**Search And Navigation Ödev Dökümantasyonu**

1. **Proje Başlığı :** Search & Navigation Software Assignment E-Ticaret Öneri Motoru

* **Hazırlayan :** Hasan Helvalı
* **Tarih:** 19/09/2024

1. **İçindekiler**

* **Giriş .............................................. 2**
* **Ana bileşenler .............................................. 2**
* **Projenin amacı .............................................. 2**
* **Projenin Kapsamı .............................................. 3**
* **Genel Mimari Diyagramı .............................................. 4**
* **Bilesenlerin Açıklanması .............................................. 4**
* **İş Akışı .............................................. 7**
* **Teknoloji ve Araç Seçimleri .............................................. 8**
* **Veritabanı Şeması .............................................. 10**
* **Uygulama Bileşenleri .............................................. 11**
* **Veri Akışı Detayları .............................................. 17**
* **Dağıtım ve Konteynerizasyon .............................................. 18**
* **Ek Açıklamalar .............................................. 20**
* **Projenin DeĞerlendirilmesi SonuÇ .............................................. 20**
* **Gelebilecek Geliştirmeler .............................................. 21**

1. **Giriş**
2. **Proje Tanımı :** Bu projenin amacı, bir e-ticaret platformu için kapsamlı bir gerçek zamanlı öneri motoru geliştirmektir. Sistem, kişiselleştirilmiş ve kişiselleştirilmemiş ürün önerileri oluşturmak için toplu ve gerçek zamanlı verileri işleyecektir. Çözüm, veri alımı, akış işleme, ETL (Extract, Transform, Load) süreçleri ve kullanıcılara gerçek zamanlı olarak öneri listeleri sunmak için yüksek performanslı bir REST API içeren çeşitli bileşenlerden oluşmaktadır.

**Ana Bileşenler:**

* **View Producer Uygulaması (Producer App):** Ürün görüntüleme verilerini Kafka'ya yayınlar.
* **Stream Reader Uygulaması (Consumer App):** Kafka'dan gelen verileri tüketir ve veritabanına kaydeder.
* **Öneri REST API:** Kullanıcılara tarama geçmişi ve en çok satan ürün önerilerini sunar.
* **Veritabanı (PostgreSQL):** Kullanıcı, ürün, sipariş ve görüntüleme verilerini depolar.
* **Veri Aktarımı (Apache Kafka):** Producer ve Cunsomer uygulamalar arasında köprü görevi görür.
* **Altyapı:** Docker ve Docker Compose kullanılarak tüm bileşenlerin konteynerleştirilmesi ve koordinasyonu sağlanır.

1. **Projenin Amacı**

Bu projenin temel amacı, kullanıcı deneyimini artırmak ve satışları desteklemek için etkili bir öneri sistemi oluşturmaktır. Proje, aşağıdaki hedeflere odaklanmaktadır:

* **Kişiselleştirilmiş Öneriler Sağlamak:** Kullanıcıların geçmişte görüntülediği ürünlere dayanarak, ilgi alanlarına uygun ürün önerileri sunmak.
* **Genel En Çok Satan Ürünleri Sunmak:** Kullanıcı tarama geçmişi bulunmayan veya genel ilgi alanlarına yönelik en çok satan ürünleri önermek.
* **Gerçek Zamanlı Veri İşleme:** Kullanıcı etkileşimlerini anında işleyerek güncel ve doğru öneriler sağlamak.
* **Yüksek Performanslı API'ler:** Kullanıcı taleplerine hızlı yanıt vererek sorunsuz bir kullanıcı deneyimi sunmak.
* **Veri Tutarlılığı ve Güvenilirlik:** Kafka ve PostgreSQL kullanarak verilerin güvenilir ve tutarlı bir şekilde işlenmesini sağlamak.

1. **Proje Kapsamı**

**Kapsam Dahilinde Olanlar:**

1. **Veri Alımı ve Yayını:**
   * product-views.json dosyasından ürün görüntüleme verilerini okuyarak Kafka'ya yayınlama.
   * Kafka üzerinden gelen ürün görüntüleme verilerini tüketip veritabanına kaydetme.
2. **Veri İşleme ve ETL:**
   * Geçen ayın sipariş verilerini kullanarak kategoriye dayalı ve genel en çok satan ürünleri belirleme.
   * Kişiselleştirilmiş ve genel öneri listelerini oluşturma ve depolama.
3. **REST API Geliştirme:**
   * Kullanıcının son on görüntülenen ürününü getiren GET /api/browsing-history uç noktası.
   * Kullanıcının tarama geçmişinden belirli bir ürünü silen DELETE /api/browsing-history uç noktası.
   * Kullanıcının tarama geçmişine dayalı veya genel en çok satan ürünleri getiren GET /api/best-sellers uç noktası.
4. **Veritabanı Tasarımı ve Yönetimi:**
   * Kullanıcılar, ürünler ve ürün detay verilerini depolamak için gerekli tabloların oluşturulması.
5. **Konteynerleştirme ve Dağıtım:**.
   * Docker Compose kullanarak hizmetlerin koordinasyonu ve yönetimi.

**Kapsam Dışında Olanlar:**

* **Kullanıcı Arayüzü Geliştirme:** Proje kapsamında, REST API'nin kullanımına yönelik bir frontend arayüzü geliştirilmemektedir.
* **Gelişmiş Güvenlik Önlemleri:** Temel güvenlik uygulamaları dahil edilmekle birlikte, kapsamlı güvenlik önlemleri (örn. OAuth 2.0, JWT) detaylı olarak ele alınmamıştır.
* **Dağıtık Sistem Optimizasyonları:** Yüksek ölçeklenebilirlik ve dağıtık sistem optimizasyonları temel düzeyde sağlanmaktadır, ancak ileri düzey optimizasyonlar proje kapsamında değildir.
* **Farklı Veri Kaynakları Entegrasyonu:** Proje, belirtilen iki veri kaynağı (PostgreSQL ve product-views.json dosyası) ile sınırlıdır ve ek veri kaynakları entegrasyonu içermemektedir.

#### ****Genel Mimari Diyagramı****

#### ****1. Diyagram****

#### Aşağıda, projenin tüm bileşenlerinin birbirleriyle nasıl etkileştiğini gösteren genel bir mimari diyagramı bulunmaktadır:



#### ****2. Bileşenlerin Açıklamaları****

##### **2.1. View Producer Uygulaması (Producer App)**

* **Fonksiyon:** product-views.json dosyasından ürün görüntüleme verilerini okur ve her bir görüntüleme olayını Kafka konusu product-views üzerine saniyede bir olay hızıyla yayınlar.
* **Teknolojiler:** .NET Core/C#, Confluent.Kafka kütüphanesi.
* **İş Akışı:**
  1. JSON dosyasını satır satır okur.
  2. Her satırı ProductView nesnesine deserialize eder.
  3. Olaylara mevcut UTC zaman damgasını ekler.
  4. Kafka üreticisini kullanarak mesajı product-views konusuna gönderir.
  5. Saniyede bir gecikme ile olayların yayınlanma hızını kontrol eder.

##### **2.2. Kafka Kümesi**

* **Fonksiyon:** Gerçek zamanlı veri akışını yönetir ve bileşenler arasında veri iletimini sağlar.
* **Bileşenler:**
  + **Zookeeper:** Kafka'nın koordinasyonunu sağlar.
  + **Kafka Broker'ları:** Mesajların depolanması ve dağıtılmasından sorumludur.
* **Topic:**
  + **product-views:** Ürün görüntüleme olaylarının yayımlandığı ana konu.

##### **2.3. Stream Reader Uygulaması (Consumer App)**

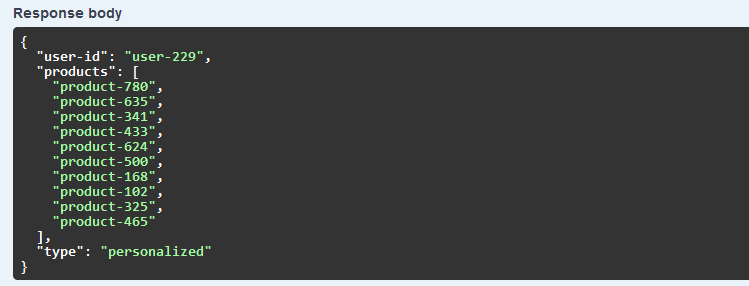
* **Fonksiyon:** Kafka konusu product-views üzerinden gelen ürün görüntüleme olaylarını tüketir ve PostgreSQL veritabanına kaydeder.
* **Teknolojiler:** .NET Core/C#, Confluent.Kafka kütüphanesi, Entity Framework Core.
* **İş Akışı:**
  1. Kafka tüketicisini yapılandırır ve product-views konusuna abone olur.
  2. Gelen mesajları sürekli olarak tüketir.
  3. Mesajları ProductViewJsonMessage nesnesine deserialize eder.
  4. İlgili varlıkları (Kullanıcı, Ürün, Bağlam) veritabanında kontrol eder ve ekler/günceller.
  5. ProductView kayıtlarını oluşturur ve veritabanına ekler.

##### **2.4. PostgreSQL Veritabanı**

* **Fonksiyon:** Kullanıcı verileri, ürün bilgileri, siparişler, sipariş öğeleri ve ürün görüntüleme geçmişini depolar.
* **Tablolar:**
  + **Users:** Kullanıcı bilgileri.
  + **Products:** Ürün bilgileri ve kategorileri.
  + **Contexts:** Olay kaynakları (örn. mobil uygulama).
  + **ProductViews:** Kullanıcıların ürün görüntüleme geçmişi.

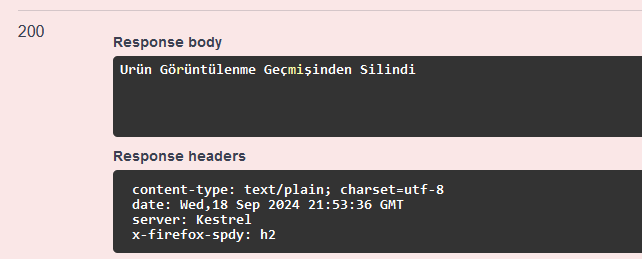
##### **2.5. Öneri REST API (Recommendation API)**

* **Fonksiyon:** Kullanıcılara tarama geçmişi ve en çok satan ürün önerilerini sağlamak için RESTful uç noktalar sunar.
* **Teknolojiler:** .NET Core/C#, ASP.NET Core, Entity Framework Core, Swagger.
* **Uç Noktalar:**
  1. **GET /api/browsinghistory/history**
     + **Girdiler:** user-id
     + **Fonksiyon:** Belirli bir kullanıcının son on görüntülenen ürününü, görüntüleme tarihine göre sıralanmış olarak getirir.
     + **Örnek Yanıt:**



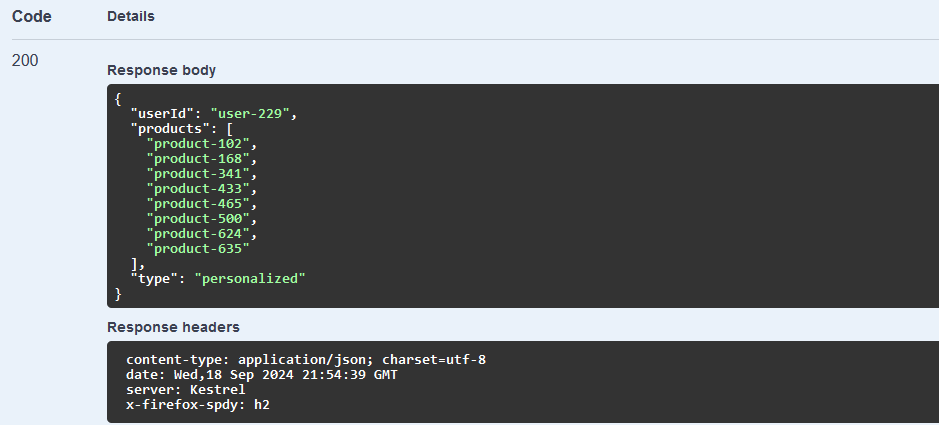
**DELETE /api/browsinghistory/history**

* **Girdiler:** user-id, product-id
* **Fonksiyon:** Kullanıcının tarama geçmişinden belirli bir ürünü siler.
* **Örnek Yanıt:**



**GET /api/bestsellers**

* **Girdiler:** user-id
* **Fonksiyon:**
  + Kullanıcının tarama geçmişi varsa, bu geçmişteki kategorilere dayalı en çok satan ürünleri önerir.
  + Tarama geçmişi yoksa, genel en çok satan ürün listesini döndürür.
* **Örnek Yanıt:**



**İş Akışı:**

1. **Browsing History:**
   * ProductViews tablosundan kullanıcıya ait son on ürün görüntüleme kaydını alır.
   * En az beş ürün olup olmadığını kontrol eder; yeterli değilse boş liste döndürür.
   * Son on ürünü JSON formatında kullanıcıya sunar.
2. **Delete Browsing History:**
   * ProductViews tablosundan belirtilen user-id ve product-id ile eşleşen kaydı bulur.
   * Kayıt varsa siler ve değişiklikleri kaydeder.
   * Kullanıcıya başarı mesajı döndürür.
3. **Best Seller Products:**
   * Verilen kayıtlarda kategori bilgisi bulunmadığından en çok satan ürün grupları saptanamadı. Bu sebeple en çok görüntülenen veriler en çok satılmış kabul edildi.
   * En az beş ürün olup olmadığını kontrol eder; yeterli değilse boş liste döndürür.
   * En çok satan ürünleri JSON formatında kullanıcıya sunar.

**Genel İş Akışı**

1. **Ürün Görüntüleme Olayları:**
   * Kullanıcılar e-ticaret platformunda ürünleri görüntülediğinde, bu görüntüleme olayları product-views.json dosyasına kaydedilir.
2. **View Producer Uygulaması:**
   * product-views.json dosyasından ürün görüntüleme olaylarını okur ve Kafka konusu product-views üzerine yayınlar.
3. **Kafka Kümesi:**
   * Görüntüleme olayları Kafka broker'larında depolanır ve dağıtılır.
4. **Stream Reader Uygulaması:**
   * Kafka'dan product-views konusunu tüketir, olayları deserialize eder ve PostgreSQL veritabanına kaydeder.
5. **PostgreSQL Veritabanı:**
   * Kullanıcılar, ürünler, siparişler ve ürün görüntüleme geçmişi veritabanında saklanır.
6. **Öneri REST API:**
   * Kullanıcılara tarama geçmişi ve en çok satan ürün önerilerini sağlamak için API uç noktalarını sunar.
   * Kullanıcılar, tarama geçmişlerine ve ilgi alanlarına göre kişiselleştirilmiş veya genel en çok satan ürünleri görüntüler.

### **Teknoloji ve Araç Seçimleri**

#### Kullanılan Teknolojiler:

1. **Programlama Dilleri ve Frameworkler**:
   * **C# (.NET 8)**: Projenin ana dili olarak seçilmiştir çünkü .NET 8, modern ve yüksek performanslı uygulamalar geliştirmek için uygun bir platform sunar. Ayrıca, EF Core ile veritabanı etkileşimleri kolaylaştırılır ve REST API geliştirmek için gerekli araçları sağlar.
   * **Entity Framework Core (EF Core)**: PostgreSQL ile veri etkileşimini kolaylaştıran bir ORM (Object-Relational Mapping) kütüphanesidir. Veritabanı işlemlerinin ve sorgularının daha basit ve güvenli bir şekilde yapılmasını sağlar.
2. **Veri Tabanları**:
   * **PostgreSQL**: İlişkisel veri tabanı yönetim sistemi olarak seçilmiştir. Açık kaynaklı, güçlü özelliklere sahip ve büyük veri işleme kapasitesine sahiptir. Ayrıca, Docker ile kolayca konteynerize edilebilir.
3. **Mesajlaşma Sistemi**:
   * **Apache Kafka**: Gerçek zamanlı veri akışı ve işleme için seçilmiştir. Kafka, yüksek verimlilik, ölçeklenebilirlik ve hata toleransı sağlar. Mesajların akışını yönetmek ve tüketmek için uygun bir çözüm sunar.
4. **Konteynerizasyon ve Orkestrasyon**:
   * **Docker ve Docker Compose**: Uygulamaları ve bağımlılıkları izole etmek ve dağıtmak için kullanılır. Docker Compose, çeşitli hizmetlerin aynı anda çalışmasını sağlayarak geliştirmenin ve dağıtımın kolaylaştırılmasını sağlar.
5. **API Geliştirme**:
   * **ASP.NET Core Web API**: RESTful API'ler oluşturmak için kullanılan bir framework'tür. Performansı yüksek, esnek ve ölçeklenebilir API'ler geliştirmek için idealdir.

#### Seçim Nedenleri:

1. **C# ve .NET 8**:
   * **Performans**: .NET 8, yüksek performanslı uygulamalar geliştirmek için optimize edilmiştir ve bu, gerçek zamanlı veri işleme için kritiktir.
   * **Ekosistem**: .NET ekosistemi, geniş bir kütüphane ve araç seti sunar, bu da geliştirme sürecini hızlandırır ve verimliliği artırır.
   * **Topluluk ve Destek**: Büyük bir geliştirici topluluğuna ve geniş bir destek ağına sahiptir.
2. **Entity Framework Core (EF Core)**:
   * **Veri Yönetimi**: Veritabanı etkileşimlerini kolaylaştırır ve veri tabanı şeması ile kod arasındaki eşleşmeyi yönetir.
   * **Migration Yönetimi**: Veritabanı şemasındaki değişiklikleri yönetmek için migration özellikleri sağlar.
3. **PostgreSQL**:
   * **Özellikler**: ACID uyumluluğu, yüksek performans ve geniş veri tipleri desteği sunar.
   * **Topluluk ve Destek**: Geniş bir topluluğa ve kapsamlı dökümantasyona sahiptir.
4. **Apache Kafka**:
   * **Gerçek Zamanlı İşleme**: Yüksek veri hacimlerini gerçek zamanlı olarak işleyebilir ve ölçeklenebilir.
   * **Yüksek Performans**: Yüksek verim ve düşük gecikme süresi sağlar.
5. **Docker ve Docker Compose**:
   * **Taşınabilirlik**: Uygulamalar ve bağımlılıkları izole ederek farklı ortamlar arasında taşınabilirlik sağlar.
   * **Orkestrasyon**: Çok sayıda servisin yönetimini ve koordinasyonunu basit hale getirir.

#### Alternatifler:

1. **Programlama Dilleri ve Frameworkler**:
   * **Java**: Spring Boot ile REST API geliştirme. Java geniş bir topluluğa ve sağlam bir ekosisteme sahiptir ancak .NET 8'in sağladığı performans ve araçları sağlamakta biraz daha karmaşık olabilir.
   * **Node.js**: Gerçek zamanlı veri işleme için iyi bir alternatif olabilir, özellikle de yüksek performanslı API'ler için. Ancak, .NET 8 ile sağlanan yerleşik destek ve araçlar ile kıyaslandığında daha fazla yapılandırma gerektirebilir.
2. **Veri Tabanları**:
   * **MySQL**: Popüler ve güçlü bir alternatif olabilir, ancak PostgreSQL'in sunduğu bazı gelişmiş özelliklerden yoksun olabilir.
   * **MongoDB**: NoSQL veritabanı olarak kullanılabilir, ancak ilişkisel veriler için PostgreSQL kadar uygun olmayabilir.
3. **Mesajlaşma Sistemi**:
   * **RabbitMQ**: Mesajlaşma için iyi bir alternatif olabilir. Ancak, Kafka'nın sağladığı yüksek performans ve veri akış yönetimi özelliklerini sunmayabilir.
4. **Konteynerizasyon ve Orkestrasyon**:
   * **Kubernetes**: Daha büyük ölçekli ve karmaşık sistemler için Kubernetes kullanılabilir. Ancak, Docker Compose daha basit ve hızlı bir çözüm sağlar.

Bu seçimler, projenin gereksinimlerini karşılamak ve verimli bir şekilde gerçekleştirmek için en uygun teknolojiler olarak belirlenmiştir. Her bir teknolojinin avantajları ve projeye uygunluğu göz önünde bulundurulmuştur.

1. **Veritabanı Şeması**

#### Tablo Yapıları

#### Products Tablosu

#### ProductId: integer (IDENTITY)

#### ProductName: text

#### **Anahtarlar**:

#### ProductId birincil anahtar (Primary Key)

#### **İlişkiler**:

#### ProductViews tablosuyla bire-çok (one-to-many) ilişkisi var.

#### ProductViews Tablosu

#### PvId: integer (IDENTITY)

#### UserId: text

#### MessageId: text

#### EventTime: timestamp with time zone

#### ProductId: text

#### **Anahtarlar**:

#### PvId birincil anahtar (Primary Key)

#### **İlişkiler**:

#### Products tablosuyla ProductId üzerinden ilişki

#### Users tablosuyla UserId üzerinden ilişki

#### Users Tablosu

#### UserId: text

#### **Anahtarlar**:

#### UserId birincil anahtar (Primary Key)

#### İndeksler

#### ****Products Tablosu****:

#### ProductId üzerinde birincil anahtar indeksi

#### ProductViews tablosuyla ProductId üzerinden ilişki

#### SQL Scriptleri

#### Her tablo için SQL oluşturma komutları aşağıdaki gibidir:

#### Products Tablosu

**SQL**

CREATE TABLE Products (

ProductId SERIAL PRIMARY KEY,

ProductName TEXT NOT NULL

);

Yapay zeka tarafından oluşturulan kod. Dikkatle gözden geçirin ve kullanın. [SSS hakkında daha fazla bilgi](https://www.bing.com/new#faq).

#### ProductViews Tablosu

**SQL**

CREATE TABLE ProductViews (

PvId SERIAL PRIMARY KEY,

UserId TEXT NOT NULL,

MessageId TEXT,

EventTime TIMESTAMPTZ,

ProductId TEXT NOT NULL,

FOREIGN KEY (ProductId) REFERENCES Products(ProductId),

FOREIGN KEY (UserId) REFERENCES Users(UserId)

);

Yapay zeka tarafından oluşturulan kod. Dikkatle gözden geçirin ve kullanın. [SSS hakkında daha fazla bilgi](https://www.bing.com/new#faq).

#### Users Tablosu

**SQL**

CREATE TABLE Users (

UserId TEXT PRIMARY KEY

);

1. **Uygulama Bileşenleri**

### **1. View Producer Uygulaması**

**Amaç:** product-views.json dosyasını okuyup Kafka'ya olay göndermek.

**Teknolojiler:**

* **Programlama Dili:** C# (.NET Core)
* **Kafka İstemci Kütüphanesi:** Confluent.Kafka

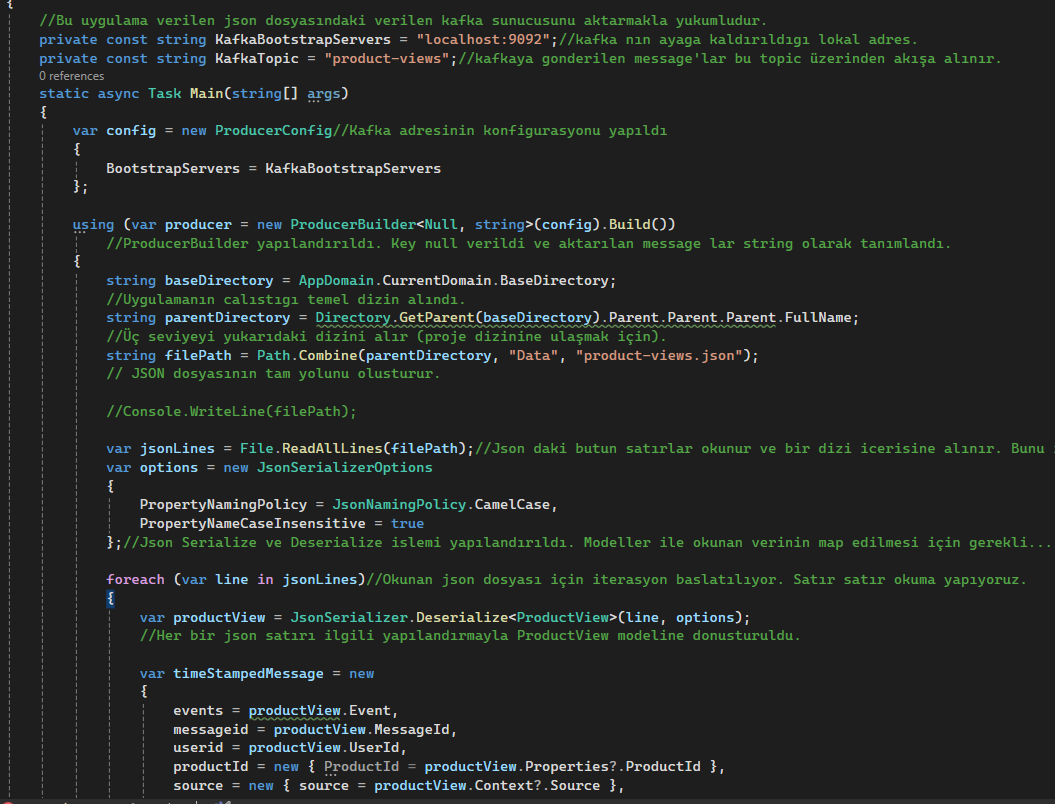
**Kurulum ve Çalıştırma:**

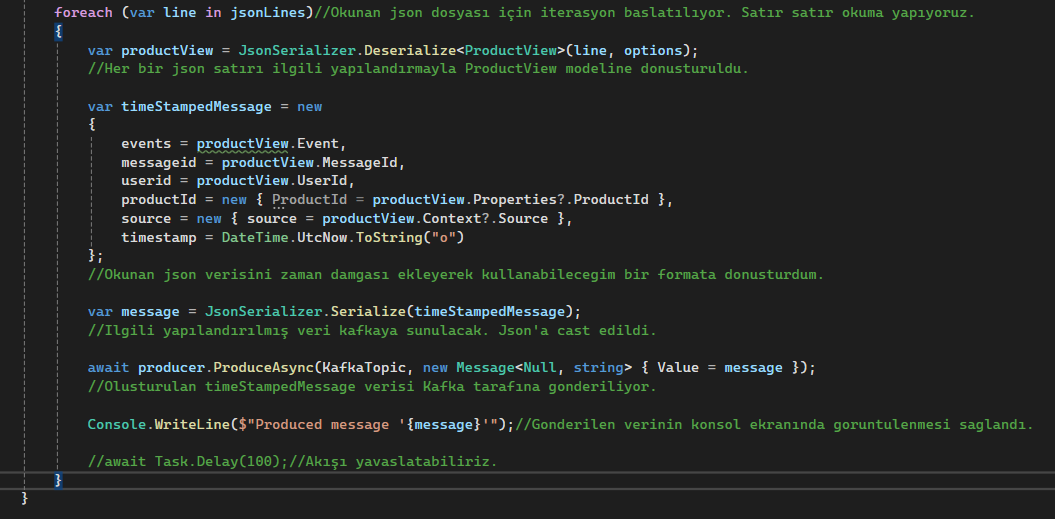
1. **Bağımlılıklar:** Confluent.Kafka NuGet paketini yükleyin.
2. **Çalıştırma:** dotnet run komutu ile uygulamayı çalıştırın.

**Kod Yapısı:**

* **Program.cs:** Uygulamanın giriş noktasıdır. JSON dosyasını okur ve mesajları Kafka'ya gönderir.
* **KafkaProducerService.cs:** Kafka'ya mesaj gönderme işlemlerini içerir.
* **ProductView.cs:** Ürün görünüm olayının yapısını temsil eder.
* **appsettings.json:** Kafka ayarlarını içeren yapılandırma dosyasıdır.

**Örnek Kod Parçaları:**





### **2. Stream Reader Uygulaması**

**Amaç:** Kafka'dan veri okuyup gerçek zamanlı işlemek.

**Teknolojiler:**

* **Programlama Dili:** C# (.NET Core)
* **Kafka İstemci Kütüphanesi:** Confluent.Kafka
* **Veritabanı Erişimi:** Entity Framework Core (EF Core)

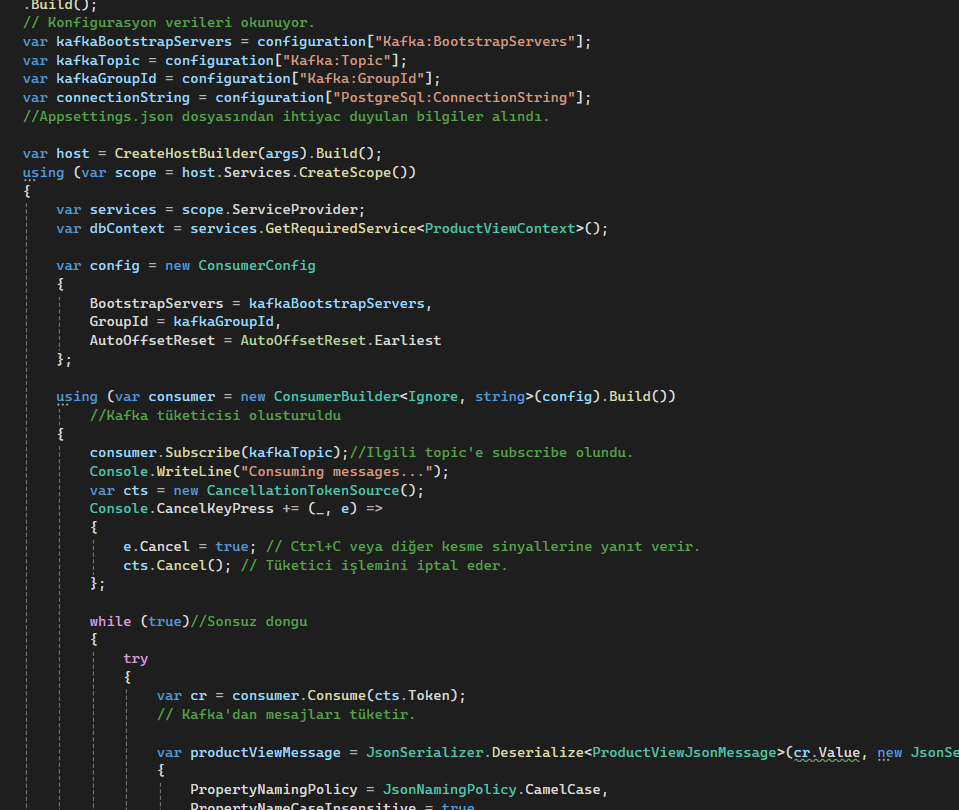
**Kurulum ve Çalıştırma:**

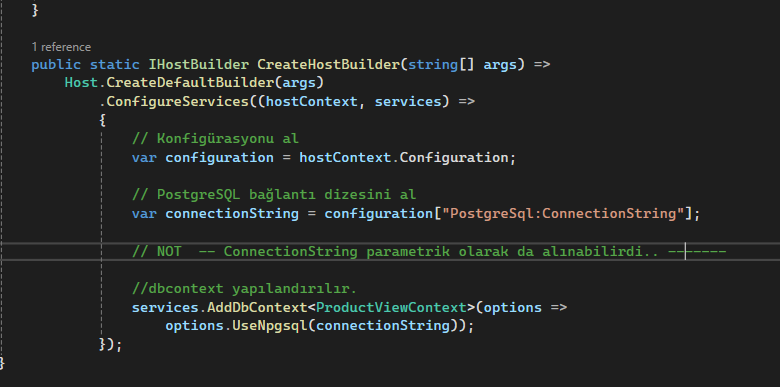
1. **Bağımlılıklar:** Confluent.Kafka ve Microsoft.EntityFrameworkCore NuGet paketlerini yükleyin.
2. **Çalıştırma:** dotnet run komutu ile uygulamayı çalıştırın.

**Kod Yapısı:**

* **Program.cs:** Uygulamanın giriş noktasıdır. Kafka tüketicisini kurar ve mesajları işler.
* **KafkaConsumerService.cs:** Kafka'dan mesaj tüketme işlemlerini içerir.
* **ProductViewContext.cs:** Veritabanı erişimi için EF Core DbContext'i içerir.
* **appsettings.json:** Kafka ve veritabanı ayarlarını içeren yapılandırma dosyasıdır.

**Örnek Kod Parçaları:**





### **4. Recommendation API**

**Amaç:** Kullanıcı isteklerini alıp öneri listelerini döndürmek.

**Teknolojiler:**

* **Web Framework:** ASP.NET Core (API geliştirme için)
* **Veritabanı Erişimi:** Entity Framework Core

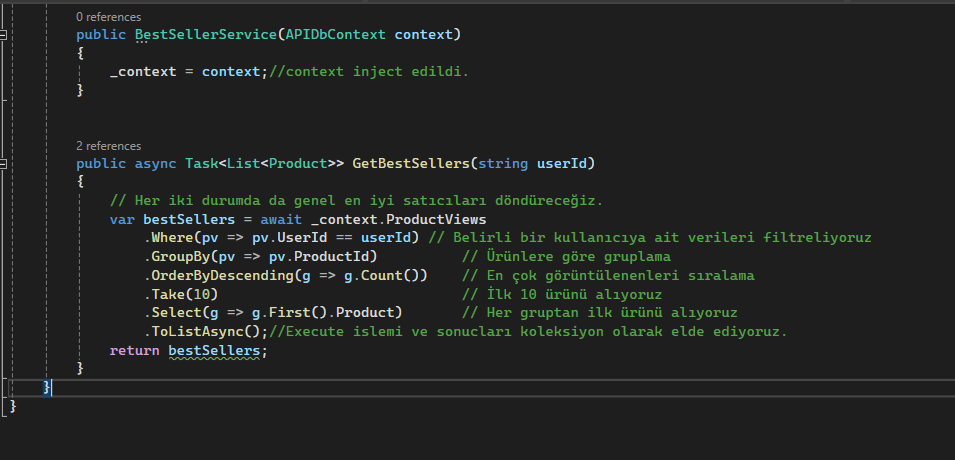
**Kurulum ve Çalıştırma:**

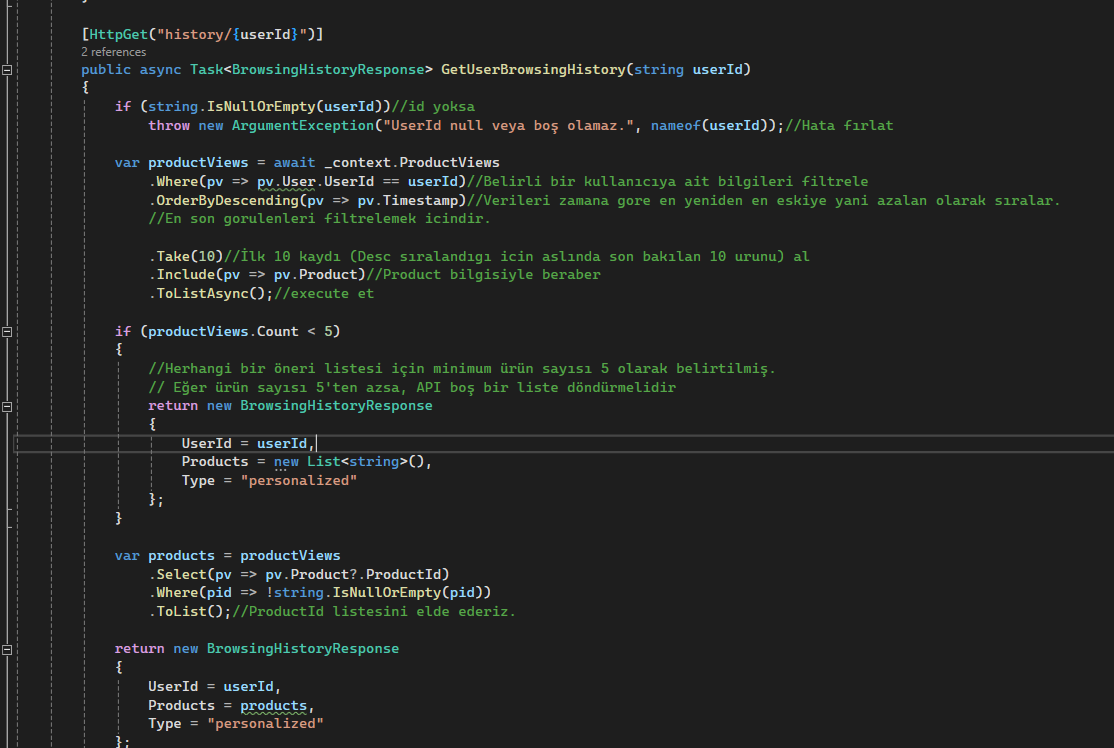
1. **Bağımlılıklar:** Microsoft.AspNetCore ve Microsoft.EntityFrameworkCore NuGet paketlerini yükleyin.
2. **Çalıştırma:** dotnet run komutu ile API'yi çalıştırın.

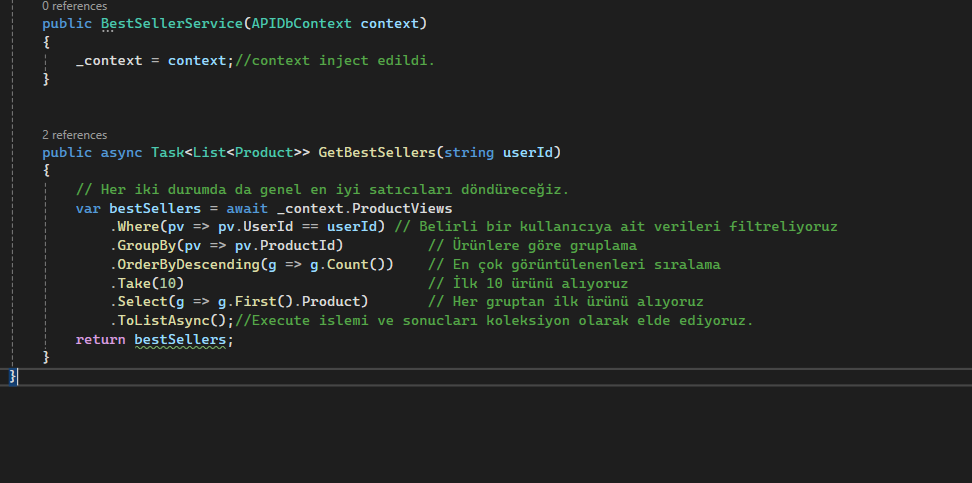
**Kod Yapısı:**

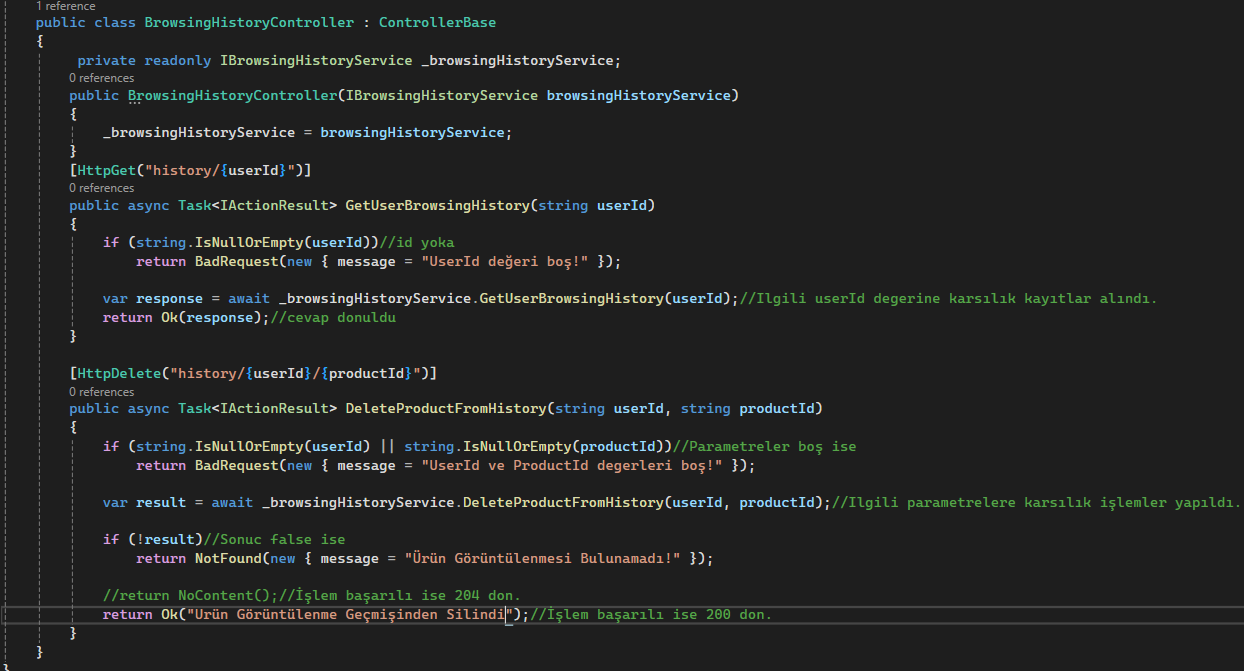
* **Kontrolcüler:**
  + **BrowsingHistoryController.cs:** Tarama geçmişi isteklerini işler.
  + **BestSellersController.cs:** En iyi satan ürün isteklerini işler.
* **Modeller:** Veri modellerini temsil eder (ProductView, Product, vb.)
* **appsettings.json:** API ayarlarını içeren yapılandırma dosyasıdır.

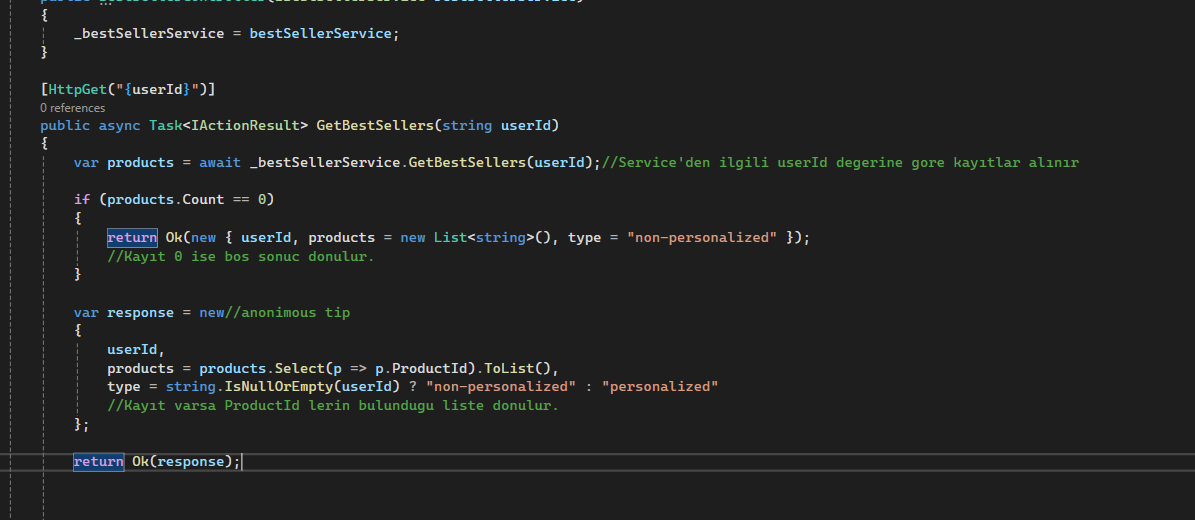
**Örnek Kod Parçaları:**











### **H. Veri Akışı Detayları**

#### 1. Veri Kaynakları

* **product-views.json:** Bu JSON dosyası, kullanıcıların ürünleri görüntüleme etkinliklerini içeren olayları barındırır. Her bir ürün görünüm olayı, kullanıcı kimliği, ürün kimliği ve diğer meta verilerle birlikte bir zaman damgası içerir. Bu veri kaynağı, Kafka'ya mesaj olarak gönderilmek üzere kullanılır.
* **PostgreSQL Tabloları:** PostgreSQL veritabanında çeşitli tablolar bulunmaktadır:
  + **ProductViews:** Kullanıcıların ürünleri görüntüleme olaylarını saklar.
  + **Product:** Ürün bilgilerini içerir.
  + **Context: Verilerdeki Source bilgisini tutar.**.
  + **User:** Verilerdeki kullanıcıları temsil eder.

#### 2. Veri İşleme Adımları

##### Gerçek Zamanlı Veri İşleme

1. **Veri Üretimi (View Producer Uygulaması):**
   * product-views.json dosyasındaki her bir ürün görünüm olayı, birer birer Kafka'ya gönderilir. Üretici uygulaması, olayları okur, gerekli dönüşümleri yapar ve Kafka'ya gönderir.
2. **Veri Tüketimi (Stream Reader Uygulaması):**
   * Kafka'ya gönderilen mesajlar, Stream Reader uygulaması tarafından tüketilir. Bu uygulama, Kafka'dan mesajları alır ve JSON formatındaki verileri PostgreSQL veritabanındaki ProductViews tablosuna kaydeder.
   * Tüketici, Kafka'dan veri alırken, her bir mesajı deserialize eder, gerekli doğrulamaları yapar ve PostgreSQL veritabanına kaydeder.

#### 3. Kafka Entegrasyonu

1. **Kafka Üretici Uygulaması:**
   * **Üretim:** product-views.json dosyasından okunan veriler, Kafka'nın belirli bir konusuna (product-views) gönderilir. Bu aşama, her bir ürün görünüm olayının Kafka'ya gönderilmesini içerir.
   * **Bağlantı:** Kafka üreticisi, Kafka broker ile bağlantı kurarak verileri gönderir. Kafka yapılandırması, bağlantı noktası ve diğer ayarları içerir.
2. **Kafka Tüketici Uygulaması:**
   * **Tüketim:** Kafka'nın product-views konusundan veri tüketilir. Tüketici uygulaması, bu verileri okuyarak PostgreSQL veritabanına kaydeder.
   * **Bağlantı:** Kafka tüketicisi, Kafka broker ile bağlantı kurar ve veri almak için konuyu dinler. Veriler alındıktan sonra işlenir ve veritabanına kaydedilir.
3. **Veri Akışı:**
   * Kafka'da üretilen mesajlar, belirli bir konuyu hedef alır. Tüketici, bu konudan gelen mesajları alır ve işleme sürecine dahil eder.
   * Gerçek zamanlı veri akışı, Kafka'daki mesajların sürekli olarak işlenmesini ve veritabanına kaydedilmesini sağlar.

**Özet:**

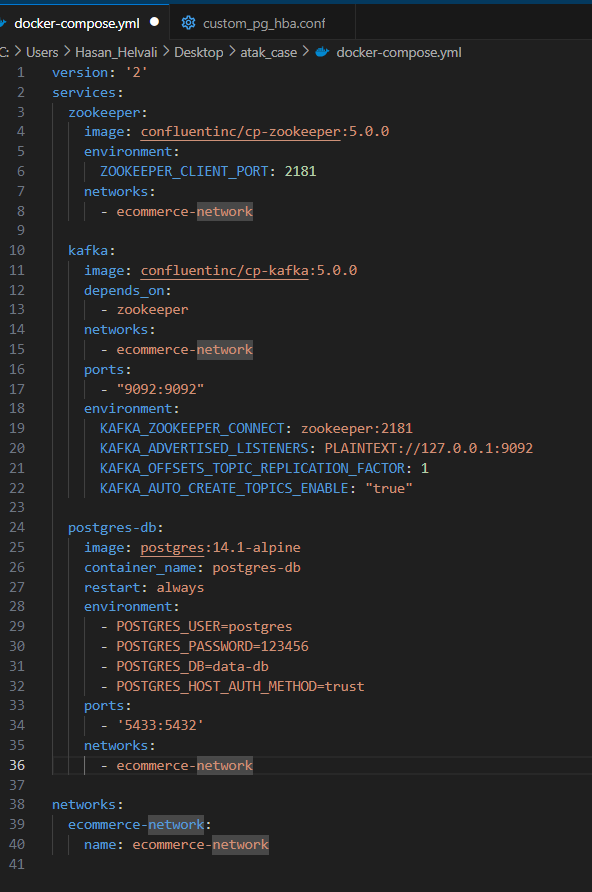
* **Veri Kaynağı:** product-views.json ve PostgreSQL tabloları.
* **Gerçek Zamanlı İşleme:** Kafka üzerinden veri gönderme ve alma.
* **Toplu İşleme:** PostgreSQL'den veri çekme, işleme ve yeniden yükleme.
* **Kafka Entegrasyonu:** Üretici uygulaması Kafka'ya veri gönderir; tüketici uygulaması Kafka'dan veri alır ve veritabanına kaydeder.

Bu detaylar, veri akışının her aşamasını ve Kafka ile entegrasyon sürecini kapsamlı bir şekilde ele almaktadır.

#### J. Dağıtım ve Konteynerizasyon

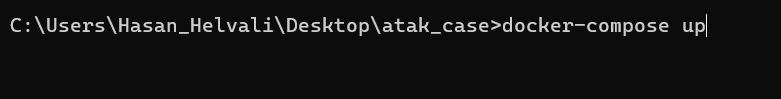
#### Docker Kullanımı

1. **Dockerfile'lar:** Dockerfile'lar, uygulama özelinde Docker imajları oluşturmak için kullanılır. Ancak, sizin durumunuzda sadece Kafka, PostgreSQL ve Zookeeper konteynerlerini çalıştıracağınız için Dockerfile'lara ihtiyaç duymayacaksınız. Bu konteynerler, Docker Compose YAML dosyasındaki servis tanımları kullanılarak çalıştırılacak.
2. **docker-compose.yml Yapılandırması:** Aşağıda verilen docker-compose.yml dosyası, Kafka, PostgreSQL ve Zookeeper konteynerlerinin yapılandırmasını içerir. Bu dosya, bu hizmetlerin tüm yapılandırmasını ve birbirleriyle nasıl etkileşime gireceğini belirler.



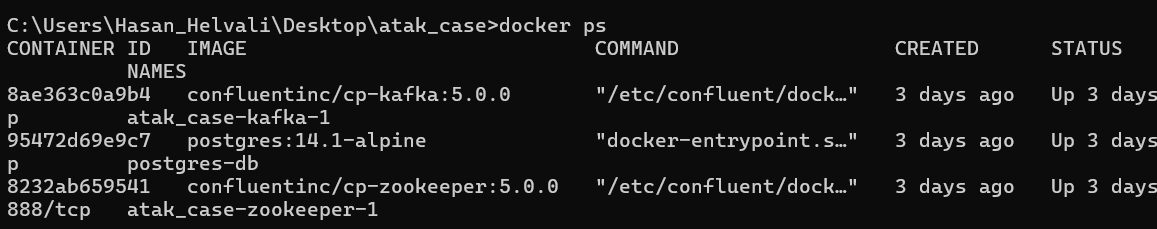
#### Kurulum Talimatları

1. **Docker ve Docker Compose Kurulumu:** Docker ve Docker Compose'un sisteminizde kurulu olduğundan emin olun. Docker ve Docker Compose'u resmi Docker belgelerine göre kurabilirsiniz.
2. **Docker Compose ile Ortamın Kurulması:**
   * Docker Compose yapılandırmasını içeren docker-compose.yml dosyasının bulunduğu dizine gidin.
   * Aşağıdaki komutu çalıştırarak konteynerleri başlatın:

 Bu komut, Kafka, PostgreSQL ve Zookeeper konteynerlerini başlatır ve arka planda çalıştırır.

 **Konteynerlerin Durumunu Kontrol Etme:**

* Çalışan konteynerleri görmek için şu komutu kullanabilirsiniz:

**Konteynerleri Durdurma ve Silme:**

* Çalışan konteynerleri durdurmak ve silmek için şu komutu kullanabilirsiniz:

#### 

#### Dağıtım Adımları

1. **Yerel Ortamda Dağıtım:**
   * Docker Compose yapılandırması ile Kafka, PostgreSQL ve Zookeeper konteynerlerini başlattıktan sonra, yerel ortamda uygulamalarınızı çalıştırabilirsiniz. Uygulama kodunuzu yerel makinenizde çalıştırabilir ve Kafka ile PostgreSQL'e erişebilirsiniz.

Bu bilgiler, Kafka, PostgreSQL ve Zookeeper için Docker ortamını nasıl kuracağınızı ve çalıştıracağınızı anlamanızı sağlar. Uygulama kodlarınız yerel olarak çalıştırılacak, bu yüzden Dockerfile'lar veya konteyner yapılandırmaları bu durumda gerekli değildir.

### **Ek Açıklamalar**

* **Özelleştirmeler:** Docker-compose dosyasında PostgreSql konfigürasyonunda port ayarı 5433:5432 olarak değiştirildi. Çünkü fiziksel olarak PostgreSql yüklü olan bilgisayarlarda 5432 portu bir servis tarafından kullanılır. Burada hata almamak ve docker üzerinden PostgreSql sunucusuna ulaşmak için bu değişikliği yaptım.

#### Proje Değerlendirmesi Ve Sonuç

**Projenin Genel Değerlendirmesi:**

Bu proje, gerçek zamanlı ve toplu veri işleme gereksinimlerini karşılamak amacıyla kapsamlı bir e-ticaret öneri motoru geliştirmeyi hedeflemiştir. Proje, aşağıdaki ana bileşenlerden oluşur:

1. **Clickstream Ürün Görüntüleyici Üretici (View Producer App):** Bu uygulama, product-views.json dosyasından verileri okuyarak Kafka'ya yayınlar. Veriler, belirli aralıklarla Kafka'ya gönderilir ve Kafka, veri akışını sağlar.
2. **Stream Reader (StreamReaderConsumer):** Kafka'dan gelen mesajları tüketir ve PostgreSQL veritabanına kaydeder. Bu uygulama, verinin gerçek zamanlı olarak işlenmesini ve depolanmasını sağlar.
3. **Öneri REST API:** Kullanıcıların son görüntüledikleri ürünleri ve en iyi satış ürünlerini alabilecekleri API'yi sağlar. Ayrıca, kullanıcıların tarayıcı geçmişinden ürünleri silmelerine olanak tanır.

**Karşılaşılan Zorluklar ve Çözümler:**

1. **Veri Doğruluğu ve Formatı:** Kafka mesajlarının doğru formatta olup olmadığını ve veritabanına doğru şekilde işlendiğini doğrulamak zor olabilir. Bu, veri akışının kesintisiz ve doğru bir şekilde gerçekleşmesini sağlamak için dikkatli bir şekilde test edilmelidir. Çözüm olarak, JSON formatının doğruluğunu kontrol eden ve veriyi işleyen testler uygulandı.
2. **Veritabanı Bağlantı Sorunları:** PostgreSQL bağlantısı sırasında yaşanan kimlik doğrulama hataları, doğru bağlantı dizelerinin kullanımı ve doğru yapılandırmanın sağlanması gerekliliğini ortaya koydu. Bu tür hataların çözülmesi için, bağlantı dizelerinin ve veritabanı yapılandırmalarının dikkatlice kontrol edilmesi gereklidir.
3. **Performans ve Ölçeklenebilirlik:** Gerçek zamanlı veri akışını ve toplu verilerin işlenmesini sağlamak için sistemin performansını ve ölçeklenebilirliğini optimize etmek önemliydi. Performans sorunlarını çözmek için, Kafka'nın doğru yapılandırılması ve veritabanı sorgularının optimize edilmesi sağlandı.

#### Gelecek Geliştirmeler

**1. Performans İyileştirmeleri:**

* **Ön Bellekleme:** API yanıt sürelerini azaltmak için ön bellekleme stratejileri uygulanabilir. Örneğin, en çok görüntülenen ürünler veya öneri sonuçları bellek içinde saklanabilir.
* **Veritabanı İyileştirmeleri:** Veritabanı sorgularının ve indekslerin optimizasyonu, özellikle büyük veri kümesi ile çalışırken performansı artırabilir.

**2. Gelişmiş Kişiselleştirme:**

* **Makine Öğrenimi Entegrasyonu:** Kullanıcıların tercihlerini daha iyi anlamak ve daha doğru önerilerde bulunmak için makine öğrenimi algoritmaları entegre edilebilir.

**3. Kullanıcı Deneyimi İyileştirmeleri:**

* **Gerçek Zamanlı Güncellemeler:** Kullanıcıların tarayıcı geçmişini anlık olarak güncelleyebilmeleri ve hızlı yanıtlar alabilmeleri için gerçek zamanlı güncellemeler eklenebilir.
* **API Geliştirmeleri:** API'nin kullanımını daha da kolaylaştırmak için daha kapsamlı dokümantasyon ve örnekler sunulabilir.

**4. Güvenlik ve Veri Koruma:**

* **Veri Şifreleme:** Kullanıcı verilerini korumak için veri şifreleme yöntemleri kullanılabilir.
* **Yetkilendirme ve Kimlik Doğrulama:** API'ye erişim kontrolünü sağlamak için gelişmiş yetkilendirme ve kimlik doğrulama mekanizmaları entegre edilebilir.

Bu geliştirmeler, sistemin performansını artırabilir, kullanıcı deneyimini iyileştirebilir ve veri güvenliğini sağlayabilir. Proje, temel işlevlerini başarıyla yerine getirirken, gelecekteki geliştirmeler ile daha güçlü ve kullanıcı dostu bir çözüm haline getirilebilir.