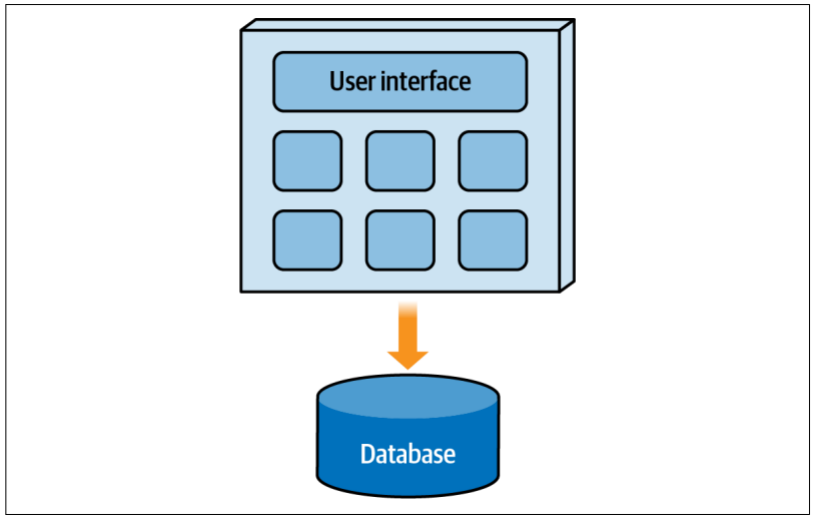
Layered Architecture

Architecture Styles

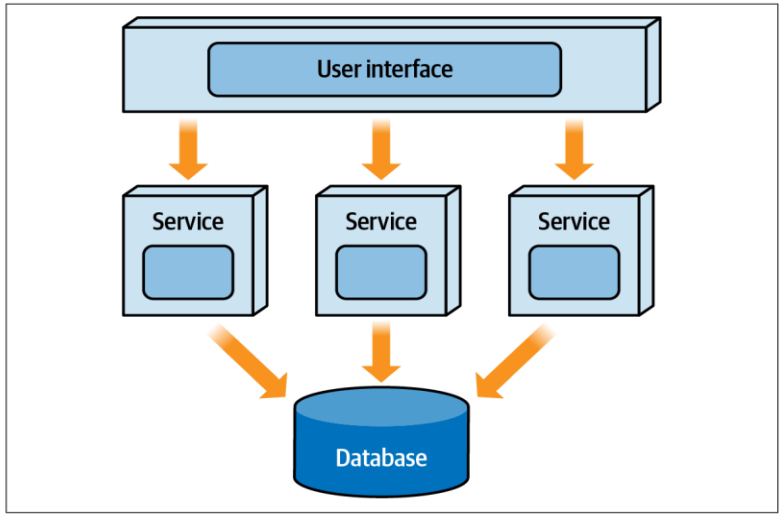
- Define the basic characteristics and behavior of an application
- defines how the components of a software system interact with each other
- Different architecture styles offer different level of scalability, flexibility, maintainability, or performance
- Some architecture styles naturally lend toward highly scalable systems, whereas some naturally lend toward quick development

Architecture Classification

- Monolithic
  - Simple, Easy, Single Deployment



- Distributed
  - Scalability, Fault tolerance

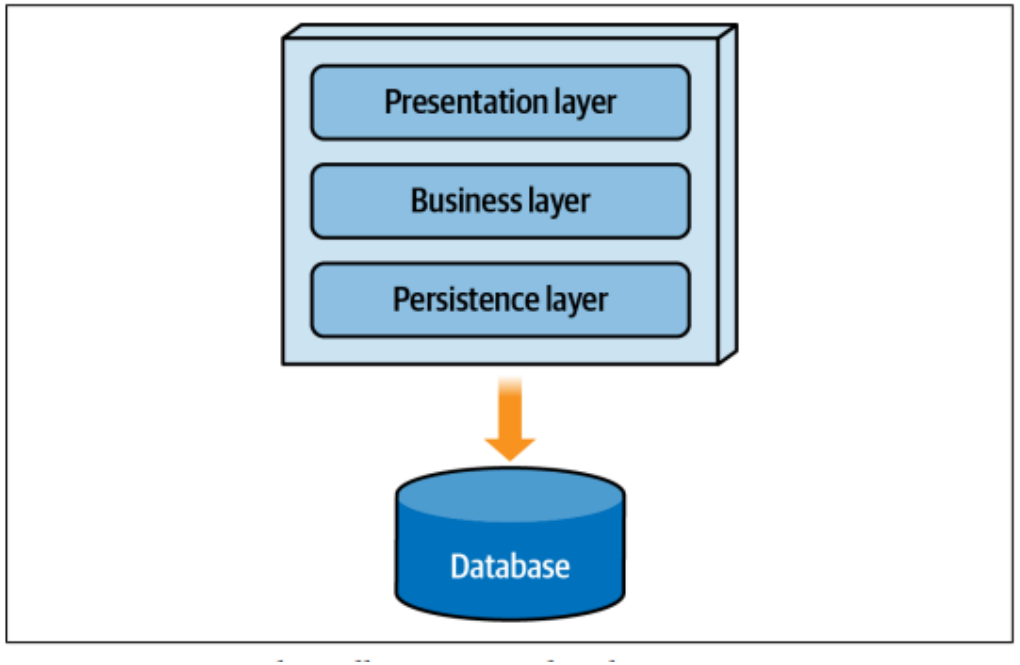


Architecture Partitioning

- Technical Partitioning
- Domain Partitioning

Technical Partitioning

- Components of the system organized by technical usage
- Useful if a majority of your changes are isolated to a specific technical area
  - For example, if you are constantly changing the look and feel of your user interface without changing the corresponding business rules, change is isolated to only the presentation layer.

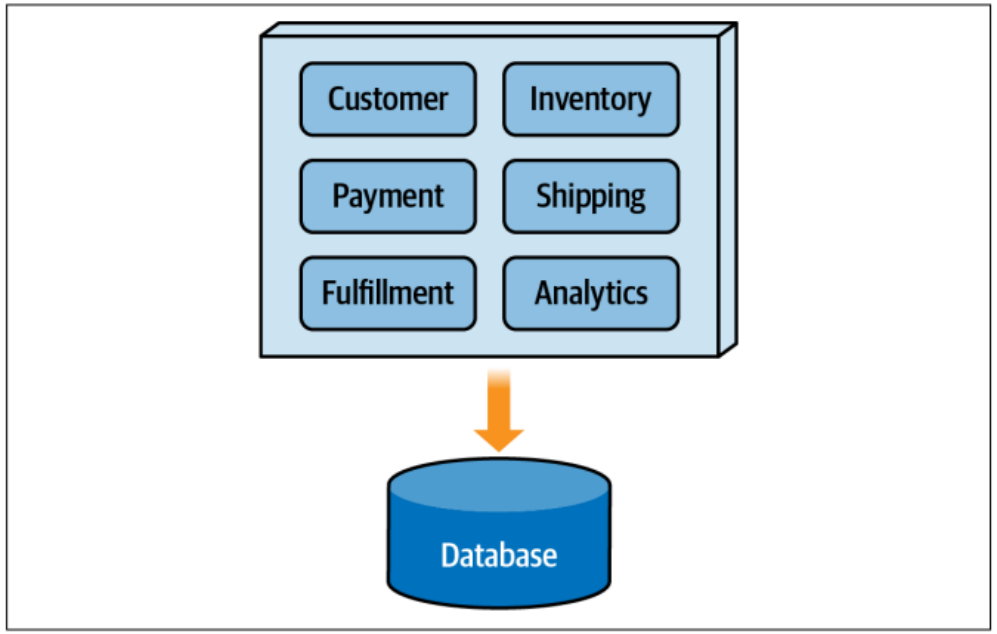


Technical Partitioning Issue

- However, imagine implementing a new requirement to add an expiration data to items for customer wish lists.
  - Add a new column to the wish list table in the database layer.
  - Update SQL queries in the persistence layer.
  - Define business rules in the business layer.
  - Modify contracts between business and presentation layers.
  - Update UI screens in the presentation layer.

Domain Partitioning

- Components of the system organized by domain areas
- all of the functionality (presentation, business logic, and persistence logic) is grouped together for each domain and subdomain area in separate areas of the application

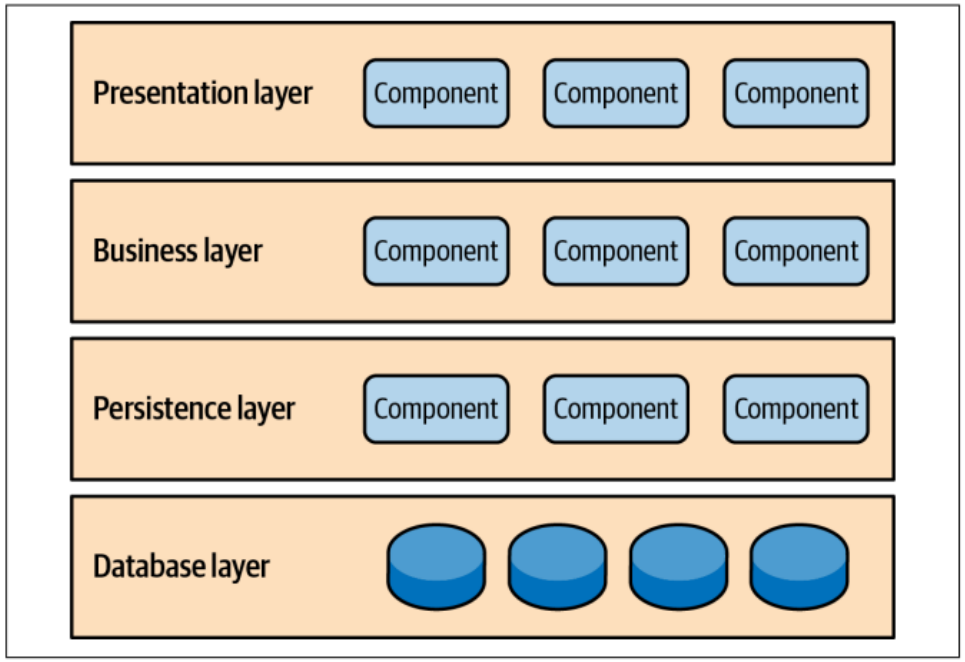


Layered Architecture

- most common architecture style
- Also known as the n-tier architecture
- Widely used because it aligns with traditional IT team structures
  - where teams are organized by technical domains (such as presentation teams, backend development teams, database teams, and so on).
- Follows Technical Partitioning

Layered Architecture

- Components organized into horizontal layers
  - each performing a specific role
- Mostly consist of four standard layers
  - Presentation
  - Business
  - Persistence
  - Database



Layered Architecture

- Presentation Layer
  - Handles user interface and browser communication logic
  - Displays information in a specific format
  - Abstracts away details of data retrieval

- Business Layer
  - Executes business rules for specific requests
  - Applies logic (e.g., calculations, aggregations)
  - Passes processed data to the presentation layer

Layered Architecture

- Persistence Layer
  - Manages data access and storage logic
  - Handles communication with the database layer
  - Abstracts database operations from the business layer
  - Provides APIs to retrieve or modify data

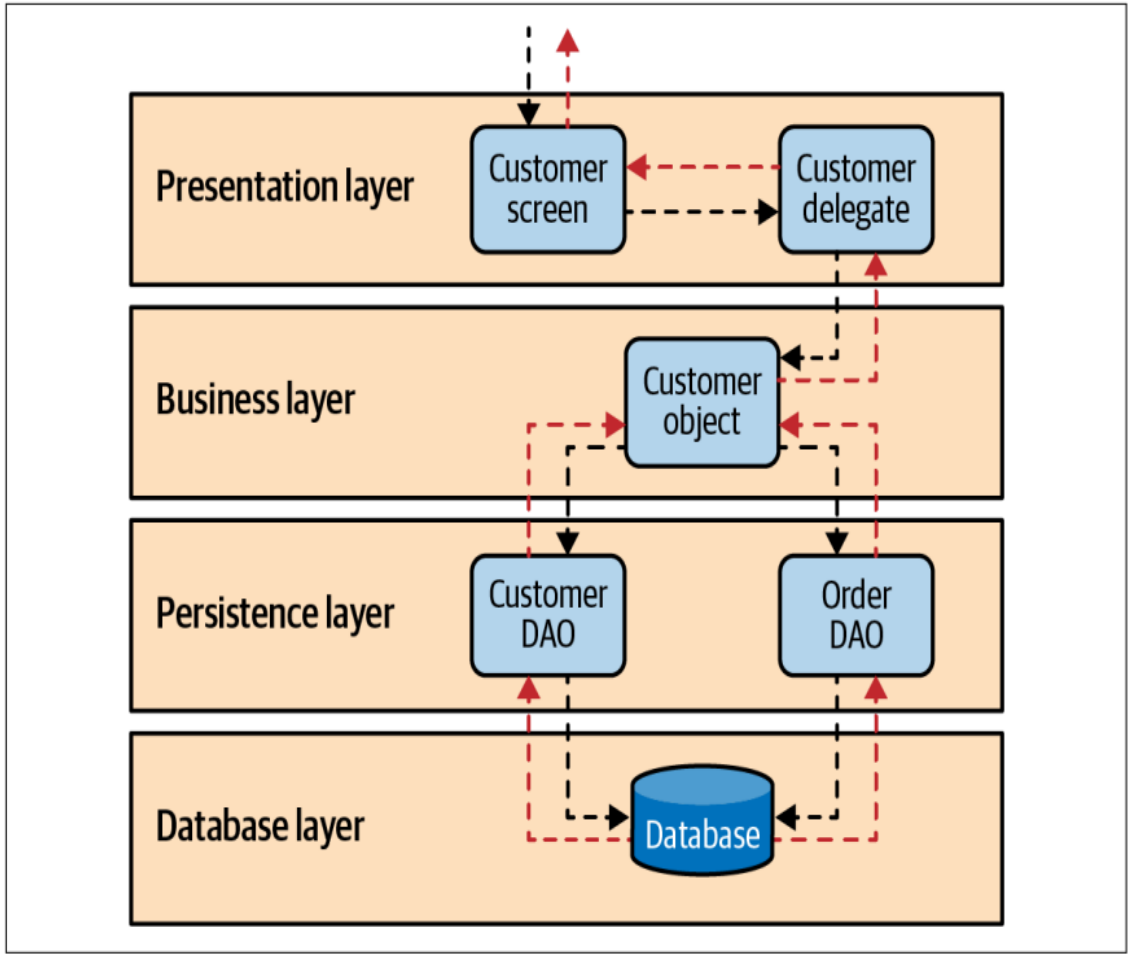
- Database Layer
  - Stores and organizes application data
  - Executes queries and transactions for data retrieval or modification

How Layered Architecture Works

- Scenario:
  - Business user requests customer information (e.g., customer and order data).

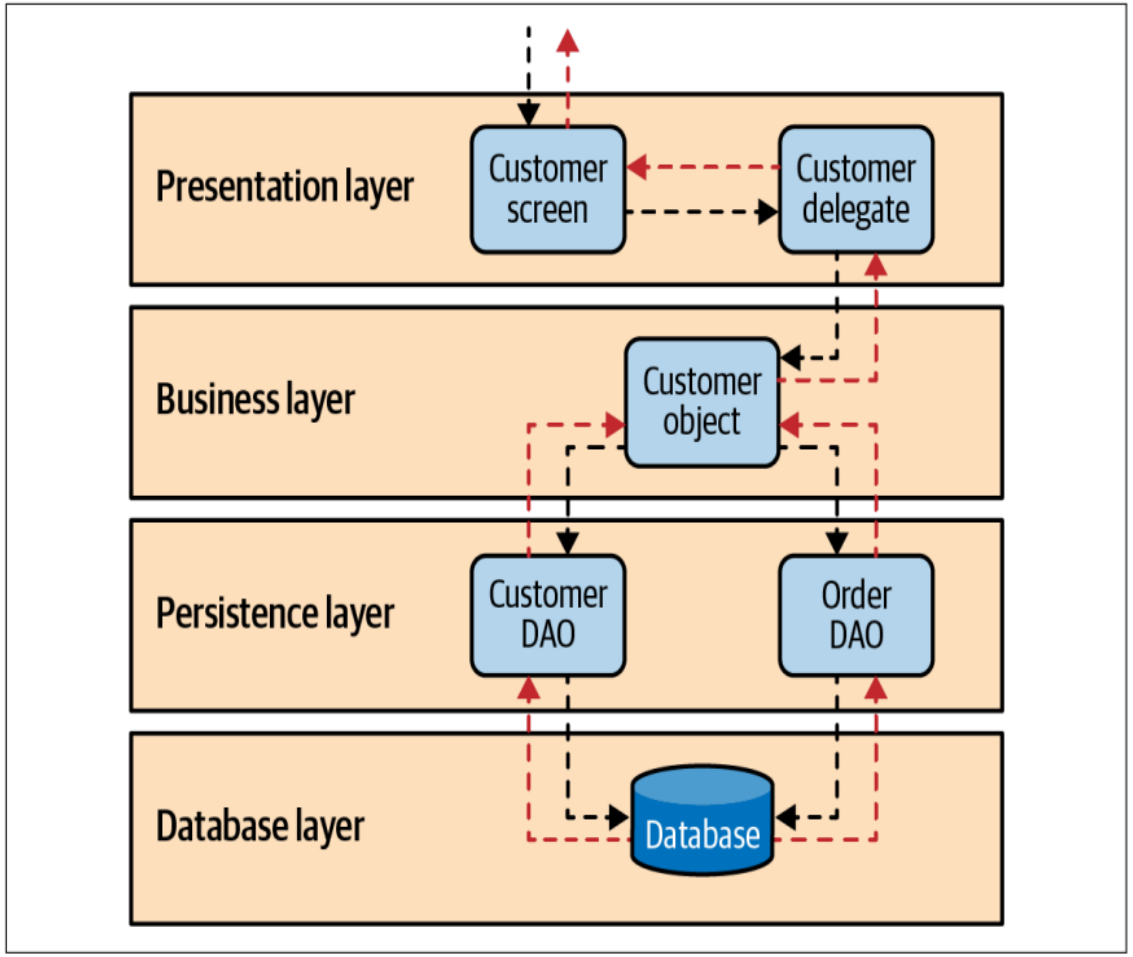
How Layered Architecture Works

- Presentation Layer
  - Customer Screen:
    - Accepts the request and displays the data.
    - Unaware of data location or retrieval process.
  - Customer Delegate:
    - Forwards request to the appropriate Business Layer module.



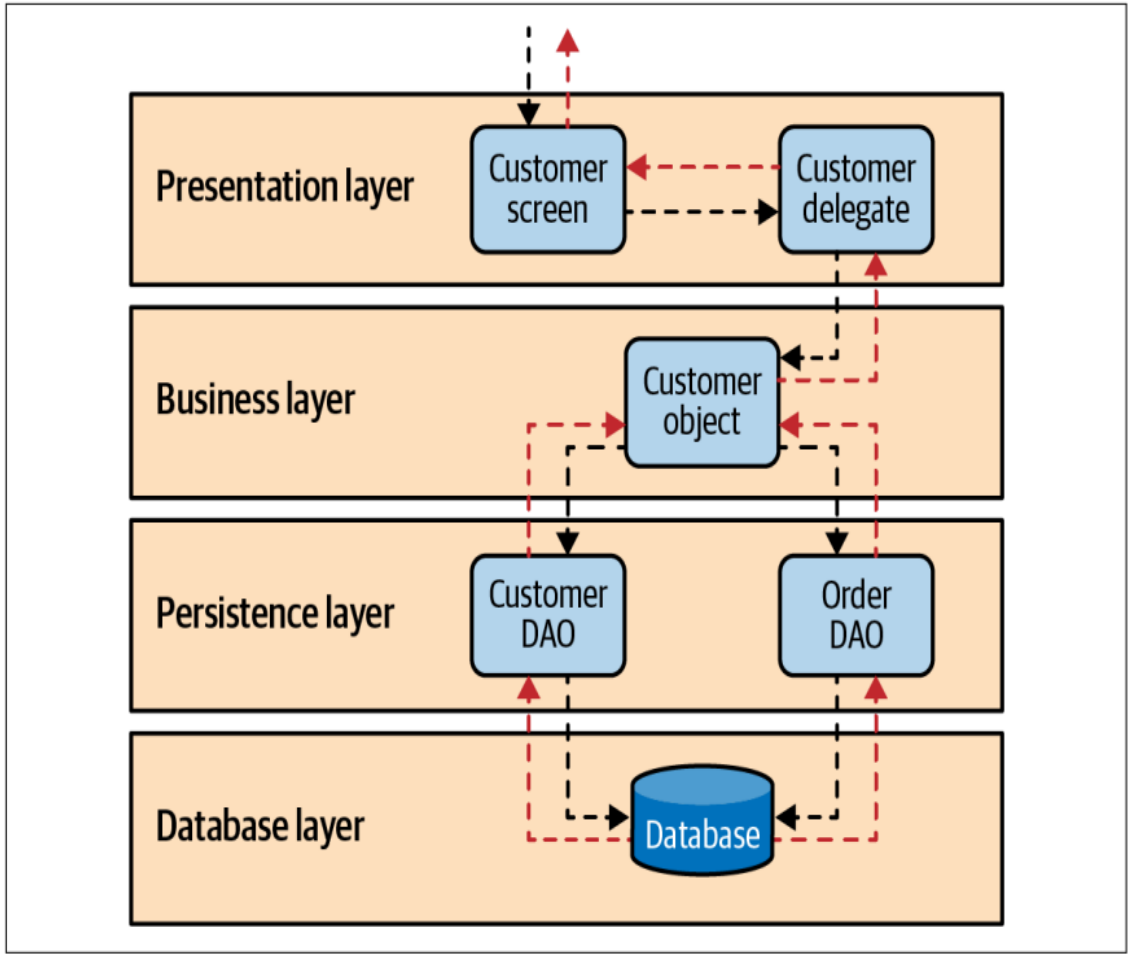
How Layered Architecture Works

- Business Layer:
  - Customer Object:
    - Aggregates customer and order data required for the request.
    - Delegates data retrieval to the Persistence Layer.



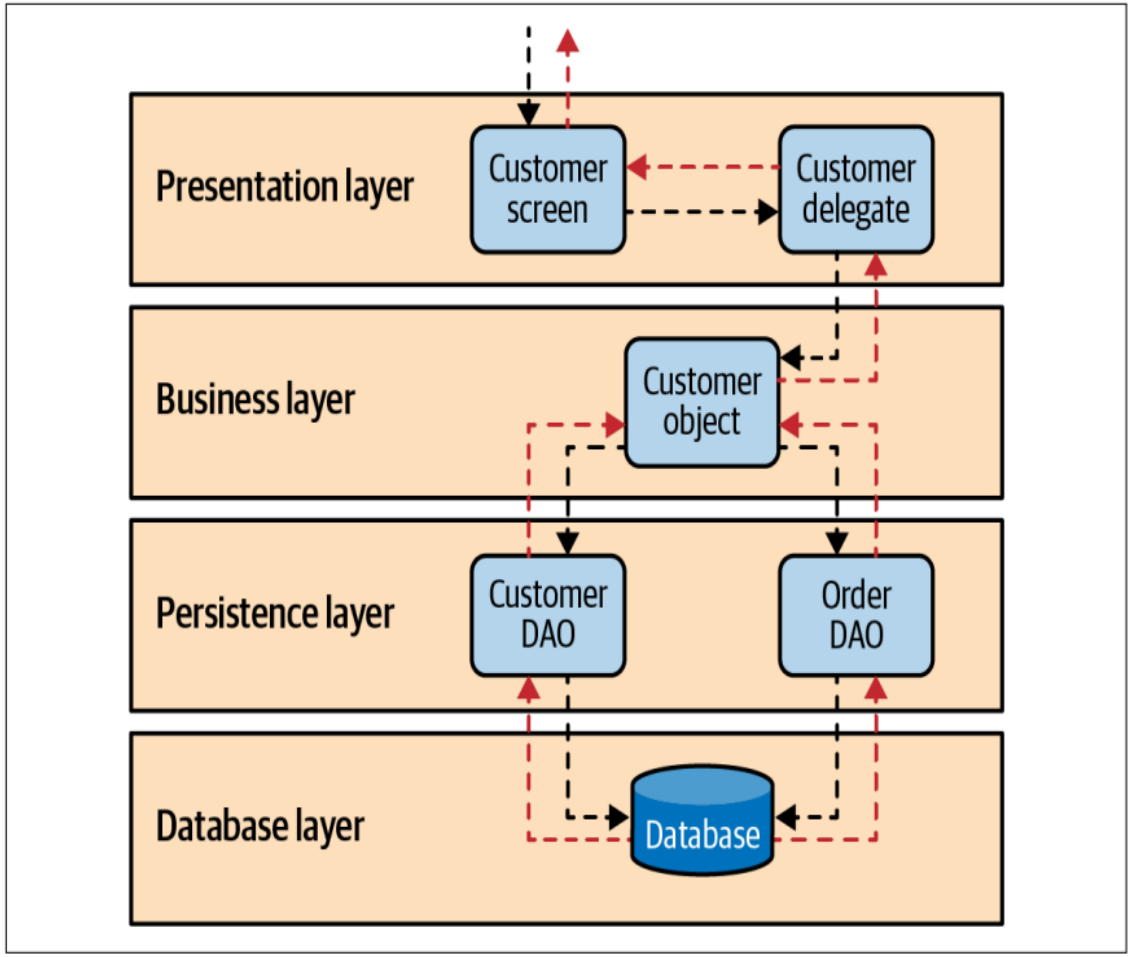
How Layered Architecture Works

- Persistence Layer:
  - Customer DAO:
    - Retrieves customer data from the database.
  - Order DAO:
    - Retrieves order data from the database.



How Layered Architecture Works

- Database Layer:
  - Executes SQL queries to fetch data.
  - Sends data back to DAO modules in the persistence layer.



Considerations and Analysis

- Advantages:
  - Well-understood and general-purpose.
  - Good starting point when unsure of the best style.
  - Suitable for projects with significant budget or time constraints.

- Challenges:
  - Risk of the Architecture Sinkhole Anti-Pattern
    - Requests pass through layers with minimal logic.
    - Ideal ratio: 80% requests with logic, 20% simple pass-through.
  - High coordination effort when multiple layers are impacted by a change.

When to Consider Layered Architecture

- When changes are isolated to specific layers
  - Example: Updating UI look-and-feel
- Aligned with technically partitioned teams
  - UI developers, backend teams, database team
- Monolithic systems with low operational concerns

When NOT to Consider Layered Architecture

- Operational Concerns
  - Applications requiring scalability, fault tolerance, and performance.
- Frequent Domain-Level Changes
  - Domain changes impact all layers (e.g., adding expiration dates to a customer’s movie list)
  - Requires coordination across multiple teams
- Domain-Partitioned Teams
  - Misaligned with cross-functional teams that manage all aspects for a specific domain

Layered Architecture: Brief Example
An E-commerce System

1. Presentation Layer

- Purpose:
  - Handles user interaction and the visual representation of data.
- Components/Services:
  - User Interface:
    - Displays product catalog, shopping cart, user profile, etc.
    - Technology: React.js, Angular, or Vue.js
  - API Gateway:
    - Acts as an entry point to connect frontend with backend services.
    - Technology: REST APIs

Layered Architecture: Brief Example
An E-commerce System

2. Business Logic Layer

- Purpose:
  - Implements the core logic of the system, such as product management, order placement, payment processing, etc.
- Components/Services:
  - ProductService: Manages product-related logic.
  - OrderService: Manages order placement and status updates.
  - PaymentService: Handles payment validation and coordination with the Integration Layer.
  - And so on…

Layered Architecture: Brief Example
An E-commerce System

3. Persistence Layer

- Purpose:
  - Abstracts interactions with the database
- Components/Services:
  - Data Access Objects (DAO): Encapsulates database operations for specific entities.
  - Technology: Hibernate (Java), Entity Framework (C#), or Sequelize (Node.js)

Layered Architecture: Brief Example
An E-commerce System

4. Database Layer

- Purpose:
  - Handles the actual storage and management of data in relational or NoSQL databases.
- Components/Services:
  - Relational Database:
    - Stores structured data (e.g., products, users, orders).
    - Technology: MySQL, PostgreSQL, or Amazon RDS
  - NoSQL Database:
    - Handles unstructured or semi-structured data (e.g., logs, user activity).
    - Technology: MongoDB, DynamoDB

Layered Architecture: Brief Example
An E-commerce System

5. Integration Layer

- Purpose:
  - Facilitates communication with external services and APIs, such as payment gateways, email notifications, and shipping providers.
- Components/Services:
  - Payment Gateway Integration:
    - Manages interaction with third-party services like PayPal or Stripe.
    - Technology: PayPal SDK, Stripe API
  - Email Notification Service:
    - Sends order confirmations and promotional emails.
    - Technology: SendGrid, Amazon SES
  - Shipping API Integration:
    - Provides tracking and logistics updates.
    - Technology: FedEx or DHL APIs

Layered Architecture: Brief Example
An E-commerce System

6. Infrastructure Layer

- Purpose:
  - Provides foundational support for hosting and deployment.
- Components/Services:
  - Hosting Platform:
    - Runs the application on a scalable cloud infrastructure.
    - Technology: AWS EC2, Azure, or Google Cloud
  - Containerization:
    - Ensures consistent deployment across environments.
    - Technology: Docker, Kubernetes
  - CI/CD Pipeline:
    - Automates the build, test, and deployment processes.
    - Technology: Jenkins

Reference

- Chapter 3 - Software Architecture Patterns, 2nd Edition by Mark Richards