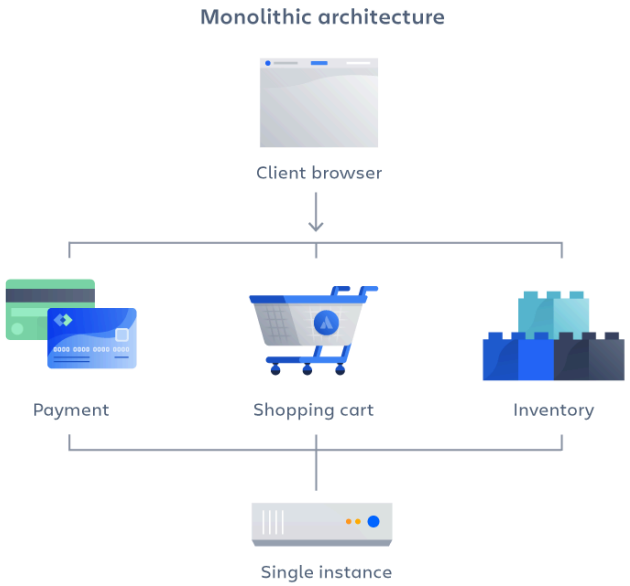
# Desarrollo de descuentos en Ecommerce usando patrones de diseño - Parcial #2

**Estudiantes:** Alfonso Murillo Lora y Sebastian Medina Garcia

## ¿Por qué usamos un monolito para este ejemplo?

Si estamos hablando de desarrollar una plataforma de ecommerce con Next.js, optar por una arquitectura monolítica puede ser una buena decisión técnica, sobre todo en las primeras etapas de un proyecto grande. Este enfoque permite tener todos los componentes del sistema la interfaz de usuario, la lógica de negocio, el backend y la conexión a la base de datos en un mismo repositorio y entorno de ejecución. Esto hace que sea más fácil organizar el código, comunicar los distintos módulos entre sí y mantener el control general del sistema. Es especialmente útil cuando se trabaja con un equipo pequeño o cuando el producto aún está en proceso de validación en el mercado.

Next.js, como framework fullstack basado en React, encaja perfectamente con esta idea de un "monolito moderno". Gracias a sus API Routes, se pueden construir endpoints del lado del servidor directamente dentro del mismo proyecto donde se desarrolla la interfaz. Esto evita tener que separar el backend en otro proyecto, lo que simplifica bastante el desarrollo y mejora la integración entre las distintas partes del sistema. Además, si usamos herramientas como Prisma (para manejar bases de datos), NextAuth (para autenticación) y Stripe (para pagos) se integrarían sin problemas en este entorno unificado, lo que facilita el trabajo del equipo y acelera el desarrollo según nuestra experiencia.



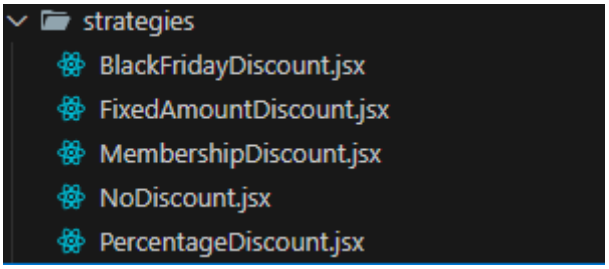
## Aplicación de los Patrones Strategy & Singleton en un Ecommerce

En el desarrollo de un ecommerce, usar patrones de diseño como Strategy y Singleton nos pareció una respuesta adecuada para la parte que se quería mostrar en este ejemplo.

Por ejemplo, el patrón Strategy es ideal para manejar diferentes tipos de descuentos que se pueden aplicar a los productos. En un entorno como el ecommerce de moda, es común tener distintas promociones conviviendo al mismo tiempo: descuentos por porcentaje, rebajas fijas, ofertas para usuarios con suscripciones o campañas como Black Friday. Encapsular cada una de estas reglas como una estrategia independiente permite separar esta lógica del resto del sistema, como el catálogo o la interfaz. Así, se pueden aplicar promociones de forma dinámica sin tener que tocar el código principal, lo que hace mucho más fácil escalar o ajustar las campañas según lo que pida el mercado tal como lo indica este patrón.

```
1  //Este es el contexto de descuento que se encarga de aplicar la estrategia de descuento que se vaya a usar
2  export class DiscountContext {
3    constructor(strategy) {
4      this.strategy = strategy;
5    }
6
7    setStrategy(strategy) {
8      this.strategy = strategy;
9    }
10
11   calculate(price) {
12     return this.strategy.applyDiscount(price);
13   }
14 }
```

Como se decía anteriormente, permite tener los distintos descuentos necesarios.



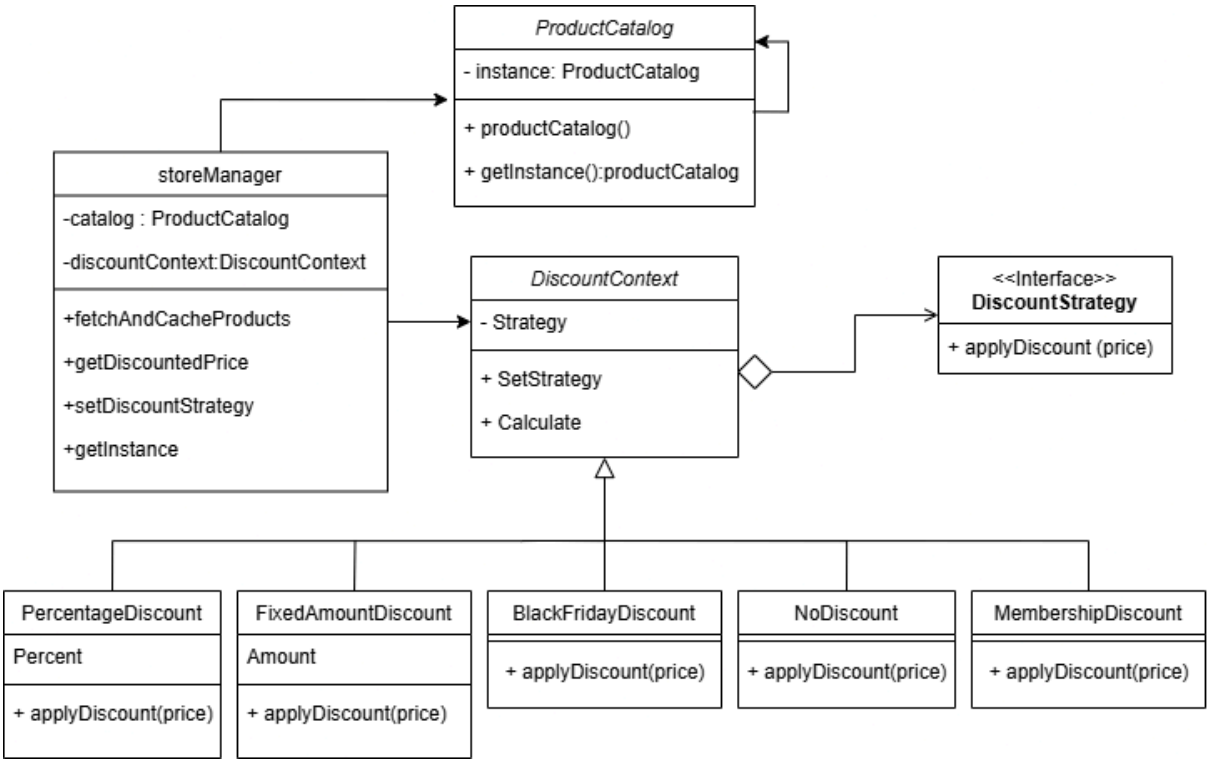

Por otro lado, el patrón Singleton se usa para asegurar que el catálogo de productos tenga una única instancia en toda la aplicación. Esto es clave porque distintos módulos como la interfaz, los descuentos o el sistema de inventario acceden a los mismos productos. Tener un catálogo centralizado evita duplicaciones, errores por datos desactualizados y asegura que cualquier cambio (precio, stock, descripción, etc.) se vea reflejado de inmediato en todo el sistema. En ese sentido, el Singleton funciona como un punto único de acceso al catálogo, manteniendo todo sincronizado, lo que responde a una de las posibles necesidades que resuelve este patrón.

```
// PATRÓN SINGLETON aplicado aquí para mantener una única instancia del catálogo de productos
export class ProductCatalog {
  static instance;
  products = [];

  constructor() {
    if (ProductCatalog.instance) {
      return ProductCatalog.instance;
    }
    ProductCatalog.instance = this;
  }

  static getInstance() {
    if (!ProductCatalog.instance) {
      ProductCatalog.instance = new ProductCatalog();
    }
    return ProductCatalog.instance;
  }
```

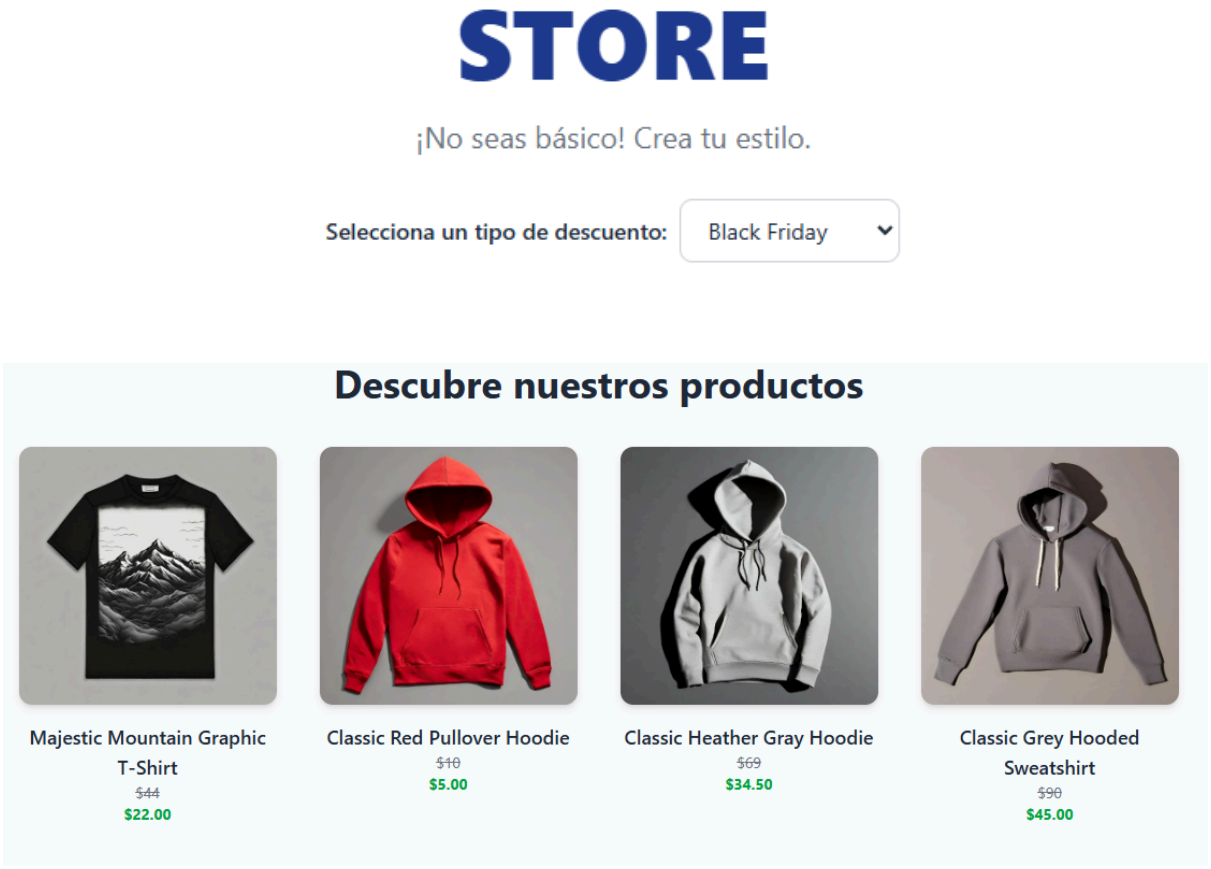
Al combinarlos, estos dos patrones ayudan a resolver dos aspectos fundamentales en un ecommerce: por un lado, la flexibilidad para manejar promociones con Strategy, y por otro, la consistencia del inventario con Singleton. Juntos ofrecen una base sólida, adaptable y alineada con las necesidades del proyecto, esa fue la razón por la que decidimos que aplicaban a este proyecto.



## Análisis del Antipatrón God Object en la Clase StoreManager

En la arquitectura actual del ecommerce, la clase StoreManager representa un caso evidente del antipatrón God Object. Este antipatrón surge cuando una sola clase asume múltiples responsabilidades dentro del sistema, concentrando lógica que debería estar distribuida en componentes más específicos y cohesionados. En el caso concreto de StoreManager, este objeto central no solo gestiona el catálogo de productos y la aplicación de descuentos, sino que también se encarga del fetch de productos, el almacenamiento en caché, el cálculo de precios con descuento y la selección de estrategias de negocio. Esta acumulación de responsabilidades va en contra del principio de responsabilidad única (SRP) y compromete directamente la mantenibilidad y escalabilidad del sistema.

Desde un enfoque técnico, el problema radica en que StoreManager actúa como intermediario de múltiples dominios del sistema: controla tanto la lógica de inventario (a través de ProductCatalog) como la lógica promocional (a través del contexto de descuentos que viene en el patrón strategy). Esta concentración de lógica genera un alto acoplamiento y dificulta la evolución independiente de cada módulo. Por ejemplo, si se desea modificar la forma en que se obtienen los productos, o introducir nuevas estrategias promocionales más complejas (como descuentos combinados o condicionales), los cambios deberán pasar necesariamente por esta clase. Todo esto ralentiza incluso el proceso de pruebas y ni hablar de cómo podríamos aplicar la reutilización de componentes.



# VIERNES DE DESCUENTOS PARA TODOS * ¡APROVECHA!

## Recompensa de la Marca

Originado en América, nuestro producto ha sido nominado como Marca del Año. Con tu apoyo, ganamos esta nominación. Costy comenzó con una idea: hacer moda que los jóvenes quieran usar explorando colores audaces que reflejen la verdadera individualidad.