

Informe: Patrones de Diseño - Sistema KairosMix1. Introducción

Este informe resume los patrones de diseño implementados en el sistema **KairosMix**, un gestor de productos desarrollado con **React** (frontend) y **Express/TypeScript** (backend). El objetivo es describir las buenas prácticas adoptadas para el desarrollo de software mantenible, escalable y testeable.

2. Patrones de Diseño Identificados y Clasificación

Patrón	Tipo	Ubicación Principal
Singleton	Creacional	<code>Database.ts</code> (Backend)
Repository	Estructural/Arquitectural	<code>ProductRepository.ts</code> (Backend)
MVC (Model-View-Controller)	Arquitectural	Backend completo
Module Pattern	Estructural	<code>productService.js</code> , controllers (Frontend/Backend)
Container/Presentational	Estructural (React)	Componentes frontend
Observer	Comportamental	React State (<code>useState/useEffect</code>)

3. Patrones de Diseño Detallados3.1 Patrón Singleton

- **Descripción:** Garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a ella.
- **Implementación en KairosMix:** Se utiliza en la clase `Database.ts` para gestionar la conexión a MongoDB.
- **Beneficios:**
 - **Control de conexión:** Asegura una única conexión a la base de datos.
 - **Gestión de recursos:** Evita múltiples conexiones simultáneas.
 - **Acceso global:** Facilita el acceso a la instancia de la base de datos desde cualquier parte del backend.

```

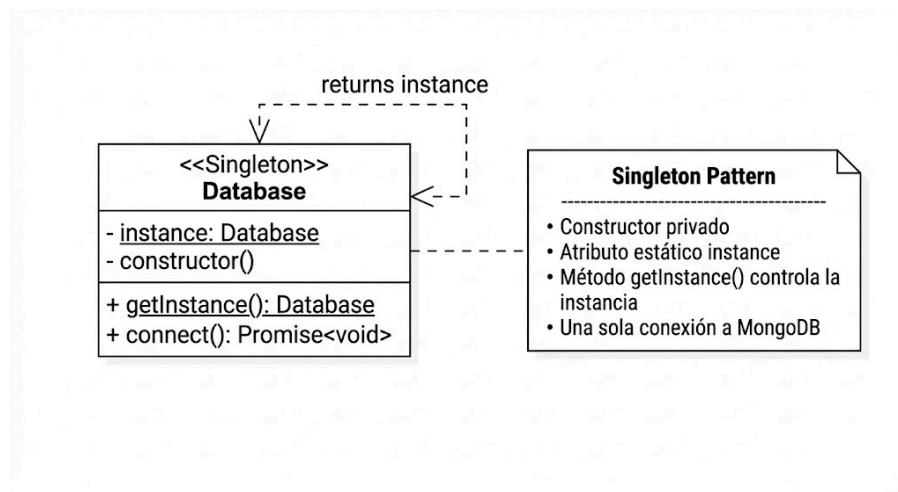
export class Database {
  private static instance: Database; // Instancia única

  private constructor() {} // Constructor privado

  public static getInstance(): Database {
    if (!Database.instance) {
      Database.instance = new Database();
    }
    return Database.instance;
  }

  public async connect(): Promise<void> {
    await mongoose.connect(process.env.MONGO_URI);
  }
}

```

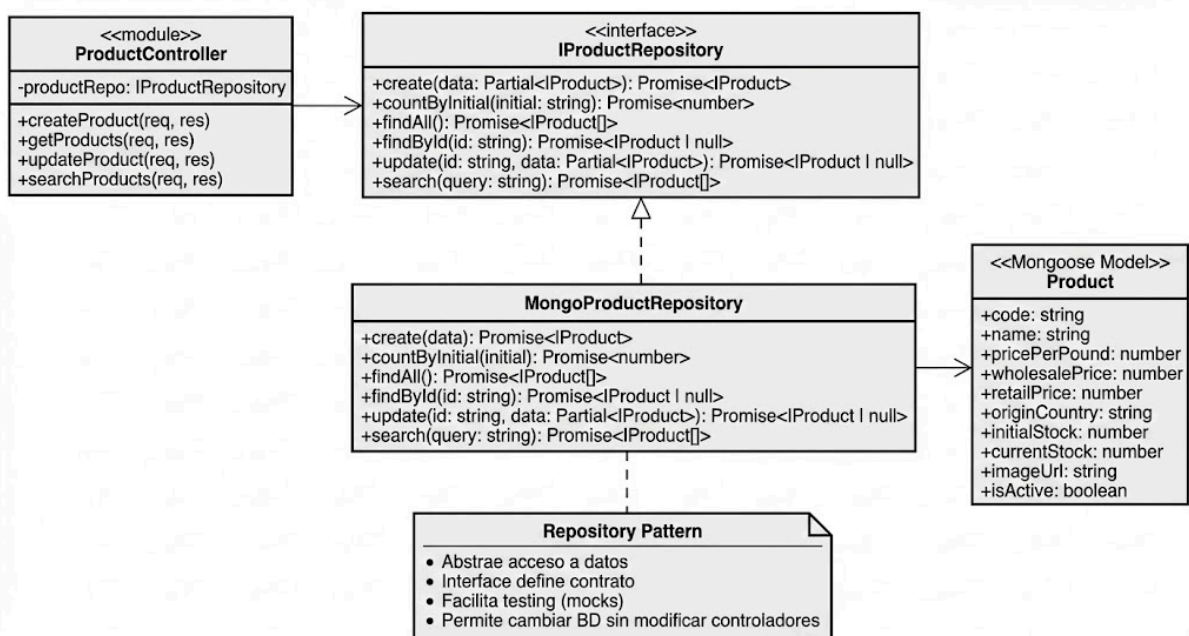


3.2 Patrón Repository

- **Descripción:** Abstrae la lógica de acceso a datos, ofreciendo una interfaz de colección para objetos del dominio, desacoplando la capa de negocio de la capa de persistencia.
- **Implementación en KairosMix:** Implementado mediante la interfaz `IProductRepository` y la clase concreta `MongoProductRepository`.
- **Beneficios:**
 - **Desacoplamiento:** El controlador no tiene dependencia directa de la implementación de MongoDB.
 - **Testabilidad:** Facilita el uso de *mocks* para pruebas unitarias.
 - **Flexibilidad:** Permite cambiar la base de datos subyacente con mínima modificación en otras capas.

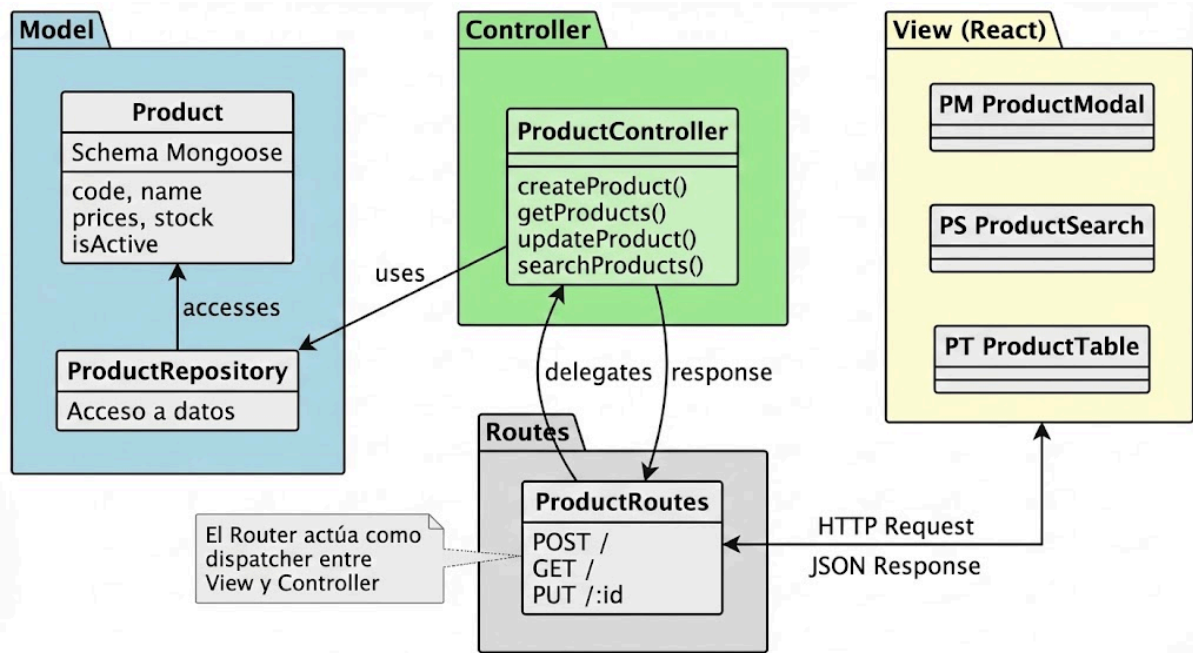
```
// Interface (Contrato)
export interface IProductRepository {
  create(data: Partial<IProduct>): Promise<IProduct>;
  findAll(): Promise<IProduct[]>;
  findById(id: string): Promise<IProduct | null>;
  update(id: string, data: Partial<IProduct>): Promise<IProduct | null>;
  search(query: string): Promise<IProduct[]>;
}

// Implementación concreta
export class MongoProductRepository implements IProductRepository {
  async findAll(): Promise<IProduct[]> {
    return await Product.find({ isActive: true });
  }
  // ... otros métodos
}
```



3.3 Patrón MVC (Model-View-Controller)

- **Descripción:** Separa la aplicación en tres componentes interconectados: Modelo (datos), Vista (interfaz de usuario) y Controlador (lógica de negocio).
- **Implementación en KairosMix (Backend):**
 - **Model:** `models/Product.ts` (esquema de datos y validaciones con Mongoose).
 - **View:** Componentes React (Interfaz de usuario - en el frontend).
 - **Controller:** `controller/productController.ts` (lógica de negocio y manejo de peticiones).



Conclusión

La aplicación KairosMix está diseñada utilizando una variedad de patrones (Creacionales, Estructurales, Comportamentales y Arquitecturales) que aseguran una arquitectura robusta, facilitando tareas cruciales como la gestión de dependencias (Singleton, Repository), la separación de preocupaciones (MVC, Container/Presentational) y la reactividad de la interfaz de usuario (Observer).