🎯 Production Readiness - ما الناقص في المشروع

> ملف يركز على الأشياء الناقصة فقط والخطوات العملية لتطبيقها

📋 جدول المحتويات

1. appsettings.Production.json

❌ الحالة الحالية:

- ما فيش appsettings.Production.json
- CORS origins من localhost (خطير!)
- Serilog يكتب Seq locally (غير آمن)

✅ الحل:

إنشاء ملف جديد: `src/ECommerce.API/appsettings.Production.json`

```
{
  "AppSettings": {
    "CorsPolicyName": "ECommercePolicy",
    "AllowedOrigins": [
      "https://yourdomain.com",
      "https://www.yourdomain.com"
    ]
  },
  "Serilog": {
    "MinimumLevel": "Warning",
    "WriteTo": [
      {
        "Name": "Console"
      }
    ]
  },
  "JwtSettings": {
    "TokenExpirationInMinutes": 15,
    "Issuer": "ECommerceApi",
    "Audience": "ECommerceUsers"
  }
}
```

📌 ملاحظات:

- استبدل `yourdomain.com` بـ domain فعلي
- Log level = Warning (مش Information)
- JWT expiration = 15 دقيقة مع refresh token
- Seq disabled (استخدم Application Insights بدل كده)

2. Rate Limiting

❌ الحالة الحالية:

- ما فيش protection من brute force attacks
- بلا حد API أي حد بيقدر يستخدم الـ

✅ الحل:

أضف في `src/ECommerce.API/DependencyInjection.cs`:

```
public static IServiceCollection AddPresentation(this IServiceCollection services, IConfiguratio
{
    services.AddHttpContextAccessor();

    // ← أضف هنا
    services.AddRateLimiter(options =>
    {
        options.AddFixedWindowLimiter(policyName: "FixedWindowPolicy", configure: options =>
        {
            options.PermitLimit = 100;
            options.Window = TimeSpan.FromMinutes(1);
            options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst;
            options.QueueLimit = 2;
        });
    });

    services
        .AddCustomProblemDetails()
        .AddCustomApiVersioning()
        // ... باقي الـ setup
}

public static IApplicationBuilder UseCoreMiddlewares(this IApplicationBuilder app, IConfiguratio
{
    app.UseExceptionHandler();
    app.UseStatusCodePages();
    app.UseHttpsRedirection();

    // ← أضف هنا قبل Serilog
    app.UseRateLimiter();

    app.UseSerilogRequestLogging();
    // ... باقي الـ middleware
}
```

أضف في `src/ECommerce.API/Controllers/ApiController.cs`:

```
[ApiController]
[Route("api/v{version:apiVersion}/[controller]")]
[RequireRateLimiting("FixedWindowPolicy")] // ← أضف هنا
public abstract class ApiController : ControllerBase
{
}
```

📌 ملاحظات:

- 100 requests per minute لكل IP
- غيّر الأرقام حسب احتياجاتك
- استخدم `[SkipRateLimiting]` للـ endpoints الحساسة

3. Health Checks

❌ الحالة الحالية:

- ما فيش health check endpoint
- ما تقدر تتحقق من حالة الـ app الـ load balancers

✅ الحل:

أضف في `src/ECommerce.Infrastructure/DependencyInjection.cs`:

```
public static IServiceCollection AddInfrastructure(this IServiceCollection services, IConfigurat
{
    // ... الكود الموجود ...

    services.AddScoped<IIdentityService, IdentityService>();
    services.AddScoped<ITokenProvider, TokenProvider>();
    services.AddScoped<IBasketService, BasketService>();

    // ← أضف هنا في الآخر
    services.AddHealthChecks()
        .AddSqlServer(connectionString, name: "Database", tags: new[] { "ready" })
        .AddRedis(configuration.GetConnectionString("Redis") ?? "localhost:6379",
                  name: "Redis",
                  tags: new[] { "ready" });

    return services;
}
```

أضف في `src/ECommerce.API/Program.cs`:

```
app.MapControllers();

// ← أضف قبل app.Run()
app.MapHealthChecks("/health");
app.MapHealthChecks("/health/ready", new HealthCheckOptions
{
    Predicate = healthCheck => healthCheck.Tags.Contains("ready")
});

app.Run();
```

📌 ملاحظات:

- `/health` = liveness check (هل الـ app شغال؟)
- `/health/ready` = readiness check (هل الـ dependencies شغالة؟)
- يمكن استخدام مع Kubernetes

4. Response Compression

❌ الحالة الحالية:

- الـ responses بتتنقل كاملة (بدون compression)
- تضيع bandwidth و performance

✅ الحل:

في `src/ECommerce.API/DependencyInjection.cs` :

```
public static IServiceCollection AddPresentation(this IServiceCollection services, IConfiguratio
{
    services.AddHttpContextAccessor();

    // ← أضف هنا
    services.AddResponseCompression(options =>
    {
        options.EnableForHttps = true;
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<BrotliCompressionProvider>();
        options.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(
            new[] { "application/json", "application/json;charset=utf-8" });
    });

    services.AddRateLimiter(options => { /* ... */ });
    // ... باقي الكود
}

public static IApplicationBuilder UseCoreMiddlewares(this IApplicationBuilder app, IConfiguratio
{
    app.UseExceptionHandler();

    // ← أضف هنا في الأول
    app.UseResponseCompression();

    app.UseStatusCodePages();
    // ... باقي الـ middleware
}
```

أضف `using` statement:

```
using System.IO.Compression;
```

📌 ملاحظات:

- Gzip و Brotli compression
- تقلل حجم الـ response 90-70 بـ%
- HTTPS required للـ Brotli

5. Dockerfile Update

❌ الحالة الحالية:

```
FROM mcr.microsoft.com/dotnet/sdk:10.0-preview AS build
```

❌ غير آمن للـ Production! .NET 10 Preview

✅ الحل:

استبدل الـ Dockerfile بالكامل:

```
# ===== البناء =====
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
WORKDIR /src

COPY ["src/ECommerce.API/ECommerce.API.csproj", "src/ECommerce.API/"]
COPY ["src/ECommerce.Application/ECommerce.Application.csproj", "src/ECommerce.Application/"]
COPY ["src/ECommerce.Infrastructure/ECommerce.Infrastructure.csproj", "src/ECommerce.Infrastruct
COPY ["src/ECommerce.Domain/ECommerce.Domain.csproj", "src/ECommerce.Domain/"]

RUN dotnet restore "src/ECommerce.API/ECommerce.API.csproj"

COPY . .
WORKDIR "/src/src/ECommerce.API"

RUN dotnet publish "ECommerce.API.csproj" -c Release -o /app/publish --no-restore

# ===== التشغيل =====
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS final
WORKDIR /app
COPY --from=build /app/publish .

ENV DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=false
RUN apt-get update && apt-get install -y icu-devtools && rm -rf /var/lib/apt/lists/*

ENV ASPNETCORE_HTTP_PORTS=8080
EXPOSE 8080

HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8080/health || exit 1

ENTRYPOINT ["dotnet", "ECommerce.API.dll"]
```

📌 ملاحظات:

- .NET 9 Stable (ليس Preview)
- مع HEALTHCHECK instruction
- أصغر حجم image

6. Redis SSL/TLS

❌ الحالة الحالية:

```
"Redis": "localhost:6379"  // ❌ plain text connection
```

✅ الحل:

في **appsettings.Production.json** :

```
{
  "ConnectionStrings": {
    "Redis": "localhost:6379,ssl=true,sslProtocols=Tls12"
  }
}
```

في **appsettings.Development.json** (للـ development تركه كما هو):

```
{
  "ConnectionStrings": {
    "Redis": "localhost:6379"
  }
}
```

📌 ملاحظات:

- Production فقط يحتاج SSL
- Azure Redis / AWS ElastiCache يدعمون SSL افتراضياً
- تأكد من certificate valid

7. SQL Server SSL/TLS

❌ الحالة الحالية:

```
TrustServerCertificate=True  // ❌ خطير!
```

✅ الحل:

في `appsettings.Production.json` :

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=your-server;Database=ECommerceDb;User Id=sa;Password=your-passw
  }
}
```

في `docker-compose.yml` (للـ development فقط):

```
services:
  sqlserver:
    environment:
      - ACCEPT_EULA=Y
      - MSSQL_SA_PASSWORD=YourStrong@Password1
```

📌 ملاحظات:

- `Encrypt=True` إجباري
- `TrustServerCertificate=False` للـ production مع valid certificate
- Development يمكن يكون `TrustServerCertificate=True`

8. Global Exception Middleware

❌ الحالة الحالية:

```
app.UseExceptionHandler();  // ← معلومات ناقصة في الـ response
```

✅ الحل:

أنشئ ملف: `src/ECommerce.API/Middleware/GlobalExceptionHandler.cs`

```
using Microsoft.AspNetCore.Diagnostics;

namespace ECommerce.API.Middleware;

public class GlobalExceptionHandler : IExceptionHandler
{
    private readonly ILogger<GlobalExceptionHandler> _logger;

    public GlobalExceptionHandler(ILogger<GlobalExceptionHandler> logger)
    {
        _logger = logger;
    }

    public async ValueTask<bool> TryHandleAsync(
        HttpContext httpContext,
        Exception exception,
        CancellationToken cancellationToken)
    {
        _logger.LogError(exception, "Unhandled exception occurred: {Message}", exception.Message

        var problemDetails = new ProblemDetails
        {
            Status = StatusCodes.Status500InternalServerError,
            Title = "An unhandled exception occurred",
            Detail = exception.Message,
        };

        httpContext.Response.StatusCode = StatusCodes.Status500InternalServerError;
        httpContext.Response.ContentType = "application/problem+json";

        await httpContext.Response.WriteAsJsonAsync(problemDetails, cancellationToken);

        return true;
    }
}
```

في `src/ECommerce.API/DependencyInjection.cs` :

```
public static IServiceCollection AddPresentation(this IServiceCollection services, IConfiguratio
{
    services.AddHttpContextAccessor();

    // ← أضف هنا
    services.AddExceptionHandler<GlobalExceptionHandler>();

    services.AddResponseCompression(options => { /* ... */ });
    // ... باقي الكود
}
```

📌 ملاحظات:

- تسجيل جميع الأخطاء (logging)
- إرجاع ProblemDetails format
- مخفي معلومات حساسة في Production

9. Logging Configuration

❌ الحالة الحالية:

- Serilog في appsettings.json نفسه مع Seq
- Seq configured locally (خطير!)

✅ الحل:

في `appsettings.json` (Development):

```
{
  "Serilog": {
    "MinimumLevel": "Information",
    "WriteTo": [
      {
        "Name": "Console"
      },
      {
        "Name": "Seq",
        "Args": {
          "serverUrl": "http://localhost:5341"
        }
      }
    ]
  }
}
```

في `appsettings.Production.json` (Production):

```
{
  "Serilog": {
    "MinimumLevel": "Warning",
    "WriteTo": [
      {
        "Name": "Console"
      },
      {
        "Name": "ApplicationInsights",
        "Args": {
          "connectionString": ""
        }
      }
    ]
  }
}
```

📌 ملاحظات:

- Development: Information + Seq
- Production: Warning + Application Insights
- حط Application Insights connection string من Azure

- لو استخدمت Splunk/ELK غيّر الـ sink

10. CORS Security

❌ الحالة الحالية:

```
"AllowedOrigins": [ "https://localhost:7001", "http://localhost:5001" ]
```

❌ localhost في production!

✅ الحل:

في `appsettings.json` (Development):

```
{
  "AppSettings": {
    "CorsPolicyName": "ECommercePolicy",
    "AllowedOrigins": [
      "https://localhost:7001",
      "http://localhost:5001",
      "http://localhost:3000"
    ]
  }
}
```

في `appsettings.Production.json` (Production):

```
{
  "AppSettings": {
    "CorsPolicyName": "ECommercePolicy",
    "AllowedOrigins": [
      "https://yourdomain.com",
      "https://www.yourdomain.com",
      "https://admin.yourdomain.com"
    ]
  }
}
```

في `src/ECommerce.API/DependencyInjection.cs` - تحديث الـ CORS:

```
services.AddCors(options =>
{
    var allowedOrigins = configuration.GetSection("AppSettings:AllowedOrigins").Get<string[]>();

    options.AddPolicy(configuration["AppSettings:CorsPolicyName"] ?? "ECommercePolicy",
        policy => policy
            .WithOrigins(allowedOrigins ?? Array.Empty<string>())
            .AllowAnyMethod()
            .AllowAnyHeader()
            .AllowCredentials()
            .WithExposedHeaders("X-Pagination"));
});
```

📌 ملاحظات:

- Development: localhost + local ports
- Production: real domains فقط
- AllowCredentials = true لـ cookies/auth
- ExposedHeaders للـ custom headers

🚀 خطوات التطبيق بالترتيب:

1. **أنشئ appsettings.Production.json** ✓ (5 دقائق)
2. **حدّث Dockerfile** ✓ (5 دقائق)
3. **أضف Rate Limiting** ✓ (15 دقيقة)
4. **أضف Health Checks** ✓ (15 دقيقة)
5. **أضف Response Compression** ✓ (10 دقائق)
6. **أضف Global Exception Handler** ✓ (20 دقيقة)
7. **حدّث CORS** ✓ (10 دقائق)
8. **حدّث Redis & SQL Connection Strings** ✓ (10 دقائق)
9. **اختبر الكل locally** ✓ (30 دقيقة)
10. **Build Docker image** (15 دقيقة) ✓ **وشغّله**

الوقت الكلي: ~2 ساعة ونصف

آخر تحديث: January 21, 2026