Layered architecture and corresponding UML class diagram

Layered architecture

- Used to model the interfacing of sub-systems.
- Organizes the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

Layer 3: User Interface

Layer 2: User interface management, Authentication and authorization

Layer 1: Core business logic/application functionality, System utilities

Layer 0: System Support (OS, database, etc)

Overview of Layered architecture pattern:

Description: Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system.

When used: Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.

Advantages: Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.

Disadvantages: In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

Let's now focus on a particular type of Layered architecture: the Three-tier or Three-layered architecture pattern.

Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where application data is stored and managed.

- *Presentation tier*

The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application. Its main purpose is to display information to and collect information from the user. This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI), for example.

- *Application tier*

The application tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information that is collected in the presentation tier is processed - sometimes against other information in the data tier - using business logic, a specific set of business rules.

- *Data tier*

The data tier, sometimes called database tier, data access tier or back-end, is where the information that is processed by the application is stored and managed. This can be a relational database system or in a NoSQL Database server.

Important info: In a three-tier application, all communication goes through the application tier.

- The **Presentation Layer** should never directly access the **Data Layer**. All requests go through the **Application Layer**.
- **Application Logic** should act as a middleman, containing all business rules.

To further clarify "Three-tier" architecture, let's explore a specific example.
<u>Three-tier architecture example:</u>
Requirements:
E-commerce websites

- *Presentation Layer: The online storefront with product catalogs, shopping carts, and checkout interfaces.*
- *Logic Layer: Handles searching, order processing, inventory management, interfacing with 3rd-party payment vendors, and business rules like discounts and promotions.*
- *Data Layer: Stores product information, customer data, order history, and financial transactions in a database.*

Class Diagram in PlantUML for above problem with Three-tier architecture:

```
@startuml
class Storefront {
        - ProductCatalog : List<Product>
        - ShoppingCart : ShoppingCart
        - CheckoutInterface : Checkout
        + viewProducts() : void
        + addToCart(product: Product) : void
        + proceedToCheckout() : void
}
package "Logic Layer" {
        class OrderProcessor {
        - paymentProcessor : PaymentProcessor
        - inventoryManager : InventoryManager
        + processOrder(order: Order) : void
        + applyDiscount(order: Order, discountCode: String) : float
        }
        class InventoryManager {
        + checkStock(product: Product) : boolean
        + updateStock(product: Product, quantity: int) : void
        }

        class PaymentProcessor {
        + processPayment(paymentInfo: PaymentInfo) : boolean
        + connectTo3rdParty(paymentInfo: PaymentInfo) : boolean
        }
}

class Database {
        + storeProductData(product: Product) : void
        + retrieveProductData(productId: int) : Product
        + storeCustomerData(customer: Customer) : void
        + storeOrderHistory(order: Order) : void
        + retrieveOrderHistory(customerId: int) : List<Order>
}


Storefront --> OrderProcessor : Uses
OrderProcessor --> PaymentProcessor : Uses
OrderProcessor --> InventoryManager : Uses
OrderProcessor --> Database : Accesses
InventoryManager --> Database : Accesses
PaymentProcessor --> Database : Accesses
@enduml
```