UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL TUCUMÁN

DESARROLLO DE SOFTWARE

TRABAJO PRÁCTICO INTEGRADOR

2025

INTRODUCCIÓN

Se desea desarrollar una **plataforma de comercio electrónico (*E-commerce*)**. En esta primera etapa el objetivo es construir el **módulo de Órdenes**, permitiendo la gestión completa de éstas.

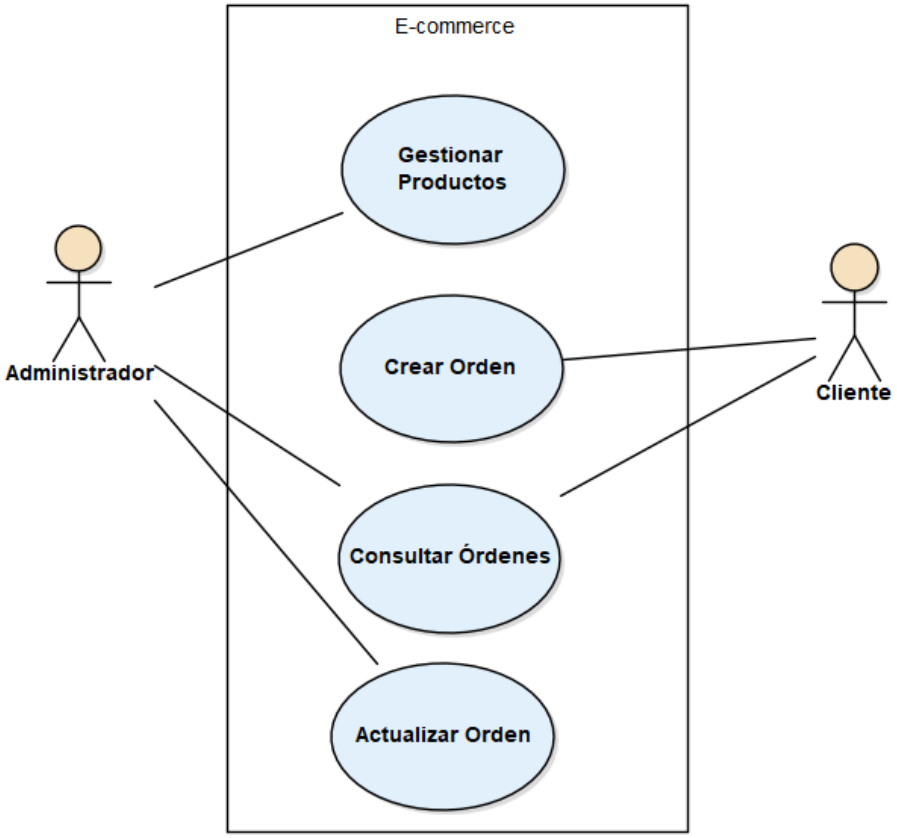
VISIÓN GENERAL DEL PRODUCTO

Del relevamiento preliminar se identificaron los siguientes requisitos:

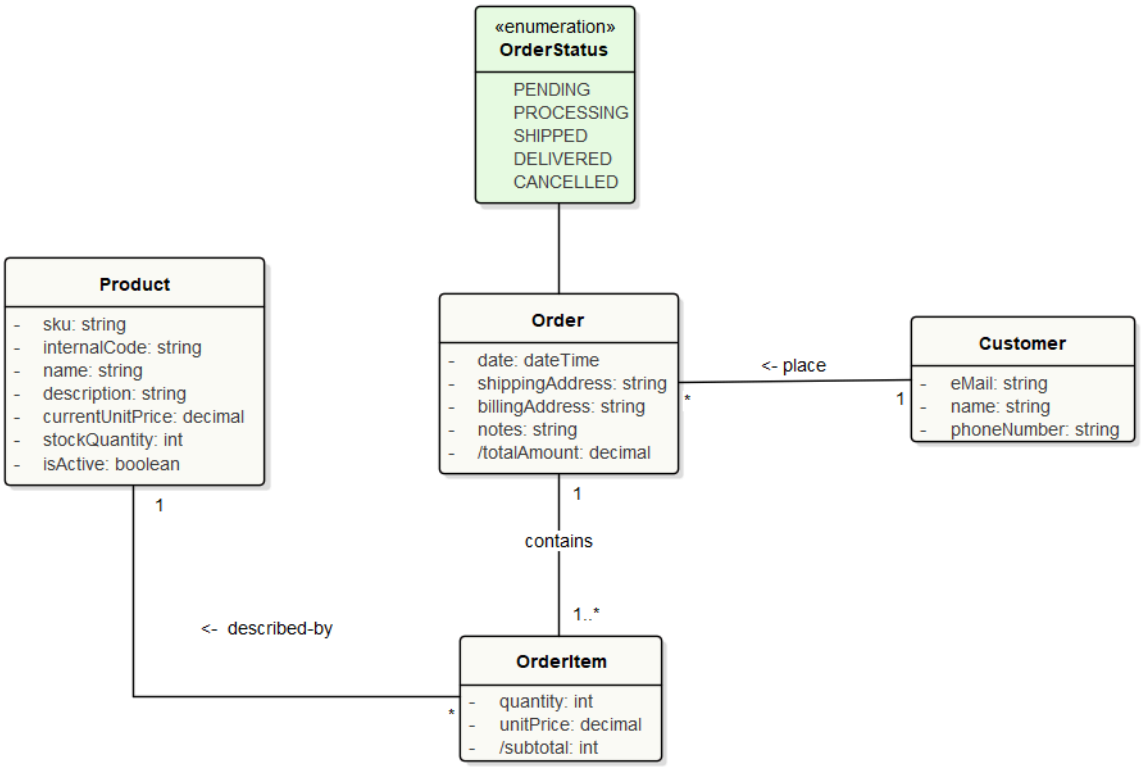
- Los visitantes pueden consultar los productos sin necesidad de estar registrados o iniciar sesión.
- Para realizar un pedido se requiere el inicio de sesión.
- Una orden, para ser aceptada, debe incluir la información básica del cliente, envío y facturación.
- Antes de registrar la orden se debe verificar la disponibilidad de stock (o existencias) de los productos.
- Si la orden es exitosa hay que actualizar el stock de cada producto.
- Se deben poder consultar órdenes individuales o listar varias con posibilidad de filtrado.
- Será necesario el cambio de estado de una orden a medida que avanza en su ciclo de vida.
- Los administradores solo pueden gestionar los productos (alta, modificación y baja) y actualizar el estado de la orden.
- Los clientes pueden crear y consultar órdenes.

ANÁLISIS

Resumen de Funcionalidades



Modelo de Dominio



DISEÑO

Arquitectura

Los factores qué se evaluaron para el diseño de la arquitectura corresponden a los requisitos básicos de seguridad y usabilidad:

Seguridad:

- Autenticación: requerir autenticación para acceder a sus recursos, utilizando estándares como OAuth 2.0, JWT o similar.
- Autorización: Los usuarios solo podrán acceder a los recursos y operaciones para los que tengan permisos explícitos.
- Validación de datos: Todos los datos recibidos deben ser validados y sanitizados antes de ser procesados o almacenados.

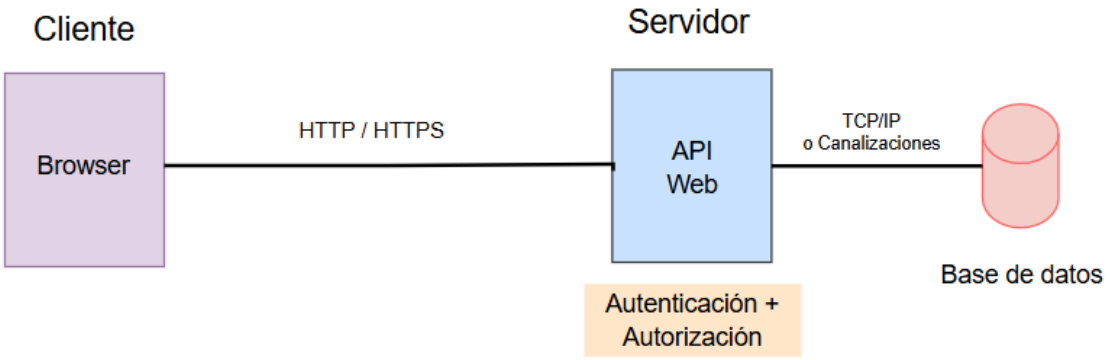
Usabilidad:

La **usabilidad** es un factor crítico que impacta directamente en la facilidad con la que el *frontend* (y otros sistemas) podrán consumirla y, por ende, en la experiencia del usuario final. Los siguientes aspectos contribuyen a la usabilidad del backend:

- **Consistencia de la API:** La API debe seguir patrones RESTful coherentes en sus rutas, métodos HTTP y formatos de solicitud/respuesta. Esto facilita que los desarrolladores del *frontend* comprendan y anticipen el comportamiento de los endpoints.
- **Claridad en las Respuestas y Errores:** Las respuestas de la API, tanto exitosas como de error, deben ser claras, predecibles y significativas. Esto incluye el uso correcto de códigos de estado HTTP y la provisión de mensajes de error descriptivos y estructurados que ayuden al *frontend* a diagnosticar problemas y mostrar feedback adecuado al usuario.
- **Rendimiento Adecuado:** Los tiempos de respuesta de la API deben ser óptimos para garantizar una experiencia de usuario fluida en el *frontend*, evitando demoras innecesarias al interactuar con el sistema (ej. al listar productos o crear una orden).
- **Robustez y Tolerancia a Fallos:** La API debe ser robusta ante entradas inválidas o situaciones inesperadas, evitando fallos catastróficos y devolviendo errores manejables. Esto permite al *frontend* gestionar adecuadamente los escenarios de error sin romperse.
- **Mantenibilidad y Extensibilidad del Código:** Un código backend bien estructurado y fácil de mantener facilita la incorporación de nuevas funcionalidades requeridas por el *frontend* en futuras etapas, lo que contribuye a la agilidad del desarrollo global.

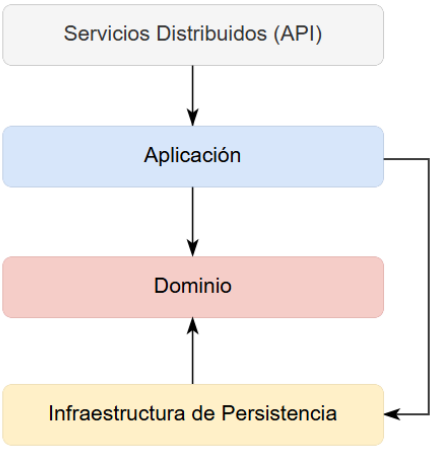
Solución

Por ello se decidió el desarrollo de una **aplicación web (cliente-servidor)** donde ***backend*** y ***frontend*** se implementarán por separado.



Backend

Solución siguiendo los principios de una **API RESTful** utilizando el patrón ***Layers* (Capas)** para la organización interna de sus elementos.



Aspectos de seguridad a tener en cuenta:

- **Protección contra ataques comunes**: implementar medidas para prevenir ataques de inyección (*SQL Injection, Command Injection*), protegerse contra ataques de *Cross-Site Scripting* (XSS) y *Cross-Site Request Forgery* (CSRF) cuando corresponda.
- **Límites y Protección contra Abuso**: implementar limitación de tasa (rate limiting) para evitar ataques de denegación de servicio (DoS) y abuso de recursos.
- **Gestión de Sesiones y Tokens**: Los tokens de acceso deben tener una expiración adecuada y ser revocables. No se deben exponer tokens o credenciales en URLs, logs o mensajes de error.
- **CORS Seguro**: Configurar CORS (*Cross-Origin Resource Sharing*) para permitir solo los orígenes necesarios y evitar el acceso no autorizado desde otros dominios.

Requisitos Técnicos:

- **Lenguaje y Framework**: C# con ASP.NET Core.
- **Base de Datos**: Para la base de datos, pueden utilizar **SQL Server**. Deberán utilizar Entity Framework Core para la interacción con la base de datos.
- **Arquitectura**: La solución debe seguir los principios de una API RESTful, utilizando los métodos HTTP apropiados para cada operación (GET, POST, PUT).
- **Validaciones**: Implementar validaciones básicas para los datos de entrada de las órdenes para asegurar la integridad de la información.
- **Manejo de Errores**: La API debe responder con códigos de estado HTTP apropiados (ej. 200 OK, 201 Created, 400 Bad Request, 404 Not Found) y mensajes de error claros y descriptivos.

Consideraciones Adicionales:

- Para CustomerId y ProductId, pueden utilizar GUID (identificadores únicos globales) o int. Para este TP, no es necesario implementar tablas de Customers reales; simplemente simulen su existencia utilizando los IDs.
- El TotalAmount de la orden y el Subtotal de cada OrderItem deben ser calculados por el *backend* al momento de crear la orden, basándose en la Quantity y UnitPrice proporcionadas para cada OrderItem.
- La UnitPrice en OrderItems representa el precio del producto en el momento exacto de la compra, lo cual es fundamental para la integridad histórica de la orden, incluso si el CurrentUnitPrice del producto cambia en la tabla Products posteriormente.
- Gestión **de Stock:** Al crear una orden, el sistema debe:
    - Verificar que la Quantity solicitada para cada OrderItem no exceda el StockQuantity disponible en la tabla Products.
    - Si hay stock suficiente, decrementar el StockQuantity de los productos correspondientes en la tabla Products por la cantidad ordenada.
    - Si no hay stock suficiente para algún producto, la operación de creación de la orden debe fallar y retornar un error 400 Bad Request con un mensaje indicando el problema de stock.

Frontend

Su diseño inicial debe contemplar las interacciones necesarias con el *backend* del módulo de Órdenes y Productos. Sus funcionalidades principales incluirán:

- Exploración de Productos:
    - Listado de Productos: Mostrar una galería de productos disponibles, visible para todos los visitantes sin necesidad de registro.

    - Detalle de Producto: Permitir a los usuarios ver información detallada de un producto específico (descripción, precio, stock, etc.).
- Gestión de Compras:
    - Carrito de Compras: Funcionalidad para agregar, eliminar y actualizar cantidades de productos en un carrito temporal antes de proceder con la compra.
    - Proceso de Creación de Orden (Checkout): Guiar al usuario a través de los pasos para finalizar una compra y la confirmación final de la orden.
- Gestión de Órdenes del Cliente:
    - Listado de Órdenes: Permitir consultar sus órdenes pasadas y actuales.
    - Detalle de Orden: Visualizar los detalles de una orden específica, incluyendo los productos comprados, cantidades, precios y estado actual.
- Funcionalidades Administrativas:
    - Gestión de Productos: Interfaz para puedan dar de alta, modificar y eliminar productos del catálogo.
    - Actualización de Estado de Órdenes: Interfaz para que los administradores puedan cambiar el estado de las órdenes (ej. de Pendiente' a 'Procesando', 'Enviado', etc.)

IMPLEMENTACIÓN

Backend

A continuación se detallan los endpoints que su API debe implementar, siguiendo las convenciones RESTful:

- **1. Crear un producto**
  - **Método HTTP:** POST
  - **Ruta:** /api/products
  - **Descripción:** Crea un nuevo producto con los datos proporcionados en el cuerpo de la solicitud.
  - **Cuerpo de la Solicitud (Ejemplo JSON):**

```
JSON
{
 "sku": "ABC123",
 "internalCode": "INT-001",
 "name": "Producto de ejemplo",
 "description": "Descripción del producto de ejemplo.",
 "currentUnitPrice": 199.99,
 "stockQuantity": 50
}
```

  - **Respuesta Exitosa:** 201 Created con el objeto del producto creado**.**
  - **Respuesta de Error:** 400 Bad Request si los datos enviados no son válidos.
- **2. Obtener todos los productos:**
  - **Método HTTP**: GET
  - **Ruta**: /api/products
  - **Descripción**: Retorna la lista completa de productos disponibles.
  - **Respuesta Exitosa**: 200 OK con una colección de objetos producto.
  - **Respuesta de Error**: 204 No Content si no hay productos registrados
- **3. Obtener un producto por ID:**
  - **Método HTTP**: GET
  - **Ruta**: /api/products/{id}
  - **Descripción**: Retorna los detalles completos de un producto específico.
  - **Respuesta Exitosa**: 200 OK con el objeto del producto solicitado.
  - **Respuesta de Error**: 404 Not Found si no se encuentra un producto con el ID proporcionado.
- **4. Actualizar un producto:**
  - **Método HTTP**: PUT
  - **Ruta**: /api/products/{id}
  - **Descripción**: Actualiza los datos de un producto existente con el ID proporcionado, usando los datos enviados en el cuerpo de la solicitud.
  - **Cuerpo de la Solicitud (Ejemplo JSON):**

```
JSON
{
 "sku": "ABC123",
 "internalCode": "INT-001",
 "name": "Producto de ejemplo",
 "description": "Descripción del producto de ejemplo.",
 "currentUnitPrice": 199.99,
 "stockQuantity": 50
}
```

- **Respuesta Exitosa**: 200 OK con el objeto del producto actualizado.
- **Respuesta de Error**:
- • 400 Bad Request si los datos enviados no son válidos.
- • 404 Not Found si no se encuentra un producto con el ID proporcionado.

- **5. Inhabilitar un producto:**
  - **Método HTTP**: PATCH
  - **Ruta**: /api/products/{id}
  - **Descripción**: Modificar el atributo IsActive para inhabilitarlo.
  - **Respuesta Exitosa**: 204 No Content si la eliminación fue exitosa.
  - **Respuesta de Error**: 404 Not Found si no se encuentra un producto con el ID proporcionado.
- **6. Crear una nueva orden:**
  - **Método HTTP:** POST
  - **Ruta:** /api/orders
  - **Descripción:** Permite registrar una nueva orden de compra en el sistema. La orden debe incluir un identificador de cliente (simulado), direcciones de envío y facturación, y una lista de ítems que componen la orden (productos con sus cantidades y precios unitarios al momento de la compra). Antes de crear la orden, se debe verificar que haya suficiente stock disponible para cada producto solicitado. Si el stock es insuficiente para algún producto, la orden no debe crearse y se debe retornar un error. Si la orden se crea exitosamente, el stock de los productos involucrados debe ser decrementado. No se debe implementar ninguna lógica de pago.
  - **Cuerpo de la Solicitud (Ejemplo JSON):**

```
JSON
{
 "customerId": "a1b2c3d4-e5f6-7890-1234-567890abcdef", // simulado para el cliente
 "shippingAddress": "Calle Falsa 123, Ciudad, País",
 "billingAddress": "Calle Falsa 123, Ciudad, País",
 "orderItems": [
    {
      "productId": 1234567890,
      "quantity": 1,
```

```
      "name": "Producto 1",
      "description": "Descripción del producto 1",
      "currentUnitPrice": 100.00
    }, …
  ]
}
```

- **Respuesta Exitosa:** 201 Created con los detalles completos de la orden creada, incluyendo su OrderId generado.
- **Respuesta de Error:** 400 Bad Request si los datos de la solicitud son inválidos o incompletos, o si no hay stock suficiente para uno o más productos.

- **7. Obtener todas las órdenes:**
  - **Método HTTP:** GET
  - **Ruta:** /api/orders
  - **Descripción:** Retorna una lista paginada de todas las órdenes registradas en el sistema.
  - **Parámetros de Consulta (Opcional):**
    - status: Permite filtrar las órdenes por su estado (ej. Pending, Processing, Shipped, Delivered, Cancelled).
    - customerId: Permite filtrar las órdenes por un ID de cliente específico.
    - pageNumber: Número de página para paginación (ej. 1).
    - pageSize: Cantidad de elementos por página (ej. 10).
  - **Respuesta Exitosa:** 200 OK con un array de objetos de órdenes.
  - **Respuesta de Error:** 500 Internal Server Error en caso de un fallo inesperado del servidor.
- **8. Obtener una orden por ID:**
  - **Método HTTP:** GET
  - **Ruta:** /api/orders/{id}
  - **Descripción:** Retorna los detalles completos de una orden específica, incluyendo todos sus ítems.
  - **Respuesta Exitosa:** 200 OK con el objeto de la orden solicitada.
  - **Respuesta de Error:** 404 Not Found si no se encuentra una orden con el ID proporcionado.
- **9. Actualizar el estado de una orden:**
  - **Método HTTP:** PUT
  - **Ruta:** /api/orders/{id}/status
  - **Descripción:** Permite cambiar el estado de una orden existente. Este endpoint debe ser idempotente y solo modificar el estado de la orden.
  - **Cuerpo de la Solicitud (Ejemplo JSON):**

```
JSON
{
  "newStatus": "Processing" // Nuevo estado de la orden
}
```

- **Respuesta Exitosa:** 200 OK con los detalles actualizados de la orden.
- **Respuesta de Error:** 400 Bad Request si el nuevo estado es inválido o la transición de estado no es permitida, 404 Not Found si la orden no existe.

ENTREGABLES

Primera parte: **Backend**

- Código fuente completo del proyecto ASP.NET Core API, subido a un repositorio Git (GitHub).
- Un archivo README.md en la raíz del repositorio que contenga:
  - Datos de los integrantes del grupo
  - Instrucciones claras y concisas para configurar y ejecutar el proyecto localmente.
  - Una breve descripción de los endpoints implementados y cómo utilizarlos.

CRITERIOS DE EVALUACIÓN

- **Funcionalidad (40%):** Correcta implementación de todos los endpoints y sus funcionalidades según lo especificado, incluyendo la verificación y actualización de stock.
- **Calidad del Código (30%):** Código limpio, legible, bien estructurado, modularizado y que siga las buenas prácticas de C# y ASP.NET Core.
- **Manejo de Errores y Validaciones (15%):** Implementación adecuada de validaciones de entrada y respuestas de error significativas con códigos de estado HTTP correctos, incluyendo mensajes específicos para problemas de stock.
- **Uso de Entity Framework Core (15%):** Correcta configuración y uso de EF Core para la persistencia de datos, incluyendo el diseño de los modelos y las migraciones.