**Universidad de las Fuerzas Armadas ESPE**

**Departamento: Ciencias de la Computación**

**Carrera : Ingeniería de Software**

**Taller académico Nª: 2**

**1. Información General**

* **Asignatura:** Análisis y Diseño
* **Apellidos y nombres de los estudiantes:** Joan Cobeña, Juan Pasquel, Ruben Benavides, Edison Verdesoto
* **NRC:** 22426
* **Fecha de realización:** 12**/**06**/**2025

**2. Objetivo del Taller y Desarrollo**

**Objetivo del Taller:**  
Analizar el patrón de arquitectura de software Layered (en capas), a través de una investigación profunda con el fin de reconocer su contexto de aplicación y utilidad.

**Desarrollo:**

**Patrón de Arquitectura Layered**

La arquitectura Layered permite modelar un sistema como un conjunto de capas, cada una con un propósito específico, trabajando de manera única y con un grado de dependencia adyacente.

El patrón organiza el sistema en un conjunto de capas horizontales, donde cada capa proporciona servicios a la capa superior y actúa como cliente de la capa inferior. Según Ian Sommerville (2016) en su libro "Software Engineering", este patrón es uno de los más utilizados para estructurar aplicaciones empresariales.

## Características Principales

* **Separación de responsabilidades**: Cada capa tiene una responsabilidad específica y bien definida.
* **Abstracción incremental**: Las capas superiores son más abstractas que las inferiores.
* **Comunicación restringida**: Una capa solo puede comunicarse con capas adyacentes.
* **Reemplazabilidad**: Las capas pueden ser reemplazadas sin afectar al resto del sistema.

Además Martin Fowler (2002) añade que este patrón facilita:

* La estandarización de interfaces entre capas.
* La reutilización de capas en diferentes sistemas.
* El desarrollo paralelo por equipos especializados.

y en "Software Architecture in Practice" escrito por Len Bass et al. (2012) se destaca de esta arquitectura su:

* Modificabilidad mejorada
* Portabilidad aumentada
* Testabilidad simplificada

## Capas Típicas

1. **Capa de Presentación (UI)**
   1. Interfaz de usuario
   2. Validación de entrada básica
   3. Formateo de datos para visualización
2. **Capa de Lógica de Negocio**
   1. Reglas de negocio
   2. Procesamiento de datos
   3. Validaciones complejas
3. **Capa de Acceso a Datos**
   1. Comunicación con bases de datos
   2. Mapeo objeto-relacional
   3. Caché de datos
4. **Capa de Base de Datos**
   1. Almacenamiento persistente
   2. Procedimientos almacenados
   3. Triggers y constraints

**Aplicación en la Industria**

El patrón de arquitectura Layered es ampliamente utilizado en diversos sectores de la industria del software debido a su capacidad para proporcionar una estructura modular, escalable y fácil de mantener. En sistemas empresariales y aplicaciones web, este patrón es crucial para la gestión de la complejidad y la facilitación del mantenimiento de grandes proyectos de software. A continuación, se describen algunas aplicaciones clave de este patrón en la industria.

1. Desarrollo de Aplicaciones Empresariales:

La arquitectura Layered es frecuentemente aplicada en sistemas de gestión empresarial (ERP), sistemas de planificación de recursos (CRM), y aplicaciones financieras debido a su estructura modular, lo que facilita la implementación de funcionalidades complejas y permite un mantenimiento y actualización continuos sin afectar otras partes del sistema. Según Bass et al. (2012), la separación de responsabilidades entre capas ayuda a gestionar aplicaciones de gran escala, mejorando la interoperabilidad entre diversos servicios internos y externos.

2. Aplicaciones Web:

En el desarrollo de aplicaciones web, este patrón se utiliza para separar la interfaz de usuario (capa de presentación), la lógica de negocio, y la gestión de datos, lo que permite un desarrollo más eficiente y escalable. La comunicación restringida entre capas facilita la integración con diferentes tecnologías y permite que los desarrolladores puedan trabajar de manera más eficiente. Fowler (2002) destaca la importancia de la separación entre la capa de presentación y la capa de lógica de negocio en aplicaciones web modernas, asegurando la adaptabilidad a futuras mejoras.

3. Microservicios:

En sistemas basados en microservicios, el patrón Layered puede usarse para estructurar cada microservicio de forma independiente con capas claramente definidas. Cada microservicio puede tener su propia base de datos, capa de lógica de negocio y presentación, y comunicarse con otros servicios a través de APIs bien definidas. Esta separación mejora la mantenibilidad, escalabilidad y permite a los equipos de desarrollo trabajar en paralelo, lo cual es esencial en arquitecturas distribuidas.

4. Desarrollo de Software para la Nube:

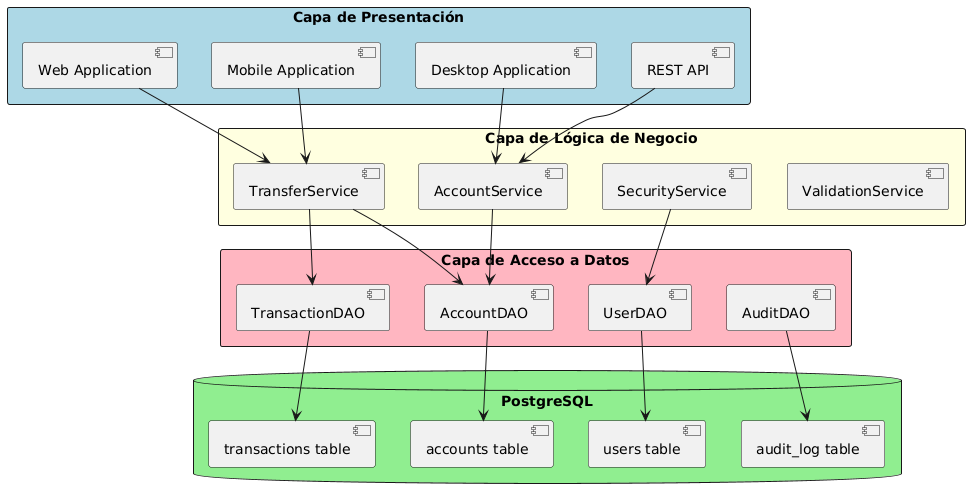
La adopción del patrón Layered también es común en el desarrollo de aplicaciones basadas en la nube, donde cada capa puede corresponder a un servicio específico dentro de la infraestructura en la nube. Esto permite que el sistema se escale dinámicamente, optimizando recursos según las necesidades del negocio. Bass et al. (2012) señalan que este patrón mejora la portabilidad de las aplicaciones, permitiendo migraciones fáciles entre plataformas de nube.

## Ejemplo: Sistema Bancario Online

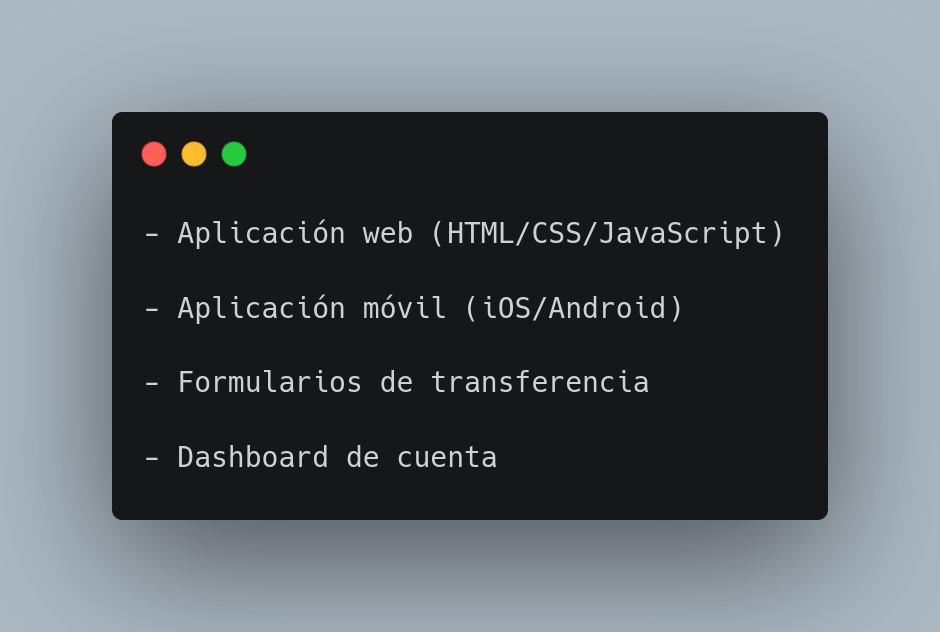
### Contexto

Un banco necesita desarrollar un sistema para que sus clientes realicen operaciones bancarias por internet, para lo cuál se desarrolla un sistema con una arquitectura con patrón de capas.

Modelando a grandes rasgos podemos tener los siguiente c



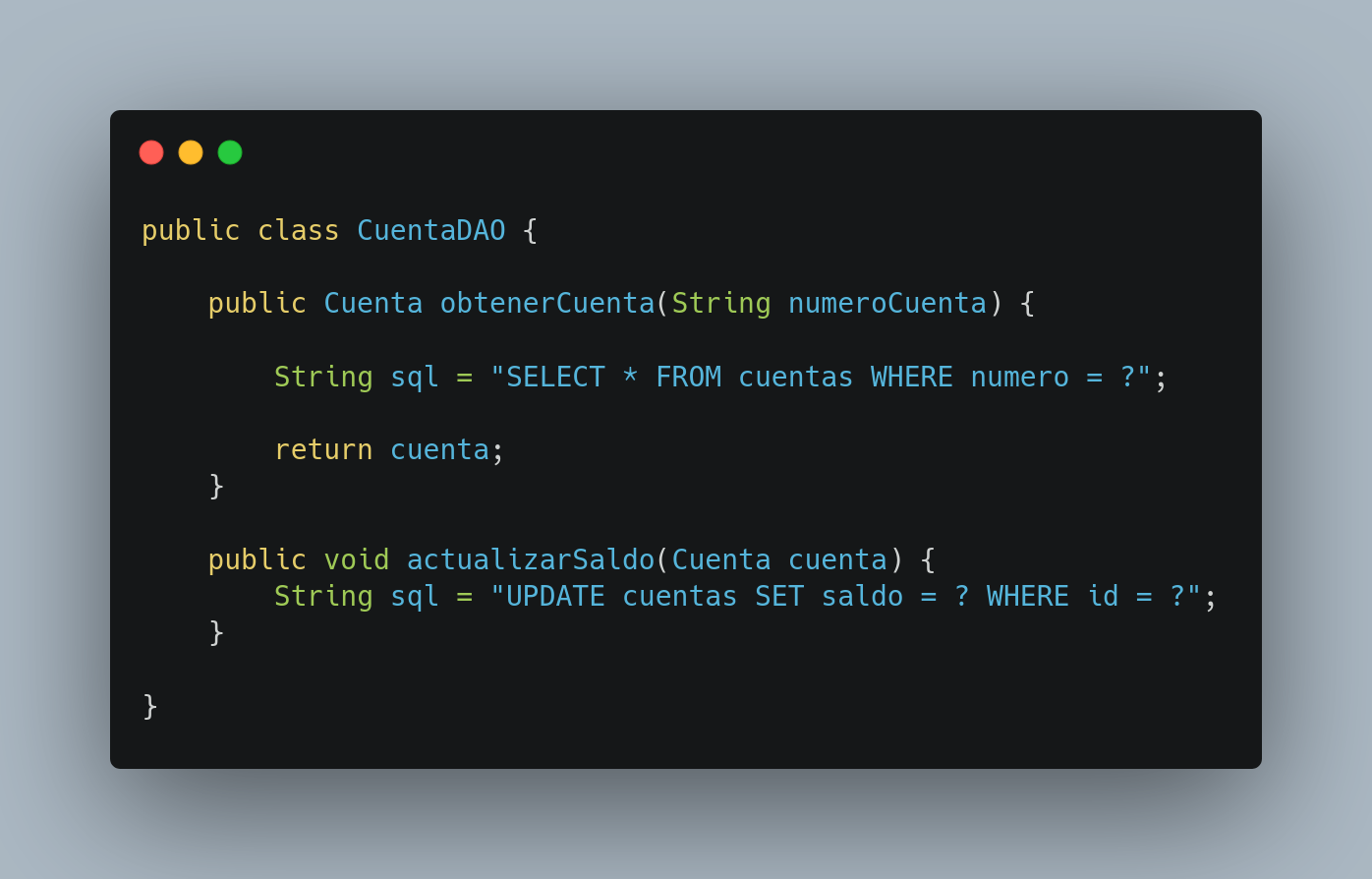
**1. Capa de Presentación**

****

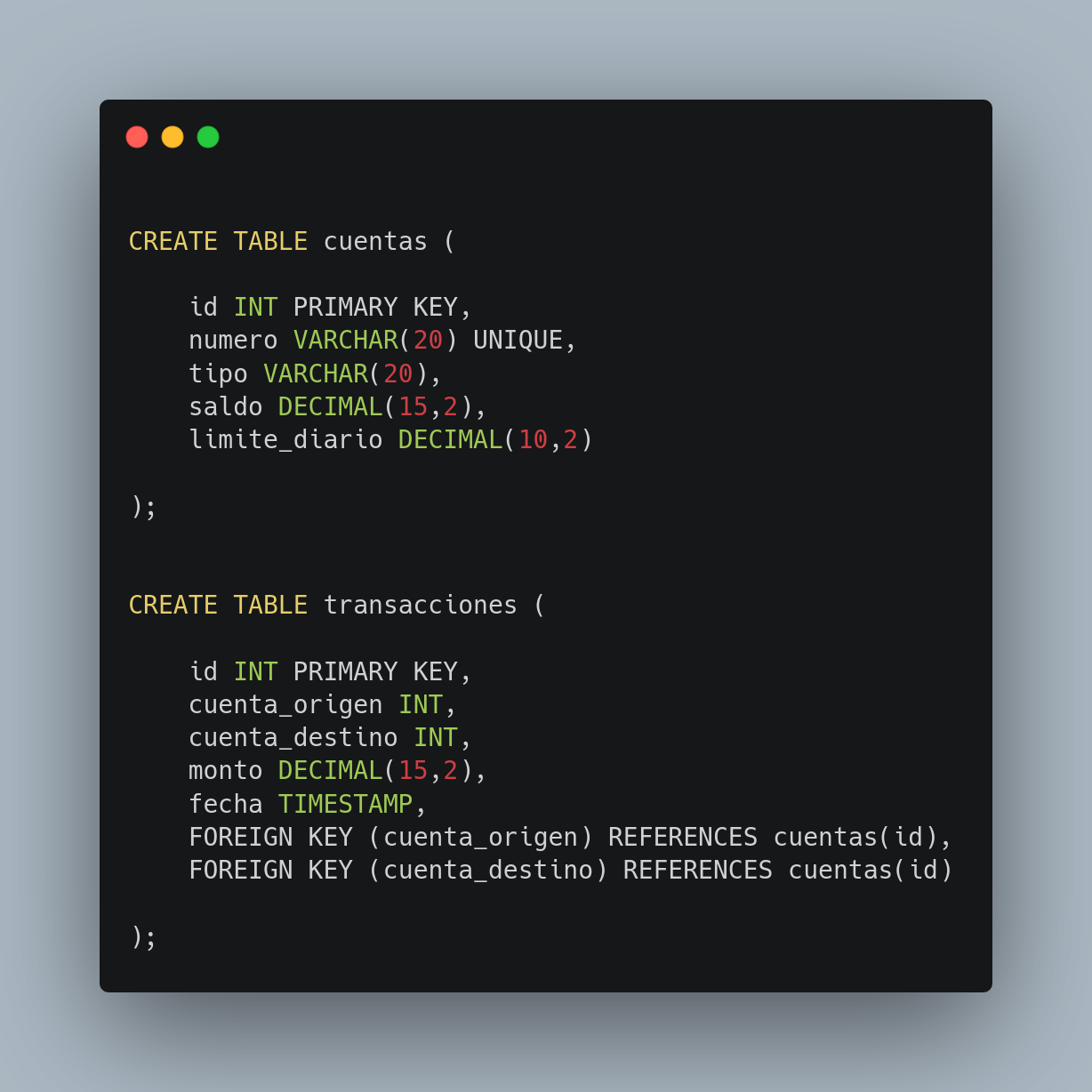
**2. Capa de Lógica de Negocio**



**3. Capa de Acceso a Datos**



**4. Capa de Base de Datos**

****

### Flujo de Operación

1. **Usuario** ingresa datos de transferencia en la interfaz web
2. **Capa de Presentación** valida formato básico y envía a la capa de negocio
3. **Capa de Negocio** aplica todas las reglas bancarias:
   * Verifica fondos disponibles
   * Calcula comisiones
   * Registra la transacción
4. **Capa de Acceso a Datos** traduce las operaciones a consultas SQL
5. **Base de Datos** persiste los cambios

### Ventajas en este Ejemplo

* **Seguridad**: La lógica crítica está aislada de la interfaz
* **Mantenibilidad**: Cambios en reglas bancarias no afectan la UI
* **Escalabilidad**: Cada capa puede escalarse independientemente
* **Reutilización**: La misma lógica sirve para web y móvil

**3. Conclusiones**

* La implementación del patrón de arquitectura Layered proporciona una estructura modular que facilita el mantenimiento del sistema a lo largo del tiempo. Cada capa tiene una responsabilidad bien definida, lo que permite realizar modificaciones en una capa sin afectar las demás, garantizando así la estabilidad del sistema a largo plazo.
* Este patrón favorece la escalabilidad del sistema, ya que las capas pueden ajustarse y optimizarse de manera independiente. Además, la modularidad facilita la integración de nuevas funcionalidades sin comprometer la integridad del sistema. Esto es esencial en aplicaciones empresariales que requieren un crecimiento continuo y una fácil adaptación a cambios en el mercado.
* Con la separación de las capas, especialmente en aplicaciones como sistemas bancarios, se puede asegurar que la lógica crítica no se vea afectada por los cambios en la interfaz de usuario. Esto mejora la seguridad y permite la reutilización de la lógica de negocio en diferentes plataformas (web, móvil), optimizando el rendimiento y la consistencia.

**4. Referencias (Norma APA 7.0)**

* **Sommerville, I.** (2016). *Software Engineering* (10th ed.). Pearson Education. Capítulo 6: Architectural Design.
* **Fowler, M.** (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional. Páginas 19-25.
* **Bass, L., Clements, P., & Kazman, R.** (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley Professional. Capítulo 13: Architecture Patterns.
* **Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M.** (1996). *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Wiley & Sons. Páginas 31-52.
* **Richards, M. & Ford, N.** (2020). *Fundamentals of Software Architecture*. O'Reilly Media. Capítulo 10: Layered Architecture Style.