PlaceOrder Documentation (English & Arabic)

1. Old Code Walkthrough | شرح الكود القديم

English

The old implementation of PlaceOrder works as follows:
1. Iterates over all items.
2. Fetches each product by name using _productService.GetProductByName.
3. Validates existence and stock.
4. Creates a new Order.
5. Iterates again, fetches product again, calculates price, deducts stock, creates OrderProducts, updates DB.
6. Updates total order amount.

بالعربي

الكود القديم في دالة PlaceOrder يشتغل كالتالي:
1. يلف على كل العناصر.
2. يجيب المنتج بالاسم باستخدام _productService.GetProductByName.
3. يتأكد أن المنتج موجود والمخزون يكفي.
4. ينشئ طلب جديد Order.
5. يرجع يلف مرة ثانية، يحسب السعر، يخصم المخزون، ينشئ OrderProducts، يحدث في قاعدة البيانات.
6. يحدّث الإجمالي للطلب.

Old Code

```
// Places an order for the given list of items and user ID
public void PlaceOrder(List<OrderItemDTO> items, int uid)
{
   Product existingProduct = null;
   decimal TotalPrice, totalOrderPrice = 0;
   OrderProducts orderProducts = null;

   // Validation loop: check if products exist and stock is enough
   // حلقة التحقق: تتأكد إذا المنتجات موجودة والمخزون يكفي
   for (int i = 0; i < items.Count; i++)
   {
      TotalPrice = 0;
      existingProduct = _productService.GetProductByName(items[i].ProductName);
      if (existingProduct == null)
```

```
        throw new Exception($"{items[i].ProductName} not Found"); // Product not found
      if (existingProduct.Stock < items[i].Quantity)
        throw new Exception($"{items[i].ProductName} is out of stock"); // Stock too low
    }

    // Create new order for user
    // إنشاء الطلب للمستخدم
    var order = new Order { UID = uid, OrderDate = DateTime.Now, TotalAmount = 0 };
    AddOrder(order);

    // Process each item again (duplicate product lookups)
    // معالجة العناصر مرة ثانية )يجيب المنتجات مرة ثانية بالاسم( //
    foreach (var item in items)
    {
      existingProduct = _productService.GetProductByName(item.ProductName); // fetch
again

      // Calculate item total
      // حساب السعر لهذا المنتج //
      TotalPrice = item.Quantity * existingProduct.Price;

      // Deduct stock
      // خصم الكمية من المخزون //
      existingProduct.Stock -= item.Quantity;

      // Add to total
      // تحديث الإجمالي //
      totalOrderPrice += TotalPrice;

      // Create relation record (OrderProducts)
      // إنشاء العلاقة بين الطلب والمنتج //
      orderProducts = new OrderProducts { OID = order.OID, PID = existingProduct.PID,
Quantity = item.Quantity };
      _orderProductsService.AddOrderProducts(orderProducts);

      // Update product in DB
      // تحديث المنتج في قاعدة البيانات //
      _productService.UpdateProduct(existingProduct);
    }

    // Finalize order with total amount
    // تحديث الطلب بالإجمالي //
    order.TotalAmount = totalOrderPrice;
```

```
    UpdateOrder(order);
}
```

2. Problems / Bottlenecks | العيوب

English:
- Duplicate DB lookups.
- Multiple DB calls in loop.
- No transaction → data inconsistency risk.
- ProductName lookup instead of ID.
- Generic exceptions.

بالعربي:
- تكرار الاستعلامات.
- تحديثات كثيرة داخل اللوب.
- ما في Transaction.
- الاعتماد على الاسم بدل الـ ID.
- استثناءات عامة.

3. Full Analysis | التحليل الكامل

The old code is logically correct but inefficient. It risks performance degradation and inconsistent database state under failures.

الكود القديم منطقيًا صحيح لكنه بطيء وغير آمن من ناحية البيانات.

4. Improved Implementation | الكود المحسّن

English & Arabic: The improved version reduces DB calls, uses IDs, applies transactions, and bulk updates.

Improved Code

```
// Improved PlaceOrder with Transaction, IDs, Bulk Updates

// نسخة محسّنة مع معاملة (Transaction)استخدام ، IDsوتحديث دفعة واحدة ،

public void PlaceOrder(List<OrderItemDTO> items, int uid)

{

   // Defensive check

   // التحقق المبدئي
```

```
  if (items == null || items.Count == 0)
    throw new ArgumentException("Order must contain at least one item.");

  using var transaction = _dbContext.Database.BeginTransaction();
  try
  {
    // Step 1: Fetch all products in one query (by IDs, not names)
    // الخطوة ١: جلب كل المنتجات باستعلام واحد بالـ ID
    var productIds = items.Select(i => i.ProductId).ToList();
    var products = _productService.GetProductsByIds(productIds)
                    .ToDictionary(p => p.PID, p => p);

    // Step 2: Validate products and stock
    // التحقق من وجود المنتجات والمخزون
    foreach (var item in items)
    {
      if (!products.TryGetValue(item.ProductId, out var product))
        throw new ProductNotFoundException($"Product ID {item.ProductId} not found.");
// استثناء مخصص

      if (product.Stock < item.Quantity)
        throw new OutOfStockException($"{product.Name} is out of stock. Requested
{item.Quantity}, Available {product.Stock}");
    }

    // Step 3: Create new order
    // إنشاء الطلب الجديد
```

```
var order = new Order
{
   UID = uid,
   OrderDate = DateTime.Now,
   TotalAmount = 0
};
AddOrder(order); // Save to DB (so it gets OID)

decimal totalOrderPrice = 0;
var orderProductsList = new List<OrderProducts>();

// Step 4: Process all items
// معالجة كل العناصر
foreach (var item in items)
{
   var product = products[item.ProductId];

   // Calculate total price for item
   // حساب السعر للمنتج
   decimal totalPrice = item.Quantity * product.Price;
   totalOrderPrice += totalPrice;

   // Deduct stock
   // خصم الكمية
   product.Stock -= item.Quantity;
```

```
        // Create OrderProducts record
        // إنشاء سجل العلاقة OrderProducts
        orderProductsList.Add(new OrderProducts
        {
            OID = order.OID,
            PID = product.PID,
            Quantity = item.Quantity
        });
    }

    // Step 5: Bulk update DB
    // تحديث دفعة واحدة
    _dbContext.Products.UpdateRange(products.Values);        // Update all products in one go
    _dbContext.OrderProducts.AddRange(orderProductsList);    // Insert all OrderProducts in one go

    // Update total order amount
    // تحديث الإجمالي
    order.TotalAmount = totalOrderPrice;
    UpdateOrder(order);

    // Save and commit transaction
    // الحفظ مع Commit
    _dbContext.SaveChanges();
    transaction.Commit();
  }
```

```
    catch
    {
        // Rollback if error
        // إلغاء لو صار خطأ
        transaction.Rollback();
        throw;
    }
}
```

5. Comparison | المقارنة بين الحلول

Aspect	Old Code (الكود القديم)	Improved Code (الكود المحسّن)
DB Queries	2× per product + per update	1 query for all + bulk update
Transaction	None	Full transaction
Scalability	Poor	Efficient
Maintainability	Harder (generic exceptions)	Cleaner (custom exceptions)
Correctness	Relies on ProductName	Uses ProductId

6. Recommendation | التوصية

English: Use the improved implementation for real e-commerce systems. It ensures performance, safety, and scalability.

بالعربي: الأفضل اعتماد الكود المحسّن في أنظمة التجارة الإلكترونية الفعلية لأنه أسرع، آمن أكثر، وقابل للتوسع.