# Software Architecture

... the shared understanding that the expert developers have of the system design.

“Architecture is about the important stuff. Whatever that is”

It means that the heart of thinking architecturally about software is to decide what is important, (i.e. what is architectural), and then expend energy on keeping those architectural elements in good condition.

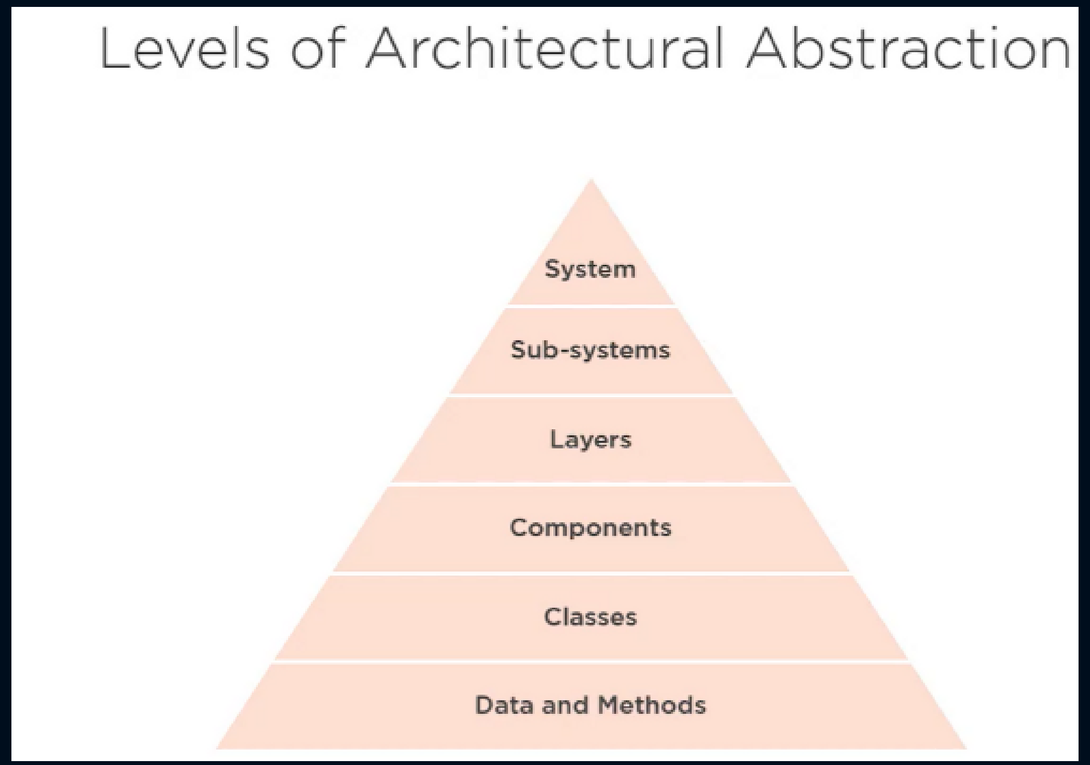
High quality software is cheaper to produce.

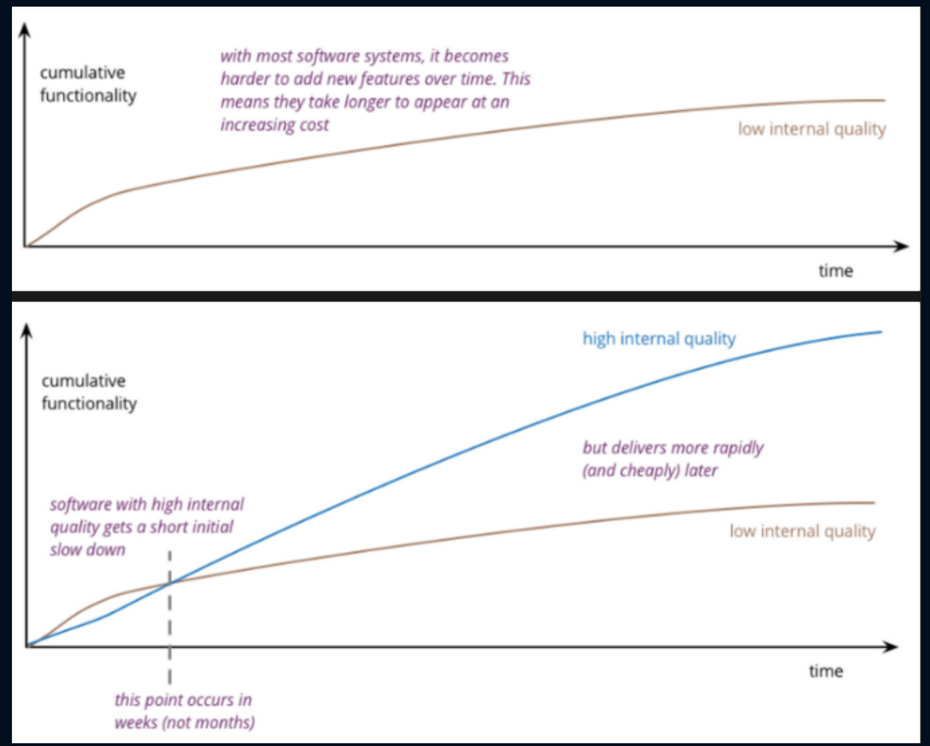
Summing all of this up:

- Neglecting internal quality leads to rapid build up of cruft
- This cruft slows down feature development
- Even a great team produces cruft, but by keeping internal quality high, is able to keep it under control
- High internal quality keeps cruft to a minimum, allowing a team to add features with less effort, time, and cost.

# Architectural patterns

Software architecture patterns are reusable solutions to common design problems in software development. They provide guidelines and best practices to organize code, components, and data in a way that promotes modularity, scalability, maintainability, and reusability.

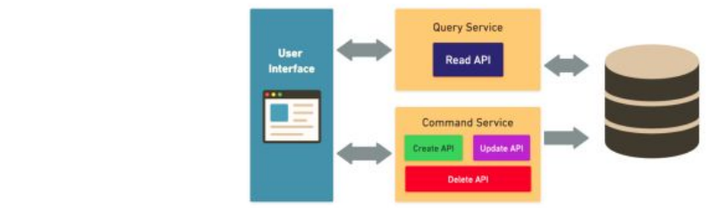



Software architecture consequences

# Architectural styles

## Software Architecture Styles




**CQRS Architecture**

separates read and write operations for a data store. It enables independent scaling of read and write workloads and optimizes them separately.


**Orchestration Architecture**

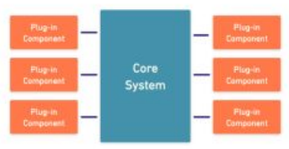
a central coordinator (often called an orchestrator) that directs the interaction between services. The orchestrator is responsible for managing the control flow and data flow between services.

**Layered (n-tier) Architecture**

separates software into logical layers

**Microkernel Architecture**

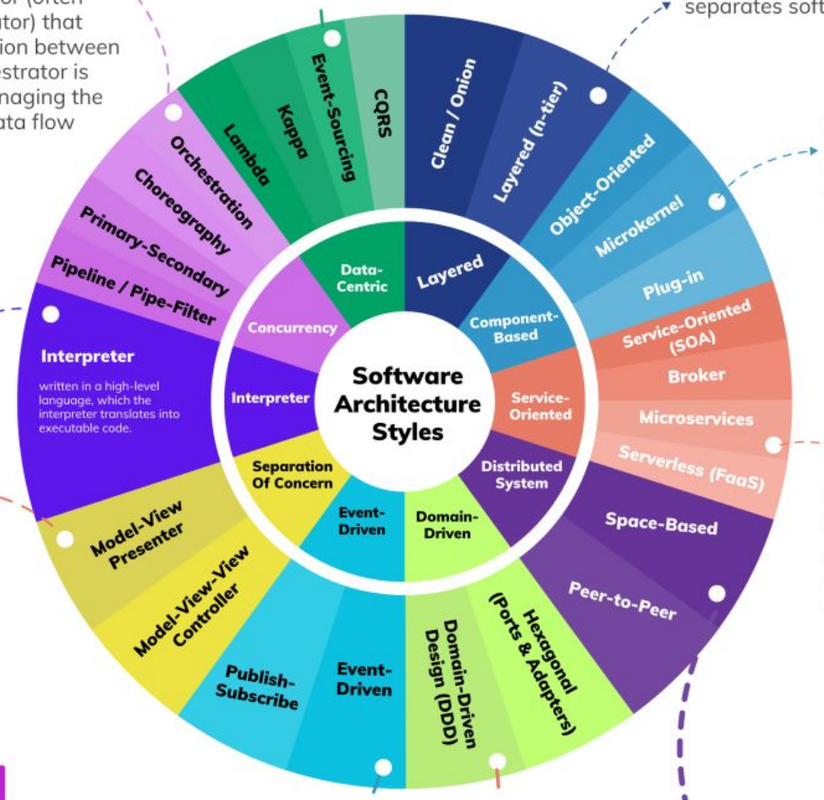
separates a minimal functional core from extended functionality and customer-specific parts.





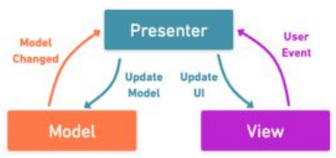
**Interpreter**

written in a high-level language, which the interpreter translates into executable code.



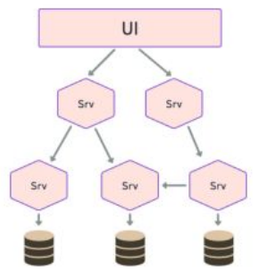

**MVP Architecture**

derivative of the Model-View-Controller (MVC) pattern, which aims to separate the concerns of data management, user interface, and control flow.



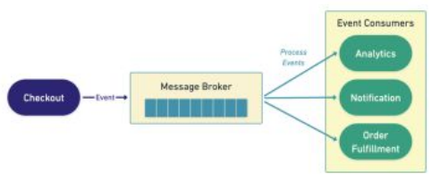

**Microservice Architecture**

This architecture designs a software application as a suite of independently deployable, small, modular services.



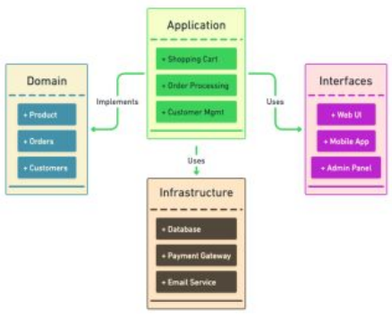

**Event-Driven Architecture**

promotes the production, detection, consumption of, and reaction to events.



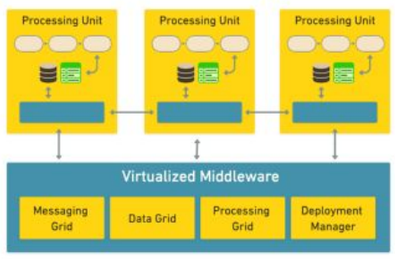

**DDD Architecture**

focuses on the domain logic and complexity rather than the technology used.




**Space-Based Architecture**

This resolves the issues of data consistency, reliable performance, and scalability for large-scale distributed systems.

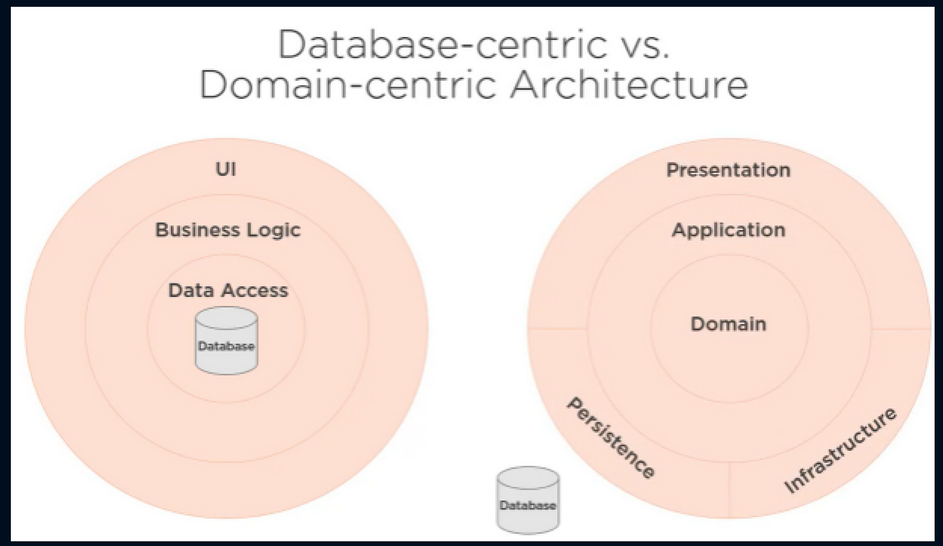



ByteByteGo

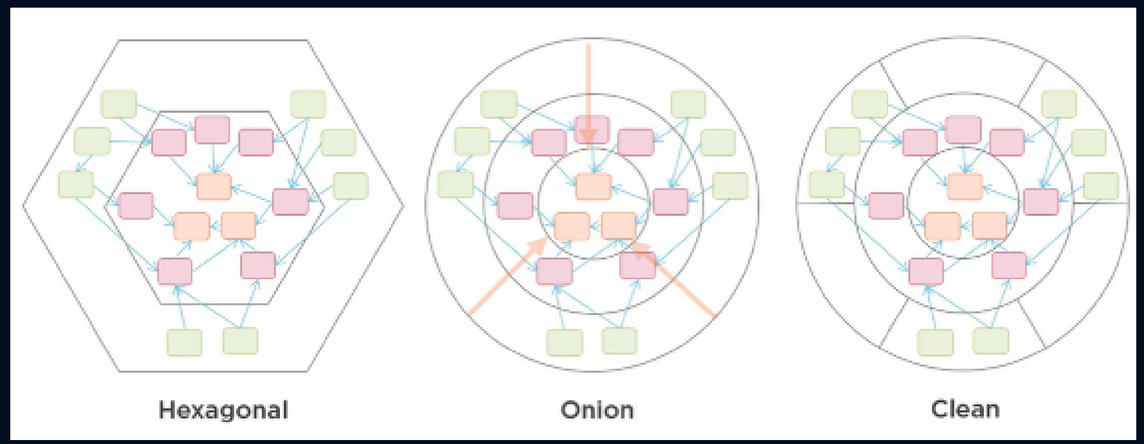
Arch styles

# Domain centric

**Domain-Centric Software Architecture is an architectural approach that places the domain model at the core of the software design. It emphasizes modeling the problem domain and capturing the business rules and logic in the codebase.**



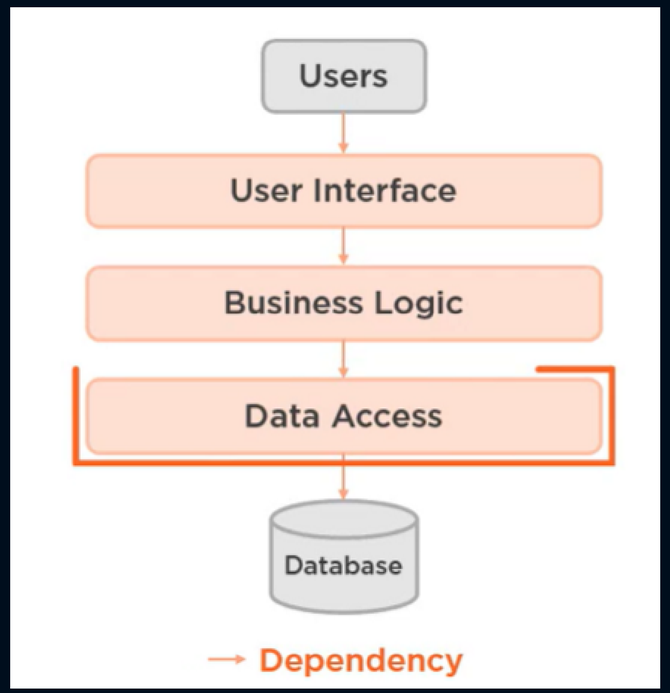

In a Database-Centric approach, the design of the application is heavily influenced by the underlying database structure. Changes to the database schema can have a significant impact on the application code, making it less adaptable to evolving business requirements.



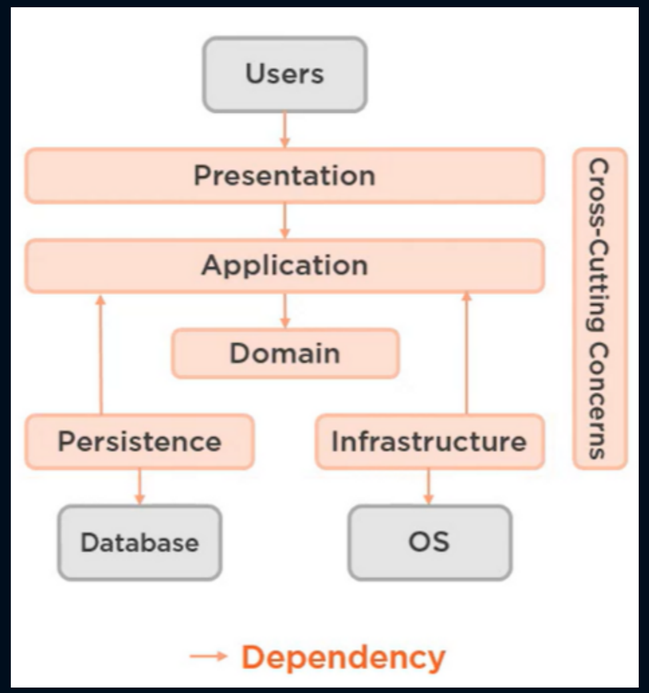

In a Domain-Centric approach, the primary focus is on modeling the problem domain and capturing the business rules and logic in the application. The domain model is at the core of the architecture, consisting of classes representing entities, value objects, and aggregates within the business domain.
Other advantages are allows for DDD, less coupling, at the other side i has an initial higher cost and requires more thought.

# Layered architecture

Are common software architectural patterns that organizes an application into logical layers, with each layer having a specific responsibility and level of abstraction. The primary goal of Layered Architecture is to promote separation of concerns and modularity, making the application more maintainable, scalable, and testable.




3 tier layered architecture




4 tier layered architecture

# Command and Query (CQRS)

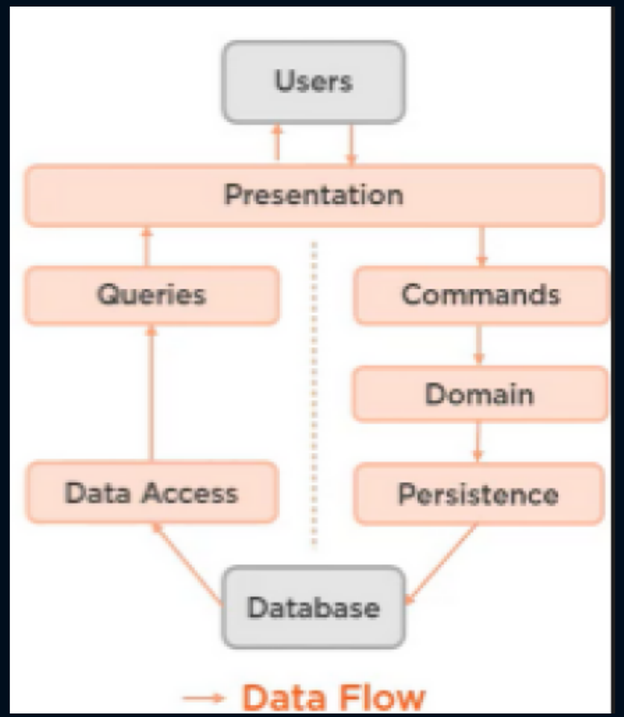
**Is a software architectural pattern that separates the read and write operations (queries and commands) of an application into distinct components. The fundamental idea behind CQRS is to use different models for reading data (query side) and writing data (command side) to improve performance, scalability, and maintainability.**

Command Model: handles operations that modify data, such as creating, updating, or deleting entities. It contains the application's business logic for processing commands and enforcing business rules. The command model is responsible for maintaining the consistency and integrity of the data.

Query Model: responsible for handling read operations, such as retrieving data for display or reporting. It is optimized for querying and doesn't contain any business logic. The query model denormalizes the data to improve query performance, providing a tailored view of the data to the query side.

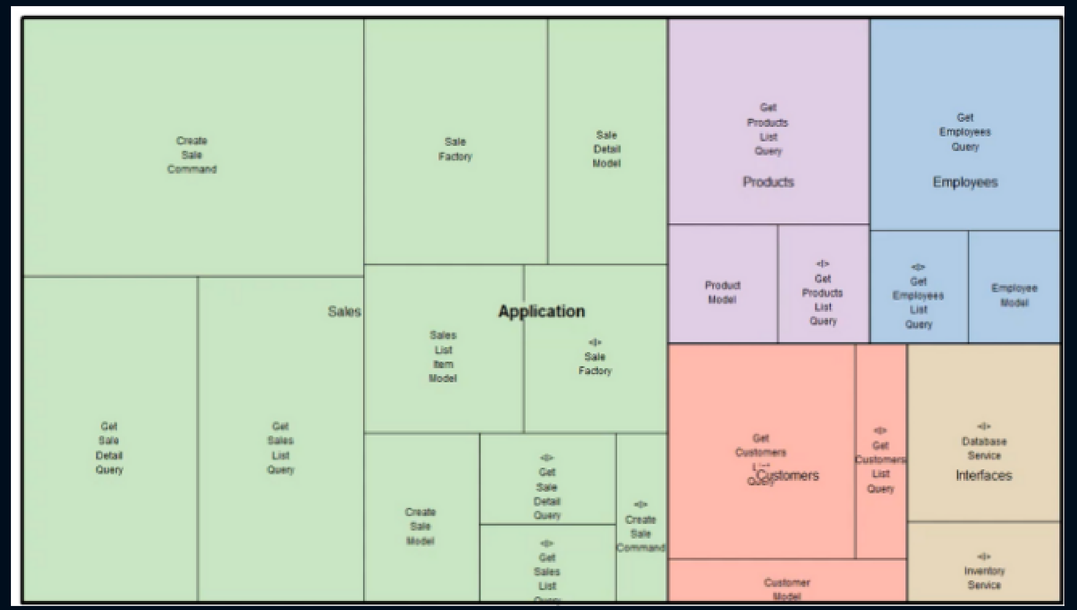
Event Sourcing: often used in conjunction with CQRS. Instead of storing the current state of entities, event sourcing records a series of domain events that represent changes to the entity's state over time. This approach provides a complete audit trail of all changes to the data and enables easy replay of events to reconstruct the current state.

Asynchronous Processing: CQRS can be used with asynchronous processing to decouple the command and query handling from the user interface. This approach improves responsiveness and resilience.



# Vertical Slicing

Vertical Slicing, also known as Functional Slicing or Feature Slicing, is a software architectural pattern that focuses on organizing code and related functionality around individual features or user stories rather than traditional horizontal layers. The primary goal of Vertical Slicing is to improve maintainability, modularity, and collaboration within a development team.



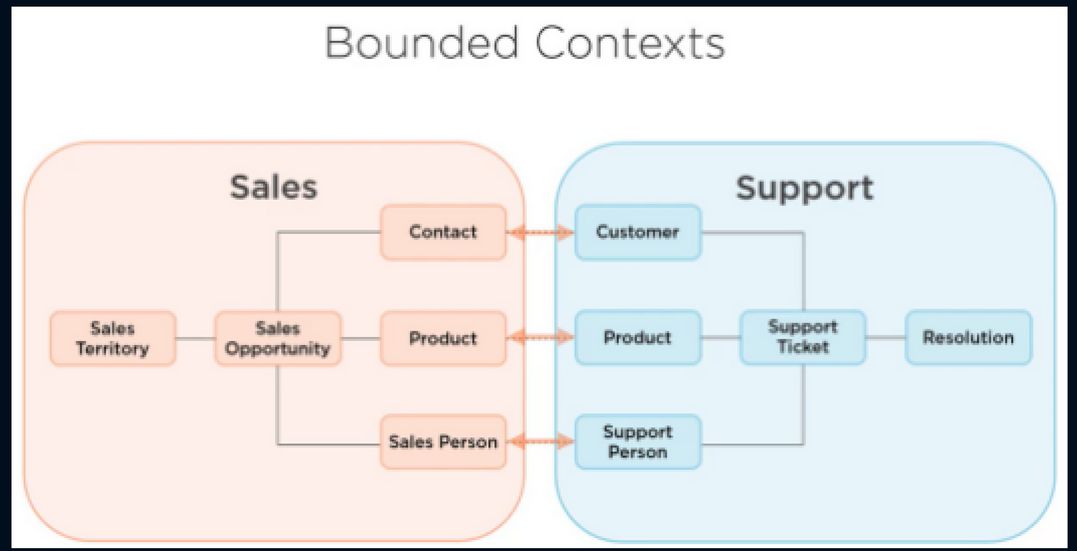

Screaming architecture


Vertical Slicing

# Microservices

Is a software architectural pattern that structures an application as a collection of small, independent, and loosely coupled services. Each service is designed to perform a specific business capability and can be developed, deployed, and scaled independently. The fundamental idea behind microservices is to break down a monolithic application into smaller, manageable, and specialized components, allowing for better agility, scalability, and maintainability.

Service Independence: Each microservice operates as an independent unit with its own codebase, database, and business logic.
Decentralized Data Management: Each microservice typically manages its data store, which can be any database or data storage technology best suited for the service's requirements.
Inter-Service Communication: Microservices communicate with each other through well-defined APIs, often over HTTP or messaging protocols.
Scalability and Flexibility: Microservices can be independently scaled based on demand. Services experiencing high traffic can be scaled horizontally, while less used services can remain unchanged, optimizing resource utilization.
Technology Diversity: Microservices allow each service to use the most appropriate technology stack for its specific needs.
Resilience and Fault Isolation: If one microservice fails, it does not impact the entire system.



# Discussion

**Exercise/ Homework**

Using markdown, complete the next Quiz

- What is software architecture, and why is it important in the development process?
- Explain the difference between architecture and design in software development.
- Describe the role of software architecture throughout the software development lifecycle.
- Discuss how Clean Architecture can be applied in a practical scenario, such as a web application or mobile app.
- Provide advantages and potential challenges of adopting Clean Architecture in a team-based development environment.

NOTE: Use images, pictures, diagrams with tools like Mermaid if it helps you