ENTREGA #3

MODELO C4 Y PRINCIPIOS SOLID

SARA XIMENA CONTRERAS QUIROGA

KAROL STEFANY ORDOÑEZ PEÑALOZA

YEFERSON PIÑEROS PEÑA

02/03/2025

CARLOS EDUARDO MUJICA REYES

PREGUNTAS ORIENTADORAS

1. Which one of the following options is needed in the lifecycle of software architecture process:
    a. Investigation
    b. Analysis
    c. prerequisites
    d. Induction

2. Which of the following are correct statements:
    a. An architecture may or may not defined components
    b. An architecture is not dependable on requirements
    c. An architecture is foremost an abstraction of a system that suppresses details of the components that do not affect how they are used.
    d. Invocation based programming sequences.

ACTIVIDAD DE TRABAJO AUTÓNOMO

Leer el siguiente artículo de IEEE: “Linux page fault analysis in android systems” Linux page fault analysis in android systems - ScienceDirect (umb.edu.co) Realizar una infografía del artículo o un resumen de 2 páginas con los puntos más relevantes, pueden usar gráficos.

Linux page fault analysis in android systems

01. Contexto

¿Que es un fallo de pagina?

- Ocurre cuando una aplicación intenta acceder a una página de memoria virtual que aún no ha sido cargada en la memoria principal (RAM).
- El sistema operativo tiene que buscar esa página en el almacenamiento interno (disco) y cargarla, lo que puede causar retrasos y afectar el rendimiento del sistema.

02. Almacenamiento

- Entre 30% y 40% del tiempo que tarda un fallo de página mayor se debe al acceso al almacenamiento (UFS).
- El resto del tiempo lo consume el sistema operativo Linux en la gestión interna del fallo de página.

03. Mecanismos Readhead

- Cuando ocurre un fallo de página, Linux no solo carga la página solicitada, sino varias páginas contiguas (pre-lectura o ReadAhead).
- Sin embargo, solo el 30% de esas páginas pre-cargadas son realmente utilizadas.

04. Tamaño Venta

Ventanas pequeñas de ReadAhead (cargar menos páginas anticipadamente) reducen el tiempo por fallo de página, pero aumentan el número total de fallos, ya que el sistema debe ir al almacenamiento más veces.

04. Usuarios

- Fallos de página mayores pueden bloquear hilos críticos, afectando directamente la fluidez de las apps.
- Google reconoce estos fallos como una fuente de problemas de jank (tirones o congelamientos en la interfaz).

05. ¿Importante?

- Es uno de los primeros trabajos que cuantifica directamente cómo los fallos de página mayores afectan el rendimiento en smartphones Android comerciales.
- Proporciona una herramienta para medir ineficiencias y optimizar el uso de almacenamiento en futuros dispositivos.

PROCEDIMIENTO Y METODOLOGÍA DE LA PRÁCTICA

1. Modelo Arquitectura

- Breve descripción de las funcionalidades del ERP.

  1. Compras/Inventario: Permitirá tener un control sobre los cambios en el inventario para gestionar los productos que se encuentren ya en stock de venta, la bodega o por ingresar como una nueva compra. Esta funcionalidad también incluye productos

  por separado como materia prima.

  características principales:

  1. Registro de productos por categoría
  2. Control de cantidades y pedidos
  3. Registro de entrada y salida

  2. Ventas: Mantiene al día el seguimiento de mercancías por motivos de venta (incluye pagos electrónicos) garantizando un registro de clientes y productos.

  características principales:

  1. Registro de venta.
  2. Generación de Factura.
  3. Control de pagos y devoluciones
  4. Actualización de salida de materia prima

  3. Empleados: Controla las actividades, cargos y accesos que puedan tener los diferentes empleados de la empresa.

  características principales:

  1. Registro de empleados
  2. identificación de accesos

3. Tipos de empleados

4. Cargo/pagos

4. Facturas: Maneja la generación, archivo y administración de facturas para los módulos de 1, 2, 5 a detalle según la necesidad

características principales:

1. Generación de factura por venta

2. Generación de factura por compra

3. Detalle de proveedor/cliente/producto

4. Validación de cambio de producto por dinero

5. Proveedores/Clientes: Permite la correcta relación con los módulos 1, 2, 4 definiendo además las diferencias y beneficios de un tipo de cliente (casual, frecuente, mayorista)

características principales:

1. Registro de Cliente/proveedor

2. Tipo de cliente

3. Histórico de terceros

4. Manejo de Descuentos/Beneficios a tipos de clientes

- Trade offs

Trade- Off= Gestion Inventario Manual vs Automatico		
Opcion	Ventajas	Desventajas
Manual (carga por admi)	Simplicidad inicial	Riesgo de error humano. mas trabajo operativo
Automatica (con Provedor)	Sincronizacion en tiempo real. menos errores	Requiere coordinacion con proveedores y desarrollo adicional
Plan=Manual al inicio. luego integracion automatica con los principales provedores		

Trade- Off= Sistema de notificaciones		
Opcion	Ventajas	Desventajas
sistema interno	Control total sin costos variables	mayor desarrollo y mantenimiento
servicio externo	Integracion rapida.alta confiabilidad	costas por uso. dependencia del proveedor
Plan=Firebase para notificaciones Push y SMS		

Trade- Off= Arquitectura Backend		
Opcion	Ventajas	Desventajas
Monolito	simplicidad inicial. menor complejidad	Dificultad de escalar partes especificas. riesgo de cuellos de botella
Microservicios	Escalabilidad por componente mayor resiliencia	Mayor complejidad inicial. mas costos de intraestructura
Plan=Monolito para Primera version. Preparado para migrar a microservicios		

Trade- Off= App-Cliente		
Opcion	Ventajas	Desventajas
Android/IOS	mejor rendimiento. mejor UX. acceso a notificaciones push	Mayor costo de desarrollo. requiere doble equipo
Web App	Un solo desarrollo. compatible con cualquier dispositivo	Limitado acceso a funciones nativas
Plan=Web app para reducir costo inicial y evaluar App Movil		

Trade- Off= Base de datos		
Opcion	Ventajas	Desventajas
sql	relaciones fuertes ideal para transaciones y reportes	Escalabilidad horizontal mas compleja
no sql	flexible. ideal para productos con estructuras cambiantes	menos eficientes para reportes complejos
Plan=SQL para asegurar integridad de pedidos e inventario. NoSQL para catalogo		

Trade- Off= Cloud		
Opcion	Ventajas	Desventajas
Servidor propio	Control total Pagos fijos	Mantenimiento propio. dificil escalar
Cloud	Escalabilidad automaticas. Pagos por uso	Costos variables. dependencia del Proveedor
Plan=Cloud para Mayor flexibilidad y capacidad de crecimiento		

- Tecnologías seleccionadas

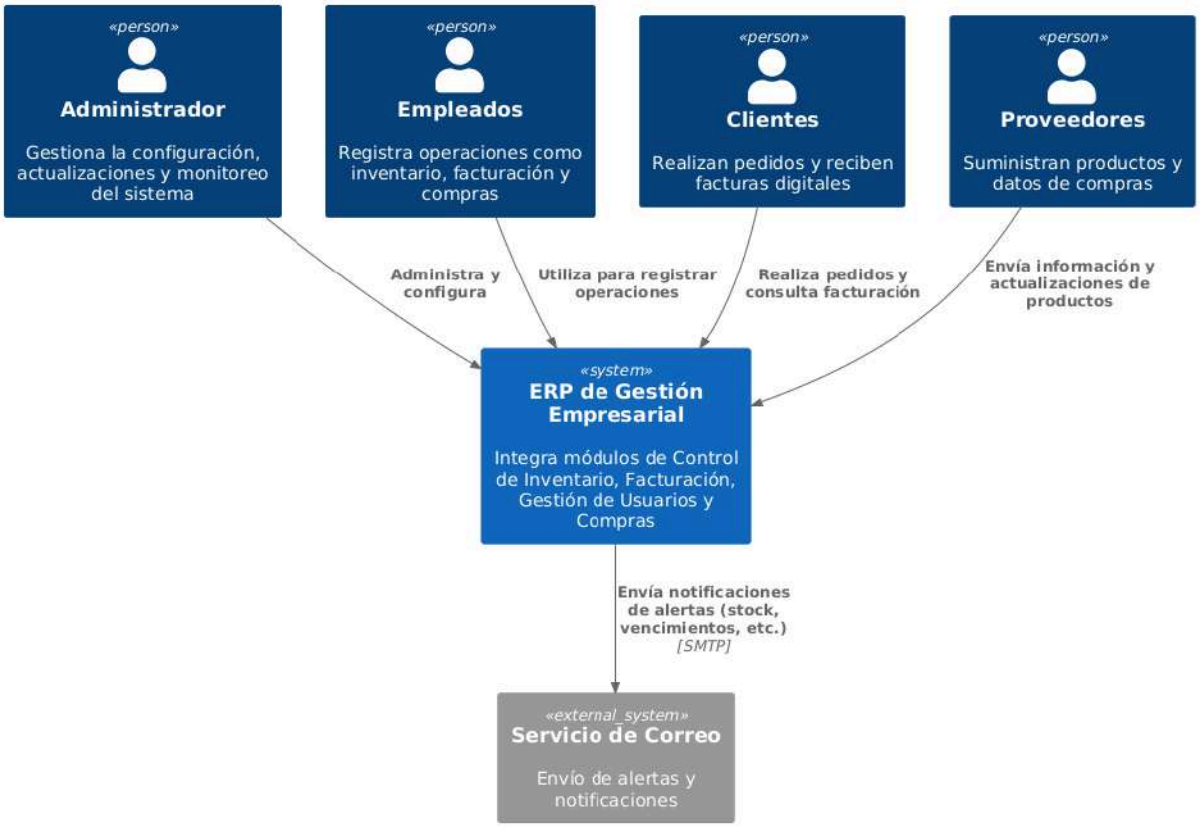
  BackEnd: Java

  FrontEnd: HTML

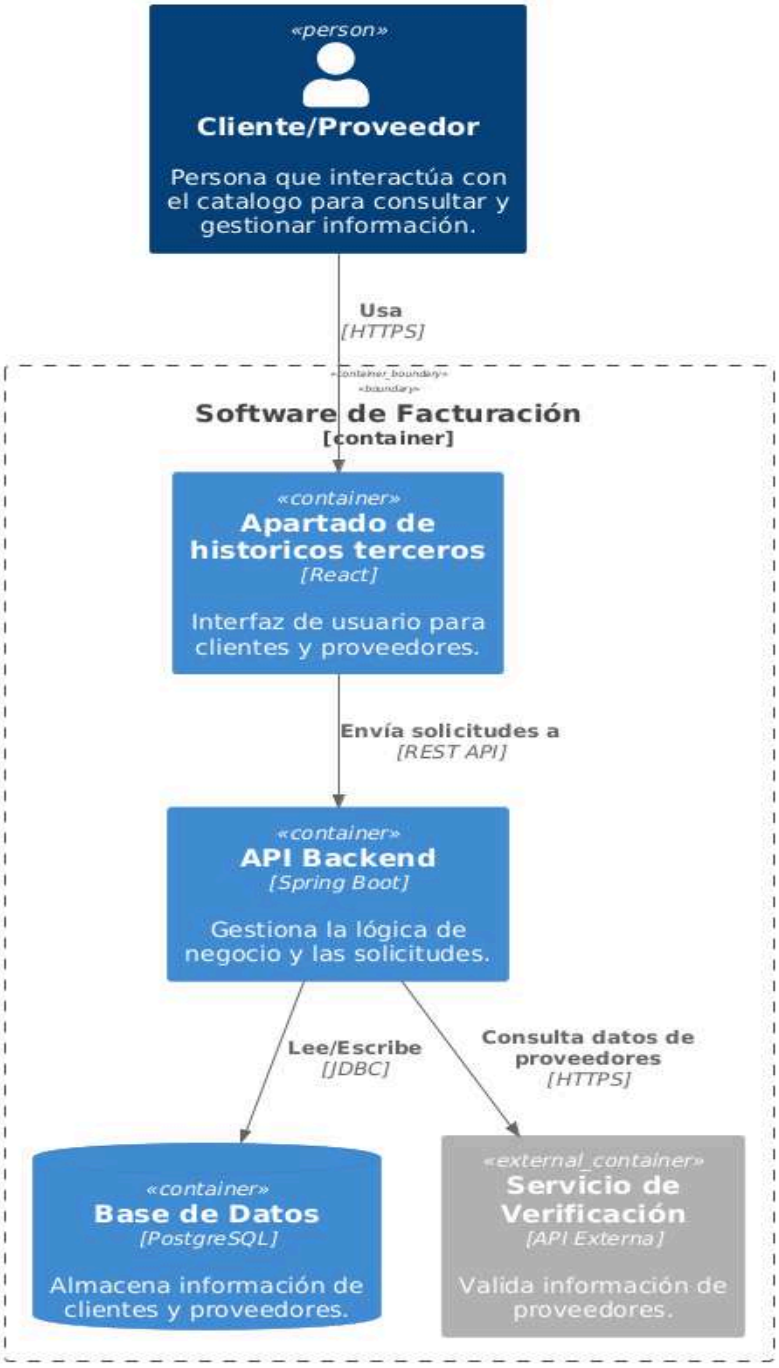
  Base de datos: MySQL/PHP

- Un ejemplo aplicable a su proyecto de cada uno de los diagramas del Modelo C4.

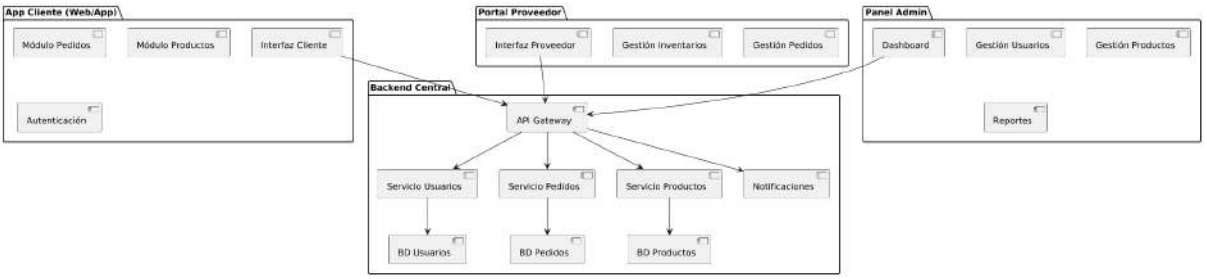
  - Diagrama de contexto del sistema



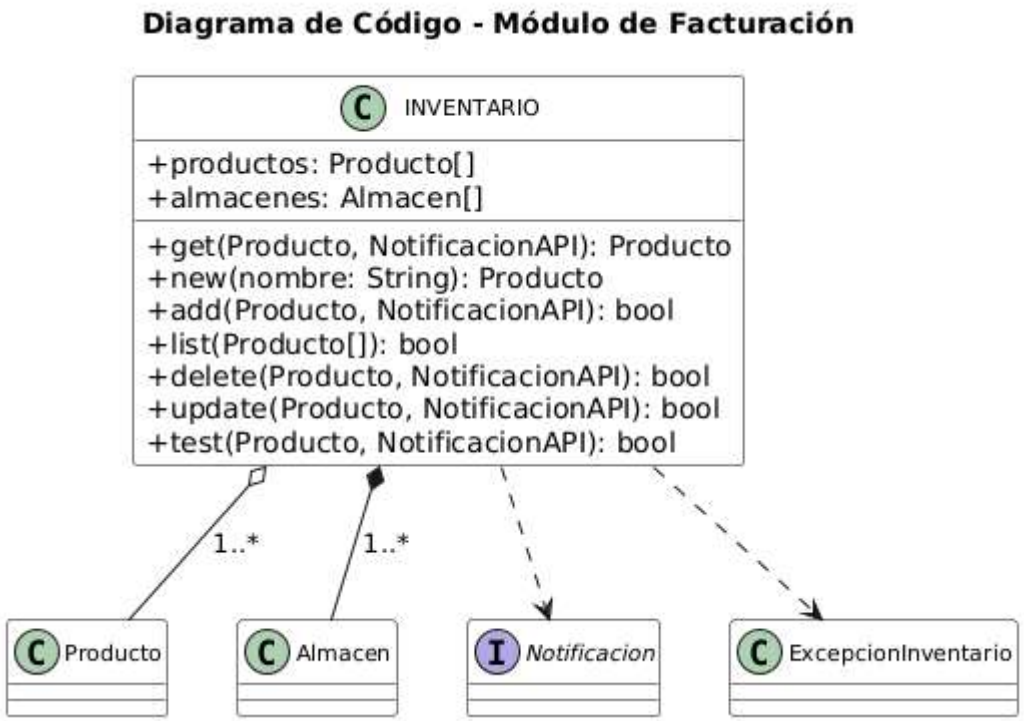
- Diagrama de contenedor



- Diagrama de componentes



- Diagrama de código



2. Patrones de Diseño

- Patrones de diseño aplicados en el proyecto

- Patrón Singleton

Aplicación: Gestión de conexión a la base de datos.

Descripción: Garantiza que solo exista una única instancia de la conexión a la base de datos, evitando sobrecarga de conexiones.

Uso: En el backend, el servicio de inventario y pedidos accede a la misma instancia de conexión.

Beneficio: Ahorra recursos y asegura un acceso centralizado y controlado.

- Patrón Factory

Aplicación: Creación de objetos de productos (Perfumes).

Descripción: Centraliza la lógica de creación de objetos, por ejemplo, perfumes de distintos tipos (Eau de Toilette, Eau de Parfum).

Uso: Cuando un producto se agrega al catálogo, la Factory decide qué tipo de objeto crear en base a atributos (fragancia, marca, tipo).

Beneficio: Flexibilidad para ampliar el catálogo sin modificar mucho el código.

- Patrón Observer

Aplicación: Notificaciones al cliente.

Descripción: Permite que distintos componentes (cliente web, app móvil) se suscriban a eventos, como actualizaciones de inventario o estado de pedidos.

Uso: Cuando un pedido cambia de estado (procesando, enviado, entregado), el sistema notifica automáticamente a las interfaces (email, notificación push).

Beneficio: Mantiene desacoplado el flujo de notificación y facilita agregar nuevos canales (WhatsApp, SMS).

- Patrón MVC (Model-View-Controller)

Aplicación: Estructura de la aplicación web/app.

Descripción: Divide la aplicación en:

Modelo: Productos, Pedidos, Clientes.

Vista: Interfaces Web/App.

Controlador: Lógica de negocio (procesar pagos, gestionar inventario).

Beneficio: Separación de responsabilidades, más fácil mantenimiento y pruebas.

- Patrón Strategy

Aplicación: Métodos de pago.

Descripción: Define una familia de algoritmos (Pago con tarjeta, Pago con PayPal, Pago contraentrega) y permite cambiar entre ellos en tiempo de ejecución.

Uso: Al finalizar el pedido, el cliente elige un método de pago y el sistema ejecuta la estrategia correspondiente.

Beneficio: Flexibilidad para añadir nuevos métodos de pago sin modificar el flujo de pedidos.

- Patrón Decorator

Aplicación: Promociones y descuentos aplicados a productos.

Descripción: Añade funcionalidades extra (descuentos, empaques especiales) a productos sin modificar su estructura.

Uso: Un perfume puede tener un descuento de temporada o un envoltorio de regalo sin alterar el objeto base.

Beneficio: Extiende funcionalidades sin alterar clases existentes.

- Patrón Command

Aplicación: Gestión de operaciones administrativas (actualizar inventario, registrar promociones).

Descripción: Encapsula cada operación administrativa como un comando, lo que permite deshacer, repetir o programar comandos.

Uso: Un administrador puede registrar una promoción, y el comando queda registrado para auditar o revertir.

Beneficio: Mejor control y flexibilidad en operaciones críticas.

- Patrón Adapter:

Aplicación: Integración con proveedores externos.

Descripción: Adapta las interfaces de los sistemas de proveedores a la estructura interna de tu aplicación.

Uso: Cada proveedor tiene un formato distinto para su catálogo, el Adapter traduce esos formatos al estándar interno.

Beneficio: Permite integrar nuevos proveedores sin alterar el sistema central.

· Patrón Proxy

Aplicación: Acceso a la API de Inventario.

Descripción: Un proxy controla el acceso a la API de inventario, permitiendo cachear o controlar permisos.

Uso: Si varios componentes (web, app, panel admin) consultan el inventario, el proxy optimiza el acceso.

Beneficio: Mejora el rendimiento y seguridad.

· Patrón Builder

Aplicación: Construcción de pedidos complejos.

Descripción: Permite crear objetos de pedidos con múltiples opciones de personalización (productos, empaques, mensajes personalizados).

Uso: Al construir un pedido, el Builder gestiona la personalización y garantiza que sea válido.

Beneficio: Maneja pedidos complejos sin sobrecargar el código.

- Principios Solid aplicados en el proyecto (ver explicación más adelante).

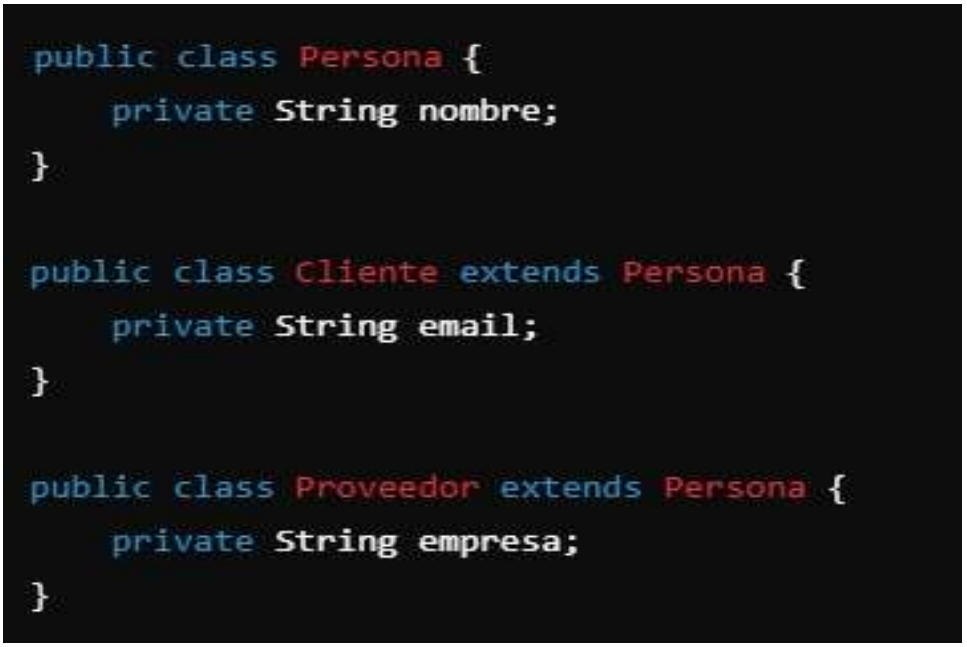
Los principios SOLID a implementar para la perfumería KYX ayudarán a mejorar la mantenibilidad del proyecto así como la escalabilidad y la claridad del cogido en el mismo

1. Principio de Responsabilidad Única: Este principio nos dice que cada componente debe tener una única responsabilidad y una única razón de cambio en su estado, por lo tanto:

- API Backend debe encargarse solo de la lógica de negocio (gestión de clientes y proveedores).
- Base de Datos solo almacena información.
- Servicio de Verificación sólo válido para proveedores externos.

Si en el backend mezclamos la validación de datos, el acceso a la base de datos y la lógica de negocio en una sola clase, será difícil de modificar y probar.

2. Principio de Sustitución Liskov: Usaremos este principio para gestionar módulos que tenemos en conjunto como Compras/inventario y Cliente/Proveedor, esto nos ayudará ya que si decidimos usar herencia las clases hijas deben poder sustituir a las clases principales sin errores. Un ejemplo podría ser:
    - Cliente y Proveedor pueden heredar de una clase base Persona sin alterar el comportamiento esperado.



```
public class Persona {
    private String nombre;
}

public class Cliente extends Persona {
    private String email;
}

public class Proveedor extends Persona {
    private String empresa;
}
```

3. Principio de Segregación de Interfaces: Al tener en uso el principio de Sustitución¿podemos hacer uso de este principio ya que nos permitirá tener interfaces

únicas. No todas las entidades necesitan implementar los mismos métodos, por ejemplo:

- Clientes y Proveedores pueden tener interfaces separadas en lugar de una interfaz única gigante con métodos innecesarios.