**Recommendation Team**

**Software/Data Engineer Projesi**

**ÖZET**

Projede bir e-ticaret sitesinin gerçek zamanlı çalışan öneri motoru gerçeklenmesi amaçlanmıştır. Bu amaçla kullanıcıların görüntüleme kayıtlarını barındıran json dosyasındaki veriler alınmış ve timestamp bilgisi eklenerek saniyede bir kafkaya yazılmıştır. Bu sayede sürekli olarak akan, gerçek zamanlı işler için kullanıma hazır bir veri yapısı meydana getirilmiştir. Json dosyasında verilerin kafkaya aktarımı ViewProducerApp isimli uygulama ile gerçekleştirilmiştir. Kafkaya gelen verilen anlık olarak StreamReaderApp uygulaması veya alternatif olarak StreamReaderAppWithoutFlink uygulaması ile alınarak MongoDB veri tabanına aktarılmıştır. Bu aktarım gerçekleşirken öncelikle kafkadan gelen string formatındaki veriler json formatına, json formatından da MongoDB veritabanına eklenebilmesi amacıyla document formatına çevirilmiştir. StreamReaderApp uygulamasına alternatif geliştirilme sebebi paketleme yapısı ile ilgilidir ve detaylı bilgi karşılaşılan zorluklar bölümünde açıklanmıştır. Projenin gerçek zamanlı çalışması için bu kısım yeterli olsa da öneri motorunu oluşturmak için PostgreSQL veritabanındaki verilere de ihtiyaç duyulmuştur. Kişiye özel veya genel anlamda çok satanlar listeleri oluşturabilmek için daha önceki kullanıcı verilerinin düzenli bir yapıda MongoDB veritabanına aktarılması gereklidir. ETLProcessApp uygulaması ile PostgreSQL veritabanındaki ilişkisel tablolarda saklanan veriler öneri motorunun ihtiyaç duyduğu yapıya dönüştürülerek MongoDB veritabanına aktarılmıştır. Bu aktarım işleminin belirli periyotlarda yapılacağı ancak her işlemin farklı bir periyotta gerçekleşeceği öngörülerek seçilen scriptin kullanlabileceği bir opsiyon sunulmuştur. Bu sayede e-ticaret sitesinin tüm ürünlerinin bulunduğu ve kişiye göre özelleştirilmemiş çok satanlar listesi günlük olarak güncellenebilirken, ürünlerin kategorisi değişirse buna yönelik scripti çalıştırabilmek mümkün hale gelmiştir.

RecommendationAPI uygulaması kullanıcının iletişime geçtiği ana uygulamadır. Kullanıcıdan alınana GET ve DEL isteklerine cevap verecek şekilde tasarlanmıştır ve kullanıcı bu uygulamaya gönderdiği istekler ile geçmişte görüntülediği ürünleri listeleyebilir, bu ürünlerden istediklerini geçmişten silebilir ve çok satan ürünleri görüntüleyebilmektedir.

**Fizibilite**

**Yazılım Fizibilitesi**

Kullanılan IDE : Intellij IDEA 2021.1.2

Kullanılan Java versiyonu : 1.8.0\_291

Kullanılan Teknolojiler :

Veritabanları : PostgreSQL, MongoDB

Mesajlaşma : Kafka

Yazılım dili ve frameworkler : Java, Spring Boot, Apache Flink, SQL

Ek yazılımlar : Docker Desktop, Robo 3T 1.4.4, Dbeaver, Postman, Visual Studio Code,   
 Github Desktop

**Donanım Fizibilitesi**

İşlemci : Intel Core i5

Ana Bellek (RAM) : 8 GB

İşletim Sistemi : Windows 10

**İş Gücü ve Zaman Fizibilitesi**

Proje 1 kişi tarafından gerçekleştirilmiş. Projenin gerçeklenmesi için verilen süre 7 gün olarak belirlenmiştir.

**Yasal Fizibilite**

Projenin gerçeklenmesi için kullanılan yazılım dili, programlar ve donanımsal enstrumanlar içerisinde lisans ihlali oluşturan bir durum bulunmamaktadır.

**Sistem Genel Yapı ve Uygulamalar**

**Klasör yapısı** :

Apps : Uygulamalara ait jar ve war dosyaları bulunur

ETLProcessApp : PostgreSQL ve MongoDB arasındaki ETL işlerini yapan uygulamanın kaynak kodu

RecommendationAPI : Uygulamanın Rest API kaynak kodları

StreamReaderApp : Apache Flink ile yapılmış Kafka -> Map -> Sink şeklinde pipeline içeren anlık uygulamanın kaynak kodu

StreamReaderAppWithoutFlink : Kafkadan gelen verileri MongoDB üzerine aktaran uygulamanın kaynak kodu

ViewProducerApp : Json dosyasındaki verileri kafkaya aktaran uygulamanın kaynak kodu

DocumentsAndScripts: Projenin dökümanları ve scriptler

**Kullanım** :

Apps klasöründeki jarları çalıştırmak için :   
 ViewProducerApp :

java -jar ViewProducerApp.jar <json dosyasının yolu> <topic ismi>  
 Örnek : java -jar ViewProducerApp.jar C:\Users\product-views.json mytopic

ETLProcessApp :

java -jar ETLProcessApp.jar <script dosyasının yolu> <script ismi[All,Product,categoryBased]>

Örnek : java -jar ETLProcessApp.jar C:\Users\abc.sql All

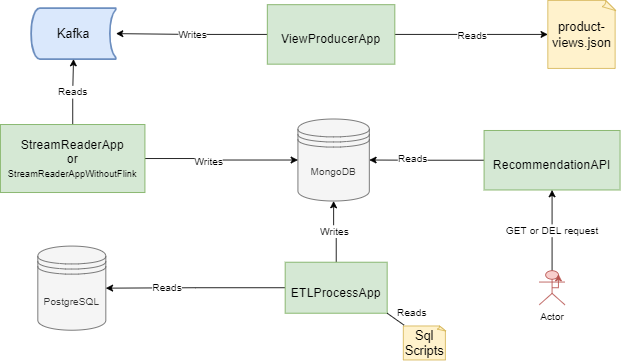
StreamReaderAppWithoutFlink:

java -jar StreamReaderAppWithoutFlink.jar <topic ismi>

Örnek : java -jar StreamReaderAppWithoutFlink.jar mytopic

**Not** : RecommendationAPI\_war.war dosyasını çalıştırmak için sunucu gereklidir.

StreamReaderApp.jar dosyasını çalıştırmak için Apache Flink kurulu olmalıdır.



**ViewProducerApp**

Uygulama dışarıdan parametre olarak json dosyasının yolunu ve kullanılmak istenen kafka topic ismini alır. Json dosyası satır satır okunur. Okunan veri String formatında olduğu için json formatına çevirilir çünkü içeriğine timestamp bilgisinin eklenmesi istenmiştir. Json formatına çevirildikten sonra anlık timestamp bilgisi eklenir ve kafkaya gönderilir. Satırı okumak ile başlayıp kafkaya göndermek ile biten bu döngü her defasında bir saniye bekletilir. Json dosyasının son satırı da kafkaya gönderildiğinde uygulama sonlanmış olur.

**StreamReaderApp**

Uygulama dışarıdan parametre olarak kullanılmak istenen kafka topic ismini alır. Kafkadan gelen veriler Map ve Sink işlemlerini içeren bir pipeline yapısından geçirilerek MongoDB’de recommendationDB

isimli bir veritabanının içerisindeki ProductView isimli collectiona’a aktarılır. Pipeline yapısını gerçekleştirmek ve anlık işlemleri hızlıca yapabilmesini sağlamak için Apache Flink tercih edilmiştir. Uygulama kapatılıncaya kadar kafkadan verilerin gelmesini bekler ve veri geldiğinde işlemeye devam eder.

**ETLProcessApp**

Uygulama dışarıdan parametre olarak işletilmek istenen script dosyasının yolunu ve kullanılacak ETL türünü alır. ETL türü benim tarafımdan oluşturulmuş bir seçenektir ve üç opsiyon içermektedir. Bunlar; All,categoryBased ve Product isimli opsiyonlardır. Aslında tek bir dosya içerisinde üçünün de gerçekleştiği ve seçimin kullanıcıya bırakılmadığı bir yapı da tasarlayabilirdim ancak sadece kategorilerle ilgili bir değişiklik yapılacaksa bütün ETL işlemlerinin tekrardan yapılması gerekmez diye böyle bir opsiyon koydum. Opsiyonlardan All seçeneği PostgreSQL veritabanındaki en çok satılan ürünleri sıralı bir şekilde MongoDB veri tabanına aktarır. CategoryBased seçeneği PostgreSQL veritabanındaki her kategoriyi MongoDB üzerinde bir collection haline getirir ve ürünleri bulundukları kategoriye ekler. Product seçeneği ise tüm ürünleri ve kategorilerini barındıran bir collection oluşturur. Tüm opsiyonlar çalıştırılmalı ve MongoDB Rest API için hazır hale getirilmelidir. ETL işlemleri sonucunca MongoDB üzerinde category-1...category-20, Products ve All isimlerine sahip 22 adet collection oluşmaktadır.

**RecommendationAPI**

Projenin Rest API kısmı bu uygulama ile gerçekleştirilmiştir. Bu kısımda Rest API oluşturmak için Spring Boot kullanılmıştır. Uygulama projenin isterlerine uygun olarak GET ve DEL isteklerini alıp geriye response döndürmektedir. History ve Bestseller iki adet controller sınıf içermektedir.

History sınıfına GET isteği gönderildiğinde kullanıcının daha önce görüntülediği 10 adet ürün geriye döndürülmektedir. Proje isterlerine uygun olarak 5 adetten daha az ürün görüntülemiş kullanıcıların GET istekleri boş liste olarak dönmektedir. History sınıfına DEL isteği gönderildiğinde kullanıcının görüntülediği ürün veri tabanından silinmektedir. Eğer ürün veri tabanında yani kullanıcının geçmişinde mevcut değilse ürünün veritabanında bulunamadığını belirten bir response geriye dönmektedir.

BestSeller sınıfına GET isteği gönderildiğinde kullanıcının geçmişte görüntülediği ürünler incelenir. O kullanıcıya ait geçmişte görüntülenen ürün yoksa ya da hepsi silinmişse tüm ürünlerin bulunduğu koleksiyondan en çok satan 10 adet ürün geriye döndürülmektedir. Eğer kullanıcının geçmişinde ürün bulunuyorsa bu ürünlerin bulunduğu kategoriler tespit edilmekte ve kullanıcının daha önce görüntülediği ürünün bulunduğu kategorinin/kategorilerin çok satan ürünleri geriye döndürülmektedir. Bu aşamada proje isterlerine uygun olacak toplam geriye döndürülen ürün sayısı 10 ile sınırlandırılmıştır. Yine proje isterlerine uygun olarak kullanıcının geçmişinde bulunan ürünlerin kategori sayısı maksimum 3 adet ile sınırlandırılarak bu 3 adet kategoriye ait ürünler geriye döndürülmüştür. Bu kısımda seçilen 3 kategorinin belirlenmesi rastgele olarak gerçekleştirilmiştir.

Geriye döndürülecek yapının sabit bir pattern içermesi sebebiyle User isimli bir model oluşturulmuş ve kullanıcıya response olarak bu model döndürülmüştür.

Uygulama yalnızca MongoDB veritabanı ile iletişim gerçekleştirmektedir. Bağlantıyı gerçekleştirmek için gerekli yapı uygulamanın dbconnnections klasörü altında bulunmaktadır. Yine uygulamanın testleri uygulamanın dizinindeki test klasörü içerisindedir.

**GET ve DEL isteklerini gönderim şekilleri** :

GET localhost:8080/history?userid=<kullanıcı id>

(Örnek : GET localhost:8080/history?userid=user-74 )

DEL localhost:8080/history?userid=<kullanıcı id>&productid=<ürün id>

(Örnek : DEL localhost:8080/history?userid=user-74&productid=product-74)

GET localhost:8080/bestseller?userid=<kullanıcı id>

(Örnek : GET localhost:8080/bestseller?userid=user-74)

**StreamReaderAppWithoutFlink**

Uygulama dışarıdan parametre olarak kullanılmak istenen kafka topic ismini alır. Sonsuz bir döngü oluşturularak kafkadan gelen verilerin uygulama kapatılana kadar dinlenmesi sağlanır. Kafkadan gelen veriler json formatına parse edilir ve timestamp değeri eklenerek MongoDB veritabanına kaydedilir. Bu uygulama **StreamReaderApp** uygulaması için geliştirdiğim bir alternatiftir ve Apache Flink kullanımı içermemektedir. Uygulamaya gerek duyulma sebebi karşılaşılan zorluklar bölümünde açıklanmıştır.

**Veritabanları**

**PostgreSQL**

PostgreSQL veritabanı tarafınızdan bana iletilen docker-compose dosyası ile gelen veritabanıdır. Erişim için;

* PostgreSQL Host: **localhost** Port: **5432**
* **Username:** postgres
* **Password:** 123456

**MongoDB**

MongoDB veritabanını kullanmak için terminal üzerinden **docker run -d -p 27017:27017 --name dbmongo -d mongo:latest** komutunun çalıştırılması gerekmektedir.Erişim için;

* MongoDB Host: **localhost** Port: **27017**
* **Username:**
* **Password:**

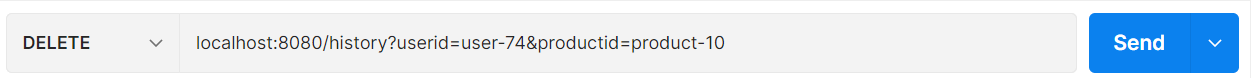
**Ek İşlemler ve Uygulamalar**

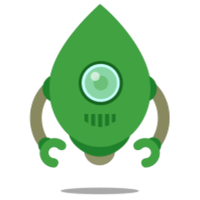


**Postman**

Rest API ile haberleşmek ve GET, DEL isteklerini göndermek için kullanılmış ücretsiz bir programdır.

Proje için aşağıdaki resimlerdeki gibi istekler gönderilebilmektedir.





**Robo 3T**

Docker üzerinde çalıştırılan MongoDB ile bağlantı bu program vasıtasıyla gerçekleştirilmiştir. Ücretsiz bir programdır.



**DBeaver**

Docker üzerinde çalıştırılan PostgreSQL ile bağlantı bu program vasıtasıyla gerçekleştirilmiştir. Ücretsiz bir programdır.

**Karşılaşılan Zorluklar**

**Apache Flink Paketleme**

Apache Flink kullanımı gerçekleştirdiğim StreamReaderApp uygulaması Intellij IDEA üzerinden sorunsuz şekilde çalışmaktadır. Uygulamanın jar dosyası ise java -jar komutu ile çalıştırılamayacak bağlantılar içermektedir. Bu sebep ile Docker üzerine Apache Flink kurulumu gerçekleştirilmiştir. Bu işlem için kullanılan docker-compose dosyası StreamReaderApp klasörü içerisinde mecvuttur. Bu dosya ile oluşturulan Flink cluster üzerine uygulamanın jar dosyası yüklenmiş ve çalıştırılmıştır. Çalıştırılan iş belirli bir süre Running modunda kaldıktan sonra Failed moduna geçmiş ve başarısız olmuştur. Log dosyaları incelenmiş ve sorunun kafka broker bulunamadığı için timeout olup kapandığı belirlenmiştir. Tarafınızdan bana iletilen docker-compose içerisindeki PostgreSQL ve Kafka’yı kullanarak proje gerçeklendiği için kafka üzerinde herhangi bir işlem bu sorun alınmadan önce yapılmamış ve denenmemiştir. Sorun için kafka ayarı yapılmak istense de gerekli ayar gerçekleştirilememiş ve aranan bazı dosyalara erişim sağlanamamıştır. Bu sebeple projenin isterlerini sağlamak amacıyla Apache Flink içermeyen farklı bir çözüme gidilmiş ve StreamReaderAppWithoutFlink uygulaması oluşturulmuştur. Bu uygulama flink bağımlılıkları içermediği için standart bir jar dosyası olarak çalıştırılabilmektedir.

Saygılarımla,

Muhammed Furkan OĞUZ