ÜÇ KATMANLI MİMARİDE
FORM VE VERİTABANI UYGULAMALARI
(Tedarik Zinciri Yönetiminde Sipariş Modülü)

Nihan SEZGİN

Yüksek Lisans Tezi

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Ekim - 2007

ÜÇ KATMANLI MİMARİDE FORM VE VERİTABANI UYGULAMALARI

Nihan SEZGİN

Dumlupınar Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik-Elektronik Mühendisliği Anabilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır.

Danışman : Yrd. Doç. Dr. Ahmet ÖZMEN

Ekim - 2007

KABUL ve ONAY SAYFASI

Nihan SEZGİN'in YÜKSEK LİSANS tezi olarak hazırladığı "Üç Katmanlı Mimaride Form ve Veritabanı Uygulamaları" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

05/10/2007

Üye : Doç. Dr. Kaan ERARSLAN

Üye : Yrd. Doç. Dr. Ahmet ÖZMEN

Üye : Yrd.Doç. Dr. Alpaslan DUYSAK

Fen Bilimleri Enstitüsün Yönetim Kurulu'nun ...../...../…..... gün ve ..... sayılı kararıyla onaylanmıştır.

Prof. Dr. M. Sabri ÖZYURT
Fen Bilimleri Enstitüsü Müdürü

ÜÇ KATMANLI MİMARİDE FORM VE VERİTABANI UYGULAMALARI
(Tedarik Zinciri Yönetiminde Sipariş Modülü)

Nihan SEZGİN

Elektrik-Elektronik Mühendisliği, Yüksek Lisans Tezi, 2007
Tez Danışmanı: Yrd. Doç. Dr. Ahmet ÖZMEN

ÖZET

Bu çalışmada günümüzde işletmelerde önemi gittikçe artan tedarik zinciri yönetimi ile ilgili olarak üç katmanlı mimaride sipariş modülü uygulaması gerçekleştirilmiştir. Bunun için öncelikli olarak sistem mimarileri incelenmiş ve bu çalışma için en doğru mimarinin üç katmanlı mimari olduğu doğrulanmıştır. ASP.net, C# ve SQL Server uygulamaları kullanarak üç katmanda program kodlaması sağlanmış ve bunların avantajlarından bahsedilmiştir. Çalışma özellikle süt sektöründeki işletmelerde Müşteri, Tedarikçi ve Üretici arasındaki bağlantıyı en kısa ve güvenli yoldan sağlamayı hedeflemektedir. Müşteri tarafından sisteme girilen siparişler ve üretici tarafından sisteme girilen üretim miktarları karşılaştırılarak tedarikçi tarafından en uygun üreticinin ürettiği ürünün en uygun müşteriye dağıtımının gerçekleşmesi planlanmaktadır. Böylelikle hem müşteri istediği ürünü kolaylıkla satınalıyor, hem de üretici sadece satabileceği kadar ürünü üreterek hem maliyetlerini azaltıyor hem de ileri vadede planlarını gerçekliyor.

Anahtar Kelimeler: ASP.net, class, e-ticaret, stored procedure, tedarik zinciri yönetimi, üç katmanlı mimari.

FORM AND DATABASE APPLICATIONS IN THE THREE TIER ARCHITECTURE
(Order Module in the Supply Chain Management)

Nihan SEZGİN

Electric&Electronic Engineering, M.S. Thesis, 2007
Thesis Supervisor: Asst. Prof. Dr. Ahmet ÖZMEN

SUMMARY

In this paper order module application in three tier architecture about supply chain management which is becoming more and more important for business administrations is implemented. For this purpose system architectures are examined primarily and it is confirmed that three tier architecture is the most suitable architecture for this research. Program encoding is ensured in three tier using ASP.net, C# and SQL Server applications and the advantages of these were mentioned. This work targets to provide the connection between the Customer, the Provider and the Producer in the shortest and the safest way especially in the administrations in the milk sector. It is planned to implement the distribution of the product that is produced by the most suitable producer to the most suitable customer by the provider comparing the orders that was entered the system by the customer and the production amount entered the system by the producer. By this way the customer can buy the product he wants easily, the producer can reduce the costs by producing only the amount that he can sell and implement his plans for future.

Keywords: ASP.net, class, e-commerce, stored procedure, supply chain management, three tier architecture.

TEŞEKKÜR

Danışman hocam Yrd.Doç. Dr. Ahmet ÖZMEN'e gösterdiği sabır ve anlayış için, aileme de hiçbir zaman eksikliğini hissetmediğim manevi destekleri için sonsuz teşekkürler…

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

1. GİRİŞ

Çalışmada tedarik zinciri yönetiminde üç katmanlı mimari kullanılması ele alınmıştır. Bunun için öncelikli olarak tedarik zinciri yönetiminin tanımına ve önemine ve daha sonra da tedarik zinciri yönetiminde bilgi teknolojilerinin önemine değinilmiştir.

Bir diğer bölümde ise; üç katmanlı mimarinin tanımına, nasıl kullanılmaya başlandığına, diğer mimarilerle arasındaki farklara değinilmiş, neden bu mimari yapısını kullandığımız açıklanmaya çalışılmıştır. Katmanlarda gerçekleşen hizmetlerinde anlatıldığı bu bölümde ayrıntılı olarak üç katmanlı mimarilerin avantajlarına da yer verilmiştir.

Üçüncü bölümde; çalışmamız tedarik zinciri yönetiminde sipariş modülü açıklamış, bu çalışmayı programlarken katmanlarda kullandığımız ASP.NET, C-Sharp teknolojisinden, sınıf yapısı ve veritabanı mantığından bahsedilmiştir.

Son olarak da tedarik zinciri yönetiminde bilgi teknolojilerinin önemi ve üç katmanlı mimari kullanılmasının gerekliliği değerlendirilmiştir.

1.1 Tedarik Zinciri Yönetimi ve Bilgi Teknolojileri

Tedarik zincirinin yönetimi, son yıllarda, işletmeler için çok önemli bir konu olmuştur. Tedarik zincirinin etkin olarak yönetilmesi işletmelerin rekabet avantajı sağlamalarında önemli avantajlar sağlamaktadır. İşletmelerin, tedarik zincirlerini etkin olarak yönetebilmeleri de büyük ölçüde tedarik zincirinin üyeleri arasında bilgi paylaşımının sağlanabilmesine bağlıdır. Teknolojide meydana gelen gelişmeler ve ekonomide ve pazarlarda meydana gelen değişimler, işletmelerin tedarik zincirlerinin yönetilmesinde bilginin önemini arttırmıştır [1].

1.1.1 Tedarik Zinciri Yönetimi (SCM) Sisteminin Tanımı

Tedarik zinciri yönetimi, en erken tedarikçiden nihai müşteriye kadar olan hattaki tüm akışların yönetiminin gelişmiş felsefesi olarak tanımlanabilir. Temel fikir, hattın, gerçek bir sistem olarak anlaşılması amacıyla, bir bütün olarak düşünülmesidir. Hattaki tüm üyeler, dolaylı ya da dolaysız olarak diğer hat üyelerini ve hat performansını etkiler. Bir şirketin tedarik zinciri; hammadde yarı mamülleri tedarik ettikleri yerlerden bitmiş ürünleri dağıtım kanallarıyla nihai tüketiciye kadar ulaştırması sırasında değer yaratan bütün unsurlardır. Bu tanımı tüketici açısından ifade ettiğimiz taktirde, tedarik zinciri bir ürün veya hizmet için talepleri yerine getirmek üzere gereken değeri meydana getiren aşamaların veya unsurların tamamıdır.

Tedarik zinciri yönetimi; firmanın iç kaynaklarının entegre edilerek dış kaynaklarla etkin biçimde çalışmasının sağlanmasıdır. Amaç, geliştirilmiş üretim kapasitesi, piyasa duyarlılığı ve müşteri tedarikçi ilişkileri gibi firmanın tüm performansını oluşturan değerlerin arttırılmasıdır. Tedarik zinciri yönetimi, hammaddelerin edinilmesinden imalat ürünlerine ve buradan da tüketiciye işlenmiş ürünlerin dağıtımına kadar tüm tedarik zinciri boyunca bilgiye dayalı karar almamıza olanak vermektedir.

Tedarik zincirinde ürün akışı, hammadde kaynakları, imalatçı, dağıtıcılar, tüketiciler vs. arasında, her iki yönde de akan arz talep işlem bilgisi tarafından denetlenmektedir. Böylece, tedarik zinciri yönetimi en basit haliyle, tedarik zinciri yönetimi sisteminin tümüne odaklanır. Bu, başarılması gereken önemli ve zor bir amaçtır. Çok az organizasyon, firmalarında çeşitli fonksiyonlar, takımlar ve diğer birimler arasındaki etkileşimi kavrayabilmiştir. Uygulamada, tedarik zinciri yönetimi, firmanın daha çok kendisine odaklandığı geleneksel yaklaşımdan farklı olarak tüm tedarik zinciri üyelerine odaklanır [2].

Özet olarak **Tedarik Zinciri Yönetimi (Supply Chain Management),** doğru mal veya hizmetin, doğru zamanda, doğru yerde, doğru fiyatta ve mümkün olan en düşük maliyetle müşteriye ulaşmasını sağlayan, malzeme, bilgi ve para akışının entegre yönetimidir. Mal veya hizmetin müşteriye ulaştırılması sürecinde mal yada hizmetin üretimden tüketimine kadar yapılacak işlemlerin hangi şekillerde yapılacağı, bunların yapılırken hangi yolların izleneceği, engellerin nasıl aşılacağı ve bu konularda yapılacak yeniden yapılanmayı kapsamaktadır. Diğer bir deyişle Tedarik Zinciri Yönetimi, doğru mal veya hizmetin, doğru zamanda, doğru yerde, doğru fiyatta ve mümkün olan en düşük maliyetle müşteriye ulaşmasını sağlayan, malzeme, bilgi ve para akışının entegre yönetimidir. Kârın maksimize edilebilmesi için maliyetlerin düşürülebilmesi ve rekabet avantajı sağlanabilmesi için tedarik zinciri yönetimi güçlendirilmeli ve müşteri ihtiyaçlarına göre farklılaştırılmalıdır [3].

Etkin bir SCM için firmalar kendilerine aşağıdaki soruları sorarak başlayabilirler:

- Firma için önemli olan değer, maliyet, fayda ve risk unsurları nelerdir?
- Hangi mal veya hizmetleri satın almalı hangi ürünleri üretmeliyiz?
- Hangi tedarik zinciri ortaklarımız sistemimize en güçlü desteği veriyor?
- İnternet teknolojileri sipariş alma, sipariş karşılama ve müşteri hizmetleri
- fonksiyonlarını nasıl destekliyor?

Etkin bir tedarik zincirinin avantajları ise;

- Daha düşük stok,
- Daha yüksek verimlilik,
- Daha çevik bir yapılanma,
- Daha kısa tedarik zamanı,
- Daha yüksek kar,
- Daha fazla müşteri bağımlılığı,
- Ayrı organizasyonları birleşik olarak hareket eden bir sisteme dönüştürebilmesi

olarak sıralanabilir.

1.1.2 Tedarik Zinciri Yönetiminde Bilgi Sistemlerinin Önemi

Teknolojik gelişmelerle birlikte, işletmeler için bilgi, ürettikleri ürünler ve hizmetler kadar önemli bir konuma gelmiştir. Tedarik zincirini, işletmeler için bir rekabet avantajı durumuna getirebilmede ilk adım, tedarik zincirinin üyelerinin açık bir biçimde bilgi paylaşımına istekli olmalarıdır [4]. İşletmeler, bilgi paylaşımına, güçlerini kaybetmelerine neden olacağını düşünmelerinden dolayı, olumlu bakmayabilmektedirler. Bu anlayış, tedarik zincirinde bilgi akışında sorunlara neden olmaktadır [5].

Tedarikzincirindeki üyeler arasında bilgi paylaşımı için bilgi teknolojilerinden yararlanılması, sanal bir tedarik zincirinin oluşmasına neden olmaktadır. Sanal tedarik zinciri fiziksel ürünlere dayalı olmayıp, bilgi akışına dayalıdır. Tedarik zincirindeki tüm üyelere doğru bilgilerin zamanında ulaştırılmasını sağlayacak bilgi sistemleri tasarlanmadan tedarik zincirinin etkin olarak yönetilebilmesi de mümkün olmayacaktır. Tedarik zincirindeki üyelerin işbirliği içerisinde olmaları, bu işletmelerin faaliyetlerinin etkinliğini artırabilecektir. İşletmelerin bilgilere gerekli olduğunda hızlı bir biçimde ulaşabilmeleri, işletmelerin, müşteri beklentilerine daha duyarlı olmalarını ve müşterilerin taleplerini rakiplerine göre daha hızlı karşılayabilmelerini sağlamaktadır.

Tedarik zincirinin üyeleri arasında bilgi akışı, malzemelerin ve ürünlerin fiziksel akışına göre daha öncelikli olarak gerçeklemesinden dolayı, stokların azaltılması ve kaynakların daha etkin olarak kullanılması olanağını artırmaktadır. İşletmeler, sipariş büyüklüğünü azaltırken sipariş sıklığını artırmaya yönelmektedirler. Bu da malzeme taşıma faaliyetlerinin

artmasına neden olmaktadır ve bağlı olarak işletmeler arasında bilgi akışı da önem kazanmaktadır.

İşletmeler ürünlerin tasarımını tedarikçileriyle işbirliği içerisinde belirlemeyi tercih edebilmektedirler. Böylece de ürünler, dünyanın farklı köşelerinde bulunan işletmelerin birbirleriyle işbirliği içerisinde çalışmaları sonucunda üretilebilmektedir. Bu işbirliğinin başarısı, işletmelerin, fiziksel sınırlarının dışındaki işletmelerle etkin olarak koordinasyonunu sağlayabilmesine büyük ölçüde bağlıdır.

Bilgi teknolojileri, tedarik zincirinin yönetiminde planlama ve uygulama aşamalarında kritik role sahiptir. Bilgi teknolojilerinin, tedarik zincirinde stratejik düzeyde planlama, taktik düzeyde planlama ve işlemsel düzeyde planlama olmak üzere üç alanda önemli etkileri bulunmaktadır.

- Stratejik düzeyde planlama, tedarikçilerin optimum sayısının ne olacağı, dağıtıcıların belirlenmesi vb. konuların saptanmasını kapsayan tedarik zinciri ağ tasarımını içermektedir.

- Taktik düzeyde planlama, ağ üzerinde ürünlerin ve hizmetlerin akışının en iyi hale gelmesini içeren tedarik planlamasını kapsamaktadır. Bu düzeydeki kararlar, hangi işletmelerde hangi ürünlerin ve ne miktarda üretileceği ve hammaddelerin nerelerden tedarik edileceği gibi konuları kapsamaktadır.

- İşlemsel düzeyde planlama, günlük veya saatlik bazda tüm işletmelerde üretim planlarının yapılmasını içermektedir .

Bir ağ üzerinde bilgi paylaşımının işlevselliğinin üç farklı türü bulunmaktadır. En basit tür olarak bazı bilgilerin bir yerden baka yere iletilmesini sağlayan basit veri iletimidir. Bu genelde talebe ilişkin bilgilerin paylaşımı biçiminde olmaktadır. Diğer bir tür, sadece mesajların iletilmesinin dışında bazı bilgilerin ortak kullanımına olanak sağlanmasıdır [6]. Üçüncü tür de ise, yetkili kişilerin bir bilgisayardaki programlara ulaşabilmelerine ve bu programları kullanabilmelerine olanak sağlanmaktadır. Tedarikçi üyeler arasında bilgi paylaşımı sağlandıktan sonra kaynakların ve ilerin tedarik zinciri üyeleri arasında değiş tokuşu gerçekleştirilebilmektedir. Bu değiş tokuşun başarılabilmesinde, tedarik zinciri üyeleri arasında ortak faaliyetlere ilişkin bilgilerin paylaşımı yeterli olmamaktadır. Aynı zamanda, tedarik zincirindeki işletmeler, diğer işletmelere göre avantajlı durumda olduğu temel yeteneklerine ilişkin bilgileri de paylaşmaya istekli olmalıdırlar [7].

1.2. Tedarik Zinciri Yönetimi ve E-ticaret

İşletmelerin, elektronik çağa uyum sağlamaları için işletmelerin tedarik zincirinin yeniden yapılandırılmasında E-ticaretin önemli desteği olacaktır. E-ticaret ile birlikte, aracıların sayısında önemli azalmaların olacağı ve işletmelerle müşterilerin direkt olarak iletişime geçebileceği tahmin edilmektedir. E-ticaret ile tedarik zincirindeki işletmeler daha etkin olarak bütünleştirilebilecek ve doğru bilgiye hızla ulaşılabilecektir. E-ticaret ile tedarik zincirinde belirsizlikler azaltılabilecek ve tüm üyelerin gerekli bilgiye zamanında ulaşabilmeleri mümkün olacaktır [8].

Bilgi yönetimi, ürünlerin tedarik zincirinde başarılı iletiminde önemli bir faktördür. Bilgi yönetimi aşamalara bölünerek daha kolay incelenebilir: Öncelikle şirketler ürünlerine olacak talebi tahmin etmek zorundadırlar. Bu tahminlere dayanarak üretim planları hazırlanır ve tedarikçiler bu planı uygulamak için gereken ara ürün miktarlarını tedarikçi firmalara bildirirler. Tedarikçiler de kendi üretim planlarını hazırlar ve ara ürünlerin dağıtımını planlarlar. Üretim tamamlandığında firmalar ürünün gönderilmesi ve gönderilen ürünün takip edilmesi gibi faaliyetlerle de ilgilenmek zorundadır. Son olarak da firmalar ürünün satış miktarlarını inceleyerek gelecekteki talebi tahminleyerek talepten fazla üretim yapmaktan ve arzın altında üretim yapmaktan kaçınırlar.

Bu aşamaların tümünde büyük miktarlarda verinin toplanması, tasnif edilmesi, güncellenmesi ve ortak firmalara iletilmesi gerekmektedir. Bu bilgi akışının elektronik olmayan yollardan yapılması genelde daha yavaş, hata yüzdesi yüksek, ve güncellenmesi zordur. Bu nedenler, firmaları bilginin bilgisayarlar aracılığıyla elektronik ortamdan iletimi için daha verimli yollar aramaya itmektedir.

Bilgi paylaşımının bu kadar önemli olduğu bir alanda bilginin güvenilir ve etkin bir şekilde kullanılması ve ulaştırılması da önemlidir. Ve bunun için kullanılan en etkin yöntemlerden biri de 3 katmanlı mimaridir.

Projemizde amaç; 3 katmanlı mimariyi kullanarak temel bir sipariş modülü ile tedarik zinciri yönetimine katkıda bulunmaktır.

2. ÜÇ KATMANLI MİMARİ

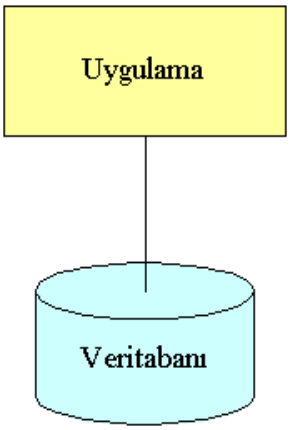
Bundan birkaç sene öncesine kadar, şirket bazındaki yazılımlarda en çok kullanılan teknolojiler istemci-sunucu alanındaydı. Bu eğilim, her geçen gün, dağıtık sistemlerin sundukları avantajlar sebebiyle çok katlı (n-tier ya da multi-tier) mimarilere doğru kayıyor.

İstemci-Sunucu mimari terimi, özel bir amaç veya fonksiyon için istemci bir programın ayrı sunucu bir program ile (muhtemelen ayrı bir makinedeki) ilişki kurduğu ağ sisteminin genel tanımıdır. İstemci, sunucu tarafından sağlanan hizmeti talep eden durumundadır. Birinci kuşak sistemler 2 katmanlı mimarilerdir, ki istemciler kullanıcıya grafik tabanlı arayüzler sunarlar ve kullanıcının girişini yaptığı veri ve işlemleri farklı bir makinede çalışan veritabanı sunucularının taleplerini yerine getirmek için kullanırlar. Bu tip mimaride uygulama mantığı istemci uygulamasına bağımlıdır ve istemci-sunucu etkileşiminde arabuluculuk yapmak üzere "ağır" bir ağ işlemi gereklidir.

Yeni kuşak istemci-sunucu mimarisi yaklaşımında bu bir adım ileri götürülerek bir orta sıra daha eklenerek 3 katmanlı mimari elde edilir. Genelde, uygulama mantığının bölündüğü yerde, istemci-sunucu yapı N katmanlı mimari şeklinde görülebilir. Bu bizi daha hızlı ağ iletişimine, daha yüksek güvenilirliğe ve daha yüksek sistem performansına ulaştırır.

2.1. İki Katmanlı Mimari

Klasik istemci-sunucu sistemler 2 katlı (2-tier) mimariler üzerine kurulmuşlardır. Bu tür sistemlerde, uygulama direk olarak veritabanı sunucusuyla ilişkidedir. Genel olarak, üretilen işin büyük bir bölümü istemci tarafından yapılırken, çoğu zaman sunucu sadece veritabanı sunucusu görevi görür. Bu, uygulamanın kabul edilebilir bir hızla çalışması için güçlü bir istemci donanım gereksinimini ve bilgisayar ağı olanaklarının gereğinden fazla kullanımını doğurur. Bunun nedeni, iş mantığının işlenmesinin büyük bir bölümünün istemci tarafından yapılıyor olması ve uygulamanın, her gerekli veri parçası için, veritabanına bağlanıyor olmasıdır. Deneyimli okuyucular, şu anda mutlaka "bilgi işlemenin bir bölümü veritabanı sunucusuna kaydırılabilir" diyorlardır: İş mantığının istemci yerine veritabanına kaydırılması ise veritabanına özel, yeniden kullanılması çok zor olan bir uygulama yaratır (genelde stored procedure ler kullanarak).



Şekil 2.1. İki Katmanlı Mimari

2 katmanlı sistemlerde ortaya sıkça çıkan başka bir problem de uygulamanın bakımı ve yapılan değişikliklerdir. Şekil 2.1'de görüldüğü gibi iki katman Uygulama ve Veritabanı katmanlarıdır. İstemcilerin iş mantığının bir bölümünü içlerinde bulunduruyor olmaları sebebiyle, iş mantığında yapılmak istenen küçük bir değişiklik bile tüm istemci bilgisayarlara yeniden yükleme yapılmasına neden olur. Bu işlemin otomatik hale getirilmesi bile her bilgisayarın güncelleştirilmesi gerçeğini ve bir takım şirket içi kullanıcı problemlerini ortadan kaldırmaz (Genelde kullanıcılar yeni gelen uygulama değişikliklerini hoş karşılamazlar veya işleri gereği buna hazır olmayabilirler, v.b.).

İstemci-sunucu sistemlerinin büyük bir dezavantajı da, uygulamanın kullanımının artması halinde (kullanıcı sayısının artması halinde), sistemin ölçeklenirliğinin (scalability) artamamasıdır. Veritabanı sunucusunun donanım kapasitesi dolduğu anda tek çözüm, onu daha güçlü bir bir sunucuyla değiştirmektir. Bu da pahalı bir operasyondur. Daha sonra da göreceğimiz gibi, dağıtık bir mimaride aynı gereksinme doğduğu takdirde, sisteme yeni bir bilgisayar eklemek yeterli ve daha ucuz bir çare olur.

2.1.1. İki Katmanlı Mimarinin İncelenmesi

İki katmanlı yapının en genel gerçekleştirimi uygulama mantığını istemciye yerleştirip "thick" istemci- "thin" sunucu mimarisi yapmaktır. Bu şekilde veritabanı basitçe sorgu sonuçlarını raporlar. Tipik olarak bu, dinamik SQL aracılığı ile ODBC türü bir çağrı düzeyi arayüzü (call level interface) kullanılarak gerçekleştirilir. İki katmanlı mimaride geçerli olabilecek bir başka yaklaşım da veritabanı sunucusunda saklanan yordamları çağıran "thin" istemci-"thick" sunucu yaklaşımıdır. Her 2 durumda da işlemleri yürütmek için SQL-Net gibi

uzak veritabanı taşıma protokolleri (remote database transport protocols) kullanılır. Sorgu başına düşen ağ "ayakizi" (footprint) çok geniştir, böylece ağın etkili bant genişliği ve ağı etkili kullanabilen kullanıcı sayısı azaltılır. Sadece ağ işlemlerinin boyutu değil, sorgu işleme hızı da bu ağır iletişim dolayısıyla yavaşlamıştır. Bu mimariler kritik görevli uygulamalar için tercih edilmez.

İki katmanlı mimari gerçekleştirimini uygulayan uygulama araçları içinde PowerBuilder, Delphi, Visual Basic ve Uniface sayılabilir. Bunlar "thick" istemciler meydana getirirler. FileNet ve C-image gibi satıcı firmalar da böyle "thick" istemci iki katmanlı mimarisini içlerine alırlar. "thick" sunucu yaklaşımında saklı yordamlar kullanmak performans elde etmek için daha etkilidir. Çünkü ağın ayakizi hala ağır olmasına rağmen, "thick" istemciden daha hafiftir. Başka bir deyişle, bu tip yordamlar tek bir satıcı firmanın yordamsal işlevselliğine dayandığı gibi özel alışkanlıkları ve kodlamayı da vurgular.

Alternatif N-katmanlı istemci-sunucu mimarisinde ağ ayakizinde önemli bir indirgeme ve performansta artış muhtemeldir. Örneğin, 3 katmanlı bir yapı elde etmek için bir "thin" istemci ile bir "thin" sunucu arasına bir orta katman ilave edilebilir. Sunucu orta katman ile DLL, API veya RPC türü standart protokoller aracılığı ile ilişki kurar. Orta katman, istemci çağrılarını veritabanı sorgularına ve dönüşte veritabanındaki veriyi istemcinin kullanabileceği şekle dönüştürerek, uygulama mantığının çoğunu içerir. Orta katman veritabanı ile aynı host'a yerleştirilirse, gömülü arayüz aracılığı ile veritabanına sıkıca bağlı hale gelir. Bu da bizi çok iyi kontrol edilmiş ve çok yüksek performanslı etkileşime götürür. Böylece SQL-Net, ODBC veya diğer CLL'lerin ağı aşırı yüklemesinin önüne geçilmeye çalışılır. Ayrıca, orta katman işlem gücü yeteneği elde etmek üzere üçüncü bir host'a dağıtılabilir.

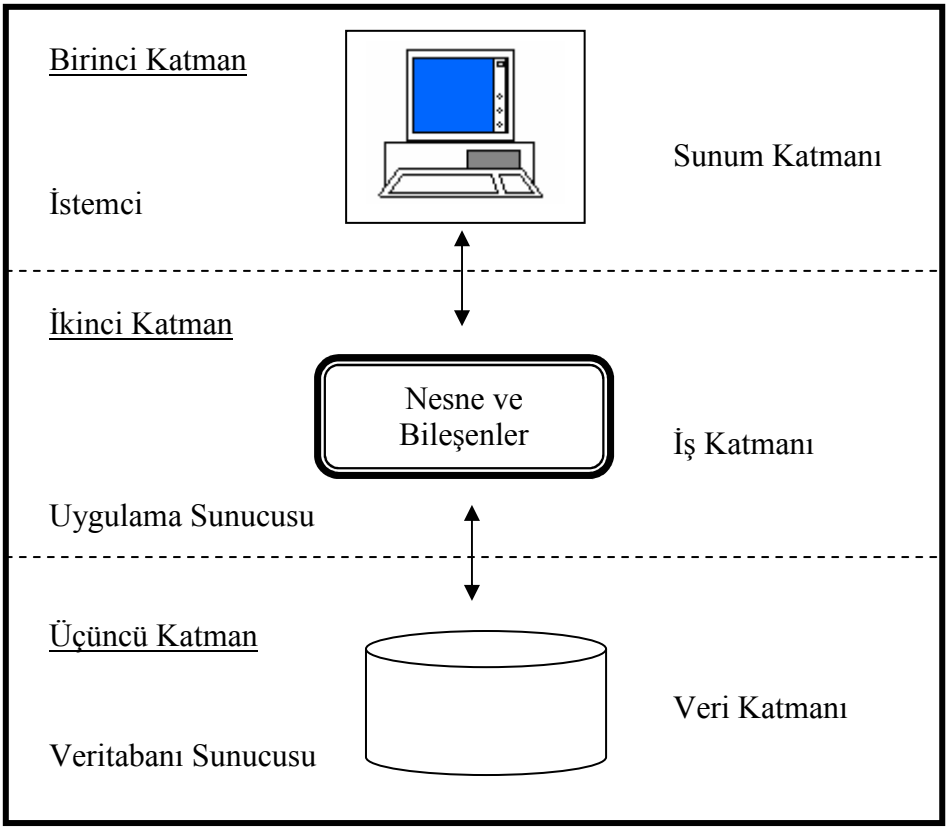
2.2. Üç Katmanlı Mimari

3 katmanlı mimari, bir istemci-sunucu mimarisidir. Adından da anlaşılacağı gibi 3 katmana (Şekil 2.2) sahiptir.

1. Sunum Katmanı (Presentation Layer)

2. İş Katmanı (Business Layer)

3. Veri Katmanı(Data Layer)



Şekil 2.2. Üç katmanlı mimari

Tipik bir üç katmanlı uygulamada, uygulama kullanıcısının iş istasyonu; kullanıcıya arayüz sağlayan (GUI) programı, uygulamaya özel giriş formlarını ve etkileşimli pencereleri içerir. (Yerel veriler veya iş istasyonunun kullanıcısına özel veriler de yerel sabit diskte saklanır).

İş mantığı, bir yerel alan ağ sunucusu veya başka bir paylaşımlı bilgisayarda bulundurulur. İş mantığı, iş istasyonlarından gelen istemci istemlerine sunucu olarak karşılık verir. Hangi verilerin gerekli olduğuna (ve nerede bulunduğuna) karar verir ve bir mainframe de bulunan 3. katman programı ile (bu kez) istemci olarak ilişki kurar.

3. katman, veritabanını ve bu veritabanına okuma ve yazma erişimini yöneten programı içerir. Bir uygulamanın oluşumu daha karmaşık olmakla birlikte, bu 3 katmanlı görüş büyük ölçekli bir programdaki parçalar için uygun bir düşünce yapısı oluşturur [11].

Üç katman bir uygulama istemci-sunucu modelini kullanır. Her 3 katman, farklı programlama dilleri ile çalışan farklı takımlar tarafından paralel olarak geliştirilebilir. Bir katmanın programı, diğer katmanlar etkilenmeden değiştirilebilir veya taşınabilir. Böylece, bir

kuruluş için yeni ihtiyaçlar doğduğunda değişiklik yapmak kolay olur. Var olan uygulamaların tamamı veya bazı kritik kısımları geçici veya sürekli olarak saklanabilir ve eklenen yeni bir katmanın içine katılabilir.

2.3. Katmanlarda Sağlanan Hizmetler

Bu mantıksal katmanlar Sunum, Uygulama ve Veri hizmetlerine ayrılmıştır.

2.3.1. Sunum Hizmetleri

Uygulamanın kullanıcıya veri sağlayan kısmına tekabül eder. Ek olarak, kullanıcının veri ile etkileşimini sağlayan yapıyı da sağlar. Kısaca, sunum hizmeti kullanıcı arayüzünü tanımlar ve onunla etkileşir.

2.3.2. İş Hizmetleri/Nesneleri

Uygulama hizmetlerinin bir kategorisidir. Bu hizmet, bir organizasyonlar işi işlevlerini ve gereklerini içerir. Bu kurallar, bir kuruluştaki günlük işlerin ortaya çıkma adımlarından türetilir. Kurallar gelen bilginin geçerli bir tip ve formatta olduğundan emin olmak için kullanılan geçerlilik kuralları veya bir işlemi tamamlamak için takip edilen uygun iş adımlarından emin olmak için kullanılan işlev kuralları olabilir.

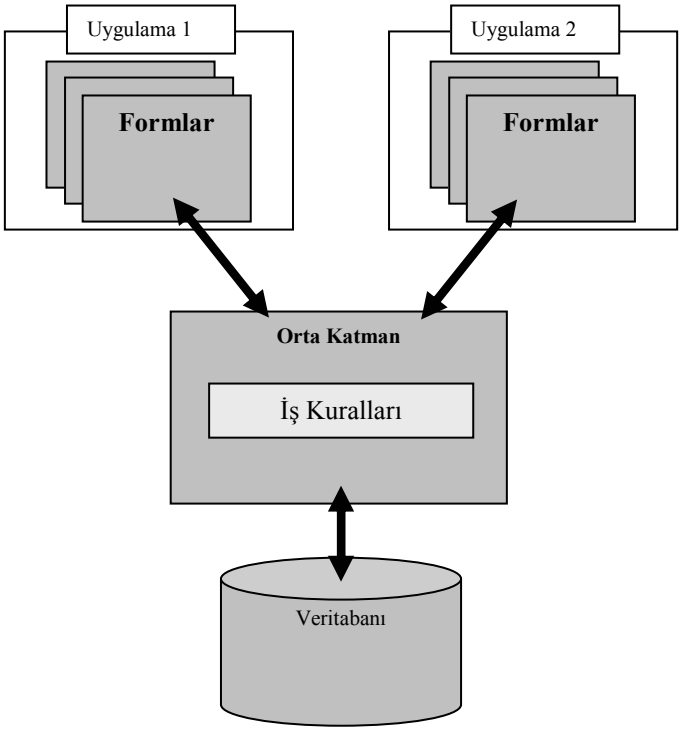
2.3.3. Veri Hizmetleri

Konumundan bağımsız olarak veriye erişimi içerir.

2.4. N-katmanlı Mimariler

Orta katman çeşitli hizmet tiplerine bağlantı sağladığında ve bu servisleri bütünleştirip istemci ile veya birbirleri ile birleştirdiğinde üç katmanlı mimari n-katmanlı mimariye genişletilebilir. N-katmanlı bir sistem, uygulama mantığını çeşitli host'lar arasında paylaştırarak da oluşturulabilir. Böyle bir durumda dağılmış işlevselliğin içeriği tekrar kullanım ve güvenilirlik gibi önemli avantajlar sağlar.

2.5. Üç Katmanlı Mimarinin Avantajları



Şekil 2.3. Üç katmanlı mimari temel akış yapısı

Bir yazılım mimarisi her probleme uymaz. Farklı yaklaşımlar farklı alanlar için uygun olur. Bir mimari kendi alanına uygun olması ile değerlendirilir. Bu alanlar:

- İş yönetimi işlemleri,
- Orta, büyük ölçekli uygulamalar,
- Uygulama geliştirme,
- İstemci/sunucu,
- Açık sistemler.

Bir üç katmanlı mimari katmanlar arası etkileşimi sayesinde (Şekil 2.3) aşağıdaki kritik alanlarda önemli avantajlar sunar:

- Kontrol,

- Güvenilirlik,
- Ölçülebilirlik ve performans,
- Esneklik, büyüme, değişme,
- Standartlar.

2.5.1. Kontrol

İstenen özellikler:

- Kararlı çevre,
- Bilgi sistemleri yönetim kontrolü,
- Denetlenebilirlik,
- Bilinen, tahmin edilebilir kullanım,
- Güvenlilik.

Üç katmanlı yapı orta-katmanı ile iş yönetimi için bir odak noktası oluşturur. Bu, uygulama için bir merkezi kontrol sağlar. Orta katmandaki sunucuda çalışan programlar, bilgi sistemlerinin kontrolü altında, güvenlidir. Bunlar test edilebilir, denetlenebilir.

2.5.1.1. İçerilen Hizmetler

Bu hizmetler, veri katmanı ile uygulama programları (1. Katman ile 3. katman) arasında bir "firewall" olarak hizmet eder. Bu, her zaman uygulamanın güvenilirliğini arttırır. Eğer kullanıcı iş ortağında bulunuyor ise, "firewall" etkisi özellikle önem kazanır.

2.5.1.2. Yazılım Dağıtımı

İş mantığı, sunucu üzerinde olduğundan, yazılım dağıtımı kabusları en aza indirilmiştir. Uygulama sunucusundaki programları günlemek, binlerce kişisel bilgisayardakini günlemekten çok daha kolaydır.

Kullanıcı arayüz kodu değiştiğinde, dağıtılması gerekmez. Uygulama sunucusu dağıtım noktası olarak kullanılabilir. "firewall" etkisi kişisel bilgisayarların dağıtım sorununu azaltır.

2.5.2. Güvenilirlik

İstenilen özellikler

- Tahmin edilebilir davranış,
- Yüksek kullanışlılık: donanım, yazılım,
- İşlem bütünlüğü,
- Düşük hata oranı,
- Sürprizlerin oluşmaması.

Birçok sebepten dolayı (Ölçülebilirlik ve Performansta anlatılıyor), üç katmanlı mimari kullanıldığında, cevap zamanları daha tahmin edilebilirdir.

Bir uygulama sunucusu (donanım olarak), bir masaüstü makinesinden daha güvenilirdir. Bu sadece makinelerdeki işletim sistemlerinin zayıf ve güçlü yanlarından kaynaklanmaz. Aynı zamanda bir masaüstü makinesinin, bir firma yerine, tek bir kişi tarafından kullanılmasından kaynaklanır. Örnek: bir işlemin ortasında kişisel bilgisayarın güç kablosu üzerinde oturan birine karşı bir koruma olamaz. Şirketlerdeki uygulama sunucuları ise cam bölmelerde korunurlar.

Yazılım hata oranları üç katmanlı mimaride daha düşük olma eğilimindedir. 2. katmandaki hizmetler özel olarak hazırlanan test sürücüleri ile denetlenebilir.

Üç katmanlı mimaride tasarım yaklaşımının modüler olması, yüksek güvenilirliğe sebep olur ve gerçekleştirimde karşılaşılabilecek sürprizleri önler.

2.5.3. Ölçülebilirlik, Performans

Bir prototip başarılı olunca ve yüzlerce kullanıcıya gönderilince ne olur? Geliştirilen ilk uygulama büyük bir başarı sağlayınca ve siz 3 tane daha benzer geliştirip, aynı ağda dağıtınca ne olur?

İstenilen özellikler:

- Bir saniyenin altında cevap süresi,
- Donanım/yazılım maliyetinin uygun olması,
- Ağ verimliliği,
- Fazla sorun çıkmadan genişleyebilirlik.

Masaüstünde, "thin client" ve destekleyen yazılımların kullanımı ile performans artar. Bu, kişisel bilgisayarın birden fazla uygulama çalıştırırken bile iyi performans göstermesini sağlar.

Sunucu tarafında; performans, aşağıdaki sebeplere dayalı olarak, arttırılabilir:

- Kullanıcıların sunucu örneklerine oranındaki uyum (genellikle 10:1 veya 20:1),
- Yükleme dengelemesi, sunucu örneklerinin uyumlu sayıda olması.

Sunucu örnekleri sayısı kontrol edilebildiğinden ve yükleme dengelemesi sağlanabildiğinden, sunucunun aşırı yüklenmesi daha kolay engellenebilir. (Sunucu üzerindeki yük bilinir ve kullanıcılar, kontrolsüz olarak, sunucuya bağlandıkça artmaz.)

İlişkisel veritabanının etkin kullanımı şöyle arttırılır:

- Veritabanı bağlantılarının en aza indirgenmesi (veritabanına sunucu programlar bağlanır, kullanıcılar değil),
- Sürekli veritabanı bağlantıları,
- Tam-derlenmiş hizmetler (bazı VTYS saklı yordamları ara-kod halindedir).

Ölçülebilirlik, iyi performansın ve aşağıdaki bazı etkilerin sonucudur:

- İyi ölçülebilir donanım platformlarının sağlanabilmesi,
- Herhangi bir anda, başka bir sunucunun mimariye eklenebilmesi,
- Sunucuların özelleştirilmesi (veritabanı, uygulama, iletişim, vs.).

2.5.4. Esneklik, Büyüme, Değişim

İstenilen özellikler:

- Genişleyebilirlik,
- Hızlı uygulama geliştirme,
- Bozulmalar olmadan değişebilme,
- Bakım,
- Tasarım ölçülebilirliği.

Başarılı uygulamalar büyür ve değişir. Uygulama yazılımları açısından bakıldığında, tasarım ölçülebilirliği ve modülerlik anahtardır.

Üç katmanlı yaklaşımı modülerliği zorunlu olarak sağlar. Şimdi bir üç katmanlı uygulamaya büyük bir alt sistem eklenmesi gerektiğini düşünün. Alt sistemi geliştirenler şöyle hareket edeceklerdir:

1. Yeni uygulama için yeni bir sunucu (donanım) ekle.
2. Tasarım yap.
3. Yeni iş mantığı (2. katman) hizmetleri geliştir.
4. Kullanıcı arayüzlerini, müşterinin isteği doğrultusunda değiştir.
5. Veritabanını genişlet ve yeni veri hizmetleri (3. katman) oluştur.

Bu aşamalar 3 katmanda da paralel olarak yürütülebilir.

Her katmandaki esnekliği inceleyelim:

2.5.4.1. Masaüstü İstemcileri

Masaüstü donanım ve yazılımını değiştirirsek, tasarım, etkiyi en aza indirger. Sadece kullanıcı arayüzü değişir; iş mantığı hizmetleri ve veri hizmetleri değişmeden kalır. Eski ve yeni arayüzler bir arada çalışabilir. Veri tabanı istemcilere şeffaf olduğundan, VTYS ile bütünleştirmede kaygımız olmaz.

Yeni uygulamalar geliştirip, bunları istemci tarafına kolayca bütünleştirdikçe; mimarimizin gücü açığa çıkar. 2. ve 3. katman hizmetleri ayrı ayrı geliştirilip test edilirler. Sonra seçilen 2. katman hizmetleri, uygun uygulamalarla bütünleştirilmek üzere, masaüstü istemcilere gönderilir.

2.5.4.2. Uygulama Hizmetleri

2. katman, 1. katmandan daha kararlıdır. İş kuralları, kullanıcı arayüzleri kadar hızlı değişmezler.

Eğer birşey değişirse, yeni bir içerilen hizmet, eskisinin yerini en az zararla alabilir. Yeni uygulama hizmetleri her zaman eklenebilir. Yeni uygulamalar var olan istemcilere yüklenebilir.

2.5.4.3. Veri Hizmetleri

Uygulama hizmetlerinde olduğu gibi, mimarinin modülerliği, veri hizmetlerinde de ekleme ve değiştirme kolaylığı sağlar. Ayrıca yeni bir VYTS'ne de adım adım geçebiliriz.

Bunların ötesinde, bütün veriler sadece bir ilişkisel veritabanında saklanmaz. *Üç katmanlı* mimaride, sunucu tamamen programlanabilirdir. Böylece;

- Sıralı ve indeksli kütükleri ilişkisel olmayan veritabanı,
- Evde hazırlanmış kütük erişim metotlarını faks sunucuları,
- EDI ve diğer ağlara çıktıları bütünleştirebiliriz.

2.5.5. Açık, Standart Tabanlı

İstenilen özellikler

- Özel araçları kullanabilme,
- Yeni yazılım araçları ve teknoloji ile kolayca çalışabilme,
- VTYS' den bağımsızlık.

2.5.5.1. İstemciler

Masaüstü en hızlı değişimin yaşandığı yerdir. Yeni bir geliştirme aracı seçmek de öyledir.

- Temel programlama dilleri henüz GUI geliştirmek için üreticiden bağımsız bir standart sağlamadılar.
- Üretkenlik gereklidir. Değişim hızlıdır. Bazen, kullanıcı arayüzlerini "atılabilir" olarak görmek en iyisidir.
- "thin client" yaklaşımı; GUI iş mantığını atmadan veya veritabanını değiştirmeden değiştirilebilir, demektir.
- Bir işlem sırasında, sadece "three-tier" yaklaşımı, eski ve yeni GUI lerin aynı iş mantığını kullanmalarına izin verir.

2.5.5.2. Hizmetler

Uygulama hizmetleri standart bir dilde (COBOL, C/C++) yazılabilir ve yazılmalıdır. Standart bir dile çeviren daha üst-düzey bir yaklaşım da kabul edilebilir.

Uygulamaya veri getirmek için, veri hizmetleri standart bir dilde yazılmalıdır. ANSI SQL ağırlıklı olarak, saklı yordamlar da çok gerektikçe kullanılmalıdır.

2.6. Programı Katmanlara Ayırmada Nesne Yönelimli Programlama

Nesne yönelimli programlamanın temelinde yeniden kullanılabilirlik (reuseability) yatar. Yani yazılan kod parçalarının olabildiğince birbirlerinden bağımsız ve gerektiğinde yeniden kullanılabilir olması gerekir. Buna neden gerek var?

Sanayi sektöründeki en büyük gelişmelerden birisi yarımamül üretiminin yapılmaya başlanmasıdır. Yarımamüller bir araya gelerek ana mamülleri oluştururlar. Mesela ayakkabı üreten bir firma için ayakkabının taban kısmı bir yarımamüldür. Aynı taban kullanılarak birçok değişik modelde ve renkte ayakkabı üretmek mümkündür. Bu sayede her model ayakkabı için ayrı bir taban tasarımı yapmaya veya ayakkabı üretimine taban üreterek başlamaya gerek kalmaz.

Yazılım sektörü de bu tür yarımamüller üretme arayışı içindedir. Mesela ticari program üreticisi bir firmanın hitap ettiği sektörlerin bir çoğunda fatura kesme özelliği vardır. Eğer fatura modülü programın diğer parçalarından bağımsız bir parça olarak tasarlandıysa yazılımevi aynı fatura modülünü birçok sektöre yönelik değişik program içinde kullanabilir. Eğer fatura modülü stok modülüne bağımlı olarak tasarlandıysa fatura modülünü kullanacağımız her yerde stok modülünü de kullanmamız gerekir. Bu durum hizmet sektörüne yönelik program yazdığımızda (bir kuaför mesela) bu fatura modülünü kullanamayacağımız anlamına gelir. Çünkü bu firmalar maldan ziyade hizmet satar ve faturaya verdiği hizmeti yazar. Kimi programlarda hizmeti stok olarak tanımlayarak çözümler aranır ancak bu durum hizmet konusunun doğasına aykırıdır. Buradaki örneğimize göre fatura modülü diğer her türlü modülden bağımsız tasarlanarak gerçek bir yarımamül haline getirilmelidir.

Aynı durum program için yazdığımız kodlarda da vardır. Mesela bir veri aktarma bileşenini sadece SQL Server veritabanından veri aktaracak şekilde yazarsak veri aktarma için yazdığımız kodlar SQL Server varsa işe yarar. Bir gün XML dosyasından veri aktarmak gerekirse eskiden yazdığımız şeylerin birçoğunu tekrar yazmamız gerekir. Gerçek yarımamül bir veri aktarma kodu veri kaynağından ve verinin yazılacağı yerden bağımsız çalışabilmelidir.

Günümüzde yazılım fabrikası terimi git gide yaygınlaşmaktadır. Hızlı ve doğru program üretme ihtiyacına yönelik olarak ortaya çıkan bu olgu yazılım üretme şeklimizin daha verimli hale getirilmesi anlamına gelir.

Bir program 3 temel iş yapar. Kullanıcıdan bilgiyi alır, bunu işler ve veritabanına yazar. Tam tersi düşünürsek veri kaynağından bilgiyi alır, bunu işler ve kullanıcıya gösterir. Modern programlamada bu üç iş birbirinden bağımsız yapılmalıdır. Yani bilgiyi işleme mantığı veritabanından ayrı olmalıdır ki biz istediğimiz herhangi bir veritabanını kullanabilelim. Ya da bilgiyi gösterme mantığı veri kaydetme mantığından ayrı olmalıdır ki biz istediğimiz şekilde kullanıcıya bunu formla veya web sayfasıyla gösterebilelim. Bu ihtiyaç 3 katmanlı yapıda program yazılmasını gerektirir. Kısa bir tekrar yapmamız gerekirse bu üç katman sunum, iş ve kayıt katmanıdır.

- Sunum katmanının görevi kullanıcıdan bilgi almak ve kullanıcıya bilgi göstermektir.
- İş katmanının görevi kullanıcıdan alınan bilgi üzerinde hesaplar yapmak veya kullanıcıya gösterilecek bilgiyi anlamlı hale getirmektir.
- Kayıt katmanının görevi bilginin veri kaynağından okunması veya kaydedilmesidir.

Mesela bir müşterimiz web üzerinden sipariş almak istiyor. Bizim yapımız katmanlara ayrıldıysa, iş katmanına veya kayıt katmanına hiç dokunmadan sadece sunum katmanında değişiklik yaparız. Siparişin hesaplama ve değerlendirilme mantığı siparişin gösterilmesinden ayrı ele alındığı için bu konularda tekrardan kodlama yapmamıza gerek kalmaz.

Nesne yönelimli diller bu gibi katmanlı yapılar oluşturabileceğimiz altyapıyı ve dil özelliklerini bize sağlar. Nesne yönelimli dillerde her olgu bir nesnedir. Yukarıdaki örneğimizi düşünürsek kabaca bir sipariş, bir sipariş gösterme ve sipariş kaydetme nesnemiz vardır. Sipariş nesnesi siparişin hangi kurallara göre işlem göreceği ile ilgili işleri yapar. Sipariş nesnesinin nasıl gösterileceğini sipariş gösterme nesnesi, nasıl kaydedileceğini de sipariş kaydetme nesnesi bilir. Biz diğer nesnelere hiç dokunmadan sipariş gösterme nesnemizi gerektiğinde web sayfası gerektiğinde de bir formla çalışacak şekilde geliştirebiliriz.

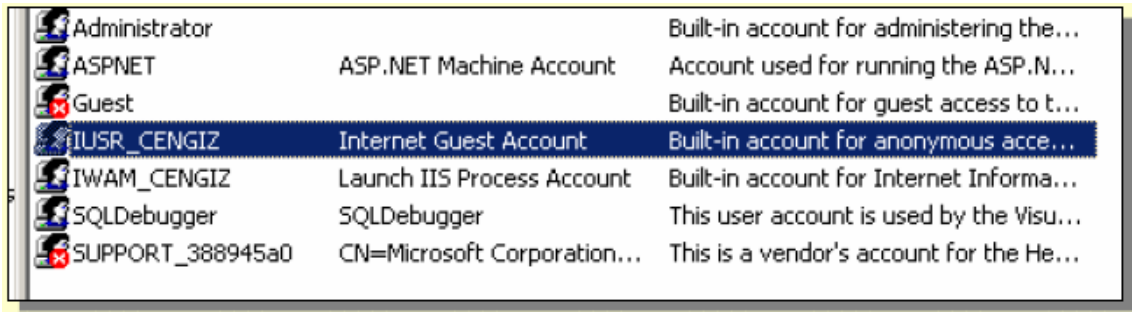
2.7. Üç Katmanlı Mimaride ASP.net ve ADO.net

ASP.NET, ASP'nin en yeni hali olup, yeni bir sürümünden ziyade, sıfırdan ve başka bir mimari üstünde geliştirilmiş şeklidir. ASP.NET'i, ASP'den ayıran en önemli noktalardan biri, 3 katman mimarisine ve nesneye dayalı programlama desteğindeki gelişmişliktir [12].

2.7.1. ASP.NET Web Uygulamalarında Güvenlik

Temel Güvenlik Kavramları

- Kimlik Denetimi (Authentication),
- Kullanıcıların tanımlanması aşaması,
- Yetkilendirme (Authorization) (Şekil 2.4),
- Kullanıcının kimliği doğrultusundan erişim haklarını belirleme aşaması Anonim (Anonymous) Erişim. (Internet üzeriden bulunan web sitelerinin çoğunda anonim erişim kullanılır.)
- Sitenin her bölümünün herkese açık olduğu ve sitenin gizli veya kişiye özel bilgi içermediği durumlarda, ASP.NET Web Uygulamaları anonim erişim için taklit etme (Impersonation) yöntemini kullanır.
- Kimliksiz erişim yapan kullanıcıya genel bir kullanıcı hesabı atanması yapılarak yetkilendirme yapılır.
- Varsayılan olarak bu kullanıcı hesabının adı ISUR_*makineadi* şeklindedir [10].



Şekil 2.4. Windows Kullanıcı Yetkilendirmesi

Kimlik Denetimi ile Erişim

- ASP.NET Kimlik Denetimi Yöntemleri,
- Microsoft Passport,
- Windows tabanlı,

- Form tabanlı.

Uygun Kimlik Denetim Sistemini Seçmek

- Anonim,
- Tanıtım siteleri,
- Kullanım kılavuzu,
- Windows tabanlı,
- Intranet uygulamaları,
- Form tabanlı,
- Alışveriş sitesi.

Web.Config Dosyasının Güvenlik Ayarları Açısından Temel Yapısı

- <authentication> Kimlik denetim sisteminin belirlenmesi için.
- <authorization> Yetkilendirme işlemi için.

2.7.1.1. Windows Tabanlı Kimlik Denetimi

Windows Tabanlı Kimlik Denetimi Nedir?

Windows tabanlı kimlik denetimi Windows işletim sistemi üzerine kurulu çalışır ve sunucu bilgisayar üzerindeki kullanıcı listesi ile kimlik denetimi yapar. Şekil 2.5'de örnek bir kod dosyası verilmiştir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.web>
    <authentication mode="Windows|Forms|Passport|None">
        <forms name="formismi"loginUrl="girissayfasiadresi"
            protection="All|None|Encryption|Validation"
            timeout="30" path="/"
            requireSSL="true|false"
            slidingExpiration="true|false">
            <credentials passwordFormat="Clear|SHA1|MD5">
                <user name="kullaniciadi" password="sifre"/>
            </credentials>
        </forms>
        <passport redirectUrl="internal"/>
    </authentication>

    <authorization>
        <allow users="kullanıcıların listesi"
            roles="rollerin listesi"
            verbs="eylemlerin listesi" />
        <deny users="kullanicilarin listesi"
            roles="rollerin listesi"
            verbs="eylemlerin listesi" />
    </authorization>

</system.web>
</configuration>
```

Şekil 2.5. Web.config dosyasından Windows tabanlı kimlik denetimi

Windows Kimlik Denetiminin Etkinleştirilmesi

Adım Adım Etkinleştirme

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <authentication mode="Windows" />
        <authorization>
            <deny users="?"/>
        </authorization>
    </system.web>
</configuration>
```

Şekil 2.6. <authentication> ve <authorization> kullanım örneği

Alt Klasör ve Dosyalar için Yetkilendirme Ayarları

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <location path="yonetim">
        <system.web>
            <authorization>
                <allow roles="BizimSirket/Yoneticiler"/>
                <deny user="*" />
            </authorization>
        </system.web>
    </location>
......
```

Şekil 2.7. Grup yetkilendirme örneği

Bir Kullanıcı için izinlerin Verilmesi ve Kısıtların Konulması

ASP.NET Web.config dosyasındaki <authorization> düğümü içerisindeki kullanıcı listesine bakarak kullanıcının uygulama üzerindeki yetkilerine karar verir. (Şekil 2.6 ve Şekil 2.7)

ASP.NET <authorization> düğümü içerisinde kullanıcıların erişim yetkilerini kontrol ederken yetkilendirme bildirimlerinden kullanıcı ile ilk uyuşanı kullanır.

Bu sebeple istenen kullanıcılara yetki verildikten sonra mutlaka <deny> düğümü ile önceki yetkilendirme düğümleri ile uyu_mayan kullanıcılar için kısıtlama yapılmalıdır. Bunu ise yıldız (*) karakteri vererek sağlayabiliriz [9].

Rol Tabanlı Kimlik Denetimini Kullanmak

- Her kullanıcıya yetki verilmek ile uğraşmadan belli bir rol üzerinde yetkilendirme işlemi gerçekleştirilir.

- Kullanıcılarda bu rolleri üstlendirilir.

Kullanıcı Bilgisini Okumak

Uygulamaya giriş yapmış olan kullanıcının bilgilerini okumak User nesnesinin Identity özelliği kullanılır. Identity özelliği kullanıcı adı ve rol bilgilerini içeren bir nesne döndürmektedir.

2.7.1.2. Form Tabanlı Kimlik Denetimi Nedir?

- Form tabanlı kimlik denetimi programcının düzenlediği bir Web Formu ile kimlik bilgilerinin kontrol edilmesi esasına dayanır. (Şekil 2.8)

- Web.config dosyasındaki kullanıcı listesine veya programcı tarafından düzenlenen ayrı bir veritabanına göre kimlik denetimi ve yetkilendirme yapılır.

- Form tabanlı kimlik denetiminde uygulamaya erişmek için kullanıcıların herhangi bir ağa üye olmalarına ihtiyaç yoktur.

- Form tabanlı kimlik denetimi daha çok herkese açık internet uygulamalarında kullanılır.

- Form tabanlı kimlik denetimi programcıya kendi kullanıcı veritabanını oluşturma imkânı tanır.

Adım Adım Etkinleştirme

1. Web.config dosyasından kimlik denetim sistemi Forms olarak ayarlanır.

2. Kullanıcı kimlik bilgilerinin (kullanıcı adı, şifre) girileceği bir Web Form oluşturulur.

3. Kullanıcı kimlik bilgilerinin saklanması için isteğe seçime bağlı olarak programcının belirlediği parametreler çerçevesinde bir veritabanı oluşturulur veya Web.config dosyasında gerekli tanımlamalar yapılır.

4. Oluşturulan Web Formuna kullanıcıların kimlik bilgilerinin denetlenmesi için gerekli kod yazılır. Buradaki denetleme için yazılacak kod parçası kullanıcı listesinin özel bir veritabanında mı, yoksa Web.config dosyasında mı tutulduğuna göre değişir.

Web.config Üzerindeki İşlemler

```
<authentication mode="Windows|Forms|Passport|None">
   <forms name="formismi"
          loginUrl="girissayfasiadresi"
          protection="All|None|Encryption|Validation"
          timeout="30" path="/"
          requireSSL="true|false"
          slidingExpiration="true|false">
      <credentials passwordFormat="Clear|SHA1|MD5">
         <user name="kullaniciadi" password="sifre"/>
      </credentials>
   </forms>
</authentication>
```

Şekil 2.8. Form Tabanlı Kimlik Denetimi örneği

- <authentication>
  - mode: Kimlik denetim sistemi (Forms)
- <forms>,
  - name: Çerezin (cookie) ismini belirler.(varsayılan : .ASPXAUTH)
  - loginUrl: Giriş sayfası.
  - protection: Kullanıcı bilgisayarında saklanacak olan çerezin (cookie) güvenliğinin nasıl sağlanacağını belirler.
  - All, Encryption, Validation, None. Varsayılan olarak All değerini alır. All değeri ile çereze yazılacak verilerin güvenliği en iyi şekilde sağlanmaktadır.
  - timeout: Kimlik denetimi çerezlerinin kaç dakika geçerli olacağını belirler. (30)
  - path: Oluşturulacak çerezlerin yol tanımlaması için kullanılır. Varsayılan değeri / dır.

- requireSSL: Kimlik denetim çerezinin transferi için güvenli bağlantı gerekip gerekmediğini belirler.

- slidingExpiration: Alabileceği değeler true ve false' dur.

  - true değerini aldığında kullanıcıdan gelen her istek ile zaman aşımı (timeout) süresi geri sayımı en baştan tekrar başlar.

  - .NET Framework 1.0 da varsayılan değer true idi, .NET Framework 1.1 versiyonunda varsayılan değer false' dur.

- <credentials> düğümü içerisine eklenen <user> düğümleri ile Web.config dosyası içerisinde kullanıcı tanımlaması yapılabilir.

- <credentials>

  - passwordFormat : Kullanıcı şifrelerine uygulanan karakter şifreleme (encrypt) yöntemi belirlenir. SHA1, MD5 ve Clear. Varsayılan değer SHA1' dir.

- <users>

  - name : Kullanıcının adını belirler.

  - password : Kullanıcının şifresini belirler. Belirlenen passwordFormat değeri ile uyumlu bir değer almalıdır.

- Bu şekilde bir uygulamada programcı yada bir yönetici tarafından Web.config dosyasına ekleme yapılmasıyla çalışabilir.

- Kullanıcıların kendi kullanıcı hesaplarını oluşturabileceği ve kimlik bilgilerinin yönetimini kendileri yapabileceği bir uygulama için kullanılamaz.

2.8. Üç Katmanlı Mimaride Sınıf Yapısı

Üç Katman Mimarisi, özellikle nesneye dayalı programlama ile birlikte güç bulmuş, veritabanı programlamanın esas dinamiklerini oluşturan bir programlama eğilimidir.

Nesne yönelimli programlamanın temelini Class'lar oluşturur. Bir class, nesne'nin kodlanarak tanımlamasıdır. Nesnelerle çalışma teorisinin temelinde, içeriğinin nasıl çalıştığını bilmeden onu kullanabilme mantığı yatar.

Bir class genel olarak su yapıdan ibarettir

- Method'lar, property'ler veya değişkenler(field) içerebilir (bunların her birine member denir).

- Method, prosedürel programlamada bildiğimiz fonksiyon'un işlevini class'larda yerine getirir.

- Property prosedürel programlamada bildiğimiz değişken'in işlevini class'larda yerine getirir. Bu işlev için, değişken de kullanılabilir.

- Property'ler yazılabilir ve okunabilir olabilir. Sadece okunabilir veya sadece yazılabilir tanımlanması methodlarla mümkündür. Ancak değişkenleri okunabilir yapmak için, Method'lar yerine, property adı verilen yarı method yarı değişken yapı kullanılabilir.

- Method'lar ve değişkenler, sadece sınıf içerisinden erişilebilir veya bütün sınıflardan veya bütün assemly'lerden erişilebilir olabilir veya erişilemez olarak kodlanabilir. Bunlar için, Public, Private, Internal, Protected, External... gibi deyimler(Erişim Tanımlayıcıları) kullanılır.

- Methodlar, aynı adda ama farklı girdi veya çıktı parametrede üst üste tanımlanabilir.(method overloading) Bu durumda, class'in kullanımı aşamasında, çağrılış sekline en uygun method otomatik olarak algılanır ve devreye girer.

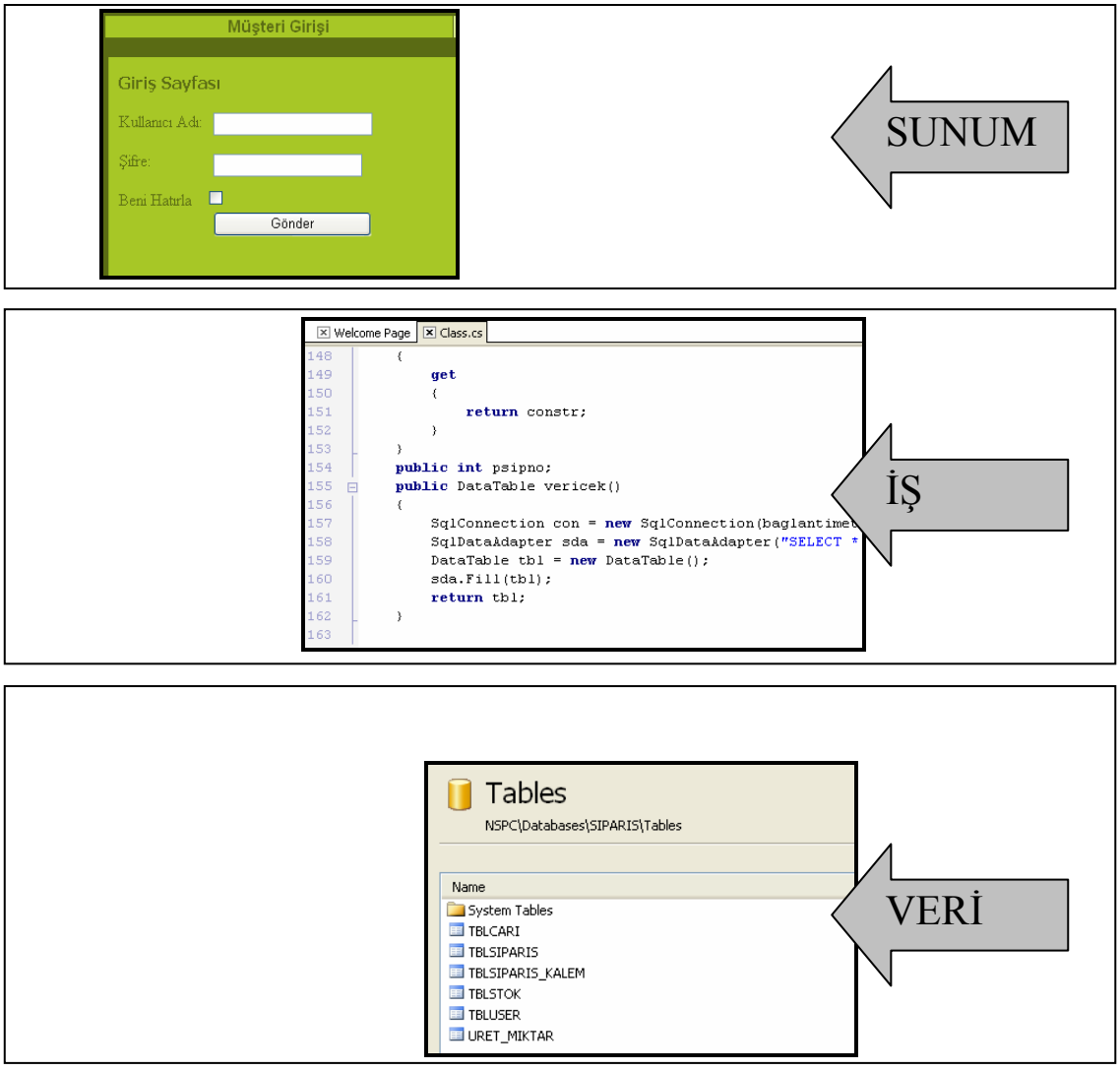
- Class'lar yazıldıktan sonra, kütüphanelerde saklanırlar. MFC, .NET gibi belli bir geliştirme ortamı ile birlikte gelen kütüphanelere, temel kütüphane(base library-base class) denir. Bu ve kullanıcı tarafından geliştirilmiş kütüphaneler, isim uzayı (namespace) denilen nokta notasyonlu yapılarla düzenlenir. Namespace kavramı, fiziksel değil mantıksal bir kavramdır. Yani, farklı dokümanlarda birbiri ile ayni seviyede class'lar olabilir.

- Düzenlenmiş Class'lara erişim, isim uzayından sonra sınıf adi gelecek şekilde bir notasyon ile sağlanır.(System.Data.DataSet gibi)

- C# gibi dillerde, kütüphane sınıflarını, uygulamanın içerisindeymiş gibi doğrudan nesne adıyla çağırmak mümkündür. Ancak bunun için uygulamanın en başında, isim uzayları ve alt isim uzayları nokta notasyonu ile sınıf adına kadar veya belli bir seviyeye kadar belirtilebilir. Geri kalanlar, (sınıf adi veya isim uzayının tamamı belirtilmedi ise, belirtilmeyen kısmı) kod içerisinde belirtilerek, sınıflara erişilebilir.(using System.Data dendikten sonra daha herhangi bir yerde doğrudan DataSet denilerek, veya basta using System denilmişse, daha sonra Data.DataSet denilerek veya başka hiç bir şey denmeden, System.Data.DataSet denilerek erişmek mümkündür.)

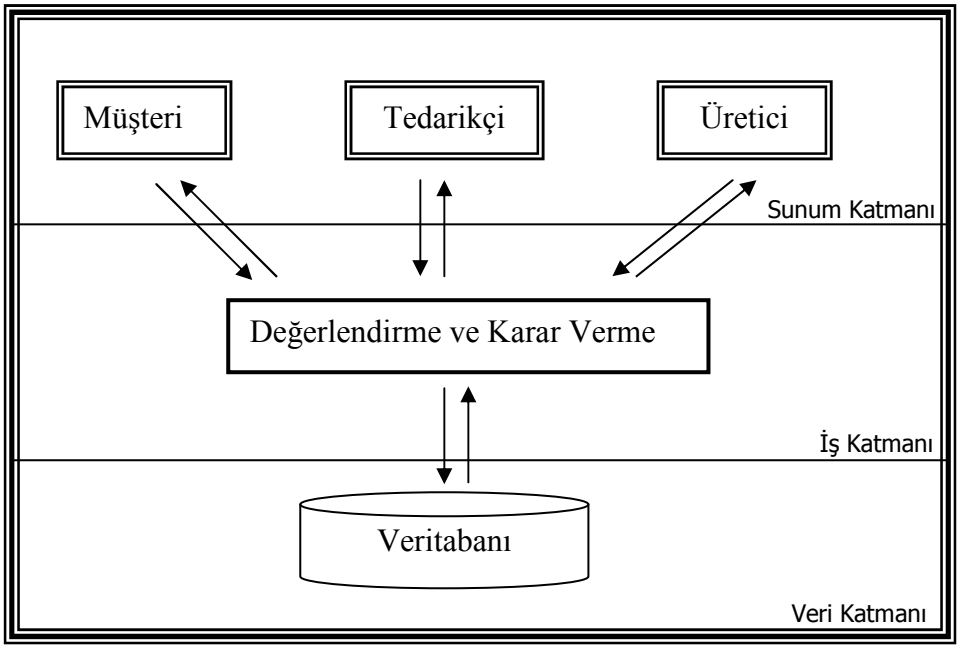
3. TEDARİK ZİNCİRİ YÖNETİMİNDE SİPARİŞ MODÜLÜ

Modülümüzde Şekil 3.1 de görüldüğü üzere üç katmanlı mimari kullanılmıştır.



Şekil 3.1. Üç Katmanlı Mimari Yapısı

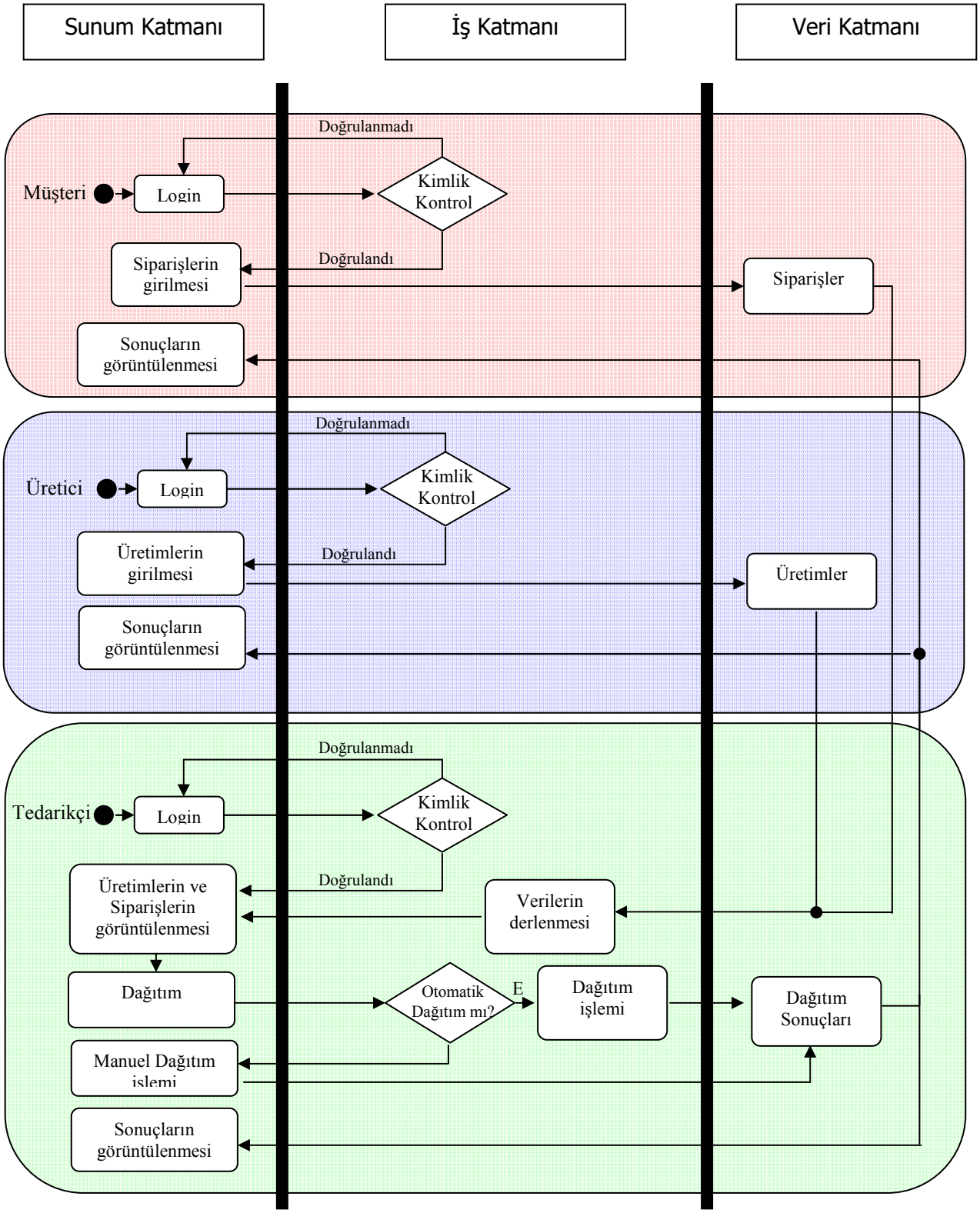
Projede çok yönlü bir Tedarik Zinciri Yönetimi (Supply Chain Management) söz konusudur. (Şekil 3.2)



Şekil 3.2. Tedarik Zinciri Yönetiminde Üç Katman

Müşteriler sisteme kendi hakları ile giriş yaparak almak istedikleri ürünleri ve miktarları günün belirlenmiş olan saatlerinde sipariş olarak sisteme girerler. Sipariş işlemlerinin sisteme girişleri sırasında üretici firmalar da yine sisteme girerek o tarih için ne kadar stok üretebileceklerine karar verip sisteme giriş yaparlar. Bu süreç de yine günün belirlenmiş bir saati içerisinde gerçekleşmektedir. Üretici firmalar da işlemlerini gerçekleştirdikten sonra sıra siparişlerin hangi üreticiler tarafından, hangi müşterilere öncelik olarak karşılanacağının karar verilmesine gelmiştir. Tedarikçinin sisteme giriş yapmasıyla da bu süreç başlar. Tedarikçi isterse sistemin tanımlanmış olan müşterilerin öncelik katsayılarına göre otomatik olarak dağıtım yapmasına izin verir isterse de bunu kendi istek ve koşullarına göre gerçekleştirir. Müşteri öncelik katsayıları, müşterilerin yıl boyunca satın aldıkları ürünlerin toplam tutarlarına bağlı olarak, yılda bir kez güncellenir. Ve tedarikçi müşterilerin ihtiyacı olduğu ürünlerin tamamını karşılayamadığı durumlarda bu katsayılara göre dağıtım yapar. Dağıtım yapmanın başka yöntemleri de vardır. Örneğin yine tedarikçi, müşterilerin toplam ihtiyaçlarını karşılayamadığı bir durumda, karşılayabildiği toplam miktar ile karşılaması gereken toplam

miktarın yüzde olarak oranını bulur ve bu oran doğrultusundan her müşteriye tam olmasa da dağıtımını gerçekleştirir. Ürünün yetersiz olması durumunda bir üçüncü alternatif ise siparişi veren ilk müşterinin ürünü ilk almasıdır. Biz bu yöntemlerden sektörde genel olarak kullanılan yöntem olduğunu düşündüğümüz öncelik katsayısını kullanacağız. Tedarikçi de kendisi için tanımlanmış saat de sisteme giriş işlemini gerçekleştirip, dağıtım işlemini tamamladıktan sonra sonuçların geri bildirimi için müşteriler ve üretici firmalar sisteme giriş yapıp siparişlerinin ne oranda karşılanabileceğini ve ne kadar ürün üretmeleri gerektiğini öğrenebilirler. Şekil 3.3'de Aktivite Diyagramı verilmiştir.

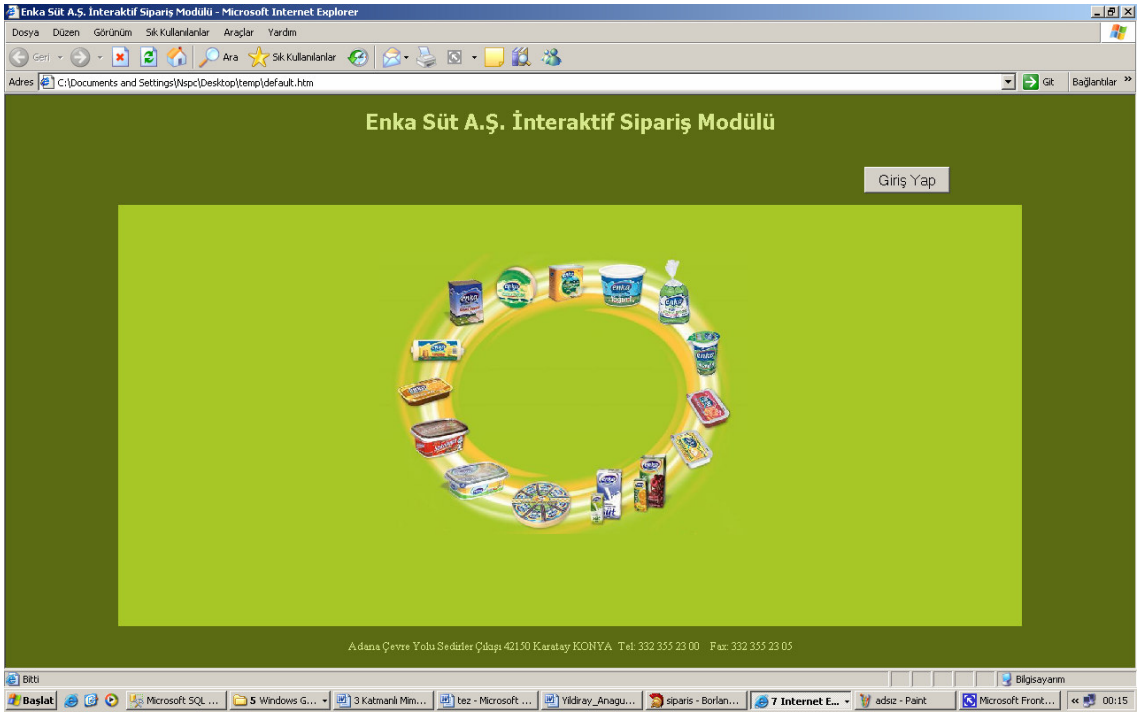


Şekil 3.3. Sipariş Modülü Aktivite Diyagramı

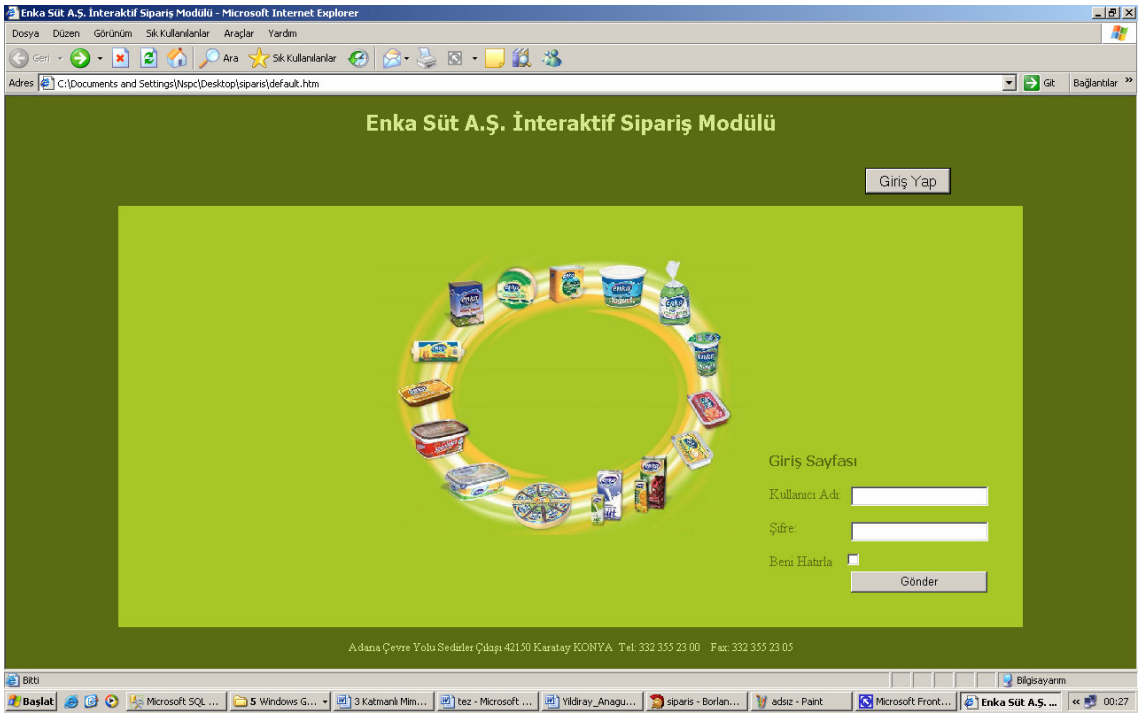
Üç katmanlı mimariye göre uygulamamız 3 ayrı katmana bölünür. Bunlar veri katmanı, iş katmanı ve sunum katmanıdır. Veri katmanında uygulamanın veritabanı ile olan kısmı gerçekleştirilir. Veritabanı ile ilgili bütün işlerimizi burada halletmek uygulama ve güvenlik açısından çok önemlidir. İş katmanında uygulamamızdaki verileri ne şekilde ve nasıl sunum katmanında sergileyeceğimizin tasarımı yapılır. Biz projemizde bunu ADO.NET ile yapmaya çalışacağız. Son olarak sunum katmanında ise verilerin istemciye sunulması ve gösterimi gerçekleştirilir. Biz uygulamalarımızda yapacağımız işi daha çok veri ve iş katmanında yapmalıyız. Böylece sunum katmanına fazla yük binmemiş olur. Şimdi katmanları ayrıntılı olarak inceleyelim.

3.1. Sunum Katmanı:

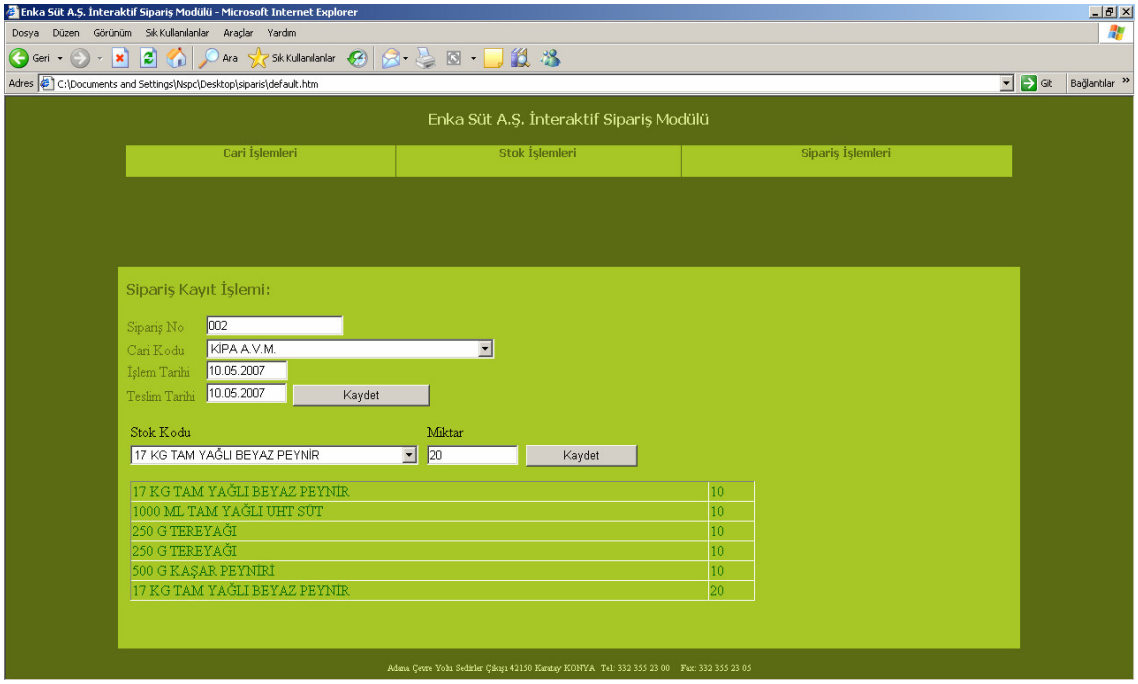
Bu katman kullanıcıların sisteme girerek kendi yetkileri dahilinde olan arayüze erişip ilgili işlemleri gerçekleştirdikten sonra iş katmanına gönderdikleri ve iş katmanından gelen verileri de görebildikleri bölümdür. İstemcinin kullanacağı ekran bu katmanda hazırlanır. Bu katman için ASP.NET kullanıldı. Bu katmanın çalışabilmesi için IIS (Internet Information Services) aktif olması gerekiyor. Bu katman için proje web sitemiz oluşturuldu ve arayüz sayfalarımız hazırlandı. Bu katmanı kullanarak müşteri, tedarikçi ve üreticiler sisteme giriş yaparak kendilerine ait arayüze ulaşıyor ve gerekli kayıtları girerek iş katmanına gönderiyorlar. Müşteri arayüzünde sipariş ekleme, düzeltme ve silme işlemleri gerçekleştirilebiliyor. Ve karar sonrasında siparişlerin durumu görüntülenebiliyor. Üretici arayüzünde sisteme girilen siparişler görüntülenebiliyor ve bu doğrultuda üreticinin üretebileceği ürün miktarlarının sisteme girişi sağlayan üretim miktarı ekleme, düzeltme ve silme işlemleri gerçekleştirilebiliyor. Ve yine karar sonrasında üretici ne kadar üretim yapacağını görüntüleyebiliyor. Tedarikçi arayüzünde ise sisteme girişi yapılan siparişler ve üretim miktarları görüntülenebiliyor ve tedarikçinin müşteri siparişlerinin hangi üreticiler tarafından üretilebileceğinin planlamasına izin veriliyor [15]. Şekil 3.4, Şekil 3.5 ve Şekil 3.6'da İnteraktif Sipariş Modülüne ait arayüz örnekleri verilmiştir.



Şekil 3.4. İnteraktif Sipariş Modülü Ana Sayfa



Şekil 3.5. İnteraktif Sipariş Modülü Login Ekranı

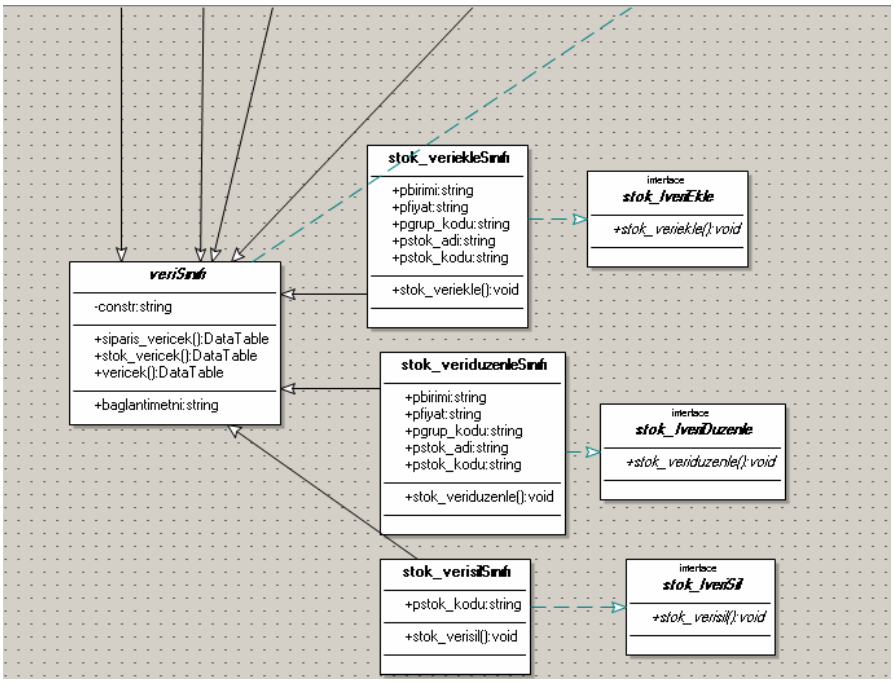


Şekil 3.6. İnteraktif Sipariş Modülü Sipariş Kayıt Ekranı

3.2. İş Katmanı

Bu katman sunum katmanından gelen bilgileri kontrol ederek ve işleyerek veri katmanına aktaran geçiş katmanı gibidir. Ama asıl önemli olan katman da budur. Çünkü bütün işlem ve denetimler bu katmanda gerçekleşir. Bu katmanda sisteme giriş yapan müşteri, tedarikçi ve üreticilerin kullanıcı kayıtları güvenlik amacıyla kontrol edilir. Ve kullanıcı sınıfına bağlı olarak sadece kendileriyle ilgili arayüze ulaşmaları sağlanacaktır. Sistem saati de bu kontrol de etkilidir. Çünkü günün belirli saatlerinde kullanıcılar kendilerine ait işlemleri gerçekleştirebileceklerdir. Sonrasında arayüz aracılığıyla girilen veriler veri katmanına iletilir ve gerektiğinde de veri katmanından çağrılarak gerekli işlemler gerçekleştirildikten sonra tekrar sunum katmanına aktarılır. Burada yazacağımız sınıflar bize sunum katmanında sadece birkaç satır ile işimizi halletmemizi sağlayacaktır. Sunum katmanına yüklenmemek programımızın hızlı olması açısından çok önemlidir, daha sonra güvenlik konusunda tedirginlik yaşamak istemiyorsak uygulamamızı veri ve iş katmanında bitirmeliyiz. Uygulamamızı güncellemek durumunda kaldığımız zamanlarda iş katmanında yapacağımız bir kaç ufak değişiklik yeterli olacaktır [14].

İş katmanını için class library uygulaması kullanacağız. Bu katman için C# kullanıldı. Şekil 3.7'de İnteraktif Sipariş Modülü sınıf yapısından bir bölüm verilmiştir.

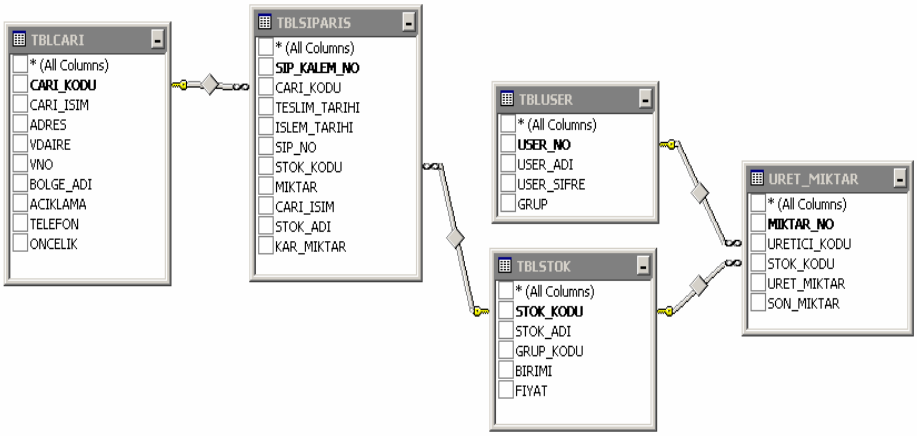


Şekil 3.7. İnteraktif Sipariş Modülü Sınıf Yapısından Bir Bölüm

3.3. Veri Katmanı:

Bu katman için Microsoft Sql Server 2005 kullanıyoruz. Sql de öncelikle veritabanımızı, daha sonra müşteri, stok, üretim miktarları ve sipariş kayıtlarımızı tutacağımız tablolarımızı ve daha sonra da ekleme, silme ve düzenleme işlemlerini gerçekleştireceğimiz procedurelerimizi oluşturuyoruz. Bu işlemler için Sql komutlarını kullanıyoruz.

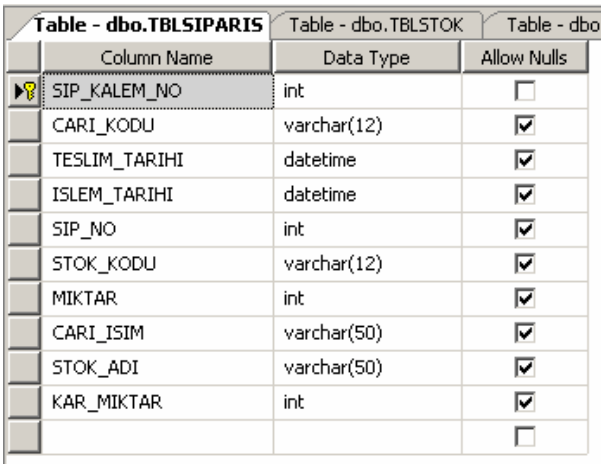
Şekil 3.8'de projemizde yer alan tabloların birbirleri ile ilişkisini gösteren şema gösterilmiştir.



Şekil 3.8. İnteraktif Sipariş İlişkisel Veritabanı Şablonu

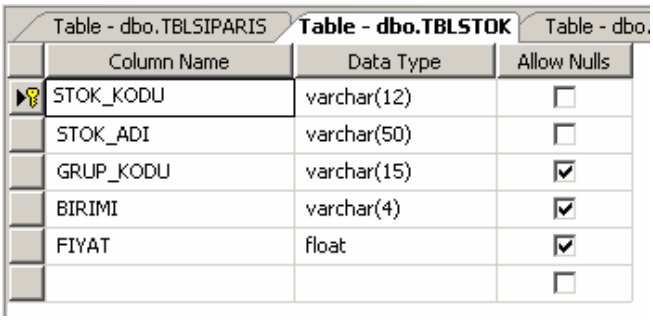
Projemizde kullandığımız tabloların alan isimleri ve veri tipleri ise şu şekildedir:

Sipariş Tablosu (Bu tablodaki CARI_KODU ve STOK_KODU alanları TBLCARI ve TBLSTOK tablosuyla ilişkilidir. SIP_KALEM_NO alanı primary keydir. Otomatik 1 artırma sayısıyla artacak şekilde tanımlanmıştır. Bu tablodaki MIKTAR alanı müşterinin istediği sipariş miktarın girildiği alan, KAR_MIKTAR ise tedarikçinin bu sipariş ile ilgili olarak karşılayabileceği miktarın girildiği alandır.) – Şekil 3.9



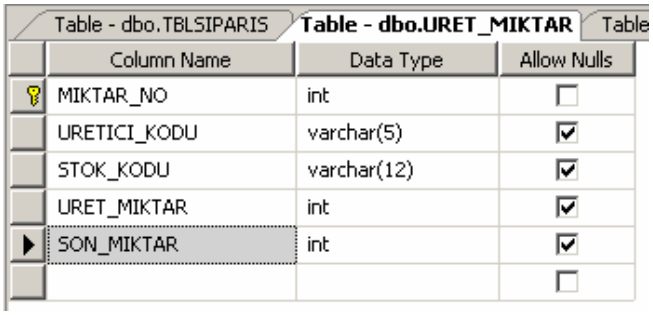
Şekil 3.9. Sipariş Tablo Yapısı

Stok Tablosu – Şekil 3.10



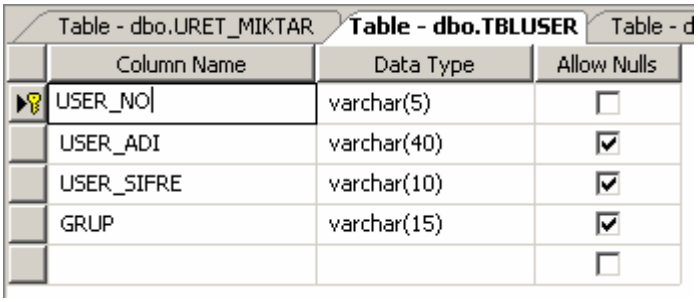
Şekil 3.10. Stok Tablo Yapısı

Üreticilerin üretim miktarlarının yer aldığı tablo (URETICI_KODU alanı TBLUSER tablosuyla ilişkilidir. Bu tablodaki URET_MIKTAR üreticinin üretebileceği miktarın girildiği, SON_MIKTAR ise tedarikçi tarafından onun üretmesini istediği miktarın girildiği alandır.) – Şekil 3.11



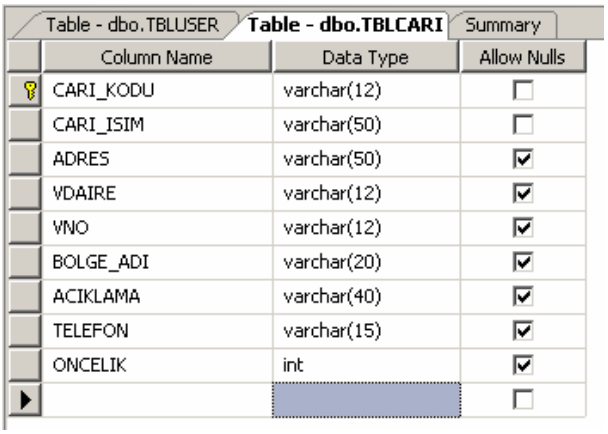
Şekil 3.11. Üretim Miktarları Tablo Yapısı

Kullanıcı Tablosu (Bu tablo URET_MIKTAR tablosuyla ilişkilidir. Kullanılan GRUP alanı kullanıcının tipini (Müşteri, Tedarikçi, Üretici) tanımlar. – Şekil 3.12



Şekil 3.12. Kullanıcı Tablo Yapısı

Müşteri Tablosu (TBLSIPARIS tablosuyla ilişkilidir. Bu tabloda kullanılan ONCELIK alanı müşterinin sipariş almadaki önceliğinin tanımlandığı katsayı için kullanılmıştır. Bu katsayıyı tedarikçi yılda bir müşterinin kendisinden ürün almasına bağlı olarak yeniler.) – Şekil 3.13



Şekil 3.13. Müşteri Tablo Yapısı

3.3.1. Saklı Yordamlar (Stored Procedures)

Saklı yordamlar, SQL Server üzerinde saklanan önceden derlenmiş SQL ifadeleridir. Önceden derlenmiş olarak bulunduklarından her türlü sorgulamada en iyi performansı verirler. SQL Server' da sistem tarafından "sp_" ile başlayan isimlerle tanımlanmış bir çok prosedür mevcuttur. Bunlar daha çok administration maksatları için ve sistem tablolarından bilgi toplamak için kullanılırlar. Kullanıcı tarafından da kendi prosedürlerini tanımlamak oldukça kolay bir işlemdir. Kullanıcı kendi ihtiyaçları doğrultusunda prosedürler oluşturabilir, kullanabilir, onların hakkında bilgi toplayabilir ve birinden ötekine parametre geçişi sağlayabilir. SQL Server' ın sunduğu imkanlar oldukça geniş olmasına rağmen biz yalnızca kullandığımız kadarıyla bu konuya değineceğiz. Saklı yordamlar, SQL Server' a güç, etkinlik ve esneklik kazandırırlar. Kullanıldıkları zaman SQL ifadelerinin ve toplu işlemlerin performansını gözle görünür bir biçimde arttırırlar.

Saklı yordamlar:

- Parametre alabilirler,
- Başka prosedürleri çağırabilirler,
- Kendisini çağıran bir prosedür veya toplu işleme başarılı olduğunu ya da olamadığını, hata oluşması durumunda hatanın nedenini bir durum değeri olarak döndürebilirler.
- Parametrelerin değerlerini kendisini çağıran bir prosedüre döndürebilirler.

Saklı yordamların tanımlandıkları andan itibaren tabi tutuldukları işlemlerin farklı olması nedeniyle diğer SQL ifadelerinden ayrılırlar. Direk SQL Server üzerinde saklandıkları

için oldukça hızlı çalışan veritabanı nesneleridirler. Bir Stored Procedure ilk kez çalıştırıldığında şu işlemlere tabi tutulur.

1. Öncelikle prosedürün bileşenleri parçalara ayrıştırılırlar.

2. Veritabanındaki table, view gibi batka nesnelere referans yapan biletenler için bu nesnelerin varlığı kontrol edilir.

3. Kontrol işlemi tamamlandıktan sonra prosedürün ismi sysobjects tablosunda ve de prosedürü oluşturan kod da syscomments tablosunda saklanır.

4. Bu adımda derleme işlemi yapılır. Derleme esnasında normalize edilmiş plan ortaya çıkar. (Buna sorgulama ağacı da denir) Oluşturulan sorgulama ağacı sysprocedures tablosunda saklanır.

5. Saklı yordam ilk defa çalıştığı zaman, sorgulama planı okunur ve tamamen bir prosedür planı içine derlenir. Daha sonra da çalıştırılır. Bu sayede Saklı yordam her çalıştırıldığında parçalama, kontrol, sorgulama ağacı oluşturma işlemleri yapılmaz. Bu şekilde de zamandan önemli ölçüde tasarruf edilmiş olunur.

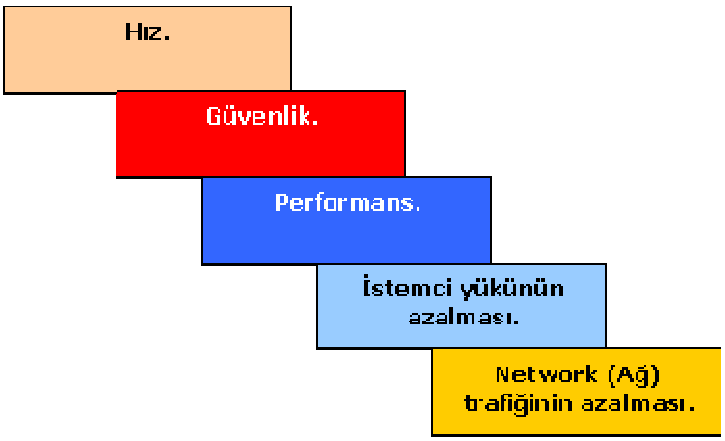
Bundan sonra, SQL Server' ın çalıştığı andan itibaren prosedürün ilk kez çalıştırılmasıyla birlikte, prosedür derlenmiş bir şekilde belleğe yerleştirilir. Çünkü diğer işlemler prosedür oluşturulurken yapılmış ve bitmiştir. Tekrar yapılmazlar. Saklı yordam kullanmanın başka bir yararı da budur. Bir kez çalıştırıldıktan sonra prosedür planı procedure cache bölgesinde muhafaza edilir. Bu da bir sonraki çağrılışında direk cache' den okunup çalıştırılması demektir. Böylece standart bir SQL sorgulamasının tekrar tekrar çalıştırılmasından çok daha üstün bir performans elde edilir. Saklı yordam kullanmanın göze çarpan diğer faydalarını şöyle sıralayabiliriz: Uygulamanın getirdiği bazı iş kuralları prosedür içinde tanımlanabilir. Bir kez oluştuktan sonra bu kurallar birden çok uygulama tarafından kullanılarak daha tutarlı bir veri yönetimi sağlanır. Ayrıca bir fonksiyonelliğin değişmesi ihtiyacı doğduğunda her uygulama için değişiklik yapmak yerine, sadece bir platformda değişiklik yapılır.

Tüm prosedürler üstün performansla çalışır ancak birden fazla çalıştırılacak olan prosedürler sorgulama planları procedure cache içinde saklandığından daha da hızlı çalışırlar.

Saklı yordamlar SQL Server start ettikten sonra otomatik olarak çalıştırılmak üzere ayarlanabilirler.

Saklı yordamlar harici olarak kullanılırlar. Trigger' lardan farklı olarak prosedürler uygulama tarafından ya da script tarafından bir şekilde çağrılmak zorundadırlar. Otomatik devreye giremezler.

Saklı yordamların içinde SQL sorgulama diline ek olarak SQL Server' ın kendi fonksiyonlarından da yararlanılabilir. Şekil 3.14'de saklı yordam kullanmanın faydaları şema olarak verilmiştir.



Şekil 3.14. Saklı Yordam Kullanmanın Faydaları

Projemizde kullandığımız saklı yordam kodları ise aşağıdaki gibidir.

Müşteri kaydını düzenlemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[cari_duzenle]
(
        @CARI_KODU nvarchar(12),
        @CARI_ISIM nvarchar(50),
        @ADRES nvarchar(50),
        @VDAIRE nvarchar(12),
        @VNO nvarchar(12),
        @BOLGE_ADI nvarchar(20),
        @ACIKLAMA nvarchar(40),
        @TELEFON nvarchar(15)
)
```

AS

UPDATE TBLCARI

SET
CARI_ISIM=@CARI_ISIM,ADRES=@ADRES,VDAIRE=@VDAIRE,VNO=@VNO,BOLGE_ADI=@BOLGE_ADI,ACIKLAMA=@ACIKLAMA,TELEFON=@TELEFON

WHERE CARI_KODU=@CARI_KODU [13].

Müşteri kaydını eklemek için kullanılan saklı yordam;

CREATE PROCEDURE [dbo].[cari_ekle]

(

@CARI_KODU nvarchar(12),
@CARI_ISIM nvarchar(50),
@ADRES nvarchar(50),
@VDAIRE nvarchar(12),
@VNO nvarchar(12),
@BOLGE_ADI nvarchar(20),
@ACIKLAMA nvarchar(40),
@TELEFON nvarchar(15)

)

AS

INSERT INTO TBLCARI
(CARI_KODU,CARI_ISIM,ADRES,VDAIRE,VNO,BOLGE_ADI,ACIKLAMA,TELEFON)

VALUES
(@CARI_KODU,@CARI_ISIM,@ADRES,@VDAIRE,@VNO,@BOLGE_ADI,@ACIKLAMA,@TELEFON)

Müşteri kaydını silmek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[cari_sil]
    @CARI_KODU nvarchar(12)
AS
DELETE FROM TBLCARI WHERE (CARI_KODU=@CARI_KODU)
```

Sipariş kayıtlarını düzenlemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[siparis_duzenle]
(
    @CARI_KODU nvarchar(12),
    @TESLIM_TARIHI datetime,
    @ISLEM_TARIHI datetime,
    @SIP_NO int,
    @STOK_KODU nvarchar(12),
                @MIKTAR int,
                @CARI_ISIM nvarchar(50),
                @STOK_ADI nvarchar(50),
                @KAR_MIKTAR int
)
AS
UPDATE TBLSIPARIS
SET
CARI_KODU=@CARI_KODU,TESLIM_TARIHI=@TESLIM_TARIHI,ISLEM_TARIHI=@
ISLEM_TARIHI,SIP_NO=@SIP_NO,
STOK_KODU=@STOK_KODU,MIKTAR=@MIKTAR,CARI_ISIM=@CARI_ISIM,
STOK_ADI=@STOK_ADI,KAR_MIKTAR=@KAR_MIKTAR
WHERE STOK_KODU=@STOK_KODU
```

Sipariş kaydı eklemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[siparis_ekle]
(
    @CARI_KODU nvarchar(12),
    @TESLIM_TARIHI datetime,
    @ISLEM_TARIHI datetime,
    @SIP_NO int,
    @STOK_KODU nvarchar(12),
    @MIKTAR int,
        @CARI_ISIM nvarchar(50),
        @STOK_ADI nvarchar(50)
)
AS
INSERT INTO
TBLSIPARIS(CARI_KODU,TESLIM_TARIHI,ISLEM_TARIHI,SIP_NO,STOK_KODU,MI
KTAR,CARI_ISIM,STOK_ADI)
VALUES
(@CARI_KODU,@TESLIM_TARIHI,@ISLEM_TARIHI,@SIP_NO,@STOK_KODU,@MIK
TAR,@CARI_ISIM,@STOK_ADI)
```

Sipariş kaydını silmek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[siparis_sil]
    @SIP_NO int
AS
DELETE FROM TBLSIPARIS WHERE (SIP_NO=@SIP_NO)
```

Stok kaydını düzenlemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[stok_duzenle]
(
    @STOK_KODU nvarchar(12),
    @STOK_ADI nvarchar(50),
    @GRUP_KODU nvarchar(15),
    @BIRIMI nvarchar(4),
    @FIYAT float
)
AS
UPDATE TBLSTOK
SET
STOK_ADI=@STOK_ADI,GRUP_KODU=@GRUP_KODU,BIRIMI=@BIRIMI,FIYAT=@F
IYAT
WHERE STOK_KODU=@STOK_KODU
```

Stok kaydı eklemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[stok_ekle]
(
    @STOK_KODU nvarchar(12),
    @STOK_ADI nvarchar(50),
    @GRUP_KODU nvarchar(15),
    @BIRIMI nvarchar(5),
    @FIYAT float
)
```

```
AS
INSERT INTO TBLSTOK
(STOK_KODU,STOK_ADI,GRUP_KODU,BIRIMI,FIYAT)
VALUES (@STOK_KODU,@STOK_ADI,@GRUP_KODU,@BIRIMI,@FIYAT)
```

Stok kaydı silmek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[stok_sil]
    @STOK_KODU nvarchar(12)
AS
DELETE FROM TBLSTOK WHERE (STOK_KODU=@STOK_KODU)
```

Üretim miktarlarını düzenlemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[uret_miktar_duzenle]
(
    @MIKTAR_NO int,
    @URETICI_KODU nvarchar(5),
    @STOK_KODU nvarchar(12),
    @URET_MIKTAR int,
    @SON_MIKTAR int
)
AS
UPDATE TBLURET_MIKTAR
SET
URETICI_KODU=@URETICI_KODU,STOK_KODU=@STOK_KODU,URET_MIKTAR=
@URET_MIKTAR,SON_MIKTAR=SON_MIKTAR
WHERE MIKTAR_NO=@MIKTAR_NO
```

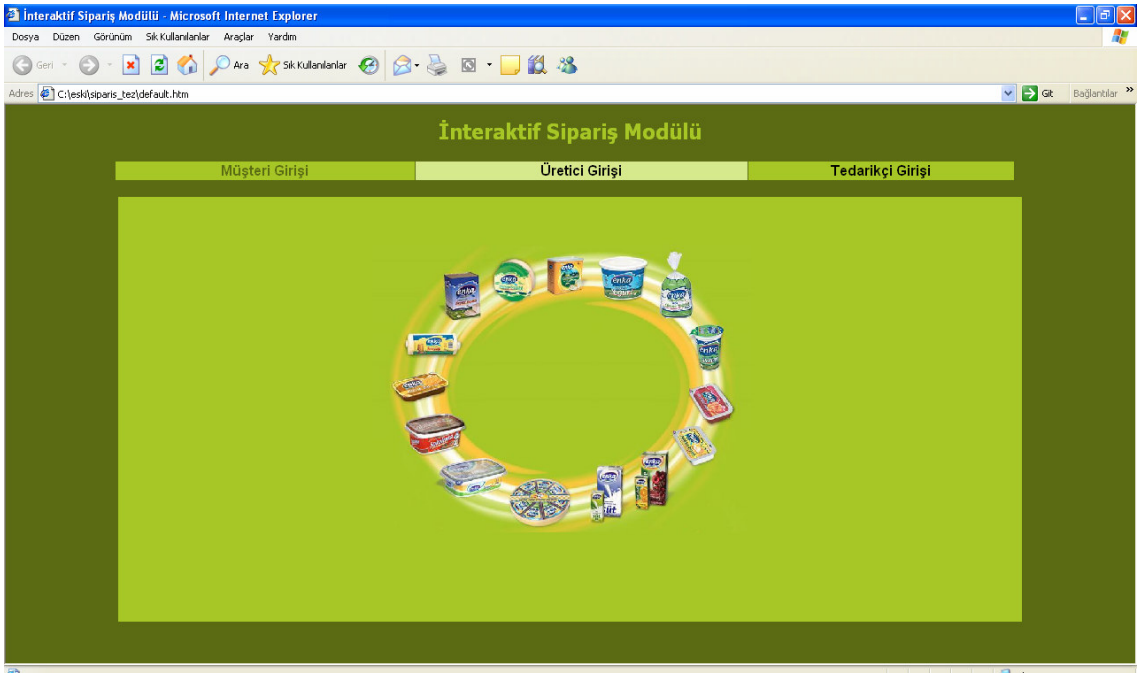
Üretim miktarlarını eklemek için kullanılan saklı yordam;

```
CREATE PROCEDURE [dbo].[uret_miktar_ekle]
(
    @URETICI_KODU nvarchar(5),
    @STOK_KODU nvarchar(12),
    @URET_MIKTAR int,
    @SON_MIKTAR int
)
AS
INSERT INTO
TBLURET_MIKTAR(URETICI_KODU,STOK_KODU,URET_MIKTAR,SON_MIKTAR)
VALUES
(@URETICI_KODU,@STOK_KODU,@URET_MIKTAR,@SON_MIKTAR)
```

Üretim miktarlarını silmek için kullanılan saklı yordam

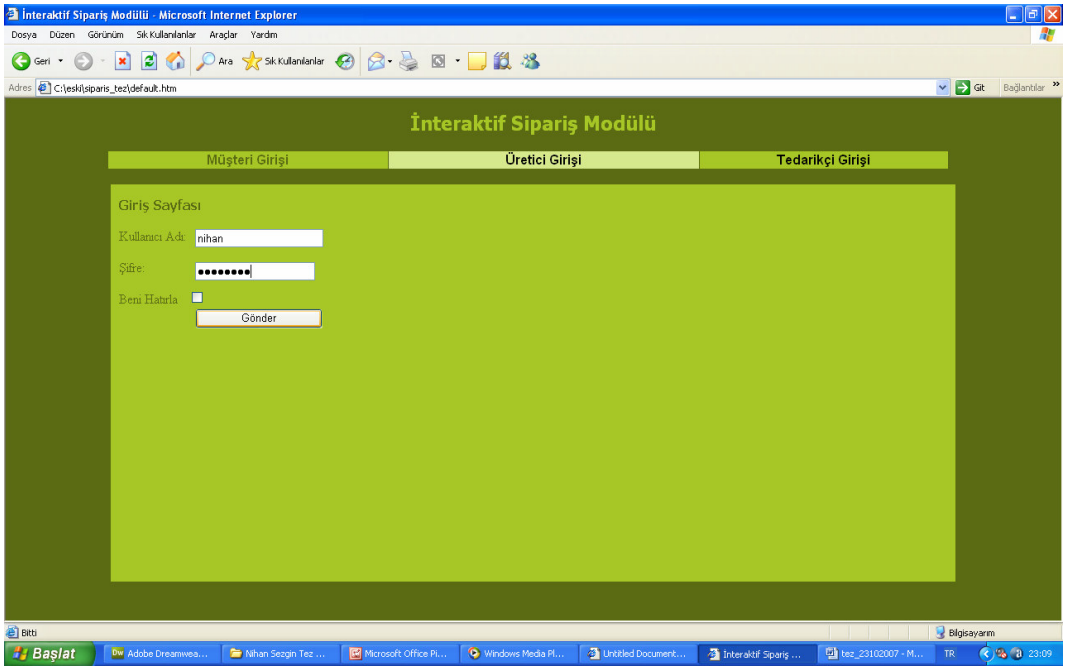
```
CREATE PROCEDURE [dbo].[uret_miktar_sil]
    @MIKTAR_NO int
AS
DELETE FROM TBLURET_MIKTAR WHERE (MIKTAR_NO=@MIKTAR_NO)
```

4. PROGRAM UYGULAMA ÖRNEĞİ



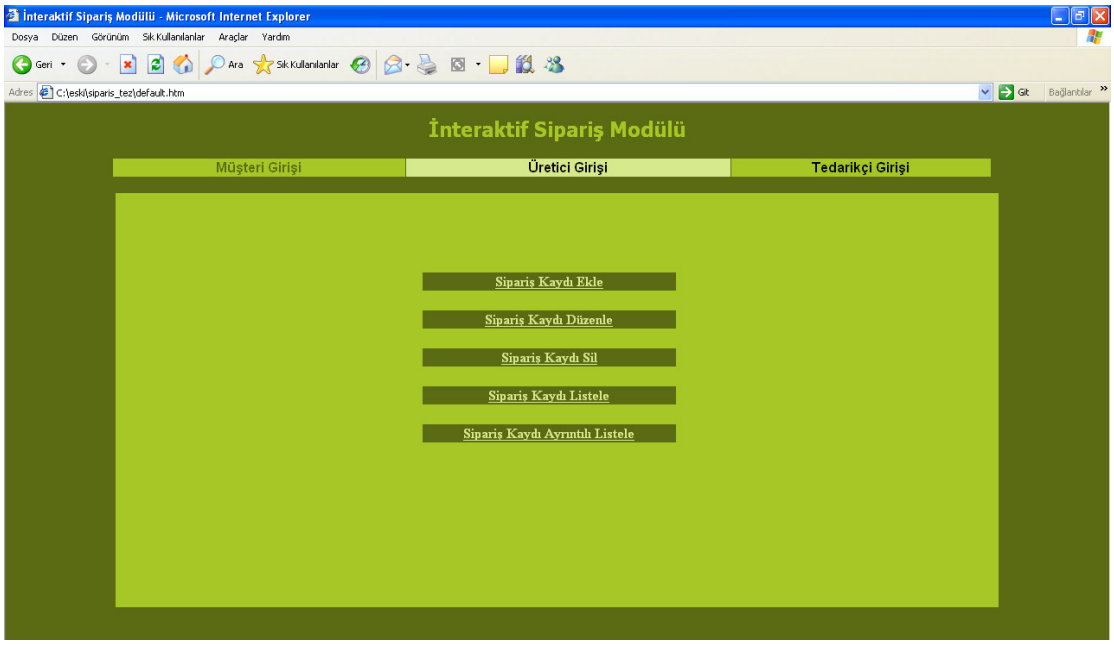
Şekil 4.1. İnteraktif Sipariş Modülü Ana Sayfa

İnteraktif Sipariş Modülü Ana Sayfasında (Şekil 4.1) Müşteri, Üretici ve Tedarikçi olmak üzere 3 adet kullanıcı giriş butonu bulunmaktadır.



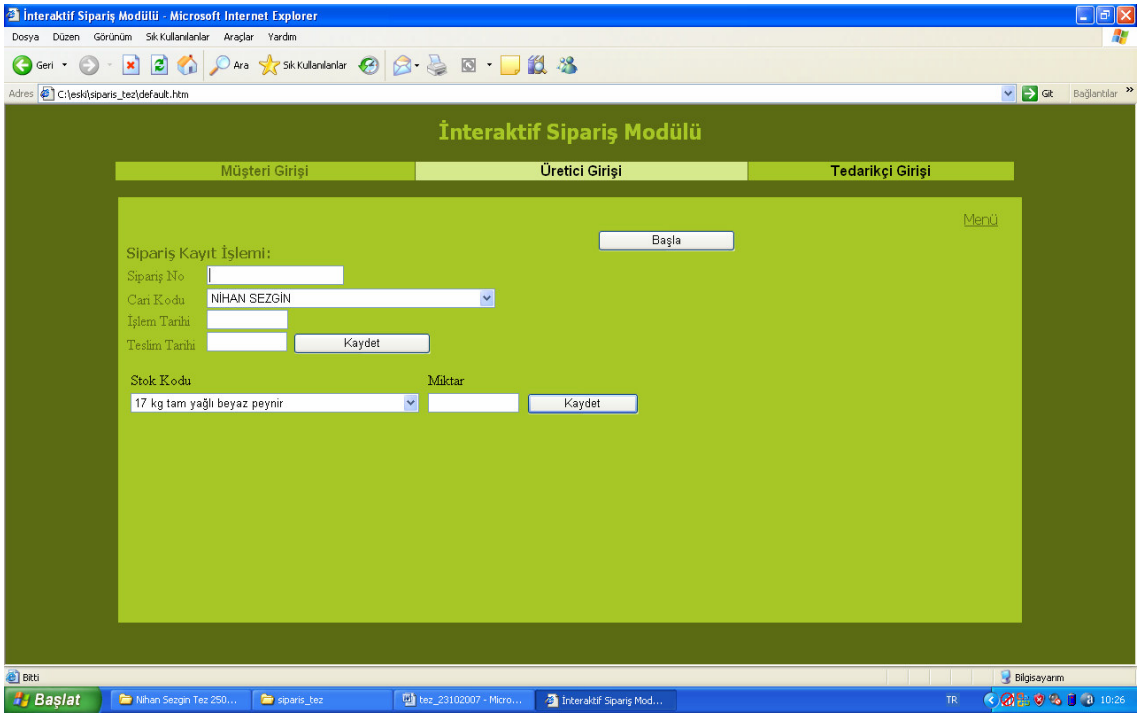
Şekil 4.2. Müşteri Girişi için Giriş Sayfası

Bunlardan herhangi birine tıklanıldığında Şekil 4.2'de görüldüğü gibi bir Login ekranı açılır. Kullanıcı Adı ve Şifre girildikten sonra Gönder butonuna tıklanarak Kullanıcılar kendilerine ait arayüze ulaşırlar.



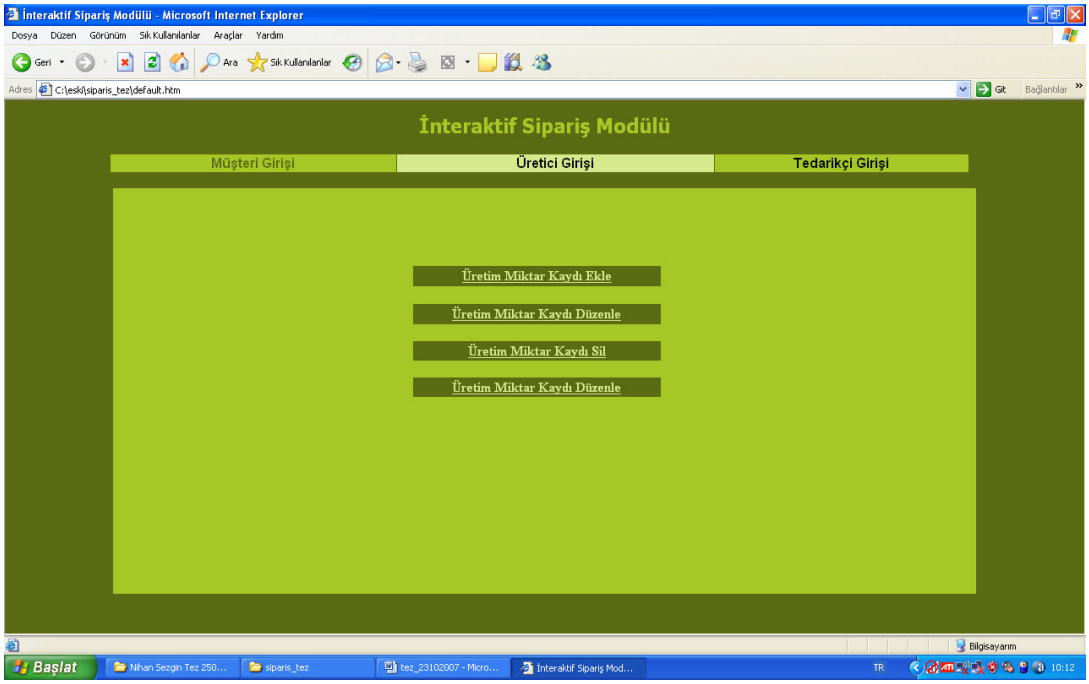
Şekil 4.3. Müşteri Ana Sayfa

Müşteriler bu arayüze (Şekil 4.3) ulaşarak Sipariş işlemlerini (Ekleme, Düzenleme, Silme, Listeleme) gerçekleştirebilirler. Şekil 4.4'de Sipariş Kayıt Ekleme işlemini görmektesiniz.



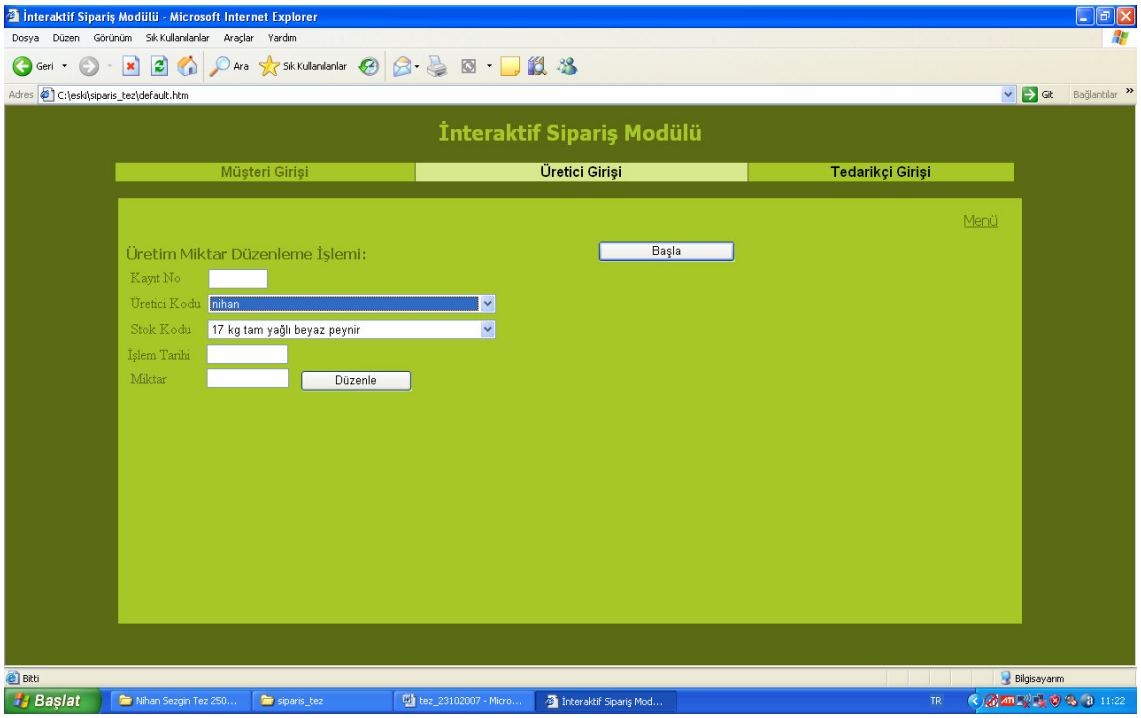
Şekil 4.4. Sipariş Kayıt İşlem Ekranı

Üreticiler de kendilerine ait login ekranından sisteme giriş gerçekleştirip üretici arayüzüne (Şekil 4.5) ulaşırlar. Ve buradan üretim miktar kayıtları ile ilgili işlemleri (Ekleme, Düzenleme, Silme ve Listeleme) gerçekleştirirler.



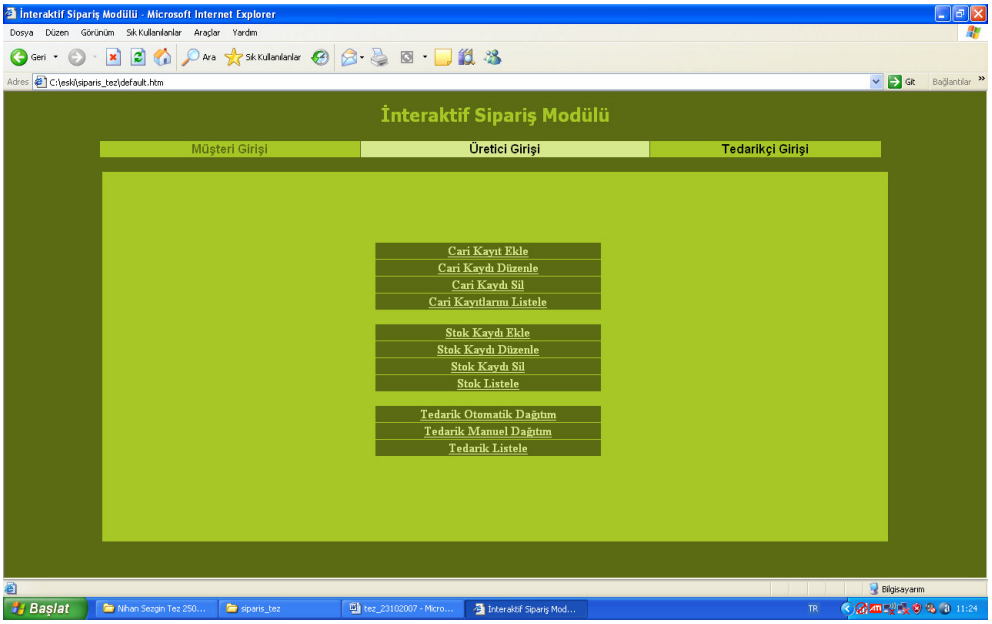
Şekil 4.5. Üretici Ana Sayfa

Şekil 4.6'da Üretim Miktar Kaydı Düzenleme işlemi ekran örneği verilmiştir.



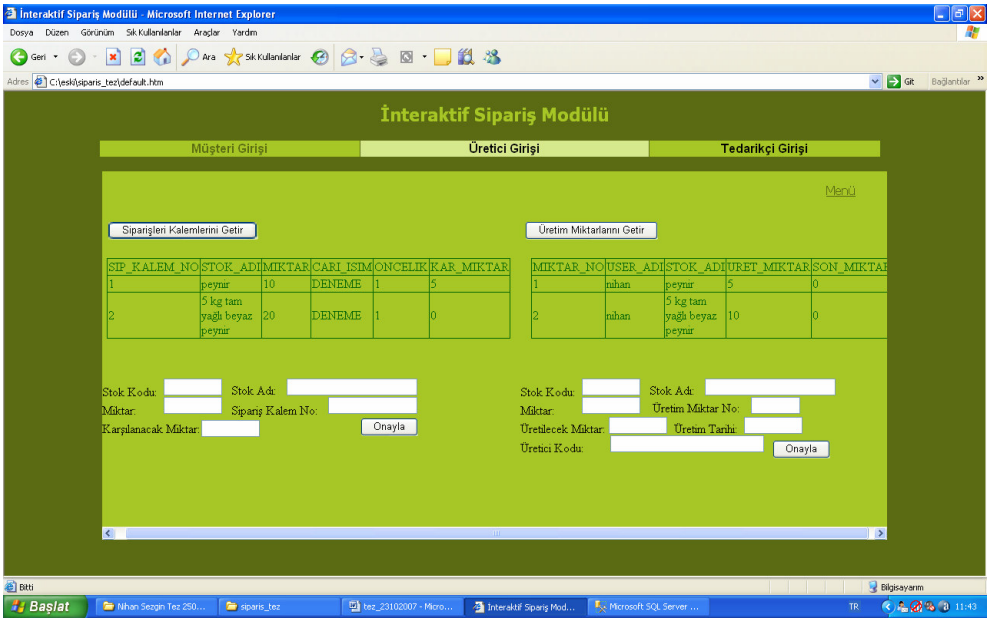
Şekil 4.6. Üretim Miktar Düzenleme İşlem Ekranı

Bu modülde en çok iş tedarikçilere düşmektedir. Müşteri kayıtları, Stok Kayıtları ve Tedarik ile ilgili tüm işlemler tedarikçi arayüzünde (Şekil 4.7) gerçekleşir.



Şekil 4.7. Tedarikçi Ana Sayfa

Tedarik işlemlerine örnek olarak Şekil 4.8'de Tedarikçi Manuel Dağıtım ekranını görebilirsiniz.



Şekil 4.8. Tedarikçi Manuel Dağıtım İşlem Ekranı

5. SONUÇLAR VE ÖNERİLER

Tedarik zincirinin yönetimi, son yıllarda, işletmeler için çok önemli bir konu olmuştur. Tedarik zincirinin etkin olarak yönetilmesi işletmelerin rekabet avantajı sağlamalarında önemli fırsatlar sağlamaktadır. İşletmelerin, tedarik zincirlerini etkin olarak yönetebilmeleri de büyük ölçüde tedarik zincirinin üyeleri arasında bilgi paylaşımının sağlanabilmesine bağlıdır.

Teknolojide meydana gelen gelişmeler ve ekonomide ve pazarlarda meydana gelen değişimler, işletmelerin tedarik zincirlerinin yönetilmesinde bilginin önemini artırmıştır. Tedarik zincirinin üyeleri arasında bilgi paylaşımında E-ticaretin önemli bir desteği olmaktadır. E-ticaret, tedarik zincirinin üyeleri arasında doğru bilginin hızlı bir biçimde akışını sağlamaktadır.

Bu çalışmada tedarik zinciri yönetimi için kullanılabilecek temel bir sipariş modülü gerçekleştirildi. Bu modül sayesinde müşteri, tedarikçi ve üretici arasındaki bilgi akışı güvenli ve hızlı bir şekilde sağlandı. Bunun için de üç katmanlı mimariden yararlanıldı. Son yıllarda büyük gelişme ve dağılım gösteren web servisleri yardımıyla da çalışma güvenli bir şekilde internet ortamına yerleştirildi. İnternet ortamında e-ticaret mantığı büyük bir hızla yaygınlaşsa da hala altyapı ve etkin kullanımda eksiklikler devam ediyor. Yapılan çalışmalar ve araştırmalarla e-ticaret mantığı iş sektöründe her geçen gün çok daha etkin bir yere sahip olacaktır.

KAYNAKLAR DİZİNİ

[1] Anderson L. D., Lee L. H. "The Internet enabled supply chain: From the "first click" to the "last mile", Information Technology Toolbox Inc. http://supplychain.ittoolbox.com/documents/document.asp?i=578.

[2] Atakan F., Kayacık G., Heywood. N. Z., Eren Ş., "E-Commerce: A Case Study of Turkey", Proceedings of Third International Conference on Telecommunications and E-Commerce (2000) http://www.cs.dal.ca/~kayacik/download/fgns-smu.doc.

[3] Amir Hartman, John Sifonis, John Kador Yayına hazırlayan Kutluk Özgüven Ağa Hazır, Ekonomideki Başarı Stratejileri, 2002.

[4] Capital-Andersen, E-business Dönüşümü, Ekim 2001.

[5] İhracatı Geliştirme Etüd Merkezi, Kobilerin Uluslar arası Rekabet Güçlerini Artırmada Tedarik Zinciri Yönteminin Önemi, Ekim 2004.

[6] Atakan F, Kayacık G, Eren Ş, **"**Firmalar Arası Elektronik Ticaret Ve Tedarik Zinciri Yönetiminde Gezici Etmen Teknolojisinin Kullanımı**"**, 2001

[7] Ayköse Melike, Güçlü Başar, www.turk.internet.com, "Etkin Tedarik Zinciri Yönetimi", Aralık 2003.

[8] Yüksel Hilmi, "Tedarik Zinciri Yönetiminde Bilgi Sistemlerinin Önemi", Dokuz Eylül Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, Sayı:3, 2002.

[9] Microsoft, 2003, ASP.NET, http://www.asp.net.

[10] Han Cengiz, "ASP.NET Uygulamalarında Güvenlik"

[11] Özkaya Mehmet, Erciyes Üniversitesi, "Üç Katmanlı Mimari ile Basit bir Uygulama Geliştirmek".

[12] Gözüdeli Yaşar, "Nesneye Dayalı Programlama, 3 Katman Mimarisi ve ASP.NET", Ağustos 2004.

[13] Şenyurt, Burak Selim, "Stored Procedure (Saklı Yordam) Yardımıyla Yeni Bir Kayıt Eklemek", Şubat 2004.

[14] http://www.c-sharpnedir.com

[15] http://www.godoro.com/Divisions/Ehil/Mahzen/Programming/ThreeTierArchitectural /txt/html/category_top.html