Ingeniería de Software

Prof. Jhon Eder Masso Daza

Tema: Diseño Arquitectónico

Introducción

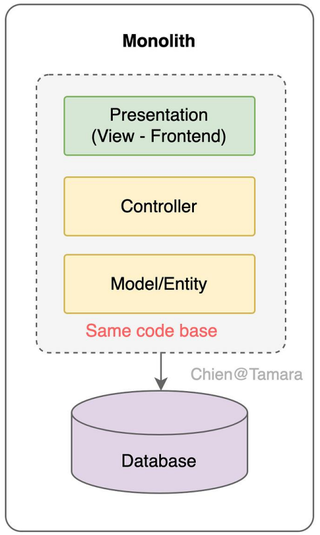
- Según Pressman [1], el DA “representa la **estructura de los datos** y de los **componentes del programa** que se requieren para construir un sistema basado en computadores”.
- Según Sommerville [2], el DA “se interesa por entender cómo debe **organizarse** un sistema y cómo tiene que **diseñarse la estructura global** de ese sistema”.
- Según Zengyang Li et al. [3], describen que el DA especifica cada uno de los **elementos** necesarios que permiten el **desarrollo**, **mantenimiento**, **evolución** y **comunicación** de los sistemas de software.

Patrones Arquitectónicos

- Un patrón arquitectónico es una solución general y reutilizable que ayuda a definir las características y el comportamiento de una aplicación [10].
- Los patrones arquitectónicos permiten describir la estructura y comportamiento de un sistema de software. Además, pretenden satisfacer los requisitos funcionales y atributos de calidad [11].
- Algunos de los patrones de arquitectura de software más comunes son: Modelo vista controlador, capas, microservicios, hexagonal, etc.

Monolito

- **Monolito:** Es un patrón arquitectónico en donde cada uno de sus componentes están interconectados y son interdependientes. Es decir, Todos los elementos, módulos y capas de la aplicación se encuentran integrados en un mismo proyecto y se despliegan como una única unidad (artefacto ejecutable o contenedor).
  - **Ventajas**:
    - Simplicidad para el desarrollo y pruebas.
    - Una base de datos unificada que facilita la integridad y la sincronización de los datos.
    - Baja sobrecarga operativa, menos piezas que monitorear y para solucionar problemas.
  - **Desventajas**:
    - Desafíos al escalar una parte específica del sistema y más aún si hay una fuerte dependencia y acoplamiento.
    - La incorporación de nuevas tecnologías o funcionalidades puede ser engorroso.
    - Mayor riesgo de fallos. Si algo falla se puede caer todo el sistema.

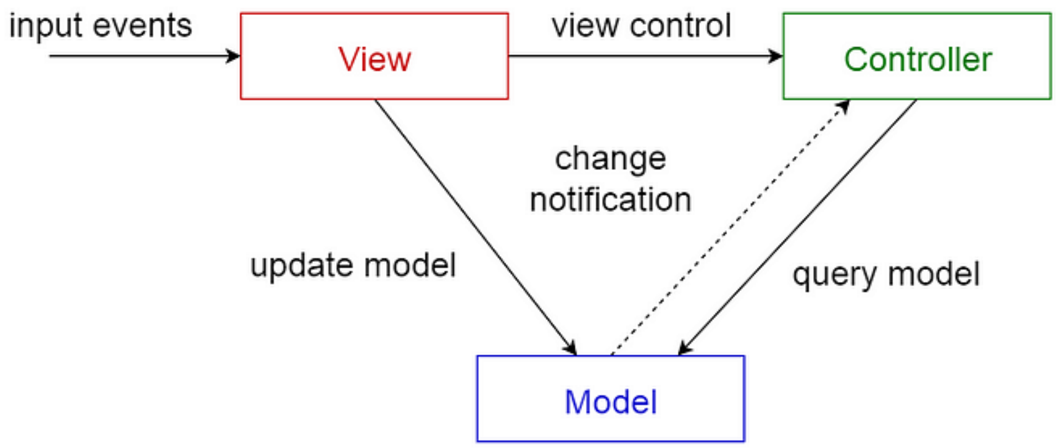


Fuente: http://bit.ly/4pdhjNJ

Nota: Una alternativa para reducir las desventajas es crear monolitos modulares. Estos permiten dividir la aplicación en módulos independientes que aún se seguirán desplegando como una única unidad.

Patrones Arquitectónicos

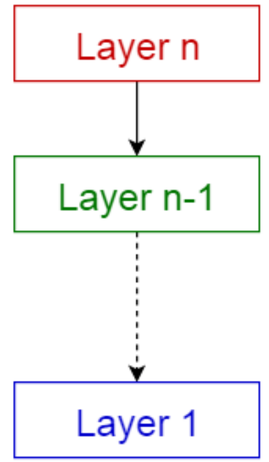
- **Modelo vista Controlador (MVC):** Este patrón permite organizar la lógica de una aplicación de software en capas, las cuales cumplen tareas específicas e interactúan entre sí para garantizar la funcionalidad del sistema.
  - **Modelo**: Responsable de la lógica de los datos. Además, de almacenar y recuperar la información.
  - **Vista**: Proporciona la interfaz de usuario para interactuar con el sistema.
  - **Controlador**: Proporciona la lógica necesaria para responder a las acciones que se solicitan en la aplicación. Es el intermediario entre la vista y el modelo.



Fuente: https://bit.ly/46CPowR

Patrones Arquitectónicos

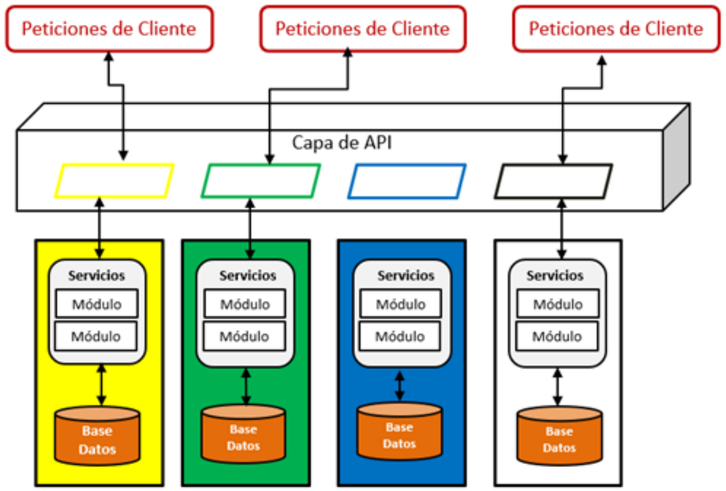
- **Capas:** Este patrón tiene como propósito dividir el código de la aplicación en capas, las cuales tienen responsabilidades y ofrecen servicios a una capa superior.
- Algunas de las capas más comunes empleadas por desarrolladores son:
  - **Capa de presentación**: conocida como la capa interfaz de usuario (Front-End). Permite la interacción del usuario con la aplicación.
  - **Capa de lógica del negocio**: conocida como capa de dominio. Es la encargada de procesar las solicitudes del usuario y coordinar las operaciones necesarias para cumplir con las funcionalidades de la aplicación.
  - **Capa de servicio:** conocida como capa de aplicación, ofrece una interfaz entre la capa del negocio y la de acceso a datos.
  - **Capa de acceso a datos:** conocida como capa de persistencia de datos, es la que permite interactuar con las fuentes de datos.



Fuente: https://bit.ly/46CPowR

Patrones Arquitectónicos

- **Microservicios:** Este es uno de los patrones más recientes. Permite descomponer una aplicación monolítica en servicios autónomos, los cuales pueden ser implementados de forma independiente. Es decir, varias aplicaciones ligeras que permiten realizar funcionalidades concretas y que en conjunto conforman una aplicación principal.

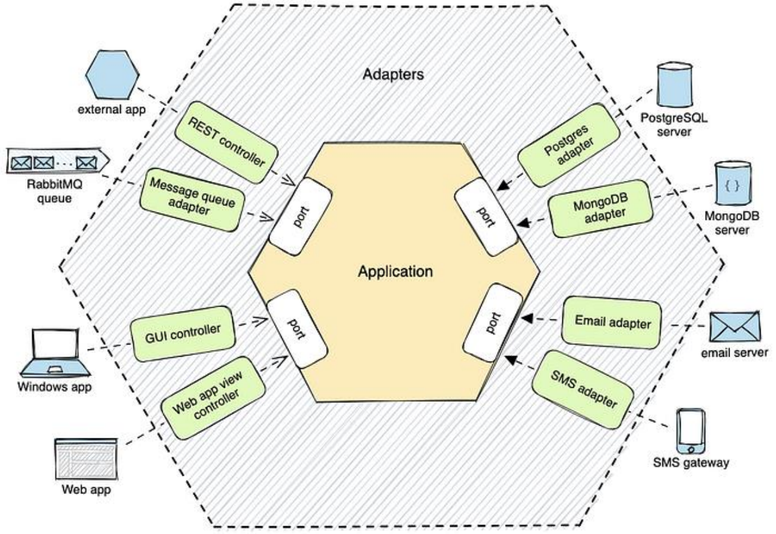


Fuente: https://bit.ly/46CPowR

- **Ventajas**: Escalado flexible ya que solo se escalarán los servicios que lo requieran. Interdependencia entre los servicios. Libertad de tecnologías. Permite el desarrollo e implementación en paralelo lo que permite acelerar el proceso.
- **Desventajas**: Complejidad operativa en la gestión y monitoreo del sistema. Trazabilidad de datos, pueden estar distribuidas o en diferentes fuentes dificultando el seguimiento y consistencia. Tiempos de respuesta de las operaciones debido a la latencia de red. La arquitectura puede aislar las fallas.

Patrones Arquitectónicos

- **Hexagonal:** Fue propuesta Alistair Cockburn, es conocida como puertos y adaptadores o arquitectura de cebolla creada. Permite crear proyectos mantenibles, adaptables y fácilmente comprobables. Se basa en el principio de gestión de la inversión de dependencias.



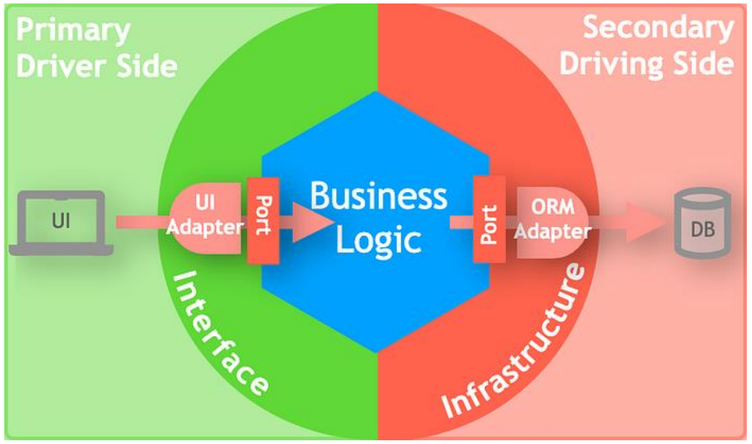
Fuente: https://bit.ly/4m051pb

Se representa como un hexágono:

- El dominio representa la lógica del negocio. Es decir el corazón de la aplicación ***entidades***, ***reglas***, ***casos de uso*** (***acciones concretas***). No depende de tecnologías.
- Los bordes del hexágono representan los puertos (interfaces o contratos). Definen qué se necesita o qué se ofrece (***repositorios, controladores, etc***.).
- Los adaptadores son implementaciones concretas que permiten conectar al dominio con el exterior con una tecnología concreta.

Patrones Arquitectónicos

- **Hexagonal:** La Arquitectura Hexagonal busca aislar la lógica de negocio de una aplicación respecto a sus dependencias externas. Los sistemas externos (bases de datos, servicios web, interfaces de usuario, etc.) se comunican con el núcleo mediante puertos (interfaces definidas por la aplicación) y adaptadores (implementaciones concretas que conectan esas interfaces con tecnologías específicas).



Fuente: https://bit.ly/4m051pb

- **Ventajas**: Independencia de frameworks y tecnologías. Alta facilidad para realizar pruebas. Flexibilidad y mantenibilidad. Favorece la longevidad del software y su evolución.
- **Desventajas**: Complejidad ya que requiere mayor esfuerzo de diseño y comprensión. Puede ser demasiado estructurado para proyectos software pequeños debido a sus múltiples componentes. La curva de aprendizaje es desafiante ya que requiere más código.

¿Monolito o Microservicios?

- No confundir un monolito con la generación de código desorganizado o ausencia de separación de responsabilidades. Un monolito bien estructurado podrá migrar a microservicios cuando tenga sentido.
- Un monolito es aconsejable para aplicaciones pequeñas o cuando se busca una solución intermedia (monolito modular). Cuando los equipos son pequeños, dominio poco cambiante, time-to-market crítico.
- Los microservicios son adecuados para aplicaciones grandes y complejas que requieren actualizaciones y escalado independiente.
- La decisión de que aplicar los patrones arquitectónicos dependerá del proyecto, recursos, experiencia del equipo y planes de crecimiento de la aplicación a desarrollar.
- Los monolitos pueden alinearse mejor con los procesos tradicionales de desarrollo e implementación, mientras que los microservicios adoptan la agilidad, flexibilidad y resiliencia en paradigmas como DevOps.

Diseño Arquitectónico (DA)

Lecturas:

1. Patrones en la Arquitectura de Software. Disponible en: https://bit.ly/46xv5Rd
2. Los 10 patrones comunes de arquitectura de software. Disponible en: https://bit.ly/46CPowR
3. Software Architecture Patterns. Disponible en: https://bit.ly/3F5MHb6
4. Diseño de arquitectura de microservicios. Disponible en: https://bit.ly/3LMjLc1
5. Software Architecture Patterns: What Are the Types and Which Is the Best One for Your Project. Disponible en: https://bit.ly/46vTdDZ

Bibliografía

[1] Pressman, Roger S., Ingeniería del Software un enfoque Práctico McGRAW-HILL. 2010. ISBN: 978-607-15-0314-5
[2] Sommerville, Ian., Ingeniería de Software. Prentice Hall. 2011. ISBN: 978-607-32-0603-7
[3] Li, Z., Liang, P., Avgeriou, P., Application of knowledge-based approaches in software architecture: A systematic mapping study, Information and Software Technology, Volume 55, Issue 5, 2013, Pages 777-794, ISSN 0950-5849
[4] Mark, Richards. Software Architecture Patterns. O'Reilly Media, Inc. 2015.
[5] Farshidi, S., Jansen, S., Van der Werf, J., Capturing software architecture knowledge for pattern-driven design, Journal of Systems and Software, Volumen 169,2020, ISSN 0164-1212
[6] Cockburn, A., The Hexagonal (Ports & Adapters) Architecture. 2005. Disponible en: http://bit.ly/3IflesT
[7] AWS. Building hexagonal architectures on AWS, AWS Prescriptive Guidance. 2025. Disponible en: https://bit.ly/45VTpir

¡ Gracias!