نظام إدارة المخزون الشامل

Table of Contents

ASP.NET Core MVC مع موديول الهوية المتقدم والـ Barcode

المحتويات

1. نظام الأدوار والمصادقة المُحسّن
2. ربط ApplicationUser مع Employee
3. نظام Barcode للمنتجات
4. ERD شامل ومنظم
5. أكواد عملية للتطبيق

الجزء الأول: نظام الأدوار والمصادقة المُحسّن

1. تبسيط الأقسام (Departments)

بدلاً من جدول Departments منفصل، سنستخدم نظام أبسط حيث:

- **Admin**: مدير النظام الرئيسي (لديه جميع الصلاحيات)
- **Manager**: مدير المخزون (يدير الطلبات والمخزون والموظفين)
- **Employee**: موظف عادي (يدخل البيانات والعمليات البسيطة)

2. ApplicationUser المُحسّنة

```
public class ApplicationUser : IdentityUser
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string FullName =&gt; $"{FirstName} {LastName}";
```

```
    public bool IsActive { get; set; } = true;
    public DateTime CreatedAt { get; set; } = DateTime.Now;
    public DateTime? LastLogin { get; set; }
    public string ProfileImageUrl { get; set; }

    // العلاقة مع Employee (One-to-One)
    public int? EmployeeId { get; set; }
    public Employee Employee { get; set; }
}
```

3. جدول Employee المحدّث

```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string FullName =&gt; $"{FirstName} {LastName}";
    public string Email { get; set; }
    public string Phone { get; set; }
    public string Position { get; set; } // مثل: Manager, Supervisor, Operator
    public decimal Salary { get; set; }
    public DateTime HireDate { get; set; } = DateTime.Now;
    public bool IsActive { get; set; } = true;

    // الربط مع ApplicationUser من IdentityDB
    public string AspNetUserId { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.Now;
    public DateTime ModifiedAt { get; set; } = DateTime.Now;

    // العلاقات
    public ICollection&lt;SalesOrder&gt; SalesOrders { get; set; }
    public ICollection&lt;PurchaseOrder&gt; PurchaseOrders { get; set; }
}
```

4. تكوين العلاقة في DbContext

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

    // علاقة Employee مع SalesOrder
    modelBuilder.Entity&lt;SalesOrder&gt;()
        .HasOne(o =&gt; o.Employee)
        .WithMany(e =&gt; e.SalesOrders)
        .HasForeignKey(o =&gt; o.EmployeeId)
        .OnDelete(DeleteBehavior.Restrict);

    // علاقة Employee مع PurchaseOrder
    modelBuilder.Entity&lt;PurchaseOrder&gt;()
        .HasOne(o =&gt; o.Employee)
        .WithMany(e =&gt; e.PurchaseOrders)
        .HasForeignKey(o =&gt; o.EmployeeId)
        .OnDelete(DeleteBehavior.Restrict);

    // Index على Email للبحث السريع
    modelBuilder.Entity&lt;Employee&gt;()
        .HasIndex(e =&gt; e.Email)
        .IsUnique();
}
```

الجزء الثاني: إنشاء Roles والتهيئة

1. تهيئة الأدوار (Roles Initialization)

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

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

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

    public static async Task CreateDefaultAdmin(IServiceProvider serviceProvider)
    {
        var userManager = serviceProvider.GetRequiredService&lt;UserManager&lt;ApplicationU
        var email = "admin@inventory.com";

        if (await userManager.FindByEmailAsync(email) == null)
        {
            var adminUser = new ApplicationUser
            {
                UserName = email,
                Email = email,
                EmailConfirmed = true,
                FirstName = "نظام",
                LastName = "الإدارة",
                IsActive = true
            };

            var result = await userManager.CreateAsync(adminUser, "Admin@123");

            if (result.Succeeded)
            {
                await userManager.AddToRoleAsync(adminUser, "Admin");
            }
        }
    }
}
```

2. استدعاء التهيئة في Program.cs

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

الجزء الثالث: ربط ApplicationUser مع Employee

1. خدمة إنشاء Employees جديدة (Admin فقط)

```
public interface IEmployeeService
{
    Task&lt;EmployeeDto&gt; CreateEmployeeWithUserAsync(CreateEmployeeDto dto, string role
    Task&lt;EmployeeDto&gt; GetEmployeeByIdAsync(int id);
    Task&lt;IEnumerable&lt;EmployeeDto&gt;&gt; GetAllEmployeesAsync();
    Task UpdateEmployeeAsync(int id, UpdateEmployeeDto dto);
    Task DeleteEmployeeAsync(int id);
    Task&lt;EmployeeDto&gt; GetEmployeeByUserIdAsync(string userId);
}

public class EmployeeService : IEmployeeService
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly UserManager&lt;ApplicationUser&gt; _userManager;
    private readonly IMapper _mapper;
    private readonly ILogger&lt;EmployeeService&gt; _logger;

    public EmployeeService(
        IUnitOfWork unitOfWork,
        UserManager&lt;ApplicationUser&gt; userManager,
        IMapper mapper,
        ILogger&lt;EmployeeService&gt; logger)
    {
        _unitOfWork = unitOfWork;
        _userManager = userManager;
        _mapper = mapper;
        _logger = logger;
    }

    /// &lt;summary&gt;
    /// إنشاء موظف جديد مع حساب في نظام الهوية
    /// هذه العملية مقصورة على Admin فقط
    /// &lt;/summary&gt;
    public async Task&lt;EmployeeDto&gt; CreateEmployeeWithUserAsync(CreateEmployeeDto dto
    {
        // التحقق من أن الدور صحيح
        if (!new[] { "Manager", "Employee" }.Contains(role))
            throw new ArgumentException("Role must be Manager or Employee");

        try
        {
            // 1. إنشاء مستخدم جديد في IdentityDB
            var appUser = new ApplicationUser
            {
                UserName = dto.Email,
                Email = dto.Email,
                EmailConfirmed = true,
                FirstName = dto.FirstName,
                LastName = dto.LastName,
                IsActive = true
            };

            // إنشاء كلمة مرور مؤقتة
            string tempPassword = GenerateTemporaryPassword();
            var userResult = await _userManager.CreateAsync(appUser, tempPassword);

            if (!userResult.Succeeded)
                throw new Exception($"Failed to create user: {string.Join(", ", userResult

            // 2. إضافة الدور للمستخدم
```

```
            await _userManager.AddToRoleAsync(appUser, role);

            // 3. إنشاء سجل موظف في InventoryDB
            var employee = new Employee
            {
                FirstName = dto.FirstName,
                LastName = dto.LastName,
                Email = dto.Email,
                Phone = dto.Phone,
                Position = role,
                Salary = dto.Salary,
                HireDate = dto.HireDate,
                AspNetUserId = appUser.Id,
                IsActive = true
            };

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

            _logger.LogInformation($"Employee created successfully: {employee.FullName}");

            return _mapper.Map&lt;EmployeeDto&gt;(employee);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error creating employee");
            throw;
        }
    }

    public async Task&lt;EmployeeDto&gt; GetEmployeeByIdAsync(int id)
    {
        var employee = await _unitOfWork.Employees.GetByIdAsync(id);
        return _mapper.Map&lt;EmployeeDto&gt;(employee);
    }

    public async Task&lt;IEnumerable&lt;EmployeeDto&gt;&gt; GetAllEmployeesAsync()
    {
        var employees = await _unitOfWork.Employees.GetAllAsync();
        return _mapper.Map&lt;IEnumerable&lt;EmployeeDto&gt;&gt;(employees);
    }

    public async Task UpdateEmployeeAsync(int id, UpdateEmployeeDto dto)
    {
        var employee = await _unitOfWork.Employees.GetByIdAsync(id);
        if (employee == null)
            throw new KeyNotFoundException($"Employee with ID {id} not found");

        _mapper.Map(dto, employee);
        employee.ModifiedAt = DateTime.Now;
        _unitOfWork.Employees.Update(employee);
        await _unitOfWork.SaveChangesAsync();
    }

    public async Task DeleteEmployeeAsync(int id)
    {
        var employee = await _unitOfWork.Employees.GetByIdAsync(id);
        if (employee == null)
            throw new KeyNotFoundException($"Employee with ID {id} not found");

        // حذف المستخدم من IdentityDB
        var user = await _userManager.FindByIdAsync(employee.AspNetUserId);
        if (user != null)
            await _userManager.DeleteAsync(user);

        // حذف الموظف من InventoryDB
```

```
        _unitOfWork.Employees.Delete(employee);
        await _unitOfWork.SaveChangesAsync();
    }

    public async Task&lt;EmployeeDto&gt; GetEmployeeByUserIdAsync(string userId)
    {
        var employee = await _unitOfWork.Employees
            .FindAsync(e =&gt; e.AspNetUserId == userId);
        return _mapper.Map&lt;EmployeeDto&gt;(employee.FirstOrDefault());
    }

    private string GenerateTemporaryPassword()
    {
        // إنشاء كلمة مرور مؤقتة قوية
        var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%";
        var random = new Random();
        var password = new string(Enumerable.Repeat(chars, 12).Select(s =&gt; s[random.Nex
        return password;
    }
}
```

2. Controller لإدارة الموظفين (Admin Area فقط)

```
[Area("Admin")]
[Authorize(Roles = "Admin")]
public class EmployeesController : Controller
{
    private readonly IEmployeeService _employeeService;
    private readonly ILogger&lt;EmployeesController&gt; _logger;

    public EmployeesController(IEmployeeService employeeService, ILogger&lt;EmployeesContro
    {
        _employeeService = employeeService;
        _logger = logger;
    }

    public async Task&lt;IActionResult&gt; Index()
    {
        var employees = await _employeeService.GetAllEmployeesAsync();
        return View(employees);
    }

    public IActionResult Create()
    {
        var model = new CreateEmployeeDto();
        ViewBag.Roles = new[] { "Manager", "Employee" };
        return View(model);
    }

    [HttpPost]
    public async Task&lt;IActionResult&gt; Create(CreateEmployeeDto dto, string selectedRol
    {
        if (ModelState.IsValid)
        {
            try
            {
                await _employeeService.CreateEmployeeWithUserAsync(dto, selectedRole);
                return RedirectToAction(nameof(Index));
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error creating employee");
                ModelState.AddModelError("", ex.Message);
            }
```

```
        }

        ViewBag.Roles = new[] { "Manager", "Employee" };
        return View(dto);
    }

    public async Task&lt;IActionResult&gt; Edit(int id)
    {
        var employee = await _employeeService.GetEmployeeByIdAsync(id);
        var updateDto = new UpdateEmployeeDto
        {
            FirstName = employee.FirstName,
            LastName = employee.LastName,
            Phone = employee.Phone,
            Salary = employee.Salary
        };
        return View(updateDto);
    }

    [HttpPost]
    public async Task&lt;IActionResult&gt; Edit(int id, UpdateEmployeeDto dto)
    {
        if (ModelState.IsValid)
        {
            try
            {
                await _employeeService.UpdateEmployeeAsync(id, dto);
                return RedirectToAction(nameof(Index));
            }
            catch (Exception ex)
            {
                ModelState.AddModelError("", ex.Message);
            }
        }
        return View(dto);
    }

    [HttpPost]
    public async Task&lt;IActionResult&gt; Delete(int id)
    {
        try
        {
            await _employeeService.DeleteEmployeeAsync(id);
            return RedirectToAction(nameof(Index));
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}
```

الجزء الرابع: نظام Barcode للمنتجات

1. موديل Product المُحسّن

```
public class Product
{
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }
```

```
    public string Description { get; set; }

    [Required]
    public decimal Price { get; set; }

    [Required]
    public int QuantityInStock { get; set; }

    [Required]
    public int MinimumStockLevel { get; set; }

    // الرمز الشريطي (Barcode)
    [Required]
    [StringLength(128)]
    public string Barcode { get; set; }

    // صورة الرمز الشريطي (Base64 أو رابط)
    public string BarcodeImage { 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&lt;PurchaseOrderDetail&gt; PurchaseOrderDetails { get; set; }
    public ICollection&lt;SalesOrderDetail&gt; SalesOrderDetails { get; set; }
    public ICollection&lt;StockMovement&gt; StockMovements { get; set; }
}
```

2. خدمة توليد Barcode

```
using BarcodeLib;

public interface IBarcodeService
{
    string GenerateBarcode(string barcodeValue);
    byte[] GenerateBarcodeImage(string barcodeValue);
}

public class BarcodeService : IBarcodeService
{
    public string GenerateBarcode(string barcodeValue)
    {
        // تحويل القيمة إلى صيغة قياسية
        if (string.IsNullOrEmpty(barcodeValue))
            throw new ArgumentException("Barcode value cannot be empty");

        // تأكد من أن الرمز فريد
        return barcodeValue.Length &gt; 128
            ? barcodeValue.Substring(0, 128)
            : barcodeValue;
    }

    public byte[] GenerateBarcodeImage(string barcodeValue)
    {
        try
        {
            Barcode barcode = new Barcode();
            Image img = barcode.Encode(BarcodeEncoding.Code128, barcodeValue, Color.Black,

            using (MemoryStream ms = new MemoryStream())
            {
```

```
                img.Save(ms, ImageFormat.Png);
                return ms.ToArray();
            }
        }
        catch (Exception ex)
        {
            throw new Exception($"Error generating barcode: {ex.Message}");
        }
    }
}
```

3. خدمة إضافة/تحديث المنتجات مع Barcode

```
public interface IProductService
{
    Task&lt;int&gt; CreateProductAsync(CreateProductDto dto);
    Task UpdateProductAsync(int id, UpdateProductDto dto);
    Task&lt;ProductDto&gt; GetProductByBarcodeAsync(string barcode);
    Task&lt;byte[]&gt; GetBarcodeImageAsync(string barcode);
}

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

    public ProductService(
        IUnitOfWork unitOfWork,
        IBarcodeService barcodeService,
        IMapper mapper)
    {
        _unitOfWork = unitOfWork;
        _barcodeService = barcodeService;
        _mapper = mapper;
    }

    public async Task&lt;int&gt; CreateProductAsync(CreateProductDto dto)
    {
        // التحقق من تفرد الرمز الشريطي
        var existingProduct = await _unitOfWork.Products
            .FindAsync(p =&gt; p.Barcode == dto.Barcode);

        if (existingProduct.Any())
            throw new Exception($"Barcode {dto.Barcode} already exists");

        // توليد الرمز الشريطي
        string barcode = _barcodeService.GenerateBarcode(dto.Barcode ?? Guid.NewGuid().ToS

        // توليد صورة الرمز الشريطي
        byte[] barcodeImageBytes = _barcodeService.GenerateBarcodeImage(barcode);
        string barcodeImageBase64 = Convert.ToBase64String(barcodeImageBytes);

        var product = new Product
        {
            Name = dto.Name,
            Description = dto.Description,
            Price = dto.Price,
            QuantityInStock = dto.QuantityInStock,
            MinimumStockLevel = dto.MinimumStockLevel,
            CategoryId = dto.CategoryId,
            Barcode = barcode,
            BarcodeImage = $"{barcodeImageBase64}"
        };
```

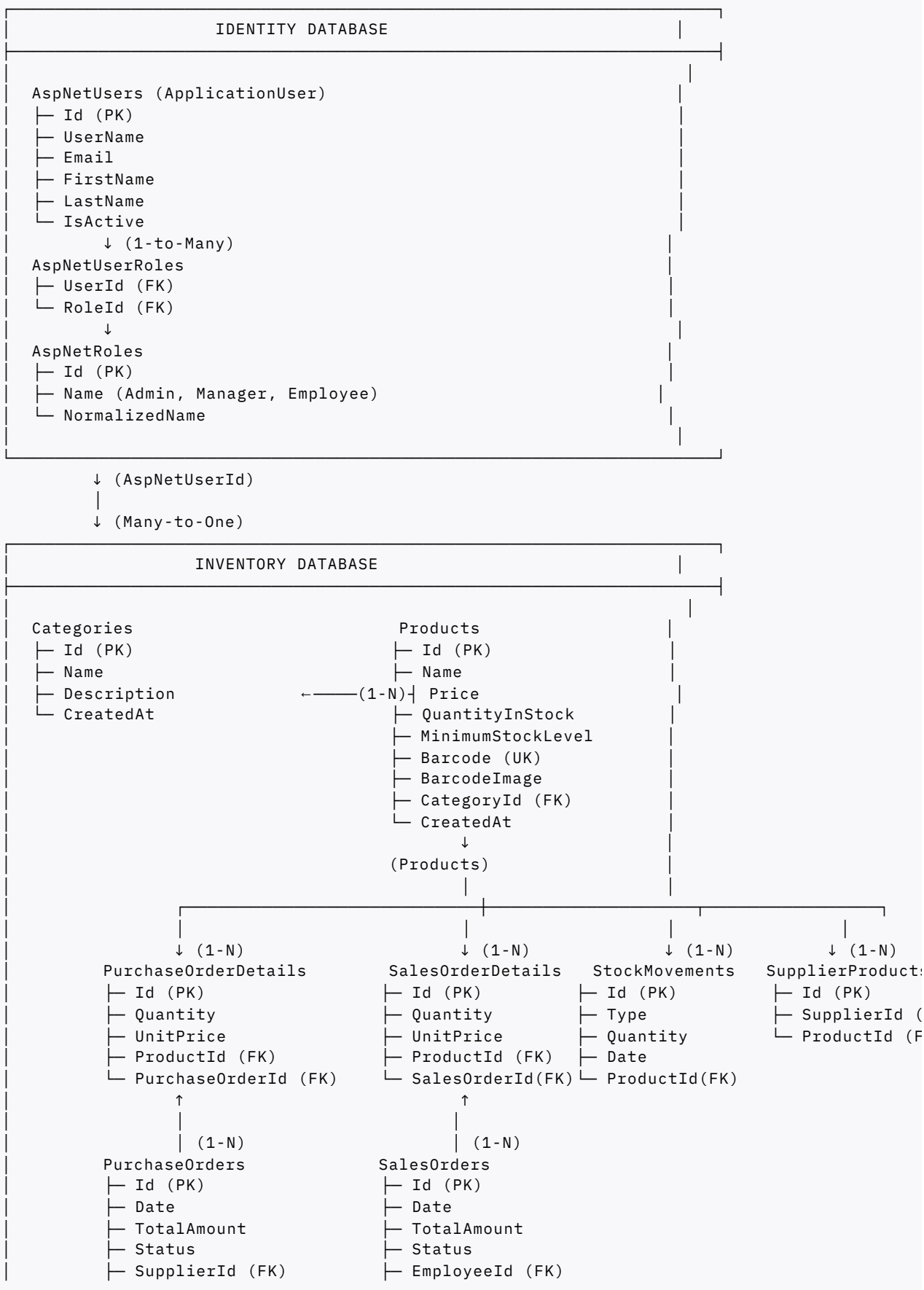
```
        await _unitOfWork.Products.AddAsync(product);
        await _unitOfWork.SaveChangesAsync();

        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");

        // إذا تغيّر الرمز الشريطي، تحقق من تفرده
        if (dto.Barcode != product.Barcode)
        {
            var existingProduct = await _unitOfWork.Products
                .FindAsync(p =&gt; p.Barcode == dto.Barcode &amp;&amp; p.Id != id);

            if (existingProduct.Any())
                throw new Exception($"Barcode {dto.Barcode} already exists");

            // توليد الصورة الجديدة
            byte[] barcodeImageBytes = _barcodeService.GenerateBarcodeImage(dto.Barcode);
            product.BarcodeImage = $"{Convert.ToBase64String(barcode
        }

        product.Name = dto.Name;
        product.Description = dto.Description;
        product.Price = dto.Price;
        product.QuantityInStock = dto.QuantityInStock;
        product.MinimumStockLevel = dto.MinimumStockLevel;
        product.CategoryId = dto.CategoryId;
        product.Barcode = dto.Barcode;
        product.ModifiedAt = DateTime.Now;

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

    public async Task&lt;ProductDto&gt; GetProductByBarcodeAsync(string barcode)
    {
        var product = (await _unitOfWork.Products.FindAsync(p =&gt; p.Barcode == barcode))
            .FirstOrDefault();

        if (product == null)
            throw new KeyNotFoundException($"Product with barcode {barcode} not found");

        return _mapper.Map&lt;ProductDto&gt;(product);
    }

    public async Task&lt;byte[]&gt; GetBarcodeImageAsync(string barcode)
    {
        return _barcodeService.GenerateBarcodeImage(barcode);
    }
}
```

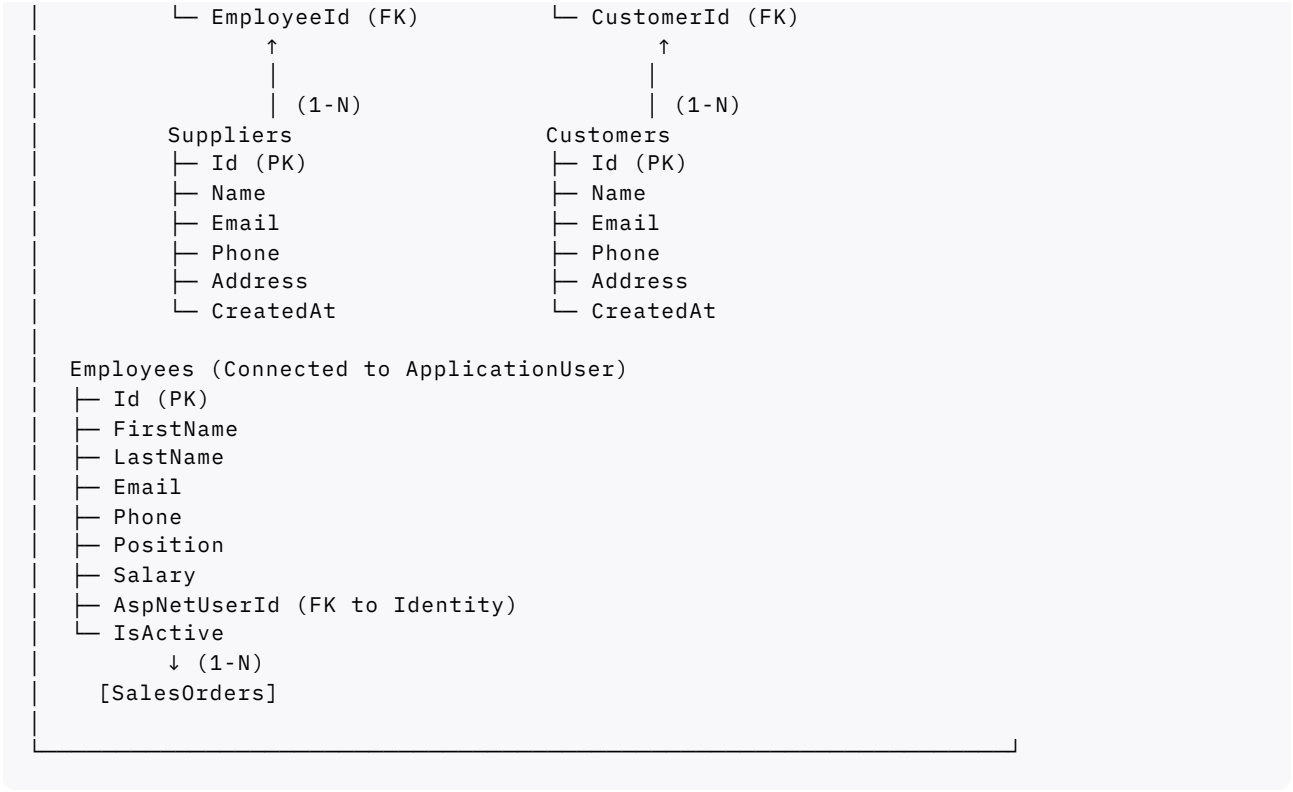
4. تسجيل الخدمات في Program.cs

```
builder.Services.AddScoped&lt;IBarcodeService, BarcodeService&gt;();
builder.Services.AddScoped&lt;IProductService, ProductService&gt;();
builder.Services.AddScoped&lt;IEmployeeService, EmployeeService&gt;();
```

الجزء الخامس: ERD الشامل والمنظم

العلاقات الرئيسية:





جدول العلاقات المفصل:

من الجدول	نوع العلاقة	إلى الجدول	ملاحظات
Categories	1:N	Products	كل فئة تحتوي على منتجات متعددة
Products	1:N	PurchaseOrderDetails	منتج واحد في طلبات شراء متعددة
Products	1:N	SalesOrderDetails	منتج واحد في طلبات بيع متعددة
Products	1:N	StockMovements	تتبع حركات المخزون
Products	1:N	SupplierProducts	علاقة متعددة الموردين
Suppliers	1:N	PurchaseOrders	موردون لهم طلبات شراء متعددة
Suppliers	1:N	SupplierProducts	موردون يوفرون منتجات متعددة
Customers	1:N	SalesOrders	عميل واحد له طلبات بيع متعددة
Employees	1:N	SalesOrders	موظف واحد ينفذ طلبات بيع متعددة
Employees	1:N	PurchaseOrders	موظف واحد ينفذ طلبات شراء متعددة
PurchaseOrders	1:N	PurchaseOrderDetails	طلب شراء واحد له تفاصيل متعددة
SalesOrders	1:N	SalesOrderDetails	طلب بيع واحد له تفاصيل متعددة
ApplicationUser	1:1	Employees	كل مستخدم له موظف واحد فقط

الجزء السادس: DTOs و ViewModels

Employee DTOs

```
public class CreateEmployeeDto
{
    [Required(ErrorMessage = "الاسم الأول مطلوب")]
    public string FirstName { get; set; }

    [Required(ErrorMessage = "الاسم الأخير مطلوب")]
    public string LastName { get; set; }

    [Required(ErrorMessage = "البريد الإلكتروني مطلوب")]
    [EmailAddress(ErrorMessage = "صيغة البريد الإلكتروني غير صحيحة")]
    public string Email { get; set; }

    [Required(ErrorMessage = "رقم الهاتف مطلوب")]
    public string Phone { get; set; }

    [Required(ErrorMessage = "الراتب مطلوب")]
    [Range(0, double.MaxValue)]
    public decimal Salary { get; set; }

    public DateTime HireDate { get; set; } = DateTime.Now;
}

public class UpdateEmployeeDto
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Phone { get; set; }
    public decimal Salary { get; set; }
}

public class EmployeeDto
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string FullName { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public string Position { get; set; }
    public decimal Salary { get; set; }
    public DateTime HireDate { get; set; }
    public bool IsActive { get; set; }
}
```

Product DTOs

```
public class CreateProductDto
{
    [Required(ErrorMessage = "اسم المنتج مطلوب")]
    public string Name { get; set; }

    public string Description { get; set; }

    [Required(ErrorMessage = "السعر مطلوب")]
    [Range(0.01, double.MaxValue)]
    public decimal Price { get; set; }

    [Required(ErrorMessage = "الكمية مطلوبة")]
```

```
    [Range(0, int.MaxValue)]
    public int QuantityInStock { get; set; }

    [Required(ErrorMessage = "الحد الأدنى للمخزون مطلوب")]
    public int MinimumStockLevel { get; set; }

    [Required(ErrorMessage = "رمز المنتج مطلوب")]
    [StringLength(128)]
    public string Barcode { get; set; }

    [Required(ErrorMessage = "الفئة مطلوبة")]
    public int CategoryId { get; set; }
}

public class UpdateProductDto
{
    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 string Barcode { get; set; }
    public int CategoryId { get; set; }
}

public class ProductDto
{
    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 string Barcode { get; set; }
    public string BarcodeImage { get; set; }
    public string CategoryName { get; set; }
    public bool IsLowStock =&gt; QuantityInStock &lt;= MinimumStockLevel;
}
```

الجزء السابع: AutoMapper Configuration

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        // Employee Mappings
        CreateMap&lt;Employee, EmployeeDto&gt;()
            .ForMember(d =&gt; d.FullName, o =&gt; o.MapFrom(s =&gt; $"{s.FirstName} {s.Las

        CreateMap&lt;CreateEmployeeDto, Employee&gt;();
        CreateMap&lt;UpdateEmployeeDto, Employee&gt;()
            .ForAllMembers(opts =&gt; opts.Condition((src, dest, srcMember) =&gt; srcMembe

        // Product Mappings
        CreateMap&lt;Product, ProductDto&gt;()
            .ForMember(d =&gt; d.CategoryName, o =&gt; o.MapFrom(s =&gt; s.Category.Name))
            .ForMember(d =&gt; d.IsLowStock, o =&gt; o.MapFrom(s =&gt; s.QuantityInStock &l

        CreateMap&lt;CreateProductDto, Product&gt;();
        CreateMap&lt;UpdateProductDto, Product&gt;()
            .ForAllMembers(opts =&gt; opts.Condition((src, dest, srcMember) =&gt; srcMembe

        // Category Mappings
```

```
        CreateMap&lt;Category, CategoryDto&gt;();
        CreateMap&lt;CreateCategoryDto, Category&gt;();
    }
}
```

الجزء الثامن: ملخص نمط الأمان

تدفق العملية:

1. **يقوم بإنشاء حساب موظف جديد Admin:**
   - يدخل: الاسم الأول، الأخير، البريد، الهاتف، الراتب، الدور Admin
   - النظام ينشئ حساب في IdentityDB (مع كلمة مرور مؤقتة)
   - النظام ينشئ سجل موظف في InventoryDB (مع رابط AspNetUserId)
   - النظام يضيف الدور (Manager أو Employee)
2. **الموظف يسجل دخوله:**
   - يستخدم بريده الإلكتروني وكلمة المرور
   - النظام يتحقق من IdentityDB
   - يتم تحميل Roles و Claims
   - يمكن للموظف الوصول إلى الصفحات المسموحة فقط
3. **التحكم في الصلاحيات:**
   - Admin: إنشاء/تعديل/حذف أي شيء
   - Manager: إدارة الطلبات والمخزون والموظفين
   - Employee: عمليات محدودة فقط

النقاط المهمة:

✅ **بساطة النظام:** تم حذف جدول Departments وتبسيط الأقسام
✅ **الأمان:** فقط Admin يمكنه إنشاء الحسابات
✅ **الربط الآمن:** Employee مرتبط بـ ApplicationUser عبر AspNetUserId
✅ **Barcode:** كل منتج له رمز شريطي فريد وصورة
✅ **Tracking:** كل عملية يتم تتبعها مع Employee وتاريخ

الخطوات التالية:

1. أنشئ Migration جديدة لإضافة حقول Barcode
2. ثبّت مكتبة BarcodeLib من NuGet
3. أنشئ Unit Tests للتحقق من العمليات
4. اختبر نظام الأدوار والصلاحيات
5. ابنِ الواجهات (Views) للإدارة