Unit-04

Architectural Design & System Modeling

4.1. Context Models

Context models are high-level diagrams that illustrate the environment in which a system operates. They show the system's boundaries and how it interacts with external entities, such as users, other systems, and organizational processes. Context models help in understanding the scope and limitations of the system, clarifying interfaces and dependencies with external elements.

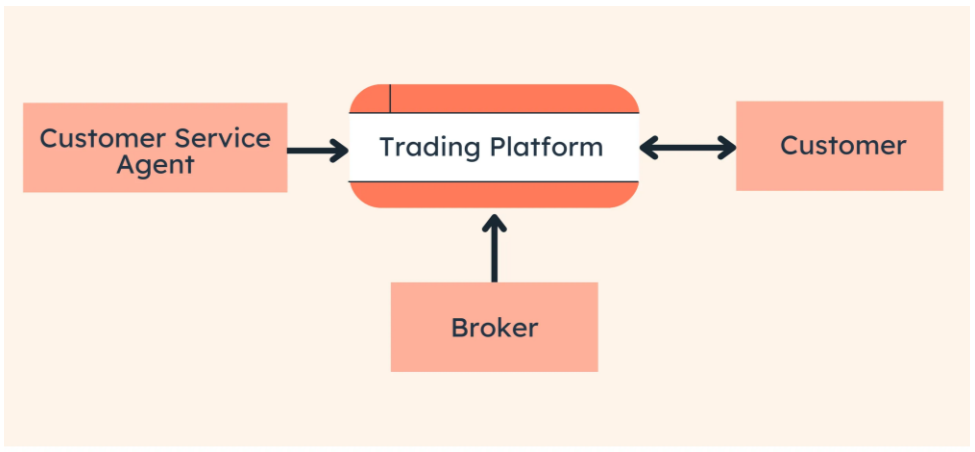
Purpose of Context Models

1. **Define System Boundaries**: Clearly delineate what is inside and outside the system to avoid scope creep and ensure that all external interactions are identified.
2. **Identify External Entities**: Recognize all external actors, systems, and devices that interact with the system.
3. **Understand Interactions**: Visualize the flow of data and control between the system and external entities.
4. **Facilitate Communication**: Provide a common understanding for stakeholders, developers, and project managers about the system's environment and interactions.
5. **Support Requirements Elicitation**: Help in gathering and clarifying requirements by showing how the system fits into its operational context.

Types of Context Models

1. Data Flow Diagrams (DFDs): Show the flow of data between the system and external entities.

- **Components**:
  - **External Entities**: Sources or destinations of data outside the system.
  - **Processes**: Represent the system functions that transform data.
  - **Data Stores**: Represent where data is stored within the system.
  - **Data Flows**: Arrows showing the direction of data movement.

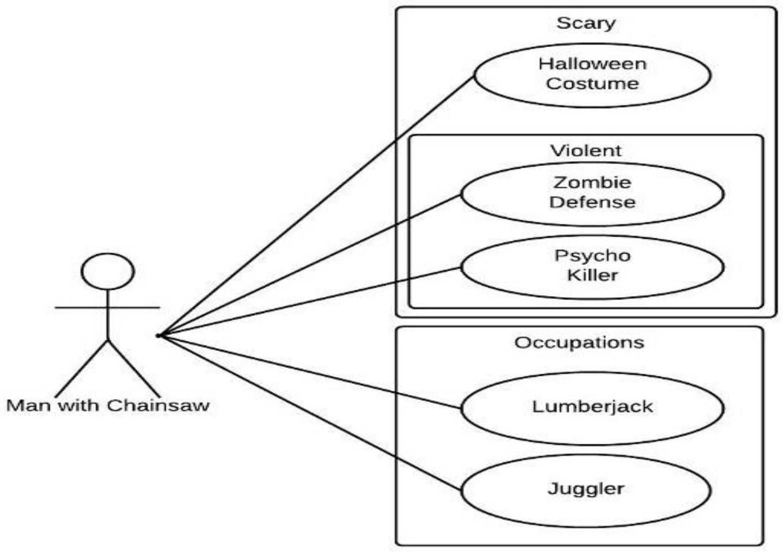


This context model shows a **Trading Platform** and its interactions with three external entities:

- **Customer Service Agent**: Provides support, interacts with the platform.
- **Customer**: Uses the platform for trading, interacts bi-directionally.
- **Broker**: Facilitates trades, interacts with the platform.

2. Use Case Diagrams (part of UML): Show interactions between actors (users or external systems) and the system through use cases.

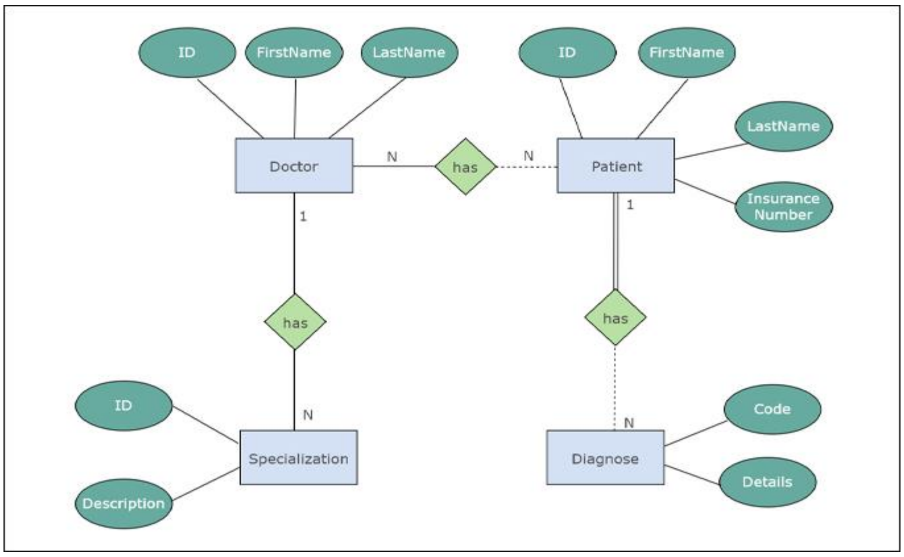
- **Components**:
  - **Actors**: Represent users or other systems that interact with the system.
  - **Use Cases**: Represent functions or services provided by the system.
  - **System Boundary**: A box that encapsulates the use cases, indicating the scope of the system.



This use case diagram represents various scenarios associated with a "Man with Chainsaw" character.

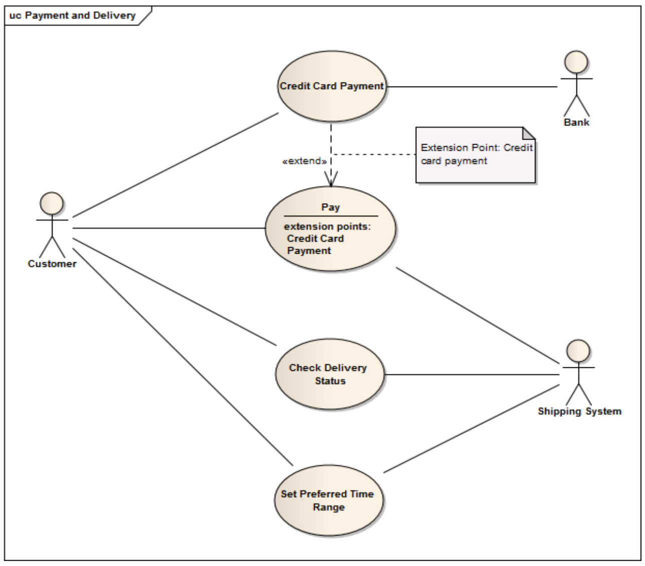
3. Entity-Relationship Diagrams (ERDs): Show the relationships between entities within the system and how they interact with external entities.

- **Components**:
  - **Entities**: Objects or concepts with distinct existence in the domain.
  - **Relationships**: Show how entities are related to each other.
  - **Attributes**: Characteristics of entities.



4.2. Interaction Models

Interaction models describe how components within a system interact with each other and with external entities to achieve a specific functionality. They provide a dynamic view of the system and are essential in understanding how different parts of the system collaborate.



Purpose of Interaction Models

1) **Detail Dynamic Behavior**: Show how components interact over time, including the sequence of events and the flow of data.
2) **Clarify Communication**: Visualize how components and external entities communicate and exchange information.
3) **Facilitate Design**: Aid in designing the interactions and interfaces between system components.
4) **Support Testing**: Provide a basis for creating test cases that verify the correctness of interactions.
5) **Enhance Understanding**: Help stakeholders understand the dynamic aspects of the system and how different parts work together.

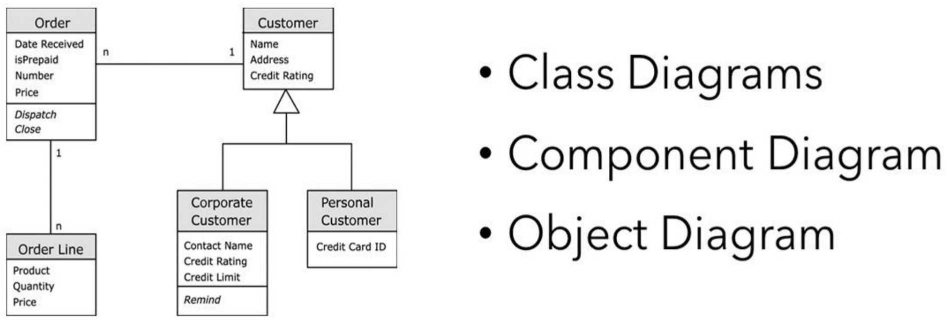
Major Components of Interaction Models

1) **Actors**: External entities (users or systems) that interact with the system.
2) **Objects/Components**: The parts of the system that participate in the interaction.
3) **Messages**: The information exchanged between objects or components.
4) **Events**: Actions or occurrences that trigger interactions or state changes.
5) **States**: Conditions or situations during the life of an object.

Interaction models are essential tools in system analysis and design that illustrate how components of a system interact with each other and with external entities. They detail the dynamic behavior of the system, clarify communication, and support both design and testing processes. Types of interaction models include use case diagrams, sequence diagrams, collaboration diagrams, activity diagrams, and state diagrams, each serving to enhance understanding and improve the overall system design.

4.3. Structural Models

Structural models focus on the static aspects of a software system. They depict the organization of the system in terms of its components, classes, objects, and their relationships. Structural models are essential for understanding how a system is built and how its parts interact with each other.



Structural Diagrams in Unified Modeling Language (UML) are used to represent the static structure of a system. They describe how different parts of the system are related to each other.

Class Diagrams:

- **Purpose**: Show the classes in the system and their relationships.
- **Components**: Classes, attributes, methods, and relationships like associations, inheritances, and dependencies.
- **Example**: The diagram shows an Order class related to a Customer class, which has two specialized classes: Corporate Customer and Personal Customer.

Component Diagrams:

- **Purpose**: Illustrate the organization and dependencies among software components.
- **Components**: Components, interfaces, and dependencies.
- **Usage**: Helpful for visualizing the high-level structure of the system's implementation.

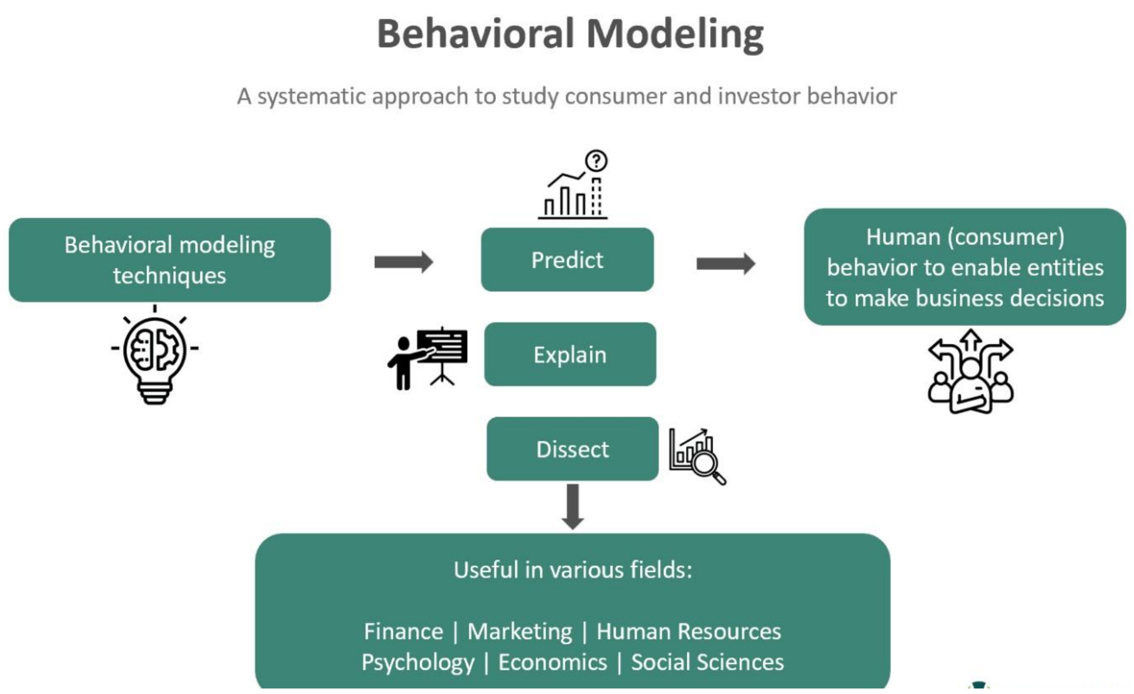
Object Diagrams:

- **Purpose**: Represent instances of classes at a particular moment in time.

- **Components**: Objects (instances of classes) and links (instances of relationships).
- **Usage**: Useful for explaining real-world examples or specific scenarios.

4.4. Behavioral Models

Behavioral models describe the dynamic aspects of a system, focusing on how it behaves over time. They capture the interactions, events, and states that objects undergo within the system. These models are essential for understanding the functioning and the flow of control within a system.


Behavioral Modeling
A systematic approach to study consumer and investor behavior
Behavioral modeling techniques
Predict
Explain
Dissect
Human (consumer) behavior to enable entities to make business decisions
Useful in various fields:
Finance | Marketing | Human Resources
Psychology | Economics | Social Sciences

1) **Behavioral Modeling Techniques**: Methods and tools used to analyze behavior patterns.
2) **Predict**: Forecast future behaviors based on observed data.
3) **Explain**: Clarify why certain behaviors occur.
4) **Dissect**: Break down behaviors into understandable components.

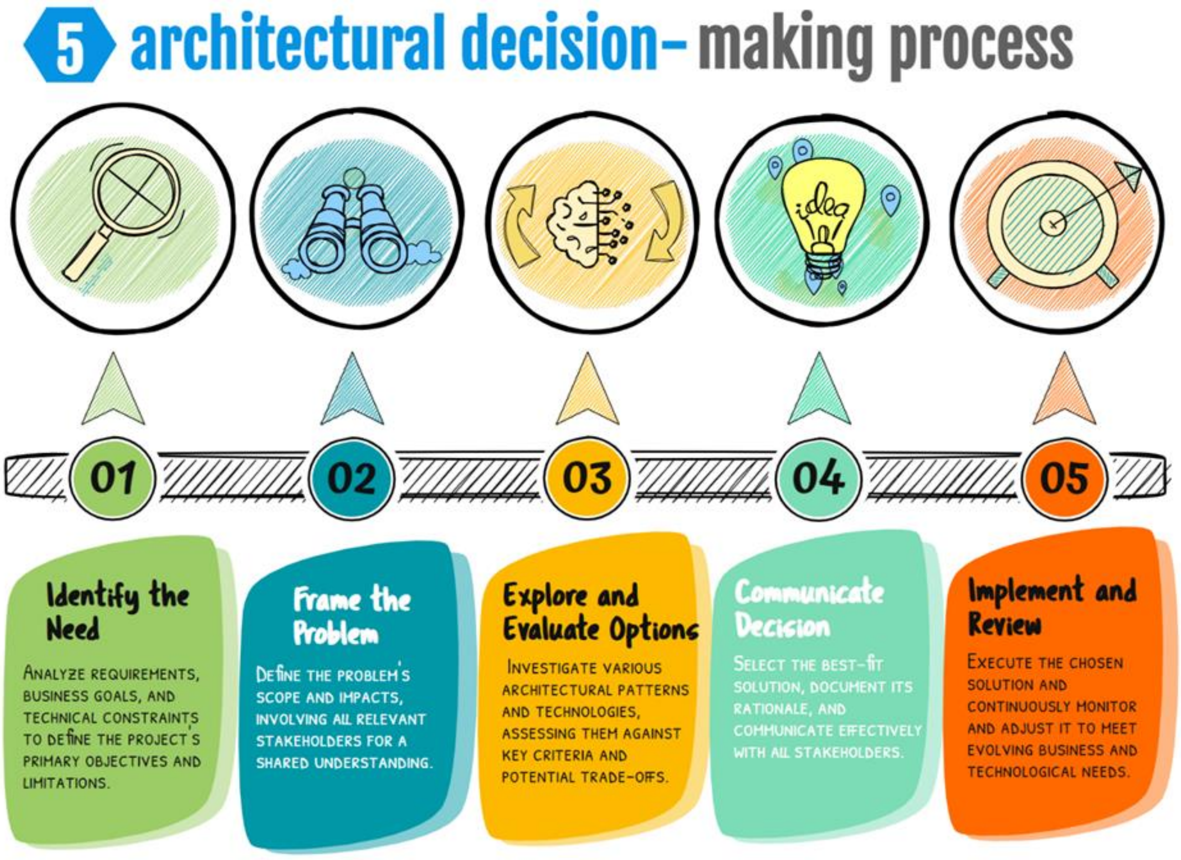
These steps help businesses understand and anticipate human behavior, aiding in decision-making processes across various fields like finance, marketing, human resources, psychology, economics, and social sciences.

Purpose of Behavioral Models

1) **Understanding System Dynamics**: Provide a clear picture of how the system operates in different scenarios, helping stakeholders understand its behavior over time.
2) **Capturing Functional Requirements**: Document how the system should respond to various inputs and interactions, ensuring that functional requirements are met.
3) **Facilitating Communication**: Help communicate complex behaviors among developers, designers, and stakeholders, ensuring everyone has a common understanding of the system's operation.
4) **Supporting Design and Implementation**: Serve as a blueprint for designing and implementing the system, guiding developers on how to code the interactions and behaviors.

4.5. Architectural Design Decisions

Architectural design decisions are critical choices made during the development of a software system's architecture. These decisions determine the fundamental structure and behavior of the system.



Identify the Need

Analyze requirements, business goals, and technical constraints to define the project's primary objectives and limitations.

Frame the Problem

Define the problem's scope and impacts, involving all relevant stakeholders for a shared understanding.

Explore and Evaluate Options

Investigate various architectural patterns and technologies, assessing them against key criteria and potential trade-offs.

Communicate Decision

Select the best-fit solution, document its rationale, and communicate effectively with all stakeholders.

Implement and Review

Execute the chosen solution and continuously monitor and adjust it to meet evolving business and technological needs.

- **Identify the Need:**
  This first step involves analyzing the project's requirements, business goals, and technical constraints to define the primary objectives and limitations. Additionally, understanding the end-users' needs and expectations ensures the solution aligns with their demands. The objective here is to establish a clear and comprehensive understanding of what the project aims to achieve and ensure that all stakeholders have a shared vision of the desired outcome.

- **Frame the Problem:**
  Framing the problem involves defining the scope and impacts of the issue at hand. This step requires identifying all relevant stakeholders and their concerns. The objective is to create a well-defined problem statement that captures the essence of the challenge and sets the stage for developing effective solutions.

- **Explore and Evaluate Options:**
  During this step, various architectural patterns and technologies are investigated to address the defined problem. This involves conducting a thorough market and technology analysis to identify viable options. The objective is to assess each option against key criteria such as cost, performance, scalability, and maintainability, enabling informed decision-making.

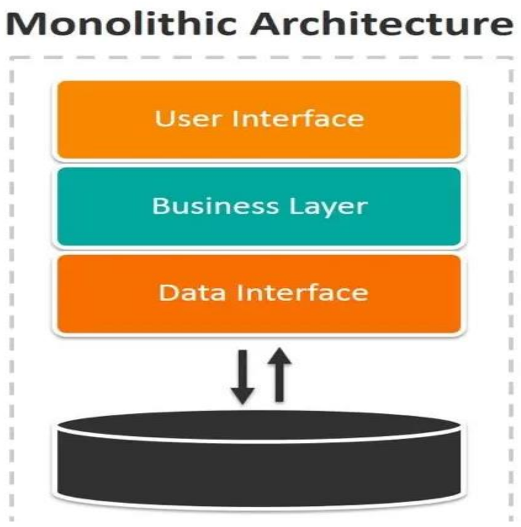
- **Communicate Decision:**
  Communicating the decision involves selecting the best-fit solution and documenting its rationale clearly. It is essential to provide a roadmap for implementation, outlining the steps required to realize the chosen solution. The objective is to ensure that all stakeholders understand and support the decision, fostering alignment and collaboration throughout the project lifecycle.

- **Implement and Review:**
  Implementing the chosen solution requires careful planning and execution. This involves setting up a project management plan, allocating resources, and defining timelines. The objective is to ensure the solution is executed effectively, meets the defined requirements, and can be adjusted as needed to adapt to evolving business and technological needs.

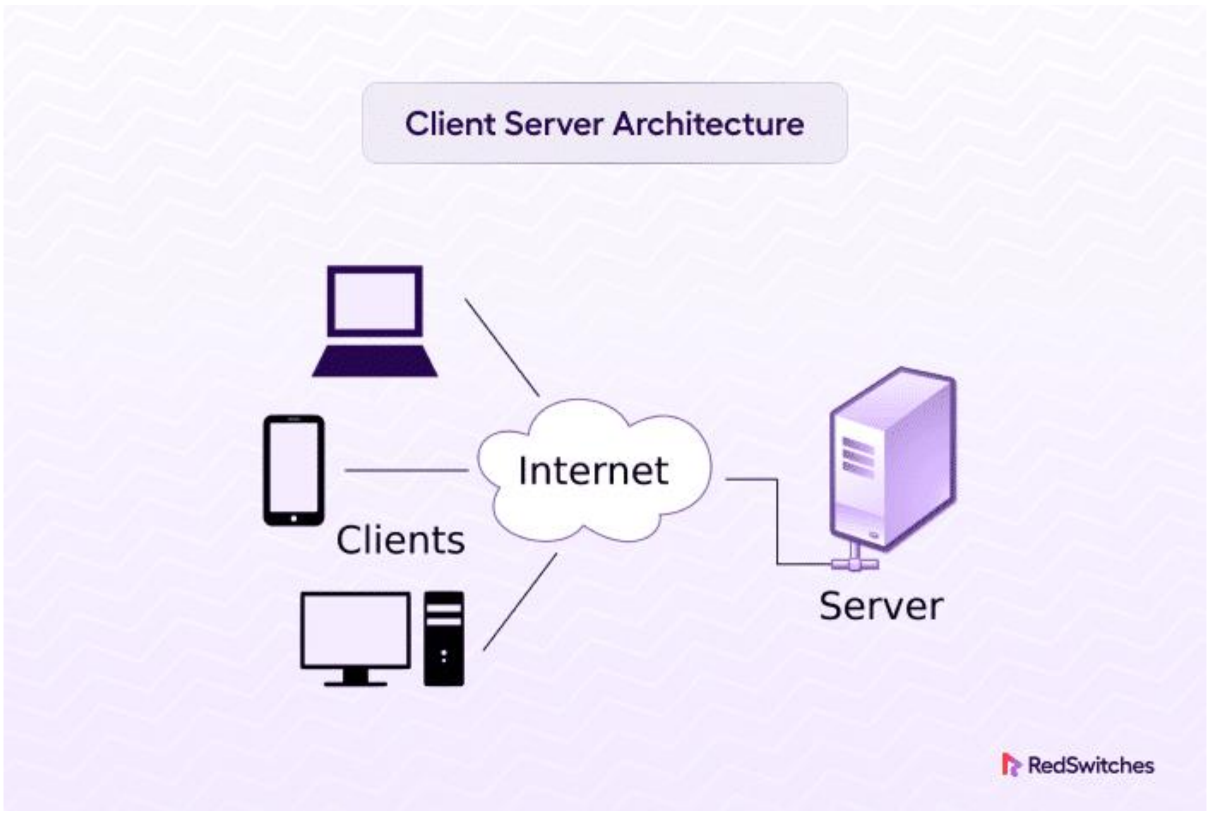
4.6. Application Architectures

In software engineering and project management, application architectures refer to the high-level structures of software systems, outlining how software components interact and are organized to achieve specific functionality and objectives. Here are some common types of application architectures:

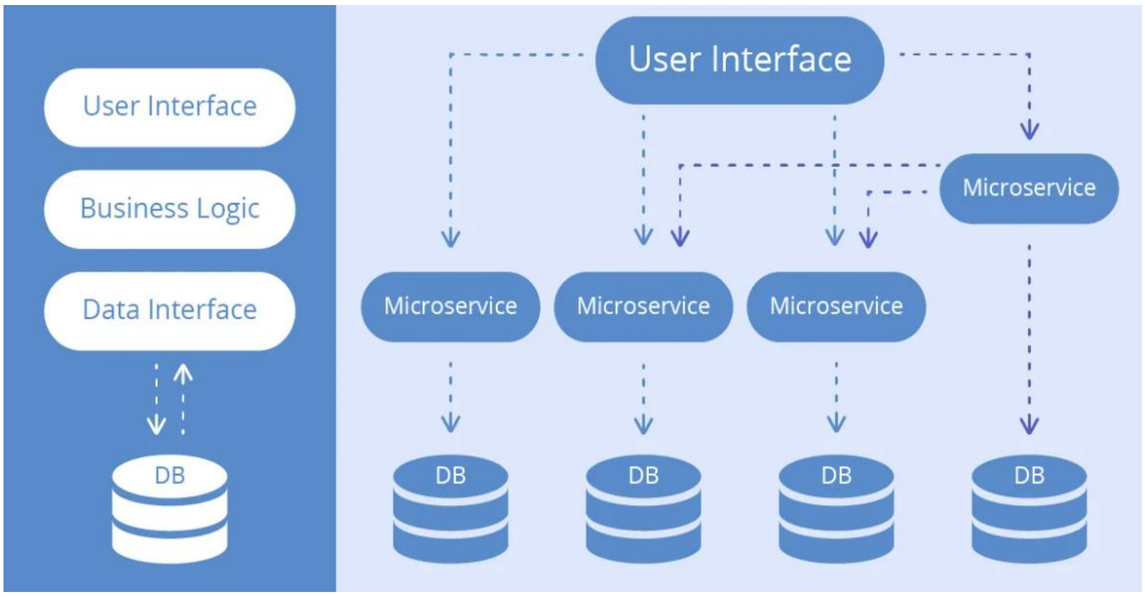
1) **Monolithic Architecture**: In this traditional approach, the entire application is built as a single unit. All components are interconnected and deployed together. It's simpler to develop initially but can become complex and difficult to scale as the application grows.



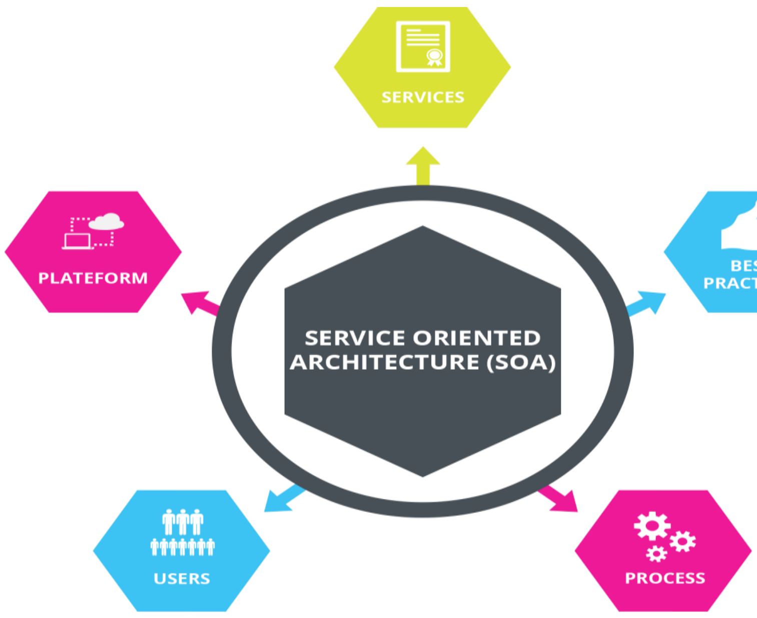
2) **Client-Server Architecture**: This architecture divides the system into two major components: clients, which request services, and servers, which provide services. This model enables easier management, scalability, and division of labor between front-end and back-end components.



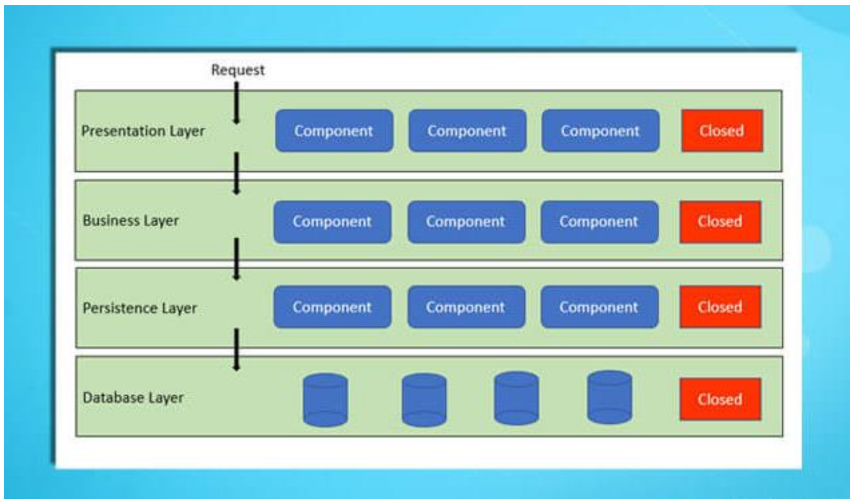
3) **Microservices Architecture**: This approach structures an application as a collection of loosely coupled services, each responsible for specific business capabilities. Microservices communicate through APIs and can be developed, deployed, and scaled independently, offering flexibility and resilience.



4) **Service-Oriented Architecture (SOA)**: SOA involves organizing software components into services that can be accessed and reused across different applications within an enterprise. It promotes reusability, interoperability, and flexibility in integrating disparate systems.



5) **Layered Architecture**: Also known as n-tier architecture, this divides an application into logical layers, such as presentation layer, business logic layer, and data layer. Each layer handles specific responsibilities, promoting separation of concerns and modularity.


Request
Presentation Layer
Component
Component
Component
Closed
Business Layer
Component
Component
Component
Closed
Persistence Layer
Component
Component
Component
Closed
Database Layer
Closed