Desarrollo Taller Práctico

CASO: Diseñar un sistema para: "Gestión de pedidos en una cafetería universitaria digital" El sistema debe permitir:

- Ver menú
- Realizar pedido
- Pagar
- Notificar preparación
- Entregar pedido

ACTIVIDAD 1 ABSTRACCIÓN:

1. Objetivo: Diseñar un software que pueda gestionar los pedidos de forma eficiente para una cafetería universitaria, desde la de un menú, el sistema debe permitirme ver la trazabilidad, gestionar un control de pagos y mostrarme un estado del pedido.

2. Actores: se identifica a los siguientes actores:

- Cliente
- Personal de cocina
- Sistema de pago

3. Funcionalidades del sistema

- Visualizar el menú en su última versión
- Crear un pedido
- Seleccionar un producto
- Editar el pedido
- Calcular el total de productos seleccionados
- Gestión de método de pago
- Notificar a la cocina de la cafetería
- Actualizar el estado del pedido
- Notificar al cliente
- Confirmar entrega

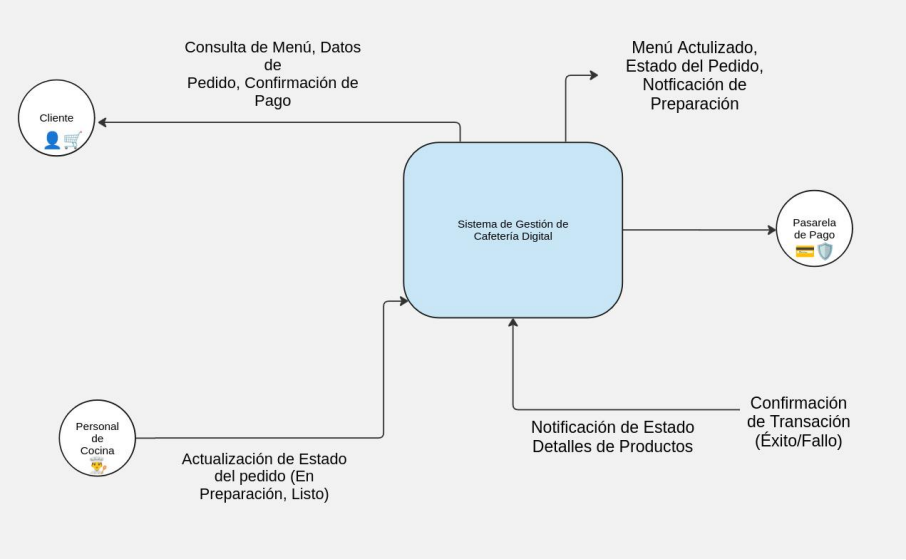
4. Limites del sistema

Dentro del sistema de gestión de pedidos en una cafetería universitaria, se puede decir que el software puede controlar:

- Visualizar un menú
- Creación y gestión de un pedido
- Cálculo del total del pedido
- Registro de un método de pago o pasarela de pagos
- Notificación al cliente
- Notificación a la cocina de la cafetería
- Registro en la base de datos

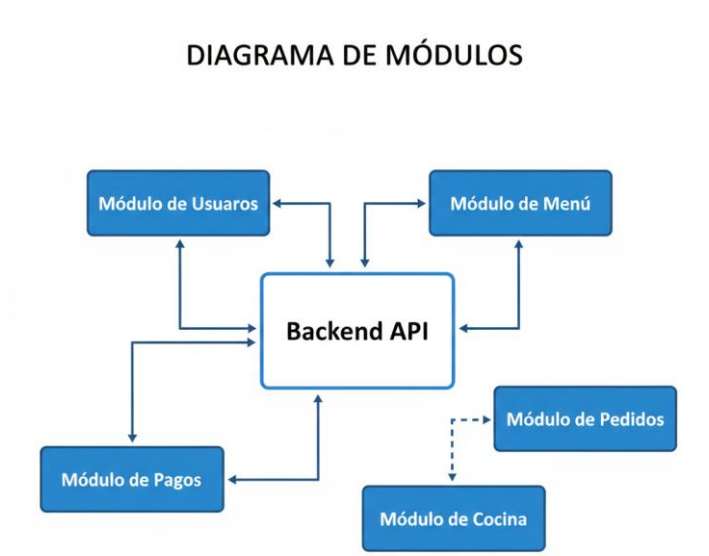
Fuera del sistema de gestión de pedidos en una cafetería universitaria, se puede decir que el software no puede controlar:

- Preparación física del producto
- Entrega del producto
- Manejo de un inventario de productos
- Gestión de los proveedores
- Contabilidad General de la cafetería
- Procesamiento interno de la pasarela de banco



ACTIVIDAD 2: MODULARIDAD

El sistema se ha dividido en módulos independientes siguiendo el principio de **alta cohesión y bajo acoplamiento**. Esta estructura permite que cada parte del software se encargue de una responsabilidad única, facilitando el mantenimiento y la escalabilidad del sistema.



ACTIVIDAD 3: REFINAMIENTO

El código completo con este trabajo se encuentra en el repositorio de GitHub y el taller en un README.MD

```java
import java.util.List;

public class Order {
    // Private attributes for encapsulation
    private final int orderID;
    private final List<Product> items;
    private final Branch branch;
    private String status;

    // Constructor
    public Order(int orderID, List<Product> items, Branch branch, String status) {
        this.orderID = orderID;
        this.items = items;
        this.branch = branch;
        this.status = status;
    }

    // Overloaded constructor: uses a default status
    public Order(int orderID, List<Product> items, Branch branch) {
        this(orderID, items, branch, "PENDING");
    }

    // Methods
    public double calculateTotalPrice() {
        double total = 0.0;
```

```
        for (Product item : items) {
            total += item.getPrice();
        }
        return total;
    }

    public void addItem(Product item) {
        this.items.add(item);
    }

    public void removeItem(Product item) {
        this.items.remove(item);
    }

    public void updateStatus(String newStatus) {
        this.status = newStatus;
    }

    public String getBranchLocation() {
        return branch.getName();
    }

    public String getStatus() {
        return this.status;
    }
}
```

ACTIVIDAD 4: APLICACION DE PRINCIPIOS

Flexibilidad: El sistema es flexible ya que aplicaría módulos independientes, esto permite modificar o extender funcionalidades del sistema a futuro sin afectar a otras partes del sistema.

Testing: Este sistema permite realizar pruebas ya que los módulos funcionan de manera independiente, el frontend se encuentra desacoplado del backend y la lógica está separada de la interfaz.

Se pueden realizar pruebas unitarias al método calcular_total() del módulo Pedidos sin necesidad de conectarse a la base de datos ni a la pasarela de pago.

Mock de pasarela de pagos: Se puede simular el pago de un producto sin necesidad de conectarse al banco y eso demostraría un buen diseño de software.

Qué pasaría si crece a varias sedes: El sistema está diseñado para escalar de manera eficiente a múltiples sedes universitarias sin necesidad de reescribir el código

base. Gracias al modularidad, donde la lógica de negocio está separada de la interfaz, podemos manejar diferentes cafeterías tratando a cada una como una instancia de la entidad **Sede (Branch)**.

En lugar de utilizar un diseño estático, la arquitectura permite que cada **Pedido (Order)** esté vinculado a una sede específica mediante **composición**. Si la universidad abre nuevas sedes, solo es necesario registrarlas en la **Capa de Datos**. El **Backend API** procesará las solicitudes de forma independiente, permitiendo que cada sede gestione su propio menú y pedidos en tiempo real, manteniendo la **flexibilidad** y facilitando el **mantenimiento** centralizado del software.

ACTIVIDAD 5: ARQUITECTURA

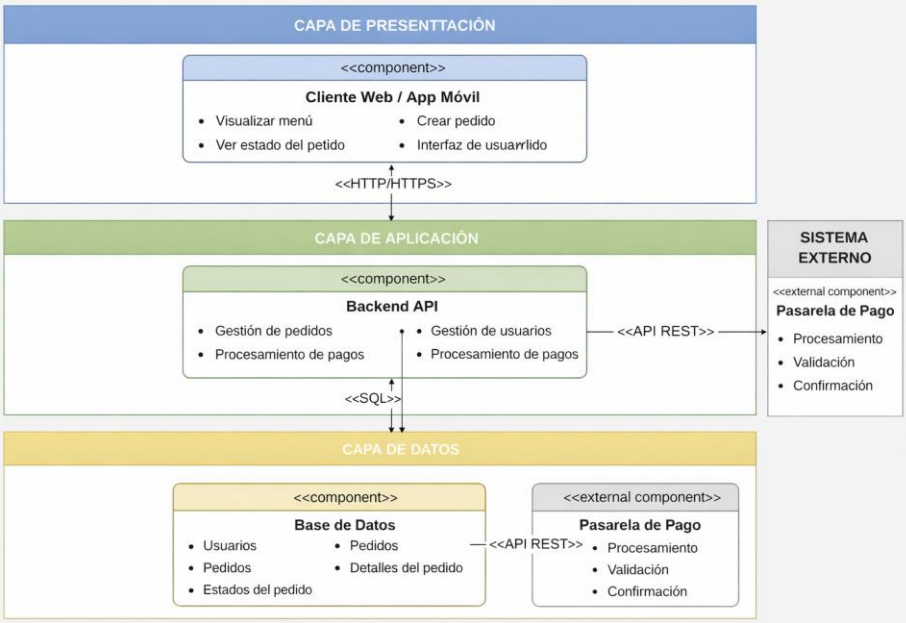
La arquitectura diseñada para la cafetería universitaria se basa en un modelo de **Niveles o Capas**, lo que garantiza el desacoplamiento y la facilidad de mantenimiento. A continuación, se describen sus componentes:

Capa de Presentación (Frontend): Es la interfaz con la que interactúan el Cliente y el Personal de Cocina mediante una App Móvil o Web. Se encarga de capturar los pedidos y mostrar el estado en tiempo real.

Capa de Aplicación (Backend API): Actúa como el cerebro del sistema. Procesa la lógica de negocio, como la gestión de usuarios y el cálculo de totales, comunicándose con el exterior mediante protocolos HTTP/HTTPS.

Capa de Datos: Es el nivel de persistencia donde se almacenan de forma organizada los menús, registros de usuarios y el historial de pedidos en una base de datos SQL.

Sistemas Externos (Pasarela de Pago): El sistema se integra con una API externa para el procesamiento seguro de pagos, validando las transacciones sin comprometer los datos bancarios dentro de nuestra infraestructura.


CAPA DE PRESENTTACIÔN
<<component>>
Cliente Web / App Móvil
Visualizar menú
Crear pedido
Ver estado del petido
Interfaz de usuarrlido
<<HTTP/HTTPS>>
CAPA DE APLICACIÓN
<<component>>
Backend API
Gestión de pedidos
Gestión de usuarios
Procesamiento de pagos
Procesamiento de pagos
<<API REST>>
SISTEMA EXTERNO
<<external component>>
Pasarela de Pago
Procesamiento
Validación
Confirmación
<<SQL>>
CAPA DE DATOS
<<component>>
Base de Datos
Usuarios
Pedidos
Pedidos
Detalles del pedido
Estados del pedido
<<API REST>>
<<external component>>
Pasarela de Pago
Procesamiento
Validación
Confirmación