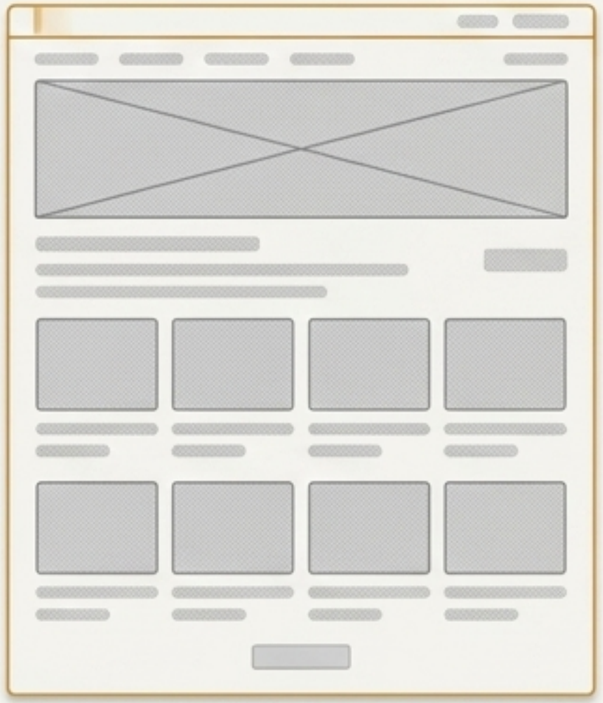
Matriz de Estados de Interfaz

Variable

`loading: true`

Context

Esperando latencia de red.

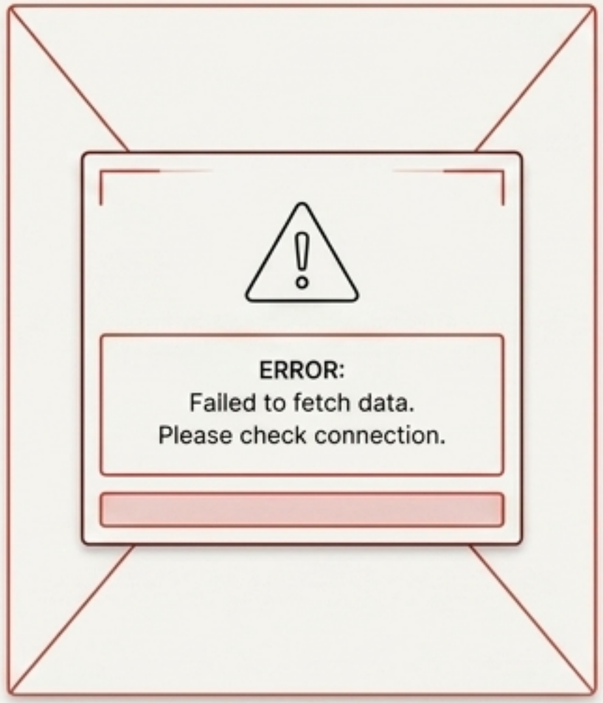


Variable

`error: true`

Context

Servidor caído o Error 404.



Variable

`data: [...]`

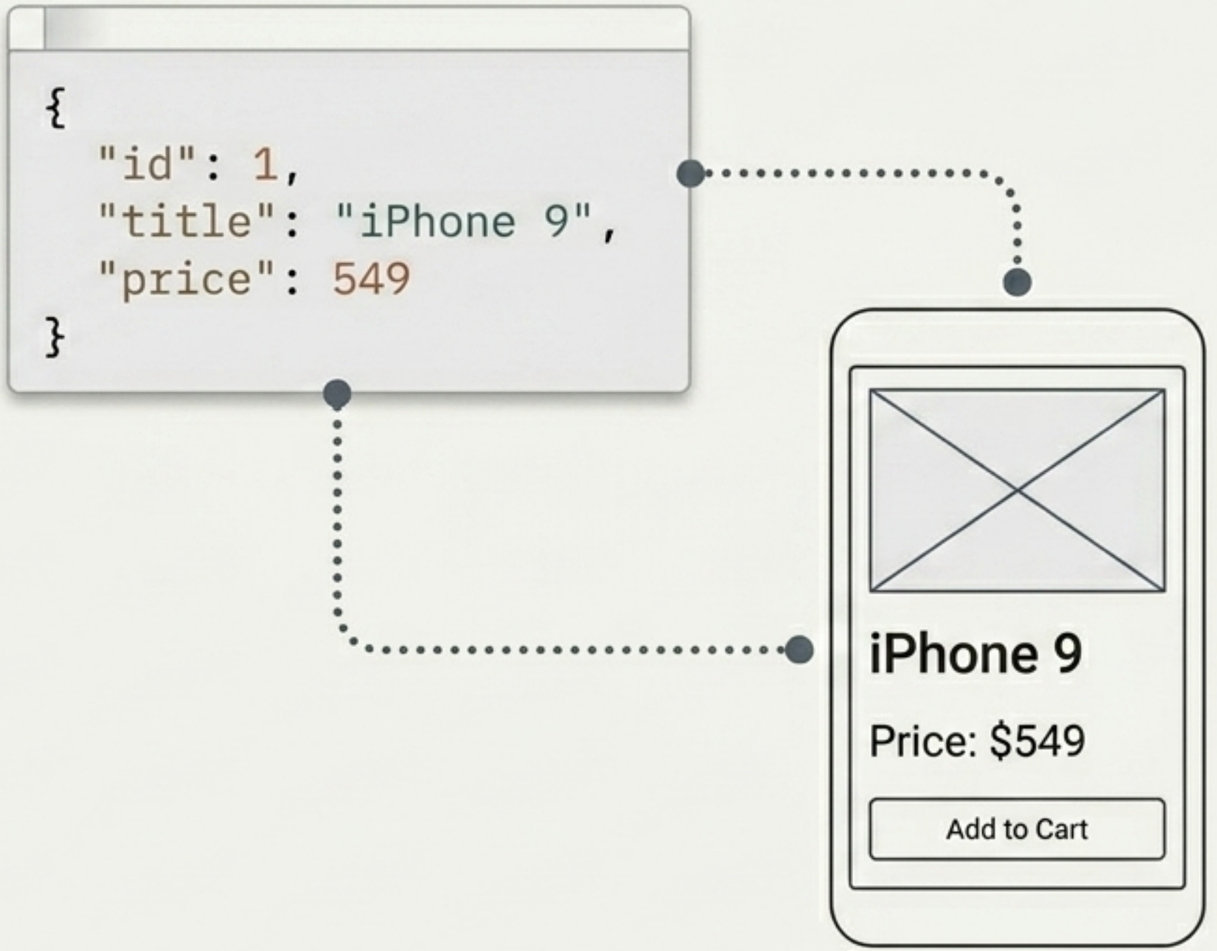
Context

Promesa Resuelta (200 OK).

Nuestro Entorno de Práctica: DummyJSON

- API pública gratuita.
- No requiere configuración de backend propio.
- Incluye: Productos, imágenes, usuarios y paginación simulada.

Data to UI Mapping



DevTools: Haciendo Visible lo Invisible

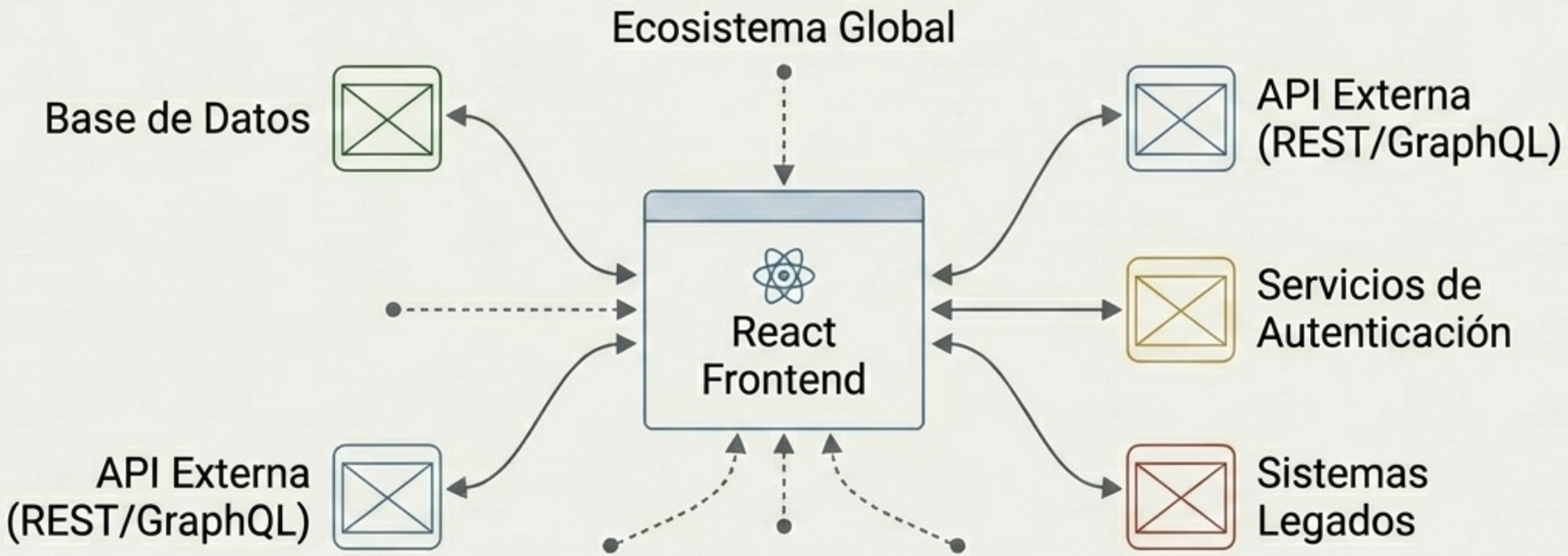
Browser Console Reapons Network F12

Name	Status	Type	Time	Size
styles.css	200 OK	stylesheet	10ms	
products	200 OK	fetch	120ms	4.2 KB
main.js	200 OK	script	30ms	5.9 KB
analytics	404 Not Found	xhr	15ms	

La herramienta de red permite inspeccionar la comunicación real: peticiones (**requests**), respuestas (**responses**), latencia temporal y códigos de estado.

El Flujo Reactivo Completo

Más Allá de la Interfaz Estática



Las aplicaciones modernas no son documentos aislados. Son interfaces vivas capaces de comunicarse, gestionar asincronía y reaccionar a un mundo de datos externos.

Próximos pasos: Axios, Autenticación y APIs Protegidas.