Eş Zamanlı Sipariş ve Stok Yönetimi Uygulaması

1. Çağla Gök
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
220201060
caglagok369@gmail.com

2. Selma Yıldız
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
220201091
selmayldz726@gmail.com

I. ÖZETÇE

Bu proje, eş zamanlı sipariş yönetimi ve stok güncellemelerini gerçekleştiren bir sistem tasarlamayı amaçlar. Amaç, multithreading ve senkronizasyon mekanizmaları kullanarak aynı kaynağa eş zamanlı erişim problemlerini çözmek ve process, thread, mutex, semafor ve process önceliği gibi kavramların kullanımını kavramaktır.

II. GİRİŞ

ASP.NET Core, React Vite ve Microsoft SQL Server kullanılarak geliştirilen Sipariş ve Stok Yönetim Sistemi'nin tasarım ve uygulanmasını sunmaktadır. JWT tabanlı kimlik doğrulama, SignalR ile gerçek zamanlı bildirimler ve katmanlı mimari kullanılarak sistemin ölçeklenebilirliği, sürdürülebilirliği ve güvenliği sağlanmıştır. Ayrıca, loglama mekanizmaları sayesinde kullanıcı ve admin işlemlerinin kayıt altına alınması sağlanmıştır. Bu çalışmanın amacı, işletmelerin operasyonel süreçlerini dijitalleştirerek verimliliklerini artırmak ve kullanıcı dostu bir çözüm sunmaktır.

Projenin isterleri şu şekildedir:

- Multithreading ve senkronizasyon mekanizmaları kullanarak aynı kaynağa eş zamanlı erişim problemlerini çözmek ve process, thread, mutex, semafor ve process önceliği gibi kavramları kullanmak
- Bütçe yönetimi yapmak
- Dinamik öncelik sistemi kurmak
- Stok yönetimi sağlamak
- Admin ve müşteri sistemi kurmak
- Sipariş sistemini kurmak
- Loglama ve izleme sağlamak
- Ui entegrasyonu
- Veritabanı ilişkileri kurmak

III. YÖNTEM VE İLERLEYİŞ

Sistemin backend kısmı, dört ana katmandan oluşan bir katmanlı mimari ile geliştirilmiştir:

- Denetleyici (Controller) Katmanı: Kullanıcı isteklerini karşılayan ve uygun hizmetlere yönlendiren katmandır.
- Hizmet (Service) Katmanı: İş kurallarının ve mantıksal işlemlerin uygulandığı katmandır.
- Depo (Repository) Katmanı: Veritabanı ile doğrudan iletişim kuran ve CRUD işlemlerini gerçekleştiren katmandır.
- Veritabanı (Database) Katmanı: ASP.NET Core'da Entity Framework Core kullanılarak tasarlanan veritabanı şemasıdır.

JWT (JSON Web Token) tabanlı kimlik doğrulama ile kullanıcıların rollerine göre erişim yetkileri sınırlandırılmıştır. Admin kullanıcıları siparişleri onaylama, ürünleri ekleme ve silme gibi ek yetkilere sahiptir. SignalR teknolojisi kullanılarak hata günlükleri ve sipariş durumu gibi önemli bilgiler gerçek zamanlı olarak iletilmiştir.

AdminController, admin yetkisine sahip kullanıcılar için özel işlevler sağlar ve bu yetki kontrolü IsAdminHelper sınıfı aracılığıyla gerçekleştirilir. Controller, kullanıcıların ve adminlerin ihtiyaçlarına göre düzenlenmiş çeşitli uç noktalar içerir. Admin, sistemdeki tüm kullanıcıları görüntüleme, belirli bir müşterinin detaylarını inceleme ve tüm siparişleri listeleme yetkisine sahiptir. Siparişler, durumlarına göre (ör. başarılı, başarısız) filtrelenebilir ve adminler bekleyen siparişleri işleyebilir. Ayrıca, sistem loglarını görüntüleyerek operasyonel süreçleri izleyebilir.

AdminService, iş mantığını yöneten katmandır. Kullanıcılardan gelen istekleri işler ve veritabanı işlemleri için AdminRepository ile etkileşim kurar. Bu katman, kullanıcı, sipariş ve log verilerini işlemek için veri transfer nesneleri (DTO) kullanır. Örneğin, ProfileDto kullanıcı profili bilgilerini taşırken, CreatedOrderDto sipariş detaylarını içerir. AdminRepository ise veritabanı işlemlerini yürütür ve Entity Framework kullanarak SQL Server üzerinde çalışır. Veritabanından kullanıcı bilgilerini alır, sipariş durumlarını

filtreler ve sistem loglarını döndürür. Loglama mekanizması, kritik olayların kaydedilmesini ve gerektiğinde gerçek zamanlı bildirimlerle paylaşılmasını sağlar.

Yetki kontrolleri titizlikle uygulanır. Admin işlemleri, kullanıcının admin rolünde olup olmadığını kontrol eder ve yetkisiz erişimler için uygun hata mesajları döndürülür. Bunun yanı sıra, tüm işlemler hata yönetimi (try-catch) ile korunmuştur, böylece sistem kararlı bir şekilde çalışabilir. Ayrıca, katmanlı mimari yapısı, Controller, Service ve Repository katmanları arasında net bir ayrım sunarak kodun okunabilirliğini ve sürdürülebilirliğini artırır. Bu tasarım, modern yazılım geliştirme prensiplerine uygun olup, genişletilebilir ve güvenli bir yapı sunmaktadır.

CustomerController, kullanıcıların sisteme kaydolması, giriş yapması, profillerini görüntülemesi ve güncellemesi gibi temel işlemleri yöneten kontrolcü sınıfıdır. Kullanıcılar, Register yöntemi ile sisteme kaydolabilir ve kayıt sırasında müşteri türü "Standart" olarak atanır. Ardından, Login yöntemi ile giriş yapabilir ve JWT tabanlı bir token alarak kimlik doğrulama işlemini tamamlayabilirler. Token, kullanıcının oturum süresince sistemde yetkilendirilmiş işlemleri gerçekleştirmesini sağlar.

Kullanıcıların profillerini görüntüleme ve güncelleme işlemleri için GetProfile ve UpdateProfile yöntemleri sunulmuştur. GetProfile, token üzerinden alınan müşteri adını kullanarak ilgili müşterinin profil bilgilerini getirir. UpdateProfile ise kullanıcının şifre ya da profil fotoğrafı gibi bilgilerini günceller. Ayrıca, kullanıcılar Logout yöntemi aracılığıyla sistemden çıkış yapabilir. Bu işlemler sırasında, token geçerliliği kontrol edilerek yetkisiz erişimlerin önüne geçilir.

API'nin iş mantığı, ICustomerService ve CustomerService sınıfları tarafından yönetilmektedir. CustomerService, müşteri kaydı, giriş, profil görüntüleme ve güncelleme gibi işlemleri doğrudan CustomerRepository ile etkileşim içinde gerçekleştirir. Veritabanı işlemleri, CustomerRepository sınıfı üzerinden yürütülür ve Entity Framework kullanılarak SQL Server üzerinde gerçekleştirilir. Veritabanına yeni bir müşteri ekleme (RegisterAsync), mevcut bir müşteriyi doğrulama (LoginAsync), profil bilgilerini getirme (GetProfileAsync) ve güncelleme (UpdateProfileAsync) gibi işlemleri kapsar.

OrdersController, müşterilerin sipariş oluşturmasına ve sipariş durumlarını görüntülemesine olanak tanır. Kullanıcılar, token tabanlı kimlik doğrulama sistemi aracılığıyla doğrulanır ve yalnızca yetkili kullanıcılar sipariş işlemleri gerçekleştirebilir. CreateOrder yöntemi, bir sipariş oluşturur ve müşteri bütçesini, ürün stoğunu ve siparişin genel geçerliliğini kontrol eder. Eğer müşterinin bütçesi yetersizse veya ürün stoğu sipariş miktarını karşılamıyorsa, sistem bir hata döndürür ve ilgili durum loglanarak gerçek zamanlı bildirimlerle adminlere iletilir. GetOrdersByCustomer yöntemi ise belirli bir kullanıcının sipariş geçmişini getirir.

OrdersService, siparişlerin iş mantığını yöneten katmandır. Siparişlerin fiyatını hesaplama (CalculateTotalPrice), müşteri ve ürün bilgilerini doğrulama ve gerçek zamanlı sipariş işleme gibi işlemleri içerir. Ayrıca, bekleyen siparişleri işleyen bir ProcessOrders yöntemi bulunmaktadır. Bu yöntem, siparişleri öncelik sırasına göre işler ve stok durumu, müşteri bütçesi gibi kriterlere göre siparişlerin durumunu günceller. Örneğin, bir ürünün stoğu yetersiz veya müşteri bütçesi yetersizse, sipariş iptal edilir ve bu durum loglanır. Başarılı siparişlerde ise müşteri profili güncellenir ve müşteri harcaması toplamına eklenir. Ayrıca, müşteri toplam harcamaları belirli bir seviyeyi aşarsa, müşteri türü "Standart"tan "Premium"a yükseltilir.

OrdersRepository, sipariş verilerinin veritabanı üzerinde yönetilmesini sağlar. Sipariş oluşturma, güncelleme ve duruma göre siparişleri filtreleme gibi işlevleri içerir. Ayrıca, ürün stoğunu kontrol etme (CheckStockAvailability) ve stoktan düşme (DeductStock) gibi işlemleri yönetir. Loglama sistemi, kritik olayların kaydedilmesini sağlar ve loglar gerçek zamanlı olarak SignalR kullanılarak adminlere iletilir.

ProductsController, ürünlerin eklenmesi, güncellenmesi, silinmesi ve listelenmesi gibi temel işlevleri sağlar. Bu işlemler sırasında, sadece admin yetkisine sahip kullanıcıların ürün üzerinde değişiklik yapabilmesini sağlamak için yetkilendirme kontrolleri vardır. AddProduct metodu, yeni bir ürün eklemek isteyen adminler için kullanılabilir. Burada, ürünün adı, stoğu, fiyatı, açıklaması ve fotoğrafı gibi bilgiler, bir ürün DTO'su aracılığıyla alınarak veritabanına kaydediliyor. Eğer admin yetkisi olmayan bir kullanıcı bu işlemi denemeye kalkarsa, sistem yetkisiz erişimi reddediyor.

Ürün güncellemeleri için geliştirilen UpdateProduct metodunda da benzer bir yetki kontrol mekanizması bulunuyor. Admin, ürünün mevcut bilgilerini güncelleyebilir ve bu işlem sırasında ürünün geçerli olup olmadığını kontrol edilir. Silme işlemi için ise DeleteProduct metodu kullanılmaktadır. Bu metod, ürünün tamamen kaldırılması yerine, ürünün "silinmiş" olarak işaretlenmesini sağlar. Böylece, sistemdeki diğer işlemlerin etkilenmesi minimuma indirilir.

Ürünlerin detaylarını görüntülemek ve listelemek için GetProductById ve GetAllProducts metotlarını kullanılır. Tüm ürünleri listelerken, yalnızca aktif olan ürünler döndürülür. Ürünlerin stoğunun dinamik olarak güncellenmesi gerektiği durumlarda ise UpdateProductStockAsync metodunu kullanılır. Loglama sistemi sayesinde, ürün ekleme, güncelleme ve silme gibi her işlemde bir log kaydı oluşturulur ve SignalR kullanarak bu logları gerçek zamanlı olarak adminlere iletilir.

Ürün yönetimi iş mantığını ProductsService sınıfı üzerinden yapılır. Bu katman, veritabanı işlemlerini yöneten ProductsRepository ile etkileşim kurar ve ürün ekleme, güncelleme ya da silme işlemleri sırasında admin yetkisini doğrular. Veritabanı tarafında, ürünlerin detaylarını ve stok durumlarını yönetmek için ProductsRepository sınıfını kullanılır. Örneğin, bir ürün güncellenirken, bu ürüne bağlı bekleyen siparişlerin iptal edilmesi gerektiğinde ilgili loglar otomatik olarak oluşturuluyor.

JWT (JSON Web Token) oluşturmak için kullanılan bir yardımcı sınıf olan JwtHelper'ı içeriyor. GenerateJwtToken metodu, bir kullanıcı için JWT üretir. Token, kullanıcının adı (customerName), türü (customerType) gibi bilgileri ve benzersiz bir kimlik (Jti) içerir.

JWT, güvenli bir şekilde oluşturulmak için bir anahtar (Key) ve HMAC-SHA256 algoritması ile imzalanır. Ayrıca, tokenın belirli bir süre sonra geçersiz hale gelmesini sağlamak

için bir son kullanma tarihi atanır. Token oluşturma işlemi sırasında, yapılandırma dosyasındaki ayarları (Issuer, Audience, TokenLifetimeMinutes) kullanır. Bu sınıf, kullanıcının kimliğini doğrulamak ve yetkilendirmek için gereken güvenli bir JWT oluşturur.

Sistemin frontend kısmında yapılanlar:

parseJwt fonksiyonu, bir JSON Web Token'ı (JWT) çözümleyerek içindeki payload verilerini ayrıştırır ve JSON formatında döndürür. Bu işlem sırasında, token içerisindeki veriler doğru formatta değilse veya bir hata oluşursa, bir hata mesajı görüntülenir ve null döndürülür. Fonksiyon, JWT'lerden gelen bilgileri anlamak ve kullanmak için bir araç görevi görür.

CustomerDetail bileşeni, bir müşterinin profil bilgilerini ve sipariş geçmişini detaylı bir şekilde görüntülemek için geliştirilmiştir. Bileşen, useParams ile URL'den alınan müşteri adını kullanarak, API üzerinden müşteri detaylarını ve sipariş geçmişini getirir. Müşterinin profil fotoğrafı, müşteri tipi, bütçesi ve toplam harcamaları gibi bilgiler ekrana yazdırılır. Aynı zamanda müşterinin siparişleri, sipariş tarihi, toplam tutar ve durumu gibi bilgilerle tablo halinde listelenir. Kullanıcılar, geri dönmek için bir "Back" butonu kullanabilir. Hatalar ve yükleme durumları yönetilerek kullanıcıya uygun mesajlar gösterilir.

CustomersAllOrders bileşeni, tüm müşterilerin siparişlerini gerçek zamanlı olarak görüntülemek amacıyla tasarlanmıştır. Bu bileşen, API'den alınan verileri sürekli olarak güncelleyerek sipariş listesini canlı tutar. Siparişlerin durumu, bekleme süresi ve öncelik skorları gibi bilgiler tablo formatında sıralanır. Ayrıca, bir "Process Orders" butonu ile tüm siparişlerin toplu olarak işlenmesi sağlanır. Bu işlem, siparişlerin detaylarının API'ye gönderilmesiyle gerçekleştirilir ve başarılı işlem sonrası kullanıcı bilgilendirilir. Sayfada hata ve yükleme durumları işlenirken, geri dönüş için bir "Back" butonu da bulunmaktadır.

LogPage bileşeni, sistemdeki log kayıtlarını gerçek zamanlı olarak listeleyen bir arayüz sunar. Her saniyede bir API çağrısı yaparak yeni logları alır ve mevcut loglarla birleştirir. Tablo görünümünde sunulan loglar, tarih, log tipi, detaylar, müşteri ve ürün bilgilerini içerir. Bu bileşen, adminlerin sistemdeki kritik olayları ve hataları takip etmesini sağlar. Kullanıcılar, logları inceleyerek sistemin çalışma durumunu ve sorunları gözlemleyebilir. Sayfadan admin paneline dönüş için "Back" butonu kullanılabilir.

ProductAdd bileşeni, yeni bir ürün eklemek için bir form arayüzü sağlar. Kullanıcı, ürün adı, stok miktarı, fiyat, açıklama ve ürün fotoğraf URL'si gibi bilgileri girerek ürünü sisteme ekleyebilir. Form, "Add Product" butonuna tıklanarak sunucuya gönderilir. Sunucuya başarılı bir şekilde ürün eklenirse kullanıcı bilgilendirilir ve ürünler sayfasına yönlendirilir. Geri dönüş işlemleri için bir "Back" butonu da bulunmaktadır. Hata durumları yakalanarak kullanıcıya bildirilir.

OrdersPage Bileşeni kullanıcının sepetindeki ürünleri ve önceki siparişlerini yönetmek için tasarlanmıştır. Kullanıcının sepetteki ürünleri bir tablo halinde gösterilir ve "Order Now" butonuyla siparişler topluca gönderilebilir. Ayrıca, API üzerinden kullanıcının geçmiş siparişleri alınır ve durumu (ör. beklemede, başarısız, başarılı) gösterilir. Sayfa, her saniyede bir güncellenerek gerçek zamanlı bir deneyim sunar. Sepetteki ürünler yerel depolamada saklanır ve sipariş sonrasında temizlenir.

ProductDetail Bileşeni bir ürünün detaylarını kullanıcıya göstermek ve sepete ürün eklemek için geliştirilmiştir. Kullanıcı, ürün fotoğrafı, açıklama, fiyat ve seçilebilir miktar gibi detayları görüntüleyebilir. Ürün sepete eklendikten sonra yerel depolamaya kaydedilir ve kullanıcı sipariş ekranına yönlendirilir. Ürün bilgileri, API çağrılarıyla alınır ve sayfa yüklendiğinde görüntülenir. Geri dönüş işlemleri için bir "Back" butonu da içerir.

ProductDetailAdmin Bileşeni admin kullanıcılarının bir ürünün detaylarını incelemesi için geliştirilmiştir. Ürün bilgileri (ör. ürün adı, stok, fiyat, açıklama ve fotoğraf URL'si) API çağrılarıyla alınır ve yalnızca okunabilir alanlar olarak formda gösterilir. Admin kullanıcıları ürün detaylarını düzenleyemez ancak ürün hakkında bilgi edinebilir. Geri dönüş için bir "Back" butonu bulunmaktadır. Bileşen, admin kullanıcıların sistemdeki ürünleri incelemesini kolaylaştırır.

IV. Deneysel Sonuçlar

Sistemin performans ve ölçeklenebilirliği farklı yük koşulları altında test edilmiştir:

A. Denklem

ÖncelikSkoru = TemelÖncelikSkoru + (BeklemeSüresi × BeklemeSüresiAğırlığı)

- Ortalama Sipariş İşleme Süresi: 150ms
- Eş Zamanlı Kullanıcı Kapasitesi: 500 kullanıcı
- Gerçek Zamanlı Bildirim Gecikmesi: <50ms
- Hata Günlükleri İletim Süresi: 30ms

Semaphore mekanizması sayesinde kaynaklara eş zamanlı erişim sırasında veri tutarlılığı sağlanmış ve çakışmalar önlenmiştir.

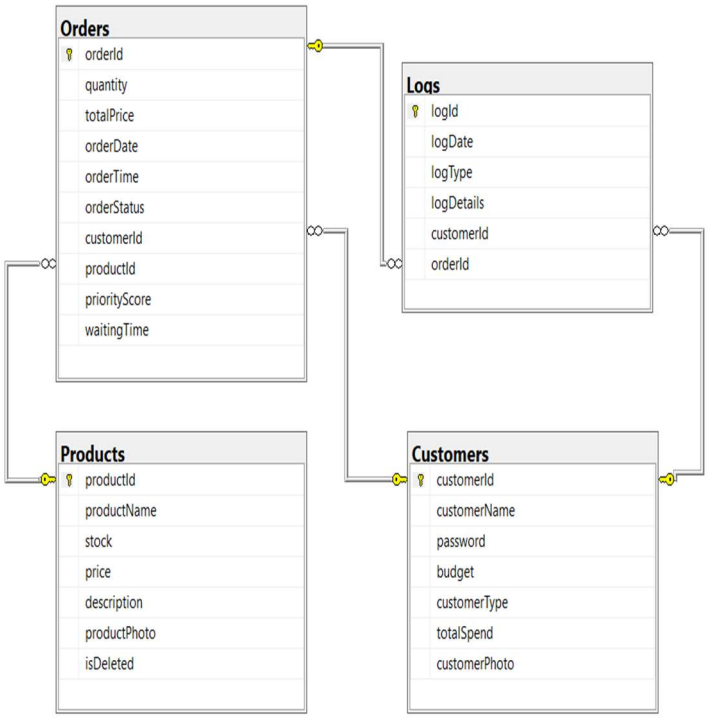
V. Sonuç

Proje, multithreading ve senkronizasyon mekanizmaları kullanılarak veri tutarlılığı ve güvenli erişim problemlerini başarılı bir şekilde çözmüştür. Eş zamanlı sipariş işlemleri güvenilir bir şekilde tamamlanmış ve gerçek zamanlı güncellemeler sağlanmıştır.

VI. Pseudo Kod

```
function CreateOrder(token, orderDetails):
if !JwtHelper.ValidateToken(token):
return "Unauthorized Access"
user = JwtHelper.GetUserFromToken(token)
if!CheckUserBudget(user,orderDetails.price)||
!CheckProductStock(orderDetails.productId,
orderDetails.quantity):
LogError("Order Failed: Insufficient budget or stock")
SignalR.NotifyAdmin("Order Issue", orderDetails)
return "Order Failed"
```

```
orderId = OrdersRepository.AddOrder(user.id, orderDetails)
SignalR.NotifyUser(user.id, "Order Created Successfully")
return orderId
function ProcessOrders():
pendingOrders = OrdersRepository.GetPendingOrders()
for order in pendingOrders:
if CheckProductStock(order.productId, order.quantity) &&
CheckUserBudget(order.userId, order.price):
UpdateOrderStatus(order.id, "Success")
DeductStock(order.productId, order.quantity)
UpdateUserBudget(order.userId, order.price)
SignalR.NotifyUser(order.userId, "Order Processed
Successfully")
else:UpdateOrderStatus(order.id, "Failed")
LogError("Order Failed: Stock or Budget Issue", order)
SignalR.NotifyAdmin("Order Failed", order)
function Login(username, password):
user = UserRepository.ValidateUser(username, password)
if user is null:
return "Invalid Credentials"
token = JwtHelper.GenerateToken(user.id, user.role)
return token
function AddProduct(adminToken, productDetails):
if!JwtHelper.ValidateToken(adminToken)
!IsAdminHelper.CheckAdminRole(adminToken):
return "Unauthorized Access"
productId = ProductsRepository.AddProduct(productDetails)
LogInfo("Product Added", productDetails)
return productId
function LogError(message, details):
LogsRepository.AddLog("Error", message, details)
SignalR.NotifyAdmin("Error Log", message)
function LogInfo(message, details):
LogsRepository.AddLog("Info", message, details)
function ValidateToken(token):
return JwtHelper.ValidateToken(token)
function CheckProductStock(productId, quantity):
stock = ProductsRepository.GetStock(productId)
return stock >= quantity
function CheckUserBudget(userId, amount):
budget = UserRepository.GetUserBudget(userId)
return budget >= amount
```



KAYNAKLAR

[1] https://learn.microsoft.com/tr-tr/training/modules/build-web-api-aspnet-core/

[2] https://tr.react.dev/

[3] https://tr.react.dev/learn

[4] https://ibrahimhkoyuncu.medium.com/java-multithreading-bolum-1-threads-process-thread-pooling-ve-executors-65f732975a42

[5] https://kadirtaban08.medium.com/multithreading-nedir-57aec486b202

[6] https://www.youtube.com/watch?v=agpZsCUllqc

[7] https://www.youtube.com/watch?v=8vh5dmBaVQw

[8] https://furkanalaybeg.medium.com/signalr-nedir-617f5873711d

[9] https://www.gencayyildiz.com/blog/asp-net-core-signalr-serisi-1-signalr-nedir-nasil-calisir/