# **Estilos y Patrones Arquitectónicos Seleccionados**

## **1. Estilos Arquitectónicos Adoptados**

La arquitectura general del sistema InnoSistemas ha sido diseñada para equilibrar la **modularidad, escalabilidad y mantenibilidad**, manteniendo una estructura clara para el desarrollo, despliegue y evolución del sistema. Se han seleccionado los siguientes estilos arquitectónicos:

### E1. Arquitectura Monolito Modular

El backend se organizará inicialmente como un **monolito modular**, estructurado en capas y paquetes bien definidos que permiten separar responsabilidades por dominios funcionales (gestión de equipos, autenticación, notificaciones, reportes, etc.). Este estilo es adecuado para la fase inicial del proyecto al facilitar el desarrollo, pruebas e integración continua, sin la complejidad operativa de un sistema distribuido.

### E2. Arquitectura Cliente-Servidor Modular con Frontend desacoplado

El sistema adoptará una arquitectura **cliente-servidor**, en la que el frontend está desacoplado del backend y se desarrollará como una aplicación modular basada en **React + Next.js**. Cada módulo funcional del sistema (por ejemplo, gestión de equipos, dashboard de seguimiento, reportes de avance) será tratado como una sección independiente dentro de una estructura unificada, lo cual permite:

* Reutilización de componentes visuales (con Tailwind y TypeScript),
* División lógica del código en rutas/páginas (usando Next.js),
* Independencia en el desarrollo y mantenimiento de módulos funcionales.

### E3. Arquitectura Basada en Contenedores

Toda la solución será empaquetada en **contenedores Docker**, lo que permite portabilidad, consistencia entre entornos y facilidad de automatización. Kubernetes gestionará los servicios en producción, permitiendo escalado horizontal y alta disponibilidad.

## **2. Patrones Arquitectónicos Relevantes**

Se adoptarán varios patrones de diseño y arquitectura que permiten mejorar la organización interna del código, la seguridad, la eficiencia en el acceso a datos y la interoperabilidad:

### P1. Patrón Controlador – Servicio – Repositorio (Capa de abstracción)

El backend seguirá una estructura en capas, con separación clara entre:

* **Controladores** (manejo de rutas y peticiones HTTP),
* **Servicios** (lógica de negocio),
* **Repositorios** (acceso a datos mediante Spring Data JPA).

### P2. Patrón Repositorio

Se utilizará para encapsular el acceso a la base de datos PostgreSQL. Esto permitirá cambiar o extender fácilmente la lógica de persistencia sin afectar otras capas del sistema.

### P3. Patrón Observador (para eventos internos)

Para notificaciones dentro del sistema (por ejemplo, cuando un estudiante se une a un equipo), se aplicará el patrón observador, permitiendo que múltiples componentes reaccionen a un evento sin acoplamiento directo entre ellos.

### P4. Patrón DTO (Data Transfer Object)

Se emplearán DTOs para definir estructuras específicas que serán enviadas desde el backend al frontend, evitando exponer directamente las entidades del modelo de dominio y controlando qué datos viajan entre capas.