
# Bir Sistemin Tasarlanış Hikayesi

## KurumsalJava.com

Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

# Okunması Tavsiye Edilen Diğer Makaleler

- DAO Tasarım Şablonu
  http://www.kurumsaljava.com/2008/12/01/data-access-object-dao-tasarim-sablonu/
- Web Aplikasyonlarında Yüksek Performans İçin Caching Mekanizmaları
  http://www.kurumsaljava.com/2008/11/28/web-aplikasyonlarinda-yuksek-performans-icin-caching-mekanizmalari/

# Giriş

BizimAlem.com 2001 ocak ayında başlamış olduğum bir web projesi. Amacım, Avrupa'da yaşayan 5 milyondan fazla Türk kökenli vatandaşımız için bir araya gelebilecekleri bir sanal ortam oluşturmaktı. Bu doküman çok basit koşullarda başlanılan bir web projesinin hangi boyutlara ulaşabileceğini ve zaman içinde yapılan eklemelerle oluşturulan teknik mimarinin hangi evrelerden geçtiğini ihtiva etmektedir. BizimAlem.com zaman içinde Türkiye ile Avrupa arasında bir sanal köprü olmayı başarmış ender web projelerinden birisidir.

İsterseniz bu proje hakkında fikir edinebilmeniz için bazı teknik bilgiler aktarayım. BizimAlem hakkında bugün (14.1.2009) itibariyle bazı teknik veriler şu şekildedir:

- 550.000 kayıtlı üye,
- Günlük ortalama yeni kayıt olan üye adedi 1000-1500,
- 40 a yakın sunucu, switch, loadbalancer, firewall sistemleri,
- Tamemen J2EE (Java 1.5) ve Open Source tabanlı,
- Tüm sunucular Linux işletim sistemiyle çalışıyor,
- Günlük ziyaretçi sayısı 55.000 – 60.000 civarında,
- Günlük sayfa gösterim adedi 1.1 milyon civarında.

BizimAlem için oluşturduğum yazılım sistemi şu komponentlerden oluşuyor:

- Forum,
- Blog,
- Okey, Tavla, Batak, Bilardo gibi multiplayer oyunlar,
- Youtube vari video modülü,
- Gruplar,
- Haberler,
- Anketler,
- Sütun,
- Arkadaş listesi,
- Kişiselleştirilebilen profil sayfaları,
- Sanal hesap,
- VIP üyelik,
- E-Card,
- Limitsiz fotograf albümleri,
- Mesaj merkezi,
- Takvim,
- Yönetim paneli (Dashboard),
- Sanal hediyeler,
- Shop,
- Favori üye listesi,
- Ziyaretçi listesi,
- Online üyeler için alt panel mesaj gönderme,
- Detaylı arama modülü,
- Yorumlar,
- vb. diğer modüller

Bu teknik bilgiler BizimAlem hakkında bir fikir sahibi olmanızı kolaylaştıracaktır. İsterseniz gelin, bu projenin başladığı ilk güne, 2001 senesinin mart ayına dönelim ve projenin hangi şartlarda start aldığını görelim.

# İlk Versiyon (v.1.0)

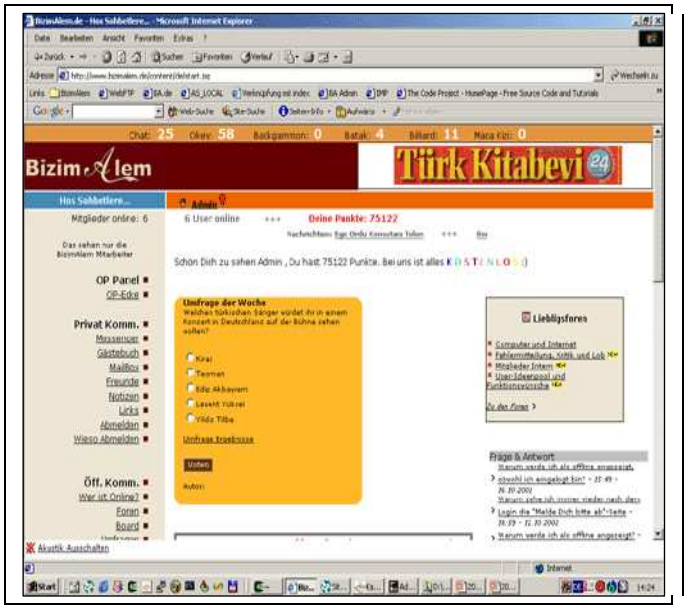
BizimAlem.com için çalışmalara Mart 2001 de başladım. Projenin ilk web adresi BizimAlem.de idi, çünkü bu web platformunu Almanya'da yaşayan Türk vatandaşları için planlamıştım. Bu sebepten dolayı BizimAlem'in ilk versiyonunda yer alan arayüzler Almanca dilinde idi. Almanya'da yaşayan genç Türkler Almanca ve Türkçe'yi hemen hemen aynı seviyede konuşabildiklerinden, BizimAlem'in ilk versiyonun da tamamen Almanca olması bir sorun teşkil etmedi. Platformun Almanca olmasının başka bir sebebi daha vardı. Almanya'da yaşayan 2.5 milyon Türk'ün yanısıra 80 milyonluk Alman toplumu vardı ve bir şekilde onlara da bu platform üzerinden ulaşmak istiyordum. Bu gibi nedenlerden dolayı ilk versiyon tamamen Almanca oldu.

1999 senesinden beri Java platformunu kullanarak freelancer olarak web projelerinde çalışma fırsatı buldum. Java'nın neler sağladığını yakından görme fırsatım olduğu için, Bizimalem'i tamamen Java platformu yardımıyla oluşturmaya karar verdim. 2000 senesinden sonra her yerde kullanılmaya başlanan Java EJB [1] teknolojisi BizimAlem için ideal gibi görünüyordu. Tabii daha sonralar acı tecrübeler yaparak ögreneceğim birşey vardı, oda bir web projesinin kesinlikle EJB gibi o tarihte henüz ermemiş bir teknoloji ile yapılmaması gerektiğiydi. Ne yazık ki çok ideal koşullardan yola çıkarak, programcı olarak tecrübesiz olmamın da etkisiyle yanlış bir teknolojiyi kullanmaya karar verdiğimi çok daha sonralar, birinci versiyonu tamamen çöpe atıp, herşeyi sil baştan yeniden yapmak zorunda kaldığımda anlamış oldum. EJB kullandığım ilk versiyon her açıdan performans problemleri ile karşılaştığım versiyondur. Bu ilk versiyon aslında gözümü yıldırmadı değil. Lakin daha iyi bir sistem ortaya koyabileceğimi düşünmem, devam etmemi kolaylaştırdı.

21.11.2001 tarihinde ilk BizimAlem versiyonu online oldu. Test sisteminde oluşturduğum ilk logo aşağıda yer almaktadır. Bu logo yerini kısa bir zaman sonra resim 2 de yer alan logoya bıraktı.


**Resim 1** İlk BizimAlem logosu

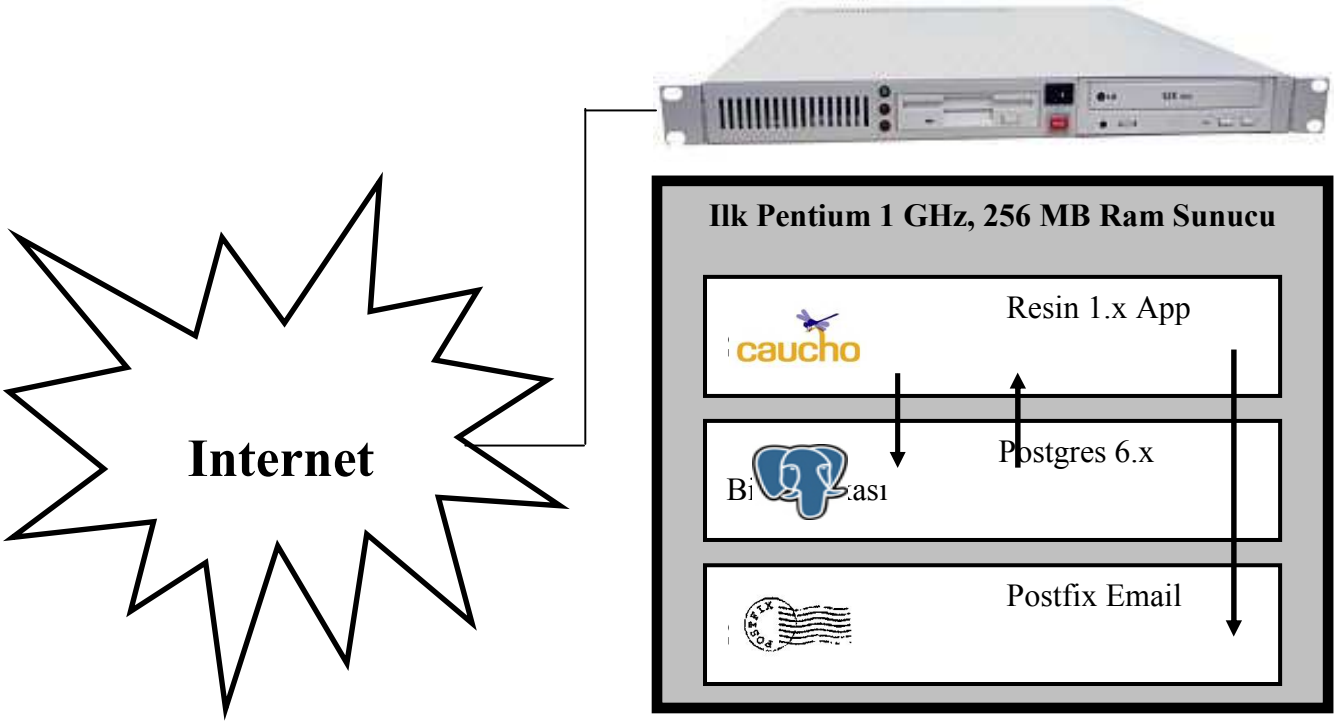
[1] Bakınız: http://java.sun.com/products/ejb/




**Resim 2** İlk tasarım

Resim 2 de BizimAlem'in ilk web tasarımı yer almaktadır. İlk versiyonu oluşturmak için kullandığım teknolojik öğeler şöyledir:

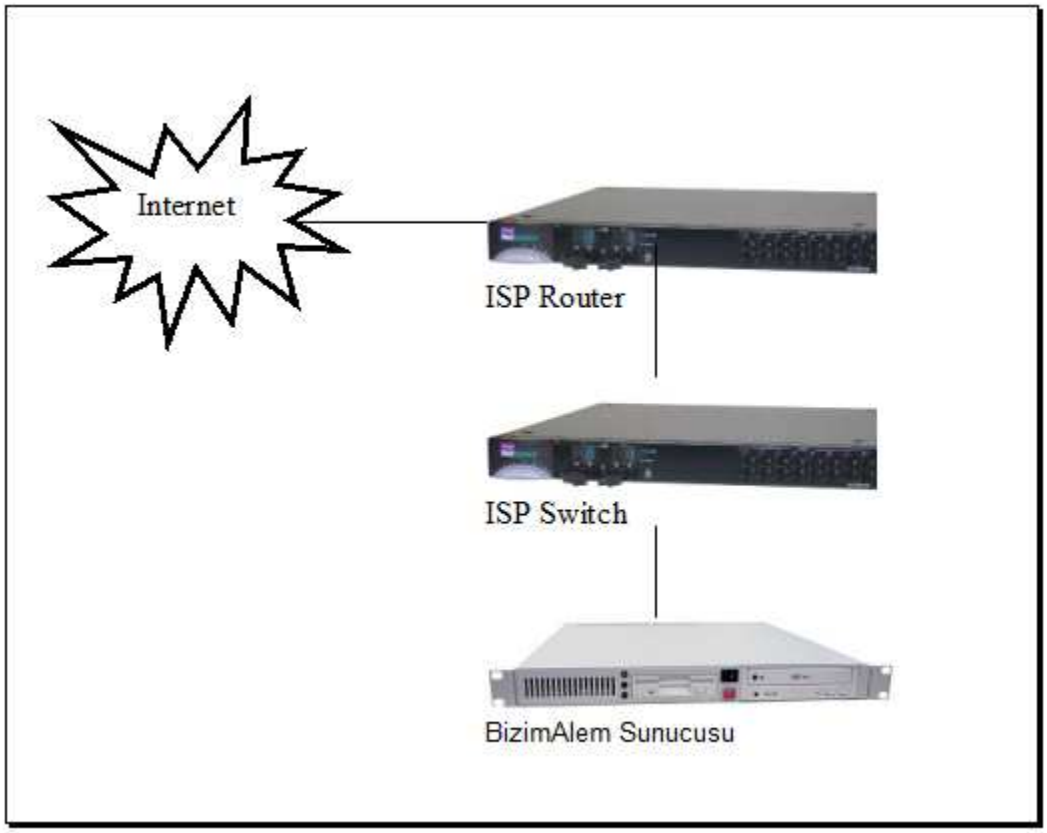
- Java 1.4
- EJB 1.1
- JBoss / Tomcat App Server
- Postres 6.x bilgibankası
- Suse Linux 6.x işletim sistemi
- Postfix email sunucu




**Resim 3** İlk versiyonda kullanılan sunucu

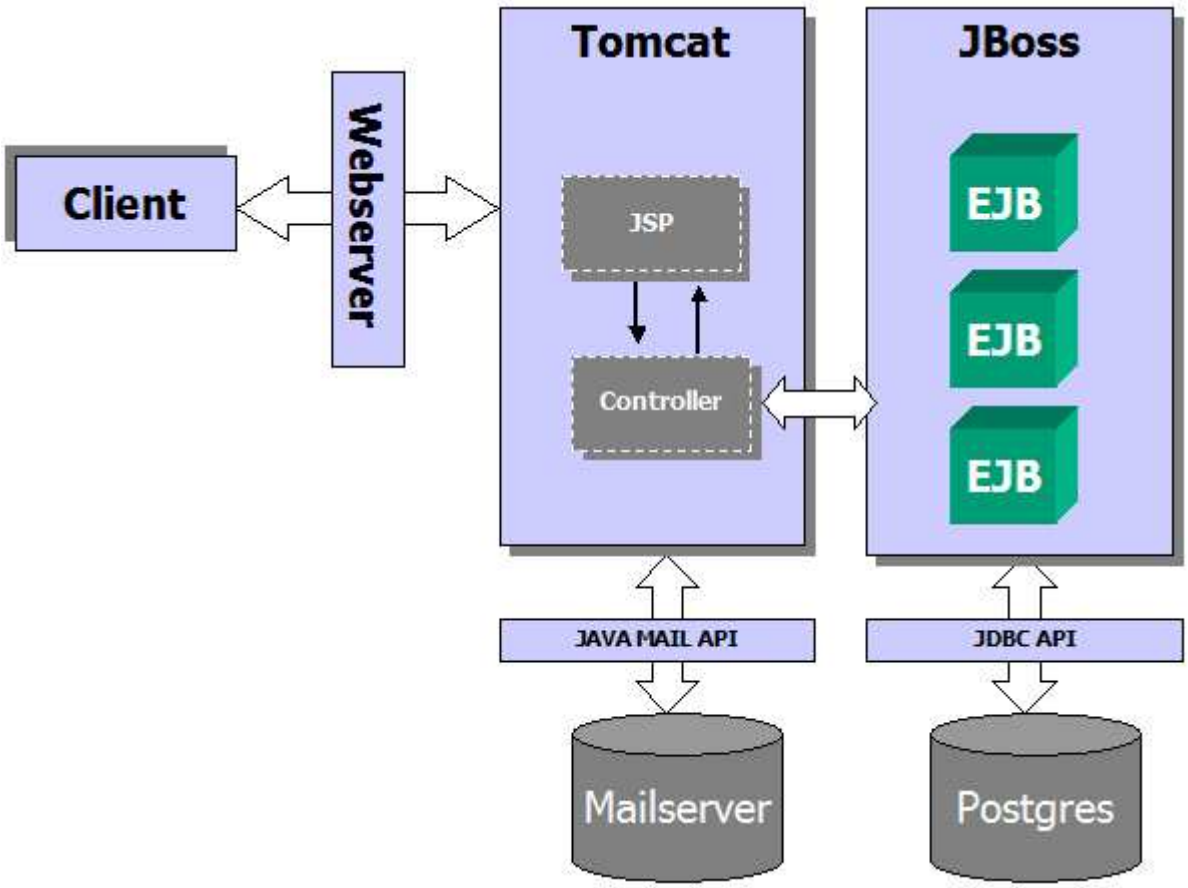
BizimAlem'in ilk versiyonunda kullandığım sunucu resim 3 ve 4 de yer almaktadır. İlk versiyon için kullandığım sunucuyu gerekli bilgisayar bileşenlerini satın alarak bir araya getirmiştim. Yazılımcı olmama rağmen, bu benim ileride server sistemlerini ve bileşenlerini daha iyi anlamamı kolaylaştırdı. Bu ayrıca benim iki şapkayı birden taşımamı sağladı. Bir şapkayı taktığımda yazılım mühendisi oluyordum ve sistem için gerekli yazılımı yapıyordum, diğer şapkayı taktığımda bilgisayar ağı ve server bileşenlerinden anlayan bir mühendis oluyor ve sunucu ve ağ üzerinde gerekli işlemleri uyguluyordum. BizimAlem aslında benim için kocaman bir laboratuar haline dönüşmüştü ve çok değişik disiplinlerde çalışma imkanı buluyordum. Bu makeleyi yazdığımda sahip olduğum teknik bilginin temelleri BizimAlem laboratuarında değişik konularda edindiğim tecrübelerle atılmış oldu. Artık sadece yazılımcı değildim. Mecbur olduğum için başka disiplinlerde de tecrübe edinmek ve oluşan sorunları çözmek zorundaydım. Şimdi geriye baktığımda, BizimAlem sayesinde çok değişik tecrübeler edinme firsatı bulduğumu görüyorum ve bunun için minnettarım.


**Resim 4** İlk versiyonda kullanılan sunucu. Hala test sunucusu olarak kullanımda.




**Resim 5** BizimAlem için kullanılan sunucunun internet bağlantısı

Doğal olarak bir sunucuyu internete bağlamak yeterli değil. Bunun yanısıra sistem yazılımının tamamlanmış olması gerekiyor. İlk versiyon için oluşturduğum mimari resim 6 de yer almaktadır.




**Resim 6** BizimAlem teknik yazılım mimarisi

Şimdi geriye baktığımda EJB teknolojisini kullanmamı şu şekilde açıklayabiliyorum: EJB yeni oluşmuş bir teknolojiydi ve herkes herhangi bir şekilde kullanıyordu ya da kullanmak istiyordu. Doğal olarak sizde aynı trene binmek istiyorsunuz ve bu teknolojinin getirisini ve götürüsünü bilmeden kullanmaya başlıyorsunuz. Ne yazik ki benim EJB serüvenim o zamanlar hüsranla bitti, çünkü yüzlerce insanın aynı anda kullandığı bir sistemde EJB 1.1 tarzı aplikasyonlar ne yazık ki performans sorunları doğuruyor! Yaşadığım performans sorunlarını şu şekilde sıralayabilirim:

- Bir EJB komponent (1.x versiyonunda) ile sadece Remote Interface üzerinden iletişim kurulduğu için, bu Tomcat içinde bulunan JSP / Controller sınıfları ile JBoss içinde bulunan EJB komponentlerin bağlantı ve işlem süresini uzattı.
- Yüzlerce insanın online olduğu bir sistemde EJB komponentleri yetersiz kalmaya başladı. Pooling mekanizmaları kullanmama rağmen bir JBoss App Server içinde bulunan EJB (Stateless Session Bean) komponentler aynı anda sadece 25-30 Thread tarafından kullanılabilir oldular. Kapasiteyi yükseltmek için kullanılan server adedini artırmam gerekti.

Performans sorunlarının yanısıra yazılım esnasında da çok zorluklar çektim. EJB komponentler ne yazik ki test edilebilir yapıda değiller. Bu en azından EJB 3.0 öncesi geçerli olan bir durumdu. Test etmek ve hataları bulmak için debugging yapmak gerçekten çok sabır isteyen iştir.

Uzun bir müddet performans sorunlarıyla ugraştıktan sonra, EJB komponentlerine veda etmem gerektiğini anladım ve BizimAlem'in ikinci versiyonu için kolları sıvadım.

# İkinci Versiyon (v.2.0)

2002 senesinin ortalarından itibaren BizimAlem.com v.2.0 için çalışmalara başladım. v.1.0 versiyonu benim için pekte tekrar kullanılır yapıda değildi, çünkü kodun merkezinde EJB komponentleri vardı ve ben EJB teknolojisinden tamamen uzaklaşmak istiyordum. Bunun yanısıra kodun bakımı ve geliştirilmesi kolay olmalıydı. Bu arada üye sayısı 2000 i aşmıştı ve aynı anda 100 ün üzerinde üye online oluyordu. Yeni sistemdeki performans sorunlarını ortadan kaldırmak için bir analiz yaptım. Analizin sonuçları şunları gösterdi:

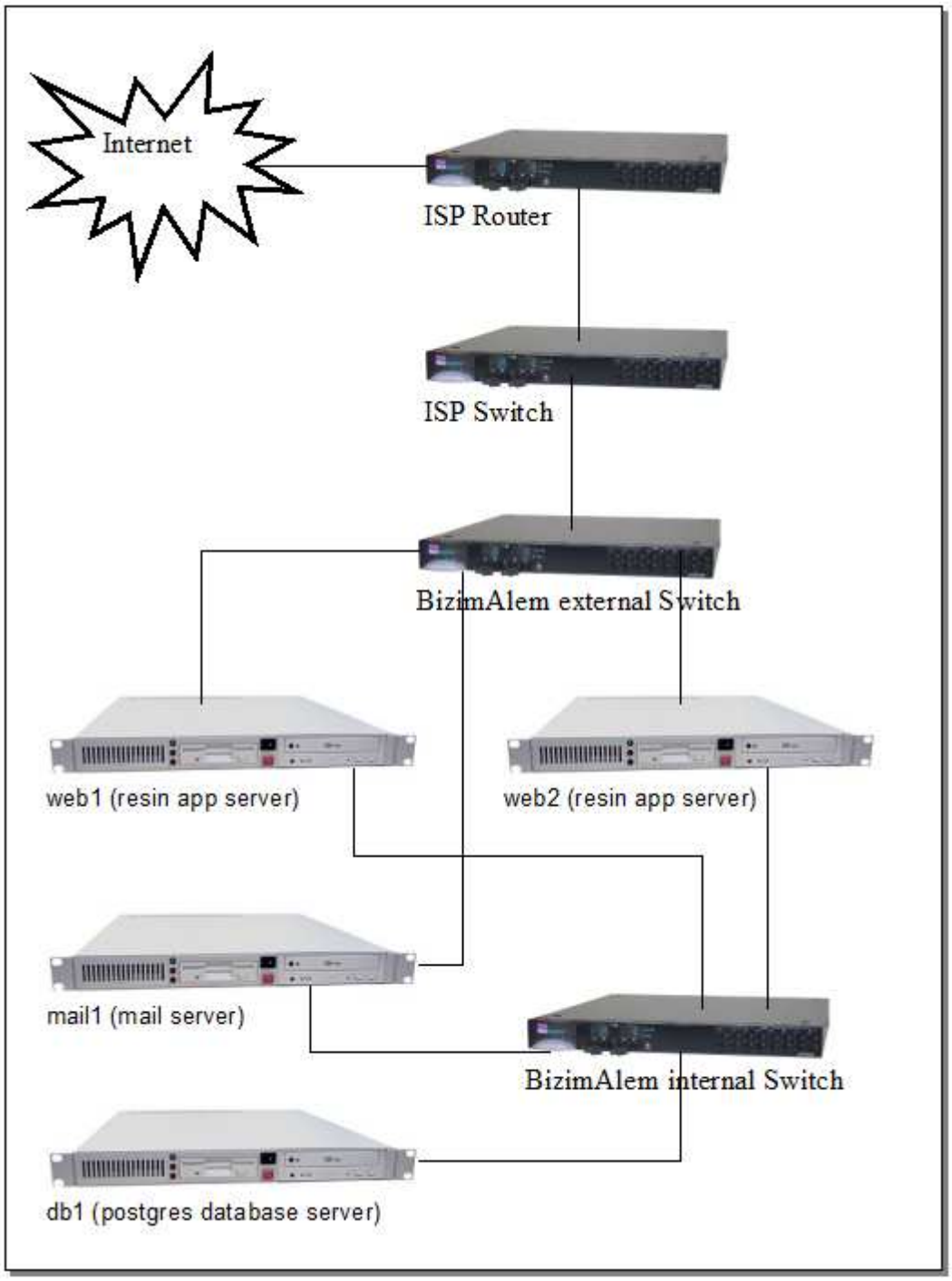
- Bilgibankası için ayrı bir sunucu kullanılması gerekiyor. Bu bilgibankası için yapılan işlemlerin performansını artırır.
- Email gönderimi için ayrı bir sunucunun kullanılması gerekiyor. Bu şekilde sistemin diger bölümlerini etkilemeden email gönderimi performansı artırılabilir.
- JBoss yerine yeni bir application server kullanılması gerekiyor, çünkü EJB kullanılmadığına göre JBoss gibi bir EJB containere olan ihtiyaç ortadan kalkmıştır. Yeni application serveri olarak Caucho Resin [2] 1.x serisini seçtim. Resin performansı yüksek olan bir Servlet / JSP containerdir, yani Tomcat gibi bir application server.
- Web aplikasyonu iki değişik Resin application server içinde paralel çalıştırabilirsem, gelen yükü bu iki server arasında paylaştırabilirim. Bu sebepten dolayı web aplikasyonu (v.2.0) için en az iki yeni sunucuya ihtiyacım var. Bu şekilde online olan üyelerin adedi 500 e kadar yükselebilir. Sunucu kapasitesi arttıkça, hizmet verilebilecek online üye adedi de artar.

İlk versiyonda sadece bir sunucu ile yetinirken, artan üye sayısıyla beraber genel sistem performansını yükseltebilmek için birden fazla ve sorumluluk alanları tanımlanmış olan sunuculara ihtiyacım olduğunu anladım. Artık BizimAlem projesi kendi bilgisayar ağını kurma aşamasına gelmişti. Birden fazla IP adresine ve çok ufak bir network adres alanına ihtiyacım vardı. Müşterisi olduğum ISP bana kendi C class network adres alanından aşağıdaki IP adreslerini tahsis etti.

- 213.221.93.10 – web1 isimli sunucu.
- 213.221.93.11 – web2 isimli sunucu
- 213.221.93.12 – mail1 isimli sunucu
- 213.221.93.1 – default route
- 213.221.9.255 – default netmask

Bilgibankasını da kendine özel bir sunucu üzerinde çalıştırdığım taktirde, genel sistem performansı çok daha iyi olabilecekti. Bu yüzden gerekli serverleri satın aldıktan sonra, aşağıda yer alan ağ planını oluşturdum.

[2] Bakınız: http://www.caucho.com




**Resim 7** BizimAlem v.2.0 ağ planı

Resim 7 de görüldüğü gibi iki yeni ağ oluşturmam gerekiyordu, çünkü bilgibankasını dış dünyaya açmak istemiyordum. Bu yüzden her sunucuya iki ağ kartı takarak biri üzerinden dış ağa (BizimAlem external Switch) diğeri üzerinden bilgibankasının bulunduğu iç ağa (BizimAlem internal Switch) bağlanmalarını sağladım. web1 ve web2 sunucuları iç ağ üzerinden bilgibankası işlemlerini gerçekleştiriyorlar. İç ağ için aşağıdaki IP şemasını kullandım:

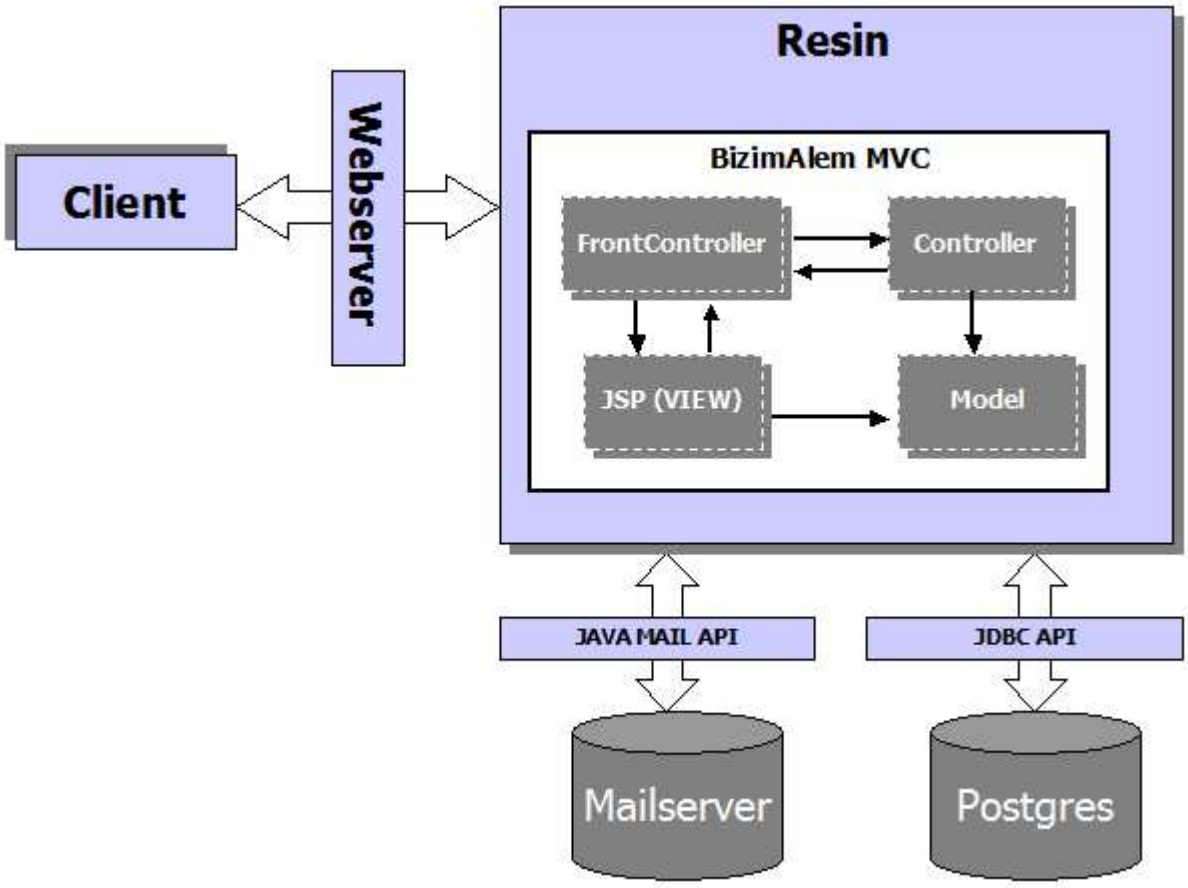
- IP Range: 192.168.1.1 – 213.221.93.255
- Networkmask: 192.168.1.255
- Default gateway: 192.168.1.1

Bu şemaya göre sunucuların sahip oldukları IP adresleri şu şekilde oldu:

- web1 dış ağ IP adresi: 213.221.93.10
- web1 iç ağ IP adresi: 192.168.1.10
- web2 dış ağ IP adresi: 213.221.93.11
- web2 iç ağ IP adresi: 192.168.1.11
- mail1 dış ağ IP adresi: 213.221.93.12
- mail1 iç ağ IP adresi: 192.168.1.12
- db1 iç ağ IP adresi: 192.168.1.2

BizimAlem v.2.0 için dört yeni sunucu (server) ve iki Switch kullanıldı. Bu şekilde network komponent adedi altıya çıkmış oldu.

Ağ oluşturma çalışmaları yanısıra v.2.0 için yazılım çalışmalarını da devam ettiriyordum. Kısa bir zaman sonra aşağıdaki mimari yapı oluştu.




**Resim 8** BizimAlem v.2.0 yazılım mimarisi

BizimAlem v.2.0 ile yeni bir yazılım konsepti ile tanıştı. Web projelerinde yazılımı kolaylaştırmak için genelde Model View Controller (MVC) tasarım şablonu kullanılır. MVC tarzı çalışan web frameworklerde kullanıcı istekleri merkezi bir Controller (FrontController) sınıfı tarafından karşılanır. Controller sınıfı validasyon ve navigasyon gibi işlemlerden sorumludur. JSP sayfalarında gösterilmesi gereken veriler Model sınıflarında tutulur. Controller, verileri ihtiva eden model nesnelerini HttpServletRequet, HttpServletResponse ya

da HttpSession (Java Servlet API sınıfları) sınıfları aracılığıyla JSP sayfaları tarafından kullanılır hale getirir. Modellerin ihtiva ettiği veriler View olarak isimlendirilen JSP sayfalarında gösterilir. JSP sayfaları HttpServletRequest, HttpServletResponse ya da HttpSession aracılığıyla gerekli model nesnelerine ulaşır. Çoğu web framework JSTL taglerini kullanarak, edinilen verilerin JSP sayfalarında gösterimini gerçekleştirir. JSP sayfalarının model nesnelerine erişimi frameworkün implementasyon tarzına göre değişen bir durumdur.

Ben o zamanlar (2002) Struts[3] gibi piyasada bulunan bir MVC web framework kullanmak yerine kendi MVC frameworkünü implemente etmeyi uygun gördüm. Bu çalışmaların sonunda Struts gibi çalışabilen BizimAlem MVC web frameworkü oluştu. Çok basit bir şekilde implemente ettiğim BizimAlem MVC web framework validasyon ve navigasyon gibi işlemleri kolaylıkla yapabilecek yapıdaydı. Kullandığım Controller sınıfları basit POJO (Plain Old Java Object) tarzı sınıflardı. Artık v.2.0 bünyesinde hiçbir EJB komponenti kullanılmıyordu ve JBoss yerini Resin application servere bırakmıştı. web1 ve web2 isimlerinde iki Resin application server üzerinden BizimAlem'in iki kopyası paralel olarak çalışmaktaydı ve kullandığım BizimAlem external Switch üzerinden gelen kullanıcıları web1 ya da web2 ye otomatik olarak yönlendirebiliyordum. Bu şekilde sistem kapasitesini, gerekli olduğu durumlarda örneğin web3 isminde yeni bir sunucu ekleyerek artırabilecektim. Bu ileride Hardware Loadbalancer komponentleri kullanarak çok geniş bir kullanıcı kitlesine hitap edebilecek bir altyapının temellerini oluşturdu. Yazılım esnasında da, oluşan sistemin birden fazla server üzerinde paralel çalışabilmesine dikkat ettim. Oluşan sistem bulunduğu servere bağımlılık oluşturmadan çalışabiliyordu ve bu şekilde yeni sunucular ekleyerek sistem kapasitesini artırmak kolaylaşmıştı.

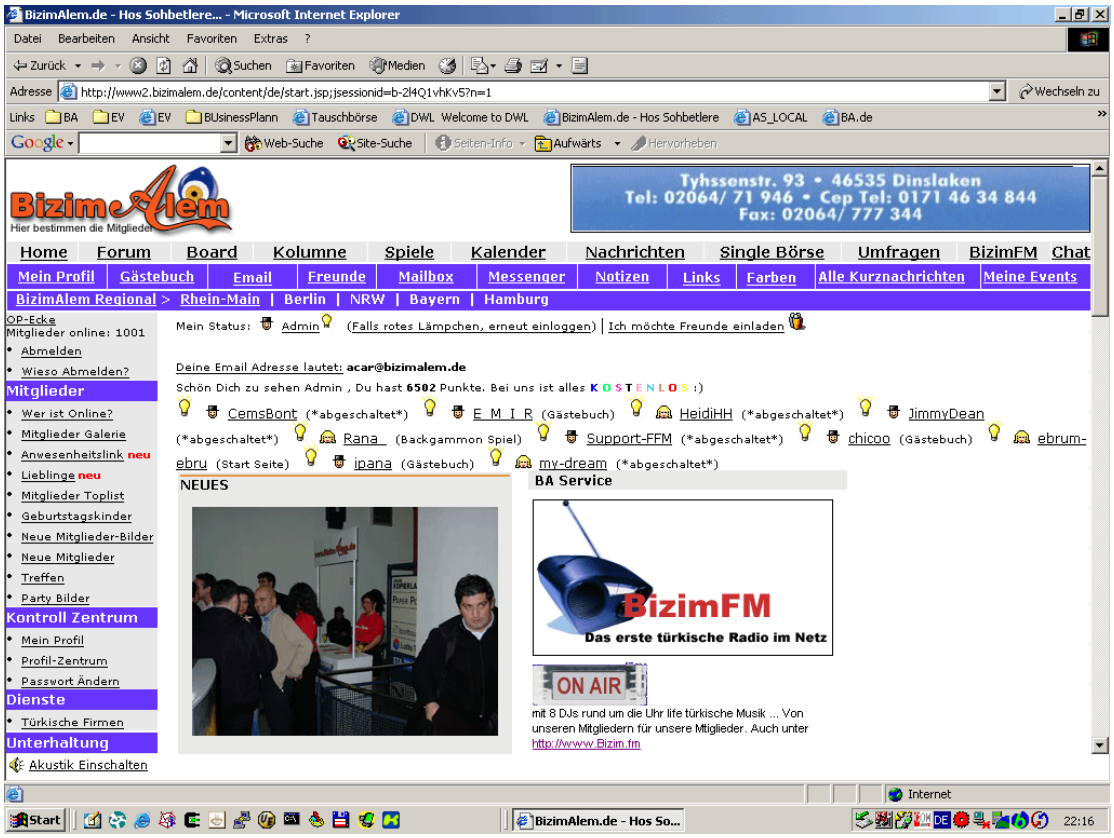
BizimAlem v.2.0 ile v.3.0 arasındaki versiyonlarda kullandığım bazı BizimAlem logoları ve tasarım çalışmaları aşağıda yer almaktadır.


**Resim 9** Tarihi bir BizimAlem logosu


**Resim 10** Tarihi bir BizimAlem logosu

[3] Bakınız: http://struts.apache.org/

**Resim 11** Tarihi bir BizimAlem logosu




**Resim 12** Tarihi bir BizimAlem tasarımı


**Resim 13** Tarihi bir BizimAlem tasarımı

BizimAlem v.2.0 2002 sonunda online oldu ve performans sorunlarını büyük ölçüde giderdi. Kısa bir zaman sonra kayıtlı üye sayısı 40.000 i, Aynı anda online olan üye sayısı 1000 i geçti. Herşey yolunda gidiyor gibi görünüyordu ve platform günden güne devamlı büyüyerek, sahip olduğu altyapı kapasitesinin sınırlarını zorlamaya başladı.

Sistemin güvenliğini sağlamak için her sunucu üzerinde Linux Firewall sistemini aktive etmem ve bakımını yapmam gerekiyordu. Bu zaman içinde sunucu sayısı arttıkça zorlaşan bir işlem haline geldi. Merkezi bir firewall sisteminin gerekli olduğunu kısa bir zaman sonra BizimAlem Hackerlerin hedefi haline geldiğinde anlamış olacaktım.

Ne yazik ki ağ ve işletim sistemi yönetiminde tecrübesiz olduğum için, düzenli aralıklarla Linux işletim sistemi için gerekli yamaların (patch) yapılması gerektiğini çok geç öğrendim. Bir takım niyetleri iyi olmayan şahışların BizimAlem serverlerini ellerine geçirmelerine mani olamadım ve bir müddet bu şahıslarla beraber (co-existing) yaşamak zorunda kaldık. Bu şahıslar serverleri ellerine geçirdikleri için, sistemi istedikleri şekilde kullanabiliyorlardı. Bunun bu şekilde gitmeyeceğini ve BizimAlem için yeni bir güvenlik konsepti geliştirmem gerektiğini anladım ve çalışmalara başladım.

Öncelikle bilgisayar ağlarında bulunan sunucuların ele geçirilmeleri hakkında kitaplar okudum ve gerekli bilgiyi edindim. Eğer saldırı yöntemlerini bilmesseniz, sistemi nasıl korumanız gerektiği hakkında pek fazla bir bilginiz olmaz. Bu yüzden kendinizi saldırıcı şahıslar yerine koyarak, nasıl çalıştıklarını anlamanız gerekiyor. Bu şekilde sistemin açıklarını da keşfederek, kısa zamanda gerekli tedbirleri almanız mümkün olur.

Altı aya varan „Hacker nasıl olunur“, - „Sunucular hackerlere karşı nasıl korunur“ eğitimimden sonra BizimAlem güvenlik konsepti üzerinde çalışmaya başladım.

Bu yazının başında da belirttiğim gibi BizimAlem benim için büyük bir ihtisas alanı haline geldi. Normal şartlarda yazılımcı olarak ilgilenmeyeceğim konular benim için faaliyet alanı

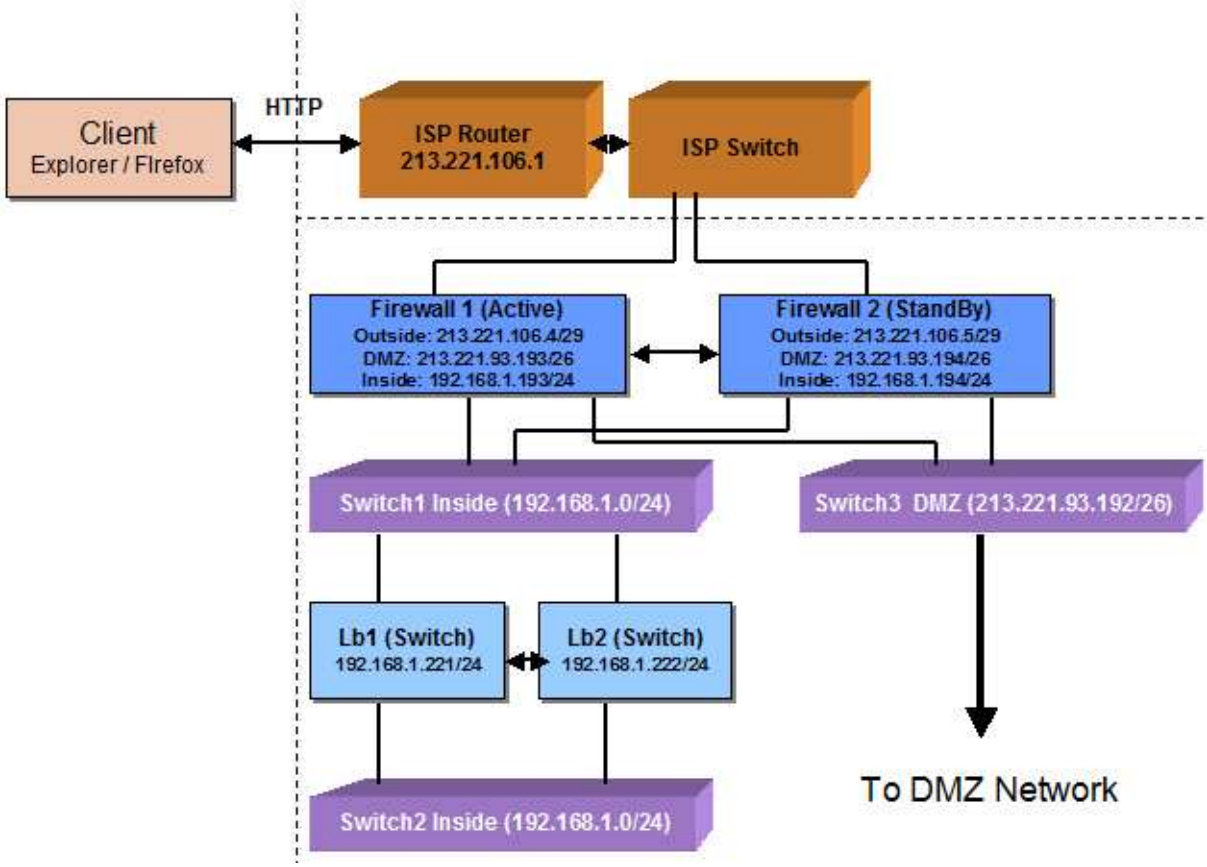
olmaya başladı, çünkü oluşan her türlü sorunu kendi başına çözmek zorundaydım. Aslında bunu da çok severek yapıyordum, çünkü bu çok zevkli bir uğraştı.

Yeni güvenlik konseptim aşağıdaki maddelerden oluşmaktaydı:

- Güçlü bir merkezi firewall sistemi ile tüm sistemin korunması gerekli. Her sunucunun tek başına lokal bir firewall sistemi tarafından korunması yeterli değil.
- İşletim sistemi düzenli olarak yamalanarak (patch) en aktüel hale getirilmek zorunda. Aksi taktirde üçüncü şahışlar sistem açıklarını kullanarak, sistemi sabote edeceklerdir.
- Kullanılan open source komponentlerin en son versiyonları serverlere kurulmalı. Eski versiyonlarda bulunan açıklar üçüncü şahışlar tarafından kullanılarak sistem sabote edilebilir.

İlk işlem olarak 25.000 EUR civarında yatırım gerektiren ve yüksek direnme ve koruma gücüne sahip firewall komponentlerini satın aldım. Firewall sistemi iki sunucudan oluşuyordu ve beraber devamlı çalışma özelliğine sahip bir ikili oluşturuyorlardı. Heart Beat olarak bilinen bir mekanizma ile firewall sistemini oluşturan komponentler birbirlerine bağlıdır ve hayatta olup olmadıklarını karşılıklı olarak kontrol ederler. Eğer aktif olan firewall komponenti devre dışı kalırsa, diğer komponent onun yerine geçerek, işlemlerin devam etmesini yani trafiğin akmasını sağlar.

Yeni güvenlik konsepti dogrultusunda BizimAlem için gerekli ağı yeniden yapılandırdım. Bir sonraki resimde yeni ağ planı yer almaktadır.




**Resim 14** BizimAlem ağ planı

Yeni ağ planına göre, sistemin tümüne erişim yeni firewall sistemi üzerinden gerçekleşmekteydi. Tüm trafik firewall komponentlerinden transit geçiş yaptığı sürece, sistemin güvenliğini optimal sağlayabilirdim.

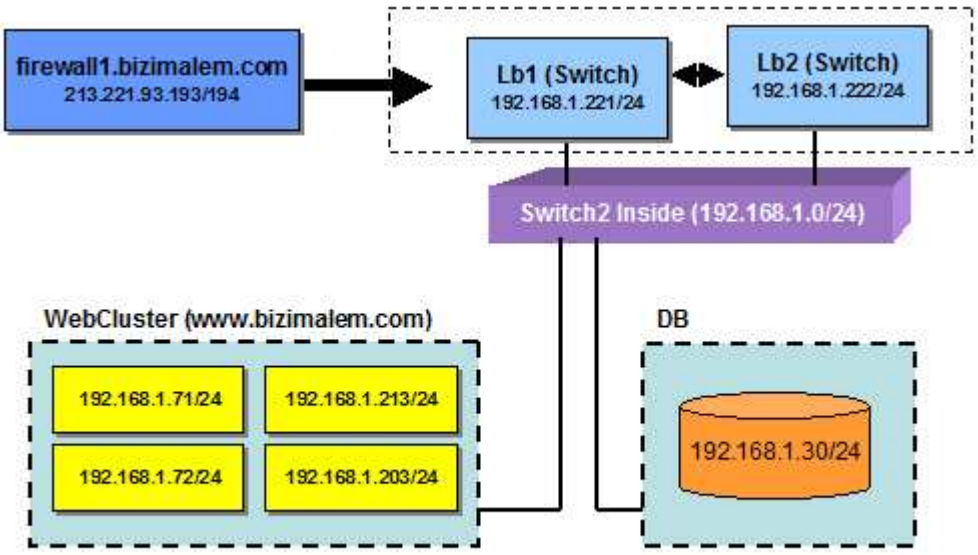
Yeni sistemi oluşturabilmek için bulunduğum ISP (Internet Service Provider) ile fikir alışverişinde bulunmaya başladım. Doğal olarak artık kendi kullanabileceğim bir alt ağa (subnetwork) ihtiyacım vardı ve ISP bana gerekli IP adres alanını tahsis etmek durumundaydı. Görüşmeler sonunda aşağıda yer alan IP adres alanı bana tahsis edildi.

- IP Range: 213.221.93.192 – 213.221.93.255
- Netmask: 213.221.93.192
- Default Gateway: 213.221.93.193
- DNS: 212.82.225.7

Bana tahsis edilen IP alanında 61 IP adresini kullanabiliyorum. Bunlar 213.221.93.193 ila 213.221.93.254 arasında olan IP adresleri. 213.221.93.193 nolu IP adresi firewall sisteminin bir ayağını oluşturuyor. Bu yüzden firewall arkasında kalan tüm serverler için 213.221.93.193 default gatewaydir.

Sunucular arasındaki yükü eşit bir şekilde dağıtabilmek için hardware loadbalancer komponentleri satın alarak sisteme dahil ettim (resimde LB1, LB2). Firewall sisteminde olduğu gibi Loadbalancer sistemi iki komponentten oluşuyor. Heart Beat mekanizmasıyla bu komponentler birbirlerine bağlı. LB1 devre dışı kalması durumda LB2 onun yerine geçerek, trafiğin akmasını sağlar.

Üye sayısı artıkça sistemin kapasitesini de artırmak gerekti. Bu yüzden yeni sunucular satın alarak bir web cluster oluşturmaya karar verdim. Cluster birden fazla sunucunun içinde bulundugu server topluluğudur. Bu sunucular aynı görevi yapmak ve oluşan yükü paylaşmak için berabar çalışırlar.



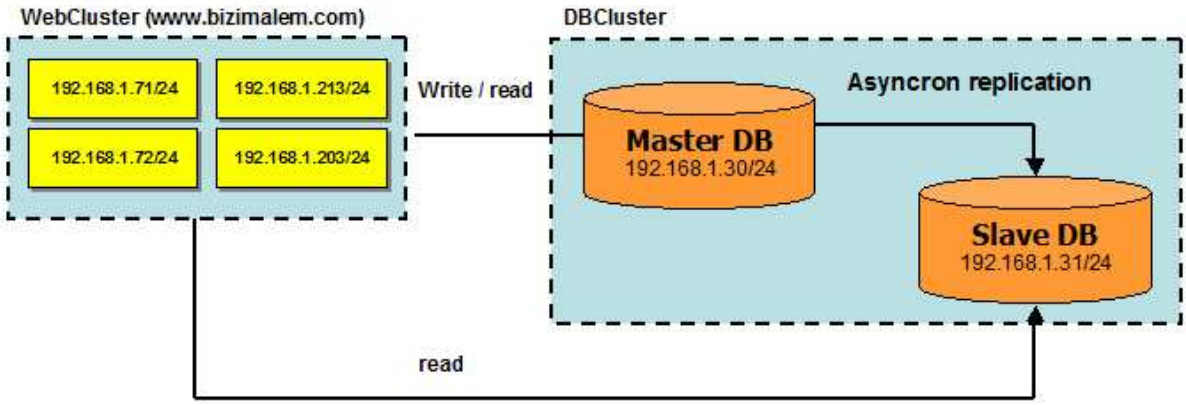

**Resim 15** BizimAlem ağ planı

web1 (192.168.1.71), web2 (192.168.1.72), web3 (192.168.1.213) ve web4 (192.168.1.203) ismini taşıyan bu sunucular web clusteri oluşturuyor. Artık eski versiyonda olduğu gibi bu sunucuların geçerli IP adresleri yok, yani 213.221.93.10 yerine iç ağdan olan 192.168.1.71 gibi bir IP adrese sahipler. Bu şekilde bu sunucuların direk internet üzerinden erişimleri engellenmiş oluyor. Bu serverlere sadece Loadbalancer (LB1) komponenti üzerinden erişilebilir. Bu da sadece http://www.BizimAlem.com, yani 213.221.93.231 IP numarası üzerinden mümkün. Loadbalancer 213.221.93.231 nolu IP ye gelen istekleri otomatik olarak web cluster içinde bulunan herhangi bir sunucuya iletir. Bu sunucunun geçerli bir IP adresi olmak zorunda değil. Yaptığım ağ ayarları ile LB1 ve web cluster aynı ağ içinde olduklarından (Switch2 üzerinden) birbirleriyle temas kurmaları kolaylaştı. LB1 ve LB2 üzerinde oluşturduğum 213.221.93.231 nolu VIP (virtuelle IP) üzerinden web cluster içinde bulunan tüm sunucuları sadece bir tek sunucuymuşcasına kullanmak mümkün hale geldi. Bu durumda kullanıcı sadece http://www.BizimAlem.com yazar ve LB1 tarafından web cluster içinde bulunan bir sunucuya yönlendirilir. Kullanıcı hangi server üzerinde olduğunu bilmez. LB1 cookieler yardımı ile kullanıcı isteklerinin hep aynı sunucuya iletilmesini mümkün kılar.

Geçen zaman diliminde internet ağ oluşumu ve yönetimi hakkında detaylı bilgi edinme firsatı buldum. Sene 2005 ocak ayını gösterirken BizimAlem 200.000 den fazla üyesiyle Avrupa'nın en büyük Türk platformu olma yolunda ilerliyordu. Online üye sayısı 2500 lere dayanmış ve yine doğal olarak sistem sahip olduğu kapasite sınırlarını zorlamaya başlamıştı. Bunun yanısıra sistem üzerinde aynı anda bir çok üye işlem yaptığı için kullandığım bilgibankası performans sorunları yaratmaya başlamıştı. Bu sorunu çok kısa bir zamanda ortadan kaldırmam gerekiyordu, çünkü BizimAlem.com, sistem yükünün çok artığı saatlerde erişilemez hale geliyordu. Bu durum BizimAlem v.3.0 versiyonunun hazırlık başlangıcı oldu.

# Üçüncü Versiyon (v.3.0)

2006 senesine gelindiğinde BizimAlem büyük bir platform haline gelmiş, lakin büyüklüğüyle oluşan problemler de büyük boyutlara ulaşmıştı. En büyük sorunlardan birisi bilgibankasının aplikasyon için dar boğaz (bottleneck) haline gelmesiydi. Bu sorunu çözebilmek için iki bilgibankasının bir cluster sistemi oluşturduğu bir yapıyı oluşturmaya başladım. BizimAlem için açık kaynaklı olan (open source) PostgreSQL bilgibankasını kullanıyorum. Birden fazla bilgibankası sunucusundan oluşan bir cluster oluşturabilmek için bilgibankaları arasında replikasyon yöntemleri kullanarak verilerin aynı seviyede ve senkron tutulması gerekiyor. Senkron veri tutulması PostgreSQL sistemlerinde hemen hemen mümkün değil. Ama verileri asenkron replikasyon yöntemleri kullanarak cluster içinde bulunan bilgibankalarında aynı seviyede tutmak mümkün. Bunun ne anlama geldiğini BizimAlem için oluşturduğum bilgibankası cluster yapısını göstererek, açıklamaya çalışayım.

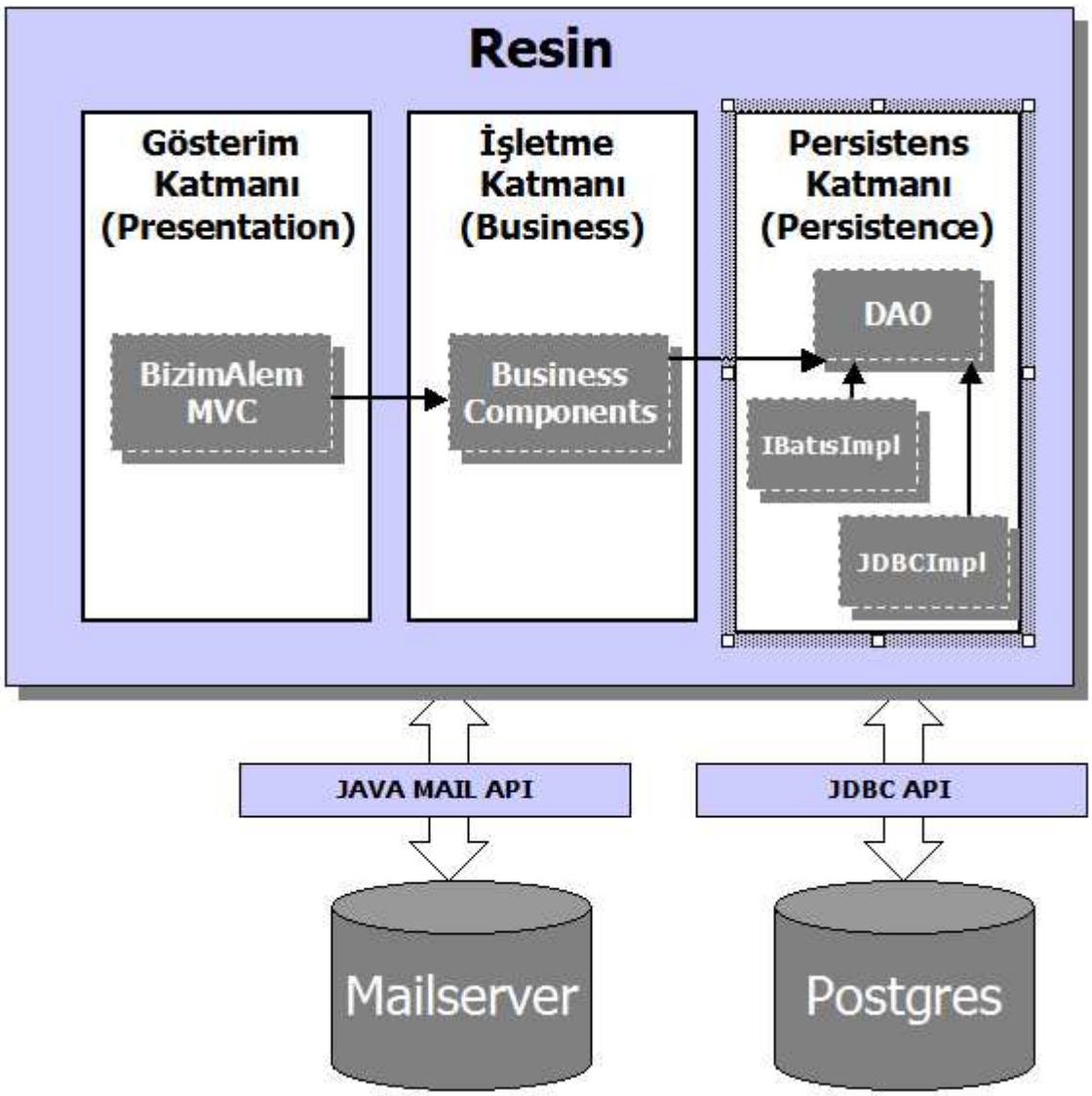



**Resim 16** BizimAlem bilgibankası cluster sistemi (dbcluster)

Web cluster tarafından kullanılan ve master (usta) db ismini taşıyan bir ana bilgibankası sunucusu var. Tüm insert,update ve delete komutları master db tarafından işlem görür, yani bütün değişiklikler bu bilgibankası üzerinde yapılır. Açık kaynaklı olan Slony[4] replikasyon sistemi ile master db bünyesinde meydana gelen tüm değişiklikler slave (çırak) db ye aktarılır. Bu işlem asenkron gerçekleşir, bu yüzden bu replikasyon tarzının ismi asenkron replikasyondur. Insert ya da update yapıldıktan sonra Slony bu değişiklikleri birkaç saniye içinde lokalize eder ve otomatik olarak bu komutları slave db ye aktarır. Bu işlemin ardından master db ve slave db aynı seviyeye gelir. Web cluster insert, update ve delete komutları için master db yi, select komutları için slave db yi kullanır. Sistemde oluşan SQL komutlarının %80 den fazlası select komutları olduğu için ikinci bir db (database) nin sistem tarafından kullanılıyor olması, bilgibankası yükünü hafifletir ve bilgibankası dar boğaz olmaktan çıkar. Böyle bir db cluster sistemi ile her iki db de select komutları için kullanılabilir, lakin insert, update ve delete komutlarının devamlı master db ye gitmesi sağlanır, çünkü master db merkezi bilgibankası sistemidir. Slave1, slave2, slave3 şeklinde yeni bilgibankası sunucuları sisteme eklenerek, bilgibankası performansı daha da artırılabilir.

BizimAlem için v.3.0 oluşturma çalışmaları 2006 Eylül ayında başladı. Amaç 500.000 den fazla üyeye hizmet verebilen ve aynı anda 10.000 üyenin online olabildiği bir platform oluşturmaktı. Bunun için BizimAlem'in yazılım mimarisini değiştirmem gerekliydi. Yaptığım performans analizleri aşağıda yer alan yazılım mimarisinin gerekli olduğunu gösterdi.

[4] Bakınız: http://www.slony.info/




**Resim 17** Üç katmanlı mimari

BizimAlem bünyesinde, üyelere sunulan hizmetler çoğaldıkça, kodun bakımı ve geliştirilmesi zor bir hal almaya başlamıştı. Bu sebepten dolayı üç katmanlı bir mimari oluşturarak, bu sorunu ortadan kaldırmaya çalıştım.

# Üç Katmanlı Mimari Nedir?

Günümüzde yapılan kurumsal projelerin temelinde üç ya da daha fazla katmanlı mimariler yatmaktadır. Program yazılımının amacı, veri oluşturmak, bu verileri depolamak, istendiği zaman depolanmış verileri elde edip, değerlendirmek ve belirli sonuçlara ulaşmaktır. Buradaki sihirli kelime „veri“ dir ve bilgisayarın ve internetin icat edilmesinde başrolü oynamıştır.

Bir firmanın günlük faaliyetlerinde hergün birçok veri oluşur, bu veriler değerlendirilir ve firmanın bir veya birden fazla bilgibankasında depolanır. Her firmanın stratejik faaliyetlerinden birisi, bu veri oluşumunun kontrollü bir şekilde yapılmasını sağlamak olmalıdır. Veri kaybı, aynı zamanda firmanın kazanç kaybı anlamına gelebileceği için, birçok

firma, sahip oldukları verilere bilgisayar ve bu bilgisayarlarda çalışan bir takım programlar aracılığı ile sahip çıkmaktadırlar.

Program yazılım amacının veriler üzerinde işlem yapmak olduğunun altını çizdik. Peki program yazma kriterleri nelerdir? Her programcı istediği şekilde, gelen istekler doğrultusunda program yazabilir mi? Evet yazabilir, ama programcının profesyonelliği, oluşan kodun kalitesi ile direk orantılıdır. Edindiğim tecrübeler doğrultusunda, bir programın bakımı ve geliştirilmesi, yazılımından daha pahalıdır diyebilirim. Tasarım şablonlarının kullanılmadığı ve profesyonel olmayan yazılımcılar tarafından oluşturulan programların bakımı imkansız ya da çok zordur. Bir firma için sahip olduğu veriler ne kadar önemli ise, verileri işlemek için kullandığı programlar ve bu programların bakımı ve geliştirilmesi de bir o kadar önemlidir.

Günümüzde firmalar sahip oldukları verileri kullanmak, saklamak ve işlemek için web tabanlı programlar kullanmaktadırlar. Web tabanlı programların bakımı ve geliştirilmesi masaüstü programlardan daha kolay ve ulaşılan kullanıcı kitlesi daha büyük olduğu icin (kullanıcılar genelde bir web tarayıcı – browser ile çalışabilirler) tercih edilmektedirler. Web tabanlı programlar bugünkü standartlara göre üç katmandan oluşacak şekilde hazırlanır. MVC (Model – View – Controller ) tasarım şablonunda da gördüğümüz gibi, sorumluluk alanları tanımlanmış katmanlar oluşturulur. Her katman kendisi için tanımlanmış görevi yerine getirmekle yükümlüdür ve işlevini yerine getirmek için diğer katmanlardan faydalanır.

Web projelerde uygulanan ilk katman gösterim (presentation) katmanıdır. Bu katmanda veriler üzerinde işlem yapılmaz. Veriler üzerinde işlem diğer katmanlar tarafından gerçekleştirilir. Gösterim katmanı başka bir katmanda hazırlanmış olan verilerin kullanıcıya gösterimi için kullanılır. Bu katmanda JSP ve Servlet gibi gösterim teknolojileri kullanılarak edinilen veriler sunulur.

Gösterim katmanına veriler işletme katmanı (business) tarafından sağlanır. İşletme katmanında veriler üzerinde yapılacak işlemler tanımlanır. Bunlar Java sınıflarında oluşturulan metodlardır. Business metodları olarak bilinen bu birimlerde, firmanın veriler üzerinde yapmak istediği işlemler implemente edilerek, istenilen neticeler elde edilir. Gösterim katmanı için gerekli veriler veri depolama / edinme (persistence) katmanı tarafından sağlanır. Bu katmanın görevi JDBC yada Hibernate gibi teknoloji ile bilgibankasında yer alan verileri edinmek ve istenilen verileri bilgibankasında depolamaktır.

Katmanlar arası komunikasyon tanımlanmış interface sınıflar üzerinden gerçekleşir. Örneğin gösterim katmanı işletme katmanında bulunan bir Facade interface üzerinden istediği verileri elde edebilir. Gösterim katmanı, işletme katmanı sadece bir interface sınıfından oluşuyormuş gibi düşünülerek, bu interface sınıfına karşı programlandığı taktirde, iki katman arasında esnek bir bağ oluşur. Bu durumda işletme katmanı, dış dünyaya sunduğu Facade interface sınıfını istekleri doğrultusunda implemente ederek gösterim katmanını etkilemeden çalışma tarzını tanımlayabilir. Facade interface sınıfında tanımlanmış metodlar değişmediği sürece, işletme katmanında yapılacak değişiklikler gösterim katmanını etkilemez. Sadece bu şekilde hazırlanmış bir program, gelecekte meydana gelen değişikliklere ayak uydurabilir yapıda olabilir. Tüm kodun bir katman içinde implemente edilmesi, kullanılan sınıflar arasındaki bağı yükselteceği gibi, bu kodun ilerideki bakımını güçleştirir.

# BizimAlem Persistence Katmanı

v.3.0 öncesi bilgibankası işlemlerini yapmak için sadece JDBC teknolojisini kullanıyordum. JDBC ile performans ve tuning işlemleri çok daha kolay bir hale geliyor. Ana derdiniz de performans olunca, o zaman doğal olarak JDBC teknolojisinde kalıyorsunuz, çünkü JDBC ile bilgibankası işlemlerine müdahale etmek çok daha kolay. Bu yüzden fırsat bulup, nesneleri bilgibankasında kalıcı hale getiren bir ORM (object relational mapping) aracı kullanmadım ve buna da açıkça gerek yoktu. Zamanla BizimAlem için geliştirdiğim yeni hizmetler ile tasarım nesnelere doğru kaymaya başlayınca, oluşan bu nesneleri JDBC ile bilgibankasına aktarmak çok kompleks bir hal almaya başladı.

Bugün kullanılmakta olan bir çok ORM aracı bulunmaktadır. Bunların başında Hibernate, Toplink, Cocobase ve IBatis gelir. Saf kan ORM aracı olan Hibernate ile nesneler birebir bilgibankası tablolarına yerleştirilir. Programcının oluşturulan JDBC koduna müdahale etme şansı pek yoktur. Tüm persistens işlemi Hibernate tarafından gerçekleştirilir. Bu doğal olarak performansın önemli olduğu alanlarda bir dezavantaj, çünkü SQL komutlarına direk müdahale etmek hemen hemen imkansız. Bu sebepten dolayı Hibernate gibi bir ORM aracını BizimAlem'de hiçbir zaman kullanma taraftarı olmadım. Benim aradığım ORM ile JDBC arasında olan bir araçtı. Hem nesneleri bilgibankasına kaydedebilmeliydim, hem de SQL komutları üzerinde değişiklik yaparak, nesnelerin bilgibankasına yerleştirilmesi ve tekrar edinilmesi işlemine müdahale edebilmeliydim. Bunu yapabilen IBatis isminde bir ORM aracı var.

IBatis herhangi bir sınıf için, o sınıfın nesneleri üzerinde işlem yapmak üzere sqlMap ismini taşıyan XML dosyalarında gerekli SQL komutların oluşturulmasını sağlamaktadır. Bir sonraki örnekte görüldüğü gibi resultClass olarak tanımlanan PortalVo sınıfından bir nesne oluşturmak için getPortalList isimli select kullanılmaktadır. Nesneyi oluşturmak için gerekli select komutu <select> tagı bünyesinde yer almaktadır. IBatis tanımlanan bu select komutunu kullanarak, PortalVo sınıfından bir nesne oluşturarak, aplikasyona bu nesneyi verir. Bu şekilde hem nesneler ile kullanım kolaylaştırılmaktadır, hem de kullanılan SQL komutlarına müdahale etmek mümkün olmaktadır.

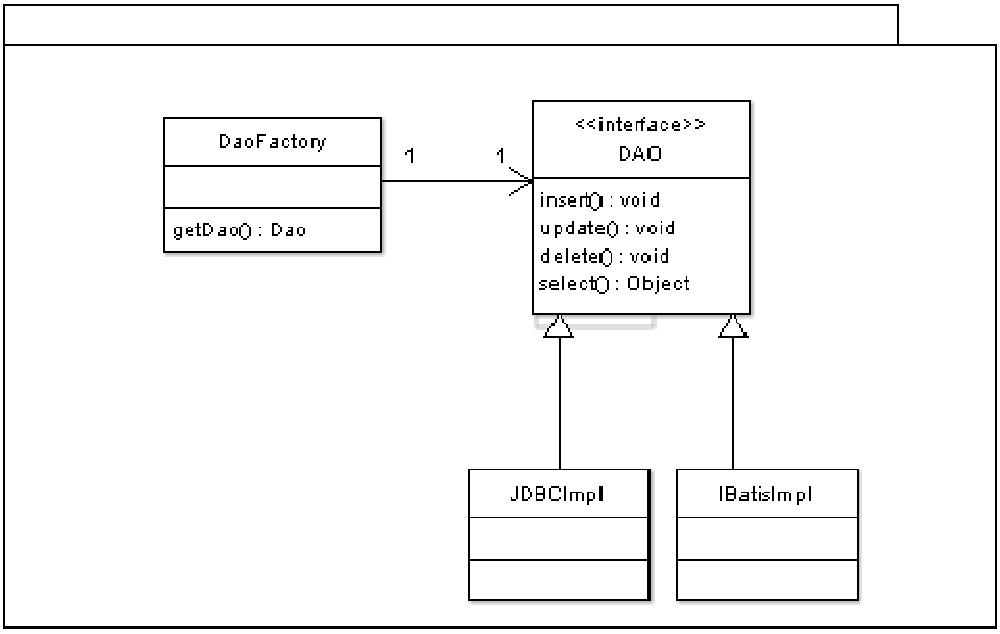
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="Poll">

      <select id="getPortalList"
resultClass="smart.core.cache.vo.portal.PortalVo">
            select
            id,username,subject,created,commentcounter from
community_portal order
            by created2 desc limit 6 offset #offset#
</select>

      <select id="getPortalById"
resultClass="smart.core.cache.vo.portal.PortalVo">
            select * from
            community_portal where id = #id#
</select>

</sqlMap>
```

Mevcut JDBC tabanlı kod yanı sıra IBatis'i kullanabilmek için DAO[5] tasarım şablonunu kullanmaya karar verdim.



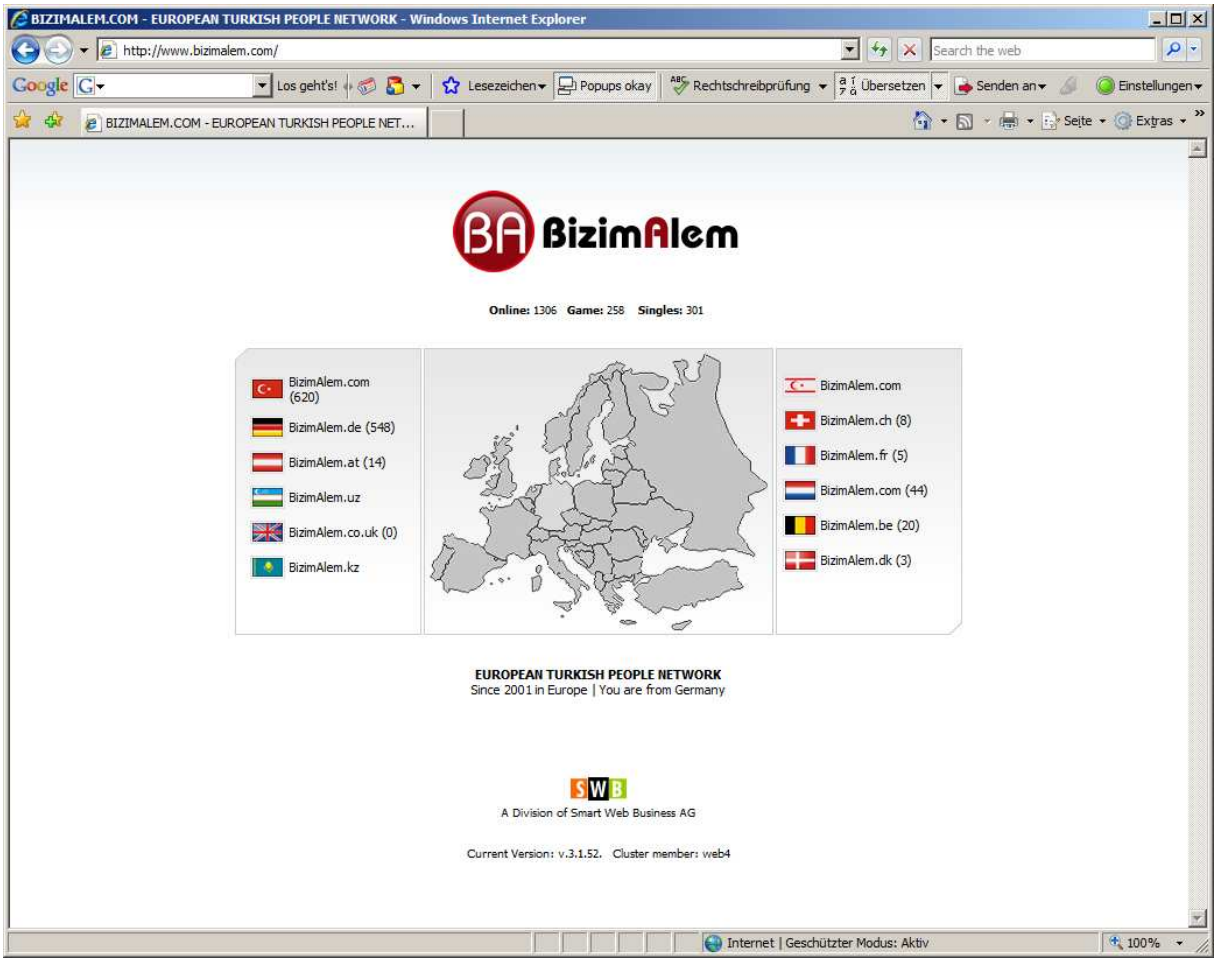

**Resim 18** DAO tasarım şablonu

DAO bir interface sınıf ve içinde insert, update, delete, select gibi metotlar yer almaktadır. İki değişik teknolojiyi, yani JDBC ve IBatis aynı anda kullanabilmek için DAO interface sınıfının iki değişik teknoloji için implemente edilmesi gerekir. Bu sebepten dolayı tamamen JDBC koduna dayalı olan JDBCImpl ve IBatis ORM aracını kullanan IBatisImpl implementasyonlarını oluşturdum. DAOFactory üzerinden yerine göre JDBCImpl ya da IBatisImpl implementasyonlarını kullanmam kolay bir hale geldi. Yeni oluşturduğum BizimAlem servisleri için sadece IBatisImpl implementasyonunu kullanmaya başladım ve bu şekilde bu servislerin tamamen nesneler ile oluşturulması kolay bir hal aldı.

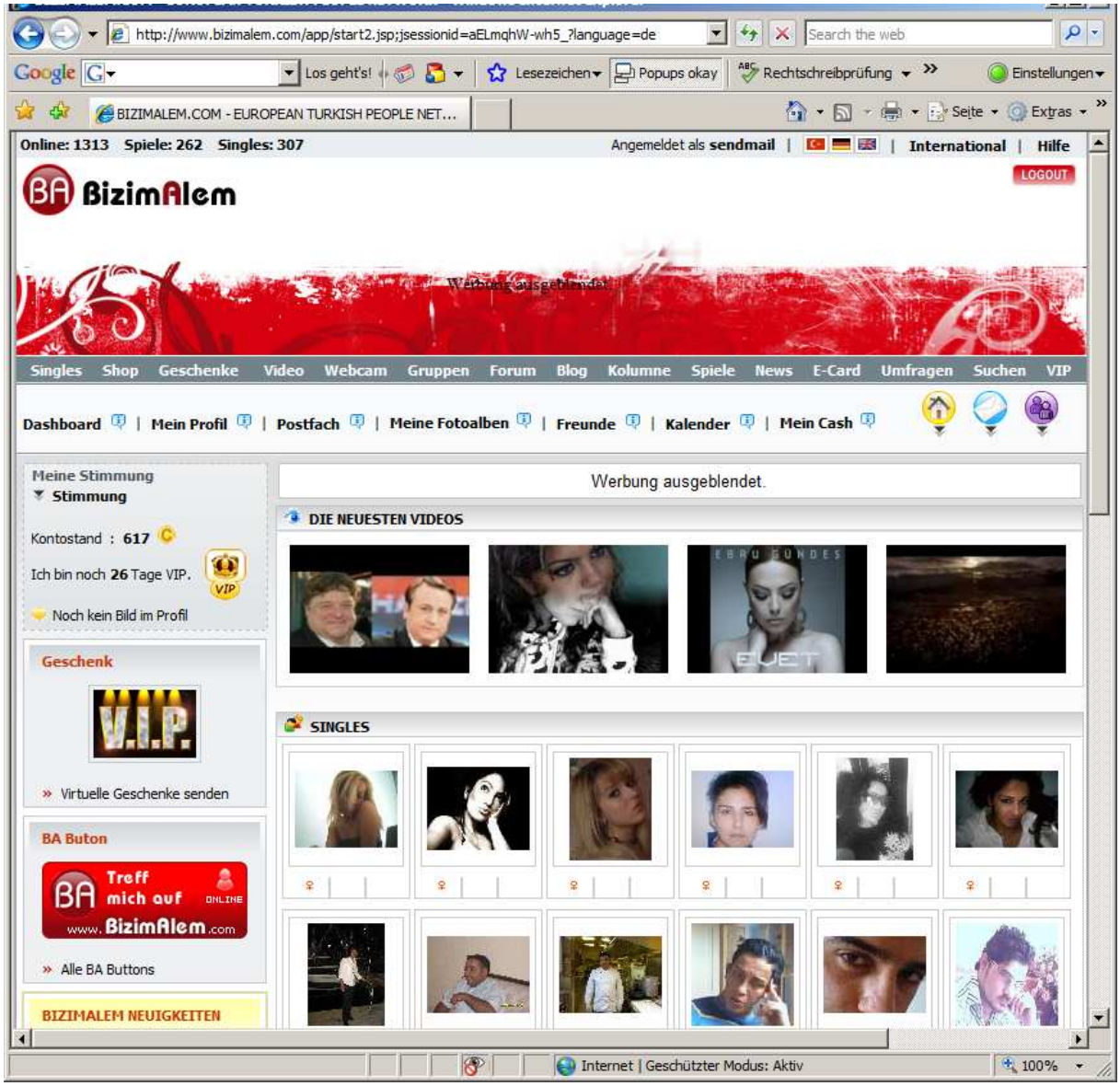
# Yeni Tasarım

v.3.0 ile tasarım ve kullandığım BizimAlem logosu da değişti. Yeni tasarım resim 19,20 ve 21 de yer almaktadır.

[5] Bakınız: http://www.kurumsaljava.com/2008/12/01/data-access-object-dao-tasarim-sablonu/




**Resim 19** BizimAlem v.3.1.52 versiyonu giriş sayfası




**Resim 20** BizimAlem v.3.1.52 versiyonu ana sayfa


**Resim 21** BizimAlem v.3.1.52 versiyonu oyun modülü

24.6.2007 tarihinde v.3.0 online oldu. Yeni versiyon online olmadan önce bilgibankasında geniş çaplı bir temizlik yaptım. İlk etapta 6 aydır kullanılmayan tüm kullanıcı hesaplarını sildim. Bu işlemin ardından aktüel kullanıcı sayısı 300.000 in altına düştü. Bu iyi bir rakamdı, çünkü bu üyelerin hemen hemen hepsi platformu aktif olarak kullanıyorlardı. v.3.0 online olduktan sonra tüm üyelere email aracılığıyla duyuru yaparak, yeni versiyonu tanıttım. Bu şekilde sistemi uzun süredir kullanmayanların da tekrar dikkatini çekme fırsatı buldum.

v.3.0 kullanıma açıldıktan sonra doğal olarak bazı sistem hataları ortaya çıktı. Bu hataları gidermeye çalışırken kullanıcıların sistem üzerindeki faaliyetleri artığından dolayı oluşturduğum bilgibankası cluster sistemi tekrardan dar boğaz haline gelmeye başladı. Yeni oluşturduğum BizimAlem servisleri ayrıca sisteme ek yük getirdiklerinden dolayı, bilgibankasının dar boğaz olması süreci hızlandırılmış oldu. Bu sorunu çözmek için artık elimde sadece bir silah kalmıştı: Caching!

# Caching Nedir?

Ön belleğe alma olarak tercüme edebileceğimiz caching mekanizmaları ile kullanılmak istenen veri, verinin kaynağına ulaşmak zorunda kalmadan, bilgisayar hafızasında (ram) tutulur. Bilgisayar hafızasına erişim, verinin bulunduğu kaynağa (örneğin harddisk) erişimden daha hızlı olduğu için, verinin işlenme süresi kısaltılmış olur. Ayrıca verinin kaynağına erişim veri transferi açısından bir dar boğaz olabileceği için, caching mekanizmaları ile dar boğazlılık sorunu giderilir.

Caching mekanizmaları birçok alanda kullanılmaktadır. Bunlardan bazıları şöyledir:

- Ana işletim birimi (cpu) hafıza köprüsü (memory bus) üzerinden gerekli verileri elde ettikten sonra bu verileri kendi bünyesinde barındırdığı cache alanlarında saklar. Bu işletim biriminin çalışma hızını artırır.
- Harddiskler cache mekanizmaları kullanarak, diskler üzerinde bulunan verileri tekrar edinmek zorunda kalmadan kullanıcı sisteme aktarabilirler. Bu harddisklerin performansını artırır.
- Veriler bilgibankasından okunduktan sonra caching mekanizmaları kullanılarak hafızada tutulur. Bu verileri kullanan programın, sıkça bilgibankasını kullanmak zorunda olmadığı için performansını artırır.

Cache içinde tutulan veri şüphesiz orijinal verinin bir kopyasıdır. Er ya da geç veri asıl bulunduğu kaynak üzerinde (örneğin bilgibankası) değişikliğe uğrayacaktır. Bu durumda cache içinde bulunan kopyanın statüsü nedir?

Eskiye rağbet olsaydı, bit pazarına nur yağardı demiş atalarımız ☺ Kopya hiç bir zaman orijinalın yerini alamayacağı için kıymet taşımaz. Lakin bu bilgisayar sistemlerinde işlenen veriler için geçerli değil. Bir kopya veri, orijinalı değişmediği sürece onunla aynı değerdedir. Ne zaman orijinal veri değişikliğe uğradı, o zaman kopya için ölüm çanları çalmaya başlar. Orijinal veri değişikliğe uğradıktan sonra, cache içinde bulunan kopyanında değişikliğe uğraması gerekmektedir, aksi taktirde cache içinde bulunan veriyi kullanan program verinin en son haline sahip olmadığı için yanlış işlem yapabilir hale gelecektir. Buradan söyle bir sonucu çıkartabiliriz: "**Bir cache içinde tutulan verinin belli bir zaman diliminden sonra kendisini imha ederek, verinin tekrar asıl kaynağından edinilmesini sağlaması gerekmektedir**".

Cache içine atılan bir veri için geçerlilik süresi tespit edilmelidir. Aksi taktirde veri sonsuza dek (bilgisayar restart edilene kadar) cache içinde kalacak ve verinin tekrar asıl kaynağından edinilmesini engelleyecektir. Örneğin bir müşterinin bilgilerini ihtiva eden bir nesne cache içine atılmadan önce 30 saniye geçerli olacak şekilde yapılandırılabilir. Her 30 saniyenin bitiminde müşteri bilgileri bilgibankasından tekrar edinilerek, cachelenir. Buradan şöyle bir sonucu çıkartabiliriz: "**Caching mekanizmasının sağlıklı çalışabilmesi ve kullanıcı programı yanıltmaması için verilerin geçerlilik süresine (expiration) sahip olmasına gerekmektedir**". Geçerlilik süresi nesneyi cache sistemine atan program tarafından belirlenen bir özelliktir ve bir zaman birimini (saniye, dakika, saat) ihtiva eder. Örneğin sıkça değişikliğe uğramayan veriler için 1 gün, 1 ay gibi geçerlilik süresi tayin edilebilir. Sıkça değişikliğe uğrayan veriler için bu süre 30 saniyenin altında olacaktır.

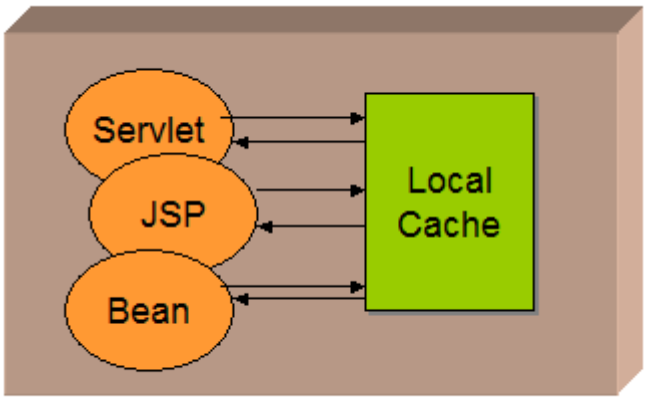
Bir veriyi caching sistemine atmamız ve geçerlilik süresini belirlememiz yeterli değildir. Veriyi tekrar edinebilmek için bu veriyi **adresleyebilmemiz** gerekmektedir. Bunu bir anahtar (key) kullanarak yapabiliriz. Bu anahtarın tüm sistem içerisinde eşsiz olması gerekmektedir. Sadece bu sayede istenilen veriye ulaşılabilir ve aynı anahtarı taşıyan iki verinin birbirlerini yok etmeleri engellenir.

# Caching Türleri

Caching sistemleri **lokal** ve **global** olmak üzere iki gruba ayrılır.

## Lokal Caching Sistemleri

Bu tür caching sistemlerinde caching sistemi kullanıcı program ile aynı hafıza alanında (in memory) faaliyet gösterir. Kullanıcı program ile aynı hafıza alanı ve JVM içinde bulunan caching sistemi, kullanıcı programa yakınlığından dolayı lokal caching sistemi olarak isimlendirilir.




**Resim 22** Lokal caching sistemi

Lokal caching sistemlerinin performansı yüksektir, çünkü caching işlemleri kullanıcı program ile aynı hafıza alanını paylaşan lokal cache üzerinde hızlı bir şekilde gerçekleştirilir.

Lokal cache sistemlerinde sadece bir tane cache kopyası (instance) mevcuttur. Her aplikasyon için bir lokal cache kopyasının oluşturulması gerekmektedir. İki değişik adres alanında faaliyet gösteren aplikasyon aynı lokal cache kopyasını kullanamazlar.

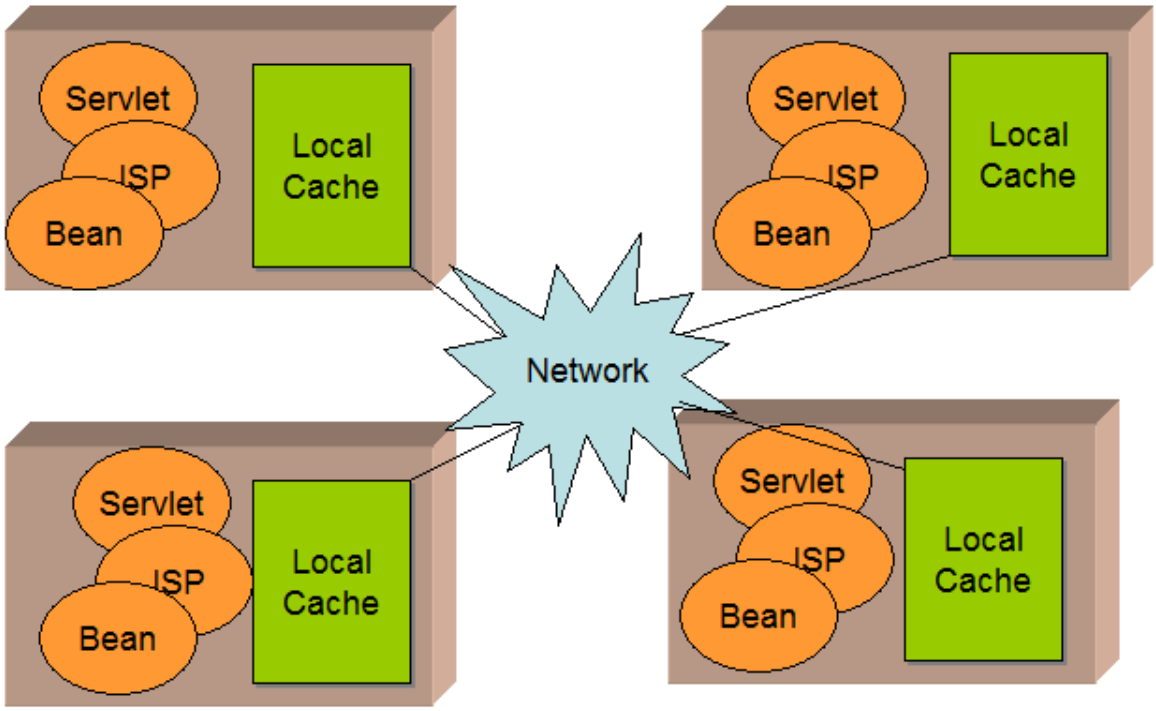
## Global Caching Sistemleri

Birden fazla server üzerine dağıtılmış (distibuted) ama tek hafıza alanıymış gibi faaliyet gösteren caching sistemlerine global caching sistemi adı verilir. Global caching sistemleri kendi aralarında iki gruba ayrılır:

- Global tek hafıza caching sistemleri
- Global bölümsel (partial) hafıza caching sistemleri

## Global Tek Hafıza Caching Sistemleri

Bu tür caching sistemlerinde her aplikasyon serveri kendi hafıza alanında bir lokal caching sistemine sahiptir. Birden fazla aplikasyon serveri ve lokal caching sisteminin kullanılması bir global cache oluşturulduğu anlamına gelmez. Global bir caching sisteminin oluşabilmesi için ağ içindeki lokal caching sistemlerinin birbirlerinden haberdar olmaları gerekmektedir. Kullanılan caching sistemleri ağ üzerinden birbirleriyle bağlantı kurarak, bünyelerinde meydana gelen değişiklikleri ağ içindeki diğer lokal cachelere bildirirler. Bu işlem için JGroups ya da JMS gibi komünikasyon teknolojileri kullanılır. Bu sayede bir global caching sistemi oluşur.



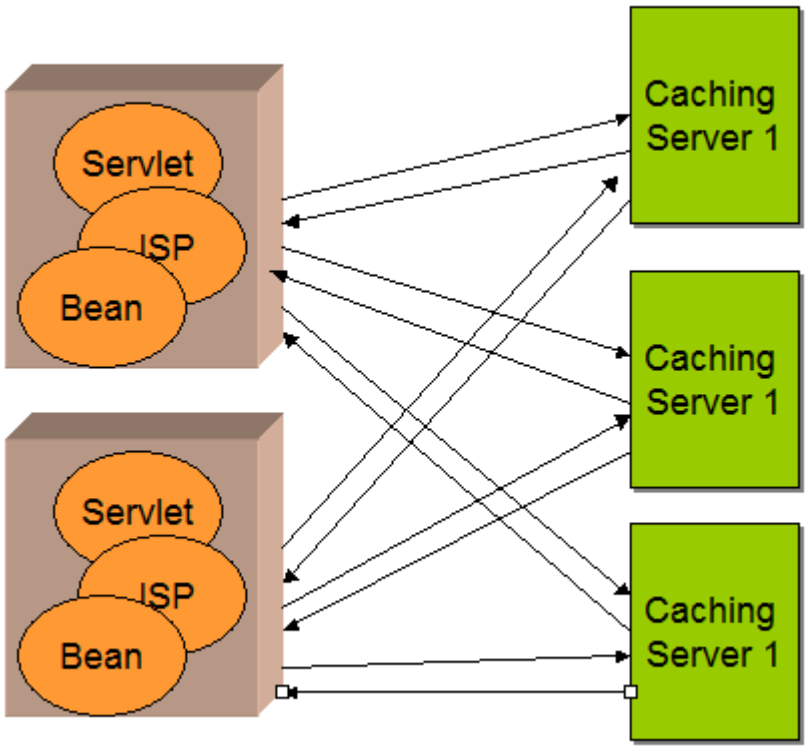

**Resim 23** Global caching sistemi

Lokal cacheler arası verinin replikasyonu (kopyalanması) ile tüm lokal cacheler aynı veriye sahip olur.

Bu tarz global bir caching sistemi oluşturabilmek için verinin lokal cache sistemleri arası replike (kopyalanması) edilmesi gerekmektedir.

## Global Bölümsel (Partial) Hafıza Caching Sistemleri

Bu tür global caching sistemlerinde caching komponentleri arasında replikasyon yapılmaz.




**Resim 24** Global bölümsel caching sistemi

Global caching sistemini kullanmak isteyen her aplikasyon, sistemdeki tüm caching serverleri ile bağlantı kurarak, caching işlemlerini gerçekleştirir. Kullanılan özel bir caching API yardımı ile veriler değişik caching serverlerine dağıtılır. Bu yüzden bu tür faaliyet gösteren caching sistemlerine global bölümsel caching sistemleri adı verilmektedir, çünkü veriler değişik caching serverlerine dağıtılarak cachelenir. Daha sonra yakından inceleyeceğimiz MemCached bu tarz caching sistemleri kurmak için kullanılabilecek bir caching sistemidir.

# Caching Konseptleri

Bu yazıda ismi geçen caching konseptlerini yakından inceleyelim:

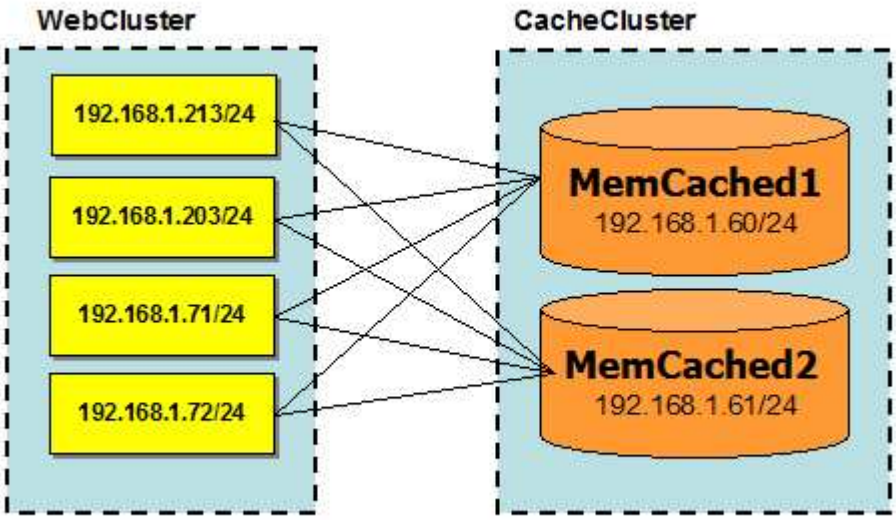
- Ön belleğe anılacak olan verinin kendisi (**cacheable object**) . Örneğin bir sınıftan oluşturulan bir nesne veri olarak ön belleğe alınabilir.
- Verinin ön bellekteki kalma süresi (**expire time**). Bu değer baz alınarak, verinin ön bellekte hangi zaman dilimi için kalacağı belirlenir. Verinin ön bellek içinde sonsuza dek kalmasını engellemek için limitli bir zaman diliminin seçilmesi gerekmektedir, örneğin 30 saniye.
- Veriyi ön belleğe yerleştirmek ve tekrar edinmek için kullanılan anahtar (**key**). Bu anahtarın sistem bünyesinde eşsiz (unique) olması gerekmektedir. Aksi

taktirde aynı anahtara sahip verilerin birbirlerini yok etmeleri ihtimali oluşmaktadır.

- Verileri ön belleğe almak için put olarak isimlendirilen işlem gerçekleştirilir. put işleminde ön belleğe alınacak verinin kendisi, verinin ön bellekteki kalış süresi ve veriyi adresleyen anahtar (key) parametre olarak kullanılır.
- Verileri ön bellekten edinmek için get işlemi gerçekleştirilir. get işleminde veriye ulaşmak için put işleminde oluşturulan anahtar (key) kullanılır.

# BizimAlem Caching Sistemi

BizimAlem için global bölümsel hafıza caching sistemi olan MemCached serverini kullanmaya karar verdim. MemCached [6] client-server tarzı çalışan bir global bölümsel caching sistemidir. BizimAlem için oluşturduğum caching sistemi resim 25 de yer almaktadır.




**Resim 25** BizimAlem Caching sistemi

Web cluster içinde bulunan her sunucu cache cluster içinde bulunan her memcached serveri ile bağlantı kurarak, caching işlemlerini gerçekleştirir.

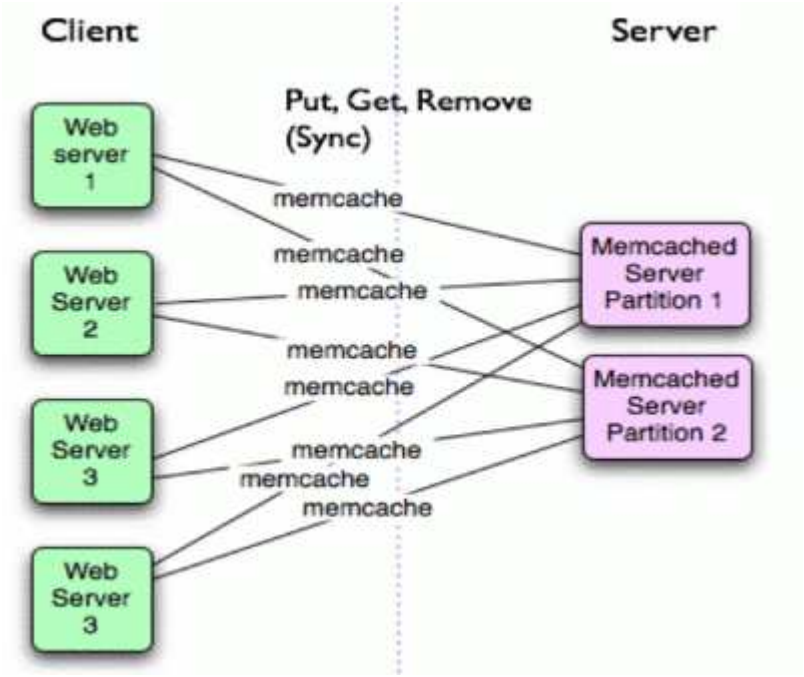
Fiziksel bir server üzerinden MemCached kurulduktan sonra, aşağıdaki şekilde çalıştırılır.

```
./memcached -d -m 2048 -l 192.168.1.10 -p 11211
```

Yukarda yer alan örnekte MemCached 192.168.1.10 IP numaralı server üzerinde 2 GB hafıza alanını kullanacak ve 11211 numaralı porttan erişilebilir olacak şekilde çalıştırılmıştır. Birden fazla MemCached ağ içinde aktif olabilir. MemCached serverler arasında veriler replikasyon (kopyalama işlemi) usulüyle senkron tutulmaz. MemCached serverlerine verileri transfer eden Client API (daha sonra detaylı olarak göreceğiz) cachelenmesi gereken verinin anahtarı yardımıyla eşsiz (unique) bir hash değeri oluşturarak, listeden bir MemCached

[6] Bakınız: http://www.danga.com/memcached/

serverini seçer ve veriyi direk bu MemCached serverine transfer eder. Veriyi edinmek için tekrar aynı anahtar kullanıldığında Client API yine aynı hash değerini oluşturarak, verinin hangi MemCached server üzerinde olduğunu belirler ve veriyi temin eder.




Caching sistemini kullanabilmek için Java dilinde implemente edilmiş bir MemCached Client API implementasyonuna ihtiyacımız var.

# MemCached Client API

MemCached serveri ile bağlantı kurup, işlem yapabilmek için değişik dillerde Client API'ler[7] oluşturulmuştur. Java dilinde Greg Whalin[8] tarafından implemente edilen MemCached Client API'yi kullanıyoruz.

```
package smart.core.cache;

import java.util.Date;
import java.util.Map;
import com.danga.MemCached.MemCachedClient;
import com.danga.MemCached.SockIOPool;


public class MemcachedImpl extends BaseCache
{

      protected static MemCachedClient client =
            new MemCachedClient(Thread.currentThread().
```

[7] Bakınız: http://www.danga.com/memcached/apis.bml
[8] Bakınız: http://whalin.com/

```
                        getContextClassLoader());

    static
    {
      // get mcache servers
      String[] serverlist =
          getProperty( "memcached.servers" ).split( "," );

      // get server weights
      Integer[] weights      =
          new Integer[serverlist.length];

      String[] serverWeights =
          getProperty("memcached.servers.weights" ).split( "," );

      for ( int i = 0; i < serverWeights.length; i++ )
      {
          weights[i] = new Integer( serverWeights[i] );
      }


      // idle time set to 4 hours
      long maxIdle = 1000 * 60 * 60 * 6;

      // socket read timeout
      int readTO = 1000 * 3;

      // initialize the pool for memcache servers
      // see javadocs on this method
      // for tuning the connection pool
      SockIOPool pool = SockIOPool.getInstance();
      pool.setServers( serverlist );
      pool.setWeights( weights );
      pool.setInitConn( Integer.parseInt(getProperty("initcon")) );
      pool.setMinConn(  Integer.parseInt(getProperty("mincon")) );
      pool.setMaxConn(  Integer.parseInt(getProperty("maxcon")) );
      pool.setMaintSleep(  Integer.parseInt(getProperty("maintsleep")) );
      pool.setNagle( Boolean.getBoolean(getProperty("nagle")) );
      pool.setSocketTO( readTO );
      //pool.setSocketConnectTO( 0 );
      pool.setMaxIdle( maxIdle );
      pool.setFailover(Boolean.getBoolean(getProperty("failover")));
      pool.setFailback(Boolean.getBoolean(getProperty("failback")));
      pool.initialize();
      getClient().setCompressEnable(
                Boolean.getBoolean(getProperty("compression")) );
  }



    public static MemCachedClient getClient()
    {
          return client;
    }



    public void set(String key, Object obj)
    {
          getClient().set(key, obj);
    }
```

```
	public void set(String key, Object obj, Date date)
	{
		getClient().set(key, obj, date);
	}


	public Object get(String key)
	{
		return getClient().get(key);
	}


	public Map getStats()
	{
		return getClient().stats();
	}

	public void delete(String key)
	{
		delete(key,null);
	}

	public void delete(String key, Date expiry)
	{
		getClient().delete(key,expiry);
	}



	public Map getStats(String[] server)
	{
		return getClient().stats(server);
	}

}
```

Yukarda yer alan MemcachedImpl sınıfı ile MemCached serverlerini kullanmaya başlayabiliriz. Sistem için gerekli ayarları (örneğin hangi MemCached serverlerin kullanıldığı) memcached.properties dosyasında tutuyoruz. Bu dosyanın içeriği aşağıda yer almaktadır.

```
#PROD

# memcached.server=server1:port,server2:port....
memcached.servers=192.168.2.198:11211,192.168.2.199:11211,192.168.2.211:11211
memcached.servers.weights=1,2,1
memcached.activated=true


initcon                 = 100
mincon                  = 100
maxcon                  = 512
maintsleep              = 1000
nagle                   = false
alivecheck              = true
failover                = false
failback                = false
```

```
compression             = false

domain                  = prod

impl                    = smart.core.cache.MemcachedImpl
```

***memcached.servers*** anahtarı, kullanılan MemCached serverlerin IP adresleri ve port numaralarını ihtiva etmektedir. Kullanılan MemCached serverler virgül ile birbirlerinden ayrılmıştır. Buna göre 192.168.2.198 - 192.168.2.199 - 192.168.2.211 IP numaralı serverler MemCached serverleridir ve 11211 numaralı port üzerinden bu servise ulaşılabilir.

Kullanılan bazı diğer parametreler:

- **initcon = 100** – Client API her MemCached server için başlangıçta 100 adet TCP bağlantısı kurar. Böylece bir bağlantı (connection) havuzu (pool) oluşturularak, caching işlemleri hızlandırılmış olur.
- **mincon = 100** – Bağlantı havuzunda en az 100 bağlantı olacak şekilde ayarlamalar yapılır.
- **maxcon = 512** – Bağlantı havuzunda en fazla 100 bağlantı olacak şekilde ayarlamalar yapılır. Client API otomatik olarak bağlantı havuzunun hacmini ayarlar.

Java için kullandığımız Client API implementasyonunda MemCached serverlere olan bağlantı devamlı kontrol edilir. Eğer MemCached serverlerinden birisi görevini yerine getirmiyorsa, otomatik olarak mevcut server listesinden alınır. Bu şekilde çalışmayan komponentlerin devre dışı kalması sağlanır.

Bir sonraki kod bölümünde MemCached caching serverlerinin BizimAlem kodu bünyesinde nasıl kullanıldığı yer almaktadır.

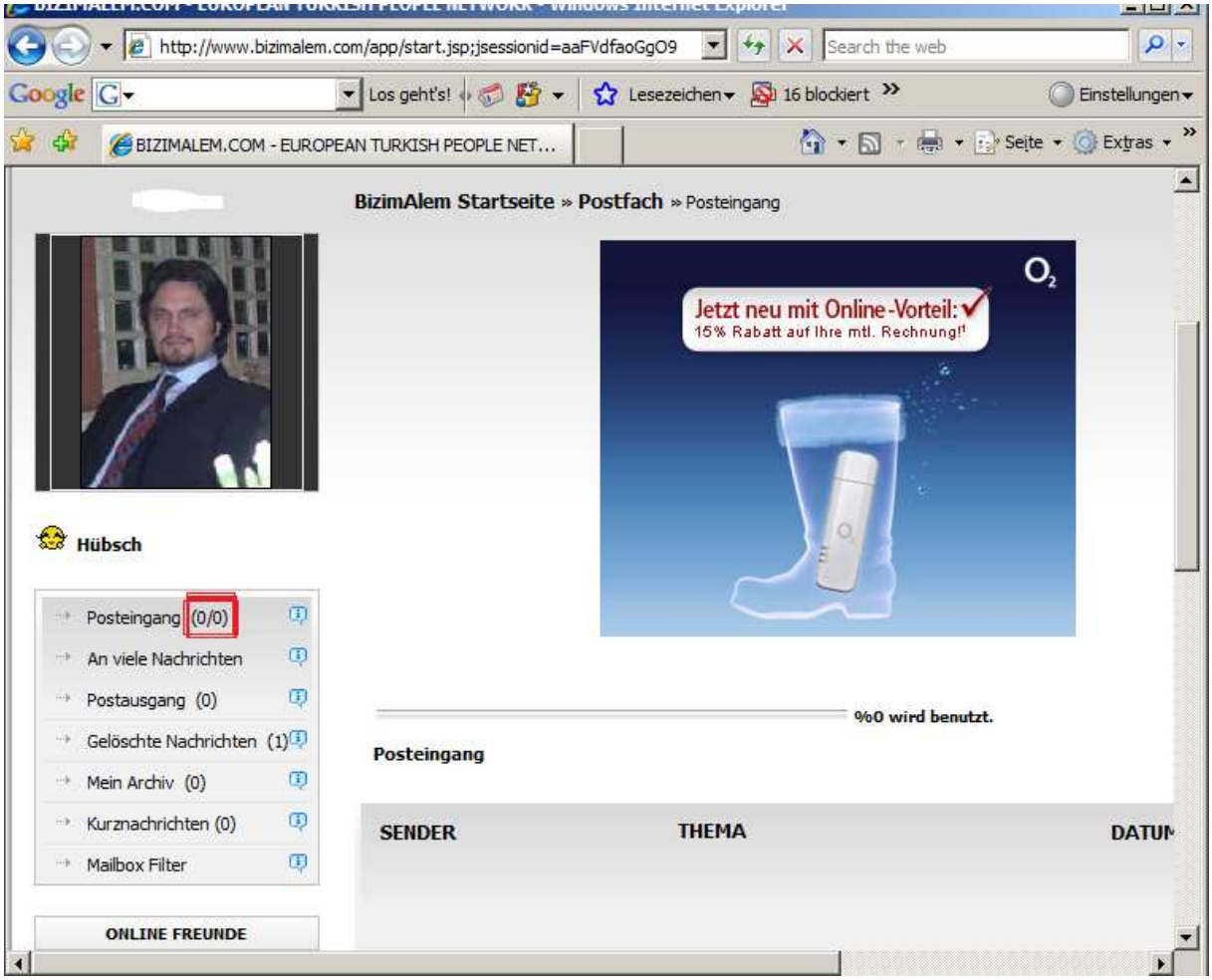
```
public String getIncomingMailCounter() throws Exception
{
      logger.debug("getIncomingMailCounter()");

      String key = KeyAttributes.getKey(
                  KeyAttributes.KEY_MAILBOX_COUNTER , getUsername());
      String vo =null;

      try
      {

            vo = (String)CacheManager.instance().get(key);

            if(vo == null)
            {
                  vo = getIncomingMailCounter();
                        CacheManager.instance().set(key, vo,
                                    Expire.EXPIRE_IN_MINUTE_5);
            }
      }
      catch(Exception e)
      {
            reportError(e);
      }
      return vo;
}
```



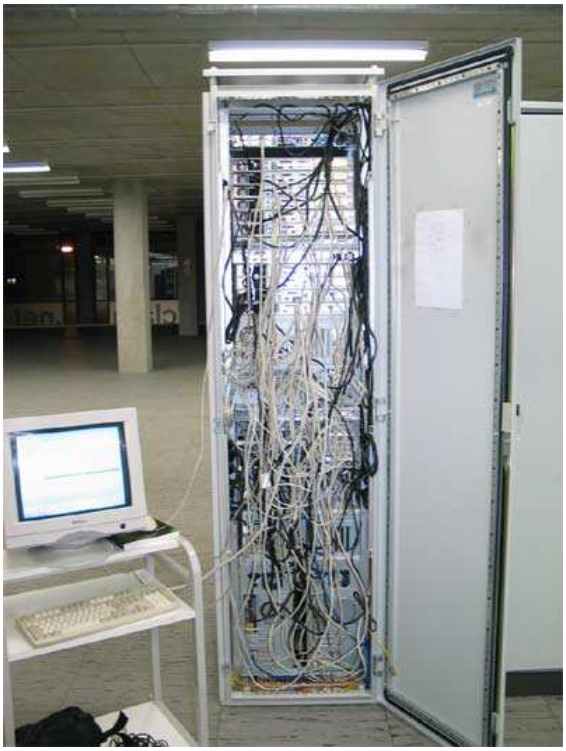

**Resim 26** Üye posta kutusu

***getIncomingMailCounter()*** metodu ile bir önceki resimde görüldüğü gibi posta kutusunda olan emaillerin adedi hesaplanmaktadır. Metot içinde vo isminde bir String tanımlanmaktadır. vo ismindeki değişken posta kutusundaki mektup adedini ihtiva eder. Bu veriyi bilgibankasından edinmeden önce, ClientManager sınıfı yardımıyla caching sisteminde arıyoruz. Bu işlem için bir anahtar oluşturmamız gerekiyor. ***KeyAttributes.getKey()*** metodu yardımıyla kullanıcıya has anahtar oluşturulmaktadır. Login yapmış bir üyeye ait bilgileri caching sisteminde tutabilmek için, anahtarın içinde üyenin ismi yer almaktadır. Üyenin isminin **test1** seklinde olacağını düşünürsek, oluşturulan anahtar **mailcounter:test1 ş**eklinde olacaktır. Bu şekilde değişik isimdeki üyeler için üyeye özel verileri caching sisteminde tutmak mümkündür.

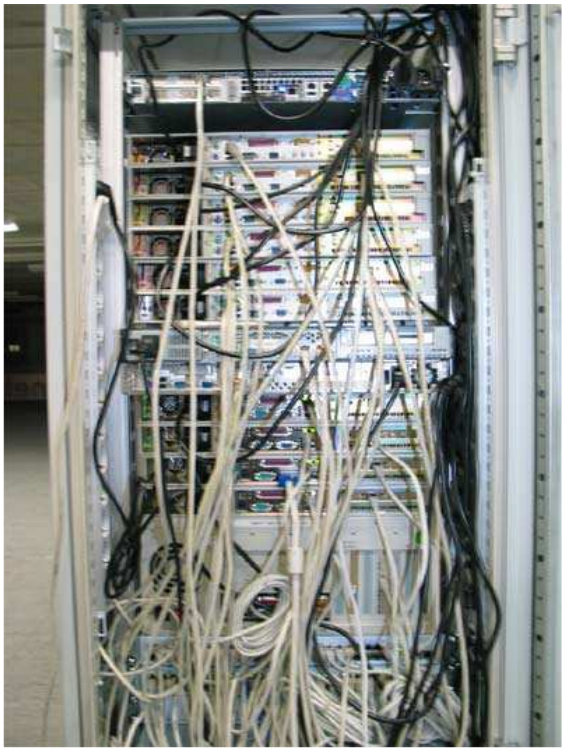
BizimAlem için oluşturduğum caching sistemi tüm performans sorunlarının kökünü kazıdı. Daha önce neden yapmamıştım acaba? ;-)

# BizimAlem Server Altyapısı

BizimAlem için aktüel sürümünde iki dolap dolusu sunucu hizmet vermektedir. Kullanılan altyapı ve sunucuların fotografları aşağıda yer almaktadır.



**Resim 27** BizimAlem server sistemleri

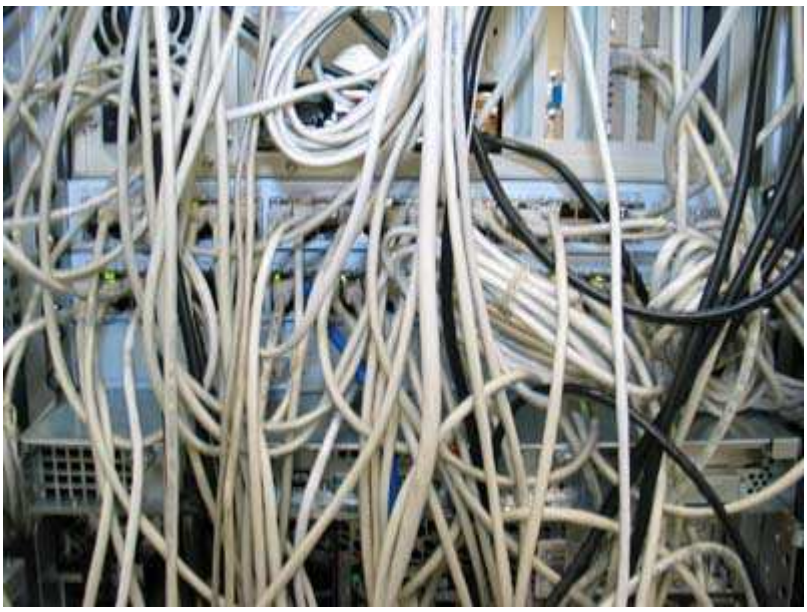


**Resim 28** BizimAlem server sistemleri

**Resim 29** BizimAlem server sistemleri

**Resim 30** BizimAlem server sistemleri

**Resim 31** BizimAlem server sistemleri

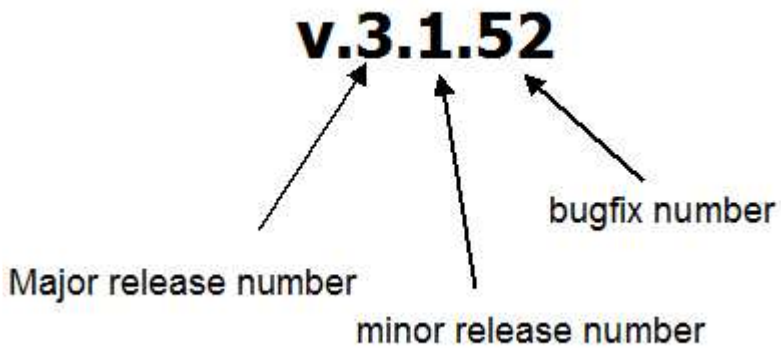


**Resim 32** BizimAlem server sistemleri

**Resim 33** BizimAlem server sistemleri

# BizimAlem Sürüm Numarası

Bugün itibariyle (14.1.2009) BizimAlem'in aktüel sürümü v.3.1.52 dir.




**Resim 34** BizimAlem sürüm numarasi

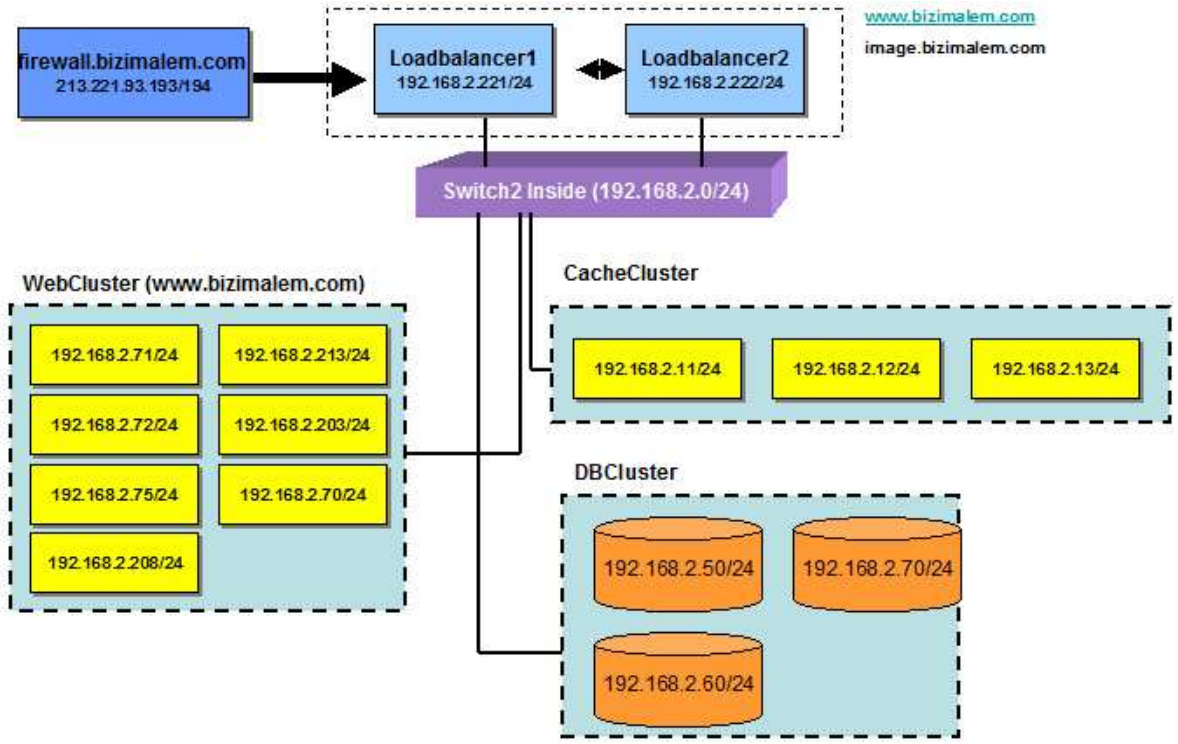
Sürüm numarası değişik bileşenlerden oluşmaktadır. Bunlar:

- **Major release**: 3 rakamı ana sürüm numarasını gösterir. BizimAlem bu hali ile 3. versiyonda. Sadece sistemde büyük çapta değişiklikler yapıldığında major release numarası bir artırılır. Büyük bir ihtimalle BizimAlem v.4 olmayacak, çünkü proje için 6 seneyi aşkın bir zamandır yapılması gereken herşey yapıldı.
- **Minor release**: 1 rakamı alt sürüm numarasını gösterir. Sisteme eklenen her yeni modül ile bu rakam bir artırılır. Örneğin su an planladığım bir chat modülü var. Bu modül sisteme eklendikten sonra v.3.2.0 sürümü oluşacak.
- **Bugfix number**: Oluşan her hatanın giderilmesiyle bu rakam bir artırılır. Ne yazık ki programalama esnasında kullanıcının yapabileceği tüm işlemler göz

önünde bulundurulamadığı için yeni hatalar oluşmakta ve bu hatalar her yeni sürümle giderilmektedir.

# Son Teknik Altyapı Şekli

v.3.1.52 ile oluşan teknik altyapı bir sonraki resimde yer almaktadır.




**Resim 35** v.3.1.52 sürümü teknik altyapı

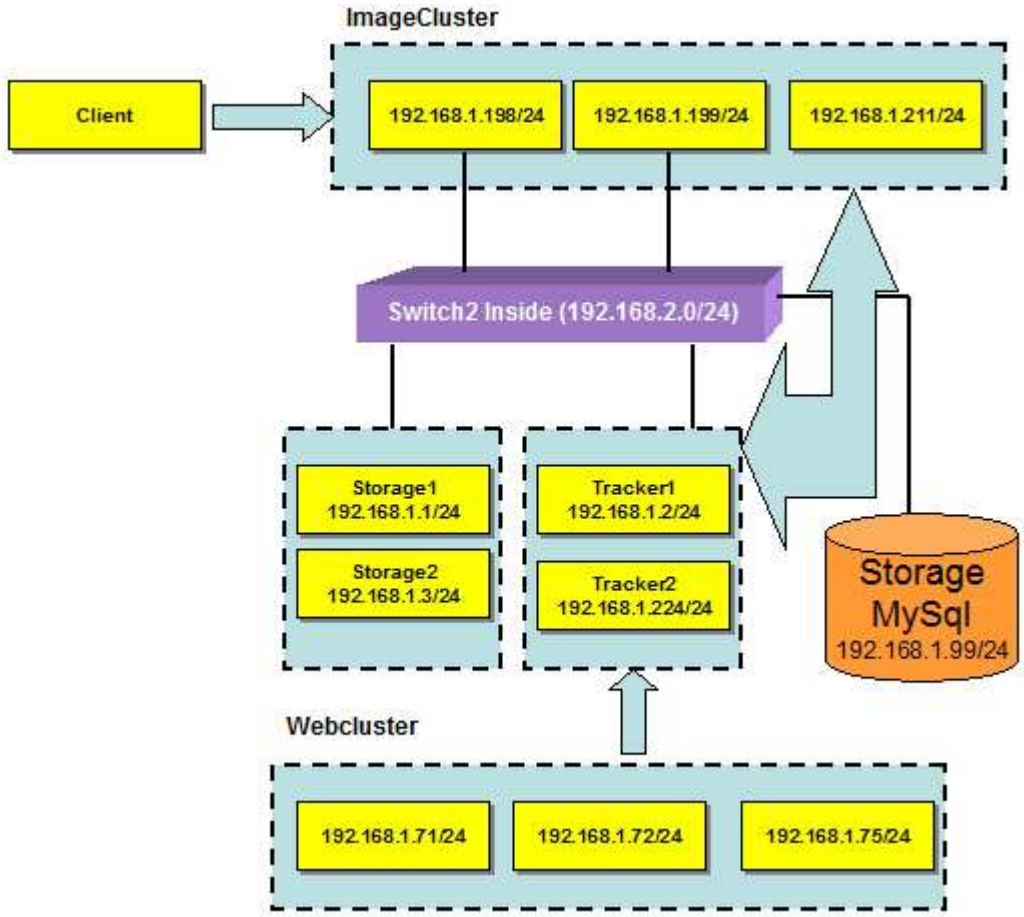
# BizimAlem Storage Subsystem

BizimAlem üyeleri kendi resimlerinin yer aldığı albümler oluşturabilirler. Her albüme limit olmadan resim yüklenebilir. Albümlerin adedi de limitli değildir. Yarım milyondan fazla üyenin olduğu bir platformda, kaç milyon resmin bu şekilde sisteme ekleneceğini düşünebilirsiniz. Bu kadar büyük bir resim kütlesini sistemde barındırabilmek için merkezi bir storage (dosya deposı) sistemi oluşturulması gerekmektedir. Bu storage sisteminin kapasitesi, sistemin diğer bölümlerini etkilemeden artırılabilmelidir, yani yukarıya doğru sınırı olmayan bir storage sistemi gerekli!

Uzun bir süre 2 TB kapasitesi olan bir server ile bu sorunları çözebileceğimi düşündüm. Lakin sunucu bir zaman sonra 1.9 TB sınırına dayandığında, sadece bir sunucu ile bu sorunu çözemeyeceğimi anladım. Bunun yanısıra bu sunucunun teknik sorunlardan dolayı bazen devre dışı kalması, tüm dosyaların sistem tarafından erişilemez hale gelmesini beraberinde getiriyordu. Uzun araştırmalar sonunda açık kaynaklı olan ve benim beklentilerime cevap verebilecek bir storage sistemi buldum. MogileFS!

# MogileFS ile Merkezi Storage Sistemi

MogileFS [9] Danga Interactive tarafından geliştirilmiş, dağınık çalışabilen (distributed) bir dizin sistemi (file system). Storage kapasitesini istediğiniz kadar, yeni sunucular ekleyerek yükseltebiliyorsunuz. MogileFS ile kurduğum storage sisteminin teknik mimarisi bir sonraki resimde yer almaktadır.




**Resim 36** BizimAlem Storage mimarisi

Storage sistemi değişik komponentlerden oluşuyor. Bunlar:

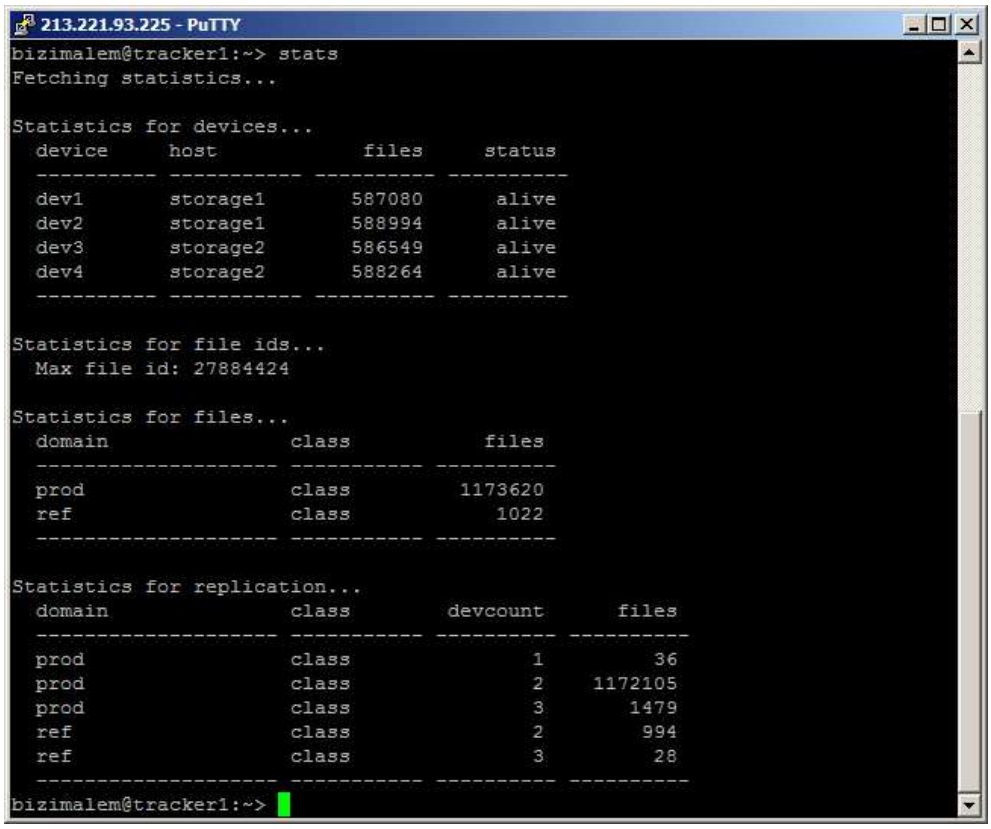
- **ImageCluster**: Storage sisteminde bulunan dosyalara ulaşmak için kullanılan sunucular. Storage sisteminde resimler, pdf dosyaları, word dokümanları vs olabilir. Bu dosyalara erişim ImageCluster üzerinden gerçekleşir.
- **Storage1:** Dosyaların depolandığı sunuculardan bir tanesi. 1 TB kapasiteli bir sunucu.
- **Tracker1:** Storage sisteminde bulunan her dosyanın unique (tek) bir numarası bulunmaktadır. Sisteme yeni bir dosya eklemek için webcluster içinde bulunan sunucular Tracker1/Tracker2 sunucuları ile bağlantı kurarak, dosyanın Storage1 ya da Storage2 de depolanmasını talep ederler. Bu sunucular dosyanın hangi Storage node üzerinde kayıtlı olduğunu tekrar Tracker1 ya da Tracker 2 üzerinden öğrenebilirler. Aynı şekilde bir kullanıcı bir dosyayı edinmek istediğinde

[9] Bakınız: http://www.danga.com/mogilefs/

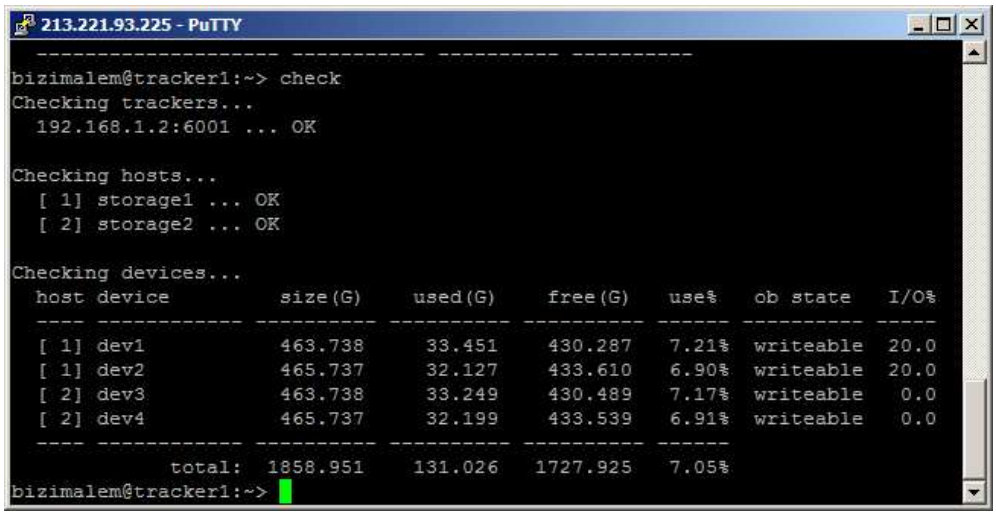
ImageCluster içinde bulunan bir sunucu Tracker1/Tracker2 ile bağlantı kurarak, dosyanın numarası ile Tracker üzerinden dosyayı edinir. Tracker hangi dosyanın hangi Storage node üzerinde olduğunu bilen komponenttir. Bu bilgiyi edinmek için MySQL bilgibankasını kullanır.

- **MySQL:** Hangi dosyanın hangi Storage node üzerinde olduğu ve hangi numaraya sahip oldugu MySQL bilgibankasında tutulur. Tracker MySQL bilgibankasına bağlantı kurarak, dosyanın hangi Storage node üzerinde olduğunu tespit eder. Tracker herhangi bir dosyayı Storage nodelardan birisine gönderdikten sonra, gerekli bilgileri MySQL bilgibankasına ekler. Storage sisteminin yönetimi Tracker ve MySQL komponentleri ile gerçekleşmektedir.

Mevcut storage kapasitesini artırmak için yeni StorageX sunucularının sisteme eklenmesi yeterli olmaktadır. Bu şekilde 1000 TB bile depolamak mümkün oluyor!




**Resim 37** Storage istatistikleri



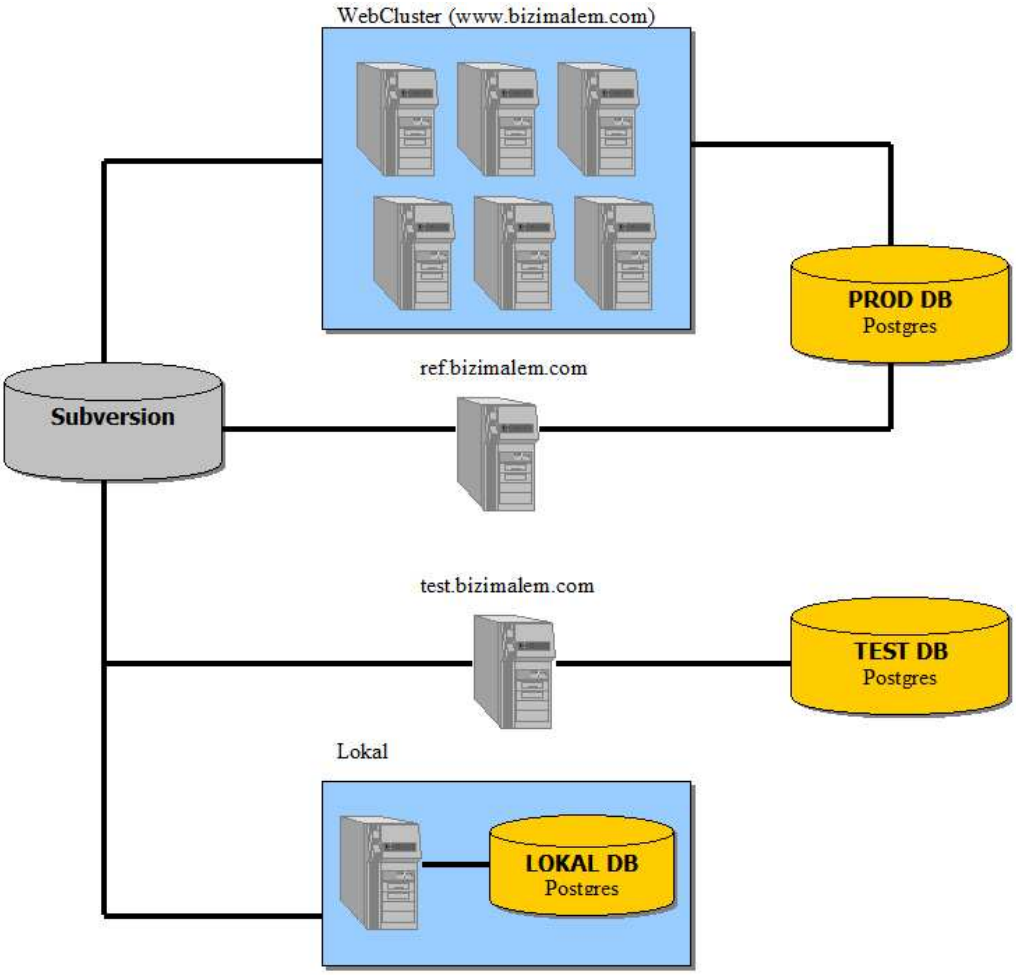

**Resim 38** Storage istatistikleri

Resim 37 aktüel storage istatistiklerini göstermektedir. Storage1 sunucu bünyesinde dev1 ve dev2 isminde iki harddisk bulunmaktadır. Bu harddisklerin kapasitesi 0.5 TB dir. Storage2 de aynı konfigürasyona sahiptir. Bu konfigürasyonda storage sisteminde iki Storage node sunucu ve 4 hardisk bulunmaktadır. Resim 37 de her harddisk bünyesinde 600.000 a yakın dosyanın bulundugu görülmektedir. Sistem tarafından oluşturulan her dosyanın iki kopyası Storage1 ve Storage2 üzerinde paylaştırılır. Bu durumda Storage nodelardan birisi devre dışı kalsa bile dosyanın ikinci kopyasına diğer Storage node üzerinden ulaşmak mümkündür.

Resim 38 de Storage node kullanım istatikstikleri yer almaktadır. dev1 ve diğer harddiskler yaklaşık 500 GB kapasiteye sahiptir. Kullanılan storage alanı 34 GB civarındadır. dev1 ve dev2 bu snapshot yapıldığında %20 I/O (Input / Output) trafiğine sahiptir.

# BizimAlem Development Staging Server

Yazılım süreci değişik evrelerden oluşur. Programcılar oluşturdukları kodu yazılım için kullandıkları bilgisayarlarında lokal test ederler. Bunun yanısıra kodun entegrasyonu için merkezi bir sunucu kullanılır. Kod merkezi versiyon kontrol sistemine eklendikten sonra, belirli aralıklarla yeni test sürümleri oluşturularak, bir test serveri üzerinde çalıştırılır. Bu test ortamının kendine ait bir bilgibankası vardır. Bu testler olumlu sonuç verdikten sonra oluşturulan sürüm referans olarak isimlendirilen server üzerinde çalıştırılır. Referans serveri merkezi bilgibankası sistemini kullandığı için, oluşan kod gerçek şartlarda test edilmiş olur.



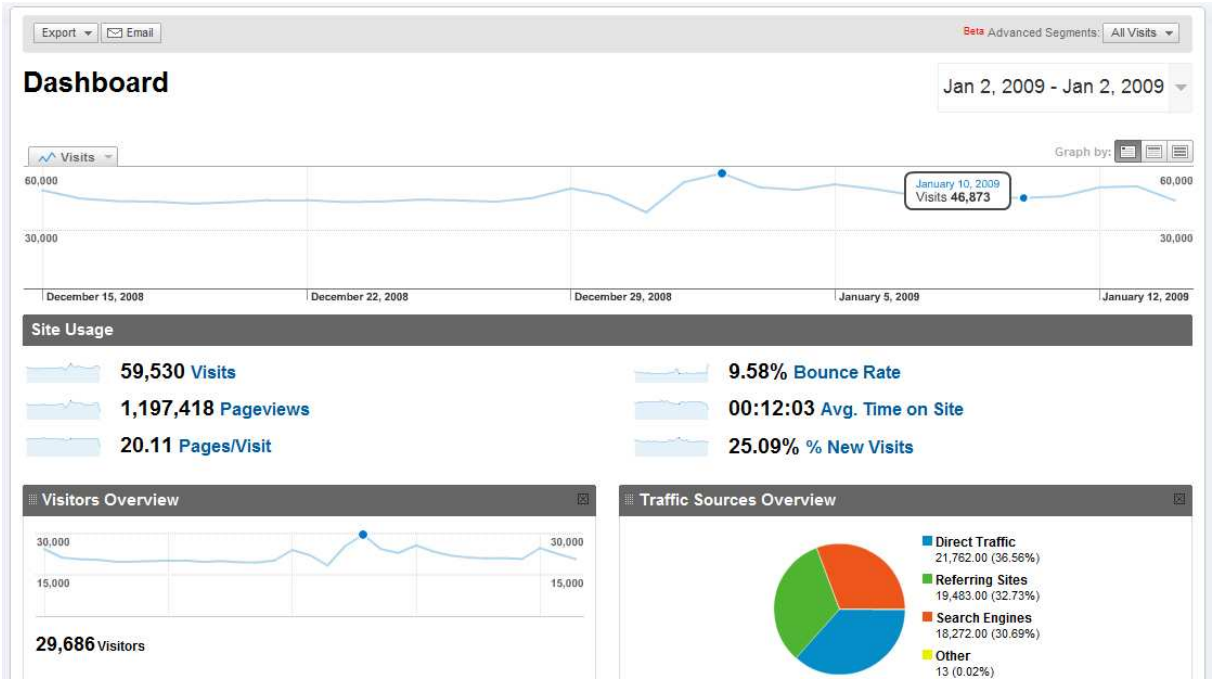

**Resim 39** BizimAlem staging server yapısı

Kodun değişik ortamlarda değişik şartlarda test edilmesi için kullanılan server sistemlerine staging server ismi verilir. BizimAlem için kullanılan staging server yapısı resim 39 da yer almaktadır. Bunlar:

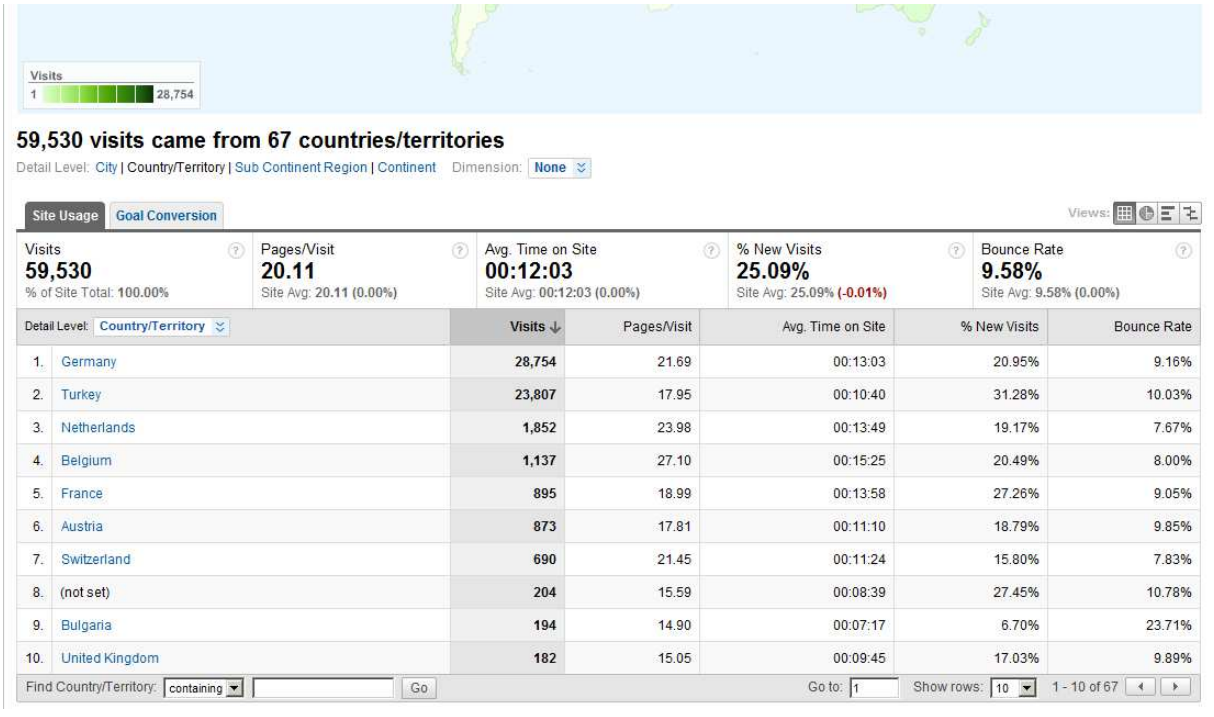
- **Lokal**: Yazılımı kendi bilgisayarimda yapıp, lokal çalışan bir bilgibankası ile test ettikten sonra oluşan kodu versiyon kontrol sistemine eklerim.
- **test.bizimalem.com**: Entegrasyon testleri test.bizimalem.com serverinde gercekleştirilir. Yeni bir test sürümü oluşturarak, bu sürümü test.bizimalem.com isimli sunucu da deploy (aplikasyonun çalışır hale getirilmesi) ederim. test.bizimalem.com kendine ait bir bilgibankasına sahiptir.
- **ref.bizimalem.com**: Entegrasyon testi tamamlanmış sürüm ref.bizimalem.com serverine deploy edilir. ref referans kelimesinin kısaltılmış halidir. Ref sistemi çalışan ana produktif sistem ile aynı bilgibankasını paylaştığı için (PROD DB), oluşturulan yeni kodun gerçek şartlarda test edilmesi sağlanır. Çoğu ref sistemlerinde PROD DB den bir kopya alınarak REF DB oluşturulur. Ben sürüm oluşturma işlemini kısaltmak için oluşturduğum ref sisteminin PROD DB yi kullanması sağladım.
- **www.bizimalem.com**: Gerçek sistem.

Kod ref sisteminde başarılı bir şekilde test edildikten sonra v.3.2.0 şeklinde yeni bir sürüm oluşturularak, bu yeni sürüm WebCluster icinde bulunan serverlere deploy edilir.

# Google Analytics İstatistikleri



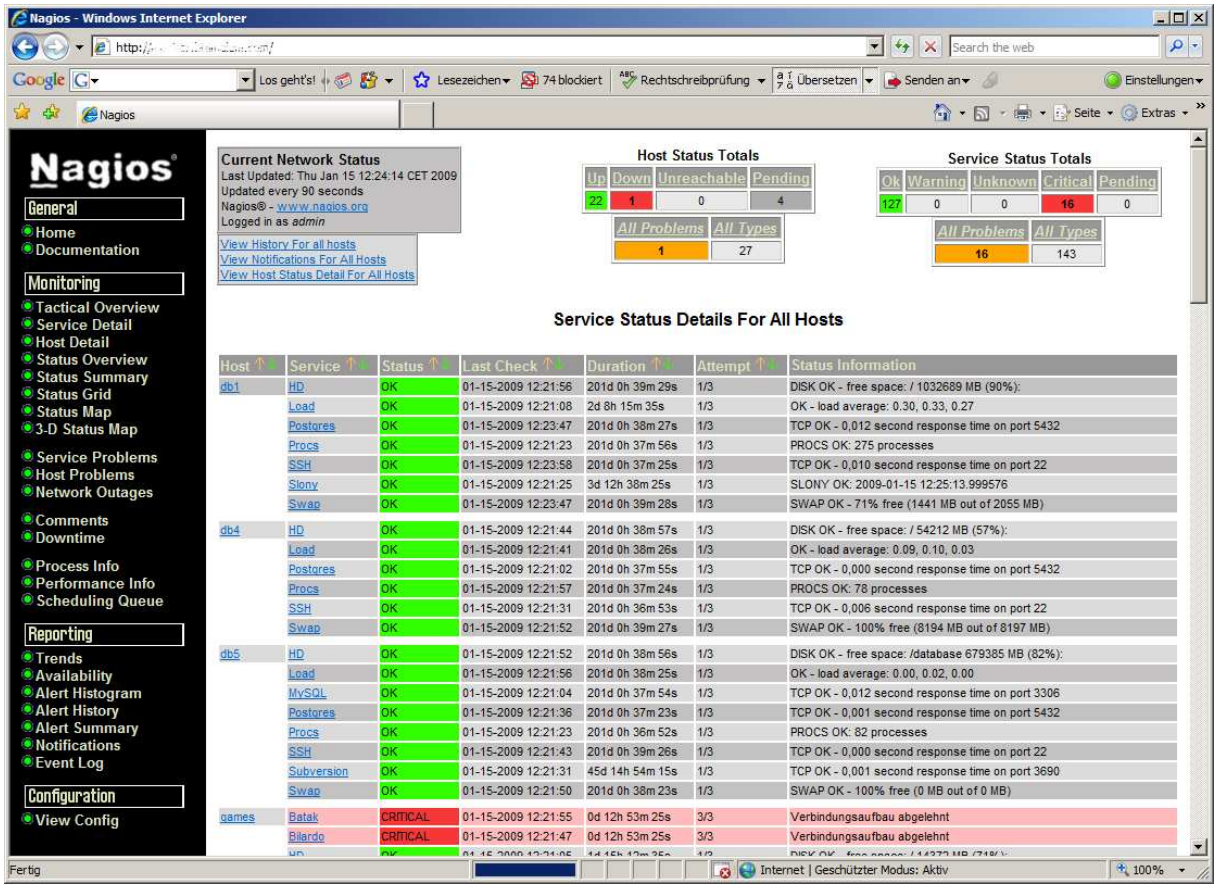

**Resim 40** Google BizimAlem.com istatistikleri




**Resim 41** Google BizimAlem.com istatistikleri

# Ağ ve Servis Yönetimi

Kırka yakın server ve ağ komponentinin oluşturduğu bir sistemi yönetmek ve kontrol altında tutabilmek için bir network monitoring sistemine ihtiyacınız var. Her sunucu üzerinde otomatik olarak kontrol (monitoring) edilmesi gereken birden fazla servis olacaktır. BizimAlem'in ağ ve sunduğu servislerin kontrolünü Nagios[10] isimli network monitoring sistemi ile yapıyorum.




**Resim 42** Nagios

Resim 42 de yer aldığı gibi her sunucu (server) için kullanılan servisler Nagios tarafından belirli aralıklarla kontrol edilmektedir. Çalışır durumda olan servisler için Status alanında OK yer almaktadır. Çalışır durumda olmayan, yani devre dışı kalmış servisler için Status alanında CRITICAL bilgisi yer alır. Devre dışı kalan her sunucu ve servis için Nagios ağ yöneticilerine otomatik olarak email göndererek, durumu bildirir.

# Son

BizimAlem.com projesi yedi yaşını doldurdu ve Avrupa'da geniş bir kitleye hitap eden bir web platformu haline geldi. Bu makalemde böyle bir projenin hangi evrelerden geçtigini ve artan talep sonucunda nasıl büyümesi gerektiğini anlatmaya çalıştım. Daha önce de belirttiğim gibi BizimAlem.com benim için bugün bile büyük bir yazılım laboratuarı. Birçok yeni

10 Bakınız: http://www.nagios.org

teknolojiyi bir laboratuarda deneme ve öğrenme imkanı buluyorum. Oluşan sorunları çözmek için değişik alanlarda ihtisas yapmam gerekiyor. Bu severek yaptığım birşey, çünkü yazılımcı olmamın yanısıra başka disiplinlerde bilgi edinme şansı buluyorum. Umarım sizinde başınıza böyle güzel birşey gelir. Büyük düşünmekten ve oynamaktan çekinmeyin!