# Presentación 1 – Arquitectura Hexagonal

## Arquitectura de Software


Pontificia Universidad
JAVERIANA
Colombia

**Integrantes:**

Juan Fernando Lesmes

Juan Pablo Sánchez

Santiago Castro

**Presentado a:**

Andrés Armando Sánchez Martín

**Pontificia Universidad Javeriana**

**Facultad de Ingeniería**

**Bogotá D.C.**

**2024**

# Contenido

## Introducción

La arquitectura hexagonal de software, también conocida como arquitectura de puertos y adaptadores o arquitectura de puertos y enchufes, es un patrón de diseño que se centra en la separación de preocupaciones y el modularidad en el desarrollo de software. Esta arquitectura se caracteriza por tener un núcleo central o hexágono, que representa la lógica principal de la aplicación, rodeado por capas de adaptadores o puertos que facilitan la interacción con componentes externos, como bases de datos, interfaces de usuario, servicios web, etc.

## Definición

La arquitectura hexagonal se basa en el principio de invertir las dependencias, lo que significa que las dependencias externas de una aplicación, como la base de datos o las interfaces de usuario, deben ser "inyectadas" en el núcleo de la aplicación en lugar de ser directamente accedidas por él. Esto facilita la prueba unitaria y la sustitución de componentes externos sin modificar la lógica principal de la aplicación.

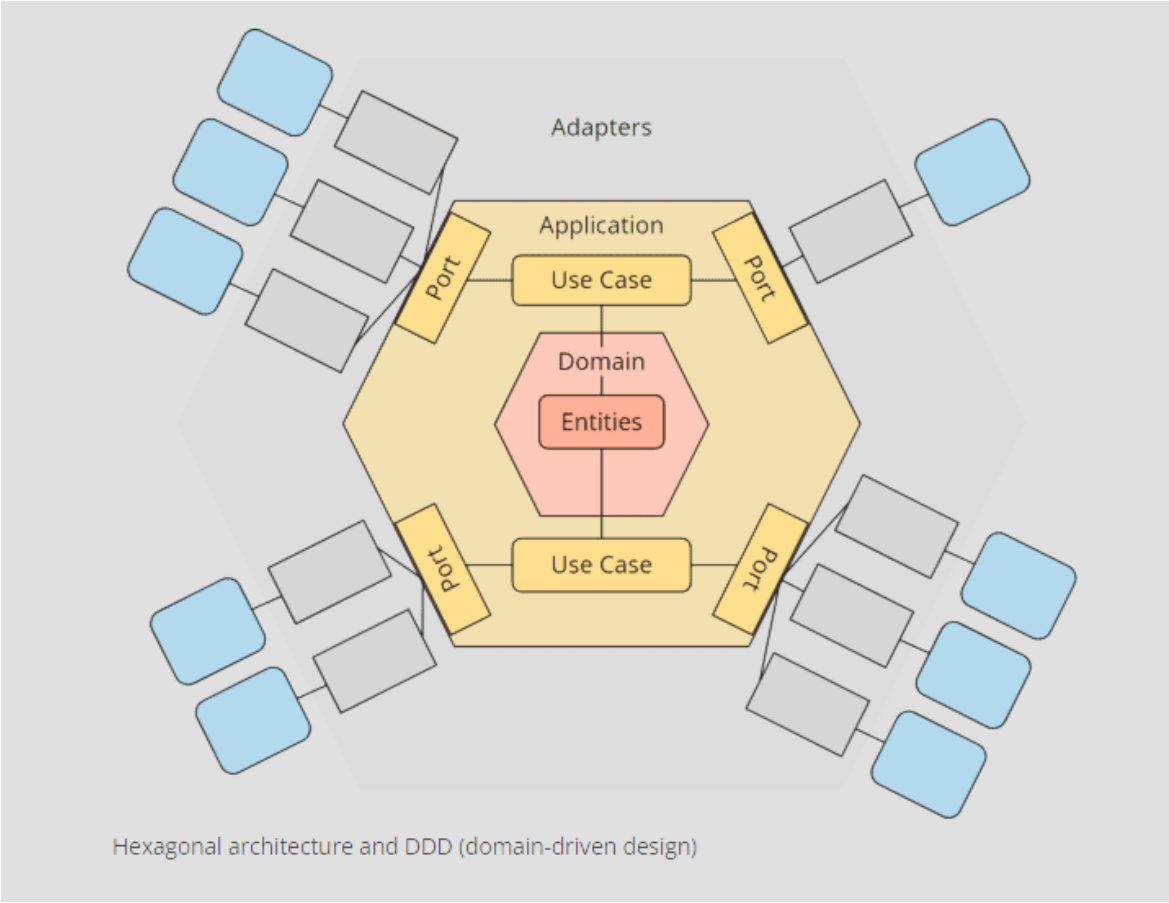


Imagen 1. Diagrama Explicativo Arquitectura Hexagonal.

## Historia

La arquitectura hexagonal fue propuesta por primera vez por Alistair Cockburn en 2005 como parte de su trabajo sobre "arquitecturas de software hexagonal" y ha sido ampliamente adoptada desde entonces en la comunidad de desarrollo de software. Sin embargo, las ideas subyacentes a esta arquitectura tienen raíces en otros enfoques de diseño, como el principio de inversión de control (IoC) y la arquitectura en capas.

## Evolución

A lo largo del tiempo, la arquitectura hexagonal ha evolucionado para adaptarse a diferentes contextos y tecnologías. Por ejemplo, se han desarrollado frameworks y bibliotecas que facilitan la implementación de esta arquitectura en diversos lenguajes de programación. Además, se han propuesto variaciones y extensiones de la arquitectura hexagonal para abordar desafíos específicos, como la gestión de transacciones distribuidas o la integración con sistemas heredados.

Algunos autores dicen que la arquitectura hexagonal dio el origen de la arquitectura de microservicios.

## Relación de la tecnología con la **arquitectura hexagonal**

La relación entre Django, Angular y PostgreSQL en el contexto de la arquitectura hexagonal puede entenderse mejor cuando se considera cómo cada una de estas tecnologías puede desempeñar un papel en la implementación de dicha arquitectura:

**Django:**

Con Django la representación de las vistas, modelos y controladores pueden representar diferentes componentes del hexágono interno, encapsulando la lógica de negocio y separándola claramente de las dependencias externas.

Django proporciona un sólido soporte para la integración con bases de datos relacionales como PostgreSQL, lo que facilita la implementación de adaptadores que interactúan con la capa de persistencia externa.

Django sigue siendo una de las mejores alternativas para el desarrollo backend debido a todo su conjunto de herramientas y su escalabilidad. Actualmente se encuentran 1.8 millones de proyectos activos con esta tecnología, y sigue siendo el framework backend de Python más usado, teniendo como segundo lugar FastApi y tercer lugar Flask.

Como se mencionó anteriormente, Django es un framework basado en el lenguaje de programación Python, el cual es el segundo lenguaje de programación más usado superando ya a Java y estando

detrás de JavaScript. Además, Python ha aumentado en popularidad por su simpleza y por su robusto ecosistema de herramientas para desarrollar IA, siendo el primero del mercado.

**Angular:**

Angular es un marco de desarrollo de aplicaciones de una sola página (SPA) que se utiliza para construir interfaces de usuario interactivas y dinámicas. En el contexto de la arquitectura hexagonal, Angular puede utilizarse para implementar la capa de adaptadores externos.

Las vistas y componentes de Angular pueden representar interfaces de usuario y adaptadores externos que interactúan con el hexágono interno a través de servicios o API REST.

Angular puede consumir los servicios RESTful proporcionados por Django para acceder a la lógica de negocio central de la aplicación.

Angular actualmente es uno de los marcos de trabajo más usado en el mundo frontend, asumiendo el segundo puesto después de React a nivel mundial. A pesar de lo anterior, este framework estuvo bajando en popularidad en los últimos años, hasta la llegada de su versión 17 el cual le dió mucha más vida en este aspecto. A comparación de React, el cual en la mayoría de los proyectos a gran escala usa un meta framework como lo que es NextJS para asegurar un orden y ofrecer funcionalidades adicionales como el renderizado del lado del servidor (SSR), angular ya ofrece todo esto nativamente, resultando en una muy buena alternativa. Además, en paises como Colombia, este framework sigue representando la mayoría de las ofertas de trabajo en el desarrollo frontend.

Junto a Angular, suele usarse el lenguaje de programación TypeScript, el cual representa una extensión del lenguaje JavaScript, el lenguaje de programación más usado por su impacto en el mundo web, ofreciendo todas las ventajas de alto nivel de JavaScript con la posibilidad de agregarle tipado al código, que permite la aplicación de patrones y estructuras para evitar posibes errores en el desarrollo.

**PostgreSQL:**

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y potente. En el contexto de la arquitectura hexagonal, PostgreSQL puede utilizarse como el almacenamiento de datos principal para la aplicación.

En la implementación de la arquitectura hexagonal, PostgreSQL puede considerarse como un adaptador externo que proporciona persistencia de datos al hexágono interno a través de modelos y consultas SQL.

Django proporciona una capa de abstracción sobre PostgreSQL a través de su ORM (Object-Relational Mapping), lo que facilita la interacción con la base de datos y la implementación de adaptadores de persistencia.

Postgresql en la actualidad sigue siendo la base de datos más usada en el mercado siendo esta una de las más robustas y estables comparado con su contraparte MySQL que le gana en rapidez. Además de esto, este motor de base de datos se comunica por SQL como una base de datos

relacional, perteneciendo al grupo más amplio y escalable de tipos de motores de bases de datos actualmente.

## Situaciones y/o problemas donde se puede usar la arquitectura hexagonal

La arquitectura hexagonal es aplicable a una variedad de situaciones y problemas en el desarrollo de software, uno de los ejemplos es:

- Aplicaciones empresariales complejas: En entornos empresariales donde se desarrollan aplicaciones complejas con múltiples flujos de datos y procesos, la arquitectura hexagonal puede ayudar a mantener una estructura clara y modular, facilitando la gestión y escalabilidad del sistema.
- Integración con sistemas externos: Cuando una aplicación necesita interactuar con varios sistemas externos, como bases de datos, servicios web, dispositivos de hardware, etc., la arquitectura hexagonal puede proporcionar una capa de adaptación clara y flexible para manejar estas integraciones de manera eficiente.
- Aplicaciones en evolución constante: En entornos donde los requisitos cambian con frecuencia o se espera una rápida evolución del software, la arquitectura hexagonal puede proporcionar la flexibilidad necesaria para adaptarse a estos cambios sin comprometer la estabilidad del sistema.
- Aplicaciones heredadas: Cuando se trabaja con aplicaciones heredadas que necesitan ser modernizadas o ampliadas, la arquitectura hexagonal puede ser útil para introducir un mayor modularidad y facilitar la integración con tecnologías más modernas y sistemas externos.

## Estilos y patrones de la arquitectura hexagonal

La arquitectura hexagonal se basa en la idea de tener un núcleo central rodeado por adaptadores o puertos que facilitan la comunicación con componentes externos. Al implementar esta arquitectura, se pueden utilizar varios estilos y patrones para organizar y estructurar el código de manera efectiva. Como:

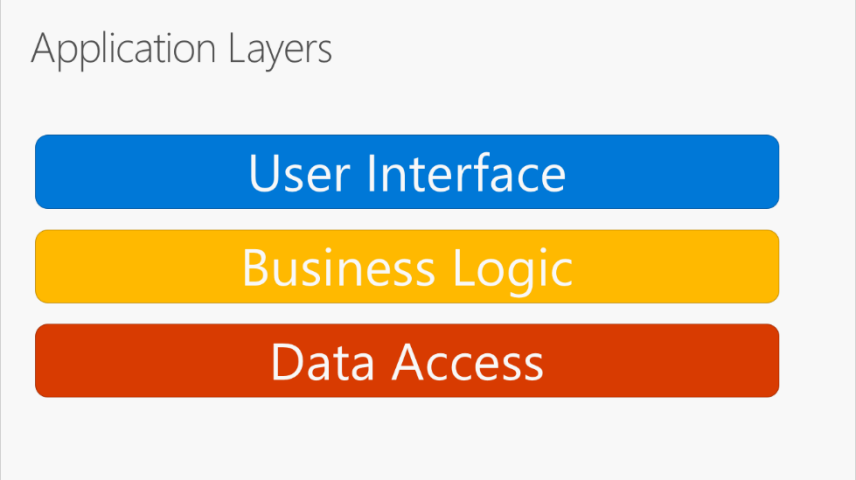


Imagen 2. Capas (Layered Architecture)

Este es un estilo arquitectónico común que se puede aplicar en conjunción con la arquitectura hexagonal. En este enfoque, se pueden dividir las responsabilidades del sistema en capas, como la capa de presentación, la capa de aplicación, la capa de dominio (núcleo hexagonal) y la capa de infraestructura (adaptadores externos). Cada capa tiene un propósito específico y se comunica con las capas adyacentes a través de interfaces bien definidas.

Arquitectura DDD (Domain-Driven Design): Se trata de un estilo de diseño de software que se centra en entender y modelar el dominio de negocio subyacente de una aplicación de manera profunda y clara. Se basa en la colaboración estrecha entre expertos en el dominio y desarrolladores, utilizando un lenguaje común y modelos de dominio expresivos para estructurar el código de manera que refleje fielmente las complejidades del negocio. DDD promueve la separación de responsabilidades en capas (como la capa de dominio, aplicación y presentación) y fomenta el uso de conceptos como agregados, entidades y objetos de valor para capturar y representar de manera efectiva las reglas y comportamientos clave del dominio en el código.

Inversión de Dependencias (Dependency Inversion Principle - DIP): Este principio SOLID es fundamental en la arquitectura hexagonal. Consiste en invertir las dependencias de manera que los módulos de alto nivel no dependan de los módulos de bajo nivel, sino de abstracciones. En el contexto de la arquitectura hexagonal, esto significa que el núcleo central (hexágono interno) no depende directamente de los detalles de implementación de los adaptadores externos, sino de interfaces que representan las operaciones que necesita realizar.

Inyección de Dependencias (Dependency Injection - DI): Este patrón se utiliza para implementar la inversión de dependencias, permitiendo que los objetos dependientes reciban las dependencias requeridas de fuentes externas en lugar de crearlas internamente. En la arquitectura hexagonal, la inyección de dependencias se utiliza para proporcionar los

adaptadores externos al hexágono interno, lo que facilita la sustitución de implementaciones concretas durante la ejecución.

Patrón Adaptador (Adapter Pattern): Este patrón se utiliza para permitir que interfaces incompatibles trabajen juntas. En el contexto de la arquitectura hexagonal, los adaptadores se utilizan para encapsular la interacción con sistemas externos y proporcionar una interfaz compatible con el hexágono interno. Esto facilita la integración de componentes externos sin afectar la lógica de negocio principal.

Patrón Estrategia (Strategy Pattern): Este patrón se utiliza para definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. En la arquitectura hexagonal, el patrón de estrategia puede utilizarse para representar diferentes estrategias de adaptación para interactuar con diferentes tipos de sistemas externos. Por ejemplo, se pueden definir diferentes estrategias de adaptación para interactuar con bases de datos SQL, servicios web REST, servicios SOAP, etc.

## Ventajas y desventajas

### Ventajas

**Separación de lógica de negocio:** La separación de la lógica de negocio central y los componentes externos facilita la comprensión y el mantenimiento del sistema.
**Flexibilidad:** La arquitectura hexagonal facilita la introducción de cambios y la adaptación a nuevos requisitos sin afectar otras partes del sistema. Los adaptadores externos proporcionan una capa de abstracción que permite cambiar los componentes externos sin necesidad de modificar la lógica interna de la aplicación.
**Pruebas unitarias y de integración**: Al separar claramente la lógica de negocio de las dependencias externas, la arquitectura hexagonal facilita la escritura de pruebas unitarias y de integración. Los componentes pueden ser probados de forma aislada, lo que facilita la identificación y corrección de errores.
**Reutilización de componentes**: La separación de preocupaciones promovida por la arquitectura hexagonal facilita la reutilización de componentes en diferentes contextos y proyectos. Los adaptadores externos pueden diseñarse de forma genérica para interactuar con sistemas externos, facilitando su reutilización.
**Escalabilidad**: La arquitectura hexagonal facilita la escalabilidad del sistema al permitir que los componentes sean escalados de manera independiente según sea necesario. Esto permite gestionar eficazmente el crecimiento del sistema y la carga de trabajo.

## Desventajas

**Complejidad inicial**: La implementación de la arquitectura hexagonal puede requerir un esfuerzo adicional de diseño y planificación debido a la necesidad de definir interfaces claras entre los componentes internos y externos del sistema.

**Overhead de abstracción**: La introducción de adaptadores externos puede introducir un overhead de abstracción adicional en el sistema, lo que puede afectar el rendimiento en algunos casos.

**Posible sobrediseño**: En algunos casos, la arquitectura hexagonal puede llevar al sobrediseño si se utiliza de manera innecesaria o excesiva en proyectos más simples o de menor escala.

**Curva de aprendizaje**: La arquitectura hexagonal puede tener una curva de aprendizaje pronunciada para aquellos que no están familiarizados con sus principios y prácticas. Los equipos de desarrollo pueden requerir tiempo y esfuerzo para comprender completamente los conceptos y aplicarlos de manera efectiva en el desarrollo de software.

## Principios SOLID relacionados con la arquitectura hexagonal

La arquitectura hexagonal se basa en los principios SOLID para promover un diseño de software robusto y flexible.

**Principio de Responsabilidad Única (SRP - Single Responsibility Principle)**: En la arquitectura hexagonal, el SRP se aplica separando claramente las responsabilidades del sistema en diferentes componentes. Por ejemplo, el núcleo interno de la aplicación (hexágono) se encarga de la lógica de negocio, mientras que los adaptadores externos se ocupan de las interacciones con sistemas externos. Cada componente tiene una única responsabilidad y motivo de cambio.

**Principio de Abierto/Cerrado (OCP - Open/Closed Principle)**: La arquitectura hexagonal favorece el OCP al permitir que el sistema sea abierto para la extensión, pero cerrado para la modificación. Esto se logra mediante la definición de interfaces claras entre el núcleo interno y los adaptadores externos. El núcleo interno no se modifica cuando se agregan nuevos adaptadores externos, sino que se extiende mediante la implementación de nuevas interfaces.

**Principio de Sustitución de Liskov (LSP - Liskov Substitution Principle)**: En la arquitectura hexagonal, el LSP se aplica mediante la garantía de que los adaptadores externos sean intercambiables sin afectar la funcionalidad del sistema. Los adaptadores externos deben implementar las interfaces definidas por el núcleo interno para que puedan sustituirse sin causar efectos no deseados en el comportamiento del sistema.

**Principio de Segregación de Interfaces (ISP - Interface Segregation Principle)**: La arquitectura hexagonal cumple con el ISP al definir interfaces específicas para cada tipo de interacción entre el núcleo interno y los adaptadores externos. Cada adaptador externo implementa solo las interfaces relevantes para su función, evitando así la dependencia de interfaces innecesarias.

**Principio de Inversión de Dependencias (DIP - Dependency Inversion Principle)**: El DIP es fundamental en la arquitectura hexagonal. Se aplica mediante la inversión de las dependencias, de modo que el núcleo interno dependa de abstracciones (interfaces) en lugar de detalles de implementación concretos. Esto permite la sustitución de adaptadores externos sin modificar el núcleo interno del sistema.

## Atributos de calidad asociados

**Modularidad:** La arquitectura hexagonal promueve la modularidad al separar claramente la lógica de negocio del resto del sistema y al utilizar adaptadores externos para interactuar con componentes externos. Esto facilita la creación de componentes independientes y reutilizables que pueden probarse y mantenerse individualmente.

**Flexibilidad**: La separación de la lógica de negocio del resto del sistema hace que sea más fácil introducir cambios y adaptarse a nuevos requisitos sin afectar otras partes del sistema. Los adaptadores externos proporcionan una capa de abstracción que permite cambiar los componentes externos sin necesidad de modificar la lógica interna de la aplicación.

**Pruebas**: La arquitectura hexagonal facilita la prueba unitaria y de integración al permitir que los componentes sean probados de forma aislada. La separación de la lógica de negocio de las dependencias externas hace que sea más fácil escribir pruebas que se centren en la funcionalidad principal de la aplicación sin preocuparse por los detalles de implementación de los adaptadores externos.

**Escalabilidad**: Al separar claramente las responsabilidades y utilizar adaptadores externos para interactuar con sistemas externos, la arquitectura hexagonal facilita la escalabilidad del sistema. Los componentes pueden ser escalados de manera independiente según sea necesario, lo que permite gestionar eficazmente el crecimiento del sistema y la carga de trabajo.

**Mantenibilidad**: La separación de preocupaciones y modularidad inherente a la arquitectura hexagonal hacen que sea más fácil mantener y actualizar el sistema a lo largo del tiempo. Los cambios en los requisitos del negocio o en las tecnologías subyacentes pueden ser manejados con mayor facilidad al tener una arquitectura bien estructurada y desacoplada.

**Reutilización**: La arquitectura hexagonal promueve la reutilización de componentes al separar claramente la lógica de negocio del resto del sistema. Los adaptadores externos pueden diseñarse de forma genérica para interactuar con sistemas externos, facilitando su reutilización en diferentes contextos y proyectos.

## Casos de estudio relevantes

- **Twitter:** La API de Twitter utiliza la arquitectura hexagonal para separar la lógica de negocio de la infraestructura subyacente. Esto permite que la API sea escalable y fácil de mantener.

- **Mercado Libre**: La plataforma de comercio electrónico Mercado Libre también utiliza la arquitectura hexagonal para permitir una mayor flexibilidad y escalabilidad
- **Netflix**: La aplicación móvil de Netflix utiliza la arquitectura hexagonal para separar la lógica de negocio de la interfaz de usuario. Esto permite que la aplicación sea más fácil de actualizar y mantener.
- **Spotify**: La aplicación de música Spotify también utiliza la arquitectura hexagonal para mejorar la testabilidad y la escalabilidad.
- **Minecraft**: utiliza la arquitectura hexagonal para permitir una mayor flexibilidad en el desarrollo de mods y plugins.
- **Reddit**: La plataforma de agregación de noticias Reddit utiliza la arquitectura hexagonal para mejorar la testabilidad y la flexibilidad.
- **GitHub**: La plataforma de alojamiento de código GitHub también utiliza la arquitectura hexagonal para mejorar la escalabilidad y la confiabilidad.
- **Google Analytics**: utiliza la arquitectura hexagonal para separar la lógica de negocio de la capa de almacenamiento de datos.
- **Sistemas financieros**: Algunos sistemas financieros, como los bancos, utilizan la arquitectura hexagonal para mejorar la seguridad y la confiabilidad.

## Tags del código de ejemplo

https://github.com/jpsanchezg/arquitectura-hexagonal-backend/releases/tag/v.1.1.0

https://github.com/jpsanchezg/arquitectura-hexagonal-frontend/releases/tag/v1.1.0

## Referencias Bibliográficas

CloudAPPi. (2021, marzo 2). *Introducción a la Arquitectura Hexagonal*. CloudAPPi. https://cloudappi.net/introduccion-a-la-arquitectura-hexagonal/

Franco. (2023, julio 5). *¿Qué es arquitectura hexagonal en programación?* ThePower Business School; ThePowerMBA. https://www.thepowermba.com/es/blog/que-es-arquitectura-hexagonal-en-programacion

Nasibov, Z. (2021, octubre 30). *Hexagonal architecture and Python - Part I: Dependency Injection and componential architecture*. Zaur's Thoughts. https://znasibov.info/posts/2021/10/30/hexarch_di_python_part_1.html

Nasibov, Z. (2022a, septiembre 18). *Hexagonal architecture and Python - part II: Domain, application services, ports and adapters*. Zaur's Thoughts. https://znasibov.info/posts/2022/09/18/hexarch_di_python_part_2.html

Nasibov, Z. (2022b, diciembre 31). *Hexagonal architecture and Python - part III: Persistence, transactions, exceptions and the final assembly*. Zaur's Thoughts. https://znasibov.info/posts/2022/12/31/hexarch_di_python_part_3.html

Woltmann, S. (s/f). *Hexagonal Architecture - what is it? Why should you use it?* Recuperado el 18 de marzo de 2024, de https://www.happycoders.eu/software-craftsmanship/hexagonal-architecture/

Woodie, A. (2024, 3 enero). *Postgres Rolls Into 2024 with Massive Momentum. Can It Keep It Up?* Datanami. https://www.datanami.com/2024/01/03/postgres-rolls-into-2024-with-massive-momentum-can-it-keep-it-up/