



SoftITo 4. DÖNEM 1. GRUP

BACKEND DEVELOPER BİTİRME PROJESİ RAPORU

Dealhawk - Oyun Fiyat Takip Ve Fırsat İzleme Platformu

Mahmut Bora CİHANGİR

Temmuz,2026

softITo Yazılım Bilişim Akademisi, İstanbul

İÇİNDEKİLER

İÇİNDEKİLER	2
1. Giriş.....	3
1.1. Projenin Amacı.....	3
1.2. Proje Kısıtları	3
2. Materyal ve Yöntem.....	4
2.1. Kullanılan Teknolojiler ve Araçlar	4
2.2. Yazılım Mimarisi (Onion Architecture)	6
2.2.1. Domain (Çekirdek/Varlık Katmanı - DealHawk.Domain)	6
2.2.2. Application (Uygulama/İş Mantığı Katmanı - DealHawk.Application)	6
2.2.3. Persistence (Veri Erişim Katmanı - DealHawk.Persistence)	7
2.2.4. Infrastructure (Altyapı/Dış Servis Katmanı - DealHawk.Infrastructure)	7
2.2.5. API ve WebMVC (Sunum Katmanları - DealHawk.API & DealHawk.WebMVC)	7
2.3.1. Temel Tablolar ve Veritabanı Varlıkları	8
2.4.1. CheapShark REST API Entegrasyonu	9
2.4.2. Hangfire ile Zamanlanmış Görev Yönetimi (PriceSyncJob).....	9
3. KURULUM VE UYGULAMA ADIMLARI	10
3.1. Ön Gereksinimler.....	10
3.2. Veritabanı Kurulumu ve Migrasyonların Uygulanması	10
3.3. API Servisinin ve Arayüzün Çalıştırılması.....	11
3.4. Hangfire Arayüzü ile Görevlerin İzlenmesi.....	12
3.5. Roller ve Yetkilendirme Senaryoları	13
4. SONUÇ VE GELECEK ÇALIŞMALAR.....	15
5. KAYNAKÇA	16

1. Giriş

Günümüz dijital oyun pazarı; Steam, Epic Games Store, GOG, PlayStation Store ve Xbox Store gibi birçok farklı dağıtım platformuna bölünmüş durumdadır. Bu bölünmüşlük, oyuncuların ilgilendikleri oyunları en uygun fiyatla satın almalarını zorlaştırmakta, her bir platformu tek tek manuel olarak kontrol etmeyi zorunlu kılmaktadır. DealHawk projesi, bu probleme modern, yüksek performanslı ve kullanıcı dostu bir çözüm sunmak amacıyla geliştirilmiştir.

1.1. Projenin Amacı

DealHawk, farklı dijital oyun mağazalarındaki fiyatları, indirim oranlarını ve tarihi en düşük fiyat verilerini tek bir merkezde toplayan bir oyun fiyat takip ve fırsat izleme platformudur. Projenin temel amaçları şunlardır:

- Fiyat Konsolidasyonu: Farklı oyun mağazalarındaki fiyat ve indirim verilerini tek bir platformda birleştirmek.
- Akıllı Alarm Sistemi: Kullanıcıların istedikleri oyunlar için hedef fiyat belirlemelerini sağlamak ve fiyat bu seviyenin altına düştüğünde sistem içi bildirimler üretmek.
- Tarihsel Analiz: Oyunların fiyat geçmişini kaydederek kullanıcılara grafiksel analiz sunmak ve "şimdi al" ya da "bekle" kararını vermelerine yardımcı olmak.
- Topluluk Etkileşimi: Kullanıcıların oyunlara puan vermesini, yorum yapmasını sağlamak ve moderatör onaylı bir inceleme sistemi kurmak.
- Modüler ve Sürdürülebilir Mimari: Projenin endüstri standardı olan Onion (Clean) Architecture şablonuna göre tasarlanarak gelecekteki geliştirmelere ve teknoloji değişimlerine kolayca uyum sağlamasını garantilemek.

1.2. Proje Kısıtları

Projenin geliştirilmesinde ve canlıya alınmasında dikkate alınan temel kısıtlar şunlardır:

- Dış Servis Bağımlılığı (API Rate Limits): Oyun verileri ve anlık fiyatlar CheapShark REST API üzerinden çekilmektedir. Bu servisin istek limitleri ve yanıt süreleri sistemin güncelleme sıklığını doğrudan etkilemektedir.

- Gerçek Zamanlı Güncelleme Maliyeti: Binlerce oyunun fiyat bilgisinin sürekli anlık olarak güncellenmesi veritabanı ve ağ trafiği üzerinde büyük bir yük oluşturmaktadır. Bu nedenle güncellemeler Hangfire aracılığıyla belirli zaman dilimlerine bölünmüş arka plan görevleri ile gerçekleştirilmektedir.
- Veritabanı Boyutu: Fiyat geçmişi verileri (PriceHistory) zamanla katlanarak büyümektedir. Bu durum veri tabanı şişmesini önlemek adına indeksleme ve optimize edilmiş SQL sorguları yazılmasını gerektirmiştir.
- Tarayıcı Tabanlı Oturum Yönetimi: REST API (JWT Bearer Token) ile MVC Web arayüzü (Cookie Authentication) arasındaki entegrasyonun güvenli bir şekilde claims yönetimi üzerinden koordine edilmesi sağlanmıştır.

2. Materyal ve Yöntem

2.1. Kullanılan Teknolojiler ve Araçlar

- ASP.NET Core (Web API & MVC), projenin arka plan çalışma çerçevesini (framework) oluşturur. Uygulama, istemci ve sunucu katmanlarının birbirinden bağımsız çalışabilmesi için decoupled (ayrık) bir mimaride tasarlanmıştır. ASP.NET Core 10.0, yüksek istek/yanıt performansı ve yerleşik bağımlılık enjeksiyonu (Dependency Injection) desteğiyle projenin ana omurgasını oluşturur.
- Microsoft SQL Server, ilişkisel verilerin tutarlı bir şekilde depolanması amacıyla tercih edilmiştir. Geliştirme ortamında hafif ve hızlı çalışan LocalDB sürümü kullanılmış, tablolar arası ilişkiler normalizasyon kurallarına uygun olarak yapılandırılmıştır.
- Entity Framework Core, veri tabanı işlemlerinin nesne yönelimli (OOP) olarak yönetilmesini sağlayan nesne-ilişkisel eşleme (ORM) aracıdır. Kod öncelikli (Code-First) yaklaşımıyla veri tabanı şemasının kod üzerinden tasarlanmasını ve migrasyonların (database migrations) hatasız şekilde yönetilmesini sağlar.
- ASP.NET Core Identity, Kullanıcıların sisteme güvenli bir şekilde üye olması, giriş yapması ve şifre sıfırlama gibi kimlik doğrulama (Authentication) işlemlerinin yönetilmesi

amacıyla kullanılmıştır. Ayrıca kullanıcıların rollerine göre (Admin, Moderator, User, Guest) yetkilendirilmesini (Authorization) sağlar.

- Hangfire, belirli zaman aralıklarıyla arka planda çalıştırılması gereken görevleri yöneten gelişmiş bir arka plan görev yöneticisidir. DealHawk projesinde, dış oyun servisinden fiyatların düzenli aralıklarla çekilerek veritabanının güncellenmesi sürecini koordine eder.
- Scalar API Reference, RESTful API uç noktalarının (endpoints) modern ve görsel bir şekilde belgelenmesi, test edilmesi amacıyla kullanılmıştır. Klasik Swagger arayüzlerine kıyasla çok daha hızlı, interaktif ve şık bir oyun alanı (playground) sunar.
- FluentValidation, iş mantığında (CQRS) kullanılan komut ve sorguların (Request) iş kurallarına göre denetlenmesini sağlar. Kullanıcı kayıt parametreleri veya fiyat alarmı isteklerinin veri doğruluğunu denetler.
- ADO.NET, yönetici paneli dashboard ekranlarında, karmaşık istatistiksel sorguların en hızlı şekilde çalıştırılması için ham SQL sorguları yazılmasını sağlayan düşük seviyeli veri erişim teknolojisidir. EF Core'un oluşturabileceği olası performans kayıplarını önlemek amacıyla tercih edilmiştir.
- Serilog, uygulama içerisinde gerçekleşen kritik olayların, hataların ve güvenlik denetim kayıtlarının konsol veya fiziksel dosya ortamına yapılandırılmış (structured) olarak loglanmasını sağlar.
- IMemoryCache ile birlikte ziyaret edilen sayfaları tekrar ziyaret edildiğinde daha hızlı ulaşırız.

Kütüphane	Sürüm
ASP.NET Core	10.0
Entity Framework Core	10.0
AutoMapper	13.0.1
FluentValidation	12.1.1
Hangfire	1.8.23
Serilog	10.0.0

Şekil 1. Kütüphaneler

2.2. Yazılım Mimarisi (Onion Architecture)

DealHawk, bağımlılıkların dıştan içe doğru aktığı, iş mantığının yani domain ve application katmanının api, framework ve veri tabanı gibi dış etkenlerden tamamen yalıtıldığı Onion Architecture şablonunu kullanır. Bu mimari yaklaşım sayesinde uygulamanın test edilebilirliği ve sürdürülebilirliği en üst seviyeye çıkarılmıştır.

2.2.1. Domain (Çekirdek/Varlık Katmanı - DealHawk.Domain)

Projenin en içteki kalbidir. Hiçbir dış kütüphaneye veya projeye bağımlılığı yoktur. Sadece C# dil özelliklerini barındırır.

- **Entities:** Sistemdeki ana nesneleri temsil eder (Game, Publisher, Genre, Platform, Store, StoreGame, CurrentPrice, PriceHistory, PriceAlert, Review, Notification, AuditLog, StoreSyncLog, ApplicationUser, Whishlist, Favorite, GameGenre, GamePlatform).
- **Enums:** Sistemde kullanılan sabit durumların (ReviewStatus, SyncStatus) tanımlandığı yerdir.

2.2.2. Application (Uygulama/İş Mantığı Katmanı - DealHawk.Application)

Sistemin iş mantığını ve kullanım senaryolarını barındırır. CQRS (Command Query Responsibility Segregation) desenini uygular.

- **MediatR:** Komutların (yazma işlemleri) ve sorguların (okuma işlemleri) tetiklenmesini ve işlenmesini yönlendirir.
- **DTOs & AutoMapper:** Veritabanı modellerinin istemciye doğrudan açılmasını önler, güvenli veri transfer nesneleri (DTO) sağlar.
- **FluentValidation:** API'ye gelen isteklerin doğruluğunu denetler (örneğin, şifre uzunluğu, geçerli e-posta veya alarm fiyatının sıfırdan büyük olması).
- **Interfaces:** Dış katmanlarda (Infrastructure, Persistence) uygulanacak olan servislerin arayüz sözleşmelerini tanımlar (IUnitOfWork, ICacheService, ICheapSharkService).

2.2.3. Persistence (Veri Eriřim Katmanı - DealHawk.Persistence)

Veritabanı işlemlerinin fiziksel olarak gerçekleştirildiğı katmandır.

- **ApplicationDbContext:** Entity Framework Core yapılandırmalarını ve tablolar arası ilişkileri yönetir.
- **Repositories & UnitOfWork:** Genel veri erişim kalıplarını (Generic Repository) uygulayarak kod tekrarını önler ve veritabanı işlemlerinin tek bir transaction içinde çalışmasını sağlar.
- **AdoNetStatsService:** EF Core'un karmaşık istatistiksel sorgulardaki performans kaybını aşmak amacıyla, doğrudan ham SQL (Raw SQL) kullanarak yönetici dashboard verilerini çeken yüksek performanslı ADO.NET servisini içerir.

2.2.4. Infrastructure (Altyapı/Dış Servis Katmanı - DealHawk.Infrastructure)

Dış dünya ile iletişim kuran üçüncü parti kütüphanelerin ve servislerin uygulandığı katmandır.

- **CheapSharkService:** Dış CheapShark API servisine HttpClientFactory kullanarak bağlanır, oyun fiyatlarını ve mağaza verilerini senkronize eder.
- **PriceSyncJob:** Hangfire tarafından tetiklenen, oyun fiyatlarını arka planda otomatik olarak güncelleyen ve fiyatı düşen oyunlar için kullanıcılara bildirim oluşturan arka plan görevidir.
- **JwtTokenGenerator:** Kullanıcı doğrulandığında güvenli JWT token üreten mekanizmadır.
- **CacheService:** Bellek içi (In-Memory) önbelleğe alma işlemlerini yönetir.

2.2.5. API ve WebMVC (Sunum Katmanları - DealHawk.API & DealHawk.WebMVC)

Sistemin dışarıya açılan iki farklı yüzüdür:

- **DealHawk.API:** Mobil uygulamalar, tarayıcılar veya MVC projesinin tüketebileceğı RESTful uç noktalar (endpoints) sunar. Kimlik doğrulamayı JWT Bearer Token ile yapar.
- **DealHawk.WebMVC:** Son kullanıcının tarayıcı üzerinden eriştiğı kullanıcı dostu arayüzdür. Bootstrap 5 ve özel CSS ile tasarlanmıştır. API ile DealHawkApiClient servisi

aracılığıyla haberleşir ve oturum yönetimini Cookie Claims içine yerleştirilen JWT ile sürdürür.

2.3. Veritabanı Tasarımı ve İlişkisel Veri Modeli

DealHawk ilişkisel veri tabanı tasarımı, verilerin tutarlılığını sağlamak ve hızlı sorgulanabilmesini kolaylaştırmak amacıyla normalizasyon kurallarına uygun olarak tasarlanmıştır.

2.3.1. Temel Tablolar ve Veritabanı Varlıkları

- **Game (Oyunlar):** Oyun başlığı, dış API'deki CheapShark ID'si ve çıkış tarihi gibi temel bilgileri tutan merkezi tablodur.
- **Publisher (Yayıncılar) & Genre (Türler) & Platform (Platformlar):** Oyunların kategorize edilmesini sağlar. GameGenre ve GamePlatform ara tabloları ile oyunlar ile çoktan-çoğa (Many-to-Many) ilişki kurarlar. Bir yayıncı ise birden fazla oyuna sahip olabilir (One-to-Many).
- **Store (Mağazalar) & StoreGame (Mağaza Oyun Eşleşmesi):** Oyunun hangi mağazada (Steam, Epic Store, GOG vb.) satıldığını ve o mağazaya ait özel sayfa kimliğini (Store ID) tutar.
- **CurrentPrice (Güncel Fiyatlar):** Mağazaya ait oyunun o anki satış fiyatını, standart fiyatını ve tasarruf yüzdesini (savings) tutar.
- **PriceHistory (Fiyat Geçmişi):** Fiyat grafiklerini çizdirmek amacıyla oyunların tarihsel fiyat değişimlerini loglar. Fiyat düşüş grafiklerinin çıkarılmasında bu tablo kullanılır.
- **PriceAlert (Fiyat Alarmları):** Kullanıcının bir oyun için belirlediği maksimum bütçe limitini temsil eder. Fiyat bu bütçenin altına indiğinde tetiklenir.
- **Review (İncelemeler) & ReviewLike (Beğeniler):** Kullanıcıların oyunlara yazdığı inceleme yorumlarını ve bu yorumların diğer kullanıcılar tarafından beğenilme durumlarını tutar. İncelemeler yayınlanmadan önce ReviewStatus = Pending (Beklemede) olarak işaretlenir.
- **Notification (Bildirimler):** Kullanıcının istek listesindeki oyunlar hedef fiyata ulaştığında oluşan in-app (sistem içi) bildirimleri tutar.
- **AuditLog (Denetim Kayıtları):** Yönetici ve moderatörlerin sistem üzerindeki işlemlerini (oyun ekleme, yorum onaylama vb.) güvenlik amacıyla günlüğe kaydeder.

- **StoreSyncLog (Senkronizasyon Kayıtları):** Arka plan görevlerinin çalışma sürelerini, başarı durumlarını ve güncellenen oyun sayılarını takip etmek için kullanılır.

- **ApplicationUser (Kullanıcılar):** ASP.NET Core Identity tarafından yönetilen, kullanıcı hesap bilgilerini, kimlik doğrulama verilerini ve rol atamalarını tutan tablodur.

- **Wishlist (İstek Listesi):** Kullanıcının favori/istek listesine eklediği oyunları ve varsa hedef fiyat bilgisini tutar.

- **Favorite (Favoriler):** Kullanıcıların beğendiği/favoriye aldığı oyunları tutan basit ilişki tablosudur.

- **GameGenre & GamePlatform (Ara Tablolar):** Game ile Genre ve Game ile Platform arasındaki çoktan-çoğa (Many-to-Many) ilişkiyi kuran ara tablolardır.

2.4. Arka Plan İşleri ve Entegrasyonlar (Hangfire & CheapShark)

DealHawk platformunun güncel kalmasını sağlayan en kritik mekanizma, dış servis entegrasyonu ve arka plan görevleridir.

2.4.1. CheapShark REST API Entegrasyonu

Uygulama, CheapShark servisinin /deals ve /games uç noktalarını sorgulayarak piyasadaki en güncel oyun indirimlerini çeker. HTTP istekleri .NET HttpClientFactory kullanılarak yönetilir. Bu sayede bağlantı havuzu (connection pooling) sunucu düzeyinde optimize edilir ve soket tükenmesi (socket exhaustion) gibi bağlantı hataları engellenir.

2.4.2. Hangfire ile Zamanlanmış Görev Yönetimi (PriceSyncJob)

Fiyatların güncellenmesi işlemi kullanıcıyı engellemek adına asenkron olarak arka planda koordine edilir. Hangfire paneli aracılığıyla tanımlanan PriceSyncJob görevi:

Saatlik veya günlük periyotlarla CheapShark API'sinden aktif indirim listesini talep eder. Çekilen fiyat verilerini veritabanındaki CurrentPrice tablosu ile karşılaştırır. Fiyatı değişen veya yeni indirim giren oyunlar için CurrentPrice tablosunu günceller ve PriceHistory tablosuna yeni bir kayıt ekler. Fiyat güncellemesi bittikten sonra, aktif PriceAlert (Fiyat Alarmları) listesini kontrol eder. Eğer güncel fiyat, kullanıcının alarm kurduğu fiyat limitinin altındaysa asenkron olarak Notification tablosuna ilgili kullanıcı için indirim uyarısı yazar.

3. KURULUM VE UYGULAMA ADIMLARI

3.1. Ön Gereksinimler

Uygulamayı yerel ortamda derlemek ve çalıştırmak için sistemde aşağıdaki yazılımların kurulu olması gerekmektedir:

- .NET 10.0 SDK: C# projesinin derlenebilmesi ve çalıştırılabilmesi için gereklidir.
- SQL Server LocalDB / Express: İlişkisel verilerin saklanacağı yerel veritabanı sunucusu.
- Visual Studio 2022 (v17.10+) veya VS Code: Kod düzenleme, hata ayıklama ve proje yönetimi ortamı.
- Entity Framework Core CLI Araçları: Veritabanı migrasyonlarını terminal üzerinden yönetmek için gerekli olan CLI araç takımı (dotnet-ef).

3.2. Veritabanı Kurulumu ve Migrasyonların Uygulanması

Projeyi bir terminal penceresinde açın veya klonlayın.

DealHawk.API/appsettings.json dosyasındaki ConnectionStrings tanımının yerel veritabanı sunucusu bağlantı ayarlarınızla eşleştiğinden emin olun:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=.;\\SQLEXPRESS;Database=DealHawkDb;Trusted_Connection=True;MultipleActiveResultSets=true;Encrypt=False"
  },
}
```

Şekil 2. Connection String

Terminalde projenin kök dizinine gidin ve aşağıdaki komutu çalıştırarak veri tabanı şemasını ve tabloları oluşturun:

```
$ dotnet ef database update --project DealHawk.Persistence --startup-project DealHawk.API
```

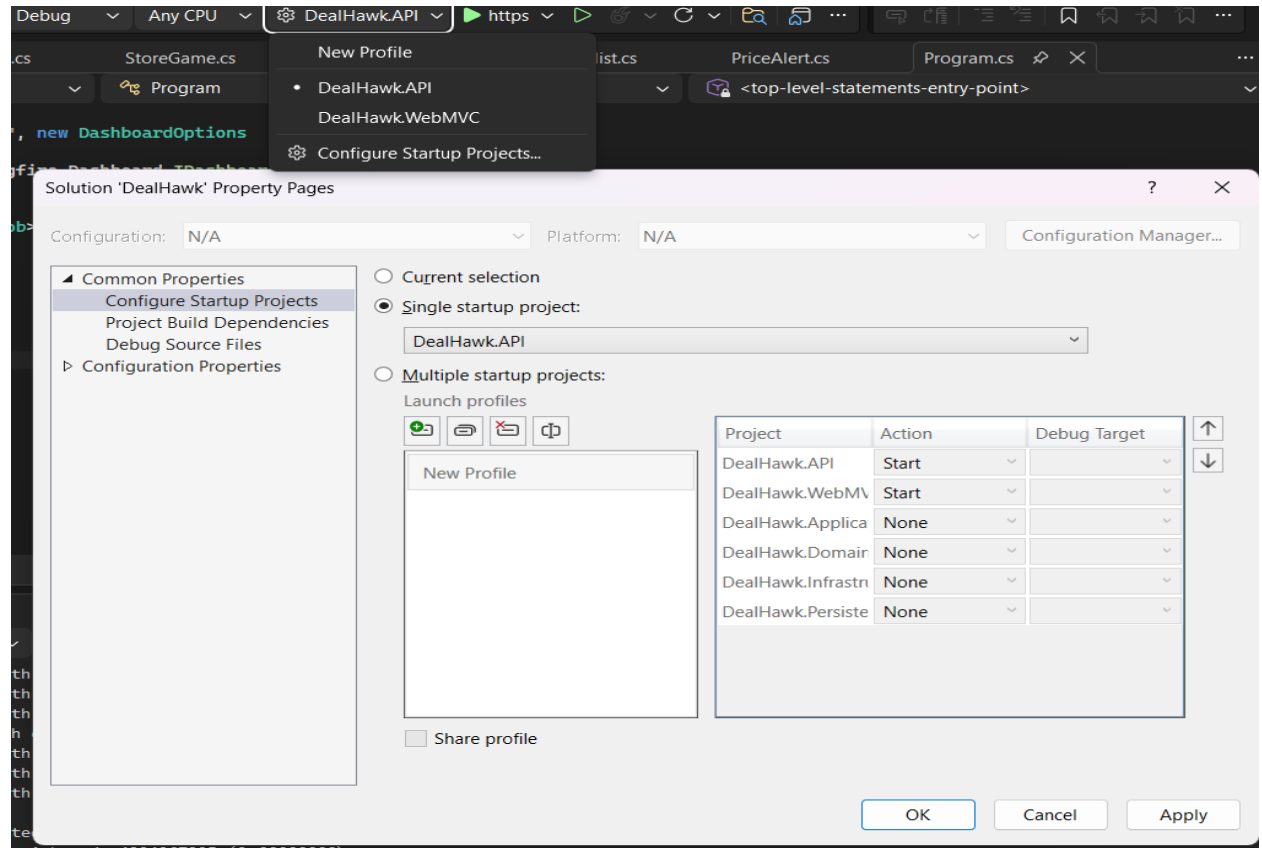
Şekil 3. Veri Tabanı Oluşturma

Bu komut persistence katmanındaki ApplicationDbContext konfigürasyonunu okuyarak SQL Server üzerinde DealHawkDb veritabanını ve gerekli tüm ilişkisel tabloları otomatik olarak kurar.

Veritabanı başarıyla oluşturulduktan sonra, ilk çalıştırmada DatabaseSeeder.cs sınıfı otomatik olarak devreye girerek test kullanıcılarını, rollerini, varsayılan mağaza bilgilerini ve popüler oyun katalog verilerini veritabanına yazar (seeding işlemi).

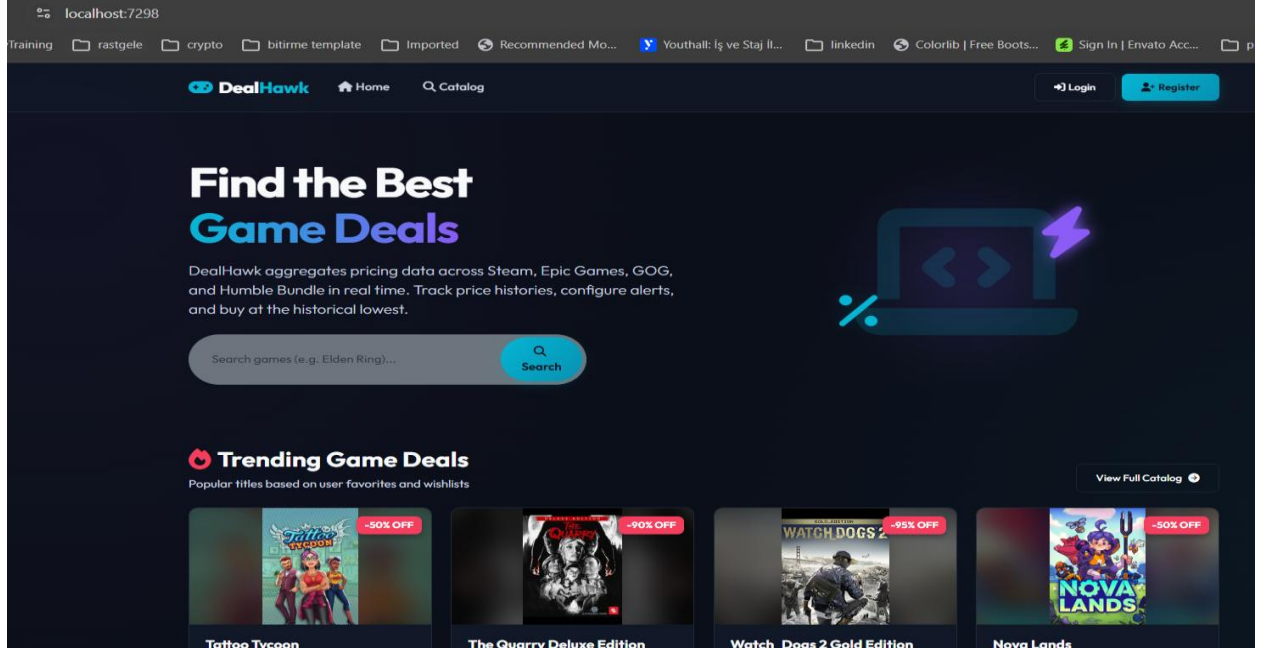
3.3. API Servisinin ve Arayüzün Çalıştırılması

Visual Studio üzerinden Start butonunu yanındaki configure new profile tıklayıp başlangıç için hem api hem arayüzü çalışacak şekilde seçmemiz gerekiyor.



Şekil 4. Proje Başlatma Profili Oluşturma

Daha sonra ok deyip devam edersek ve projeyi başlatırsak <https://localhost:7298/> üzerinden projemiz başlatılacaktır ve bize ana sayfa arayüzümüzü gösterecektir.

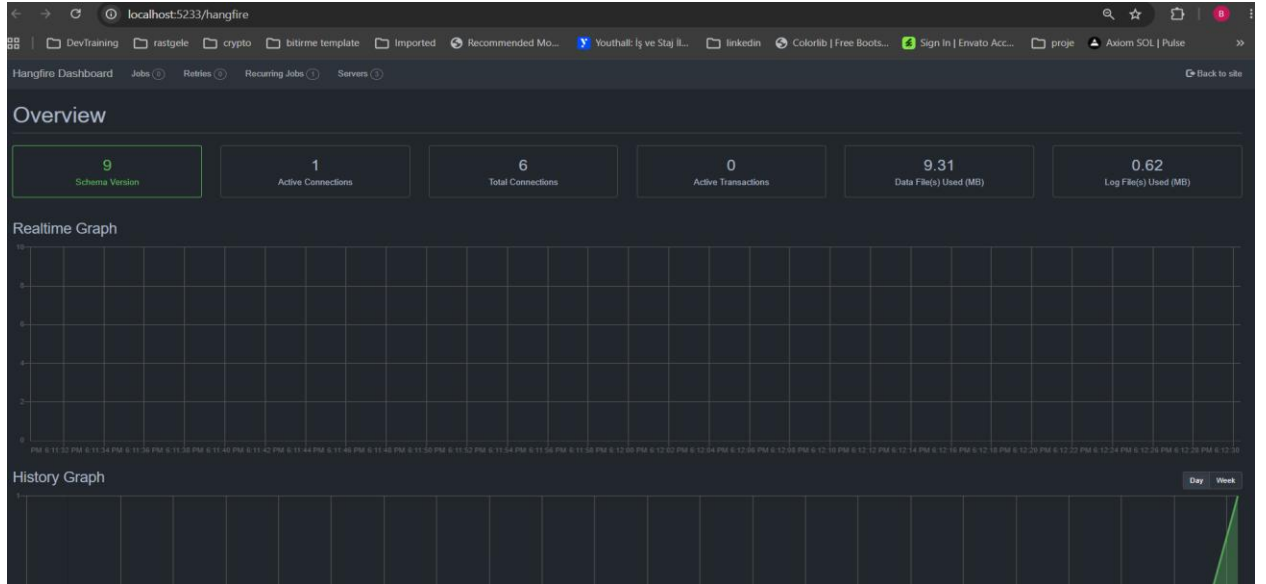


Şekil 5. Anasayfa

3.4. Hangfire Arayüzü ile Görevlerin İzlenmesi

Arka plan iş kuyruklarını ve zamanlanmış senkronizasyon görevlerini takip etmek amacıyla Hangfire dashboard entegrasyonu yapılmıştır.

Tarayıcınızdan <http://localhost:5233/hangfire> adresine gidin.



Şekil 5. Hangfire Arayüzü

Bu panel üzerinden:

Zamanlanmış görevlerin (Recurring Jobs) çalışma sıklıklarını (cron ifadelerini) görebilirsiniz.

Kuyrukta bekleyen (Enqueued), başarısız olan (Failed) veya başarıyla tamamlanan (Succeeded) arka plan görevlerini takip edebilirsiniz.

İndirim senkronizasyon görevini (PriceSyncJob) beklemeden anlık olarak manuel tetikleyebilirsiniz (Trigger Now).

3.5. Roller ve Yetkilendirme Senaryoları

DealHawk uygulamasında rol tabanlı yetkilendirme (Role-based Authorization) modeli aktif olarak kullanılmaktadır. Sistemde 4 temel rol tanımlanmıştır:

Guest (Ziyaretçi):

- Giriş yapmadan platformu kullanabilir.
- Oyun kataloglarını arayabilir, filtreleyebilir ve oyun detay sayfalarındaki fiyat karşılaştırmalarını inceleyebilir.
- Ancak fiyat alarmı oluşturamaz, favori listesi yapamaz ve oyunlara inceleme yazamaz.

User (Kayıtlı Kullanıcı):

- Kendi profiline giriş yapabilir.
- Oyunları favorilerine ekleyebilir veya istek listesine (Wishlist) alabilir.
- Oyunlar için hedef fiyat belirleyerek Fiyat Alarmı (PriceAlert) oluşturabilir.
- Satın aldığı veya deneyimlediği oyunlara inceleme (puan ve yorum) yazabilir.
- Fiyat düşüşlerinde oluşan bildirimleri kendi profil sayfasından okuyabilir ve okundu olarak işaretleyebilir.

Moderator (Moderatör):

- Kayıtlı kullanıcının tüm yetkilerine sahiptir.
- Ek olarak, kullanıcılar tarafından yazılan oyun incelemelerini denetlemekle görevlidir.

- Moderasyon panelinden bekleyen inceleme yorumlarını okuyarak topluluk kurallarına göre onaylayabilir veya reddedebilir. Onaylanan yorumlar anında oyun sayfasında halka açık olarak listelenir.

Admin (Yönetici):

- Sistemdeki en yüksek yetki seviyesine sahiptir.
- Yönetim paneli üzerinden sistemin genel istatistiklerini (toplam oyun, kullanıcı, aktif alarmlar vb.) görebilir.
- Güvenlik denetim günlüğünü (AuditLog) ve senkronizasyon günlüklerini (StoreSyncLog) inceleyebilir.
- CheapShark API'sinden toplu oyun veri aktarımını (Bulk Import) başlatabilir veya tekil oyunları sisteme ithal edebilir.
- Hangfire arka plan işlerini doğrudan yönetebilir.

4. SONUÇ VE GELECEK ÇALIŞMALAR

DealHawk projesi, günümüz oyun pazarındaki fiyat karmaşasına ve çoklu platform takibinin getirdiği zorluklara karşı modern bir mimariyle geliştirilmiş başarılı bir platformdur.

Geliştirilen kararlı sürümün ardından platformun performansını, kullanıcı deneyimini ve ölçeklenebilirliğini daha da artırmak amacıyla çeşitli geliştirmelerin hayata geçirilmesi planlanmaktadır. Bu kapsamda, sık ziyaret edilen oyun detay sayfaları ile anlık popüler indirimlerin veritabanına erişmeden sunulabilmesi için Redis Dağıtık Önbellekleme (Distributed Caching) altyapısı kullanılarak sunucu yükünün azaltılması hedeflenmektedir. Kullanıcılara fiyat düşüşleri ve önemli gelişmeler hakkında sayfa yenilemeye gerek kalmadan anlık bildirimler gönderebilmek amacıyla SignalR tabanlı gerçek zamanlı bildirim sistemi entegre edilmesi planlanmaktadır. Ayrıca, mevcut SQL tabanlı arama altyapısının yerine Elasticsearch kullanılarak daha hızlı, ölçeklenebilir ve yazım hatalarını tolere eden (fuzzy search) gelişmiş bir katalog arama deneyimi sunulması amaçlanmaktadır. Bunun yanında, kullanıcı kayıt ve giriş süreçlerini kolaylaştırmak amacıyla OAuth 2.0 desteği eklenerek kullanıcıların Steam, Discord veya Google hesapları üzerinden tek tıkla platforma üye olabilmeleri ve güvenli bir şekilde giriş yapabilmeleri hedeflenmektedir. Bu geliştirmeler sayesinde platformun hem teknik altyapısının güçlendirilmesi hem de modern kullanıcı beklentilerine uygun, daha hızlı ve etkileşimli bir deneyim sunması amaçlanmaktadır.

5. KAYNAKÇA

- [1] learn.microsoft.com/dotnet
- [2] learn.microsoft.com/ef/core
- [3] docs.hangfire.io
- [4] cheapshark.com/api
- [5] scalar.com
- [6] <https://github.com/DapperLib/Dapper>