خطة بناء نظام إدارة المخزون الشامل

ASP.NET Core MVC و Areas للأمان مع موديول الهوية المنفصل

مقدمة المشروع

هذا المستند يحتوي على خطة تفصيلية شاملة لبناء نظام إدارة مخزون احترافي باستخدام ASP.NET Core MVC، مع فصل Areas للأمان والتحكم وتطبيق نظام (Identity) قاعدة بيانات الهوية في الصلاحيات.

الهدف من هذا المشروع هو تطبيق أفضل الممارسات في التطوير وتوظيف جميع المبادئ والتقنيات التي تعلمتها في مجال .NET و ASP.NET Core بشكل عملي وحقيقي.

الجزء الأول: التحليل والتخطيط الأساسي

1. تحديد أهداف المشروع والميزات الأساسية

المشروع سيركز على الوظائف التالية:

- **إدارة المنتجات والتصنيفات:** إضافة وتعديل وحذف المنتجات مع تصنيفاتها
- **تتبع المخزون:** مراقبة الكميات المتاحة لكل منتج بشكل فوري
- **إدارة الموردين:** تسجيل وإدارة الموردين والتعامل معهم
- **إدارة العملاء:** حفظ بيانات العملاء والتعامل معهم
- **أوامر الشراء والبيع:** تسجيل أوامر الشراء من الموردين وأوامر البيع للعملاء
- **التنبيهات الذكية:** تنبيهات تلقائية عند انخفاض المخزون عن حد معين
- **التقارير والإحصائيات:** تقارير شاملة عن المبيعات والمشتريات والمخزون
- **نظام الصلاحيات:** توزيع الصلاحيات على المستخدمين (Admin، Manager، Employee)

2. رسم الـ ERD (Entity Relationship Diagram)

الكيانات الرئيسية للمشروع:

الكيان	الوصف
Products	المنتجات الموجودة في المخزن
Categories	تصنيفات المنتجات
Suppliers	الموردين الذين نشتري منهم
Customers	العملاء الذين نبيع لهم
PurchaseOrders	أوامر الشراء من الموردين
PurchaseOrderDetails	تفاصيل أوامر الشراء
SalesOrders	أوامر البيع للعملاء
SalesOrderDetails	تفاصيل أوامر البيع
StockMovements	حركات المخزون (البيع والشراء)
Employees	الموظفون والمستخدمين
Departments	الأقسام في الشركة

العلاقات الأساسية:

- Product ← Categories (Many-to-One)
- Product ← Suppliers (Many-to-Many عبر SupplierProducts)
- PurchaseOrder ← Suppliers (Many-to-One)
- PurchaseOrderDetails ← PurchaseOrder (One-to-Many)
- PurchaseOrderDetails ← Products (Many-to-One)
- SalesOrder ← Customers (Many-to-One)
- SalesOrderDetails ← SalesOrder (One-to-Many)
- SalesOrderDetails ← Products (Many-to-One)
- StockMovements ← Products (Many-to-One)
- Employees ← Departments (Many-to-One)
- Employees ← AspNetUsers (One-to-One من خلال AspNetUserId)

3. تخطيط موديول الأمن والهوية (Identity)

سيتم فصل نظام الهوية في قاعدة بيانات منفصلة (IdentityDB) عن قاعدة البيانات الأساسية (InventoryDB)، مع ربطهما من خلال:

- IdentityDB من AspNetUsers يشير إلى جدول Employees مفتاح خارجي في جدول
- استخدام Dependency Injection للوصول إلى UserManager و RoleManager
- استخدام Claims و Roles للتحكم في الصلاحيات

الجزء الثاني: إنشاء الحل البرمجي (Solution Structure)

1. إنشاء الـ Solution والمشاريع

في Visual Studio، أنشئ Solution جديد باسم **InventoryManagementSystem** وأضف المشاريع التالية:

المشاريع الأساسية:

1. **Inventory.Web** (ASP.NET Core MVC)
   - المشروع الرئيسي الذي يحتوي على Areas والـ Views والـ Controllers
2. **Inventory.Core** (Class Library)
   - يحتوي على ViewModels والـ Interfaces والـ Entities والـ Models
3. **Inventory.Data** (Class Library)
   - يحتوي على Repositories والـ Migrations والـ DbContext
4. **Inventory.Identity** (Class Library)
   - مشروع منفصل لإدارة الهوية والمصادقة
5. **Inventory.Services** (Class Library)
   - طبقة الخدمات والمنطق التجاري
6. **Inventory.Tests** (Unit Test Project)
   - اختياري - لكتابة الاختبارات

2. بنية المجلدات والملفات

```
InventoryManagementSystem/
│
├── Inventory.Web/
│   ├── Areas/
│   │   ├── Admin/
│   │   │   ├── Controllers/
│   │   │   ├── Views/
│   │   │   └── Models/
│   │   ├── Manager/
│   │   │   ├── Controllers/
│   │   │   ├── Views/
│   │   │   └── Models/
│   │   └── Employee/
│   │   ├── Controllers/
│   │   ├── Views/
│   │   └── Models/
│   ├── Controllers/
│   ├── Views/
│   ├── wwwroot/
│   │   ├── css/
│   │   ├── js/
│   │   └── images/
│   ├── appsettings.json
│   └── Program.cs
│
```

```
├── Inventory.Core/
│   ├── Entities/
│   │   ├── Product.cs
│   │   ├── Category.cs
│   │   ├── Supplier.cs
│   │   └── ... (كل الكيانات)
│   ├── Interfaces/
│   │   ├── IRepository.cs
│   │   ├── IUnitOfWork.cs
│   │   └── IService.cs
│   └── ViewModels/
│
├── Inventory.Data/
│   ├── DbContext/
│   │   ├── InventoryDbContext.cs
│   ├── Repositories/
│   │   ├── Repository.cs (Generic)
│   │   ├── ProductRepository.cs
│   │   └── ... (Repositories مخصصة)
│   ├── Migrations/
│   └── UnitOfWork/
│   └── UnitOfWork.cs
│
├── Inventory.Identity/
│   ├── Models/
│   │   └── ApplicationUser.cs
│   ├── IdentityDbContext.cs
│   └── Services/
│
├── Inventory.Services/
│   ├── Services/
│   │   ├── ProductService.cs
│   │   ├── OrderService.cs
│   │   └── ... (Services)
│   └── Implementations/
│
└── Inventory.Tests/
├── RepositoryTests/
├── ServiceTests/
└── ControllerTests/
```

الجزء الثالث: إعداد قاعدتي البيانات

1. إنشاء InventoryDbContext

```
public class InventoryDbContext : DbContext
{
public InventoryDbContext(DbContextOptions<InventoryDbContext> options)
: base(options)
{
}
```

```
  public DbSet<Product> Products { get; set; }
  public DbSet<Category> Categories { get; set; }
  public DbSet<Supplier> Suppliers { get; set; }
  public DbSet<Customer> Customers { get; set; }
  public DbSet<PurchaseOrder> PurchaseOrders { get; set; }
  public DbSet<PurchaseOrderDetail> PurchaseOrderDetails { get; set; }
  public DbSet<SalesOrder> SalesOrders { get; set; }
  public DbSet<SalesOrderDetail> SalesOrderDetails { get; set; }
  public DbSet<StockMovement> StockMovements { get; set; }
  public DbSet<Employee> Employees { get; set; }
  public DbSet<Department> Departments { get; set; }

  protected override void OnModelCreating(ModelBuilder modelBuilder)
  {
     base.OnModelCreating(modelBuilder);

     // تحديد العلاقات والقيود
     modelBuilder.Entity<Product>()
        .HasOne(p => p.Category)
        .WithMany(c => c.Products)
        .HasForeignKey(p => p.CategoryId);

     // يمكن إضافة Seed Data هنا إذا لزم الأمر
  }
```

```
}
```

2. إنشاء IdentityDbContext

public class IdentityDbContext : IdentityDbContext<ApplicationUser>
{
public IdentityDbContext(DbContextOptions<IdentityDbContext> options)
: base(options)
{
}

```
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    // إذا لزم الأمر أضف تخصيصات لـ Identity
}
```

}

3. تكوين الـ Connection Strings في appsettings.json

{
"ConnectionStrings": {
"InventoryConnection": "Server=.;Database=InventoryManagementDB;Integrated Security=true;",
"IdentityConnection": "Server=.;Database=InventoryIdentityDB;Integrated Security=true;"
},
"Logging": {
"LogLevel": {
"Default": "Information"
}
}
}

4. تسجيل الـ DbContexts في Program.cs

builder.Services.AddDbContext<InventoryDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("InventoryConnection")))
;

builder.Services.AddDbContext<IdentityDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("IdentityConnection")));

5. إنشاء Migrations

في Package Manager Console:

Add-Migration InitialInventoryDB -Context InventoryDbContext -Project Inventory.Data
Add-Migration InitialIdentityDB -Context IdentityDbContext -Project Inventory.Identity

Update-Database -Context InventoryDbContext
Update-Database -Context IdentityDbContext

الجزء الرابع: بناء الـ Entities والـ Models

1. كيان Product

```
public class Product
{
public int Id { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public decimal Price { get; set; }
public int QuantityInStock { get; set; }
public int MinimumStockLevel { get; set; }
public int CategoryId { get; set; }
public Category Category { get; set; }
public DateTime CreatedAt { get; set; } = DateTime.Now;
public DateTime ModifiedAt { get; set; } = DateTime.Now;
```

```
public ICollection<PurchaseOrderDetail> PurchaseOrderDetails { get; set; }
public ICollection<SalesOrderDetail> SalesOrderDetails { get; set; }
public ICollection<StockMovement> StockMovements { get; set; }
```

```
}
```

2. كيان Category

```
public class Category
{
public int Id { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public DateTime CreatedAt { get; set; } = DateTime.Now;
```

```
public ICollection<Product> Products { get; set; }
```

```
}
```

3. كيان Employee مع ربط الهوية

```
public class Employee
{
public int Id { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
```

```
public string Email { get; set; }
public string Phone { get; set; }
public string Position { get; set; }
public int DepartmentId { get; set; }
public Department Department { get; set; }

    // الربط مع الهوية
    public string AspNetUserId { get; set; }
    public DateTime HireDate { get; set; } = DateTime.Now;

    public ICollection<SalesOrder> SalesOrders { get; set; }

}
```

4. كيانات الطلبات (SalesOrder و PurchaseOrder)

```
public class SalesOrder
{
public int Id { get; set; }
public int CustomerId { get; set; }
public Customer Customer { get; set; }
public int EmployeeId { get; set; }
public Employee Employee { get; set; }
public DateTime OrderDate { get; set; } = DateTime.Now;
public decimal TotalAmount { get; set; }
public string Status { get; set; } = "Pending";

    public ICollection<SalesOrderDetail> SalesOrderDetails { get; set; }

}

public class SalesOrderDetail
{
public int Id { get; set; }
public int SalesOrderId { get; set; }
public SalesOrder SalesOrder { get; set; }
public int ProductId { get; set; }
public Product Product { get; set; }
public int Quantity { get; set; }
public decimal UnitPrice { get; set; }
public decimal Subtotal => Quantity * UnitPrice;
}

public class PurchaseOrder
{
public int Id { get; set; }
public int SupplierId { get; set; }
```

```
public Supplier Supplier { get; set; }
public DateTime OrderDate { get; set; } = DateTime.Now;
public decimal TotalAmount { get; set; }
public string Status { get; set; } = "Pending";

    public ICollection<PurchaseOrderDetail> PurchaseOrderDetails { get; set; }

}

public class PurchaseOrderDetail
{
public int Id { get; set; }
public int PurchaseOrderId { get; set; }
public PurchaseOrder PurchaseOrder { get; set; }
public int ProductId { get; set; }
public Product Product { get; set; }
public int Quantity { get; set; }
public decimal UnitPrice { get; set; }
public decimal Subtotal => Quantity * UnitPrice;
}
```

الجزء الخامس: بناء نمط Repository و Unit of Work

1. الـ Generic Repository Interface

```
public interface IRepository<T> where T : class
{
Task<T> GetByIdAsync(int id);
Task<IEnumerable<T>> GetAllAsync();
Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>> predicate);
Task AddAsync(T entity);
Task AddRangeAsync(IEnumerable<T> entities);
void Update(T entity);
void UpdateRange(IEnumerable<T> entities);
void Delete(T entity);
void DeleteRange(IEnumerable<T> entities);
Task<bool> AnyAsync(Expression<Func<T, bool>> predicate);
Task<int> CountAsync();
}
```

2. الـ Generic Repository Implementation

```
public class Repository<T> : IRepository<T> where T : class
{
protected readonly InventoryDbContext _context;
```

```
public Repository(InventoryDbContext context)
{
    _context = context;
}

public async Task<T> GetByIdAsync(int id)
{
    return await _context.Set<T>().FindAsync(id);
}

public async Task<IEnumerable<T>> GetAllAsync()
{
    return await _context.Set<T>().ToListAsync();
}

public async Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>> pr
{
    return await _context.Set<T>().Where(predicate).ToListAsync();
}

public async Task AddAsync(T entity)
{
    await _context.Set<T>().AddAsync(entity);
}

public async Task AddRangeAsync(IEnumerable<T> entities)
{
    await _context.Set<T>().AddRangeAsync(entities);
}

public void Update(T entity)
{
    _context.Set<T>().Update(entity);
}

public void UpdateRange(IEnumerable<T> entities)
{
```

```
    _context.Set<T>().UpdateRange(entities);
  }

  public void Delete(T entity)
  {
    _context.Set<T>().Remove(entity);
  }

  public void DeleteRange(IEnumerable<T> entities)
  {
    _context.Set<T>().RemoveRange(entities);
  }

  public async Task<bool> AnyAsync(Expression<Func<T, bool>> predicate)
  {
    return await _context.Set<T>().AnyAsync(predicate);
  }

  public async Task<int> CountAsync()
  {
    return await _context.Set<T>().CountAsync();
  }
```

}

3. Product Repository Specific

```
public interface IProductRepository : IRepository<Product>
{
Task<IEnumerable<Product>> GetProductsByCategory(int categoryId);
Task<IEnumerable<Product>> GetLowStockProducts();
Task<Product> GetProductWithDetails(int id);
}

public class ProductRepository : Repository<Product>, IProductRepository
{
public ProductRepository(InventoryDbContext context) : base(context)
{
}
```

```
  public async Task<IEnumerable<Product>> GetProductsByCategory(int category
  {
```

```
    return await _context.Products
        .Where(p => p.CategoryId == categoryId)
        .ToListAsync();
}

public async Task<IEnumerable<Product>> GetLowStockProducts()
{
    return await _context.Products
        .Where(p => p.QuantityInStock <= p.MinimumStockLevel)
        .ToListAsync();
}

public async Task<Product> GetProductWithDetails(int id)
{
    return await _context.Products
        .Include(p => p.Category)
        .FirstOrDefaultAsync(p => p.Id == id);
}
```

}

4. Unit of Work

```
public interface IUnitOfWork : IDisposable
{
IProductRepository Products { get; }
IRepository<Category> Categories { get; }
IRepository<Supplier> Suppliers { get; }
IRepository<Customer> Customers { get; }
IRepository<SalesOrder> SalesOrders { get; }
IRepository<PurchaseOrder> PurchaseOrders { get; }
IRepository<Employee> Employees { get; }
Task SaveChangesAsync();
}

public class UnitOfWork : IUnitOfWork
{
private readonly InventoryDbContext _context;
private IProductRepository _productRepository;
private IRepository<Category> _categoryRepository;
private IRepository<Supplier> _supplierRepository;
private IRepository<Customer> _customerRepository;
private IRepository<SalesOrder> _salesOrderRepository;
private IRepository<PurchaseOrder> _purchaseOrderRepository;
private IRepository<Employee> _employeeRepository;
```

```
    public UnitOfWork(InventoryDbContext context)
    {
        _context = context;
    }

    public IProductRepository Products =>
        _productRepository ??= new ProductRepository(_context);

    public IRepository<Category> Categories =>
        _categoryRepository ??= new Repository<Category>(_context);

    public IRepository<Supplier> Suppliers =>
        _supplierRepository ??= new Repository<Supplier>(_context);

    public IRepository<Customer> Customers =>
        _customerRepository ??= new Repository<Customer>(_context);

    public IRepository<SalesOrder> SalesOrders =>
        _salesOrderRepository ??= new Repository<SalesOrder>(_context);

    public IRepository<PurchaseOrder> PurchaseOrders =>
        _purchaseOrderRepository ??= new Repository<PurchaseOrder>(_context);

    public IRepository<Employee> Employees =>
        _employeeRepository ??= new Repository<Employee>(_context);

    public async Task SaveChangesAsync()
    {
        await _context.SaveChangesAsync();
    }

    public void Dispose()
    {
        _context?.Dispose();
    }
}
```

الجزء السادس: تكوين الـ Areas والأمان

1. إنشاء Areas

في Inventory.Web، أنشئ مجلد Areas وأضف ثلاث أقسام:

- **Admin**: (إدارة كاملة للنظام) للإداريين
- **Manager**: (إدارة الطلبات والمخزون) لمديري المخزون
- **Employee**: (عمليات محدودة) للموظفين

2. تسجيل Areas في Program.cs

```
builder.Services.AddControllersWithViews();

var app = builder.Build();

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
name: "areas",
pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");

app.MapControllerRoute(
name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

3. مثال Controller في Admin Area

```
[Area("Admin")]
[Authorize(Roles = "Admin")]
public class ProductsController : Controller
{
private readonly IUnitOfWork _unitOfWork;

public ProductsController(IUnitOfWork unitOfWork)
{
    _unitOfWork = unitOfWork;
}

public async Task<IActionResult> Index()
{
    var products = await _unitOfWork.Products.GetAllAsync();
    return View(products);
```

```
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(Product product)
    {
        if (ModelState.IsValid)
        {
            await _unitOfWork.Products.AddAsync(product);
            await _unitOfWork.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(product);
    }
```

}

الجزء السابع: طبقة الخدمات (Services)

1. الـ Service Interface

```
public interface IProductService
{
Task<IEnumerable<ProductDto>> GetAllProductsAsync();
Task<ProductDto> GetProductByIdAsync(int id);
Task<int> CreateProductAsync(CreateProductDto dto);
Task UpdateProductAsync(int id, UpdateProductDto dto);
Task DeleteProductAsync(int id);
Task<IEnumerable<ProductDto>> GetLowStockProductsAsync();
}
```

2. الـ Service Implementation

```
public class ProductService : IProductService
{
private readonly IUnitOfWork _unitOfWork;
private readonly IMapper _mapper;
```

```
public ProductService(IUnitOfWork unitOfWork, IMapper mapper)
{
    _unitOfWork = unitOfWork;
    _mapper = mapper;
}

public async Task<IEnumerable<ProductDto>> GetAllProductsAsync()
{
    var products = await _unitOfWork.Products.GetAllAsync();
    return _mapper.Map<IEnumerable<ProductDto>>(products);
}

public async Task<ProductDto> GetProductByIdAsync(int id)
{
    var product = await _unitOfWork.Products.GetByIdAsync(id);
    return _mapper.Map<ProductDto>(product);
}

public async Task<int> CreateProductAsync(CreateProductDto dto)
{
    var product = _mapper.Map<Product>(dto);
    await _unitOfWork.Products.AddAsync(product);
    await _unitOfWork.SaveChangesAsync();
    return product.Id;
}

public async Task UpdateProductAsync(int id, UpdateProductDto dto)
{
    var product = await _unitOfWork.Products.GetByIdAsync(id);
    if (product == null)
        throw new KeyNotFoundException($"Product with ID {id} not found");

    _mapper.Map(dto, product);
    _unitOfWork.Products.Update(product);
    await _unitOfWork.SaveChangesAsync();
}
```

```
    public async Task DeleteProductAsync(int id)
    {
        var product = await _unitOfWork.Products.GetByIdAsync(id);
        if (product == null)
            throw new KeyNotFoundException($"Product with ID {id} not found");

        _unitOfWork.Products.Delete(product);
        await _unitOfWork.SaveChangesAsync();
    }

    public async Task<IEnumerable<ProductDto>> GetLowStockProductsAsync()
    {
        var products = await _unitOfWork.Products.GetLowStockProducts();
        return _mapper.Map<IEnumerable<ProductDto>>(products);
    }
```

}

الجزء الثامن: إعداد الهوية والمصادقة

1. تسجيل الهوية في Program.cs

```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>()
.AddEntityFrameworkStores<IdentityDbContext>()
.AddDefaultTokenProviders();

builder.Services.ConfigureApplicationCookie(options =>
{
options.LoginPath = "/Identity/Account/Login";
options.LogoutPath = "/Identity/Account/Logout";
options.AccessDeniedPath = "/Identity/Account/AccessDenied";
});
```

2. إنشاء الأدوار الأساسية (Seed Roles)

```
public class RoleInitializer
{
public static async Task InitializeRoles(IServiceProvider serviceProvider)
{
var roleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

    string[] roles = { "Admin", "Manager", "Employee" };
```

```
    foreach (var role in roles)
    {
        if (!await roleManager.RoleExistsAsync(role))
        {
            await roleManager.CreateAsync(new IdentityRole(role));
        }
    }
}
```

}

3. استدعاء التهيئة في Program.cs

```
using (var scope = app.Services.CreateScope())
{
var services = scope.ServiceProvider;
await RoleInitializer.InitializeRoles(services);
}
```

الجزء التاسع: الربط بين قاعدتي البيانات

عند إنشاء موظف جديد:

```
// 1. إنشاء حساب في IdentityDB
var user = new ApplicationUser
{
UserName = email,
Email = email,
FirstName = firstName,
LastName = lastName
};

var result = await _userManager.CreateAsync(user, password);

// 2. إضافة دور للمستخدم
await _userManager.AddToRoleAsync(user, role);

// 3. إنشاء موظف في InventoryDB مع ربط الـ AspNetUserId
var employee = new Employee
{
FirstName = firstName,
LastName = lastName,
Email = email,
AspNetUserId = user.Id,
DepartmentId = departmentId
};
```

```
await _unitOfWork.Employees.AddAsync(employee);
await _unitOfWork.SaveChangesAsync();
```

الجزء العاشر: الاختبار (Unit Testing)

1. مثال اختبار لـ Product Service

```
[TestClass]
public class ProductServiceTests
{
private Mock<IUnitOfWork> _unitOfWorkMock;
private Mock<IMapper> _mapperMock;
private ProductService _productService;
```

```
[TestInitialize]
public void Setup()
{
    _unitOfWorkMock = new Mock<IUnitOfWork>();
    _mapperMock = new Mock<IMapper>();
    _productService = new ProductService(_unitOfWorkMock.Object, _mapperM
}

[TestMethod]
public async Task GetAllProductsAsync_ReturnsProductList()
{
    // Arrange
    var products = new List<Product>
    {
        new Product { Id = 1, Name = "Product 1" },
        new Product { Id = 2, Name = "Product 2" }
    };

    _unitOfWorkMock.Setup(u => u.Products.GetAllAsync())
        .ReturnsAsync(products);

    var productDtos = new List<ProductDto>
    {
        new ProductDto { Id = 1, Name = "Product 1" },
        new ProductDto { Id = 2, Name = "Product 2" }
    };
```

```
    _mapperMock.Setup(m => m.Map<IEnumerable<ProductDto>>(products))
        .Returns(productDtos);

    // Act
    var result = await _productService.GetAllProductsAsync();

    // Assert
    Assert.AreEqual(2, result.Count());
    _unitOfWorkMock.Verify(u => u.Products.GetAllAsync(), Times.Once);
}

[TestMethod]
public async Task DeleteProductAsync_WithInvalidId_ThrowsException()
{
    // Arrange
    int invalidId = 999;
    _unitOfWorkMock.Setup(u => u.Products.GetByIdAsync(invalidId))
        .ReturnsAsync((Product)null);

    // Act & Assert
    await Assert.ThrowsExceptionAsync<KeyNotFoundException>(() =>
        _productService.DeleteProductAsync(invalidId));
}
```

}

الجزء الحادي عشر: التصميم والـ Views

1. تخطيط الصفحات الرئيسية

الصفحات الأساسية التي ستحتاج إليها:

في Admin Area:

- Dashboard لعرض الإحصائيات العامة
- صفحة إدارة المنتجات (Index, Create, Edit, Delete)
- صفحة إدارة التصنيفات
- صفحة إدارة الموردين والعملاء
- صفحة إدارة المستخدمين والأدوار

في Manager Area:

- Dashboard خاص بالمديرين
- صفحة عرض أوامر البيع والشراء
- صفحة تقارير المخزون
- صفحة التنبيهات

في Employee Area:

- صفحة تسجيل عمليات البيع والشراء
- صفحة عرض المخزون الحالي

2. مثال Layout شامل

```
@{
ViewData["Title"] = "Inventory Management System";
}
```

نظام إدارة المخزون

@if (User.IsInRole("Admin")) {

- إدارة المنتجات

}

@RenderBody()

جميع الحقوق محفوظة © 2025 - نظام إدارة المخزون

@await RenderSectionAsync("Scripts", required: false)

الجزء الثاني عشر: أفضل الممارسات والأداء

1. تحسين الأداء

- **استخدام AsNoTracking:** عند القراءة فقط بدون تعديل
- **Pagination:** لتقليل البيانات المرسلة
- **Lazy Loading:** تحميل البيانات عند الحاجة فقط
- **Indexes:** على الأعمدة المستخدمة بكثرة في البحث

2. الأمان

- **Prevent SQL Injection:** استخدم Parameterized Queries (EF يفعل هذا تلقائياً)
- **Input Validation:** تحقق من البيانات المدخلة
- **Authorization:** تأكد من الصلاحيات على كل Action
- **CSRF Protection:** استخدم Anti-Forgery Tokens
- **Password Security:** لا تحفظ passwords بشكل مباشر

3. Logging والا Error Handling

public class ErrorHandlingMiddleware
{
private readonly RequestDelegate _next;
private readonly ILogger<ErrorHandlingMiddleware> _logger;

```
public ErrorHandlingMiddleware(RequestDelegate next,
    ILogger<ErrorHandlingMiddleware> logger)
{
    _next = next;
    _logger = logger;
}

public async Task InvokeAsync(HttpContext context)
{
    try
    {
        await _next(context);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An unhandled exception occurred");
        context.Response.StatusCode = 500;
        await context.Response.WriteAsJsonAsync(new { message = "Internal Serv
    }
}
```

}

4. استخدام AutoMapper

public class MappingProfile : Profile
{
public MappingProfile()
{
CreateMap<Product, ProductDto>().ReverseMap();
CreateMap<Product, CreateProductDto>().ReverseMap();
CreateMap<Product, UpdateProductDto>().ReverseMap();
CreateMap<Category, CategoryDto>().ReverseMap();
}
}

الجزء الثالث عشر: خطة التطوير التفصيلية

المرحلة 1: الأساس (الأسبوع الأول)

- [ ] إنشاء Solution والمشاريع
- [ ] تصميم وتطبيق الـ Entities والـ Models
- [ ] إنشاء Migrations وقاعدتي البيانات
- [ ] تطبيق Unit of Work و Generic Repository

المرحلة 2: الهوية والأمان (الأسبوع الثاني)

- [ ] تكوين الـ Identity في قاعدة بيانات منفصلة
- [ ] إعداد الأدوار والصلاحيات
- [ ] ربط قاعدتي البيانات
- [ ] إنشاء نظام تسجيل الدخول

المرحلة 3: الـ Areas والـ Controllers (الأسبوع الثالث)

- [ ] إنشاء Admin Area مع Controllers
- [ ] إنشاء Manager Area
- [ ] إنشاء Employee Area
- [ ] تطبيق Authorization على الـ Controllers

المرحلة 4: الخدمات والـ Business Logic (الأسبوع الرابع)

- [ ] بناء Services للعمليات الأساسية
- [ ] خدمة إدارة الطلبات (Purchase & Sales Orders)
- [ ] خدمة حركات المخزون
- [ ] نظام التنبيهات

المرحلة 5: الواجهات (Views) (الأسبوع الخامس)

- [ ] Dashboard مع الإحصائيات
- [ ] صفحات إدارة المنتجات والتصنيفات
- [ ] صفحات الطلبات والمخزون
- [ ] صفحات التقارير

المرحلة 6: التقارير والإصدارات (الأسبوع السادس)

- [ ] نظام التقارير المتقدم
- [ ] Export to PDF و Excel
- [ ] الرسوم البيانية والإحصائيات
- [ ] نظام الإخطارات

المرحلة 7: الاختبارات (الأسبوع السابع)

- [ ] كتابة Unit Tests
- [ ] Integration Tests
- [ ] اختبار الأداء
- [ ] اختبار الأمان

المرحلة 8: التحسينات والإطلاق (الأسبوع الثامن)

- [ ] تحسين الأداء
- [ ] مراجعة الكود
- [ ] التوثيق
- [ ] الإطلاق والـ Deployment

الخاتمة والنصائح الذهبية

هذا المشروع سيعطيك فرصة ذهبية لتطبيق:

1. **Architecture Patterns:** Repository, Unit of Work, Dependency Injection
2. **Authentication & Authorization:** Identity Framework
3. **Database Design:** Relationships، Migrations
4. **Clean Code:** SOLID Principles
5. **Testing:** Unit و Integration Tests
6. **Best Practices:** Security، Performance، Logging

نصائح للنجاح:

- ابدأ بخطوات صغيرة ومضطردة
- كتب tests وأنت تطور
- اعرض تقدمك على GitHub منذ البداية
- وثق كل decision معماري تتخذه
- اطلب feedback من مطورين آخرين
- لا تتردد في إعادة Refactoring
- اختبر على أنظمة مختلفة قبل الإطلاق

بالتوفيق في مشروعك! هذا سيكون إضافة قوية لـ Portfolio الخاص بك في المقابلات والعمل.