



ARQUITECTURA HEXAGONAL

(Y por qué entenderla va a cambiar
tu manera de desarrollar software)



PABLO DEL ÁLAMO



¿Qué es la Arquitectura Hexagonal?

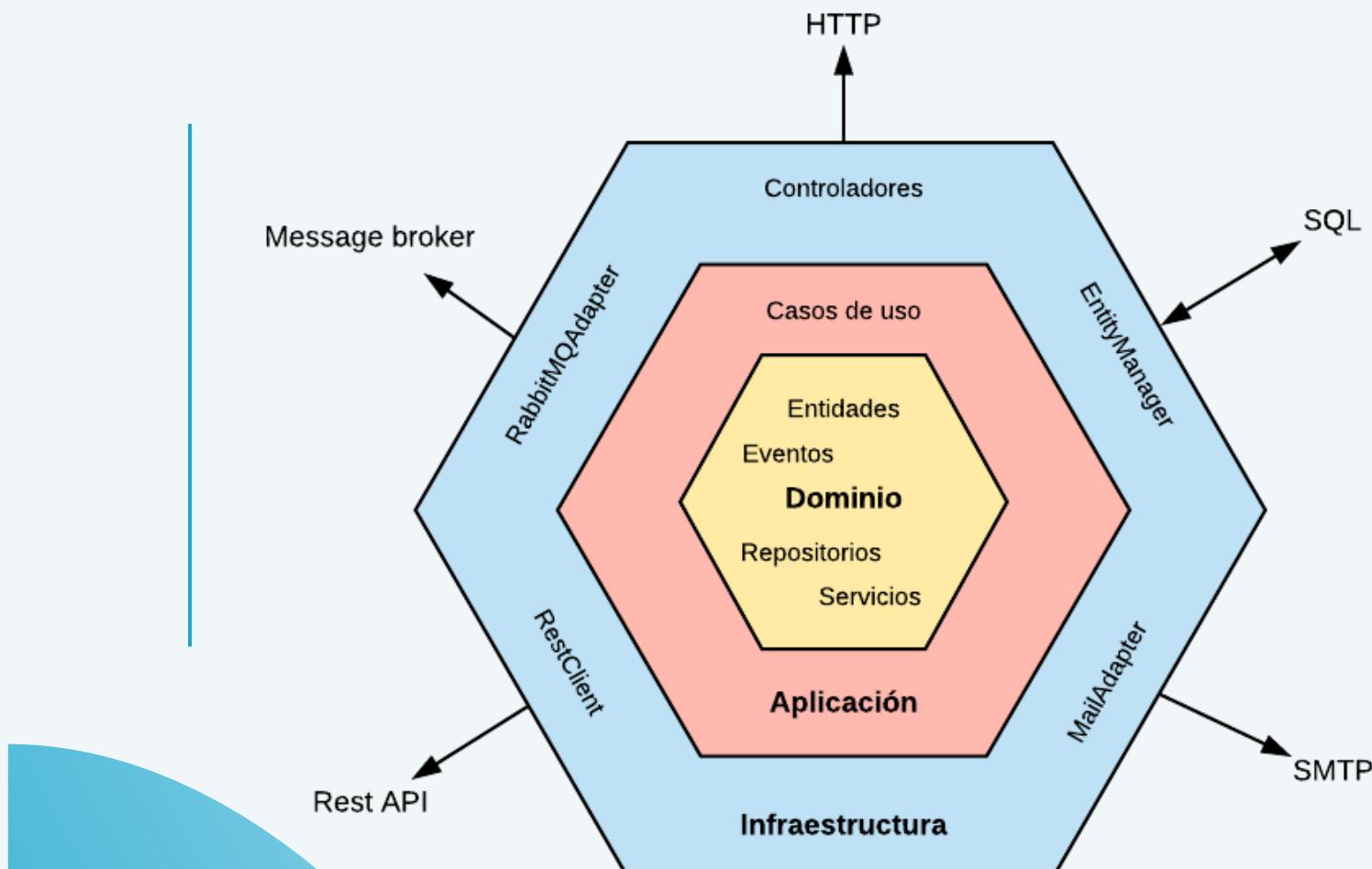
También conocida como Arquitectura Puertos y Adaptadores, se centra en la separación de la lógica de negocio del mundo exterior.



PABLO DEL ÁLAMO



¿Qué es la Arquitectura Hexagonal?



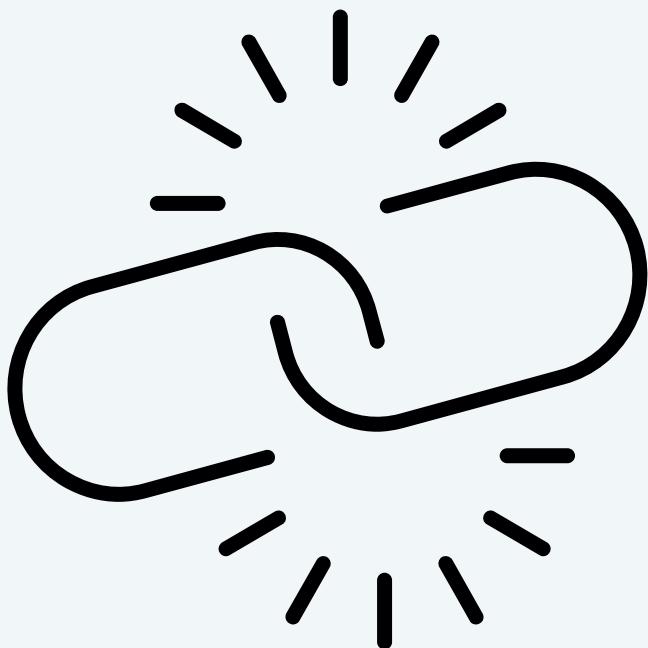
PABLO DEL ÁLAMO



El Problema que Resuelve

¿Estás hart@ de que tu aplicación sea difícil de mantener y probar?

La arquitectura hexagonal ayuda a desacoplar componentes.

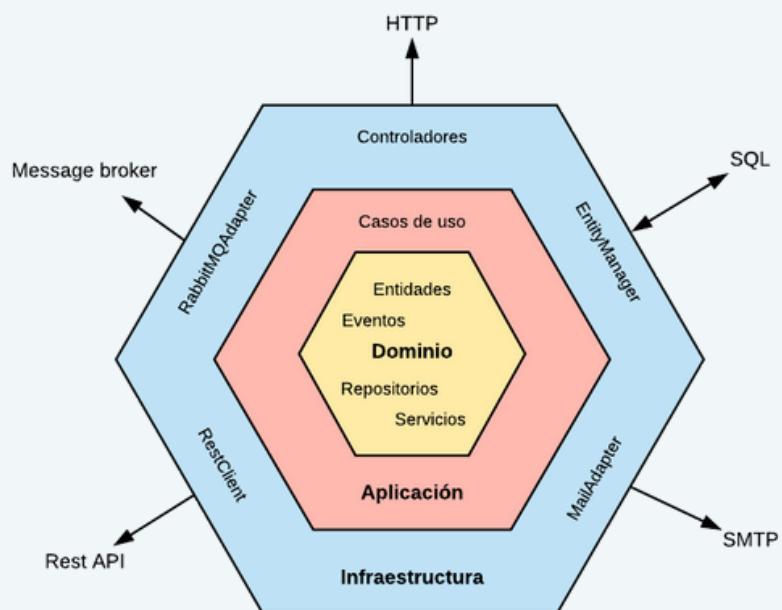


PABLO DEL ÁLAMO



La Metáfora del Hexágono

El hexágono representa un núcleo central con puertos y adaptadores hacia el exterior, como una colmena

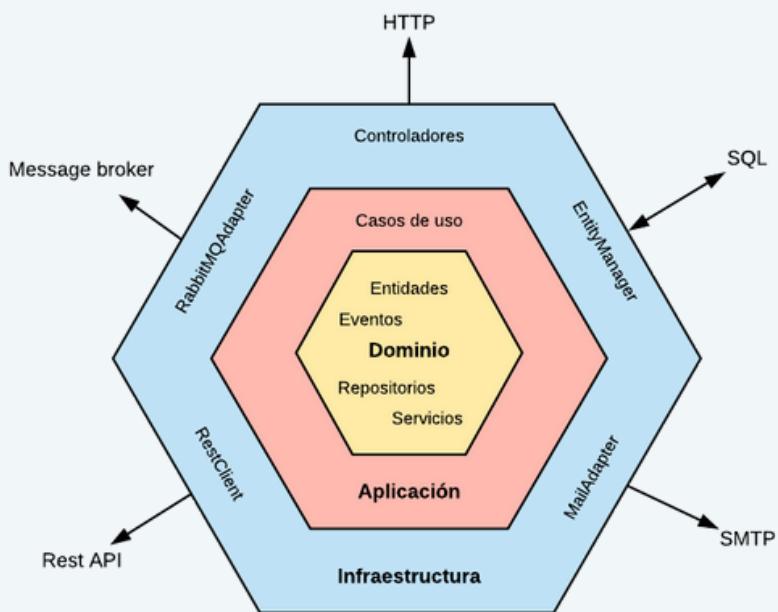


PABLO DEL ÁLAMO



Estructura Básica

Imagina tu aplicación estructurada con tres capas principales: Dominio, Aplicación e Infraestructura.



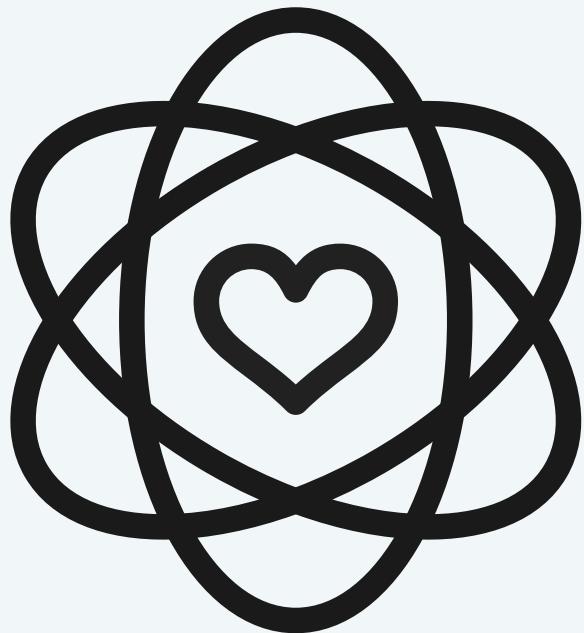
PABLO DEL ÁLAMO



1

La Capa de Dominio

Aquí reside la lógica core de tu negocio.
Modela las reglas y conceptos clave de tu
aplicación.

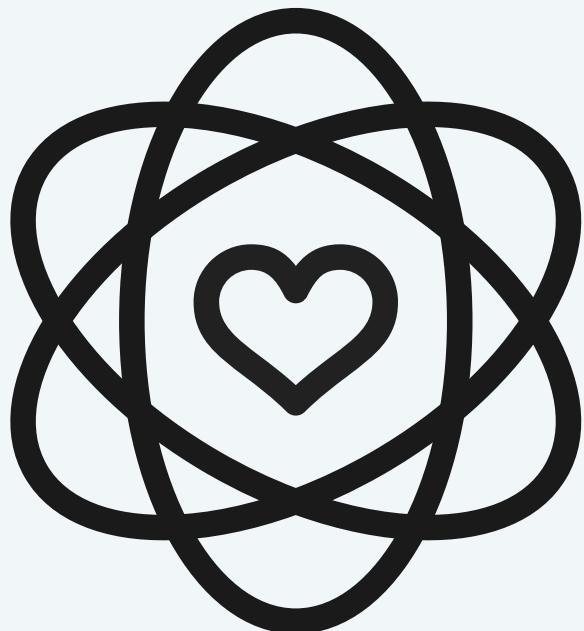


PABLO DEL ÁLAMO



La Capa de Dominio

Las clases en esta capa representan las reglas del negocio (entidades, objetos de valor, agregados), las operaciones que las manipulan (servicios de dominio, repositorios) y eventos importantes (eventos de dominio) que ocurren en el contexto de ese negocio.



PABLO DEL ÁLAMO



Ejemplo de Dominio

Ejemplos de clases que van en la capa de dominio en un sistema bancario:

- **CuentaBancaria:** representa una cuenta bancaria individual. Puede contener comportamientos como depósitos, retiros y verificaciones de saldo.
- **Dinero:** El objeto de valor Dinero se usa para representar cantidades monetarias y podría incluir lógica para manejar divisas y cálculos relacionados con dinero.
- **TransferenciaBancaria:** sería un agregado que involucra dos cuentas bancarias (origen y destino) y el objeto de valor Dinero. Gestiona la lógica de transferir fondos entre cuentas.



PABLO DEL ÁLAMO



Ejemplo de Dominio

- **RepositorioDeCuentas:** define las operaciones que deben realizarse sobre las cuentas bancarias, como obtener una cuenta por su número de cuenta. Solo es la interfaz; la implementación concreta estaría en una capa externa (adaptador) que maneja la persistencia (e.g., en una base de datos)
- **ServicioDeTransferencias:** contiene la lógica para gestionar la transferencia de fondos entre cuentas bancarias, interactuando con el agregado de TransferenciaBancaria y el repositorio.
- **Evento de dominio:** representa algo que ha sucedido en el dominio que es importante para otros componentes del sistema. En este caso, cuando una transferencia es completada, se podría generar un evento que informe a otros servicios.



PABLO DEL ÁLAMO



2

La Capa de Aplicación

Coordina la actividad de las capas.
Define casos de uso para reglas del dominio.

Tiene el propósito de coordinar las operaciones entre las diferentes capas, especialmente entre la capa de dominio y las interfaces externas.

Esta capa no contiene lógica de negocio (esa reside en la capa de dominio), sino que orquesta el flujo de las acciones que el sistema debe realizar.



PABLO DEL ÁLAMO



Ejemplos Capa de Aplicación

Algunas clases que podrían ir en la capa de aplicación dentro del ejemplo que estamos siguiendo serían:

- ServicioDeTransferencias
- DTOs (Data Transfer Objects)



PABLO DEL ÁLAMO



ServicioDeTransferencias (TransferService)

Este servicio coordina el proceso de transferir fondos entre cuentas.

Llama a los métodos en la capa de dominio para realizar la lógica de negocio.



PABLO DEL ÁLAMO

```
1 class ServicioDeTransferencias {
2     private readonly cuentaRepository: RepositorioDeCuentas;
3     private readonly transferenciaService: ServicioDeTransferencias;
4
5     constructor(cuentaRepository: RepositorioDeCuentas, transferenciaService:
6         ServicioDeTransferencias) {
7         this.cuentaRepository = cuentaRepository;
8         this.transferenciaService = transferenciaService;
9     }
10
11     public async realizarTransferencia(numeroCuentaOrigen: string,
12         numeroCuentaDestino: string, monto: Dinero): Promise<void> {
13         // Lógica para validar datos de entrada, como comprobar que el monto es
14         // positivo, etc.
15
16         // Obtener las cuentas del repositorio
17         const cuentaOrigen = await this.cuentaRepository.buscarPorNumeroCuenta
18             (numeroCuentaOrigen);
19         const cuentaDestino = await this.cuentaRepository.buscarPorNumeroCuenta
20             (numeroCuentaDestino);
21
22         // Validar que ambas cuentas existen
23         if (!cuentaOrigen || !cuentaDestino) {
24             throw new Error('Una de las cuentas no fue encontrada');
25         }
26
27         // Crear la transferencia y ejecutarla
28         const transferencia = new TransferenciaBancaria(cuentaOrigen,
29             cuentaDestino, monto);
30         transferencia.ejecutar();
31
32         // Guardar las cuentas actualizadas en el repositorio
33         await this.cuentaRepository.guardar(cuentaOrigen);
34         await this.cuentaRepository.guardar(cuentaDestino);
35     }
36 }
```



DTOs (Data Transfer Objects)

Estos son objetos que se utilizan para transportar datos entre la capa de presentación y la capa de aplicación.



PABLO DEL ÁLAMO



```
1 class TransferenciaDTO {  
2     public cuentaOrigen: string;  
3     public cuentaDestino: string;  
4     public monto: number;  
5  
6     constructor(cuentaOrigen: string, cuentaDestino: string, monto: number) {  
7         this.cuentaOrigen = cuentaOrigen;  
8         this.cuentaDestino = cuentaDestino;  
9         this.monto = monto;  
10    }  
11 }  
12 |
```



PABLO DEL ÁLAMO



3

La Capa de Infraestructura

Gestiona tecnología específica:
persistencia de datos, APIs
externas, redes, etc.



PABLO DEL ÁLAMO



- **Persistencia de Datos:** Maneja la interacción con bases de datos, archivos, o sistemas externos para almacenar y recuperar datos.
- **Integraciones Externas:** Se encarga de la comunicación con servicios externos, como APIs de terceros, servicios de mensajería, o sistemas de pago.
- **Configuración de la Aplicación:** Puede manejar la configuración y la inicialización de componentes del sistema.
- **Detección de Errores y Logging:** Implementa mecanismos para el manejo de errores y el registro de eventos (logging) que son críticos para el monitoreo y la depuración.



PABLO DEL ÁLAMO

Ejemplos Clases Capa de Infraestructura

- **Repositorio de Datos:** Implementación de las interfaces de repositorio que interactúa con la base de datos.
- **Cliente de API Externa:** Clase que se encarga de la comunicación con APIs de terceros.
- **Adaptador de Mensajería:** Clase que maneja la comunicación a través de un sistema de mensajería.
- **Configuración de la Aplicación:** Clase que gestiona la carga y acceso a la configuración de la aplicación.
- **Servicio de Logging:** Clase que se encarga de registrar errores e información relevante para el monitoreo del sistema.



PABLO DEL ÁLAMO



4

Adaptadores

Los adaptadores actúan como traductores entre las capas y el mundo exterior (como interfaces de usuario, bases de datos, servicios de terceros, etc.).



PABLO DEL ÁLAMO



Tipos de Adaptadores

- **Adaptadores de Entrada (Input Adapters):**
Se encargan de recibir las solicitudes externas y traducirlas a un formato que la aplicación pueda entender. Ejemplos incluyen controladores de API REST, interfaces de usuario, y manejadores de eventos.
- **Adaptadores de Salida (Output Adapters):**
Se encargan de enviar las respuestas de la aplicación hacia el exterior y traducirlas al formato esperado por el sistema externo. Ejemplos incluyen conexiones a bases de datos, servicios de mensajería, y llamadas a APIs externas.



PABLO DEL ÁLAMO



Ejemplos de Adaptadores

- Adaptador de Entrada: Controlador REST (REST Controller)
- Adaptador de Salida: Repositorio de Cuentas (Account Repository)
- Adaptador de Mensajería: Adaptador de RabbitMQ (RabbitMQ Adapter)



PABLO DEL ÁLAMO

Adaptador de Entrada: Controlador REST (REST Controller)

```
1 import { Request, Response } from 'express';
2
3 class TransferenciasController {
4     private readonly servicioDeTransferencias: ServicioDeTransferencias;
5
6     constructor(servicioDeTransferencias: ServicioDeTransferencias) {
7         this.servicioDeTransferencias = servicioDeTransferencias;
8     }
9
10    public async transferirFondos(req: Request, res: Response): Promise<void> {
11        const { cuentaOrigen, cuentaDestino, monto } = req.body;
12
13        try {
14            await this.servicioDeTransferencias.realizarTransferencia
15                (cuentaOrigen, cuentaDestino, new Dinero(monto, 'USD'));
16            res.status(200).send({ message: 'Transferencia realizada con éxito' });
17        } catch (error) {
18            res.status(400).send({ message: error.message });
19        }
20    }
21 }
```



PABLO DEL ÁLAMO

Adaptador de Salida: Repositorio de Cuentas (Account Repository)

```
1 import {getRepository} from 'typeorm';
2 import {CuentaBancariaEntity} from '../entities/CuentaBancariaEntity';
3
4 class RepositorioDeCuentasDB implements RepositorioDeCuentas {
5   public async guardar(cuenta: CuentaBancaria): Promise<void> {
6     const cuentaEntity = new CuentaBancariaEntity();
7     cuentaEntity.numeroCuenta = cuenta.numeroCuenta;
8     cuentaEntity.titular = cuenta.titular;
9     cuentaEntity.saldo = cuenta.obtenerSaldo();
10
11     await getRepository(CuentaBancariaEntity).save(cuentaEntity);
12   }
13
14   public async buscarPorNumeroCuenta(numeroCuenta: string): Promise
15     <CuentaBancaria | null> {
16     const cuentaEntity = await getRepository(CuentaBancariaEntity).findOne(
17       { where: { numeroCuenta } });
18
19     if (!cuentaEntity) {
20       return null;
21     }
22
23     return new CuentaBancaria(cuentaEntity.numeroCuenta, cuentaEntity
24       .titular, cuentaEntity.saldo);
25   }
26 }
```



Adaptador de Mensajería: Adaptador de RabbitMQ (RabbitMQ Adapter)

```
import amqp from 'amqplib';

class AdaptadorMensajeriaRabbitMQ {
    private connection: amqp.Connection;
    private channel: amqp.Channel;

    constructor(url: string) {
        this.connect(url);
    }

    private async connect(url: string) {
        this.connection = await amqp.connect(url);
        this.channel = await this.connection.createChannel();
    }

    public async enviarMensaje(cola: string, mensaje: string): Promise<void> {
        await this.channel.assertQueue(cola);
        this.channel.sendToQueue(cola, Buffer.from(mensaje));
        console.log(`Mensaje enviado a la cola ${cola}: ${mensaje}`);
    }

    public async recibirMensajes(cola: string, callback: (mensaje: string) => void): Promise<void> {
        await this.channel.assertQueue(cola);
        this.channel.consume(cola, (msg) => {
            if (msg) {
                const mensaje = msg.content.toString();
                callback(mensaje);
                this.channel.ack(msg); // Confirma que el mensaje ha sido
                // procesado
            }
        });
    }
}
```

PABLO DEL ÁLAMO





¿Te ha resultado útil?



- Comparte esta guía con tu equipo o amigos desarrolladores.
- Guárdala para tenerla siempre a mano.
- ¡Dale un like o comenta si tienes preguntas!

