



**softITo BACKEND DEVELOPMENT
BİTİRME PROJESİ RAPORU**

BEHZATELIER STOK, SATIN ALMA VE SATIŞ YÖNETİM SİSTEMİ

ASP.NET Core 8 Web API ve MVC

Behzat Çalışkan

Temmuz 2026
softITo Yazılım Bilişim Akademisi, İstanbul

İÇİNDEKİLER

1. GİRİŞ
 - 1.1. Projenin Amacı ve Problem Tanımı
 - 1.2. Proje Kapsamı ve Kısıtları
2. MATERYAL VE YÖNTEM
 - 2.1. Kullanılan Teknolojiler
 - 2.2. Katmanlı Mimari ve Çözüm Yapısı
 - 2.3. Veritabanı Tasarımı
3. UYGULAMA GELİŞTİRME
 - 3.1. Entity Framework, Repository ve UnitOfWork
 - 3.2. Dapper, Raporlama ve Önbellekleme
 - 3.3. Identity, JWT, Swagger ve Loglama
4. KULLANICI ARAYÜZLERİ
 - 4.1. Mağaza Ana Sayfası ve Ürün Kataloğu
 - 4.2. Sepet ve Satış Akışı
 - 4.3. Yönetim Paneli
5. İŞ SÜREÇLERİ VE RAPORLAMA
 - 5.1. Stok, Satın Alma ve Satış
 - 5.2. PDF/Excel, Chatbot ve Audit Log
6. TEST, GÜVENLİK VE GELİŞTİRME ÖNERİLERİ
7. SONUÇ
8. KAYNAKÇA

1. GİRİŞ

1.1. Projenin Amacı ve Problem Tanımı

Behzatelier, çelik takı satışı yapan bir işletmenin ürün, varyant, stok, tedarik, satın alma ve satış süreçlerini tek bir sistem üzerinden yönetebilmesi amacıyla geliştirilmiştir. Proje yalnızca bir ürün kataloğu değil; son kullanıcı mağazası, güvenli yönetim paneli, RESTful API, raporlama altyapısı ve işlem kayıtlarını bir araya getiren bütünlük bir iş uygulamasıdır.

Takı sektöründe aynı ürünün renk, kaplama, taş ve ölçü gibi farklı varyantları bulunduğundan stok takibinin yalnızca ürün bazında yapılması yetersiz kalmaktadır. Bu nedenle stok miktarı, kritik stok seviyesi, alış ve satış fiyatları **ProductVariant** düzeyinde tutulmuştur.

1.2. Proje Hedefleri

- Kategori, ürün ve varyantların yönetilmesi.
- Satın alma girişlerinde stokların otomatik artırılması.
- Satış ve mağaza ödeme işlemlerinde stokların otomatik azaltılması.
- Tedarikçi, sipariş ve stok hareketlerinin izlenmesi.
- Dapper ile hızlı rapor sorgularının hazırlanması.
- PDF ve Excel çıktılarının üretilebilmesi.
- Identity ve JWT ile rol bazlı güvenlik sağlanması.
- İşlem geçmişinin AuditLog kayıtlarıyla takip edilmesi.

1.3. Proje Kapsamı

Sistem iki temel kullanıcı deneyimi sunmaktadır. Mağaza tarafında ziyaretçiler ürünleri kategori ve arama kriterlerine göre görüntüleyebilir, ürün detaylarına erişebilir, sepete ürün ekleyebilir ve ödeme sürecini tamamlayabilir. Yönetim tarafında ise kategori, ürün, varyant, tedarikçi, stok, satın alma, satış ve rapor işlemleri yönetilir.

TEMEL MODÜLLER

Mağaza: Ana sayfa, kategori vitrinleri, ürün kataloğu, ürün detay, sepet ve ödeme.

Yönetim: Dashboard, CRUD ekranları, stok hareketleri, satın alma, satış siparişleri, raporlar, chatbot ve işlem kayıtları.

API: Kimlik doğrulama, mağaza servisleri, yönetim servisleri, rapor ve dış aktarma uç noktaları.

1.4. Kısıtlar

Proje geliştirme ortamında SQL Server LocalDB bağlantısı kullanılmaktadır. Yönetici girişindeki telefon doğrulama alanı demo kod ile çalışacak biçimde hazırlanmıştır; gerçek SMS gönderimi için Twilio, Netgsm veya benzeri ücretli bir servis sağlayıcısının API bilgileri gereklidir.

İnceleme notu: Bu rapor proje kaynak kodları, Razor görünümüleri, veritabanı modeli ve yapılandırma dosyaları incelenerek hazırlanmıştır. Kullanılan çalışma ortamında .NET SDK ve SQL Server LocalDB bulunmadığından canlı derleme ve veritabanı bağlantı testi yapılmamıştır.

2. MATERYAL VE YÖNTEM

2.1. Kullanılan Teknolojiler

.NET 8 / ASP.NET Core

API ve MVC web uygulamalarının temel çalışma platformudur.

Entity Framework Core

İşlem tabanlı CRUD ve ilişki yönetiminde kullanılmıştır.

Dapper

Dashboard ve rapor sorgularında doğrudan SQL erişimi sağlar.

SQL Server

İş verileri ve ASP.NET Core Identity tablolarını tutar.

Identity + JWT

Admin/User rolleri ve token tabanlı API güvenliği sağlar.

Razor MVC

Mağaza ve yönetim paneli arayüzlerini oluşturur.

ClosedXML

Dashboard ve kritik stok Excel raporlarını üretir.

PdfSharpCore

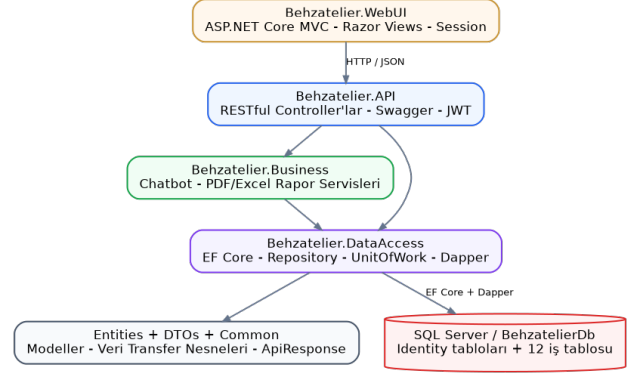
PDF rapor dosyalarının hazırlanmasını sağlar.

Serilog

İstek ve uygulama loglarını günlük dosyalara yazar.

MemoryCache

Sık kullanılan rapor sonuçlarının tekrar sorgulanmasını azaltır.



Şekil 2.1. Behzatelier katmanlı mimari ve veri akışı

API iletişimi: WebUI içinde tanımlanan HttpClient, *ApiSettings:BaseUrl* üzerinden API uç noktalarına istek gönderir. Başarılı giriş sonrasında JWT token ve kullanıcı bilgileri Session içinde saklanarak korumalı yönetim isteklerinde kullanılır.

RESTful API

Katmanlı Mimari

DTO

Dependency Injection

Async/Await

2.2. Mimari Yaklaşım

Proje, sorumlulukların ayrılması ilkesine uygun olarak bağımsız projelerden oluşan katmanlı mimari kullanır. WebUI doğrudan veritabanına erişmez; HTTP istemcisi üzerinden API ile haberleşir. API ise iş servisleri ve veri erişim katmanlarını kullanır.

2.2. KATMANLI MİMARİ VE ÇÖZÜM YAPISI

```
Behzatelier.sln
├── Behzatelier.API
│   ├── Controllers / Services / Logs
│   └── Program.cs (JWT, Swagger, CORS, Serilog)
├── Behzatelier.Business
│   ├── ChatbotServices
│   └── ExportServices (PDF / Excel)
├── Behzatelier.DataAccess
│   ├── Context / Migrations
│   ├── Repositories / UnitOfWork
│   └── DapperReports
├── Behzatelier.Entities
│   └── 12 iş tablosuna ait modeller
├── Behzatelier.DTOs
│   └── İstek / yanıt modelleri
├── Behzatelier.Common
│   └── ApiResponse<T>
├── Behzatelier.WebUI
│   ├── Controllers / Views / ViewModels
│   └── Mağaza ve yönetim paneli arayüzü
```

Şekil 2.2. Visual Studio çözüm yapısı

Behzatelier.API

Controller uç noktaları, JWT doğrulaması, Swagger, CORS, Serilog, MemoryCache ve servis kayıtlarını içerir. Store API dışındaki yönetim servisleri ağırlıklı olarak **[Authorize]** ile korunmuştur.

Behzatelier.Business

Rapor çıktıları ve kural tabanlı chatbot servisleri bu katmanda yer alır. ClosedXML ve PdfSharpCore bağımlılıkları burada kullanılmıştır.

Behzatelier.DataAccess

ApplicationDbContext, migration dosyaları, Generic Repository, UnitOfWork ve Dapper rapor deposu bu katmandadır.

Behzatelier.Entities ve DTOs

Veritabanı varlıkları ile API giriş/çıkış modelleri ayrılmıştır. Böylece kullanıcıya doğrudan entity döndürülmesi önlenmiş, veri sözleşmeleri daha kontrollü hale getirilmiştir.

Behzatelier.Common

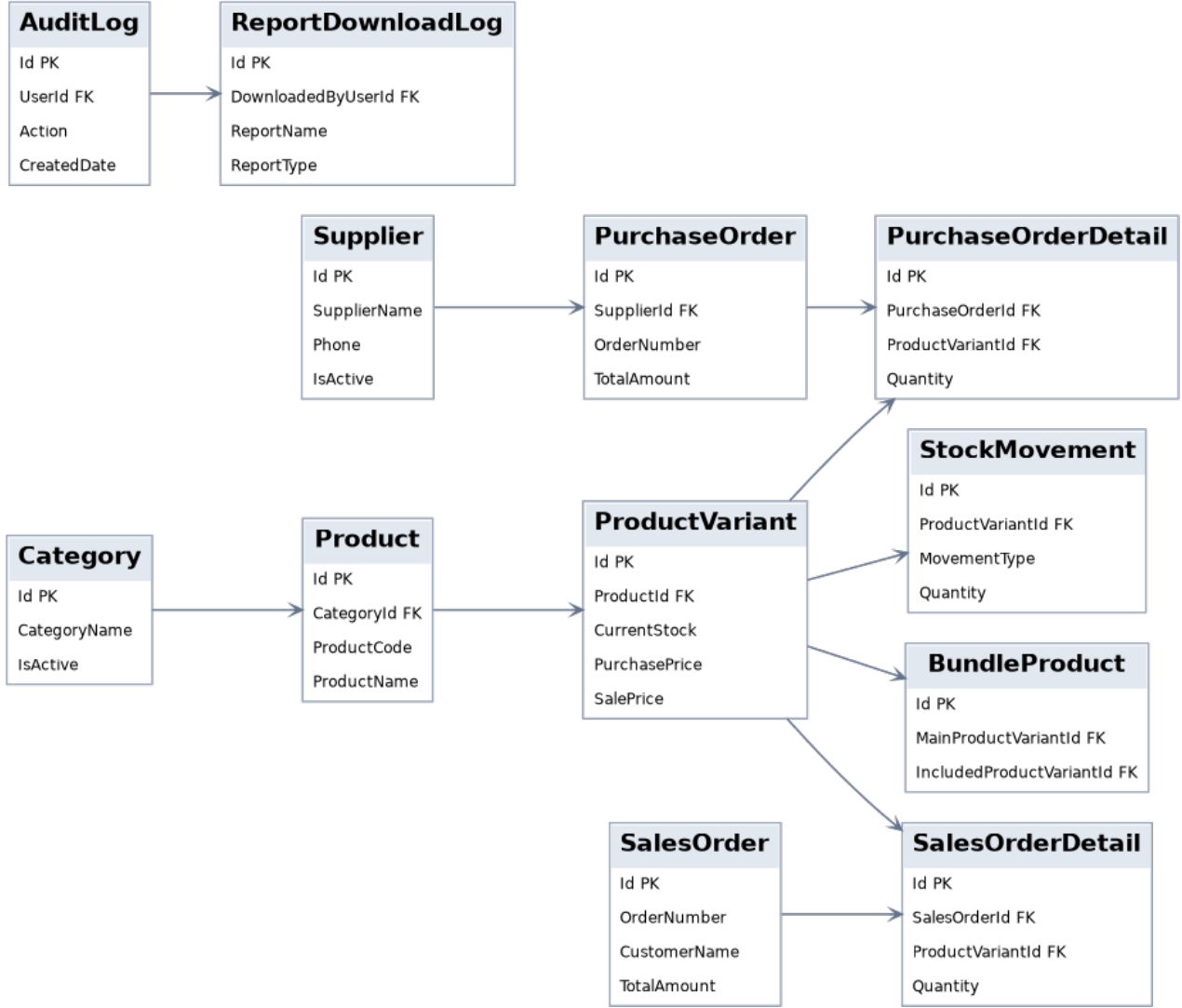
Tüm uç noktalarda tutarlı cevap formatı sağlayan **ApiResponse<T>** sınıfını içerir.

Behzatelier.WebUI

Mağaza ve yönetim panelinin controller, view, view model ve statik görsellerini içerir. Kullanıcı ve yönetici için farklı layout tasarımları kullanılmıştır.

Bağımlılık yönü: Üst katmanlar alt katmanlara referans verir; veri erişim ayrıntıları kullanıcı arayüzünden ayrılır. Bu yapı bakım, test ve gelecekte yeni istemci uygulamalarının eklenmesini kolaylaştırır.

2.3. VERİTABANI TASARIMI



Şekil 2.3. Behzatelier temel iş tabloları ve ilişkileri

Ürün ve Stok Yapısı

Category ürün gruplarını, **Product** temel ürün bilgilerini, **ProductVariant** ise renk, kaplama, taş, ölçü, barkod, alış/satış fiyatı ve stok seviyesini tutar. Kritik stok kontrolü varyant seviyesinde yapılır.

StockMovement satın alma girişi, satış çıkışı ve manuel hareketleri tarih, miktar ve açıklama bilgisiyle kaydeder.

Sipariş ve İzlenebilirlik

PurchaseOrder ve **PurchaseOrderDetail** tedarikçi satın alma kayıtlarını; **SalesOrder** ve **SalesOrderDetail** müşteri satışlarını tutar. **AuditLog** kullanıcı, IP, işlem ve açıklama verilerini; **ReportDownloadLog** rapor indirme geçmişini saklamak için tasarlanmıştır.

3. UYGULAMA GELİŞTİRME

3.1. Entity Framework Core

ApplicationDbContext, ASP.NET Core Identity için **IdentityDbContext<AppUser>** sınıfından türetilmiştir. İş tabloları DbSet olarak tanımlanmış; decimal alanların hassasiyeti ve ilişkilerin silme davranışları Fluent API üzerinden yapılandırılmıştır.

Ürün-kategori, ürün-varyant ve sipariş-detay ilişkilerinde **DeleteBehavior.Restrict** kullanılarak bağlı iş verilerinin yanlışlıkla silinmesi engellenmiştir. Sipariş detayları ise ana sipariş silindiğinde **Cascade** davranışıyla temizlenmektedir.

3.1.1. Generic Repository

GetAll, GetById, Find, Add, Update, Delete ve Query işlemleri ortak bir GenericRepository sınıfında toplanmıştır. Controller'ların doğrudan DbContext bağımlılığı azaltılmıştır.

3.1.2. UnitOfWork

Tüm repository nesneleri tek bir DbContext örneğini paylaşır. Satış ve satın alma gibi birden fazla tabloyu etkileyen işlemler, tek SaveAsync çağrısıyla kaydedilir.

Satış oluşturulurken SalesOrder, SalesOrderDetail, ProductVariant ve StockMovement kayıtları aynı iş akışında güncellenir. Böylece stok ile sipariş bilgilerinin uyumlu tutulması amaçlanır.

```
1 public class UnitOfWork : IUnitOfWork
2 {
3     private readonly ApplicationDbContext _context;
4
5     public IGenericRepository<Category> Categories { get; }
6     public IGenericRepository<Product> Products { get; }
7     public IGenericRepository<ProductVariant> ProductVariants { get; }
8     public IGenericRepository<Supplier> Suppliers { get; }
9     public IGenericRepository<StockMovement> StockMovements { get; }
10    public IGenericRepository<PurchaseOrder> PurchaseOrders { get; }
11    public IGenericRepository<SalesOrder> SalesOrders { get; }
12
13    public UnitOfWork(ApplicationDbContext context)
14    {
15        _context = context;
16        Categories = new GenericRepository<Category>(_context);
17        Products = new GenericRepository<Product>(_context);
18        ProductVariants = new GenericRepository<ProductVariant>(_context);
19        Suppliers = new GenericRepository<Supplier>(_context);
20        StockMovements = new GenericRepository<StockMovement>(_context);
21        PurchaseOrders = new GenericRepository<PurchaseOrder>(_context);
22        SalesOrders = new GenericRepository<SalesOrder>(_context);
23    }
24
25    public Task<int> SaveAsync() => _context.SaveChangesAsync();
26 }
```

Şekil 3.1. UnitOfWork sınıfının temel yapısı

```
1 builder.Services.AddMemoryCache();
2 builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
3 builder.Services.AddScoped<IDapperReportRepository>(sp =>
4 {
5     var connection = builder.Configuration
6     .GetConnectionString("DefaultConnection");
7     return new DapperReportRepository(connection!);
8 });
9 builder.Services.AddScoped<IReportExportService, ReportExportService>();
10 builder.Services.AddScoped<IChatbotService, ChatbotService>();
11
12 builder.Services.AddDbContext<ApplicationDbContext>(options =>
13 options.UseSqlServer(builder.Configuration
14 .GetConnectionString("DefaultConnection")));
15
16 builder.Services.AddIdentity<AppUser, IdentityRole>()
17 .AddEntityFrameworkStores<ApplicationDbContext>()
18 .AddDefaultTokenProviders();
```

Şekil 3.2. Program.cs servis ve veritabanı kayıtları

3.2. DAPPER, RAPORLAMA VE ÖNBELLEKLEME

3.2.1. Dapper Rapor Deposu

Rapor ekranlarında çok sayıda tablo üzerinden toplam, gruplama ve join işlemleri yapılmaktadır. Bu sorgular için Dapper kullanılarak SQL ifadeleri doğrudan kontrol edilmiştir. Dashboard özeti tek sorgu içinde ürün, varyant, toplam stok, kritik stok, satın alma, satış, tahmini kâr, tedarikçi ve sipariş sayılarını üretir.

Rapor Başlıkları

- Dashboard özeti ve finansal göstergeler
- Kritik stoktaki ilk 10 varyant
- En çok satan ürünler
- Aylık satış raporu
- Tedarikçi satın alma özeti
- Kategori ürün ve varyant sayıları
- Son satış siparişleri
- Stok hareket ve stok değer raporları

3.2.2. MemoryCache

Dashboard, kritik stok, en çok satanlar ve aylık satış uç noktalarında sonuçlar bellek önbelleğine alınır. Kayıtlar için 5 dakikalık mutlak ve 2 dakikalık kayan süre kullanılmıştır. Böylece aynı raporun kısa aralıklarla tekrar istenmesi halinde veritabanı yükü azaltılır.

```
1 SELECT
2 (SELECT COUNT(*) FROM Products WHERE IsActive = 1) AS TotalProductCount,
3 (SELECT ISNULL(SUM(CurrentStock), 0) FROM ProductVariants
4 WHERE IsActive = 1) AS TotalStockQuantity,
5 (SELECT ISNULL(SUM(TotalAmount), 0) FROM SalesOrders) AS TotalSalesAmount,
6 (SELECT ISNULL(SUM((sod.UnitPrice - pv.PurchasePrice) * sod.Quantity), 0)
7 FROM SalesOrderDetails sod
8 INNER JOIN ProductVariants pv ON sod.ProductVariantId = pv.Id) AS EstimatedProfit;
9
10 if (!_memoryCache.TryGetValue("dashboard_summary", out DashboardSummaryDto? result))
11 {
12     result = await dapperReportRepository.GetDashboardSummaryAsync();
13     _memoryCache.Set("dashboard_summary", result,
14         new MemoryCacheEntryOptions {
15             AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5),
16             SlidingExpiration = TimeSpan.FromMinutes(2)
17         });
18 }
```

Şekil 3.3. Dapper dashboard sorgusu ve rapor önbellekleme örneği

EF Core ve Dapper birlikte kullanım amacı: EF Core, veri ekleme-güncelleme ve ilişkili nesne yönetiminde geliştirme hızını artırırken; Dapper raporların SQL düzeyinde açık ve performans odaklı hazırlanmasına olanak sağlar.

3.3. KİMLİK DOĞRULAMA, API GÜVENLİĞİ VE LOGLAMA

3.3.1. ASP.NET Core Identity

AppUser sınıfı IdentityUser'dan türetilmiş; FullName, IsActive ve CreatedDate alanları eklenmiştir. Uygulama başlatıldığında Admin ve User rolleri kontrol edilerek eksik roller oluşturulur. İlk kullanıcı veya belirlenen yönetici hesabı Admin rolüne atanır.

3.3.2. JWT

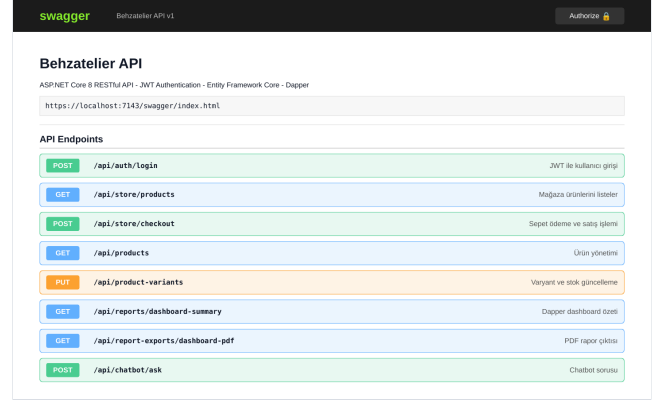
Başarılı girişten sonra kullanıcı kimliği, ad-soyad, e-posta ve rol claim'lerini içeren HMAC-SHA256 imzalı JWT üretilir. Token süresi yapılandırma dosyasında 60 dakika olarak tanımlanmıştır.

```
1 var claims = new List<Claim>
2 {
3     new Claim(ClaimTypes.NameIdentifier, user.Id),
4     new Claim(ClaimTypes.Name, user.FullName),
5     new Claim(ClaimTypes.Email, user.Email ?? ""),
6     new Claim(ClaimTypes.Role, role)
7 };
8
9 var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey!));
10 var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
11
12 var token = new JwtSecurityToken(
13     issuer: issuer,
14     audience: audience,
15     claims: claims,
16     expires: DateTime.Now.AddMinutes(expireMinutes),
17     signingCredentials: credentials);
18
19 return new JwtSecurityTokenHandler().WriteToken(token);
```

Şekil 3.4. JWT üretim kodu

3.3.3. Swagger ve CORS

Swagger arayüzüne Bearer güvenlik şeması eklenmiş, böylece korumalı uç noktalar token kullanılarak test edilebilir. CORS politikası geliştirme aşamasında tüm kaynaklara izin verecek biçimde tanımlanmıştır.



Şekil 3.5. Behzatelier API uç noktalarının Swagger görünümü

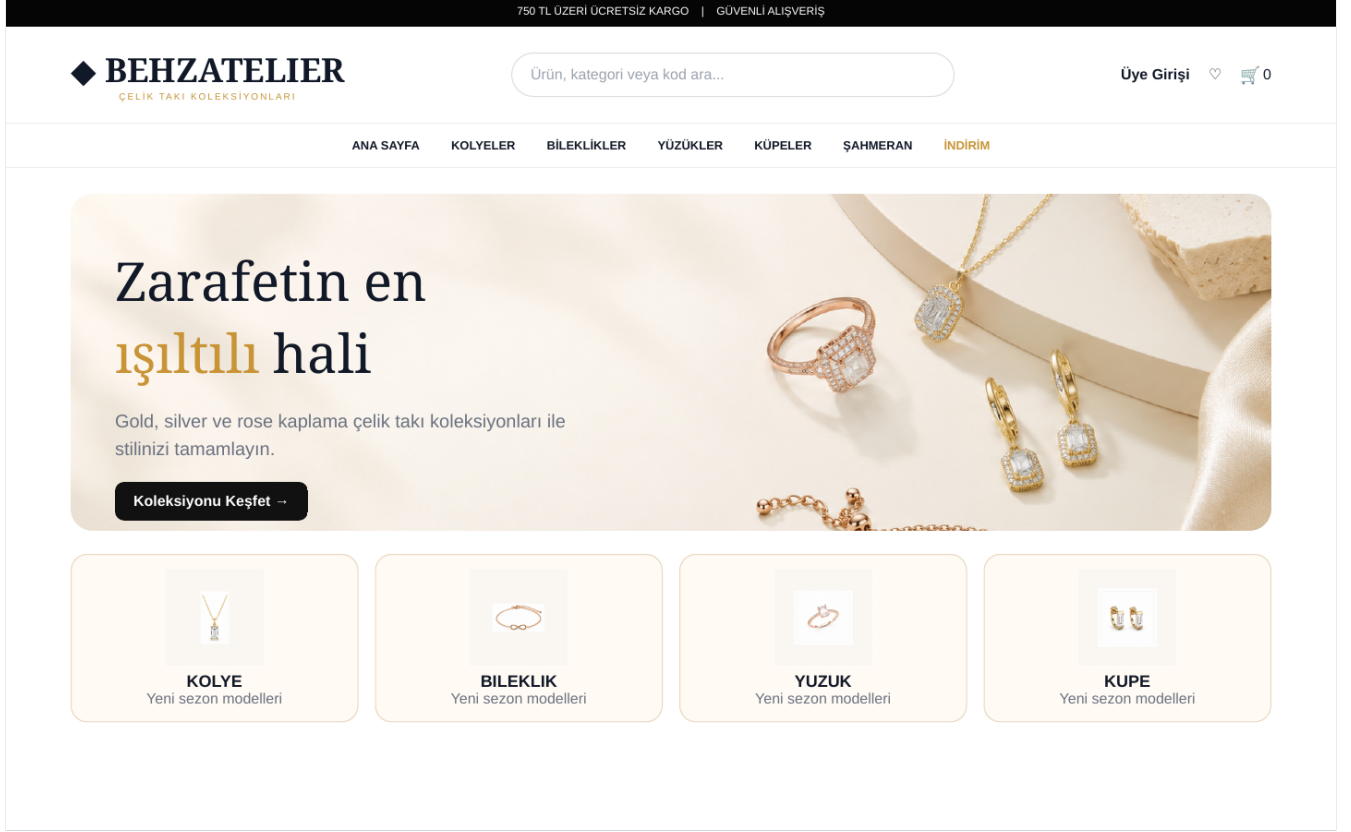
3.3.4. Serilog ve AuditLog

Serilog istek günlüklerini ve uygulama loglarını tarih bazlı dosyalara yazar. AuditLogService ise iş işlemlerini kullanıcı kimliği, IP adresi, tablo adı, işlem türü ve açıklama ile veritabanına kaydeder.

4. KULLANICI ARAYÜZLERİ

4.1. Mağaza Ana Sayfası

Son kullanıcı arayüzünde altın tonları, açık arka plan ve ürün odaklı kart tasarımı tercih edilmiştir. Ana sayfada geniş hero alanı, kategori kartları, yeni ürünler, güvenli alışveriş bilgileri ve kullanıcı/sepet bağlantıları bulunmaktadır. Ürün görselleri ProductCode değerine göre eşleştirilerek aynı görselin farklı ürünlerde tekrar edilmesi azaltılmıştır.



Şekil 4.1. Behzatelier mağaza ana sayfası tasarımı

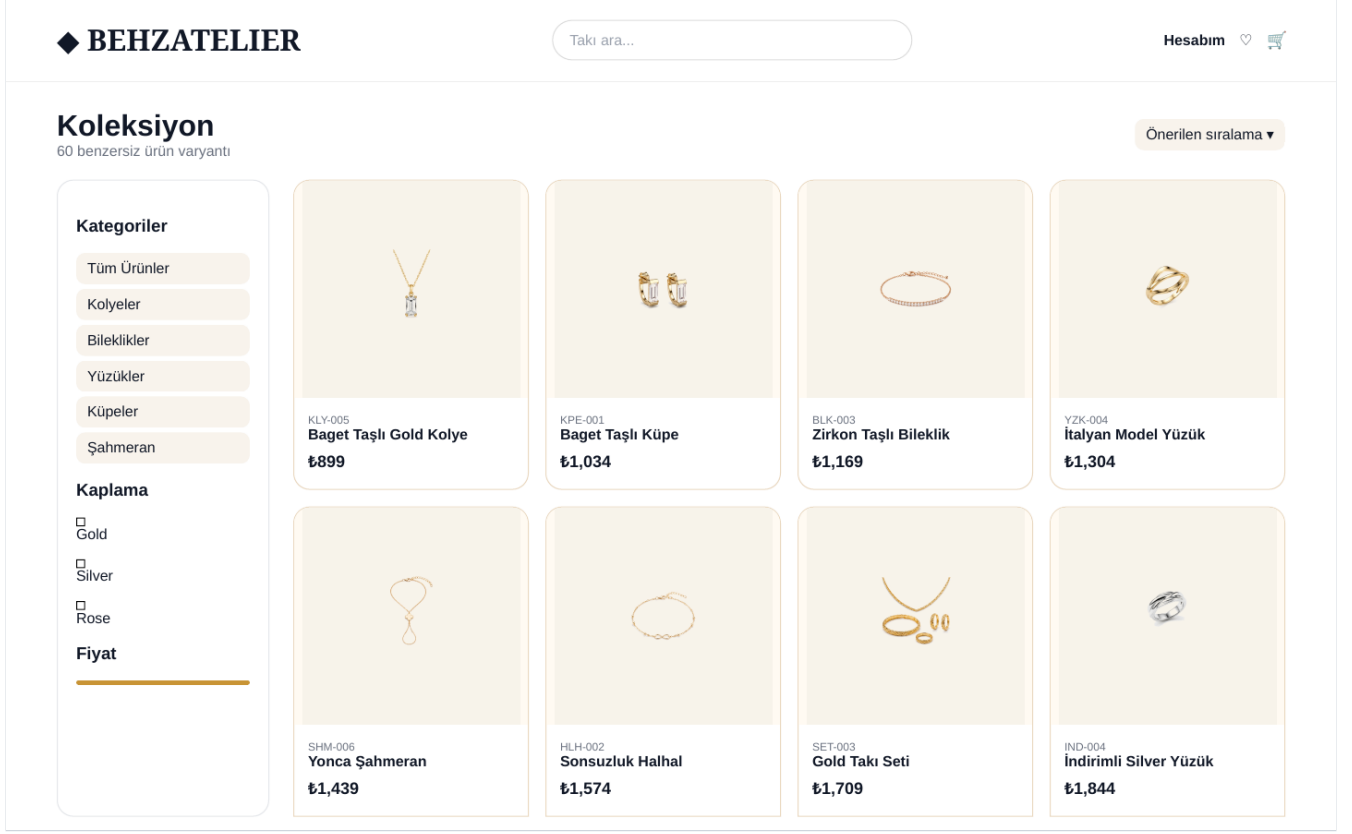
Kategori Yapısı

Kolye, bileklik, yüzük, küpe, şahmeran, halhal, bilezik, aksesuar, set ve indirim kategorileri kullanıcıya ayrı giriş noktaları sunar.

Görsel Kalite Kontrolü

Projede 1200x1200 ürün görselleri, kategori görselleri, hero banner ve fallback görseli yer almaktadır. Ürün kartlarında hover büyütme ve dengeli object-fit kullanımı uygulanmıştır.

4.2. ÜRÜN KATALOĞU, SEPET VE ÖDEME AKIŞI



Şekil 4.2. Kategori ve ürün koduna göre oluşturulan mağaza kataloğu

Ürün Listeleme

StoreApiController, aktif ürün ve varyantları Product, Category ve ProductVariant tablolarıyla birlikte getirir. Kategori ve arama parametreleri URL üzerinden alınır. WebUI tarafında ürün kodu, kategori öneki, isim ve görsel kontrolleriyle tekrar eden veya görseli bulunmayan kayıtlar filtrelenebilir.

Sepet Yönetimi

Sepet öğeleri Session üzerinde tutulur. Kullanıcı adetleri güncelleyebilir, ürün silebilir ve ödeme ekranına geçebilir.

Ödeme ve Stok

Checkout isteği Store API'ye gönderilir. Her ürün için stok yeterliliği kontrol edilir; satış siparişi ve detayları oluşturulur, varyant stokları azaltılır ve stok hareketi kaydedilir. İşlem sonunda sepet temizlenir.

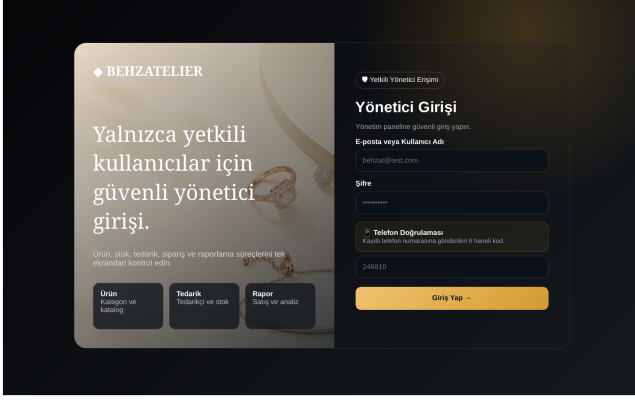


Şekil 4.3. Mağaza satış ve stok güncelleme akışı

4.3. YÖNETİM PANELİ

4.3.1. Yönetici Girişi

Yönetici giriş ekranı mağaza arayüzünden ayrılmış, koyu tema ve yetkili erişim vurgusuyla tasarlanmıştır. E-posta/kullanıcı adı, şifre ve telefon doğrulama kodu alanları bulunur.



Şekil 4.4. Yönetici giriş ekranı

4.3.2. Dashboard

Dashboard toplam sipariş, ciro, ürün ve kritik stok kartlarını; aylık satış grafiğini, kategori dağılımını, kritik stokları ve en çok satan ürünleri gösterir. Veriler rapor API'lerinden alınır.



Şekil 4.5. Yönetim paneli dashboard ekranı

Telefon doğrulaması: Projede mevcut ekran demo kod ile çalışır. Gerçek SMS için harici servis anahtarı, gönderici bilgisi ve yönetici telefon numarasının güvenli yapılandırılmaya eklenmesi gerekir.

5. İŞ SÜREÇLERİ VE RAPORLAMA

5.1. Stok, Satın Alma ve Satış Yönetimi

BEHZATELIER

YÖNETİM PANELİ

Yönetim Paneli
Stok ve satış yönetimi

ANA SAYFA

Dashboard

ÜRÜNLER

Kategoriler

Ürünler

Varyantlar

Stok Yönetimi

SİPARİŞLER

Satın Alma

Satış Siparişleri

Tedarikçiler

RAPORLAR

Satış Raporları

SİSTEM

Chatbot

İşlem Kayıtları

Stok ve Sipariş Yönetimi
Ürün varyantları, satın alma ve satış hareketleri

Transaction Ready

Stok Hareketleri
Giriş ve çıkış kayıtları

+ Yeni Hareket

Tarih	Ürün	Tür	Miktar	Kullanıcı
17.07.2026	Baget Taşı Kolye	Satış Çıkışı	-2	Behzat
17.07.2026	Gold Takı Seti	Satın Alma Girişi	+20	Behzat
16.07.2026	Zirkon Bileklik	Satış Çıkışı	-1	Behzat
16.07.2026	İtalyan Model Yüzük	Manuel Giriş	+8	Admin
15.07.2026	Yonca Şahmeran	Satış Çıkışı	-3	Behzat

Yeni Satış Siparişi

Müşteri
Ayşe Yılmaz

Ürün Varyantı
KLY-005 / Gold

Adet
2

Birim Fiyat
₺1.190

Toplam
₺2.380

Siparişi Kaydet

Satın Alma Siparişleri

No	Tedarikçi	Tutar	Durum
PO-2026071701	Altın Çizgi	₺24.500	Tamamlandı
PO-2026071603	Nova Takı	₺13.840	Onaylandı

Satış Siparişleri

No	Müşteri	Tutar	Durum
SO-2026071708	Ayşe Yılmaz	₺2.380	Oluşturuldu
SO-2026071707	Elif Kaya	₺1.690	Oluşturuldu

Şekil 5.1. Stok hareketleri ve sipariş yönetimi ekranı

Satın Alma

Satın alma siparişi oluşturulurken tedarikçi ve ürün varyantları doğrulanır. Her detayın miktarı ve birim fiyatı üzerinden satın ve sipariş toplamı hesaplanır. Varyant stoğu artırılır ve “Satın Alma Girişi” hareketi eklenir.

Satış

Satış siparişinde mevcut stok, istenen miktarla karşılaştırılır. Yetersiz stokta işlem durdurulur. Başarılı satışta stok azaltılır ve “Satış Çıkışı” hareketi kaydedilir.

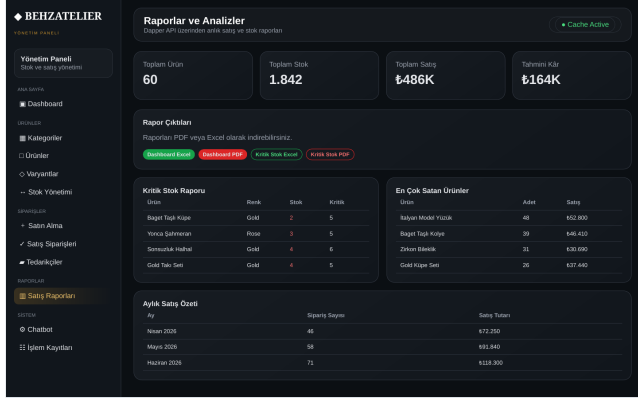
```
1 if (variant.CurrentStock < detailDto.Quantity)
2     return BadRequest(ApiResponse<object>.Fail(
3         $"Yetersiz stok. Mevcut: {variant.CurrentStock}"));
4
5 var lineTotal = detailDto.Quantity * detailDto.UnitPrice;
6 totalAmount += lineTotal;
7
8 salesOrder.SalesOrderDetails.Add(new SalesOrderDetail
9 {
10     ProductVariantId = detailDto.ProductVariantId,
11     Quantity = detailDto.Quantity,
12     UnitPrice = detailDto.UnitPrice,
13     LineTotal = lineTotal
14 });
15
16 variant.CurrentStock -= detailDto.Quantity;
17 _unitOfWork.ProductVariants.Update(variant);
18
19 await _unitOfWork.StockMovements.AddAsync(new StockMovement
20 {
21     ProductVariantId = detailDto.ProductVariantId,
22     MovementType = "Satış Çıkışı",
23     Quantity = detailDto.Quantity,
24     Description = "Satış siparişi ile stok çıkışı yapıldı."
25 });
```

Şekil 5.2. Satış siparişi sırasında stok kontrolü ve hareket kaydı

5.2. RAPOR ÇIKTILARI, CHATBOT VE İŞLEM KAYITLARI

5.2.1. PDF ve Excel

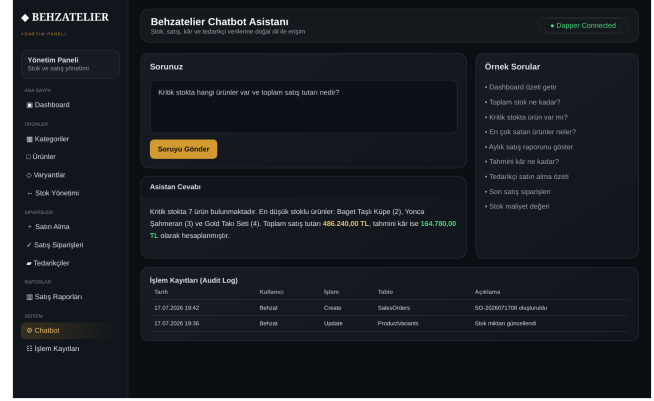
ReportExportService; dashboard ve kritik stok verilerini Dapper üzerinden alır. ClosedXML ile çalışma sayfaları, PdfSharpCore ile tablo biçiminde PDF dosyaları oluşturulur. WebUI üzerinden ayrı indirme butonları sunulur.



Şekil 5.3. Rapor ekranı ve dışa aktarma seçenekleri

5.2.2. Veri Odaklı Chatbot

ChatbotService kullanıcının mesajındaki anahtar ifadeleri analiz eder. Toplam stok, kritik stok, satış, kâr, tedarikçi, kategori, son sipariş ve stok değeri sorularında ilgili Dapper rapor metodu çağrılır ve Türkçe cevap oluşturulur. Sistem harici bir yapay zekâ servisine bağlı değildir; kural tabanlı ve veritabanı odaklı çalışır.



Şekil 5.4. Chatbot ve AuditLog ekranı

6. TEST, GÜVENLİK VE GELİŞTİRME ÖNERİLERİ

6.1. Fonksiyonel Kontrol Senaryoları

1. Kullanıcı kayıt ve giriş işlemlerinin doğru rol ve JWT üretmesi.
2. Yetkisiz isteklerin korumalı API uç noktalarına erişememesi.
3. Kategori, ürün, varyant ve tedarikçi CRUD işlemleri.
4. Satın alma sonrasında varyant stoklarının artması.
5. Satışta yetersiz stok kontrolünün çalışması ve stokların azalması.
6. StockMovement ve AuditLog kayıtlarının oluşması.
7. Dapper rapor sonuçları ile işlem tablolarının uyumlu olması.
8. PDF ve Excel dosyalarının indirilebilmesi.
9. Mağaza kategori/arama filtrelerinin doğru sonuç döndürmesi.
10. Chatbot sorularının ilgili raporlarla cevaplanması.

6.2. Güvenlik Değerlendirmesi

- JWT anahtarı kaynak kod deposunda tutulmamalı; Secret Manager veya ortam değişkeni kullanılmalıdır.
- Üretim ortamında CORS yalnızca izin verilen alan adlarıyla sınırlandırılmalıdır.
- Identity parola kuralları geliştirme ayarlarından daha güçlü hale getirilmelidir.
- Yönetici ve kullanıcı yetkileri controller veya policy seviyesinde daha ayrıntılı ayrılmalıdır.
- Giriş denemeleri için kilitleme, rate limiting ve güvenli hata mesajları uygulanmalıdır.

6.3. Teknik İyileştirme Önerileri

- Satış ve satın alma işlemlerine açık veritabanı transaction yönetimi eklenmesi.
- Rapor cache'lerinin stok veya sipariş değişimlerinde temizlenmesi.
- FluentValidation ile DTO doğrulamalarının merkezi hale getirilmesi.
- AutoMapper ile entity-DTO dönüşümlerinin sadeleştirilmesi.
- API ve servis katmanları için xUnit testleri hazırlanması.
- Gerçek SMS/e-posta doğrulama servisi entegrasyonu.
- Refresh token, şifre yenileme ve e-posta onayı akışları.
- Docker ve CI/CD ile otomatik build, test ve dağıtım.
- SQL sorguları için indeks ve execution plan analizi.
- Chatbot için niyet sınıflandırma veya kontrollü LLM entegrasyonu.

Kaynak Kod İncelemesinde Görülen Güçlü Yönler

Katmanlı çözüm yapısı, EF Core ve Dapper'ın amaçlarına uygun birlikte kullanımı, stok hareketlerinin siparişlerle ilişkilendirilmesi, rapor dışı aktarma, loglama, JWT, Swagger ve kullanıcı mağazasının aynı çözümde bir araya getirilmesi projenin kapsamını güçlendirmektedir.

Doğrulama sınırı: Canlı build ve SQL bağlantısı bu ortamda çalıştırılmadığı için rapordaki fonksiyonel değerlendirmeler kaynak kod akışı ve yapılandırmalar üzerinden yapılmıştır.

7. SONUÇ

Behzatelier projesi, bir çelik takı işletmesinin e-ticaret ve operasyon ihtiyaçlarını aynı sistemde birleştiren kapsamlı bir bitirme projesidir. Kullanıcı mağazası ürünlerin profesyonel bir vitrinle sunulmasını sağlarken yönetim paneli stok, satın alma, satış ve raporlama süreçlerini merkezileştirir.

Katmanlı mimari sayesinde kullanıcı arayüzü, API, iş servisleri, veri erişim ve model katmanlarının sorumlulukları ayrılmıştır. Entity Framework Core ile işlem ve ilişki yönetimi; Dapper ile raporlama; Identity ve JWT ile güvenlik; MemoryCache ile performans; Serilog ve AuditLog ile izlenebilirlik sağlanmıştır.

Satış ve satın alma işlemlerinin doğrudan stok hareketleriyle bağlanması projenin temel iş değeridir. PDF/Excel raporları ve veritabanı odaklı chatbot ise sistemin yalnızca CRUD uygulaması olmasının ötesine geçmesini sağlamaktadır.

Gerçek SMS, daha güçlü güvenlik politikaları, otomatik testler, transaction iyileştirmeleri ve canlı ortam dağıtımı tamamlandığında proje üretim ortamına daha yakın, ölçeklenebilir bir yapıya ulaşacaktır.

Proje sahibi: Behzat Çalışkan

Proje: Behzatelier Stock Management System

Teknoloji: .NET 8, ASP.NET Core Web API, MVC, EF Core, Dapper ve SQL Server

8. KAYNAKÇA

1. Microsoft Learn - ASP.NET Core 8 Web API ve MVC dokümantasyonu.
2. Microsoft Learn - Entity Framework Core ve SQL Server sağlayıcısı.
3. Microsoft Learn - ASP.NET Core Identity ve JWT Bearer Authentication.
4. Dapper resmi GitHub deposu ve kullanım dokümantasyonu.
5. ClosedXML dokümantasyonu - Excel dosyası oluşturma.
6. PdfSharpCore dokümantasyonu - PDF üretimi.
7. Serilog ASP.NET Core ve File Sink dokümantasyonu.
8. Swagger / OpenAPI ve Swashbuckle ASP.NET Core dokümantasyonu.
9. Bootstrap 5 ve Razor View dokümantasyonu.
10. Behzatelier proje kaynak kodları ve sürüm düzeltme notları, Temmuz 2026.

Raporun görsel düzeni, kullanıcı tarafından sağlanan bitirme projesi rapor örneğinin akademik sayfa yapısı referans alınarak Behzatelier projesine özgü biçimde hazırlanmıştır.