T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİ TEKNOLOJİLERİ PROGRAMI

WEB UYGULAMALARINDA ÇERÇEVE MODEL KULLANIMININ PERFORMANS FAKTÖRLERİNE ETKİLERİNİN İNCELENMESİ

YÜKSEK LİSANS TEZİ

Rind Devran Tukan

Tez Danışmanı: Doç. Dr. Adem Karahoca

İstanbul, 2009

İÇİNDEKİLER

TABLOLAR

ŞEKİLLER

KODLAR

KISALTMALAR VE TERİMLER

MVC: Model-Görünüm-Denetleyici

Framework: Çerçeve modeli

ORM: Nesne İlişkisel Eşleştirme

API: Uygulama Programlama Arabirimi

XML: Gelişmiş işaretleme dili

DSN: Veri Kaynağı

AJAX: Ansenkron Javascript ve XML

DQL: Doctrine Sorgu Dili

SQL: Yapısal programlama dili. Yapısal programlamanın araçları ile donatılmış dil.

WAF: Web Uygulama Çerçeve Modeli

.**Net:** Microsoft tarafından geliştirilmekte olan yazılım çerçeve modeli

EE: Kurumsal Versiyon

CSS, XSS: Siteler Arası Betikleme

CRSF: Siteler Arası İstem Sahtekârlığı

UI: Kullanıcı arayüzü

PAC: Sunum/Soyutlama/Denetim

APC: Gelişmiş PHP Önbelleği

DBMS: Veritabanı Yönetim Sistemleri

PHP: Php Web Programlama Lisanı

Mysql: Sun Microsystems tarafından geliştirilmekte olan SQL ürünü

Postgresql: PostgreSQL Development Group tarafından geliştirilmekte olan SQL ürünü

YAML: YAML Ain't Markup Language. YAML bir işaretleme lisanı değildir. Tüm programlama dilleri için tasarlanmış seri veri gönderimi için insan odaklı bir veri standartıdır.

EJB: işe yönelik yeniden kullanılabilir Java elemanları

OO: Nesne yönelimli

OOP: Nesne yönelimli programlama

RIA: Zengin içerikli Internet uygulamaları

URL: İnternet kaynak belirteci

CGI: Bir grafik sisteminin aygıta bağımlı ve aygıttan bağımsız kısımlarının standartlaştırılmış arayüzü.

HTTP: İstemci ve sunucular arasında üstmetin belgelerinin gönderimini sağlayan uygulama protokolü.

JSON: Javascript nesne tanımlaması

HQL: Java platformunda kullanılan bir ORM olan hibernate'in kullandığı sorgu dili

W3C: Web standartları konusunda çalışmalar yapan bir şirketler birliği

Script.aculo.us: Html efektleri için geliştirilmiş güncel bir javascript kütüphanesi

ÖZET

WEB UYGULAMALARINDA ÇERÇEVE MODEL KULLANIMININ PERFORMANS FAKTORLERİNE ETKİLERİNİN İNCELENMESİ

Tukan, Rind Devran

Bigi Teknolojileri

Tez Danışmanı: Doç. Dr. Adem Karahoca

Temmuz, 2009, 145

Php, 1993 senesinden bu güne oldukça kalabalık ve etkin uygulama geliştiriciler tarafından 2006 senesinin itibariyle tüm dünya çapında 25 milyondan daha fazla alan adı tarafından kullanılmakta ve sayısı giderek artmaktadır. Yazılım mühendisliğinin güncel ve yenilikçi bir dalı olan tasarım örüntülerinin kullanılmasına başlamasıyla yazılım çerçeve modelleri gelişimi mümkün olmuştur. Çerçeve model kullanımı artık birçok uzman yazılım firması tarafından kullanılmaya başlanmıştır. Zengin Internet uygulamalarında günümüzün en değerli web uygulamaları çerçeve model kullanmaktadır. Bu çalışma içerisinde Mvc tasarım örüntüsü üzerine kurulmuş Mvc mimarisinin sunduğu avantajlar detaylı olarak incelenmiştir. Ayrıca bu çalışmada bir çerçeve modelin uygulamaya beraberinde getirdiği ORM kullanımı, AJAX kullanımı gibi yeni yazılım teknolojilerin kullanımı incelenmiş performans, güvenlik ve veri erişimi gibi yazılımda kaliteyi belirleyen kritik noktalara getirdiği yenilikler analiz, test edilmiştir.

Anahtar Kelimeler: Çerçeve Model, Tasarım Örüntüsü, MVC

ABSTRACT

THE EFFECTS OF FRAMEWORK USAGE ON PERFORMANCE FACTORS OF WEB APPLICATIONS

Tukan, Rind Devran

Information Technology

Supervisor: Assoc. Prof. Dr. Adem Karahoca

Temmuz, 2009, 145

Php is the most used programming language in web application development. Php has developed since 1993 and now over 25 million domain user. After foundation of design patterns in computer science it is possible to use software frameworks. Nowadays, most complex rich Internet applications are develeped by web frameworks. This work contains detailed architectural information related with MVC architecture. Also this thesis contains informations and various performance tests about the new web technologies such AJAX, ORM, Caching. The main concept of this thesis is to give developers intensive information about how frameworks affects sofware development and what kind new technologies can be used with frameworks.

Keywords: Web Frameworks, Design Patterns, MVC

1.GİRİŞ

Web'in gelişmesi, insanların ve kurumların web uygulamalarından beklentilerinin artması web uygulamalarının daha gelişmiş, daha karmaşık ve daha güvenli bir yapıda olma zorunluluğunu gündeme getirmiştir. Bütün bu gereksinimler ise yazılım geliştiricilerini yazılım mühendisliği prensiplerine, W3C tarafından konulmuş standartlara uygun daha kolay ölçeklenebilir, kolay geliştirilebilen, daha güvenli ve bunların neticesinde daha kaliteli uygulamalar geliştirmeye yöneltmiştir. Yazılım mühendisliğindeki tasarım örüntülerinden bir tanesi olan modal-view-controller in web de kullanılması ile kullanıcı arayüzlerindeki yapılması istenen değişikliklerin uygulama katmanında herhangi bir değişiklik yapılmasına gerek kalmadan gerçekleştirilebilmesi sağlanabilir hale geldi. Güncel web programlama dillerinde MVC tasarım örüntüsünün belirli prensipler ile beraber kullanılması sonucu web uygulamaları geliştirilmesinde MVC pattern ve framework kullanımı dönemi başlamış oldu.

Framework'ler temel olarak sık kullanılan fonksiyon, class, kütüphane ve kod parçacıklarının yer aldığı paketler olarak tanımlanabilir. Kimileri PHP Framework'lerinde olduğu gibi yalnızca belirli bir programlama diline hizmet ederken, kimileri ise birçok programlama dili tarafından ortak kullanılabilir (.NET Framework gibi). Framework'ler sayesinde uygulama geliştiriciler daha önceden belirlenmiş yazım kuralları dâhiline, framework içerisinde bulunan metodları kullanarak bir adım önde başlarlar. Frameworkler uygulama geliştiricilere düzen ve tekrardan kullanılabilirlik konusunda oldukca yardımcı olurlar. Framework kullanılarak geliştirilmiş bir projedeki modüller bir benzer projeye oldukça kolay bir şekilde uyarlanabilir. Bu şekilde daha önce yazılan kodlar tekrardan kullanılabilir hale gelmiş olur. Bu web uygulamalarında büyük oranda zaman kazanmayı sağlarken, masa üstü uygulamalarda uygulamaların daha az yer kaplamasına da yardımcı olur. Frameworklerin getirdiği kod yazma standardı uygulama geliştiricileri daha yüksek standartda kod yazmaya yöneltir. Bu durum da üretilen yazılımın kalitesini oldukça arttırmış olur. Frameworkler ayrıca içinde barındırdıkları özellikleriyle yazılım geliştiricilere güvenlik, konfigurasyon kolaylığı ve esneklik sağlar.

1.1. ÇALIŞMANIN AMACI

Bu çalışmanın amacı, web uygulamalarının geliştirilmesinde MVC çerçeve modellerinin kullanımının yazılımın performans ve güvenliğine olan etkilerini yazılım mühendisliği bakış açısı altında incelemektir.

1.2. ÇALIŞMANIN YÖNTEMİ VE İÇERİĞİ

Bu çalışmada öncelikle tasarım örüntülerinden(Fowler,2002) biri olan MVC tasarım örüntüsü(Carillo, 2006) ile geliştirilmiş yazılım çerçeve modellerinin ortak özellikleri incelenmesi, tanımlaması yapılmıştır. Gelişmekte olan zengin tabanlı web uygulamalarında AJAX kullanımı ile bilgiler ve yazılım çerçeve modellerinin bu konuda sağladığı yeniliklerden bahsedilmiştir. Symfony ve Standart Ajax uygulaması arasındaki kolaylıklar incelenmiştir. Object relational mapping (ORM) tanımı, uygulama örnekleri (Doctrine, Propel) ve web uygulamalarının geliştirilmesindeki öneminin incelenmesi. İncelenen ORM'lerde aynı betikler kullanılarak test ve testlerin değerlendirilmesi yer almaktadır. Web uygulamalarındaki önbelleklemenin öneminin vurgulanması, Php ve Symfony(Potencier, 2007) ile örnek önbellekleme uygulamaları yazımı ve bunların testlerinin değerlendirilmesi yer almaktadır. Web uygulamalarının güvenliğini tehdit eden unsurların incelenmesi bu tehditlerden nasıl kaçınılabileceği ile ilgili bilgiler verilmekte ayrıca Symfony kullanılarak bir form uygulamasının nasıl güvenli hale getirilmesi anlatılmaktadır.

2. WEB UYGULAMALARINDA YAZILIM ÇERÇEVE MODELLERİ

Bilişim sistemleri ortamında, Çerçeve Model, diğer yazılım uygulamalarının düzenlenip geliştirilmesine olanak tanıyan tanımlı bir destek yapısıdır. Çerçeve Model, bir yazılım uygulamasının farklı bileşenlerinin geliştirilmesine ve birbirine yapıştırılmasına yardım etmek için destek programları, kod kitaplıkları, komut dosyası dili, ortak hizmetler, arayüzler ya da diğer yazılım paketleri/yardımcı programlar içerebilir. Yazılım çerçevesi, bir yazılım sistemi ve/veya alt-sistemi için tekrar kullanılabilir bir tasarım ve yapıtaşıdır. Bir yazılım çerçevesi, mali modelleme uygulamaları ya da karar destek sistemleri gibi, farklı etki alanlarında uygulamalar oluşturmak üzere uyarlanabilir. Bir yazılım çerçevesi, donmuş noktalardan ve sıcak noktalardan oluşur. Donmuş noktalar, bir yazılım sisteminin genel mimarisini(temel bileşenlerini ve aralarındaki ilişkileri) tanımlar. Bunlar uygulama çerçevesinin herhangi bir örneklemesinde aynı (donmuş) kalır. Diğer yandan, sıcak noktalar, yazılım çerçevesinin bireysel yazılım sistemleri olan parçalarını temsil eder. Sıcak noktalar genel olacak şekilde tasarlanmıştır. Diğer bir deyişle, geliştirilmekte olan uygulamanın ihtiyaçlarına uyarlanabilir.

Yazılım çerçeveleri, mimaride belirli işlevsellik için uyarlamaların yapılması gereken yerleri – sıcak noktaları – tanımlar. Nesneye yönelik bir ortamda, çerçeve, soyut ve somut sınıflardan oluşur. Böyle bir çerçevenin örneklemesi, mevcut sınıfları oluşturmak ve altsınıflara bölmekten oluşur. Bir yazılım çerçevesiyle somut bir yazılım sistemi geliştirilirken, sıcak noktalar, sistemin özel ihtiyaç ve gerekliliklerine göre özelleştirilir. yazılım çerçeveleri, Hollywood Prensibine dayanır: "Bizi aramayın, biz sizi ararız.". Bu da yeni altsınıflar gibi kullanıcı tanımlı sınıfların önceden tanımlanmış çerçeve sınıflarından mesajlar aldığı anlamına gelir. Bunlar genellikle üst sınıf soyut yöntemler uygulanarak, Şablon tasarım örüntüsü kullanımına benzer bir şekilde işlenir.

2.1. YAZILIM ÇERÇEVESİ TÜRLERİ

Genel olarak, bilişim sistemleri alanında yedi yazılım çerçevesi türü vardır:

- Kavramsal Çerçeve – Zachman Çerçevesi gibi kapsamlı mimari model;
- Uygulama Çerçevesi – WebWork gibi bir uygulama çözümü için iskelet yapı;
- Etki Alanı Çerçevesi – IBM Bilişim Çerçevesi (IFW) gibi belirli iş sektörlerine özel yapılmış olanlar;
- Platform Çerçevesi – .Net ve Java EE(Shan, 2006) çerçevesi gibi programlama modeli ve çalışma ortamı;
- Bileşen Çerçevesi – nesneye dayalı eşleştirme için Hibernate, iBatis veya Cayenne gibi bir uygulamanın yapıtaşları;
- Hizmet Çerçevesi – Semantic(anlamsal) Web Hizmetleri Çerçevesi gibi hizmete yönelik programlama için iş ve teknik hizmetler modeli;
- Geliştirme Çerçevesi – Eclipse, Netbeans veya OSGi gibi, tipik olarak IDE'ler için zengin istemci geliştirme aracı oluşturmaya yönelik bir yapı temeli.

2.2. WEB UYGULAMA ÇERÇEVESİ TANIMI

Web Uygulama Çerçevesi (WAF), yaygın olarak Http protokolü aracılığıyla web tarayıcılarına hizmet eden özel web uygulamaları üretmek için özelleştirilebilecek, yeniden kullanılabilir, yarı tamamlanmış iskelet bir modüler platformdur. Gelişmiş özellikler açısından zengin iş hizmeti ve işbirliği sistemleri oluşturmak için elzem olan hizmetlerin ve bileşenlerin yapıtaşlarını içerir. WAF, Java EE ve .Net modellerinde web tabanlı istem-yanıt uygulamaları geliştirmek için genellikle Model-Görünüm-Denetleyici (MVC) tasarım örgüsünü, tipik olarak Model 2 mimarisini uygular. Ayrıca arama, sürüm belirleme ve temel iş nesnelerine izinler gibi hizmetleri de bünyesinde barındırarak, uygulamaların çok az ek çalışmayla çerçeve hizmetlerini desteklemesine olanak tanır. Bu bağlamda, WAF, bir önceki bölümde tanımlandığı üzere, özellikle HTML/XML'e hizmet eden http tabanlı iletişimler için bir Uygulama Çerçevesi türüdür.

Web Uygulama Çerçevesi etki alanı katmanı, genellikle kullanıcı, grup ve izin gibi temel kavramları örneklendirir. WAF ayrıca kullanıcı arayüzü (UI) çerçevesi, hızlı

geliştirme ve web kullanıcı arayüzlerinin yeniden kullanımı için tasarlanmış bir UI bileşen kitaplığı ve güçlü bir nesneye dayalı kalıcı motor/yardımcı program gibi diğer ilgili parçaları da içerir.

2.3. WEB UYGULAMA ÇERÇEVELERİ NEDEN KULLANILMALIDIR?

2.3.1. Açık Standart Mimarisi

Yazılım çerçeveleri, web uygulamaları geliştirmek ve sürdürmek için gerekli zamanı, çabayı ve kaynak miktarını önemli ölçüde azaltır. Ayrıca bir çerçeve, yaygın olarak kabul edilen standartları (örn. Java, .Net, XML, XSLT, JAAS, Servlet, JSP, JDBC, ADO.Net) ve teknolojileri (örn. JUnit, XUnit, Ant, Log4j, JDom, Xalan, Xerces, Lucene) temel alarak herhangi bir deneyimli geliştiricinin dik bir öğrenme eğrisi olmadan sistemi hızla geliştirip desteklemesine olanak tanıyan, açık bir mimaridir. Teknolojileri benimseme ve entegre etmeye ilişkin bu en iyi harman yaklaşımı, uygulama tasarımcılarının iş sorunlarını çözmeye odaklanmalarına olanak tanır. Bu nedenle Web Uygulama Çerçevelerini web uygulamaları için standart geliştirme altyapısı olarak benimsemek, geliştirmenin herhangi bir özel ve çıkmaz yapıda kilitlenmemesini sağlamanın en iyi yoludur. Endüstri standardı olan açık kaynak çerçeveler tüm dünyada son derece yetenekli profesyoneller tarafından aktif olarak sürdürüldüğü ve geliştirildiği için, bu yaklaşım teknoloji kaçışını ve riskini ciddi ölçüde azaltır. Bu geliştiriciler, uygun teknolojileri tanımlama, yazılımı entegre etme, bu teknolojileri test etme ve mevcut kullanıcılar için en son teknolojiye geçiş yolu sağlama sorumluluğunu üstlenirler.

Web Uygulama Çerçevesi tipik olarak bir n-katmanı mimarisinde konuşlandırılır ve kanıtlanmış, standart bir teknoloji kullanır. Standart teknolojiler kullanan bir çerçeve, mevcut şirket altyapıları dâhilinde kolayca konuşlandırılarak, mevcut donanım, yazılım, süreçler ve insanları destekler.

2.3.2. Etki Alanı Hizmetleri

Esas itibarıyla tüm web uygulamaları, kullanıcı yönetimi (örn. güvenli kullanıcı girişi, parola kurtarma), grup yönetimi ve erişim yetkisi gibi, ortak bir temel gereklilikler

grubuna sahiptir. Web Uygulama Çerçevesi genellikle tüm bu işlevsellikleri yüzlerce üretim dağıtımı aracılığıyla geliştirilmiş olarak içerir ve geliştirmecileri kendi özel uygulamalarının ihtiyaçlarına odaklanmalarını sağlayacak şekilde özgürleştirir. Temel bir hizmetler grubuna ek olarak, web uygulamaları tipik olarak iki önemli benzerlik daha taşır: önemli verileri bağlantısal bir veritabanında saklarlar ve kullanıcılarla web tabanlı bir kullanıcı arayüzü aracılığıyla etkileşirler. Gelişmiş bir nesneye dayalı kalıcı katman, model nesnelerinin veritabanında nasıl saklanacağını otomatik olarak yönetir. Kalıcı katman, üst-veriden optimize edilmiş SQL üreterek, veritabanı şemalarını örneklemek ve yeniden faktörlemek ya da ek veritabanı mimarilerini desteklemek için gereken çaba miktarını ciddi ölçüde azaltır.

Web Uygulama Çerçevesi ayrıca çerçeve oluşturan bileşen tabanlı bir sunum da içererek geliştiricilerin mevcut UI bileşenlerini genişletme ya da XSL ve etiket kitaplıkları gibi, bir uygulama boyunca yeniden kullanılabilecek yeni bileşenler oluşturma olanağı sunar. Veri doğrulama, sürüm belirleme, sınıflandırma, baskı, sayfa gezinme ve tam metin arama gibi geniş bir etki alanı hizmetleri yelpazesi temin edilebilir. Bir Web Uygulama Çerçevesi'nin üstünde yazılı herhangi bir uygulama net bir şekilde ve derhal bu temel hizmetlerden faydalanabilir.

2.4. WEB UYGULAMA ÇERÇEVELERİ ÇÖZÜMLERİ

2.4.1. Tasarım Felsefesi

Bir web uygulama çerçevesi geliştirmek için uygulanan temel tasarım ilkeleri aşağıdaki gibidir.

- **Sadelik** – bir çerçeve kullanmak için daha az ve daha basit kodlar yazılmalıdır. XML yapılandırma dosyalarının aşırı kullanımından kaçınılmalı, POJO merkezli tasarımdan faydalanılmalıdır.
- **Tutarlılık** - bileşenler, kapsayıcılar ve kurallar tutarlı olmalıdır.
- **Verim** – uygulamalar iyi bir performans sergilemeli ve tercih edilen yapışkan oturumlar aracılığıyla kümeleme desteğiyle ölçeklenmelidir.

- **Entegrasyon** – bir çerçeve mevcut iyi çözümlerle rekabet etmemeli; kusursuz bir entegrasyon sağlamalıdır.
- **Yeniden kullanılabilirlik** – bir çerçevedeki yapılar tamamen yeniden kullanılabilmeli ve dağıtımı kolay olmalıdır.
- **Kesintisiz** – HTML ya da diğer biçimlendirmeler; programlama anlambilimi, sıradan HTML editörleriyle uyum ve grafik tasarımcılarının çerçeve etiketlemesini tanıyıp kaçınmasını sağlayacak kolay kullanımla kirlenmemiş olmalıdır.
- **Tanı** – işler ters gittiğinde, model akıl karıştırmamalıdır; faydalı tanı ve hata ayıklama bilgileri temin etmelidir.
- **Geliştirme araçları** – özel araçlara minimum bağımlılıkla maksimum araç desteği.

2.4.2. Web Uygulama Çerçevesi Türleri

Neredeyse tüm Java Web uygulama çerçeveleri(Halabi, 2005), MVC örgüsünü temel alır. Genel olarak, şu anda beş büyük web uygulama çerçevesi ekolü vardır: istek tabanlı, bileşen tabanlı, karma, üst ve RIA tabanlı çerçeve. İstek tabanlı bir çerçeve, orijinal CGI tanımlamasına çok yakındır. Gelen istekleri doğrudan işleyen denetçiler ve aksiyonlar kullanır. Her aksiyon temelde statiktir. Sunucu taraflı oturum uygulamasıyla belli bir dinamizm derecesine ulaşılır. Çeşitli çerçeveler temelde mantığı URL'lere eşleştirme ve veriyi yapılandırma ve iş işleyicilerine temin etme şekliyle farklılaşırlar. Bileşen tabanlı bir çerçeve, aksiyonu işlemenin doğasını özetler ve mantığı genelde web ortamından bağımsız, yeniden kullanılabilir bileşenlerle açıklar. Durum, her bileşen oluşumunda mevcut veriye bağlı olarak, çerçeve tarafından otomatik olarak işlenir. Bazı olay işleme biçimleriyle birlikte, bu geliştirme modeli, masaüstü GUI araç takımlarının sunduğu özelliklere çok benzerdir. Çeşitli çerçeveler temelde temin edilen bileşen API'si ve bileşenlerin birbirlerine entegre olma biçimleriyle farklılaşır. Karma çerçeve, istek tabanlı bir çerçevedeki tüm veri ve mantık akışını kontrol altına alarak, istek tabanlı ve bileşen tabanlı çerçeveleri birleştirir. Geliştirmeciler, CGI uygulamalarının mimarisine yakınlığı korurlar ve URL'ler, formlar, parametreler, tanımlama bilgileri ve yol bilgileri üzerinde tam kontrol sahibidirler. Bununla birlikte, karma çerçeve, işlemleri ve denetleyicileri doğrudan istekle eşleştirmek yerine, bireysel sayfalar, yolu kesilen

istekler, portal benzeri sayfa parçaları ve entegre edilebilir parçacıklar gibi birçok farklı durumda aynı şekilde hareket eden bir bileşen nesne modeli temin eder. Bileşenler birbirine bağlanabilir ve kendi başlarına birer bileşen olan gruplar halinde paketlenebilir. Ayrı ayrı dağıtılabilir ve diğer projelere kusursuzca entegre edilebilirler. Bu da bileşen tabanlı çerçevelerdeki yeniden kullanılabilme özelliğini istek tabanlı yaklaşımın ham kontrolüyle birleştirir.

Üst çerçevede ortak hizmetler için bir dizi çekirdek arayüz ve bileşenlerle hizmetleri entegre etmek için son derece genişletilebilir bir omurga vardır. Yapı tipik olarak bağlantıların diğer çerçeve ve bileşenleri esnek şekilde içermek üzere ayrılması için Denetim Evirme örgüsünü(Fowler, 2002) uygular. Üst çerçeve zaman zaman çerçeveler çerçevesi olarak değerlendirilir.

Zengin Internet Uygulaması (RIA) ile sürükle ve bırak, ağaç denetimleri, sekmeli paneller gibi, "şişman" istemcide yaygın olan zengin kullanıcı arayüz özelliklerine sahip bir tarayıcıda çalışan web sayfası tabanlı bir uygulama kastedilir. RIA tabanlı çerçeve, sunucu taraflı iletişimi en aza indiren bir istemci kapsayıcısı modeli kullanır – yani bu çerçeve, kullanıcının her tıklayışında bütün bir HTML sayfasını yüklemek yerine, ya tıklamayı yerel olarak (sunucuyu dâhil etmeden) işler ya da veriyi sunucudan XML formatında ister. Bu da gerçekten dinamik bir istemci taraflı uygulama ve kullanıcı etkileşim modeli olduğu anlamına gelir – yani istemci bir sunucuda oluşturulmuş bir web sayfasından çok fazlasıdır.

2.4.2 Php Web Uygulama Çerçevelerinin Gelişimi

Tablo 2.1 - Güncel Php çerçeve modelerinin kıyaslanması

Framework	Ajax	MVC	i18n & i10n	ORM	Test Framework	Güvenlik	Caching	Doğrulama
Agavi	Evet	Evet	Evet	Evet	Evet	Evet	Evet	Evet
Akelos	Evet	Evet	Evet	Evet	Evet	Evet	Evet	Evet
CakePHP	Evet	Evet	Evet	Active Record	Evet	Evet	Evet	Evet
CodeIgniter	Hayır	Evet	Evet	3. parti	Evet	Hayır	Evet	Evet
Drupal	Evet	Evet		Çoklu	Evet	Evet		Form

					(basit)			API
eZ Components	Hayır	Hayır	Evet	Evet	Evet	Evet	Evet	Evet
FUSE	Evet	Evet		Evet	Evet (basit)	Evet	Evet	Evet
SilverStripe	Evet	Evet	Evet	Active Record	Evet	Evet	Evet	Evet
Kohana	Hayır	Evet	Evet	3. parti	Evet	Evet	Evet	Evet
Symfony	Evet	Evet	Evet	Propel, Doctrine	Evet	Evet	Evet	Evet
Zend	Evet	Evet	Evet	Evet	Evet	Evet	Evet	Evet

3. SYMFONY

Symfony Framework, yazılım geliştirme sürecini kullanılan örüntüleri otomatikleştirerek hızlandırır. Ayrıca bir framework yazılan kodu yapısallastırarak daha iyi, daha okunabilir ve daha düzeltilebilir bir halde yazılmasını saglar. En önemlisi, framework karmasık işlemleri basit ifadelere indirgeyerek programlamayı kolaylaştırır.

Symfony ağ yazılımı geliştirme surecini optimize etmek icin tasarlanmış bütünsel bir frameworktur. En başta web yaziliminin calisma kurallarini, sunucu mantigini ve gorunumu ayirir. Karmaşık web yazılım geliştirme sürecini kısaltacak araçlar ve sınıflar içerir. Tüm bunlara ek olarak, sık kullanılan görevleri otomatikleştirerek geliştiricinin tamamen yazılımın kendine has işlemlerine odaklanmasını sağlar. Tüm bu avantajlarin sonucunda, her projede tekrar tekerleği icat etmeye gerek kalmaz.

Symfony tamamen PHP 5(Suravski, 2004) ile geliştirilmis olup gerçek yaşamdan projelerle eksiksiz bir biçimde denenmiş, e-ticaret projelerinde son derece popülerdir. Şu an piyasada yaygın olarak kullanılmakta olan veritabanı sistemlerinin (MySQL, PostgreSQL, Oracle ve Microsoft SQL Server) buyuk bir çoğunluyla da uyumludur, Windows ve Güncel Linux dağıtımları üzerinde çalışabilmektedir.

3.1. SYMFONY'NİN ÖZELLİKLERİ

Symfony aşağıdaki gereksinimlere cevap verebilmek icin geliştirilmiştir.

- Kolay kurulum ve yapılandırma (standart bir windows ve unix,.. üzerinde çalışabilme garantisi)
- Veritabani platformundan bağımsızlık(Zaninetto, 2007)
- Coğu zaman sade bir kullanım ama karmasık bir problemi çözebilecek kadar da esnek
- Yapılandırması gelenekler üzerine kurulmuştur, geliştiricinin sadece alışılmadık durumlarda yapılandırmaya müdahale etmesi gerekir.

- Çoğu web teamulleri ve tasarim örüntüleriyle uyumludur.
- Kurumsal yapıya hazır, mevcut bilişim teknolojisi mimarilerine ve kurallarına uyum sağlayabilen ve uzun vadeli projeler için yeterince kararlı yapıdadır.
- Okunabilir kod, phpDocumentor yorumları ile kolay bakım.
- Diğer sağlayıcıların kütüphaneleriyle entegrasyonu sağlayan kolay genişletilebilirlik.
- Otomatikleştirilmiş Proje Özellikleri

3.2. SYMFONY'DE GERÇEKLEŞTİRİLEN PROJELERE OTOMATİK DÂHİL OLAN ÖZELLİKLER

Projelerin çoğu ortak özellikleri Symfony içerisinde aşağıdaki gibi otomatikleştirilmiştir:

- Yerleşik uluslararasılaştırma katmanıyla veri ve ara yüz çevirisine olanak sağlanır, buna ek olarak içerik yerelleştirmesi de mümkündür.
- Sunum, HTML tasarımcıları tarafından Symfony framework bilgisi gerektirmeksizin tasarlanabilen şablonlar ve görünümler kullanır. Ayrıca yardımcılar, kodun büyük bir kısmını sade fonksiyon çağırımlarına indirgeyerek sunum kodunu azaltırlar.
- Formlar otomatik doğrulama ve doldurmayı destekler, bu da veritabanında tutulan bilginin kalitesini arttırarak kullanımı iyileştirir.
- Çıktıdan kaçınma metotları verileri korur.
- Önbellek yönetim özelliğiyle bant genişliği ve sunucu yükünü azaltır.
- Doğrulama ve kimlik özelliğiyle kısıtlamalı bölümler ve kullanıcı güvenliği yönetimi sağlar.
- Yönlendirme ve akıllı URL özelliğiyle sayfa adreslerini kullanım ara yüzü ve arama motorlarıyla uyumlu hale getirir.
- Yerleşik e-mail ve API yönetimi web uygulamalarını klasik web tarayıcı kullanımının ötesine taşır.
- Otomatikleştirilmiş sayfalama, sıralama ve filtreleme sayesinde listeler daha kullanıcı dostu bir biçimde kullanılır.

- Factories, plug-ins ve mixinler da yüksek seviye bir genişletilebilirlik sağlar.
- Javascript efektlerinin farklı tarayıcılarda düzgün çalışmasını da sağlayan tek satırlık yardımcılar sayesinde Ajax etkileşimleri kolayca yürütülür.

3.3. UYGULAMA GELİŞTİRME ORTAMI VE ARAÇLAR

Kendi kodlama talimatları ve proje yönetim kuralları olan işletmelerin gerekliliklerini karşılamak için, Symfony tamamen özelleştirilebilir. Varsayılan olarak birçok geliştirme ortamı sunar ve yaygın yazılım mühendisliği görevlerini otomatikleştiren çok sayıda araçla donatılmıştır:

- Kod üretim araçları, prototiplendirme ve tek tıklamalı arka uç yönetimi açısından mükemmeldir.
- Yerleşik birim ve işlevsel test çerçevesi, teste dayalı geliştirmeye olanak tanımak açısından mükemmel araçlar sunar.
- Hata ayıklama paneli, geliştiricinin üzerinde çalıştığı sayfaya ilişkin ihtiyaç duyduğu tüm bilgileri görüntüleyerek, hata ayıklama sürecini hızlandırır.
- Komut satırı ara yüzü, iki sunucu arasındaki uygulama dağıtımını otomatikleştirir.
- Canlı yapılandırma değişiklikleri mümkün ve etkilidir.
- Günlükleme özellikleri, yöneticilere bir uygulamanın faaliyetleri hakkında tüm ayrıntıları verir.

3.4. MVC (MODEL – GÖRÜNÜM - DENETLEYİCİ)

Herkesçe bilinen Model/Görünüm/Denetleyici (ya da MVC) tasarım örgüsü, etkileşimli yazılım sistemleri tasarlamanın faydalı bir yoludur. Aynı zamanda Sunum/Soyutlama/Denetim (ya da PAC) tasarım örgüsü olarak da bilinen tasarım örgüsünün ana fikri, kullanıcı ara yüzlerini, kullanıcı ara yüzünün temsil ettiği temel verilerden ayırmaktır. "Klasik" MVC tasarım örgüsü(Ping Y.,2003) gerçekte bireysel tuş vuruşları ya da Mouse düğmelerinin etkinleşmesi gibi düşük seviyeli kullanıcı etkileşimi için geçerlidir. MVC' de, Görünüm, kullanıcıya bilgi gösterir ve kullanıcının etkileşimini işleyen Denetleyici ile birlikte, uygulamanın kullanıcı ara yüzünü oluşturur. Model, hem Görünüm'ün temsil ettiği bilgileri, hem de bu bilgiyi kullanıcı etkileşimine

yanıta dönüştüren mantığı içeren uygulama parçasıdır. PAC tasarım örgüsü benzer şekilde uygulamanın kullanıcı ara yüzünden gelen bilgilerini de ayrıştırır. Burada MVC'nin Görünüm ve Denetleyicisi, bir Sunum'da birleştirilir; uygulamanın verilerine Soyutlama bileşeni ve Denetim (ayrıştırılmış Sunum ve Soyutlama bileşenleri arasındaki iletişimden sorumlu bileşen) denir. MVC ve PAC tasarım örgüsünün kullanımı, bir uygulama geliştirmeyi ve sürdürmeyi kolaylaştırır; çünkü veri yapıları ve iş mantığı değiştirilmeden uygulamanın "görünümü" büyük ölçüde değiştirilebilir, uygulama çoklu diller ya da farklı kullanıcı izin grupları gibi farklı ara yüzleri kolayca koruyabilir. Konuşma dilinde, "MVC" terimi, bir uygulamanın Modelinde yapılan geniş ölçekli değişikliklerin sadece kullanıcı etkileşimlerini kabul edip işlemekten değil, aynı zamanda bir uygulamanın kullanıcının etkileşimi tarafından oluşturulan olaya yanıt durumunu değiştiren mantıktan sorumlu olan bir Denetleyici tarafından yönlendirilme şeklini tanımlayacak şekilde genişletilmiştir. Model'deki değişikliklere yanıt olarak, Denetleyici, uygulamanın yeni Görünümü'nün oluşturulmasına başlar. Bu tez, MVC terimini, bir uygulamayı iş mantığını, sunum mantığını ve istek işlemeyi bilinçli olarak ayıracak şekilde tasarlamanın kastedildiği bu daha geniş ve "PAC benzeri" anlamda kullanır.

Örüntü Web Uygulamaları ve MVC Tasarımı Web uygulamaları(Sacowicz, 2004), diğer etkileşimli yazılım sistemleri gibi, MVC tasarım örgüsüyle tasarlanabilir. Örneğin, bir "hepsi Java" yaklaşımı, Model olarak Entity Enterprise JavaBeans (EJB'ler) kullanır; Görünümü, HTML ve Java Server Pages aracılığıyla oluşturur ve Denetleyicileri, Servlets ve Session EJB'leri aracılığıyla uygular.

3.5. SYMFONY MVC

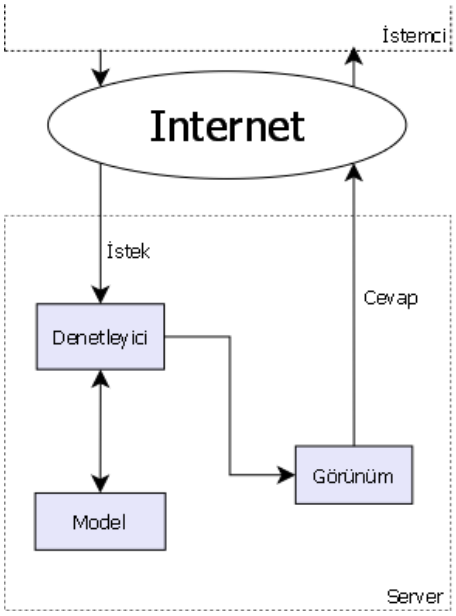
İlk bakışta, Symfony'ye dayalı bir uygulamanın ardındaki kod gayet korkutucu görünebilir. Bu kod birçok dizin ve komut dizisinden oluşur; dosyalar PHP sınıfları ve HTML karışımından oluşur, hatta ikisi birbirinin içine geçmiştir. Ayrıca aslında uygulama klasöründe hiçbir yerde bulunmayan sınıflara başvurular da görülecektir ve dizin derinliği altı seviyeye genişlemektedir. Ancak bütün bu görünen karmaşıklığın ardındaki mantığı anlaşıldığında, birden bire her şey Symfony uygulama yapısını başkasıyla değiştirilemeyecek kadar doğal gelecek.

3.5.1. Mvc Örüntüsü

Symfony, MVC yapısı olarak bilinen klasik web tasarım örüntüsünü temel alır. MVC yapısı üç seviyeden oluşur:

- Model, uygulamanın işleyişinin temelindeki bilgileri – işletme mantığını – temsil eder.
- Görünüm, modeli kullanıcıyla etkileşim için uygun bir web sayfasına dönüştürür.
- Denetleyici, kullanıcı işlemlerine karşılık verir ve model ya da görünümde gerekli değişiklikleri gerçekleştirir.

MVC mimarisi(Eiiti, 2004), işletme mantığını (model) ve sunumu (görünüm) ayırarak, daha büyük bir sürdürülebilirlik sağlar. Örneğin, uygulamanızın hem standart web tarayıcılarında, hem de el cihazlarında çalıştırılması gerekiyorsa, sadece yeni bir görünüme ihtiyacınız vardır; bu durumda ilk denetleyiciyi ve modeli koruyabilirsiniz. Denetleyici, istem için kullanılan protokolün ayrıntısını (HTTP, konsol kipi, posta, vs.) modelden ve görünümden saklamayı sağlar. Model de verilerin mantığını soyutlar ve böylece görünüm ve işlemin örneğin uygulamanın kullandığı veritabanı türünden bağımsız olması sağlanır.



Şekil 3.1 - Mvc tasarım örüntüsü

3.5.2. Mvc Katmanlaması

MVC'nin avantajlarını anlaşılmasına yardımcı olmak için, temel bir PHP uygulamasını nasıl MVC mimarili bir uygulamaya(Yu Ping, 2003) dönüştüreceğimizi görelim. Web günlüğü uygulamasına ilişkin bir posta listesi oldukça faydalı bir örnektir.

3.5.2.1. Düz Programlama

Düz bir PHP dosyasında, veritabanı girişleri listesinin görüntülenmesi, Kod 3.1'de sunulan komut dizisine benzeyebilir.

Kod 3.1 - Düz PHP betiği

```
<?php
// Veritabanına bağlanmak ve veritabanını seçmek
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);

// SQL sorgulamasını çalıştırmak
$result = mysql_query('SELECT date, title FROM post', $link);
?>
<html>
  <head>
    <title>List of Posts</title>
  </head>
  <body>
   <h1>List of Posts</h1>
   <table>
     <tr><th>Date</th><th>Title</th></tr>

<?php
// Sonuçları HTML olarak yazdırmak
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
echo "\t<tr>\n";
printf("\t\t<td> %s </td>\n", $row['date']);
printf("\t\t<td> %s </td>\n", $row['title']);
echo "\t</tr>\n";
}
?>

    </table>
  </body>
</html>
<?php
```

```
// Bağlantıyı kapatmak
mysql_close($link);
?>
```

Hızla yazılıp hemen uygulanabilir ama sürdürülmesi imkânsızdır. Bu kodun başlıca sorunları aşağıdadır:

- Hata kontrolü yoktur (veritabanına bağlantı başarısız olursa ne olacak?).
- HTML ve PHP kodu birbirine karışmış, hatta iç içe geçmiştir.
- Kod bir MySQL veritabanına bağlıdır.

3.5.3. Sunum Yalıtımı

Kod 1'deki echo ve printf çağrıları, kodun okunmasını zorlaştırır. Sunumu iyileştirmek için HTML kodunu değiştirmek, mevcut sözdizimiyle büyük bir zorluk yaratır. Bu nedenle kod iki bölüme ayrılabilir. Öncelikle, Kod 3.2'de gösterildiği gibi, tüm işletme mantığıyla birlikte saf PHP kodu bir denetleyici komut dizisine gider.

Kod 3.2 - Kontrol istemcisi kısımı

```
<?php
// Veritabanına bağlanmak ve veritabanını seçmek
$link = mysql_connect('localhost', 'myuser', 'mypassword');
mysql_select_db('blog_db', $link);
// SQL sorgulamasını çalıştırmak
$result = mysql_query('SELECT date, title FROM post', $link);
// Görüntü için dizilimi doldurma
$posts = array();
while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
   $posts[] = $row;
}
// Bağlantıyı kapatma
mysql_close($link);
// Görüntüyü isteme
require('view.php');
```

Şablon benzeri PHP sözdizimini içeren HTML kodu, Kod 3.2'de gösterildiği gibi, bir görünüm komut dizisine kaydedilir.

Kod 3.3 - Sunum Kısmı

```
<html>
  <head>
    <title>List of Posts</title>
  </head>
  <body>
    <h1>List of Posts</h1>
    <table>
      <tr><th>Date</th><th>Title</th></tr>
    <?php foreach ($posts as $post): ?>
      <tr>
        <td><?php echo $post['date'] ?></td>
        <td><?php echo $post['title'] ?></td>
      </tr>
    <?php endforeach; ?>
    </table>
  </body>
</html>
```

Görünümün yeterince temiz olup olmadığını belirlemenin temel kurallarından biri, PHP bilgisi olmayan bir HTML tasarımcısı tarafından anlaşılabilmesi için sadece asgari miktarda PHP kodu içermesi gerektiğidir. Görünümlerdeki en yaygın komutlar echo, if/endif, foreach/endforeach'tir ve hepsi budur. Ayrıca HTML etiketlerini tekrarlayan PHP kodları olmamalıdır.

Tüm mantık, denetleyici komut dizisine taşınır ve içinde HTML olmayan, saf PHP kodu içerir. Gerçekte aynı denetleyicinin belki PDF dosyası ya da XML yapısında olan tamamen farklı bir sunum için tekrar kullanılabileceği düşünülmelidir.

3.5.4. Veri İşleme Yalıtımı

Denetleyici komut dizisi kodunun çoğunluğu, veri işlemeye ayrılmıştır. Peki ya bir başka denetleyici için, örneğin web günlüğü postalarının RSS beslemesinin çıktısını alacak bir denetleyici için posta listesine ihtiyaç duyulduğunda ne olur? Ya kod kopyalanmasından kaçınmak için tüm veritabanı sorgularını tek yerde tutmak istenilirse? Eğer posta tablosu adının weblog_post olarak değişmesi için veri modelini değiştirmeye karar verilirse veya MySQL yerine PostgreSQL'e geçiş yapmak istenirse yapılması gerekenler vardır. Tüm bunları mümkün kılmak için, Kod 3.4'de gösterildiği

gibi, veri işleme kodunu denetleyiciden alıp model adı verilen bir başka komut dizisine koyulması gerekir.

Kod 3.4 - Model Katmanı

```
<?php
function getAllPosts()
{
  // Veritabanına bağlanmak ve veritabanını seçmek
  $link = mysql_connect('localhost', 'myuser', 'mypassword');
  mysql_select_db('blog_db', $link);
  // SQL sorgulaması
  $result = mysql_query('SELECT date, title FROM post', $link);
  // Dizilimi doldurmak
  $posts = array();
  while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
  {
    $posts[] = $row;
  }
  // Bağlantıyı kapatmak
  mysql_close($link);
  return $posts;
}
```

Kod 3.5 - Kontrol istemcisi Gözden Geçirilmiş Denetleyici Bölümü

```
<?php
// Modeli istemek
require_once('model.php');
// Postların listesini almak
$posts = getAllPosts();
// Gösterim katmanını istemek
require('view.php');
```

Denetleyicinin okunması kolaylaşmıştır. Tek görevi, modelden verileri almak ve görünüme geçirmektir. Daha karmaşık uygulamalarda, denetleyici ayrıca işlem, kullanıcı oturumu, kimlik doğrulama, vs ile de ilgilenir. Modelin işlevleri için açık adların kullanımı, denetleyicide kod yorumlarını bile gereksiz kılar.

Model komut dizisi, veri erişimine ayrılmıştır ve buna uygun olarak düzenlenebilir. Veri katmanına bağlı olmayan tüm parametrelere (istem parametreleri gibi) model tarafından doğrudan erişilmemesi, bunların denetleyici tarafından verilmesi gerekir. Model işlevleri başka bir denetleyicide kolaylıkla tekrar kullanılabilir.

3.5.5 Mvc'nin Ötesinde Katman Ayrımı

Kısacası, MVC mimarisinin prensibi, kodu doğasına göre üç katmana ayırmaktır. Veri mantığı kodu modele, sunum kodu görünüme ve uygulama mantığı denetleyiciye yerleştirilir.(Potencier F., 2007)

Diğer ek tasarım örüntüleri kodlama deneyimini daha da kolaylaştırabilir. Model, görünüm ve denetleyici katmanları tekrar alt bölümlere ayrılabilir.

3.5.5.1 Veritabanı Soyutlaması

Model katmanı, veri erişim katmanı ve veritabanı soyutlama katmanına bölünebilir. Bu şekilde, veri erişim işlevleri, veritabanına dayalı sorgu komutları kullanmayacak, sorguları kendileri yapacak olan bazı başka işlevleri çağıracaktır. Daha sonra veritabanı sistemi değiştirilirse, sadece veritabanı soyutlama katmanını güncellemeye ihtiyaç duyulacaktır.

Kod 3.6'da örnek bir veritabanı soyutlama katmanı ve ardından Kod 3.7'de de MySQL'e özel bir veri erişim katmanının örneği sunulmuştur.

Kod 3.6 - Model katmanında veritabanı soyutlama bölümü

```
<?php
function open_connection($host, $user, $password)
{
 return mysql_connect($host, $user, $password);
}
function close_connection($link)
{
 mysql_close($link);
}
function query_database($query, $database, $link)
{
 mysql_select_db($database, $link);
 return mysql_query($query, $link);
}
function fetch_results($result)
{
 return mysql_fetch_array($result, MYSQL_ASSOC);
}
```

Kod 3.7 - Model katmanının veri erişim kısmı

```
function getAllPosts()
{
  // Veritabanına bağlanmak
  $link = open_connection('localhost', 'myuser', 'mypassword');
  // SQL sorgulaması
  $result = query_database('SELECT date, title FROM post', 'blog_db', $link);
  // Dizilimi doldurmak
  $posts = array();
  while ($row = fetch_results($result))
  {
    $posts[] = $row;
  }
  // Bağlantıyı kapatmak
  close_connection($link);
  return $posts;
}
```

Veri erişim katmanında hiçbir veritabanı motoruna bağımlı işlev bulunmamasını sağlayabilirsiniz ve bu da bu katmanı veritabanından bağımsız kılar. Ayrıca veritabanı soyutlama katmanında oluşturulan işlevler, veritabanına erişim gerektiren birçok başka model işlevi için tekrar kullanılabilir.

Kod 3.6 ve 3.7'deki örnekler yine de çok yeterli değil ve tam bir veritabanı soyutlaması elde etmek için halledilmesi gereken birkaç iş daha yapılmalıdır. (veritabanından bağımsız bir sorgu oluşturucusu aracılığıyla SQL kodunu soyutlama, tüm işlevleri bir sınıfa taşıma, vs.)

3.5.5.2 Görünüm Unsurları

Görünüm katmanı da kod ayrımından faydalanabilir. Bir web sayfası genellikle bir uygulama boyunca tutarlı unsurlar içerir: sayfa başlıkları, grafik yerleşim, alt başlık ve genel gezinme. Sadece sayfanın iç bölümü değişir. Bu yüzden görünüm yerleşime ve şablona bölünür. Yerleşim genellikle uygulama ya da bir sayfa grubu için geneldir. Şablon sadece denetleyicinin temin ettiği değişkenleri şekle sokar. Bu bileşenlerin birlikte çalışması için bir mantık gerekir ve bu görünüm mantığı katmanı, görünüm adını alır. Bu prensiplere göre, Kod 3.3'ün görünüm bölümü, Kod 3.8, 3.9 ve 3.10'da gösterildiği gibi, üç bölüme ayrılabilir.

Kod 3.8 - Gösterim katmanının şablon kısmı

```
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php foreach ($posts as $post): ?>
  <tr>
    <td><?php echo $post['date'] ?></td>
    <td><?php echo $post['title'] ?></td>
  </tr>
<?php endforeach; ?>
</table>
```

Kod 3.9 - Gösterim katmanının gösterim mantığı kısmı

```
<?php
$title = 'List of Posts';
$content = include('mytemplate.php');
```

Kod 3.10 - Gösterim katmanının gösterim yerleşimi kısmı

```
<html>
  <head>
    <title><?php echo $title ?></title>
  </head>
  <body>
    <?php echo $content ?>
  </body>
</html>
```

3.5.5.3. Aksiyon ve Ön Denetleyici

Denetleyici önceki örnekte fazla bir şey yapmıyor; ancak gerçek web uygulamalarında, denetleyicinin çok işi vardır. Bu işin önemli bir bölümü, uygulamanın tüm denetleyicileri tarafından yapılır. Ortak görevler arasında istem işleme, güvenlik işleme, uygulama yapılandırması yükleme ve benzer angaryalar yer alır. Bu yüzden denetleyici genellikle tüm uygulama için tek olan bir ön denetleyici ve sadece tek bir sayfaya özgü denetleyici kodunu içeren işlemlere bölünür.

Ön denetleyicinin en büyük avantajlarından biri, tüm uygulamaya tek bir giriş noktası sunmasıdır. Uygulamaya erişimi kapatmak istenirse, tek yapılması gereken şey, ön denetleyici komut dizisini düzenlemek olacaktır. Ön denetleyicinin olmadığı bir uygulamada, her bir denetleyicinin kapatılması gerekecektir.

3.5.5.4. Nesne Yönlendirmesi

Önceki tüm örnekler yordamsal programlama kullanır. Modern dillerin OOP kabiliyetleri programlamayı daha da kolaylaştırır; çünkü nesneler mantığı içerebilir, birbirlerinden devralabilir ve temiz adlandırma kuralları temin edebilir.

MVC mimarisini nesne yönlendirmeli olmayan bir dile uygulamak, ad boşluğu ve kod kopyalama sorunlarını doğurur ve genel kodun okunması zordur.

Nesne yönlendirmesi, geliştirmecilerin görünüm nesnesi, denetleyici nesnesi ve model sınıfları gibi şeylerle ilgilenmesine ve önceki örneklerdeki tüm işlevleri yöntemlere dönüştürmesine olanak tanır. Bu, MVC mimarileri için bir zorunluluktur.

3.5.6. Symfony'nin Mvc Uygulaması

Bir web günlüğündeki postaları listeleyen tek bir sayfa için, kaç tane bileşen gerektiği Şekil 3.2'de gösterildiği gibidir.

- Model katmanı
    - Veritabanı soyutlaması
    - Veri erişimi
- Görünüm katmanı
    - Görünüm
    - Şablon
    - Yerleşim
- Denetleyici katmanı
    - Ön denetleyici
    - Aksiyon

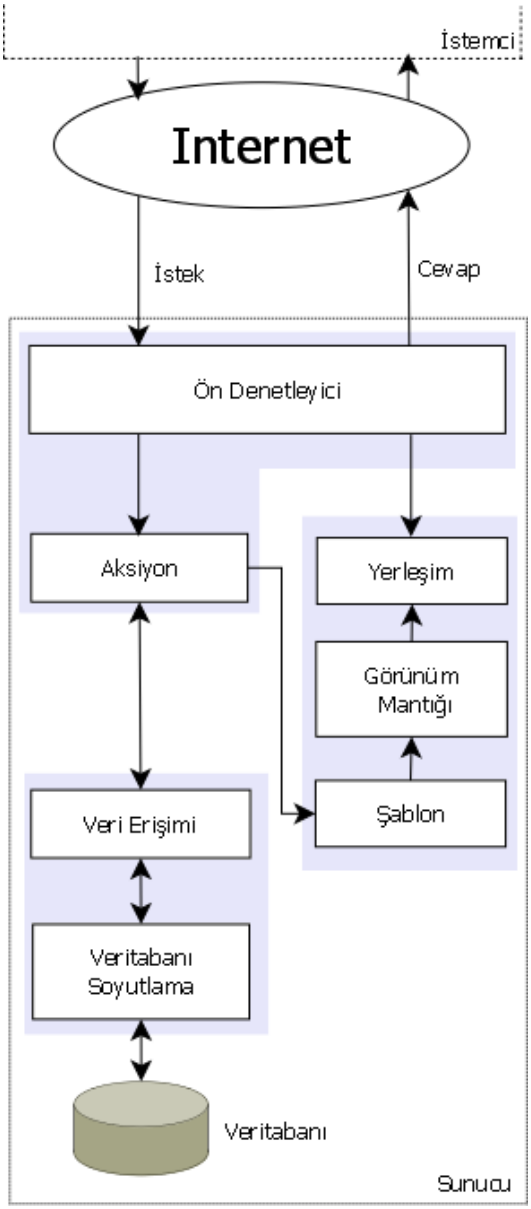
MVC mimarisinin en iyi yönlerini alan Symfony, alışıldığı takdirde uygulama gelişimini hızlı ve zahmetsiz kılacak bir şekilde uyguluyor.

Her şeyden önce, ön denetleyici ve yerleşim, bir uygulamadaki tüm aksiyon için geçerlidir. Çok sayıda denetleyici ve yerleşim olabilir; ancak sadece her birinden bir taneye ihtiyaç vardır. Ön denetleyici(Yonglei Tao, 2004), saf bir MVC mantık

bileşenidir ve asla tek bir tane bile yazılması gerekmez; çünkü Symfony onu uygulama geliştiriciler için üretecektir.

Ayrıca model katmanının sınıflarının da veri yapısına bağlı olarak otomatik olarak üretilmesidir. Bu, sınıf iskeletleri ve kod üretimi temin eden Propel kitaplığının işidir. Propel yabancı kilit kısıtlamalar ya da tarih alanları bulursa, veri işlemeyi çok kolaylaştıracak özel erişimci ve dönüştürücü yöntemleri temin edecektir. Ayrıca veritabanı soyutlaması uygulama geliştiriciler açınızdan görünmezdir; çünkü doğal olarak PHP Veri Nesneleri tarafından işlenir. Bu nedenle herhangi bir zamanda veritabanı motorunun değiştirilmesine karar verilirse, hiçbir kodun yeniden faktörlenmesi gerekmez. Sadece bir yapılandırma parametresinin değiştirilmesi yeterlidir.

Son olarak, görünüm mantığı, hiçbir programlama gerekmeden, kolayca basit bir yapılandırma dosyasına dönüştürülebilir.



Şekil 3.2 - Symfony veri akış şeması

Bu da örnekteki tanımlanan posta listesinin, Symfony'de, Kod 3.11, 3.12 ve 3.13'te gösterildiği gibi, sadece üç dosya üzerinde çalışılmasını gerektirdiği anlamına gelir.

Kod 3.11 - Listeleme aksiyonu - actions/actions.class.php

```
<?php
class weblogActions extends sfActions
{
  public function executeList()
  {
    $this->posts = PostPeer::doSelect(new Criteria());
```

```
  }
}
```

Kod 3.12 - Listeleme aksiyonunu gösterim katmanı kısmı - templates/listSuccess.php

```
<?php slot('title', 'List of Posts') ?>
<h1>List of Posts</h1>
<table>
<tr><th>Date</th><th>Title</th></tr>
<?php foreach ($posts as $post): ?>
  <tr>
    <td><?php echo $post->getDate() ?></td>
    <td><?php echo $post->getTitle() ?></td>
  </tr>
<?php endforeach; ?>
</table>
```

Ayrıca Kod 3.13'te gösterildiği gibi, yine de bir yerleşim tanımlanması gerekir; ancak tekrar tekrar kullanılması mümkündür.

Kod 3.13 - Ana yerleşim planı - templates/layout.php

```
<html>
  <head>
    <title><?php include_slot('title') ?></title>
  </head>
  <body>
    <?php echo $sf_content ?>
  </body>
</html>
```

Daha önce Kod 3.1'de gösterilen düz komut dizisiyle tamı tamına aynı sayfayı görüntülemek için gereken tam kod budur. Gerisi (tüm bileşenlerin birlikte çalışmasını sağlayarak) Symfony tarafından halledilir. Satırlar sayılırsa, Symfony ile MVC mimarisinde posta listesini oluşturmak için düz bir dosya yazmaktan daha fazla zamana ya da kodlamaya gerek olmadığı görülecektir. Bununla birlikte açık kod düzenlemesi, yeniden kullanılabilirlik, esneklik ve daha fazla eğlence gibi birçok muazzam avantaj sağlar. Ayrıca ek olarak, XHTML uyumu, hata ayıklama kabiliyetleri, kolay yapılandırma, veritabanı soyutlaması, akıllı URL yönlendirmesi, çoklu ortamlar ve daha birçok geliştirme aracına da sahip olunur.

3.5.6.1. Symfony'nin Ana Sınıfları

Symfony'deki MVC uygulaması birçok sınıf kullanır:

- sfController, denetleyici sınıfıdır. İstemin kodunu çözer ve onu işleme verir.
- sfRequest, tüm istem unsurlarını (parametreler, tanımlama bilgileri, başlıklar, vs) kaydeder.
- sfResponse, yanıt başlıklarını ve içeriklerini içerir. Bu, en sonunda bir HTML yanıtına dönüştürülerek kullanıcıya gönderilecek nesnedir.
- Bağlam (sfContext::getInstance() ile alınan), tüm çekirdek nesnelere ve güncel yapılandırmaya başvuruyu kaydeder; buraya her yerden erişilebilir.(Potencier, 2007)

Tüm Symfony sınıfları tıpkı Symfony'nin şablonlardaki çekirdek değişkenleri gibi sf öneki kullanıyor, Bu, sınıflarla ve değişkenlerle isim çakışmalarından kaçınılmasını ve çekirdek çerçeve sınıflarının sosyal olmasını ve kolayca tanınmasını sağlar.

4. AJAX UYGULAMALARINDA YAPILAN İYİLEŞTİRMELER

4.1. AJAX

(http://en.wikipedia.org/wiki/Ajax_(programming), 2008)

AJAX İnternet sayfalarında JavaScript ve XMLHttpRequest kullanımı ile etkileşimli uygulamalar yaratan tekniğin adıdır.

En yaygın kullanım alanı, sayfayı yeniden yüklemeye gerek kalmaksızın, sayfada görünür değişiklikler yapmaktır. XMLHttpRequest kullanılarak birden fazla bağımsız işlem yapılabilir. Bazı bilişim uzmanları, AJAX'ın HTML ve XML'den sonra en yenilikçi İnternet yazılımı olduğunu ve Web 2.0.'ı sonlandırıp, 3. evrenin kapısını açtığını öne sürmüşlerdir.

Asynchronous JavaScript and XML sözcüklerinin kısaltması olan **Ajax**, etkileşimli (interaktif) web uygulamaları yaratmak için kullanılan bir web programlama tekniğidir. Temel amacı arka planda sunucuyla ufak miktarda veri değişimi sayesinde sayfayı daha hızlı güncellenebilen web sayfaları yapmak, dolayısıyla kullanıcının istediği her anda bütün web sayfasını güncellemek derdinden kurtulmaktır. Bu da web sayfasının etkileşimini, hızını ve kullanılabilirliğini artırmak demektir.

Ajax tekniği aşağıdaki teknolojileri kullanır:

- Bilgiyi biçimlendirmek ve görüntüsünü değiştirmek için XHTML (ya da HTML) ve CSS.
- Görüntülenecek bilgiyi dinamik olarak göstermek ve onunla etkileşimli çalışmak için özellikle JavaScript ve JScript gibi ECMAScript olan kullanıcı tarafındaki bir scripting dili yardımıyla erişilebilen DOM.
- Web sunucusu ile bilgi değiş tokuşunda kullanılan XMLHttpRequest objesi. Bazı Ajax frameworklerde ve bazı durumlarda web sunucusuyla bilgi alışverişi için XMLHttpRequest yerine IFrame de kullanılır; diğer uygulamalarda ise dinamik olarak eklenen JavaScript TAG'leri de kullanılabilir.

- Kullanıcıyla sunucu arasındaki bilgi alışverişlerinde genelde XML formatı kullanılır. Önceden hazırlanmış HTML, düz metin, JSON ve hatta EBML dahi olmak üzere herhangi bir format da kullanılabilir. Bu tip dosyalar sunucu tarafında herhangi bir script (PHP, ASP vs.) ile bile üretilmiş olabilir.
- DHTML, LAMP ve SPA gibilerin aksine Ajax kendi başına bir teknoloji değildir, ancak terim birçok teknolojinin bir bütün olarak kullanıldığını ifade eder.

4.1.1. Ajax Kullanmanın Avantajları

4.1.1.1. Bant Genişliğinin Ayarlanması

HTML'yi ağ tarayıcısı (browser) programında oluştururken, JavaScript işlemlerini ve işlenecek verileri kullanırken, sunucudan gelen sayfa bilgisi gerektiğinden daha az yer tuttuğundan dolayı Ajax web sayfalarının görece olarak daha hızlı yüklendiği gözlenebilir. Bu tekniğe örnek olarak bilginin birden çok sayfadan oluştuğu büyük miktarda veriyi ele alalım. Ajax ile sayfanın HTML kodu, örneğin bir tablo kontrolü ya da ilgili TD, TR etiketleri (tag) ağ tarayıcısı (browser) içinde oluşturulabilir, dolayısıyla dökümanın ilk sayfasını indirmeye gerek kalmaz.

İçeriğin "isteğe bağlı olarak yüklenmesine" ek olarak, bazı web uygulamaları ilk önce Olay Yöneticilerini (event handler), ardından ilgili fonksiyonları indirir. Bu teknik, karmaşık mekanizması ve fonksiyonu olan web uygulamalarının gereksinim duyduğu yüksek miktarda bantgenişliği tüketimini önemli ölçüde hafifletir.

4.1.1.2. Kullanıcı Arabirimi

Ajax kullanmak için en önemli neden kullanıcı arabirimindeki iyileştirmedir. Ajax kullanan sayfalar tipik bir web sayfasından çok, tarayıcının çalıştığı bilgisayarda bir stand-alone uygulama gibidir. Sayfanın tamamının güncellenmesinin gerektiği linklere tıklandığı zaman işlemin epeyi "ağır (hantal)" olduğu hissedilir, ancak Ajax ile sayfa çoğu kez dinamik olarak güncellenebilir..

4.1.2. Ajax Kullanmanın Dezavantajları

Google ve benzeri site indeksleyen robotlar tarafından henüz ikinci plandadır ve en büyük dezavantajı budur. Ancak içeriğe uygun site haritaları ile içeriğin arama motorları tarafından indekslenmesi sağlanabilir. Bu konuda en iyi yöntem sitenin arama motorları için tasarlanmış ayrı bir sürümünü oluşturmaktır. Herhangi bir sunucu tarafı programlama dili ile senkronize edildiğinde genelde yazılması gereken kod miktarını neredeyse iki katına çıkarır. Bunun sebebi olarak ajax ile yazılmış fonksiyon ya da sınıfın talep cevap sistemi ile çalışması gösterilebilir. Ajax uygulamasının gönderdiği her bir talep için sunucu tarafında cevap verecek bir dosya olması gerekmektedir.

4.1.2.1. Tarayıcı Uyumluluğu

Dinamik olarak oluşturulan sayfalar tarayıcının önceden işlediği sayfaları yöneten mekanizmalarına kendini otomatik olarak kaydetmez, bu yüzden tarayıcının "Geri" tuşuna basıldığında istenen sayfa elde edilemeyebilir.

Geliştiriciler bu problemi çözmek için değişik çözümler geliştirmiştir. Bu tip çözümler genelde tarayıcının Geri tuşu kullanıldığında geçmiş listesiyle ilgili işlemleri çalıştıran görünmez IFRAME'leri kullanır. Örneğin Google Maps, arama suretiyle istenen bilgileri önce görünmez bir IFRAME içinde işler, sonra sayfanın görünen kısmında ilgili elemanın içine yerleştirir. World Wide Web Consortium (W3C) iframe elemanını XHTML 1.1 çalışmasına eklememiştir, bunun yerine object elemanını önermiştir.

4.1.2.2. Yanıt Verme Süresiyle İlgili Kısıtlamalar

Ajaxla geliştirme esnasında ağ gecikmesi ya da kullanıcı isteğine sunucunun verdiği yanıt arasında geçen süre özellikle dikkate alınmalıdır. Gecikme denen şey, kullanıcıya bilginin düzgün gönderilmemesi ya da XMLHttpRequest objesinin doğru bir şekilde kullanılmaması durumunda kullanıcının beklemediği ya da anlamadığı bir olaydır.

Buna ek olarak, bütün bir sayfa render edilirken sayfanın içeriği değişimi esnasında insan gözünün adaptasyonu için geçen belli bir süre vardır. Ekranın değişiminin daha ufak adımlarla ayarlanması yapılmazsa bu gecikme daha çok hissedilir. Kullanıcıya arka

planda bir işlemin yapıldığı ya da içeriğin yüklenmeye çalışıldığını söylemek için kullanılan görsel efektler bu tip gecikme problemlerine çözüm olarak önerilmektedir.

Genel olarak ağ gecikmesinin zamana bağlı olarak değişmesi gibi gecikmenin potansiyel etkileri, hâlihazırda bulunan açık kaynak Ajax araçlarının ve framework'lerin hiçbirince henüz çözümlenememiştir.

4.1.2.3. Arama Motorlarıyla İlgili İyileştirmeler

Arama motorlarınca endekslenmesi gereken bilgileri yüklemek için Ajax kullanılan web siteleri, Ajax işlevselliği için gerekli olan JavaScript kodunu arama motorlarının genelde çalıştırmamasından dolayı, arama motorlarının okuyabileceği bir formatta ve bir URL linki şeklinde bilgi koymalıdır. Bu problem Ajax'a özgü değildir, çünkü örneğin form gönderme sonrası gereken tam sayfa güncellemesi gibi dinamik sayfalar üreten sitelerde de bu problem vardır (bu gibi problemler genelde hidden web olarak adlandırılır).

4.2. SYMFONYDE AJAX KULLANIMI

Symfony'de ajax kullanımı kod yardımcı kütüphanelerinin kullanımı sonucu oldukça kolaylaşmıştır. İlerleyen bölümdeki örneklerde görüleceği üzere Symfony web uygulamalarının geleceği niteliğinde önem taşıyan Ajax uygulamalarının istikrarlı bir ortamda geliştirilmesine imkân sağlamaktadır. Örnekler aynı görevi gerçekleştiren iki Ajax uygulamasının direk html uygulanması ve Symfony kod yardımcı kütüphanelerinin kullanılması ile nasıl birkaç satırda yapılabildiğini göstermektedir.

4.2.1. Uzak form uygulaması:

Uzak form uygulamalarında kullanıcılardan alınan veri ansenkron bir XMLHTTP çağrısı ile veritabanında sorgulanır ve sayfayı tekrardan yüklemeye gerek kalmaksızın ilgili html elemanının içinde gösterir.

Kod 4.14 ve Kod 4.15'de remote olarak işlem yapan bir form uygulaması Xhtml formatında kodlanmıştır. Kod 4.16 ve Kod 4.17'de ise aynı işlem Symfony kod yardımcı kütüphanelerinin kullanılması ile Symfony sözdiziminde olarak kodlanmıştır.

Xhtml kodu:

Kod 4.14'de Prototype yazılım çerçevesi ve scipt.aculo.us kütüphanelerini yüklemek için gerekli kod yeralmaktadır.

Kod 4.14 - Prototype Javascript çerçevesini html e ilave etmek için gerekli kısım

```
<script type="text/javascript" src="/sf/prototype/js/prototype.js"></script>
<script type="text/javascript" src="/sf/prototype/js/effects.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/prototype.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/builder.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/effects.js"></script>
<script type="text/javascript" src="/sf/prototype/js/controls.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/scriptaculous.js"></script>
<script type="text/javascript" src="/sfModalBoxPlugin/js/modalbox.js"></script>
```

Kod 4.15'de ise kullanıcıdan gelecek reservasyon numarasına göre arama yapan uzak bir form uygulaması yeralmaktadır.

Kod 4.15 - Uzak form işlemi kodunun xhtml versiyonu

```
<form method="post" action="/operation/dosearchtickets" onsubmit="new
Ajax.Updater('operationreservationscontainer', '/operation/dosearchtickets',
{asynchronous:true, evalScripts:false, parameters:Form.serialize(this)}); return false;">
```

Symfony kodu:

Kod 4.16'de Prototype yazılım çerçevesi ve scipt.aculo.us kütüphanelerini otomatik olarak yüklemek için gerekli Javascript kod yardımcı kütüphaneleri çağırılaması gösterilmiştir.

Kod 4.16 - Symfonyde Ajax yardımcı kütüphanesi çağırılması

```
<?php use_helper('Javascript') ?>
```

Kod 4.17'de Kod 4.15'de yapılan bilet arama işleminin Symfony ile yapılmasıyla oluşan kod yeralmaktadır.

Kod 4.17 - Uzak form işlemi kodunun Symfony versiyonu

```
<?php echo
form_remote_tag(array('update'=>'operationreservationscontainer','url'=>'operation/dose
archtickets')); ?>
```

4.2.2. Uzak Link Uygulaması:

Uzak link uygulamalarında kullanıcı web sayfasındaki birlinke tıklar ve uzak bir ajax çağrısı başlatılır. Bu çağrı içerdiği işlemi uygulayarak yanıtı ilgili html elemanının içerisinde listeler.

Xhtml kodu:

Kod 4.18'de Prototype yazılım çerçevesi ve scipt.aculo.us kütüphanelerini yüklemek için gerekli kod yeralmaktadır.

Kod 4.18 - Prototype Javascript çerçevesini html e ilave etmek için gerekli kısım

```
<script type="text/javascript" src="/sf/prototype/js/prototype.js"></script>
<script type="text/javascript" src="/sf/prototype/js/effects.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/prototype.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/builder.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/effects.js"></script>
<script type="text/javascript" src="/sf/prototype/js/controls.js"></script>
<script type="text/javascript" src="/sfPrototypePlugin/js/scriptaculous.js"></script>
<script type="text/javascript" src="/sfModalBoxPlugin/js/modalbox.js"></script>
```

Kod 4.19'da rezervasyon numarasına göre bilet detayları gösteren bir ajax uygulaması yer almaktadır. Örnekte reservasyon numarası code parametresi olarak myreservation details aksiyonuna gönderilir. İşlemde aynı zamanda sonuçları gösterirken Script.aculo.us efekleri kullanılmıştır.

Kod 4.19 - Uzak link işlemi kodunun xhtml versiyonu

```
<a onclick="new Ajax.Updater(&#39;myreservationsdetail_1025008&#39;,
&#39;/reservation/myreservationdetails/code/1025008&#39;, {asynchronous:true,
evalScripts:false, onComplete:function(request, json){new
Effect.BlindDown(&#39;myreservationsdetail_1025008&#39;, {});new
Effect.Appear(&#39;myreservationsdetail_1025008&#39;, {});},
```

onLoading:function(request,
json){$('myreservationsdetail_1025008').innerHTML = '<img
border=\"0\" src=\"/images/bigloading.gif\"
alt=\"Bigloading\" />';
}});; return false;" href="#">1025008

Symfony kodu:

Kod 4.20'de Prototype yazılım çerçevesi ve Script.aculo.us kütüphanelerini otomatik olarak yüklemek için gerekli Javascript kod yardımcı kütüphaneleri çağırılaması gösterilmiştir.

Kod 4.20 - Symfonyde Ajax yardımcı kütüphanesi çağırılması

```
<?php use_helper('Javascript') ?>
```

Kod 4.21'de Kod 4.19'da yapıldığı gibi bilet detaylarını gösteren aynı uygulama link_to_remote kod yardımcısı kullanılarak yazılmıştır.

Kod 4.21- Uzak link işlemi kodunun Symfony versiyonu

```
<?php
   echo link_to_remote($divid ,array(
   'update' => 'myreservationsdetail_'.$userreservation->getReservationcode(),
   'url' => 'reservation/myreservationdetails?code='.$userreservation-
>getReservationcode(),
   'complete'=>
visual_effect('BlindDown','myreservationsdetail_'.$divid).visual_effect('Appear','myres
ervationsdetail_'.$divid),
   'loading'    => loadingImage('myreservationsdetail_'.$divid,'bigloading')
   ));
 ?>
```

Örnek uygulamalarda görülebileceği üzere Symfony kod yardımcı kütüphaneleri kullanılarak karmaşık ajax çağrılarının basite indirgendiği gözlemlenmiştir. link_remote, submit_to_remote, periodically_call_remote, link_to_function, observe_field metotları kullanılarak bir çok ajax uygulaması oldukça etkili birer fonksiyonla yapılabilmesi sağlanmıştır. İlgili metotların kullanımıyla kod yazımı, okunabilirliği ve taşınabilirliği Symfony kodunun sadeliği ile oldukça arttırılmıştır.

Kompleks ajax çağrılarını standartlaştırması ile Symfony zengin içerikli internet uygulamalarında yazılım geliştirme süreçlerini hızlandırmakta ve gelecekteki olası öne çıkabilecek yapısal uyuşmazlıkların önüne geçilmektedir.

5.ORM (NESNE İLİŞKİSEL EŞLEŞTİRME) VE SOYUTLAMA KATMANI

5.1. PROPEL

Veritabanları ilişkiseldir. Php 5 ve Symfony ise object oriented bir yapıya sahiptir. Object oriented context te bir veritabanına en verimli şekilde erişebilmek için obje mantığı ilişkisel mantığa çeviren bir ara yüz gereklidir. Bu ara yüze ORM denir ve dataya erişim veren ve iş kurallarını kendi içinde barındıran objelerden oluşur.

ORM'in esas faydası tekrar kullanılabilirliktir. Bir veri objesinin metodarının bir uygulamanın çeşitli bölümlerinden ve hatta farklı uygulamalardan çağrılmasına olanak tanır. ORM katmanı veri erişim mantığını kapsüller içerisinde uygular. Örneğin bir forum kullanıcısının yaptığı katkı sayısına ve bu katkıların popülaritesine göre reytinginin hesaplanması söz konusu olduğunda:

Bir web sayfasında böyle bir reyting gösterileceğinde hesapların detayı ile ilgilenilmeden veri modelinin bir metodu çağırılır. Eğer sonradan yapılacak bir hesaplama değişikliğinde model katmanındaki reyting metodunu değiştiririz. Böylece uygulamanın geri kalanına dokunulmamış olur.

Kayıtlara erişmek için objeler ve tablolar kullanmak yerine class kullanmanın bir faydası daha vardır. Bir tablodaki sütunla eşleşmeyen objelere yeni accessorlar eklemeye olanak tanırlar. Örnek olarak "ilk_isim" ve "son_isim" alanlarına sahip bir "müşteri" tablosunda sadece bir "isim" e gereksinim duyulması söz konusu olabilir. Object oriented yapılarında bu "müşteri" classına yeni bir accessor eklemek kadar kolaydır.

Kod 5.22 – İlave erişimci eklemek

```
public function getIsim()
{
  return $this->getIlkIsim().' '.$this->getSonIsim();
}
```

Uygulama açısından "müşteri" classında "ilk_isim", "son_isim", "isim" attributeleri arasında fark yoktur. Hangi attribute ların bir veritabanının sütunuyla ilişkili olduğunu sadece classın kendisi belirler.

Tekrar edilen veri erişim fonksiyonların ve verinin kendi iş mantığı bu gibi objelerde saklanabilir. Örnek olarak ürünleri olan bir "alışverişSepeti" classı verecek olursak toplamı hesaplamak için içinde sadece hesaplamayı yapan bir method yazabiliriz

Kod 5.23 - Toplam hesaplama fonksiyonu

```
public function getToplam()
{
  $toplam = 0;
  foreach ($this->getUrunler() as $urun)
  {
    $toplam += $urun->getFiyat() * $item->getMiktar();
  }

  return $toplam;
}
```

Veri erişim prosedürlerini yazarken bir başka önemli nokta göz önünde bulundurulmalıdır. Veritabanı üreticileri farklı SQL yazım seçenekleri kullanırlar. Başka bir DBMS sisteme kullanılması söz konusu olduğunda daha önce yazılmış bulunan SQL sorgularını baştan yazmak gerekir. Eğer sorgular veritabanından bağımsız bir şekilde yazılırsa ve SQL 'e üçüncü parti komponentlere çevrilirse veritabanı sistemi zahmetsizce değiştirilebilir. Bu veritabanının soyutlama katmanının amacıdır. Soyutlama katmanı uygulama geliştiricileri belli bir yapıda yazılmasına zorlar, DBMS ile uyum sorunlarını giderir ve SQL kodunu optimize eder.

Soyutlama katmanının esas faydası taşınabilirliği arttırmaktır. Çünkü proje gereksinimlerine bağlı veritabanı seçimi diğer sistemlerde oldukça teferruatlı bir iş iken soyutlama katmanı sayesinde bir satır değişimi ile DBMS seçimi mümkün olmaktadır.

Örnek olarak bir projeyi mysqlden oracle a taşımak istiyor isek veri tabanı erişim ayar dosyasındaki

propel.database = mysql

satırını

propel.database = oracle

Olarak tanımlamamız yeterlidir.

Symfonyde standart ORM seçimi Propel dir. Bunun yanı sıra Doctrine 'de kullanılabilmektedir. Propel soyutlama katmanı için Creoleyi kullanır. Propel tarafından geliştirilmiş bu iki üçüncü parti komponent Symfony' ye muntazam olarak entegre edilmiştir ve bunlar framework'ün bir parçası olarak düşünülebilir. Söz dizim ve uzlaşımları Symfony standartında uyarlanmıştır.

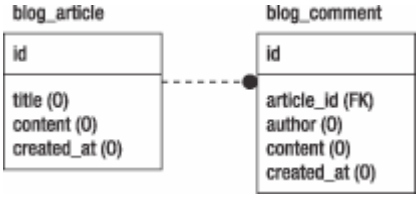
Bir Symfony projesinde bütün uygulamalar aynı modeli paylaşır. Ortak iş kurallarına bağlı uygulamaları tekrardan gruplama proje seviyesi kullanımının sebebidir. Modelin uygulamalardan bağımsız olma sebebi de bunun neticesinde olmaktadır. Symfony de bütün model dosyaları proje'nin root klasöründeki lib/model klasöründe saklanırlar.

5.1.1. Veritabanı Şeması

Symfony'nin kullanacağı veri nesne modeli yaratmak için veritabanının ilişkisel modelini object data modele çevrilmelidir. ORM in eşleştirme yapabilmesi için ilişkisel modelin tanımına ihtiyaç vardır ve buna şema denir. Bir şemada tablolar, ilişkiler ve sütunların özellikleri tanımlanır. Symfony'nin şema sözdizimi YAML formatındadır. schema.yml dosyası proje'nin root klasöründeki /config klasöründe bulunmalıdır.

Symfony YML kullanımının yanı sıra Propel Native XML şema formatını da anlar.

Makale ve yorumlardan oluşacak bir veri tabanının şema dosyası Kod 5.24 de yeralmaktadır. Şekil.5.3 de ise makaleler ve yorumlardan oluşacak bir veri tabanı için çizilmiş veri modeli yer olmaktadır.



Şekil 5.3 - Örnek veri modeli

Kod 5.24 - Örnek şema dosyası

```
propel:
  blog_article:
    _attributes: { phpName: Article }
    id:
    title:       varchar(255)
    content:     longvarchar
    created_at:
  blog_comment:
    _attributes: { phpName: Comment }
    id:
    article_id:
    author:      varchar(255)
    content:     longvarchar
    created_at:
```

Şema dosyasının içerisinde veritabanının ismi bulunmaz. Veritabanı bir bağlantı adının altında tanımlanmıştır (propel:) bunun sebebi bağlantı ayarlarının uygulamanın çalıştığı moda göre değişiklik gösterebilmesidir. Örnek olarak uygulama geliştirme modunda çalıştığında geliştirme veritabanına (ör. blog_dev) ulaşılır fakat üretim veritabanının şemasıyla aynı şema kullanılarak gerçekleşir. Bağlantı ayarları için gerekli databases.yml dosyası proje'nin root klasöründeki /config klasöründe bulunur. Şemada ayarlara detaylı bağlantı bulunmamasının sebebi veritabanlarına soyutlama sağlanması için sadece bağlantı adının bulunmasıdır.

5.1.1.1. Temel Şema Sözdizimi

schema.yml dosyasında ilk anahtar bağlantı adını temsil eder. Yaml sözdiziminde anahtarlar iki nokta üst üste bitirilir ve yapıda iç girinti vardır (bir ya da daha fazla boşluk ama tab değil) .

Bir tablonun phpName'i(yaratılacak classın adı) eğer attribute olarak tanımlanmamışsa Symfony bunu camelCase versiyonunu kullanır bu duruma örnek olarak yazılmalıdır.

Kod 5.25 - Öznitelik tanımlama

```
blog_comment:
  _attributes: { phpName: Comment }
```

blog_comment in phpName'i : Comment olarak tanımlanmamış olsaydı symfony bunu ismine bakarak ve camelCase olarak yani Blog_Comment olarak anlar ve bu isimle tanımlar.

Bir tabloda sütunlar vardır. Sütun değerleri üç şekilde tanımlanabilir.

1. Bir şey tanımlanmazsa Symfony sütun adlarını ve niteliklerini birkaç uzlaşıma göre tahmin eder. Örnek olarak id sütunlarının tanımlanmasına gerek yoktur Symfony otomatik artan sayı ve tablonun (PK)Birincil Anahtar olarak tanımlar. article_id: alanını ise phpName 'i Article olan tablonun id alanı ile ilişkilendirip foreign key olarak tanımlanır. created_at: alanı ise otomatik olarak timestamp türüne ayarlanır. Bu sütunlar için tür belirtmeye gerek yoktur schema.yml dosyasını kolay yazılabilir olmasının sebebi budur.

2. Eğer sadece bir öznitelik türü tanımlanmışsa sütün türü olarak algılanır. Symfony bilindik türleri algılar: boolean, integer, float, date, varchar(size), longvarchar gibi. Date ve timestamp türleri için Unix işletim sistemindeki sınırlamalar geçerlidir ve 1970-01-01 'den daha eski bir tarih olamaz. Doğum

günü gibi eski tarihler kullanılacaksa “before_unit” tarih formatı olan bu_date bu_timestamp kullanılır.

3. Başka sütun türleri tanımlanacaksa değer seti olarak ({ type: integer, foreignTable: cv, foreignReference: id, required: true, onDelete: cascade }) yazılmalıdır.

Tabloların belirtik yabancı anahtar ve indeksleri birkaç veritabanına özgü tariflerinin de oluşu schema.yml dosyası içerisinde tanımlanabilir.

5.1.2. Model Class’ları

ORM katmanı’nın model classları’nı yapmak için kullanılır. Bu classlar çalıştırma zamanından kazanmak için komut satırı komutuyla propel-build-model yaratır.

>Symfony propel-build-model

Modeli yarattıktan sonra symfony cc ile symfonyninin dâhili cache’isilinir böylece Symfony yeni yaratılan modelleri bulabilir.

Bu komuta şema analizlerini ve projenin lib/model/om/ klasöründeki base veri model classlarının yaratılmasını başlatır.

-Base Article.Php

-Base ArticlePer.Php

-Base Comment.Php

-Base CommentPeer.Php

Buna ek olarak lib/model/’de gerçek veri modeli sınıfları yaratılır.

-Article.Php

-ArticlePeer.Php

-Comment.Php

-CommentPeer.Php

5.1.2.1. Base ve Custom Classlar:

Veri object modelin 2 farklı klasörde tutulmasının sebebi:

Gerektiğinde model objelerine custom metotlar ve özellikler eklenmelidir. Fakat proje geliştikçe tablo ve sütunlar eklenecektir. schema.yml dosyası değiştirildiği zaman object model classlar propel-build-model çağırılarak yeniden yaratılmalıdır. Custom metotlar esas classların içinde ise her yaratılıştan sonra silinirler.

lib/model/om klasöründeki base classlar şemadan direk yaratılmış olanlardır. Model her yerinden yaratıldığında silinecekleri için hiçbir zaman modifiye edilmemelidir.

Öbür yandan, lib/model klasöründeki custom object classlar aslında base classlardan inherit eder. Var olan bir modelin üstüne propel-build-model çağırılığında bu classlar değiştirilmez. Bu yüzden custom metodlar buraya eklenmektedir.

Kod 5.26 - propel-build-model'in ilk kez çağırılması ile yaratılan bir custom model class örneği

```
class Article extends BaseArticle
{
}
```

Base Article class'ın bütün metotlarını inherit eder fakat şemadaki bir değişiklik onu etkilemeyecektir.

Base classları extend eden custom classların mekanizması veritabanın son ilişkisel modelini bilmeden kodlamaya başlama imkânı verir. İlişki(ilgi) dosya yapısı modeli hem değiştirilebilir hem de yenilikçi kılar.

5.1.2.2. Object ve Peer Classlar

Article ve Comment veritabanında bir kaydı temsil eden obje classlardır. Bir kaydın sütunlarına ve ilgili kayıtlara erişim sağlar. Bu demektir ki Article objesinin bir metodunu çağırarak bir makalenin başlığını öğrenebiliriz.(Kod 5.27)

Kod 5.27 - Nesnelerde metot çağırımı

```
$article = new Article();
...
$title = $article->getTitle();
```

ArticlePeer ve CommentPeer peer classlarıdır yani tablolar üstünde çalışarak statik metotları barındıran classlardır. Metotları genellikle ilgili obje classın bir objesini veya obje topluluğunu return eder.(Kod 5.28)

Kod 5.28 - retrieveByPks() fonksiyonu kullanımı

```
$articles = ArticlePeer::retrieveByPks(array(123, 124, 125));
// $articles Article classının bir nesne dizilimidir.
```

Not: data modeli açısından bakarsak peer obje olamaz.Bu yüzden peer classların metotları :: (statik metod call) ile çağırılır.(- (instance metod call) yerine)

Base ve Custom versiyonlarda obje ve peer classlar birleştirilince şemadaki her tablo başına 4 adet class yaratılır. Aslında lib/model/map klasöründeki runtime environment için gerekli olan tablonun metadata bilgisini barındıran bir class vardır. Bu class büyük ihtimalle hiç değiştirilemeyeceği için ihmal edilebilir.

5.1.3. Data Erişimi:

Symfony de verilere objelerle erişilir. İlişkisel modele ve SQL kullanarak veri almaya ve değiştirmeye alışık olan uygulama geliştiriciler için obje model metotları muhtemelen karmaşık görünecektir. Fakat object orientation'ın veri erişimindeki gücü standart metotlardan oldukça güçlüdür.

İlişkisel ve obje veri modeli benzer konulardır ama ikisinin kendine has adlandırma biçimleri vardır.

5.1.3.1. Sütun Değerlerini Elde Etmek

Symfony modeli yarattığı zaman schema.yml'de tanımlı tabloların her biri için bir base obje classı yaratır.Bu class –ların her biri için sütun tanımlarına dayanan default constructor,accessor ve mutatorlara sahiptir.new,get xxx( ) set xxx ( ) metotları obje yaratmaya yardımcı olurlar ve obje özelliklerine erişim sağlarlar(Kod 5.29)

Kod 5.29 - Nesne özelliklerine erişim

```
$article = new Article();
$article->setTitle('My first article');
$article->setContent('This is my very first article.\n Hope you enjoy it!');

$title   = $article->getTitle();
$content = $article->getContent();
```

Yaratılmış olan obje classı Article'dır. blog-article tablosuna verilmiş olan php Name'dir. Eğer phpName şemada tanımlanmış olmasaydı class Blog Article olurdu. Accessors ve mutators sütun adlarının camelCase halini kullanır. Yani getTitle ( ) metodu title sütununun değerini alır.

Birden fazla alanı bir kerede almak için her obje classı için yaratılan form Array ( ) metodu kullanılır.(Kod 5.30)

Kod 5.30 - formArray() metodu

```
$article->fromArray(array(
  'title'   => 'My first article',
  'content' => 'This is my very first article.\n Hope you enjoy it!',
));
```

5.1.3.2. İlişki Kayıtları Alma

Blog yorumları tablosundaki article_id sütunu blog_article tablosunun yabancı anhtarıdır. Her yorum bir makale ile ilişkilidir. Her makalenin birden fazla yorumu olabilir. Bu ilişki nesneye dayalı olarak çeviren 5 metot yaratılan classlardan kaynaklanmaktadır. getArticleId( ) ve setArticleId ( ) metotları article_id sütununu normal sütun olarak düşünebileceğimizi ve ilişkileri elle ayarlayabileceğimizi gösterir.

Nesneye dayalı yaklaşımın faydası diğer 3 metoda çok daha açıktır. Kod 5.31 yaratılmış setterların nasıl kullanılacağını gösterir. Aynı zamanda metot çağrılarının model objelere nasıl zincirleyeceğini gösterir.

Kod 5.31 - Nesne setterların kullanımı

```
$comment = new Comment();
$comment->setAuthor('Selim');
$comment->setContent('Merhaba dünya');

// Bu commenti önceki Article classına iliştir
$comment->setArticle($article);

// Alternatif söz dizimi
// eğer nesne veritabanında kayıtlıysa mantıklıdır
$comment->setArticleId($article->getId());
```

getArticle() metoda getTitle() accessoründen yararlanan Article classın bir objesini döndürür. Bu birleştirişi kendimizin yapmasından daha iyidir çünkü $comment–>getArticle( ) 'den başlayarak birkaç satır kod tutabilir.

Kod 5.32'deki $comments değişkeni. Comment classın objelerinin bir arrayını bulundurur. İlkini $comment [0] ile gösterebilirsiniz veya foreach ($comment as $comment) ile arttırabiliriz.

Kod 5.32 - İlişkisel yapılar

```
// Çoktan bire ilişki
echo $comment->getArticle()->getTitle();
 => My first article
echo $comment->getArticle()->getContent();
 => This is my very first article.
    Hope you enjoy it!

// Birden çoğa ilişki
$comments = $article->getComments();
```

Modelin objeleri tekil isimlerle tanımlanırlar. Blog_comment tablosundaki FK Comment obje isminin sonuna bir s ekleyerek getComments() metodunun yaratılmasına neden olur. Model objesine çoğul bir isim verilseydi metot adı get Commentss() olacaktı.

5.1.3.3. Datayı Kaydetmek Ve Silmek

New custom motorunu çağırarak yeni bir obje yarattık ama blog_article tablosuna gerçek bir kayıt yaratılmadı. Objeyi değiştirmenin veritabanına da bir etkisi olmaz. Datayı veritabanına kaydetmek için objenin save() metodu çağırılır.

Kod 5.33 - Kayıt işlemi

```
$Article->save ( )
```

ORM objeler arasındaki ilişkiyi algılayacak kadar akıllıdır ve $ Article objesini kaydetmek ilgili olan $ comment objesini de kayıt eder. Kayıt edilmiş objenin veri tabanında bir eşinin olup olmadığını da bilir. Böylece save ( ) metoduyla SQL 'de bazen INSERT bazen de UPDATE'dir. PK save() metoduyla otomatik olarak ayarlanır. Böylece kayıt edildikten sonra $article->get id( ) ile yeni PK alınabilir.

isnew() ile bir objenin yeni olup olmadığını kontrol edilebilir. Bir objenin değiştirilip kaydedilmesi gerektiğinin kontrolü için de isModified ( ) metodu çağırılır.

Yorumları delete() metoduyla silebiliriz.(Kod 5.34)

Kod 5.34 - Silme işlemi

```
foreach ($article->getComments() as $comment)
{
  $comment->delete();
}
```

delete ( ) metodu çağırıldıktan sonra bile istek sonlanana kadar obje mevcuttur.Objenin veritabanından silinip silinmediğini belirlemek için isDeleted( ) metodu çağırılır.

5.1.3.4. Kayıtları Pk Kullanarak Almak

Bir kayıtın birincil anahtarı biliniyorsa ilgili objeyi almak için peer classın retrieveByPk ( ) metodu kullanılır.

Kod 5.35 - retrieveByPk metodu

```
$article=ArticlePeer::retrieve ByPk (7);
```

Schema.yml dosyası id alanını blog-article tablosunun PK'sı olarak tanımlar.Böylece bu satır id'si 7 olan makaleyi döndürür.Sadece bir kayıtın döndürüleceği bilinir.$article değişkeni Article classın bir objesine sahip olur.

Bazı durumlarda PK birden fazla sütundan oluşabilir.Böyle durumlarda retrieveByPK( ) metodu her PK sütunu için bir olmak üzere çok parametre kabul eder.Parametre olarak PK array'i bekleyen retrieve ByPK( ) metodu çağırılarak birincil anahtarlarına göre çok obje seçilebilir.

5.1.3.5. Criteria() Fonksiyonu İle Kayıt Almak

Birden fazla kayıt elde edilmek isteniyor ise elde edilecek objelerin peer classının doSelect( ) metodu çağırılmalıdır. Örneğin Article classın objeleri alınacaksa Article Peer::doSelect( ) çağırılmalıdır.

doSelect( ) metodunun ilk parametresi Criteria classın bir objesidir Criteria veritabanı Soyutlama için SQL siz tanımlanmış bir basit sorgu tanımlama classıdır.

Boş Criteria classın bütün objelerini döndürür. Kod 5.36'deki kod bütün makaleleri alır.

Kod 5.36 - doSelect () metodu

```
$c = new Criteria();
$articles = ArticlePeer::doSelect($c);

// aşağıdaki SQL sorgulamasıyla aynıdır
SELECT blog_article.ID, blog_article.TITLE, blog_article.CONTENT,
     blog_article.CREATED_AT
FROM   blog_article;
```

HYDRATING:

doSelect() çağırısı aslında basit bir SQL komutundan çok daha güçlüdür. İlk olarak, SQL seçtiğiniz DBMS için optimize edilmiştir.İkinci olarak Criteria'ya gönderilen herhangi bir değer SQL koduna entegre edilmeden kaçılır ve SQL injection riskleri önlenir.Üçüncü olarak metot sonuç seti yerine obje array'i döndürür. ORM otomatik

olarak veritabanı sonuç setine dayanarak objeleri yaratır ve çoğaltır. Bu işleme Hydrating denir.

Daha karmaşık obje seçimi için WHERE, ORDER BY, GROUP BY ve diğer SQL komutlarının denklerine ihtiyaç vardır. Criteria objesinin bütün bu durumlar için metotları ve parametreleri vardır.

Add( ) metodlarına parametre olarak gönderilen class sabitleri özellik isimlerine, sütun isimlerinin büyük harfleridir.

blog_article tablosunun comment sütununa ArticlePeer::CONTENT sahibi ile değinilir.

Tablo 5.2 - SQL ile Criteria obje yazım biçimlerini karşılaştırır.

SQL	**Criteria**
WHERE column = value	->add(column, value);
WHERE column <> value	->add(column, value, Criteria::NOT_EQUAL);
Other Comparison Operators	
> , <	Criteria::GREATER_THAN, Criteria::LESS_THAN
>=, <=	Criteria::GREATER_EQUAL, Criteria::LESS_EQUAL
IS NULL, IS NOT NULL	Criteria::ISNULL, Criteria::ISNOTNULL
LIKE, ILIKE	Criteria::LIKE, Criteria::ILIKE
IN, NOT IN	Criteria::IN, Criteria::NOT_IN
Other SQL Keywords	
ORDER BY column ASC	->addAscendingOrderByColumn(column);
ORDER BY column DESC	->addDescendingOrderByColumn(column);
LIMIT limit	->setLimit(limit)
OFFSET offset	->setOffset(offset)
FROM table1, table2 WHERE table1.col1 = table2.col2	->addJoin(col1, col2)
FROM table1 LEFT JOIN table2 ON table1.col1 = table2.col2	->addJoin(col1, col2, Criteria::LEFT_JOIN)
FROM table1 RIGHT JOIN table2 ON table1.col1 = table2.col2	->addJoin(col1, col2, Criteria::RIGHT_JOIN)

Tablo 5.2.'de standart SQL sözdizimlerinin nasıl Criteria() kullanılarak yazıldığı gösterilmiştir.

Yaratılan classlarda hangi metotların bulunduğunu görmek ve anlamak için en iyi yol yarattıktan sonra lib/model/om klasöründeki Base dosyalarına bakmaktadır. Metot isimleri gayet açıktır ama daha fazla yorum görmek için config/propel.ini'deki propel.builder.addComments parametresi true yapılır ve model yeniden yapılır.

Kod 5.37 çok koşullu başka bir Criteria örneğidir. Selim tarafından iyi kelimesini içeren makalelerin bütün yorumlarını tarihe göre sıralayarak alır.

Kod 5.37 - Çok koşullu criteria örneği

```
$c = new Criteria();
$c->add(CommentPeer::AUTHOR, 'Selim');
$c->addJoin(CommentPeer::ARTICLE_ID, ArticlePeer::ID);
$c->add(ArticlePeer::CONTENT, '%iyi%', Criteria::LIKE);
$c->addAscendingOrderByColumn(CommentPeer::CREATED_AT);
$comments = CommentPeer::doSelect($c);

// Bu SQL sorgusunu yaratır

SELECT blog_comment.ID, blog_comment.ARTICLE_ID, blog_comment.AUTHOR,
    blog_comment.CONTENT, blog_comment.CREATED_AT
FROM   blog_comment, blog_article
WHERE  blog_comment.AUTHOR = 'Selim'
    AND blog_article.CONTENT LIKE '%iyi%'
    AND blog_comment.ARTICLE_ID = blog_article.ID
ORDER BY blog_comment.CREATED_AT ASC
```

SQL çok karmaşık sorgular yapabilen basit bir dildir. Criteria objesi de her karmaşıklık seviyesindeki koşulları halledebilir. Programcıların çoğu bir koşulu Nesneye dayalı programcılık mantığına çevirmeden önce SQL düşündüğünden ilk başta Criteria objesini anlamak zor olabilir.

Parametre olarak geçirilen kriteri sağlayan kayıt sayısını sayıp sayıyı integer olarak döndüren do Count( ) metodu doSelect( ) metoduna ilaveten her peer classta vardır.Bu durumda hiç obje döndürülmediği için hydrating olmaz ve doCount( ) metodu doSelect ( )'den daha hızlıdır.

Peer classları Criteria parametre olarak bekleyen doDelete( ), doInsert( ) ve doUpdate( ) metotlarını da sağlar. Bu metotlar veritabanına DELETE, INSERT ve UPDATE sorgularının yönetilmesine olanak tanır.

Son olarak ilk döndürülen obje isteniyor ise doSelect( ) yerine doSelectOne( ) kullanılır. Bir Criteria'nın sadece bir sonuç döndüreceğini biliyor ise kullanılır ve obje array'i yerine bir obje döndürdüğünden avantajlıdır.

5.1.3.6. Ham Sql Sorgularının Kullanımı

Bazen obje çoğaltmak istemeyebiliriz. Sadece veritabanı tarafından hesaplanmış sonuçları isteyebiliriz. Örneğin bütün makalelerin son yaratılma tarihleri elde etmek için bütün makaleleri çağırmak ve array üzerinde döngü yapmak mantıklı değildir. Bunun yerine veritabanından sadece sonucu döndürmesi istenir. Çünkü object hydrating işlemi atlanır.

Diğer taraftan veri tabanı yönetimi için olan PHP komutları direkt olarak çağırılmaz çünkü o zaman veritabanı soyutlamanın faydası yitirilir. Bu demektir ki ORM(propel) bypass edilmelidir ama veritabanı soyutlama(Creole) bypass edilmektedir.

Veritabanını Creole ile sorgulamak aşağıdakileri gerektirir:

1. Bir veritabanı bağlantısı sağlayın.
2. Bir query stringi yaratın.
3. Ondan bir statement yaratın.
4. İfadenin çalıştırılmasıyla elde edilen result seti 1 tur döndürün.

Kod 5.38 - Veritabanını Creole ile sorgulamak

```
$connection = Propel::getConnection();
$query = 'SELECT MAX(%s) AS max FROM %s';
$query = sprintf($query, ArticlePeer::CREATED_AT, ArticlePeer::TABLE_NAME);
$statement = $connection->prepareStatement($query);
$resultset = $statement->executeQuery();
$resultset->next();
$max = $resultset->getInt('max');
```

Bu işlem es geçilerek veritabanına direkt erişilirse Creole tarafından sağlanan güvenlik ve soyutlama kaybedilir. Creole yolu uzundur ama uygulamanın performansını, taşınabilirliğini ve güvenliğini garantiler. Özellikle internet kullanıcısı gibi güvenilir olmayan kaynaklardan gelen parametreleri barındıran sorgular için geçerlidir. Creole gerekli korunma hamlesini yapar ve veritabanını emniyete alır. Veritabanına direkt ulaşmak SQL-injection saldırıları için risk oluşturur.

5.1.3.7. Özel Tarih Sütunları Kullanma

Genellikle bir tablonun created_at sütunu kaydın yaratıldığı tarihi kaydetmek için kullanılır. Aynı şey updated_at sütunu için de geçerlidir. O da kayıt her yenilendiğindeki zamanı kaydeder.

Symfony bu sütunların adlarını tanır ve güncellemelerini otomatik yapar. Aynı şey created_on ve updated_on sütunları için de geçerlidir.

Kod 5.39 - Özel tarih sütunları kullanmak

```
$comment = new Comment();
$comment->setAuthor('Selim');
$comment->save();

// Yaratma tarihini göster
echo $comment->getCreatedAt();
  => [date of the database INSERT operation]
```

Ayrıca tarih sütunlarının getterları argument olarak bir tarih biçimini kabul ederler:

```
echo $comment->getCreatedAt('Y-m-d');
```

5.1.4.Veritabanı Bağlantıları

Data modeli kullanılan veritabanından bağımsızdır ama muhakkak bir veritabanı kullanılacaktır. Proje veritabanına istek göndermek için Symfony için gerekli asgari bilgi veritabanının ismi, erişim kodları ve veritabanının türüdür. Bu bağlantı ayarları config/ klasöründeki databases.yml dosyasına girilmelidir.

Kod 5.40 - Örnek databases.yml dosyası

```
prod:
  propel:
    param:
      hostspec:         mydataserver
      username:          myusername
      password:          xxxxxxxxxx

all:
  propel:
    class:              sfPropelDatabase
    param:
      phptype:          mysql     # Database vendor
      hostspec:         localhost
      database:         blog
      username:          login
      password:          passwd
      port:             80
      encoding:          utf8      # Default charset for table creation
      persistent:       true      # Use persistent connections
```

Bağlantı ayarları ortama bağlıdır. Uygulamanın prod, dev, test ve diğer ortamları için ayrı ayarlar tanımlanabilir. apps/myapp/config/databases.yml gibi uygulamaya özgü bir dosyanın içinde farklı değerler vererek bu konfigürasyon her uygulama için geçersiz kılınabilir. Bu yolla front-end ve back-end uygulamalar için farklı güvenlik kuralları tanımlanabilir ve veritabanında farklı haklara sahip pek çok kullanıcı tanımlanabilir.

Her ortam için çok bağlantı tanımlanabilir. Her bağlantı aynı isimle etiketlenmiş bir şema gösterir. Kod 5.41'deki propel bağlantısı Kod 5.24'deki Propel şemasını gösterir.

phptype parametresinin izin verilen değerleri Creole tarafından desteklenen veritabanı sistemlerinin değerleridir:

- mysql
- sqlserver
- pgsql
- sqlite
- oracle

hostspec, database, username ve password alışılmış veritabanı bağlantı ayarlarıdır. Veri kaynak adı (DSN) olarak daha kısa şekilde de yazılabilirler. Kod 5.41'in all: bölümüne denktir.

Kod 5.41 - DSN kullanımı örnek 1

```
all:
  propel:
    class:        sfPropelDatabase
    param:
      dsn:        mysql://login:passwd@localhost/blog
```

SQLite veritabanı kullanılıyorsa hostspec parametresi veritabanı dosyasının pathına eşitlenir. Eğer data/blog.db'de blog veritabanı saklanıyorsa databases.yml dosyası Kod 5.42'deki gibi olur.

Kod 5.42 - Dsn kullanımı örnek 2

```
all:
  propel:
    class:        sfPropelDatabase
      param:
        phptype:  sqlite
        database: %SF_DATA_DIR%/blog.db
```

5.1.4.1 Model Katmanını Genişletme

lib/model klasöründe yaratılmış olan boş model classlarına yeni metotlar eklenebilir. Geçerli objenin metotları $this ile ve geçerli classın statik metotları self:: ile çağrılır. Custom classlar metodları lib/model/om klasöründeki Base classlardan inherit eder.

Örneğin Kod 5.43'de bahsi geçen Article objesi için __toString() metodu eklenebilir böylece Article classının bir objesi yazdırılınca objenin başlığı Kod 5.43'deki gibi gösterilir.

Kod 5.43 - Modele yeni metotlar eklemek örnek 1

```
class Article extends BaseArticle
{
  public function __toString()
  {
    return $this->getTitle();  // getTitle() is inherited from BaseArticle
```

```
  }
}
```

Peer classlar da mesela yaratılma tarihine göre bütün makaleleri sıralayarak çağırmak için Kod 5.44'de gösterildiği gibi genişletilebilir.

Kod 5.44 - Modeli genişletmek örnek 2

```
class ArticlePeer extends BaseArticlePeer
{
  public static function getAllOrderedByDate()
  {
    $c = new Criteria();
    $c->addAscendingOrderByColumn(self::CREATED_AT);
    return self::doSelect($c);

  }
}
```

Yeni metodların yaratılmış olanlar gibi mevcut olduğu Kod 5.45'da gösterilir.

Kod 5.45 - Yeni metodların kullanımı

```
foreach (ArticlePeer::getAllOrderedByDate() as $article)
{
  echo $article;      // Sihirli __toString() methodunu çağırır.
}
```

5.1.4.2. Var Olan Metodları Geçersiz Kılma

Base classlardaki yaratılmış metotların bazıları gereksinimlere uymuyorsa custom classlarla geçersiz kılınabilirler. Sadece aynı argüman sayısı kullanıldığından emin olunmalıdır.

Örneğin $article->getComments() metodu belli bir sıralama yapmadan Comment objelerinin arrayini döndürür. Sonucu son yorum ilk gelecek şekilde yaratma tarihine göre sıralamak istersek Kod 46'deki gibi getComments() metodu geçersiz kılınır. lib/model/om/BaseArticle.php'de bulunan orijinal getComments() metodu bir criteria ve bir bağlantı değerlerini parametre olarak bekler. Bu yüzden fonksiyon da aynısını yapmalıdır.

Kod 5.46 - Var olan metodları geçersiz kılmak

```
public function getComments($criteria = null, $con = null)
{
 if (is_null($criteria))
 {
  $criteria = new Criteria();
 }
 else
 {
// PHP5 de nesneler göstergesiyle geçirilir. Orijinali değiştirmemek için
//klonlanmalıdır.
  $criteria = clone $criteria;
 }
 $criteria->addDescendingOrderByColumn(CommentPeer::CREATED_AT);

 return parent::getComments($criteria, $con);
}
```

Custom metot sonunda parent Base classın custom metodunu çağırır. Fakat bu tamamen es geçilip istenilen sonuç döndürülebilir.

5.1.4.3. Model Davranışları Kullanma

Bazı model değişiklikleri geneldir ve yeniden kullanılabilir. Model objesini sort eden ve concurrent obje kayıtları arasındaki anlaşmazlıkları önleyen iyimser bir kilit yapan metotlar pek çok classa eklenebilen genel genişletmelerdir.

Symfony bu genişletmeleri davranışlar olarak paketler. Davranışlar model classlara ilave metotlar sağlayan harici classlardır. Model classlarda kancalar bulunur ve Symfony bunları sfMixer yoluyla nasıl genişleteceğini bilir.

Model classlarda davranışları etkin hali getirmek için config/propel.ini dosyasındaki bir ayar değiştirilmelidir:

propel.builder.AddBehaviors = true //Varsayılan değer false

Symfony ile birlikte gelen behavior yoktur ama plug-inlerle yüklenebilirler. Bir behavior plug-ini yüklendiğinde behavior bir classa tek bir satırla atanabilir. Örneğin uygulamaya sfPropelPranoidBehaviorPlugin yüklenirse Article.class.php'ye aşağıdaki satırlar eklenerek Article classı bu behavior ile genişletilebilir.

```
sfPropelBehavior::add('Article', array(
  'paranoid' => array('column' => 'deleted_at')
));
```

Model yaratıldıktan sonra silinmiş Article objeleri veritabanında kalırlar ve behavior sfPropelParanoidBehavior::disable() ile geçici olarak hizmet dışı bırakılmadıkça ORM kullanan sorgulara görünmezler.

5.1.4.4 Genişletilmiş Şema Sözdizimi

Kod 43'de gösterildiği gibi bir schema.yml dosyası basit olabilir ama relational modeller genellikle karmaşıktır. Bu yüzden şemanın her durumu halledebilmek için zengin sözdizimi vardır.

5.1.4.5 Öznitelikler

Bağlantı ve tabloların belirli attributeları olabilir (Kod 47). _attributes ile ayarlanırlar.

Kod 5.47 - Özniteliklerin tanımı

```
propel:
  _attributes:   { noXsd: false, defaultIdMethod: none, package: lib.model }
  blog_article:
    _attributes: { phpName: Article }
```

Kod yaratılmadan önce şemanın doğrulanması gereklidir. Bunu yapmak için bağlantının noXSD özniteliği devre dışı bırakılır. Bağlantı defaultIdMethod attrıbutunu de destekler. Hiçbiri sağlanmamışsa veritabanının ID yaratan yerli metodu kullanılır. Örneğin MySQL için autoincrement veya PostgreSQL için sequences. Olası bir diğer değer de value'dur.

package attributu namespace gibidir. Yaratılan classların depolandığı pathı belirler. Varsayılan path lib/model/'dir ama model alt paketler halinde düzenlenecekse değiştirilebilir. Örneğin aynı klasördeki ana business classlar ve veritabanında depolanan bir istatistik motorunu tanımlayan classlar karıştırılmak istenmezse lib.model.business ve lib.model.stats paketleriyle iki şema tanımlanır.

Tabloyu eşleştiren yaratılmış classın adını set eden phpName tablo attributunu biliyoruz. Yerel içerikleri (uluslar arası hale getirmek için içeriğin farklı versiyonlarının ilgili tabloda bulunması) barındıran tablolar iki tane daha attribute alabilirler.(Kod 5.48)

Kod 5.48 - Özniteliklerde I18n kullanımı

```
propel:
  blog_article:
    _attributes: { isI18N: true, i18nTable: db_group_i18n }
```

5.1.4.6. Sütun Detayları

Temel sözdizimi iki seçenek verir: Symfonynin sütun özelliğini isminden çıkartmasına bir boş değer vererek izin vermek veya tipi herhangi bir tip anahtar kelimesiyle tanımlamak. Bu seçenekler Kod 5.49'da gösterilir.

Kod 5.49 - Sütun detayları tanımlamak örnek 1

```
propel:
  blog_article:
    id:              # Let symfony do the work
    title: varchar(50)  # Specify the type yourself
```

Bir sütun için çok daha fazlası tanımlanabilir. Eğer tanımlanacaksa Kod 50'de gösterildiği gibi sütun ayarları associative array olarak tanımlanmalıdır.

Kod 5.50 - Sütun detayları tanımlamak örnek 2

```
propel:
  blog_article:
    id:       { type: integer, required: true, primaryKey: true, autoIncrement: true }
    name:     { type: varchar(50), default: foobar, index: true }
    group_id: { type: integer, foreignTable: db_group, foreignReference: id, onDelete:
cascade }
```

Sütun parametreleri aşağıdaki gibidir:

type: Sütun türü. Seçenekler boolean, tinyint, smallint, integer, bigint, double, float, real, decimal, char, varchar(size), longvarchar, date, time, timestamp, bu_date, bu_bu_timestamp, blob ve clob'dur.

required: Boolean. Sütun gerekliyse true yapılır.

default: Varsayılan değer.

primaryKey: Boolean. Primary keyler için true'dur.

autoIncrement: Boolean. Otomatik artan değer alan tamsayı türündeki sütunlar için true'dur.

sequence: autoIncrement sütunları için sequences kullanan veritabanları için sequence adı (PostgreSQL ve Oracle)

index: Boolean. Basit index için true veya benzersiz bir index yaratılacaksa unique.

foreignTable: Başka bir tabloya foreign key yaratmaya yarayan tablo ismidir.

foreignReference: Foreign key foreignTable ile yaratılmışsa ilgili sütunun adıdır.

onDelete: Tabloda bir kayıt silindiğinde çalıştırılacak actionı belirler. setnull ise foreign key sütunu null olur. cascade ise kayıt silinir. Ayarlanan behaviorı veritabanı motoru desteklemiyorsa ORM taklit eder. Bu sadece foreignTable ve foreignReference ile ilgili sütunlarla alakalıdır.

isCulture: Boolean. Yerelleşmiş içerik tablolarının culture sütunları için true'ya ayarlanır.

5.1.4.7. Yabancı Anahtar Tanımlamaları

foreignTable ve foreignReference sütun attributelarına alternatif olarak bir tablodaki _foreignKeys: anahtarının altına foreign keyler eklenebilir. Kod 5.51'deki şema blog_user tablosundaki id sütunu ile eşleşen user_id sütununa foreign key yaratır.

Kod 5.51 -Yabancı anahtar tanımlamak örnek 1

```
propel:
  blog_article:
    id:
    title:   varchar(50)
```

```
  user_id: { type: integer }
  _foreignKeys:
    -
      foreignTable: blog_user
      onDelete:     cascade
      references:
        - { local: user_id, foreign: id }
```

Alternatif sözdizimi çok referanslı foreign keyler ve foreign keylere isim vermek için yararlıdır (Kod 5.52).

Kod 5.52 - Yabancı anahtar tanımlamak örnek 2

```
  _foreignKeys:
    my_foreign_key:
      foreignTable:  db_user
      onDelete:      cascade
      references:
        - { local: user_id, foreign: id }
        - { local: post_id, foreign: id }
```

5.1.4.8. Indeksler

index sütun attributune alternatif olarak bir tablodaki _indexes: anahtarının altına indexler eklenebilir. Benzersiz indeksler tanımlanacaksa _uniques: header'ı kullanılmalıdır. Kod 5.53 indeksler için alternatif sözdizimini gösterir.

Kod 5.53 - İndeksler için alternatif sözdizimi

```
propel:
  blog_article:
    id:
    title:          varchar(50)
    created_at:
    _indexes:
      my_index:       [title(10), user_id]
    _uniques:
      my_other_index: [created_at]
```

5.1.4.9. Boş Sütunlar

Symfony değeri olmayan bir sütunla karşılaştığında biraz sihir yapar ve kendine ait bir değer ekler. Boş sütunlara eklenen detaylar Kod 5.54'dedir.

Kod 5.54 - Boş sütunlar

```
// id isimli boş sütunlar birincil anahtar
id:         { type: integer, required: true, primaryKey: true, autoIncrement: true }

// XXX_id isimli boş sütunlar yabancı anahtar
foobar_id:  { type: integer, foreignTable: db_foobar, foreignReference: id }

// created_at, updated at, created_on ve updated_on isimli boş sütunlar
// tarihtirler ve otomatik olarak timestamp olurlar.
created_at: { type: timestamp }
updated_at: { type: timestamp }
```

Foreign keyler için Symfony sütun adının başıyla aynı phpName'e sahip bir tablo arar ve bulursa bu tablo adını foreignTable olarak alır.

5.2. DOCTRINE

Doctrine(Bauer, 2008) PHP 5.2.3+ için güçlü bir database abstraction layer (DBAL) üstünde bulunan bir object relational mapper'dır(ORM). Kilit özelliklerinden biri veritabanı sorgularını Doctrine Query Language (DQL) denen ve Hibernate(Zoe Bin-Gan, 2005) SQL'den esinlenen proprietary object oriented SQL dialectiyle yazma seçeneğidir. Bu geliştiricilere gereksiz kod çoğaltma gerektirmeden esnekliği koruyan SQL'e güçlü bir alternatif sağlar.

5.2.1 Doctrine Gelişiminin Sebebi

Object oriented programcılıkta veri yönetim görevleri skalar olmayan değerler halindeki objeleri değiştirerek uyarlanırlar. Örneğin, sıfır ya da daha fazla telefon numarası ve sıfır ya da daha fazla adresi olan bir kişiyi temsil eden bir adres defteri girişi object oriented olarak kişi objesi ve entry'nin verisini tutan kişinin adı, telefon numarası listesi ve adres listesi gibi alanlarla modellenir. Telefon numarası listesinin telefon numarası objeleri olabilir. Adres defteri girişi programcılık dili tarafından tek bir değer olarak değerlendirilir. Tercih edilen telefon numarası ve ev adresi gibi çeşitli metotlar objeyle ilişkilendirilebilir.

SQL DBMS gibi pek çok veritabanı ürünü sadece tablolarla düzenlenmiş integer ve stringler gibi skalar değerleri saklayabilir ve değiştirebilir.

Programcı obje değerlerini ya veritabanında depolamak için basit değer gruplarına çevirecek ve çağırdıktan sonra eski haline çevirecek veya program içerisinde skalar değerler kullanacak. İlk yaklaşım nesne ilişkisel eşleştirme ile yapılır.
Sorun bu objeleri veritabanında saklanacak ve kolay çağrılacak formlara objelerin özelliklerini ve ilişkilerini koruyarak çevirmektir. Bu objelere persistent denir.

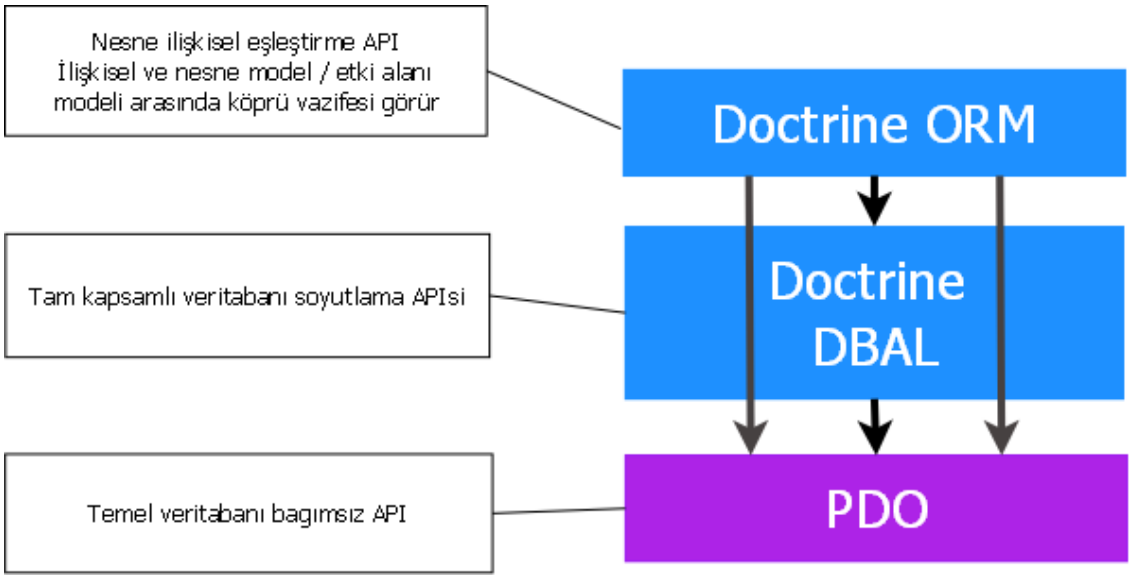
5.2.2. Minimum Gereklilikler

Doctrine PHP 5.2.3'den yukarısını gerektirir ama harici kütüphaneye gerek duymaz. Veritabanı fonksiyon çağrı abstraction için Doctrine http://www.php.net'den edinebilinen PHP resmi sürümüyle gelen PDO kullanır.

Windows'ta Uniform Server ve MAMP gibi bir 3 in 1 paketi veya herhangi bir resmi olmayan paket kullanılıyorsa ilave ayarlamalar gerekebilir.

5.2.3. Temel Bakış

Doctrine PHP' de Nesne İlişkisel Eşleştirme için bir alettir. PDO'nun en üstünde bulunur ve DBAL ve ORM diye iki ana katmana ayrılır. Aşağıdaki resim Doctrine katmanlarının nasıl beraber çalıştığını gösterir.



Şekil 5.4 - Doctrine ORM Modeli

DBAL(Veritabanı Soyutlama Katmanı) PDO tarafından sağlanan temel veritabanı soyutlama/bağımsızlığı tamamlar ve geliştirir. Eğer PDO üstünde güçlü bir veritabanı abstraction katmanı isteniyorsa DBAL kitaplığı kendi başına kullanılabilir. ORM katmanı DBAL'a bağlıdır ve bu yüzden ORM paketi yüklendiğinde DBAL da dâhil edilir.

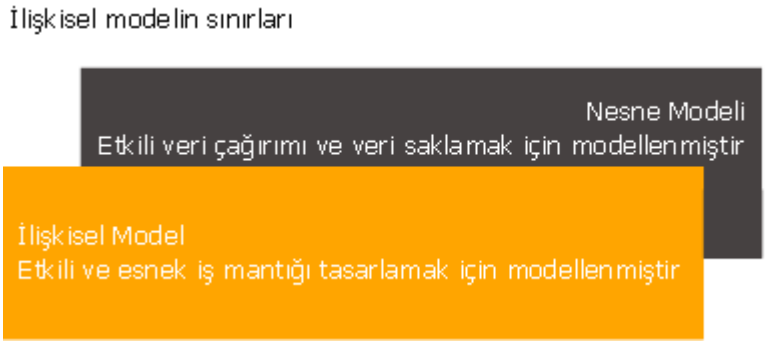
5.2.4. Doctrine

Doctrine ORM genel olarak Active Record, Data Mapper ve Meta Data Mapping patternleri etrafında yapılandırılmıştır.

Doctrine_Record(Bauer Jon, 2007) isimli bir temel class geliştirilerek bütün child classlar tipik ActiveRecord (kayıt, silme vs.) ara yüzünü elde eder ve Doctrine kolaylıkla kayıtların hayat döngülerine iştirak eder ve onları izler. Ama esas iş çoğunlukla Doctrine_Table classı gibi komponentlere gönderilir. Bu class createQuery(), find(id), findAll(), findBy*(), findOneBy*() vs. ile oluşan Data Mapper Arayüz'üne sahiptir. Böylece ActiveRecord temel classı Doctrine'in kayıtları yönetmesine izin verir ve mapping işi başka yerde yapılırken tipik ActiveRecord ara yüzünü sağlar.

ActiveRecord yaklaşımının kendine has sınırlamaları vardır. En açık olanı bir classın bir temel classı persistent olması için gelişimini uygulamasıdır. Genel olarak domain modelin dizaynı relational modelin dizaynıyla sınırlıdır. Bunun ancak bir istisnası vardır. İnheritance yapılarıyla uğraşırken Doctrine domain modelin relational modelden esnek kılınmasına izin veren gelişmiş mapping stratejileri sağlar ve bu yüzden biraz daha özgürlük sağlar.

Doctrine devam eden bir geliştirme sürecindedir ve her zaman daha özgür domain modellemeleri yapabilmek için yeni özellikler eklenmesine çalışılmaktadır. Fakat Doctrine daha çok bir ActiveRecord yaklaşımı olarak kaldıkça bu iki model arasında her zaman büyük benzerlikler olacaktır.



Şekil 5.5 - Nesne modeli ve ilişkisel model arasındaki ilişki

Resimde görüldüğü gibi domain modeli relational modelin sınırlarından fazla uzağa gidemez.

Bu sınırlamalardan bahsedildikten sonra ActiveRecord yaklaşımının avantajlarından bahsedilme zamanı gelmiştir. Daha basit programlama modelinin yanında relational ve OO domain modellerinin(Orenstein J., 2003) benzerliğinin bir de avantajı vardır: Var olan relational şemadan temel domain model yaratan güçlü yaratma aletlerini kolaylıkla sağlar. Dahası yukarıdaki nedenlerden ötürü domain modelin relational modelden fazla uzağa erişememesinden dolayı yaratma ve senkronizasyon araçları geliştirme süreci boyunca kolaylıkla kullanılabilir. Bu araçlar Doctrine'in artılarından biridir.
Web uygulamalarının çoğu için ActiveRecord yaklaşımının bu kısıtlamalarının çok sorun olmayacağı düşünülür çünkü iş domainler inin karmaşıklığı çoğu zaman orta derecededir. Aynı zamanda ActiveRecord yaklaşımının karmaşık iş mantığına kesinlikle uygun olmadığı itiraf edilir çünkü çok fazla kısıtlama getirir ve domain model üzerinde çok fazla etkisi vardır.

Doctrine basit veya orta halli karmaşık domain modellerinin sürerliğini sağlamak için iyi bir araçtır. Domain modeli daha fazla veritabanı merkezli yapmakla eşleştirmeyi kendimiz yapmamız arasındaki tradeoff göz önünde bulundurulduğunda karmaşık domain modeller için de Doctrine iyi bir araçtır.

5.2.5. Ana Konular

Doctrine Sorgulama Dili (DQL) (Bauer J., 2007) nesne sorgulama dilidir. Domain modelin terminolojisini (class isimleri, alan isimleri, classlar arasındaki ilişki vs.) kullanarak tam nesne grafikleri veya tek nesneler için sorgulama yapılmasına olanak tanır. Bu domain modelin (alan adları, class adları vs) relational modelden (tablo adları, class adları vs) ayrımını bozmadan nesneleri çağırmak veya manipüle etmek için güçlü bir araçtır. DQL SQL'e SQL bilen insanların kolayca algılaması için kasıtlı olarak çok benzer. Birkaç önemli fark vardır:
Aşağıdaki örnek DQL sorgulamasına bakınız:

FROM User u
LEFT JOIN u.Phonenumbers where u.level > 1

Bu sorgulamada fark edilmesi gerekenler:

- Tablolardan değil classlardan seçilir. User class/modelinden seçiyoruz.
- Associationlar join edilir (u.Phonenumbers)
- Alanlar reference edilir (u.level)
- Join condition yoktur (ON x.y=y.x). Classlar arasındaki birliktelikler ve bunların veritabanı tarafından nasıl belirtileceği Doctrine'ce bilinir.

DQL sorgulamaları domain model açısından ifade eder(classlar, Öznitelikleri, diğer classlarla ilişkileri vs.)
En düşük seviyede Doctrine veritabanı şemasını PHP classlarla temsil eder. Bu classlar şemayı ve modelin davranışını tanımlar.
Bir web uygulamasındaki kullanıcıyı gösteren basit bir model kod 5.55'deki gibidir.:

Kod 5.55 - Basit Doctrine model örneği

```
class User extends Doctrine_Record
{
   public function setTableDefinition()
   {
      $this->hasColumn('username', 'string', 255);
```

```
        $this->hasColumn('password', 'string', 255);
    }

    public function setUp()
    {
        $this->actAs('Timestampable');
    }
}
```

Her Doctrine_Record child classının setTableDefinition() ve setUp() metotları olabilir. setTableDefinition() metodu tablo şemasındaki sütun, indeks ve diğer bilgileri tanımlamak içindir. setUp() metodu Doctrine_Record child classlar arasındaki ilişkileri tanımlamak ve behaviorlar eklemek içindir.

5.2.5.1. Model Yaratma

Var olan veritabanlarından yaratmak kolayca başlamak için bir yoldur. Veritabanından classlar yaratıldıktan sonra ayarlamalar yapılmalıdır.

5.2.5.2. Mevcut Veritabanları

Doctrine gibi ORM araçlarında veritabanı ve ona erişen kod büyür ve karmaşıklaşır. Manuel SQL kodundan daha sağlam bir araca ihtiyaç vardır.
Doctrine mevcut veritabanından Doctrine_Record classları yaratmayı destekler. Domain model için Doctrine_Record classlarının elle yazılmasına gerek yoktur.

5.2.5.3. İlk Aktarma

Diyelim ki doctrine_test diye user isimli tek bir tablolu bir MySQL veritabanımız var. user tablosu aşağıdaki SQL ifadesi ile yaratılır.

Kod 5.56 - Kullanıcı tablosu yaratan SQL sorgusu

```
CREATE TABLE user (
  id bigint(20) NOT NULL auto_increment,
  first_name varchar(255) default NULL,
  last_name varchar(255) default NULL,
  username varchar(255) default NULL,
  password varchar(255) default NULL,
  type varchar(255) default NULL,
  is_active tinyint(1) default '1',
  is_super_admin tinyint(1) default '0',
```

```
  created_at TIMESTAMP,
  updated_at TIMESTAMP,
  PRIMARY KEY  (id)
) ENGINE=InnoDB
```

Şimdi bunu Doctrine_Record classa çevirelim.

İlk önce bootstrap.php dosyası sqlite yerine MySQL kullanacak şekilde değiştirilir:

Kod 5.57 - Bootstrap dosyası

```
// bootstrap.php

// ...
$conn =
Doctrine_Manager::connection('mysql://root:mys3cr3et@localhost/doctrine_test',
'doctrine');
// ...
```

$conn->createDatabase() metoduyla veritabanı eğer mevcut değilse ve bağlı kullanıcının veritabanı yaratma izni varsa yaratılır ve CREATE TABLE ifadesiyle tablo yaratılır.

Yaratılan classları depolamak için doctrine_test klasöründe models klasörü yaratılır.

```
$ mkdir doctrine_test/models
```

test.php'ye kod eklenerek model classlar yaratılır:

Kod 5.58 - model class yaratılışı

```
// test.php

// ...
Doctrine::generateModelsFromDb('models', array('doctrine'),
array('generateTableClasses' => true));
```

generateModelsFromDb metodu sadece bir parametre ister ve o da yaratılan dosyaların aktarılacağı klasördür. İkinci argüman veritabanı bağlantı isimlerinin arrayı ve üçüncüsü model yaratmada kullanılacak seçeneklerin arrayıdır.

Şimdi doctrine_test/models/generated klasöründe Baseuser.php diye aşağıdaki Kod 5.59'daki gibi bir dosya olmalıdır:

Kod 5.59 – yaratılmış baseuser.php dosyası

```
// models/generated/BaseUser.php

/**
 * Bu class Doctrine ORM Framework tarafından otomatik yaratılmıştır.
 */
abstract class BaseUser extends Doctrine_Record
{
  public function setTableDefinition()
  {
    $this->setTableName('user');
    $this->hasColumn('id', 'integer', 8, array('type' => 'integer', 'length' => 8, 'primary' =>
true, 'autoincrement' => true));
    $this->hasColumn('first_name', 'string', 255, array('type' => 'string', 'length' => 255));
    $this->hasColumn('last_name', 'string', 255, array('type' => 'string', 'length' => 255));
    $this->hasColumn('username', 'string', 255, array('type' => 'string', 'length' => 255));
    $this->hasColumn('password', 'string', 255, array('type' => 'string', 'length' => 255));
    $this->hasColumn('type', 'string', 255, array('type' => 'string', 'length' => 255));
    $this->hasColumn('is_active', 'integer', 1, array('type' => 'integer', 'length' => 1,
'default' => '1'));
    $this->hasColumn('is_super_admin', 'integer', 1, array('type' => 'integer', 'length' =>
1, 'default' => '0'));
    $this->hasColumn('created_at', 'timestamp', null, array('type' => 'timestamp', 'notnull'
=> true));
    $this->hasColumn('updated_at', 'timestamp', null, array('type' => 'timestamp', 'notnull'
=> true));
  }
}
```

Doctrine_test/models klasöründe User.php diye aşağıdaki gibi bir dosya olmalıdır.

Kod 5.60 - user.php

```
// models/User.php

/**
 * Bu class Doctrine ORM Framework tarafından otomatik yaratılmıştır.
 */
class User extends BaseUser
{

}
```

Doctrine generateTableClasses true değeri geçirdiği için doctrine_test/models/UserTable.php'deki model için Doctrine_Table iskelet classı otomatik olarak yaratır. Dosya Kod 5.61'daki gibidir:

Kod 5.61 - Otomatik yaratılan iskelet classı

```
// models/UserTable.php

/**
 * Bu class Doctrine ORM Framework tarafından otomatik yaratılmıştır.
 */
class UserTable extends Doctrine_Table
{

}
```

User ve UserTable classlarının içine modellerin işlevselliğini özelleştirmek için özel fonksiyonlar konulabilir.

Kod 5.62 - Örnek classa metot eklenişi

```
// models/User.php

// ...
class User extends BaseUser
{
   public function setPassword($password)
   {
      return $this->_set('password', md5($password));
   }
}
```

Yukarıdaki password accessor'unun doğru çalışabilmesi için bootstrap.php dosyasındaki auto_accessor_override attribute'u aşağıdaki gibi aktif hale getirilmelidir.

Kod 5.63 - Accessor çağırımı

```
// bootstrap.php

// ...
$manager->setAttribute(Doctrine::ATTR_AUTO_ACCESSOR_OVERRIDE, true);
```

Şimdi bir kullanıcıya şifre verildiğinde md5 şifreli olacaktır. İlk önce bootstrap.php models klasöründen modelleri kendi kendine yüklemek için bazı kodlar bulunduracak şekilde değiştirilmelidir:

Kod 5.64 - Değiştirilmiş bootstrap.php

```
// bootstrap.php

// ...
Doctrine::loadModels('models');
```

test.php File modeline yapılan değişiklikleri test edecek kodlar eklenerek değiştirilir:

Kod 5.65 - Değiştirilmiş test.php

```
// test.php

// ...

$user = new User();
$user->username = 'dogan';
$user->password = 'changeme';

echo $user->password; // outputs md5 hash and not change me
```

test.php çalıştırıldığında aşağıdakiler görülür:

```
$ php test.php
4cb9c8a8048fd02294477fcb1a41191a
```

Aşağıdaki örnek UserTable classa eklenebilecek bazı özel fonksiyonları göstermektedir:

Kod 5.66 - Class'a metod eklemek

```
// models/UserTable.php

// ...
class UserTable extends Doctrine_Table
{
   public function getCreatedToday()
   {
      $today = date('Y-m-d h:i:s', strtotime(date('Y-m-d')));
      return $this->createQuery('u')
```

```
            ->where('u.created_at > ?', $today)
            ->execute();
    }
}
```

Özel Doctrine_Table classlarının yüklenebilmesi için bootstrap.php dosyasındaki autoload_table_classes özniteliği aktif hale getirilmelidir.

Kod 5.67 - bootstrap.php

```
// boostrap.php

// ...
$manager->setAttribute(Doctrine::ATTR_AUTOLOAD_TABLE_CLASSES, true);
```

Böylece UserTable instance'ı ile çalışıldığında bu fonksiyona erişim izni vardır:

Kod 5.68 - Doctrine fonksiyon erişimi

```
// test.php

// ...
$usersCreatedToday = Doctrine::getTable('User')->getCreatedToday();
```

5.2.5.4. Şema Dosyaları

Modeller alternatif olarak YAML şema dosyalarıyla yaratılabilirler ve onlardan PHP classları yaratılabilir. Önce kolaylık açısından mevcut modellerden bir YAML şema dosyası yaratılır. test.php aşağıdaki gibi değiştirilir:

Kod 5.69 - değiştirilmiş test.php dosyası

```
// test.php
// ...
Doctrine::generateYamlFromModels('schema.yml', 'models');
```

test.php çalıştırılır:

```
$ php test.php
```

Şimdi doctrine_test klasöründe yaratılmış bir schema.yml dosyası görülür(Kod 5.70)

Kod 5.70 - Doctrine ile yaratılmış bir schema.yml dosyası

```
---
User:
  tableName: user
  columns:
    id:
      type: integer(8)
      primary: true
      autoincrement: true
    is_active:
      type: integer(1)
      default: '1'
    is_super_admin:
      type: integer(1)
      default: '0'
    created_at:
      type: timestamp(25)
      notnull: true
    updated_at:
      type: timestamp(25)
      notnull: true
    first_name: string(255)
    last_name: string(255)
    username: string(255)
    password: string(255)
    type: string(255)
```

Şimdi geçerli bir YAML şema dosyası olduğuna göre PHP classları yaratılabilir. Generate.php diye yeni bir script yaratılır. Bu script her şeyi yeniden yaratır ve çağrıldığı her seferde veritabanının tekrardan somutlanabilir olmasını sağlar:

Kod 5.71 - generate.php betiği

```
// generate.php

require_once('bootstrap.php');

Doctrine::dropDatabases();
Doctrine::createDatabases();
Doctrine::generateModelsFromYaml('schema.yml', 'models');
Doctrine::createTablesFromModels('models');
```

Schema.yml değiştirilir ve aşağıdaki komutla modeller yeniden yaratılır:

$ php generate.php

YAML şema dosya kurulumu olduğuna ve şema dosyalarından modeller yeniden yaratılabildiğine göre biraz temizlik yapılarak Doctrine'in gücü görülebilir:

Kod 5.72 - Doctrine temizlenmiş model

```
---
User:
  actAs: [Timestampable]
  columns:
    is_active:
      type: integer(1)
      default: '1'
    is_super_admin:
      type: integer(1)
      default: '0'
    first_name: string(255)
    last_name: string(255)
    username: string(255)
    password: string(255)
    type: string(255)
```

Yapılan bazı değişiklikler:

1) User olacağı varsayıldığından açık tableName tanımı kaldırılır.

2) Timestampable behavior eklenir.

3) Primary key tanımlanmamışsa otomatik olarak ekleneceğinden id sütunu kaldırılır.

4) Timestampable davranışıyla otomatik olarak ele alınabildiğinden updated_at ve created_at sütunları kaldırılır.

Şimdi YAML çok daha temizdir.

5.2.6 Dbms Fonksiyonları

5.2.6.1. Seçim Sorgulamaları

Öncelikle sorgulamalarda DBMS fonksiyonlarının nasıl kullanılacağı gösterilir. Diyelim ki yorumları olan her blog postunun kendisi ve yorum sayısı elde edilmek istendiğinde kullanılacak kod Kod 5.73 de gösterildiği gibidir.

Kod 5.73 - Doctrine seçim sorgusu

```
$q = Doctrine_Query::create()
  ->select('p.*, COUNT(c.id) as num_comments')
  ->from('BlogPost p')
  ->leftJoin('p.Comments c')
  ->groupBy('p.id');
$results = $q->execute();
echo $results[0]['num_comments'];
```

Fonksiyonlar herhangi bir kombinasyonda kullanılabilir ve istenilen derinlikte iç içe geçirilebilir.

5.2.6.2. Çoklu Birleştirmeler

Doctrine çok tablodan veri almayı kolaylaştırır. Bu örnekte bir kullanıcının sahip olduğu bütün izinler (bağlı olduğu gruplarca verilenler) bile elde edilir.

Kod 5.74 - Doctrine çoklu birleştirmeler I

```
$q = Doctrine_Query::create()
  ->from('User u')
  ->leftJoin('u.Permissions p')
  ->leftJoin('u.Groups g')
  ->leftJoin('g.Permissions p2')
  ->where('u.id = ?', 1);
$user = $q->fetchOne();
```

Şimdi kullanıcının sahip olduğu bütün izinlerin Doctrine_Collection'ı oluşturulur:

Kod 5.75 - Doctrine çoklu birleştirmeler II

```
$permissions = new Doctrine_Collection('Permission');
```

```
foreach ($user['Groups'] as $group)
{
  foreach ($group['Permissions'] as $permission)
  {
    $permissions[] = $permission;
  }
}
foreach ($user['Permissions'] as $permission)
{
  $permissions[] = $permission;
}
```

Bir blog uygulamasında tek bir sorgulamada bir blog postu ve ilişkili olduğu Author, Comments ve Tags'in elde edilmek istenmesi genel ihtiyaçtır. Doctrine'de bu çok kolaydır.

Kod 5.76 - Doctrine çoklu birleştirmeler III

```
$q = Doctrine_Query::create()
  ->from('BlogPost p')
  ->leftJoin('p.Author a')
  ->leftJoin('p.Comments c')
  ->leftJoin('p.Tags t')
  ->where('p.id = ?', 1);
```

5.2.6.3. Alt Sorgulamalar

Alternatif olarak aynı izin Doctrine_Collection'ı Doctrine'den direkt olarak alt sorgulamalarla alınabilir.

Kod 5.77 - Doctrine alt sorgu kullanımı

```
$userId = 1;
$q = Doctrine_Query::create()
  ->from('Permission p');

$q2 = $q->createSubquery()
  ->select('p2.permission_id')
  ->from('UserPermission p2')
  ->where('p2.user_id = ?');

$q3 = $q->createSubquery()
  ->select('p3.id')
```

```
  ->from('Permission p3')
  ->leftJoin('p3.GroupPermissions gp')
  ->leftJoin('gp.Group g')
  ->leftJoin('g.Users u')
  ->where('u.id = ?');

$q->where('p.id IN (' . $q2->getDql() . ')')
  ->orWhere('p.id IN (' . $q3->getDql() . ')');

$permissions = $q->execute(array($userId, $userId));
```

5.2.6.4. Kısa Sql Birleştirmeleri

Doctrine'in bir diğer kolaylığı joinleri bir tür kısa sözdizimiyle belirtebilmesidir. Bu sayede bir sorgulamadaki kod satır sayısı hatırı sayılır ölçüde azalacaktır. Joinler from() kısmında modellerden bahsedilerek leftJoin() kullanmayla aynı şekilde değiştirilebilirler.

Kod 5.78 - SQL birleştirmeleri I

```
$q = Doctrine_Query::create()
  ->from('User u, u.Profile p, u.Groups g');
```

Yukarıdaki kod aşağıdakiyle aynı şeyi yapar:

Kod 5.79 - SQL birleştirmeleri II

```
$q = Doctrine_Query::create()
  ->from('User u')
  ->leftJoin('u.Profile p')
  ->leftJoin('u.Groups g');
```

5.2.6.5. Silme Sorgulaması

Burada bir kullanıcı kullanıcı adı kullanılarak silinir.

Kod 5.80 - Doctrine silme fonksiyonu

```
Doctrine_Query::create()
  ->delete()
  ->from('User u')
```

```
  ->where('u.username = ?', 'jwage')
  ->execute();
```

5.2.6.6. Güncelleme Sorgulaması

Burada bir kullanıcının şifresi güncellenir.

Kod 5.81 - Doctrine güncelleme sorgusu

```
Doctrine_Query::create()
  ->update('User u')
  ->set('u.password', '?', 'newpassword')
  ->where('u.username = ?', dtukan)
  ->execute();
```

set() fonksiyonunun 3 argümanı vardır. İlki ayarlanacak alanın adı, ikincisi PDO' ya dokunulmadan aktarılan kısım ve üçüncüsü parametre/değerdir.

Bir diğer örnek DBMS fonksiyonlarıyla bir timestamp alanının ayarlanmasıdır. Üçüncü argüman NOW()'ın PDO' ya değişmeden geçmesi için kullanılmaz.(Kod 5.82)

Kod 5.82 - Zaman fonksiyonları kullanımı

```
Doctrine_Query::create()
  ->update('User u')
  ->set('u.last_login', 'NOW()')
  ->where('u.username = ?', 'dtukan')
  ->execute();
```

DQL'in silme ve güncelleme fonksiyonlarının faydası istenileni elde etmek için bir sorgulamaya gerek duyulmasıdır. Obje kullanılsaydı önce obje çağrılacaktı. Sonra güncellenecekti veya silinecekti yani iki ayrı sorgulama gerekecekti.

5.2.6.7. El İle Dql Yazmak

DQL sorgulamaları el ile de yazılabilir ve Doctrine_Query instance'ına parse edilir veya çalıştırılır.

```
$dql = "FROM User u, u.Phonenumbers p";
$q = Doctrine_Query::create()->parseQuery($dql);
```

Veya Doctrine_Query'nin query() metodu ile çalıştırılır.

```
$dql = "FROM User u, u.Phonenumbers p";
$q = Doctrine_Query::create()->query($dql);
```

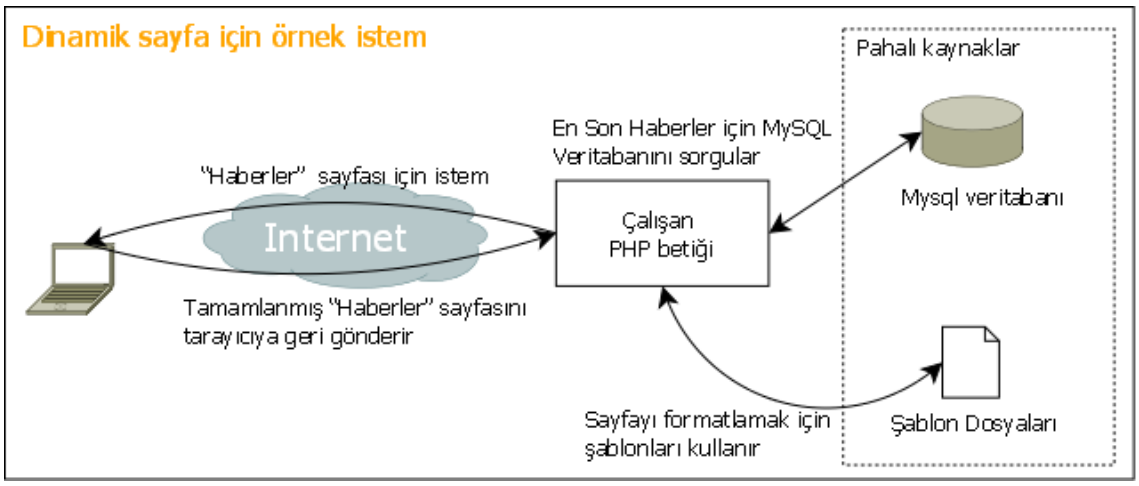
6. ÖNBELLEKLEME

6.1. ÖNBELLEKLEMEYE GİRİŞ

Önbellekleme uygulama geliştiricilere şu şekilde yardımcı olur: bir siteyi çalıştıran komut dizileri, sayfayı her seferinde tekrar oluşturmak yerine, sadece sorulduğu ilk sefer oluşturur, sonra tarayıcınıza gönderdiği sayfanın bir kopyasını saklar. Bir ziyaretçinin aynı sayfayı talep ettiği bir dahaki sefer, komut dizisi bu sayfayı yakın zamanda zaten oluşturduğunu bilir ve veritabanı sorgulamalarını ve aramaları tekrar çalıştırma külfeti olmadan, mevcut kopyayı tarayıcıya gönderir.

Örnek, bir web sitesindeki "Haberler" sayfası istemini gösteriyor; Haberler günlük olarak değiştiği için, bunu statik dosya olarak bulundurmak yerine bir veritabanında bulundurmak daha anlamlı; böylece kolayca güncellenip aranabilir. Haberler sayfası, aşağıdakileri yapan bir PHP komut dizisidir:

- Bir MySQL Veritabanına bağlanır
- En yeni 5 haberi talep eder
- Haberleri en yenisinden en eskisine sınıflandırır
- Bir şablon dosyasını okur ve içeriğin değişkenlerini yerine koyar
- Biten sayfayı kullanıcıya gönderir



Şekil 6.6 - Dinamik sayfa için örnek istem

Bu süreç ciddi bir zaman alır; saatte bir ya da iki ziyaretçi alıyorsanız, bunu önemsemeyebilirsiniz; ancak saatte 500 ziyaretçi alınıyorsa bu durum büyük bir fark yaratır.

Bu ve normal bir .html dosyasına yönelik düz bir istem arasındaki farkı ele alın. Web sunucusunun bir .html dosyasına hizmet vermek için çok çalışması gerekmez, sadece dosyayı bulur ve içeriğini tarayıcıya aktarır... önbellekleme kullanmak bu hız kazancını dinamik sitelerde bile elde etmenize olanak tanır.

Aynı örnekle devam edip önbelleklemenin devreye girdiği noktaya gelindiğinde, Haberler sayfasını talep edecek ilk kullanıcı komut dizisinin aynen yukarıdakini yapmasına ve ek olarak sonucu tarayıcının yanı sıra bir dosyaya yazdırarak yükün aslında artmasına yol açacaktır. Ancak sonraki istemler şekildeki gibi işleyecektir(Şekil 6.7)

Şekil 6.7 - Önbelleklenmiş Dinamik Sayfa için Örnek İstem

Görüldüğü gibi, MySQL veritabanı ve Şablonlara dokunulmuyor, web sunucusu sadece düz bir html dosyasının içeriğini tarayıcıya geri gönderiyor. İstem çok kısa sürede tamamlanıyor, kullanıcı sayfalarını daha hızlı alıyor ve sunucunun yükü daha az oluyor.

6.2. PHP'DE ÖNBELLEK UYGULAMA

Bunu yapmak için önbellek uygulamanın çeşitli yolları vardır; ancak en kolayı (en etkilisi olmasa bile), komut dizilerinizde fazladan bir PHP kod parçası kullanmaktır. Bu

örneğin büyük bölümü bu siteyi temel alır; ancak kolaylıkla herhangi bir siteye uygulanabilir.

Bu örneğin amacına yönelik olarak, web sitemin anlaşılmasına yardımcı olur. Temelde her sayfa konumunda (yani "site/önbellekleme") '/' işareti '.' ile değiştirilmiştir bu dosya (tüm içerikleri içerir) şablona dâhil edilmiştir (yani bu örnekte includes/design.caching). Mevcut dosya adı, $reqfilename adı verilen bir değişkenle sonuçlanır.

6.2.1 Çıktı Tamponu

PHP'nin son sürümlerinde uygulamaya sokulan Çıktı Tamponu, bunun için idealdir. Temelde programın başlangıcında ob_start() çağırılırsa, özel olarak çıktı tamponunun dökümünü yapana kadar tüm çıktıları bastırır. Böylece herhangi bir PHP komut dizisinin çıktısına kolayca ulaşılabilinir.

6.2.2. Basit Bir Önbellek

En temel ve nispeten faydasız önbellek uygulamasını inceleyek olursak:

Kod 6.83'deki kod parçacığı, home.html adı verilmiş bir dosyadaki "ana" sayfa çağrısının çıktısını kaydedecektir.

Kod 6.83 - Basit bir önbellek kullanımı

```
<?php
// çıktı arabelleğini başlat
ob_start(); ?>


<?php
$cachefile = "cache/home.html";
// yazmak için "cache/home.html" cache dosyasını aç
$fp = fopen($cachefile, 'w');
// çıktı arabelleğinin içeriğini bu dosyaya kaydet
fwrite($fp, ob_get_contents());
// dosyayı kapat
fclose($fp);
// çıktıyı gezgine gönder
```

```
ob_end_flush();
?>
```

Yöntem çok fazla işe yaramamaktadır; çünkü şu anda sahip olunan tek şey, her çalıştırıldığında "**cache/home.html**" adlı bir dosya oluşturan bir komut dizisidir. Ancak bir önbellek için iyi bir temeldir; PHP komut dizisinin oluşturduğu içeriği bir dosyaya kaydeder. Bir web tarayıcısında **cache/home.html**'i ziyaret edecek olunursa, sanki onu oluşturan komut dizisini ziyaret etmişçesine tamamen aynı sayfayı görülür, ancak kullanıcı nereye bakacağını bilmediği sürece bunun bir faydası yoktur.

6.2.3. Önbellek Dosyalarını Kullanma

İki tür istem vardır: '**MISS**' (**ISKALA**) ve '**HIT**' (**YAKALA**).

Bir kullanıcı, daha önce talep edilmemiş ya da zamanaşımına uğrayacak kadar uzun bir süre önce talep edilmiş bir sayfayı talep ederse, bu bir MISS (ISKALA) olarak değerlendirilir; bu durumda komut dizisinin sayfayı veritabanı (ya da diğer) kaynaklarından yeniden oluşturması ve yeni bir önbellek dosyası kaydetmesi gerekir.

Bir kullanıcı yakın zamanda talep edilmiş ve önbellekte olan bir sayfayı talep ederse, komut dizisi sadece bu dosyayı kullanıcıya aktarır ve başka bir şey yapmasına gerek yoktur. Bu HIT (YAKALA) olarak bilinir.

Bir sayfanın daha önce önbelleklenmiş olup olmadığını kontrol etmek kolaydır:

Kod 6.84 - Önbellekleme kontrolü

```
<?php
$cachefile = "cache/home.html";
if (file_exists($cachefile)) {
        include($cachefile);
        exit;
}
?>
```

Bu kodu komut dizinizin başlangıcına koymanız, mevcutsa önbelleğe alınan dosyayı kullanmasına ve ardından komut dizisinden çıkmasına yol açar (böylece gerisi asla çalıştırılmayacaktır). Asla değişmeyen bir site söz konusu olduğunda, bu yeterlidir; ancak asla değişmeyen çok az site vardır. Bu ufak parça ayrıca sadece günlük olarak

değişen bir uygulama varsa da yeterli olur; bu durumda her gün önbellek dizinini boşaltmak için cron kullanabilinir. Bu birçok site için uygun olmaz; süresiz olarak kullanılmaması için önbellekteki içeriği sona erdirmenin bir yolunu bulmamız gerekir.

6.2.4. Önbellek Verilerini Sona Erdirme

Bir önbellek dosyasının güncellenmesi gerekip gerekmediğini kontrol etmenin sayısız yolu vardır; bu bölümde en yaygın olan ikisi incelenecektir;

6.2.5. Basit Süre Sonu

Bu hiç şüphesiz birçok site için en iyi seçenektir; önbellek dosyalarına bir ömür verirsiniz – örneğin 5 dakika, 20 dakika, 1 saat – ve bu ömrün sonunda önbellek dosyaları sona erer ve sayfa yeniden oluşturulur. Aşağıdaki örnek, bunun nasıl işlediğini ve 2 saatlik bir süre sonu kullanıldığı takdirde değişikliklerin kullanıcı tarafından ne zaman görüleceğini gösterir. Günün ilk ziyareti 12.00'de gerçekleşir; geçerli bir önbellek yoktur, bu yüzden sayfa yeniden oluşturulur ve 14.00'e kadar geçerli kalır. Bu durumda veritabanı (ve dolayısıyla oluşturulan sayfanın içeriği) 13.20'de güncellenmiş olsa da o saat ile 14.00 arasında alınan herhangi bir istem, önbellek sona erdiğinde, eski bilgiler içerecektir. 14.00'deki bir sonraki istem nihayet tekrar veritabanı kaynaklarına başvuracaktır ve kullanıcı 13.20'de eklenen bilgileri görecektir.

Veritabanı ardından is 15.00'de tekrar güncellenir; ancak bu değişiklikler 16.00'dan sonra, yani yapıldıktan bir saat sonra görünecektir.

Bu yaklaşım çoğu site için uygun olsa da en güncel haber siteleri ya da içerikleri düzenli olarak değişen siteler için kesinlikle uygun değildir.

Bunu uygulamak için, sadece yukarıdaki if (file_exists($cachefile)) komutunu önbellek dosyasının değişiklik zamanının kontrolünü dâhil edecek şekilde genişletilmesi gerekir:

Şekil 6.8 - Örnek zamansal önbellek yönetimi

Kod 6.85 - 5 dakikalık bir önbellekleme

```
<?php
                    // 5 dakika
      $cachetime = 5 * 60;
      // $cachetime'dan azsa önbellekten gönder
      if (file_exists($cachefile) &&
        (time() - $cachetime < filemtime($cachefile)))
      {
        include($cachefile);
        echo "<!-- From cache generated ".date('H:i',
        filemtime($cachefile))."
        -->n";
        exit;
      }
?>
```

Bunu önceki kodla birleştirince, bir sayfanın çıktısını 5 dakika boyunca önbelleğe alacak temel bir yapı elde edilmiş olur:

Kod 6.86 - iki betiğin birleştirilmesi ile oluşan önbellekleme yapısı

```
<?php
    $cachefile = "cache/".$reqfilename.".html";
    $cachetime = 5 * 60; // 5 minutes
    // $cachetime'dan azsa önbellekten gönder
    if (file_exists($cachefile) && (time() - $cachetime
      < filemtime($cachefile)))
    {
      include($cachefile);
      echo "<!-- Cached ".date('jS F Y H:i', filemtime($cachefile))."
```

```
        -->n";
        exit;
    }
    ob_start(); // çıktı arabelleğini başlat
?>

<?php
    // yazmak için önbellek dosyasını aç
    $fp = fopen($cachefile, 'w');
    // çıktı arabelleğinin içeriğini kaydet
        fwrite($fp, ob_get_contents());
            // dosyayı kapat
    fclose($fp);
            // çıktıyı gezgine gönder
    ob_end_flush();
?>
```

6.2.6. Sadece Gerektiğinde Yeniden Oluşturma

Alternatif bir yöntemde veri kaynaklarının değiştirilip değiştirilmediği kontrol edilir ve bu da her istemin yükünü biraz artırır; çünkü DB tabanlı sitelerde bir veritabanı bağlantısı ya da potansiyel olarak birkaç dosyanın dosya değişiklik süresini sorgulamayı gerektirir ve ayrıca komut dizisini biraz daha karmaşıklaştırır. Ancak bu yöntem bir sayfaya eklemek üzere veri almak için yapılan gereksiz BÜYÜK sorgulamaları önler ve hiçbir şey değişmediğinde bile sayfaların düzenli olarak yeniden oluşturulmasını önler. Bu sitede kullanılan da bu yaklaşımdır.

Burada yapılması gereken tek şey if() tümceciğini değiştirmektir; örneğin:

Kod 6.87 - Önbelleğin yeniden oluşturulması

```
<?php
    $cachefile = "cache/".$reqfilename.".html";
    // (includes/$reqfilename) ile dahil edilen dosyanın en son değişikliği kadar
    // eski veya daha yeniyse önbellekten gönder
    if (file_exists($cachefile) && (filemtime("includes/".$reqfilename)
      < filemtime($cachefile))) {
      include($cachefile);
      echo "<!-- Cached ".date('H:i', filemtime($cachefile))."
      -->n";
      exit;
    }
            // çıktı arabelleğini başlat
    ob_start();
```

```
?>


..HTML aralığı...


<?php
      // önbellek dosyasını yazmak için aç
      $fp = fopen($cachefile, 'w');
                  // çıktı arabelleğinin içeriğini kaydet
      fwrite($fp, ob_get_contents());


                  // dosyayı kapat
      fclose($fp);
                  // çıktıyı gezgine gönder
      ob_end_flush();
?>
```

Bu, 'değiştirilen tarih' ya da benzeri bir şeyin sütununu içeren bir veritabanını sorgulamaya kolayca uyarlanabilir.

6.2.7. Önbelleklemenin Kullanılmaması Gereken Yerler

Önbellekleme bazı durumlarda kullanılmamalıdır ve bunun en bariz örneğini, içeriğin an be an güncel olması gereken ve kullanıcı girdisine göre değişen arama sonuçlarında, forumlarda, vs. görürüz. Ayrıca bu yöntemin "En Son Haberler" gibi durumlarda kullanılmaması da tavsiye edilir; genelde bu yöntemi nihai kullanıcının tarayıcısını ya da yetkilisini önbelleğe almak istemeyeceğiniz herhangi bir sayfada kullanılmamalıdır.

6.3. SYMFONY ÖNBELLEKLEME

Symfony, esnek bir sunucu önbellekleme sistemi sunar. YAML dosyalarını temel alan çok sezgisel bir kurulum aracılığıyla tüm yanıtı, bir işlemin sonucunu, bir parçayı ya da bir şablon bölümünü bir dosyaya kaydetme olanağı sunar. Temel veriler değiştiğinde, komut satırı ya da özel işlem yöntemleriyle önbelleğin seçici kısımlarını kolayca temizleyebilirsiniz. Symfony ayrıca HTTP 1.1 başlıkları aracılığıyla istemci taraflı önbelleği kontrol etmenin kolay bir yolunu da sunar.

6.3.1. Yanıtı Önbellekleme

HTML önbellekleme prensibi basittir: İstem üzerine bir kullanıcıya gönderilen HTML kodunun tümü ya da bir parçası benzer bir istemde tekrar kullanılabilir. Bu HTML kodu, ön denetleyicinin bir işlem yürütmeden önce bakacağı özel bir yerde (Symfony'de önbellek/klasör) saklanır. Eğer önbelleklenmiş bir sürüm bulunursa, işlem yürütülmeden bu sürüm gönderilir; böylece süreç büyük ölçüde hızlandırılır. Önbelleklenmiş bir sürüm bulunamazsa, işlem yürütülür ve sonucu (görünüm) daha sonraki istemler için önbellek klasöründe saklanır.

Bütün sayfalar dinamik bilgi içerebileceğinden, HTML önbelleği varsayılan olarak devre dışı bırakılmıştır. Performansı artırmak için HTML önbelleğini etkinleştirmek site yöneticisine bağlıdır.

Symfony'de üç farklı HTML önbellekleme türü işlenir:

- İşlem önbelleği (yerleşimsiz ya da yerleşimli)
- Parça, bileşen ya da bileşen yuvası önbelleği
- Şablon bölümü önbelleği

İlk iki tür, YAML yapılandırma dosyalarıyla işlenir. Şablon bölümü önbelleği, şablondaki yardımcı işlevlerine çağrılarla yönetilir.

6.3.2. Genel Önbellek Ayarları

Bir projenin her uygulaması için, HTML önbellek mekanizması, `settings.yml` dosyasının önbellek ayarında ortama göre etkinleştirilebilir ya da devre dışı bırakılabilir (varsayılan). Kod 6.88 önbelleğin etkinleştirilmesini gösterir.

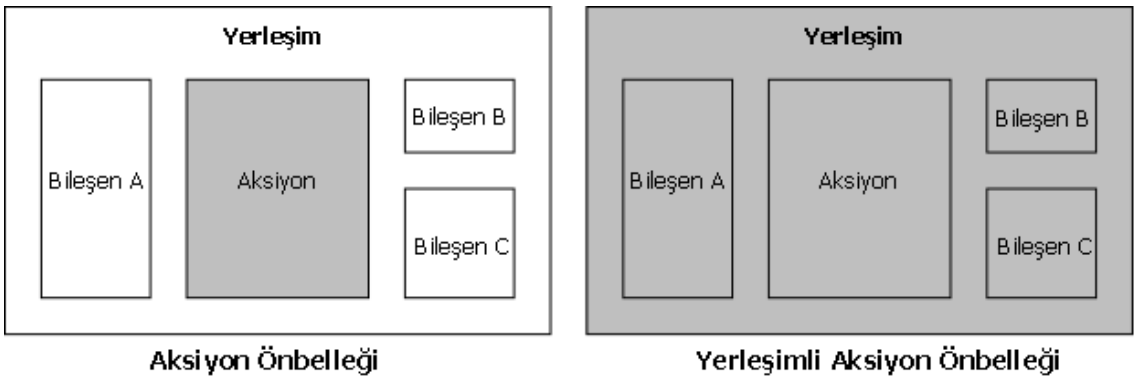
Kod 6.88 - Symfonyde önbellek kullanımının etkinleştirilmesi /config/settings.yml'de

```
dev:
  .settings:
    cache:              on
```

6.3.3. İşlemi Önbellekleme

Statik bilgi görüntüleyen işlemler (veritabanına ya da oturuma dayalı verilere bağlı olmadan) ya da bir veritabanından bilgileri değiştirmeden okuyan işlemler (tipik olarak GET (AL) istemleri) genellikle önbellekleme için idealdir. Şekil 9, bu durumda sayfanın hangi öğelerinin önbelleklendiğini gösterir: ya işlem sonucu (şablonu) ya da yerleşimle birlikte işlem sonucu.

Şekil 6.9 - İşlemi önbellekleme



Örneğin, bir web sitesinin tüm kullanıcılarının listesini geri gönderen bir user/list işlemini ele alalım. Bir kullanıcı değiştirilmediği, eklenmediği ya da çıkarılmadığı (ve bu konu daha sonra "Önbellekten Öğeleri Çıkarma" bölümünde ele alınacak) sürece, bu liste daima aynı bilgileri görüntüler; dolayısıyla önbellekleme için iyi bir adaydır.

Önbellek etkinleştirmesi ve ayarları işlemden işleme config/ modülü dizininde bulunan bir cache.yml dosyasında tanımlanmıştır. Örnek için Kod 6.89'ye bakılırsa:

Kod 6.89 - Bir işlem için önbelleği etkinleştirme

```
list:
  enabled:     on
  with_layout: false   # Default value
  lifetime:    86400   # Default value
```

Bu yapılandırma, önbelleğin liste işlemi için devrede olmasını ve yerleşimin işlemle birlikte önbelleğe alınmamasını (bu, varsayılan davranıştır) öngörür. Bu da işlemin önbelleklenmiş bir sürümü olsa bile yerleşimin (parçaları ve bileşenleriyle birlikte)

halen çalıştırıldığı anlamına gelir. Eğer with_layout ayarı true olarak yapılmışsa, yerleşim işlemle birlikte önbelleğe alınır ve tekrar çalıştırılmaz.

Önbellek ayarlarını test etmek için, tarayıcınızdan geliştirme ortamındaki işlemi çağırın.

http://myapp.example.com/myapp_dev.php/user/list

Sayfadaki işlem alanının etrafında bir kenarlık fark edilecektir. İlk seferinde alanın mavi bir kenarlığı vardır ve bu önbellekten gelmediğini gösterir. Sayfa yenilediğinde işlem alanının sarı bir kenarlığı olacaktır ve bu önbellekten geldiğini (yanıt süresinde fark edilir bir artışla) gösterir.

Yuvalar şablonun parçasıdır ve bir işlem önbelleklendiğinde aynı zamanda işlemin şablonunda tanımlanan yuvaların değeri de kaydedilir. Bu nedenle önbellek yuvalar için doğal olarak çalışır.

Önbellekleme sistemi ayrıca bağımsız değişkenli sayfalar için de çalışır. Kullanıcı modülünün örneğin bir kullanıcının ayrıntılarını görüntülemek için bir id bağımsız değişkenini bekleyen bir Show aksiyonu olabilir. Önbelleği bu aksiyon için de etkinleştirmek amacıyla, Kod 6.90'da gösterildiği gibi, cache.yml dosyası değiştirilmelidir.

cache.yml'nizi düzenlemek için, yine Kod 6.90'da gösterildiği gibi, bir modülün tüm işlemlerinin ayarlarını all: tuşu altında gruplanabilir.

Kod 6.90 - Tam bir cache.yml örneği /config/cache.yml

```
list:
  enabled:    on
show:
  enabled:    on

all:
  with_layout: false   # Default value
  lifetime:    86400   # Default value
```

Şimdi farklı bir id bağımsız değişkenine sahip user/show işlemine her çağrı önbellekte yeni bir kayıt oluşturur. Bu nedenle

http://myapp.example.com/user/show/id/12

için önbellek,

http://myapp.example.com/user/show/id/25

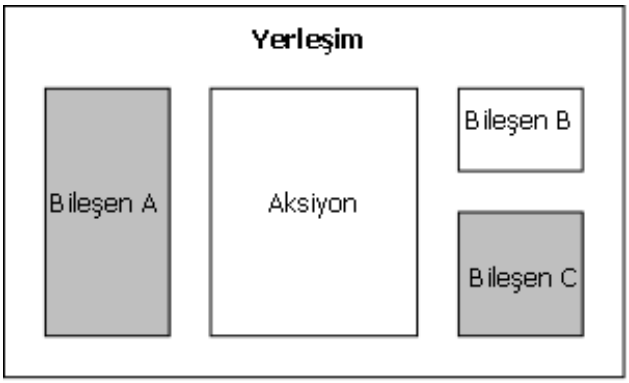
için önbellekten farklı olacaktır. POST yöntemiyle ya da GET parametreleriyle çağrılan işlemler önbelleklenmez.

with_layout ayarı, hakkında birkaç kelime daha edilmesini hak ediyor. Gerçekte önbellekte ne tür verilerin kaydedildiğini belirler. Yerleşimsiz önbellek için, sadece şablon uygulamasının sonucu ve işlem değişkenleri önbelleğe kaydedilir. Yerleşimli önbellek için, tüm yanıt nesnesi kaydedilir. Bu da yerleşimli önbelleğin yerleşimsiz önbellekten daha hızlı olduğu anlamına gelir.

İşlevsel olarak bunun altından kalkabilirseniz (yani yerleşim oturuma bağlı verilere dayalı değilse), yerleşimsiz önbelleği seçmelisiniz. Ne yazık ki yerleşim genellikle bazı dinamik unsurlar (örneğin, bağlanan kullanıcının adı) içerir; bu nedenle yerleşimsiz işlem önbelleği en yaygın yapılandırmadır. Ancak tanımlama bilgilerine bağlı olmayan RSS beslemeleri, açılır menüleri ve sayfaları yerleşimsiz olarak önbelleklenebilir.

6.3.4. Parçayı, Bileşeni Ya Da Bileşen Yuvasını Önbellekleme

Bir parçanın önbelleklenmesi bir işlemin önbelleklenmesi kadar kolaydır ve önbellek etkinleştirmesi, Şekil 6.10'da gösterildiği gibi, aynı kuralları izler.



Bileşen Önbelleği

Şekil 6.10 - Parçayı, bileşeni ya da bileşen yuvasını

Örneğin, Kod 6.91, kullanıcı modülünde yer alan bir my_partial.php parçasında önbelleği etkinleştirmek için cache.yml dosyasının nasıl düzenleneceğini gösterir. with_layout ayarının bu örnekte bir anlam ifade etmediğine dikkat edilmelidir.

Kod 6.91 - Bir parçayı önbellekleme

```
_my_partial:
  enabled:    on
list:
  enabled:    on
...
```

Şimdi bu parçayı kullanan bütün şablonlar parçanın PHP kodunu uygulamak yerine önbelleklenmiş sürümü kullanacaklardır.

```
<?php include_partial('user/my_partial') ?>
```

İşlemlerde olduğu gibi, parça önbelleklemesi de parçanın sonucu parametrelere bağlı olduğunda anlamlıdır. Ne kadar farklı parametre değeri varsa, önbellek sistemi o kadar çok şablon sürümü kaydedecektir.

```
<?php include_partial('user/my_other_partial', array('foo' => 'bar')) ?>
```

İşlem önbelleği, parça önbelleğinden daha güçlüdür, çünkü bir işlem önbelleklendiğinde, şablon uygulanmaz bile; eğer şablon parçalara çağrı içeriyorsa, bu çağrılar uygulanmaz. Dolayısıyla parça önbelleklemesi sadece çağrı işleminde işlem önbelleklemesi kullanmıyorsan ya da yerleşimde yer alan parçalar için faydalıdır.

Bileşen, bir parçanın üstüne konmuş hafif bir işlemdir ve bileşen yuvası da ilgili işlemin çağrı işlemlerine göre değiştiği bir bileşendir. Bu iki kapsama türü parçalarla çok benzerdir ve önbelleklemeyi aynı şekilde destekler. Örneğin, genel yerleşiminiz, güncel tarihi göstermek için include_component('general/day') ile day adlı bir bileşen içeriyorsa, bu bileşendeki önbelleği etkinleştirmek için genel modülün cache.yml dosyasını aşağıdaki gibi ayarlayın:

```
_day:
  enabled: on
```

Bir bileşeni ya da parçayı önbelleklerken, tüm çağrı şablonları için tek bir sürüm mü, yoksa her şablon için bir sürüm mü kaydedeceğinize karar vermeniz gerekir. Varsayılan olarak, bir bileşen onu çağıran şablondan bağımsız olarak kaydedilir. Ancak her işlemle farklı bir kenar çubuğu görüntüleyen bir bileşen gibi, bağlamsal bileşenler onu çağıran kaç tane şablon varsa, o kadar çok kaydedilmelidir. contextual parametresini true olarak ayarlamanız kaydıyla, önbellekleme sistemi bu durumu işleyebilir:

```
_day:
  contextual: true
  enabled:   on
```

Önbellek ayarlarını cache.yml uygulamasında tanımlamanız kaydıyla, genel bileşenler (templates/ dizin uygulamasında yer alanlar) önbelleklenebilir.

6.3.5. Şablon Bölümünü Önbellekleme

İşlem önbelleklemesi sadece bir işlem altkümesine uygulanır. Diğer işlemler için - verileri güncelleyenler ya da şablondaki oturuma bağımlı bilgileri görüntüleyenler - yine önbellek gelişimi mümkündür; ancak başka bir şekilde. Symfony, şablon bölümlerine yönelik olan ve doğrudan şablonun içinde etkinleştirilen üçüncü bir önbellek türü sunar. Bu türde, Şekil 6.11'de gösterildiği gibi, işlem daima uygulanır ve şablon uygulanmış bölümler ve önbellekteki bölümler olarak ayrılır.

Şekil 6.11 - Şablon bölümünü önbellekleme

Örneğin, son erişilen kullanıcının bağlantısını gösteren bir kullanıcı listeniz olabilir ve bu bilgi dinamiktir. cache() yardımcısı, bir şablonun önbelleğe konulacak parçalarını tanımlar. Sözdizimine ilişkin ayrıntılar için Kod 6.92'ye bakıldığında:.

Kod 6.92 - cache() fonksiyonunun kullanımı

```
<!-- Code executed each time -->
<?php echo link_to('last accessed user', 'user/show?id='.$last_accessed_user_id) ?>

<!-- Cached code -->
<?php if (!cache('users')): ?>
  <?php foreach ($users as $user): ?>
    <?php echo $user->getName() ?>
  <?php endforeach; ?>
  <?php cache_save() ?>
<?php endif; ?>
```

Bu, aşağıdaki gibi çalışır:

- 'Kullanıcılar' adlı bölümün önbelleklenmiş bir sürümü varsa, satırlar arasındaki kodu değiştirmek üzere kullanılır.
- Yoksa bu satırlar arasındaki kod işlenir ve özgün parça adıyla tanımlanmış olarak önbelleğe kaydedilir.

Bu satırlar arasına dâhil edilmeyen kod daima işlenir ve önbelleklenmez.

İşlemin (örnekteki liste) etkinleştirilmiş bir önbelleklemesi olmaması gerekir; çünkü bu tüm şablon uygulamasını atlayacak ve parça önbelleği tanımlamasını yok sayacaktır.

Şablon bölümü önbelleğini kullanmanın yol açtığı hız artışı, işlem önbelleğindeki kadar önemli değildir; çünkü işlem daima uygulanır, şablon kısmen işlenir ve yerleşim daima dekor olarak kullanılır.

Aynı şablonda ek bölümler tanımlanabilir; ancak Symfony önbellek sisteminin onları daha sonra bulabilmesi için her birine özgün bir ad verilmesi gerekir.

İşlemler ve bileşenlerde olduğu gibi, önbelleklenmiş bölümler de cache() yardımcısına çağrının ikinci bağımsız değişkeni olarak saniye biriminde bir ömür alabilir.

<?php if (!cache('users', 43200)): ?>

Yardımcıya hiçbir parametre verilmezse varsayılan önbellek ömrü (86400 saniye ya da bir gün) kullanılır.

Bir işlemi önbelleklemenin bir başka yolu da onu değiştiren değişkenleri işlemin yönlendirme örüntüsüne eklemektir. Örneğin, bir ana sayfa bağlanan kullanıcının adını görüntülüyorsa, URL kullanıcının takma adını içermediği sürece önbelleklenmez. Bir başka örnek de uluslararası uygulamalardır: Birçok çevirisi olan bir sayfada önbelleklemeyi etkinleştirmek isteniliyorsa, dil kodunun bir şekilde URL örüntüsüne dâhil edilmesi gerekir. Bu hile, önbellekteki sayfa sayısını çoğaltır; ancak son derece etkileşimli uygulamaların hızlandırılmasında büyük yardımı olur.

6.3.6. Önbelleği Dinamik Olarak Yapılandırma

cache.yml dosyası, önbellek ayarlarını tanımlamanın bir yoludur; ancak sabit olmak gibi bir sakıncası vardır. Bununla birlikte, Symfony'de alışılageldiği üzere, YAML yerine düz PHP kullanabilinir ve bu da önbelleği dinamik olarak yapılandırılmasına olanak tanır.

Önbellek ayarlarını dinamik olarak değiştirmek istenilmesine bir örnek olarak sayfa kimliği doğrulanan kullanıcılar ve adsız kullanıcılar için farklıyken URL'sinin aynı kalması verilebilir. Makaleler için bir değerlendirme sistemi olan bir article/show sayfası hayal edin. Değerlendirme özelliği adsız kullanıcılar için devre dışı bırakılır. Bu kullanıcılar için değerlendirme bağlantıları bir bağlanma formunun görüntülenmesini tetikler. Sayfanın bu sürümü önbelleklenebilir. Diğer yandan, kimliği doğrulanmış kullanıcılar için, bir değerlendirme bağlantısına tıklamak bir POST istemi yapar ve yeni bir değerlendirme oluşturur. Bu kez önbellek sayfa için devre dışı bırakılarak Symfony'nin onu dinamik olarak oluşturmasının sağlanması gerekir.

Dinamik önbellek ayarlarını tanımlamak için doğru yer, sfCacheFilter'den önce uygulanan bir filtredir. Aslında önbellek, tıpkı web hata ayıklama araç çubuğu ve güvenlik özellikleri gibi, Symfony'de bir filtredir. Sadece kullanıcının kimliği doğrulanmadığı takdirde article/show sayfası için önbelleği etkinleştirmek amacıyla,

Kod 6.93'de gösterildiği gibi, lib/ dizin uygulamasında bir conditionalCacheFilter oluşturulmalıdır.

Kod 6.93 - Durumsal önbellekleme filtresi

```
class conditionalCacheFilter extends sfFilter
{
  public function execute($filterChain)
  {
    $context = $this->getContext();
    if (!$context->getUser()->isAuthenticated())
    {
      foreach ($this->getParameter('pages') as $page)
      {
        $context->getViewCacheManager()->addCache($page['module'], $page['action'],
array('lifeTime' => 86400));
      }
    }

    // bir sonraki filtreyi çalıştır
    $filterChain->execute();
  }
}
```

Kod 6.94'da gösterildiği gibi, bu filtreyi filters.yml dosyasına sfCacheFilter'den önce kaydetmeniz gerekir.

Kod 6.94 - Özel Filtre kullanımı

```
...
security: ~

conditionalCache:
  class: conditionalCacheFilter
  param:
    pages:
      - { module: article, action: show }

cache: ~
...
```

Önbellek temizlendiğinde (yeni filtre sınıfını otomatik yüklemek için), koşullu önbellek hazırdır. Bu, sadece kimliği doğrulanmamış kullanıcılar için sayfalar parametresinde tanımlanan sayfaların önbelleğini etkinleştirir.

sfViewCacheManager nesnesinin addCache() yöntemi, bir cache.yml dosyasında tanımlayacaklarınızla aynı parametrelere sahip bir modül adı, bir işlem adı ve bir çağrışımsal dizin bekler. Örneğin, article/show action işleminin yerleşimle ve 3600 saniyelik bir ömürle önbelleklenmesi gerektiğini tanımlamak istenirse, aşağıdaki gibi bir kod yazılmalıdır:

Kod 6.95 - 3600 saniyelik önbellek kullanımı

```
$context->getViewCacheManager()->addCache('article', 'show', array(
  'withLayout' => true,
  'lifeTime'   => 3600,
));
```

6.3.7. Alternatif Önbellekleme Kaydetmesi

Varsayılan olarak, Symfony önbellek sistemi, dosyalardaki verileri web sunucusunun sabit diskine kaydeder. Önbelleği belleğe (örneğin, memcached aracılığıyla) ya da bir veritabanına (özellikle önbelleğinizi birçok sunucu arasında paylaşmak ya da önbelleği kaldırmayı hızlandırmak istendiğinde) kaydetmek istenebilir. Symfony'nin varsayılan önbellek kaydetme sistemini kolaylıkla değiştirilebilir; çünkü Symfony görünüm önbelleği yöneticisinin kullandığı önbellek sınıfı, factories.yml'de tanımlanır.

Varsayılan görünüm önbelleği kaydetme yeri sfFileCache sınıfıdır:

Kod 6.96 - Örnek factories.yml dosyası

```
view_cache:
   class: sfFileCache
   param:
     automaticCleaningFactor: 0
     cacheDir:              %SF_TEMPLATE_CACHE_DIR%
```

Sınıfı, kendi önbellek kaydetme sınıfınızla ya da Symfony'nin alternatif sınıflarından biriyle (örneğin sfSQLiteCache) değiştirebilirsiniz. param tuşunun altında tanımlanan parametreler, sınıfınızın initialize() yöntemine çağrışımsal bir dizin olarak geçer. Herhangi bir görünüm önbellek kaydetme sınıfının soyut sfCache sınıfında bulunan tüm yöntemleri uygulaması gerekir.

6.3.8. Aşırı Hızlı Önbellek Kullanma

Önbelleklenmiş bir sayfa bile bazı PHP kod uygulamaları içerir. Böyle bir sayfa için, Symfony yine de yapılandırmayı yükler, yanıtı oluşturur, vs. Bir sayfanın bir süre için değişmeyeceğinden gerçekten eminseniz, ortaya çıkan HTML kodunu doğrudan web/ klasörüne koyarak Symfony'yi tümüyle atlanabilinir. Bu, yönlendirme kuralının son eksiz olarak ya da .html'le biten bir örüntüyü belirlemesi kaydıyla, Apache mod_rewrite ayarları sayesinde mümkündür.

Bunu sayfa sayfa, basit bir komut satırı çağrısıyla, elle yapılabilir:

> curl http://myapp.example.com/user/list.html > web/user/list.html

Bundan sonra, user/list işleminin her talep edilmesinde, Apache, denk düşen list.html sayfasını bulur ve Symfony'yi tamamen atlar. Artık sayfa önbelleğini Symfony ile kontrol edilemez (ömür, otomatik silme, vs); ancak hız kazanımı çok etkileyicidir.

6.3.9. Diğer Hızlanma Taktikleri

HTML önbelleğine ek olarak, Symfony'de tamamen otomatik ve geliştiriciye saydam olan iki başka önbellek mekanizması daha bulunur. Üretim ortamında, yapılandırma ve şablon dönüşümleri, myproject/cache/config/ ve myproject/cache/i18n/ dizinlerinde kayıtlı dosyalara hiçbir müdahale olmadan önbelleklenir.

Aynı zamanda opcode önbellekleme modülleri adı da verilen PHP hızlandırıcıları (eAccelerator, APC, XCache, vs), PHP komut dizilerini derlenmiş durumda önbellekleyip kod ayrıştırması ve derlemesinin ek yüklerinin neredeyse tamamen ortadan kaldırılmasını sağlayarak, PHP komut dizilerinin performansını artırır. Bu özellikle çok büyük miktarda kod içeren Propel sınıfları için etkilidir. Bu hızlandırıcılar, Symfony ile uyumludur ve bir uygulamanın hızını kolayca üç katına çıkarabilir. Geniş bir hedef kitlesi olan herhangi bir Symfony uygulaması için üretim ortamlarında tavsiye edilirler.

Bir PHP hızlandırıcısıyla, her istem için aynı işlemeyi yapmaktan kaçınmak amacıyla, sfProcessCache sınıfıyla kalıcı verileri belleğe elle kaydedebilirsiniz. Eğer CPU yoğun

bir işlemin sonucunu bir dosyaya kaydetmek istiyorsanız, sfFunctionCache nesnesi kullanılacaktır.

6.3.10. Önbellekten Öğeleri Kaldırma

Eğer bir uygulamanızın komut dizileri ya da verileri değişiyorsa, önbellek eski bilgiler içerecektir. Tutarsızlık ve hatalardan kaçınmak için, önbelleğin parçalarını, ihtiyaçlara göre kaldırılabilir. Symfony komut satırının clear-cache görevi, önbelleği (HTML, yapılandırma ve i18n önbelleği) siler. Kod 6.97'de gösterildiği gibi, önbelleğin sadece bir altkümesini silmek için bunu bağımsız değişkenlere geçirebilirsiniz. Bunu sadece bir Symfony projesinin kökünden çağırmayı unutmayın.

Kod 6.97 - Önbelleğin temizlenmesi

```
// bütün önbelleği sil
> symfony clear-cache

// kısa sözdizimi
> symfony cc

// sadece myapp uygulamasının önbelleğini sil
> symfony clear-cache myapp

// sadece myapp uygulamasınınHTML önbelleğini sil
> symfony clear-cache myapp template

// sadece myapp uygulamasının yapılanış önbelleğini sil
> symfony clear-cache myapp config
```

6.3.11. Symfony Yapılandırma Önbelleği

YAML'yi ayrıştırma ve çalıştırmada yapılandırma basamaklaması, her istem için önemli bir ek yük gerektirir. Symfony'de istemleri hızlandırmak için tasarlanmış yerleşik bir yapılandırma önbellek mekanizması mevcuttur.

Formatları ne olursa olsun, yapılandırma dosyaları, işleyici denilen ve bu dosyaları hızlı işlem OOP koduna dönüştüren bazı özel sınıflar tarafından işlenir. Geliştirme ortamında, işleyiciler, etkileşimi desteklemek için, yapılandırmada her istemde meydana gelen değişiklikleri kontrol eder. Yeni değiştirilmiş dosyaları ayrıştırarak YAML dosyasındaki bir değişikliğin derhal görülmesi sağlar. Ancak üretim ortamında, işlem

ilk istem sırasında bir kez gerçekleşir ve sonra işlenen PHP kodu sonraki istemler için önbelleğe kaydedilir. Performans garanti edilmiştir; çünkü üretimdeki her istem sadece bazı optimize edilmiş PHP kodlarını uygulayacaktır.

Örneğin, app.yml dosyası aşağıdakini içerirse:

Kod 6.98 - Konfigürasyon dosyası

```
all:                # Setting for all environments
  mail:
    webmaster:        webmaster@example.com
```

Projenizin `cache/` klasöründe yer alan `config_app.yml.php`, dosyası aşağıdakini içerecektir:

Kod 6.99 - Konfigurasyon dosyasının yarattığı önbellek

```
<?php

sfConfig::add(array(
  'app_mail_webmaster' => 'webmaster@example.com',
));
```

Sonuç olarak, çoğu zaman, YAML dosyaları, çerçeve tarafından bile ayrıştırılmaz; çünkü çerçeve yapılandırma önbelleğine dayanır. Bununla birlikte, geliştirme ortamında, Symfony, YAML dosyalarının ve önbelleklenmiş dosyaların değişiklik tarihlerini sistematik olarak karşılaştıracak ve sadece bir önceki istemden beri değişmiş olanları yeniden işleyecektir.

Bu da yapılandırma dosyalarının üretimde bile her istemde derlendiği birçok PHP çerçevesi karşısında büyük bir üstünlük sağlar. Java'nın aksine, PHP, istemler arasında bir uygulama bağlamı paylaşmaz. Diğer PHP çerçeveleri için, XML yapılandırma dosyalarının esnekliğini korumak, her istemde tüm yapılandırmayı işlemek açısından büyük bir performans gerektirir. Symfony'de durum böyle değildir. Önbellek sistemi sayesinde, yapılandırmanın yol açtığı ek yük çok düşüktür.

Bu mekanizmanın önemli bir sonucu vardır. Üretim ortamında yapılandırmayı değiştirirseniz, değişikliğinizin dikkate alınması için tüm yapılandırma dosyalarının yeniden ayrıştırılmasını zorlamanız gerekir. Bu nedenle, ya cache/ dizininin içeriğini

silerek ya da daha kolay bir şekilde cache:clear görevini çağırarak, önbelleğin temizlenmesi gerekir:

> php symfony cache:clear

6.4. APC ÖNBELLEKLEME

Komut dizisi dillerinin kodu alması, derlemesi ve sonra derlenen kodu uygulaması gerekir. Bir sayfa her yüklendiğinde, kod derlenir ve bu safha her sayfa yüklemesinde zamanın ve kaynakların % 90'ını alabilir. Bu, bilgisayarınızda bulunan, kodun zaten derlenmiş olduğu bir programdan farklıdır. Bu safha ortadan kaldırılabilirse, sunucu daha fazla sayfa istemini işleyebilir. İşte APC ve diğerleri gibi opcode önbellekleri bu noktada çoğu zaman bu safhayı atlamamıza olanak tanıyarak performansı büyük ölçüde artırır.

APC olarak bilinen Alternatif PHP Önbelleği, web sunucunuzun derlenen PHP'leri kaydetmesine olanak tanıyarak derleme safhasının atlanmasını sağlar. Bu sayede sunucunuzun performansı ciddi ölçüde artar.

7. GÜVENLİK ÖZELLİKLERİ

7.1. FORM DOĞRULAMA

Kullanıcı verilerini sisteme alırken çeşitli şekillerde doğrulama işlemlerinden geçirerek, hem zararlı kodları filtrelemek, hem de belli bir düzende veri girişini sağlamaktır.

7.1.1 Doğrulama Dosyası

Oturum açma formu bir takma ad ve bir şifre alanına sahiptir. Ama eğer kullanıcı yanlış veri gönderirse ne olacak? Bu sorunu çözmek için /frontend/modules/user/validate klasörünün altında bir login.yml dosyası oluşturun. (login doğrulama için gerekli aksiyona verilmiş olan addır) Daha sonra aşağıdaki içerik eklendiğinde:

Kod 7.100 - Örnek form doğrulama dosyası

```
methods:
 post: [takma ad, şifre]
names:
 nickname:

 required: true
 required_msg: takma adınızı yazmanız gerekir
 validators: nicknameValidator
 password:
 required: true
 required_msg: şifrenizi yazmanız gerekir
nicknameValidator:
 class: sfStringValidator
 param:
 min: 5
 min_error: takma adınız 5 ya da daha fazla karakterden oluşmalıdır
```

İlk olarak, methods başlığının altında, form metotları için doğrulanacak alanların bir listesi tanımlanmıştır. (Burada yalnızca POST metodunu tanımlıyoruz, çünkü GET metodu giriş formunu göstermek içindir ve doğrulama gerektirmez) Daha sonra, names başlığı altında, denetlenecek alanların her biri için gereksinimler ve ilgili hata mesajları listelenmektedir. Son olarak, 'takma ad' alanı diğerlerinden farklı doğrulama kuralları

gerektirdiği için, ilgili kurallar kendilerine ait başlıkların altında listelenmiştir. Bu örnekte, sfStringValidator doğrulayıcısı, bir karakter dizisinin formatını denetleyen Symfony'nin içinde yerleşik bir doğrulayıcıdır.

7.1.2 Hataların Ele Alınması

Eğer bir kullanıcı yanlış veri girerse ne olur? login.yml dosyasında yazılmış olan şartlar gerçekleşmeyecektir, ve Symfony denetçisi isteği, form_tag argümanında planlandığı gibi, executeLogin() metodunun yerine userActions sınıfının handleErrorLogin() metoduna geçirecektir. Eğer metot mevcut değilse, varsayılan davranış loginError.php şablonunu göstermek olacaktır. Çünkü önseçili handleError() metodu aşağıdaki değeri döndürmektedir:

Kod 7.101 - handleError() fonksiyonu

```
public function handleError()
{
 return sfView::ERROR;
}
```

Bu aslında doldurulacak yepyeni bir şablondur. Ama biz yanlış girilmiş alanların yanında hata mesajları görünecek şekilde oturum açma formunu tekrar göstermeyi tercih ediyoruz. Şimdi, bu sefer loginSuccess.php şablonunu görüntülemek için oturum açma hatası davranışının değiştirilmesi gerekmektedir:

Kod 7.102 - Şablon görüntüleme davranışlarının değiştirilmesi

```
public function handleErrorLogin()
{
 return sfView::SUCCESS;
}
```

7.1.3. Şablon Hatası Yardımcıları

loginSuccess.php şablonu tekrar ekrana geldiğinde, hataları görüntüleme zamanı gelmiş demektir. Bu amaç için Validation yardımcı grubunun içindeki form_error() yardımcı fonksiyonu kullanılabilir. Aşağıdaki gibi şablonun iki form-sırası div'i değiştirildiğinde:

Kod 7.103 - Şablon hatası yardımcıları

```
<?php use_helper('Validation') ?>
<div class="form-row">
 <?php echo form_error('nickname') ?>
 <label for="nickname">nickname:</label>
 <?php echo input_tag('nickname', $sf_params->get('nickname')) ?>
</div>
<div class="form-row">
 <?php echo form_error('password') ?>
 <label for="password">password:</label>
 <?php echo input_password_tag('password') ?>
</div>
```

Form_error() yardımcı fonksiyonu, eğer parametre olarak verilen alanda bir hata bildirilmişse, login.yml şablonunda tanımlanan hata mesajını görüntüleyecektir.

Takma ad alanına 5 karakterden daha az bir metin girerek veya iki zorunlu alandan birini ihmal ederek form doğrulama test edildiğinde hata mesajları şekilde ilgili alanların üzerinde görüntülenecektir.

7.2. WEB UYGULAMALARINDA OLASI GÜVENLİK AÇIKLARI

7.2.1 Sql Injection

Web uygulamalarında bazı bilgilerin tutulabilmesi için SQL veritabanları kullanılmaktadır. Uygulama geliştiricileri, bazı durumlarda kullanıcılardan gelen verileri beklenen veri türü ile karşılaştırmayarak SQL sorguları içinde kullanmaktadırlar. Genel olarak problemler, uygulama geliştiricinin SQL sorgularında anlam ifade edebilecek **‘ ; UNION** gibi kötü niyetli karakterlere karşı bir önlem almadığı zaman ortaya çıkmaktadır. Bu durum kullanıcıya önceden planlanmamış uygulama düzeyinde erişim sağlayabilir. İçinde SQL sorgulama barındıran birçok ürün SQL sorguları değiştirilebilmesine (SQL Injection) karşı savunmasızdır. Saldırganlar SQL sorgularını değiştirme tekniklerini web sitelerine ve uygulamalara zarar vermek amaçlı kullanmaktadırlar. SQL enjeksiyon(Benjamin L., 2004) ile saldırgan tablo yaratabilir, değişiklikler yapabilir, veritabanı üzerinde erişim sağlayabilir veya veritabanı kullanıcısının hakları doğrultusunda sunucuda komut çalıştırabilir.

Bir web uygulamasının Sql Injection'dan korunabilmesi için tüm bileşenlerinde kullanılan değişkenler için kontroller oluşturulmalı ve değişkene atanması beklenen veri türü ile kullanıcı girdisi karşılaştırılmalıdır. Beklenen girdi türünden farklı karakterler saptanması durumunda, karakterler SQL sorgularında anlam ifade etmeyecek biçimde değiştirilmeli, silinmeli veya kullanıcıya uyarı mesajı döndürülmelidir. Tercihen uygulamanın tamamı için geçerli olacak, değişken türü ve atanabilecek girdi türünü parametre olarak alan ve kontrolleri yaptıktan sonra girdi kabul sonucu üreten sabit bir fonksiyon tercih edilmelidir.

7.2.2. Css / Xss / Crsf

Başka siteden kod çalıştırma (Cross-Site Scripting) açıkları, bir saldırganın hedef web sitesi aracılığıyla site ziyaretçilerinin sisteminde komut çalıştırabilmesine olanak tanımaktadır.(Jovanıvich, 2006) Saldırı sonucu olarak site ziyaretçilerinin browser'larında bulunabilecek güvenlik açıklarının kullanılması, JavaScript/ActiveX ve VBScript komutlarının çalıştırılmasını mümkün kılmaktadır. Bu tür komutlar ile kullanıcıya ait site çerezleri alınabilir, kaydedilmiş şifreler çalınabilir veya tarayıcıda bulunabilecek güvenlik açıkları ile kullanıcı sistemi ele geçirilebilir. Ayrıca elektronik ticaret veya bankacılık uygulamaları için sahte giriş ekranları oluşturularak ziyaretçilerin yanıltılması ve sonucunda kullanıcıya ait önemli bilgilerin ele geçirilmesi mümkün olabilir.

Bir web uygulamasının Remote Scripting den korunabilmesi için tüm bileşenlerinde kullanılan değişkenler için kontroller oluşturulmalı ve değişkene atanması beklenen veri türü ile kullanıcı girdisi karşılaştırılmalıdır. Beklenen girdi türünden farklı karakterler (örn. **<>/;()**) saptanması durumunda, karakterler anlam ifade etmeyecek biçimde değiştirilmeli, silinmeli veya kullanıcıya uyarı mesajı döndürülmelidir. Tercihen uygulamanın tamamı için geçerli olacak, değişken türü ve atanabilecek girdi türünü parametre olarak alan ve kontrolleri yaptıktan sonra girdi kabul sonucu üreten sabit bir fonksiyon tercih edilmelidir.

8. TEST SONUÇLARI VE BULGULAR

8.1. TEST PLATFORMU

Çerceve model kullanımının performans olan etkilerini analiz edebilmek için lokal ağda çalısan bir istemci ve sunucu kullanıldı. Paketler sunucuya 100 mbit performansda bir ağdan gönderildi.

8.1.1. İşlemci Bilgisi (Cpu Info)

Test sunucusunda kullanılan işlemci özellikleri aşağıdaki gibidir.

```
root@devil:/var/www/devosphere# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 2
model name      : Intel(R) Celeron(R) CPU 2.40GHz
stepping        : 9
cpu MHz         : 2400.273
cache size      : 128 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe up pebs bts cid xtpr
bogomips        : 4805.46
clflush size    : 64
```

8.1.2. Hafıza Bilgisi (Mem Info)

Sunucuda aşağıda özellikleri verilmiş 1 gb hafiza kullanılmıştır. Swap alan 512 mb olarak yapılandırılmıştır.

```
root@devil:/var/www/devosphere# cat /proc/meminfo
MemTotal:      1002776 kB
MemFree:        568392 kB
Buffers:        105608 kB
Cached:         175288 kB
```

SwapCached: 0 kB
Active: 264496 kB
Inactive: 130384 kB
HighTotal: 98288 kB
HighFree: 224 kB
LowTotal: 904488 kB
LowFree: 568168 kB
SwapTotal: 498004 kB
SwapFree: 498004 kB
Dirty: 48 kB
Writeback: 0 kB
AnonPages: 114044 kB
Mapped: 36796 kB
Slab: 28128 kB
SReclaimable: 17360 kB
SUnreclaim: 10768 kB
PageTables: 2612 kB
NFS_Unstable: 0 kB
Bounce: 0 kB
CommitLimit: 999392 kB
Committed_AS: 325368 kB
VmallocTotal: 118776 kB
VmallocUsed: 2932 kB
VmallocChunk: 115600 kB

8.1.3. Test Bilgisayarı Özellikleri

8.1.3.1. Donanım

Test neticelerindeki farkların doğru gözlemlenebilmesi için tek işlemcili bir sunucu tercih edilmiştir.

İşlemci: Intel(R) Celeron(R) CPU 2.40GHz
Hafıza: 1 Gb
Sabit disk: Barracuda 7200.10 SATA

8.1.3.2. Yazılım

Test platformu için Debian temelli Ubuntu Hardy Heron kullanılmıştır. Sunucu standart özellikler ile yapılandırılmış performans optimizasyonları yapılmamıştır.

İşletim sistemi: Ubuntu Hardy Heron Server Edition
Veritabanı Sunucusu: MySQL 5.0, Tablo tipi- InnoDB
Web Sunucusu: Apache HTTP Server 2.2.8
PHP versiyonu: 5.2.4

Propel versiyonu: 1.3
Doctrine versiyonu: 0.10.2

8.2 UYGULANAN TESTLER

8.2.1. Jmeter Testi

Apache JMeter statik ve dinamik kaynakların performansını test etmek için kullanılabilir (dosyalar, sunucu uygulamacıkları, Perl betikleri, Java nesneleri, veritabanları ve sorgulamaları, FTP sunucuları ve dahası). Bir server/ağ/nesne'nin gücünü test etmek için ağır yük simüle eder veya değişik yük durumlarında bütün performansı test eder. Performans analizini yapmak için veya sunucu/betik/nesne davranışını test etmek için ağır yük altında kullanılır.

8.1.2. Apachebench Testi

ApacheBench özellikle Apache HTTP Sunucusu gibi HTTP web sunucularının performansını ölçmeye yarayan bir komut satırı programıdır. Bir Apache kurulumunun performansı hakkında fikir vermek için dizayn edilmiştir. Özellikle bir sunucunun saniyede kaç isteğe cevap verebildiğini gösterir.

8.2. ORM TESTİ

Bu testte içerisinde Insert, Delete, Select, Joined Select, Many Select işlemlerini içeren bir script dosyasının çalışımı ve bitimi arasında geçen işlem süresi baz alınmıştır. Her testten sonra hafızadaki objeler temizlenmiştir.

Her test 5 defa yapılarak ardından 5 sonuç izlenerek averaj hesaplandı. Yüzde pariteler averaj baz alınarak hesaplandı. Birim olarak saniye kullanılmış olup virgülden 5 basamak sonrası yuvarlanmıştır. Averajlar hesaplanırken kesin değerler kullanılmıştır. Yüzdesel değerlerde virgulden 1 basamak sonrası yuvarlanmıştır.

8.2.1. Eklemek / Insert Testi

Bu testte 1000 zincirlik 3 birbirine bağlı objeden diğer testlerde kullanılacak 3000 kayıt girilecektir. Test için aynı işlemi yapan hem Doctrine hem Propel kodu kullanılmıştır. Aksiyon bir for döngüsü içerisinde 1000 defa satıcılar, müşteriler ve satış tablolarına bir kayıt atar.

8.2.1.1. Propel Kodu

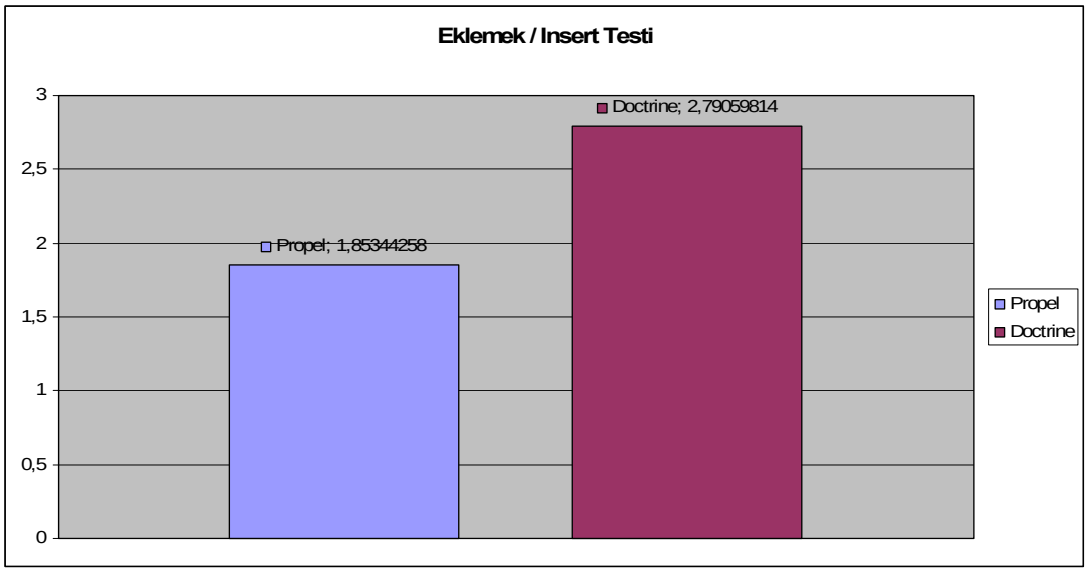
```
$conn->beginTransaction();
for ($i = 0; $i < 1000; $i++) {
   $satici = new Saticilar();
   $satici->setIsim('Satici  1');

   $musteri = new Musteriler();
   $musteri->setIsim('Musteri 1');

   $satis = new Satis ();
   $satis->setToplam(100);
   $satis->setSaticilar($satici);
   $satis->setMusteriler($musteri);
   $satis->save();
}
$conn->commit();

unset($satici);
unset($musteri);
unset($satis);
SaticilarPeer::clearInstancePool();
MusterilerPeer::clearInstancePool();
SatislarPeer::clearInstancePool();
```

8.2.1.2. Doctrine Kodu

```
$conn->beginTransaction();
for ($i = 0; $i < 1000; $i++) {

   $satici = new Satici();
   $satici->isim   = ‘Satıcı Adı’;

   $musteri = new Musteri();
   $musteri->isim = 'Musteri Adı’;

   $satis = new Satis();
   $satis ->toplam     = 100;
```

```
    $satis->Satici   = $seller;
    $satis->Musteri = $musteri;
    $satis->save();

    $satici->free();
    $musteri->free();
    $satis->free();
 }
$conn->commit();
unset($satici);
unset($musteri);
unset($satis);
```

Şekil 8.12 incelediğimizde iki farklı ORM platformunda yapılan testler neticesinde Propelin Ekleme işleminde Doctrine'den %50 daha hızlı çalıştığı gözlemlenmiştir.



Şekil 8.12 - Ekleme / Insert testi sonuçları

8.2.2. Seçme / Select Testi

Bu testte tüm satıcılar listelenmektedir. Bir foreach döngüsü içerisinde her seferinde ORM ile satıcılar tablosundan bir kayıt alınmış ve getIsim() methodu çağırılarak isim değişkenine değer kayıt edilmiştir. Testte toplam 1000 kayıt listelenmiştir.

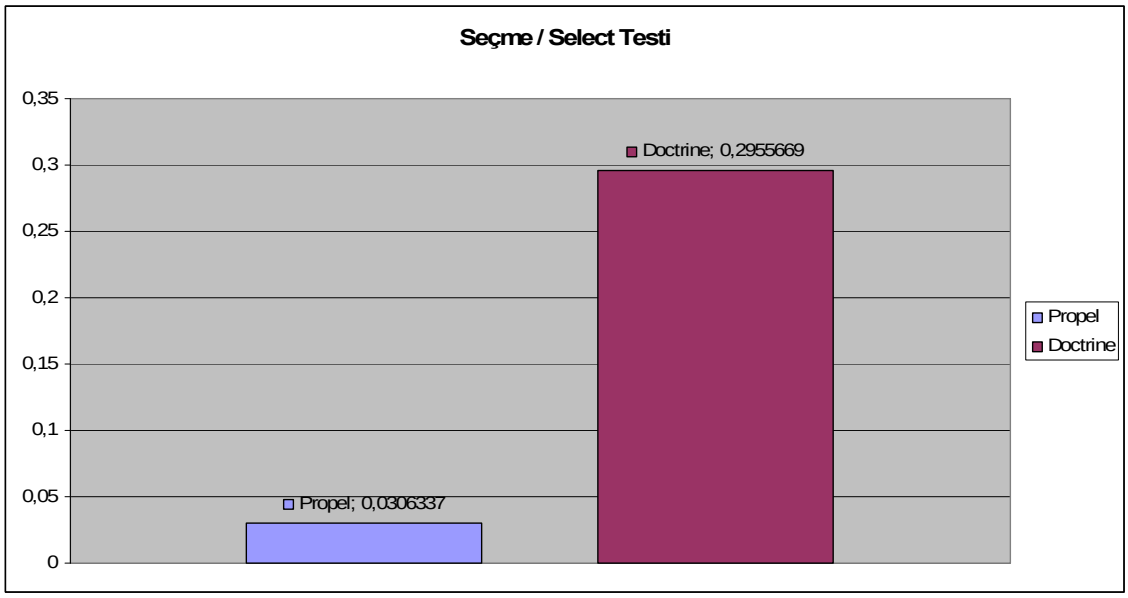
8.2.2.1. Propel Kodu

```
foreach (SaticilarPeer::doSelect(new Criteria()) as $satici) {
   $isim = $satici->getIsim();
}
unset($satici);
SaticilarPeer::clearInstancePool();
```

8.2.2.2. Doctrine Kodu

```
foreach ($conn->getTable('Satici')->findAll() as $satici) {
   $isim = $satici->isim;
   $satici->free();
 }
unset($satici);
```

Şekil 8.13 incelediğimizde iki farklı ORM platformunda yapılan testler neticesinde Propelin seçme (select) işleminde Doctrine'den %964 daha hızlı çalıştığı gözlemlenmiştir.



Şekil 8.13 - Seçme / Select Testi sonuçları

8.2.3. Birleştirerek Seçme / Join Select Testi

Bu testte Satislar ve Saticilar tablosu birleştirilerek seçim işlemi yapılıyor. getToplam() metotu ile toplam değişkenine değer atanmıştır. Daha sonra satıcılar tablosu ile birleştirme işlemi yapılıp satış tablosunun içeriği(1000) kadar tekrarlanmıştır.

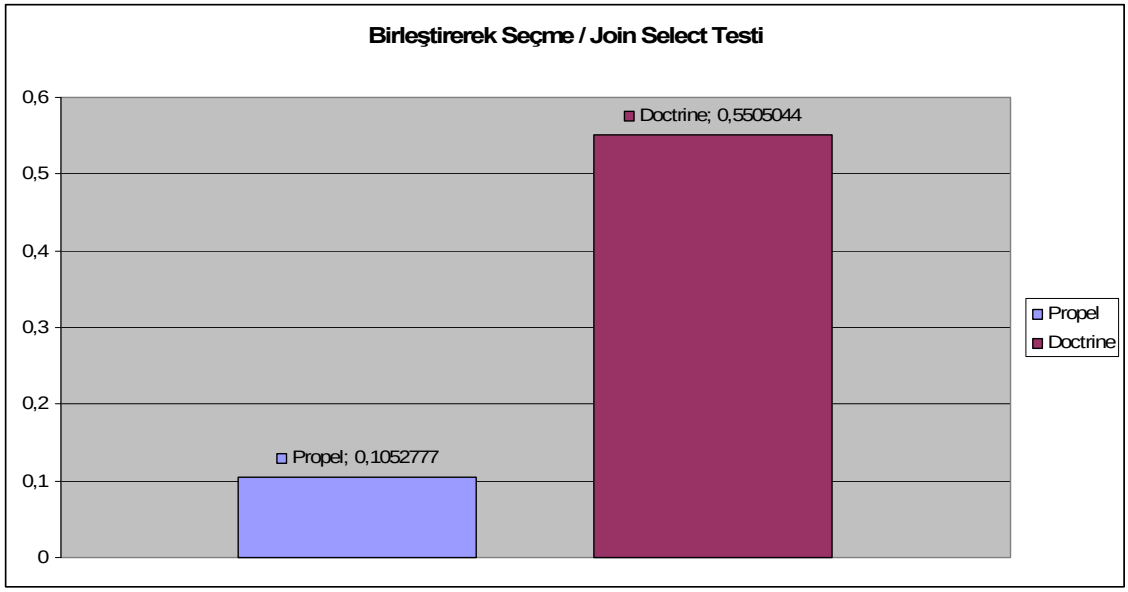
8.2.3.1. Propel Kodu

```
foreach (SatislarPeer::doSelectJoinSaticilar(new Criteria()) as $satis) {
   $toplam = $satis->getToplam();
   $isim = $satis->getSaticilar()->getIsim();
}
unset($satis);
SaticilarPeer::clearInstancePool();
SatislarPeer::clearInstancePool();
```

8.2.3.2. Doctrine Kodu

```
$satislar = Doctrine_Query::create()
      ->from('Satis o')
      ->leftJoin('o.Satici s')
      ->execute();

foreach ($satislar as $satis) {
   $toplam = $satis->toplam;
   $isim = $satis->Satici->isim;
   $satis->free(true);
 }
unset($satis);
$conn->getTable('Satis')->clear();
unset($satislar);
```

Şekil 8.14 incelediğimizde iki farklı ORM platformunda yapılan testler neticesinde Propelin birleştirerek seçme işleminde Doctrine'den %522 daha hızlı çalıştığı gözlemlenmiştir.



Şekil 8.14 - Birleştirerek Seçme / Join Select Testi sonuçları

8.2.4. Çoklu Seçme / Many Select Testi

Bu testte Satıcılar tablosundan çoklu seçme operasyonu yapılarak getIsim() metotu çağırılarak satıcıların isimleri isim değişkenine kayıt edilmiştir ve işlem satıcı adedi (1000) kadar tekrarlanmıştır.

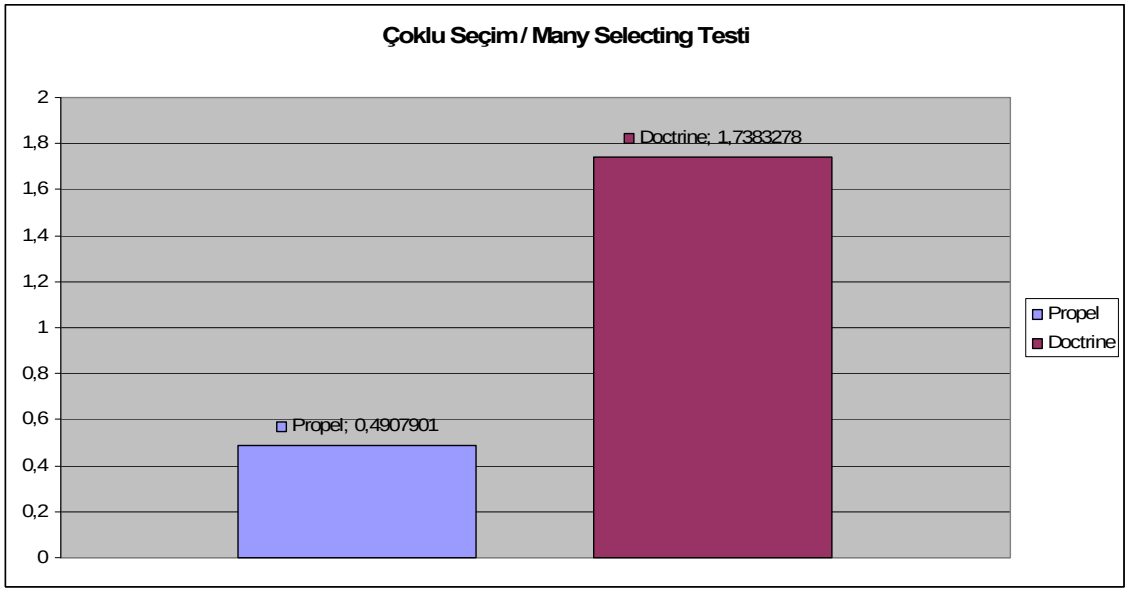
8.2.4.1. Propel Kodu

```
for ($i = 0; $i < 1000; $i++) {
   foreach (SaticilarPeer::doSelect(new Criteria()) as $satici) {
      $isim = $satici->getIsim();
   }
   SaticilarPeer::clearInstancePool();
 }
unset($satici);
```

8.2.4.2. Doctrine Kodu

```
for ($i = 0; $i < 1000; $i++) {
   foreach ($conn->getTable('Satici')->findAll() as $satici) {
      $isim= $satici->isim;
      $satici->free();
   }
}
unset($satici);
```

Şekil 8.15 incelediğimizde iki farklı ORM platformunda yapılan testler neticesinde Propelin çoklu seçme işleminde Doctrine'den %354,1 daha hızlıdır.



Şekil 8.15 - Çoklu Seçme / Many Select Testi sonuçları

8.2.5. Güncelleme / Update Testi

Güncelleme testinde satıcılar tablosun alınan her kayıtın isim değeri ‘Yeni İsim’ olarak güncellenmiş ve işlem kayıt sayısı kadar tekrarlanmıştır.

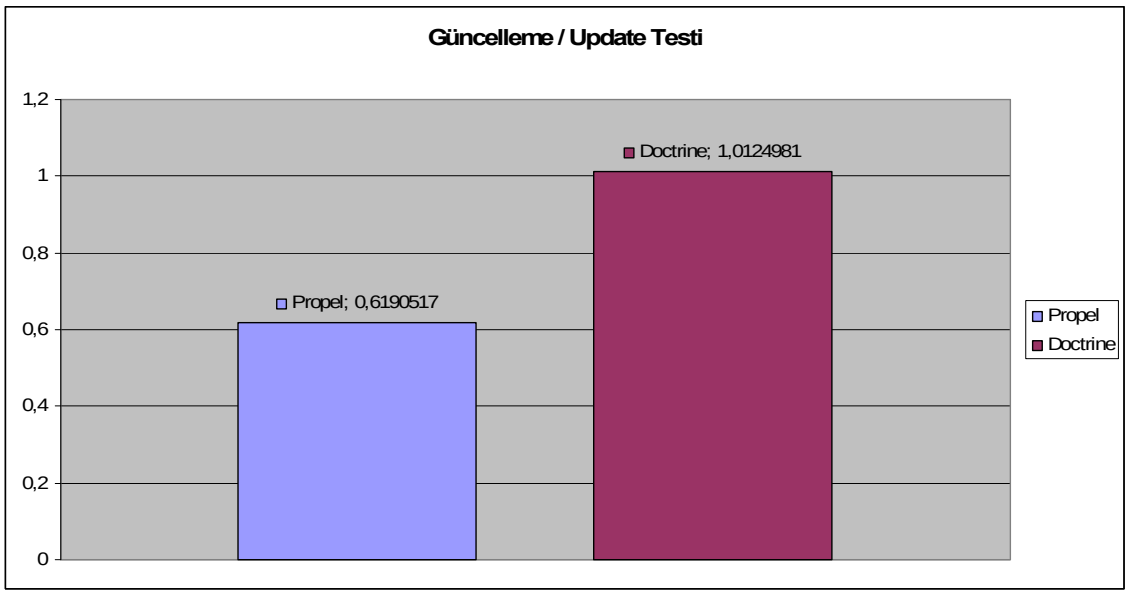
8.2.5.1. Propel Kodu

```
$conn->beginTransaction();

foreach (SaticilarPeer::doSelect(new Criteria()) as $satici) {
   $satici->setIsim('Yeni İsim');
   $satici->save();
}
 $conn->commit();

unset($satici);
SaticilarPeer::clearInstancePool();
```

8.2.5.2. Doctrine Kodu

```
$conn->beginTransaction();

foreach ($conn->getTable('Satici')->findAll() as $satici) {
   $satici->isim = 'Yeni İsim';
   $satici->save();
   $satici->free();
}
 $conn->commit();

unset($satici);
```

Şekil 8.16 incelediğimizde iki farklı ORM platformunda yapılan testler neticesinde Propelin güncelleme işleminde Doctrine’den %163,55 daha hızlı çalıştığı gözlemlenmiştir.



Şekil 8.16 - Güncelleme / Update Testi sonuçları

8.2.6. Silme / Delete Testi:

Bu test ile verinin yok edilmesi yani silme işlemi kontrol edilmiştir. Testte satışlar, satıcılar ve müşteriler tablosunda test amaçlı oluşturulmuş 3000 kayıt silinmiştir.

8.2.6.1. Propel Kodu

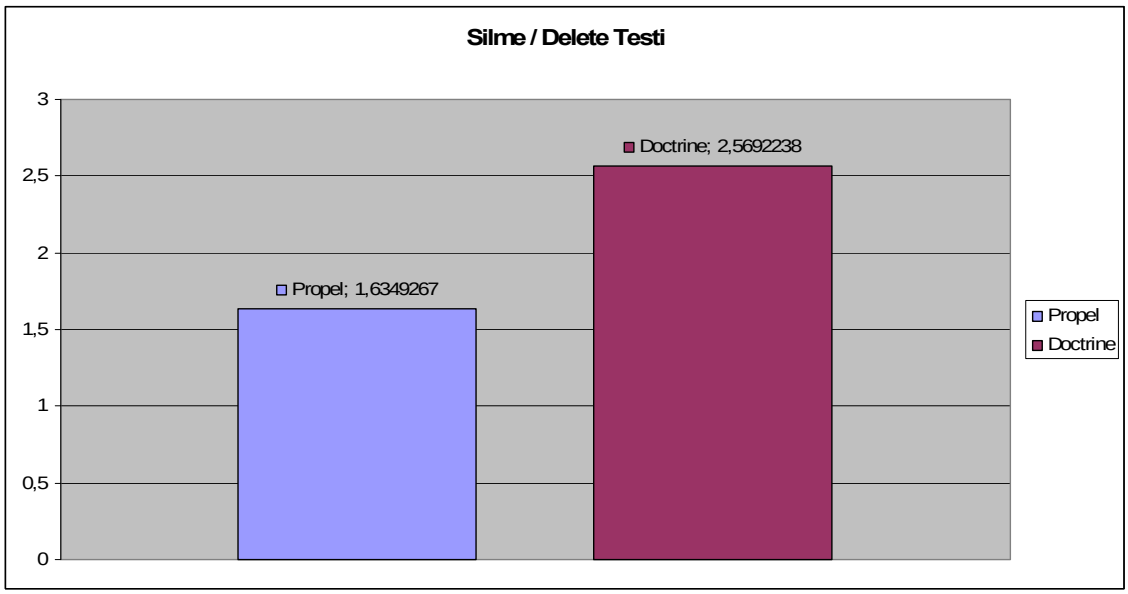
```
$conn->beginTransaction();

foreach (SatislarPeer::doSelect(new Criteria()) as $satis) {
   $satis->delete();
}

foreach (SaticilarPeer::doSelect(new Criteria()) as $satici) {
   $satici->delete();
}

foreach (MusterilerPeer::doSelect(new Criteria()) as $musteri) {
   $musteri->delete();
}

$conn->commit();

unset($satici);
unset($musteri);
unset($satis);
SaticilarPeer::clearInstancePool();
```

```
MusterilerPeer::clearInstancePool();
SatislarPeer::clearInstancePool();
```

8.2.6.2. Doctrine Kodu

```
$conn->beginTransaction();

foreach ($conn->getTable('Satis')->findAll() as $satis) {
   $satis->delete();
   $satis->free();
}
foreach ($conn->getTable('Satici')->findAll() as $satis) {
   $satici->delete();
   $satici->free();
}
foreach ($conn->getTable('Musteri')->findAll() as $musteri) {
   $musteri->delete();
   $musteri->free();
}
 $conn->commit();

unset($satici);
unset($musteri);
unset($satis);
```

Şekil 8.16 incelediğimizde iki farklı ORM platformunda yapılan testler neticesinde Propelin silme işleminde Doctrine'den %157,14 daha hızlı çalıştığı gözlemlenmiştir.



Şekil 8.17 - Silme / Delete Testi sonuçları

8.2.7. Tüm Orm Testlerinin Sonuçları

Testler incelendiğinde Propel'in tüm işlemlerde Doctrine'den performans olarak üstün olduğu görülmektedir. Doctrine platformu henuz yeni gelişiyor olmasına rağmen mimari özellikleri ve yazım kolaylığı bakımından birçok uygulama geliştirici tarafından destekleniyor oluşu ilerleyen dönemde performans ve mimari olarak endüstiriye yeni kazanımlar getireceği düşünülmektedir. Ayrıca Doctrine'deki yazım mantığının HQL yazımına benzemesi, ActiveRecord tasarım örüntüsüne benzerliği Java programlama dilinde Hibernate kullanan uygulama geliştiricilere uyum kolaylığı sağlamaktadır.

Tablo 8.3 - Orm testlerinin sonuçları

	Eklemek / Insert	**Seçme / Select**	**Birleştirerek Seçme**	**Many Selecting**	**Güncelleme / Update**	**Silme / Delete**
Propel	1,85	0,031	0,10	0,49	0,61	1,63
Doctrine	2,79	0,30	0,55	1,738	1,01	2,57
%	%50	%964	%522	%354,1	%163,55	%157,14

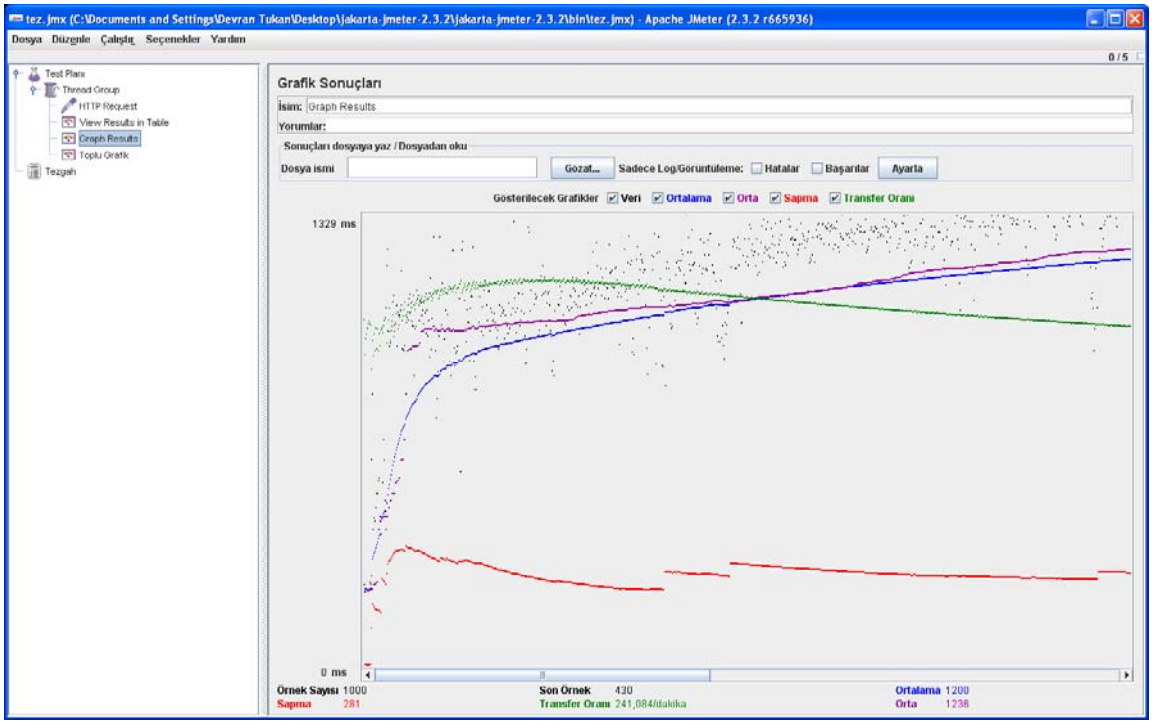
8.3. ÖNBELLEK TESTİ

Önbelleğin performansının analizi için Jmeter ve apachebench kullanılmıştır. Jmeter kullanımında erişim, ortalama erişim ve transfer oranı hakkında platformlar gözlemlenmiş; apachebench kullanımında ise saniyedeki işlem adedi, işlem zamanı ölçümlenmiştir.

8.3.1. Jmeter Testi

8.3.1.1. Apc Kapalı + Cache Kapalı

Bu test sırasında elde edilen değerler 1000 örnek gönderimi sonucu APC kapalı ve uygulama önbelleği kapalı durumdadır.



5 kullanıcı yoğunluğu ve kullanıcı başına 200 örneğin simule edildiği Jmeter testi sonuçlarına göre:

1000 örnek için

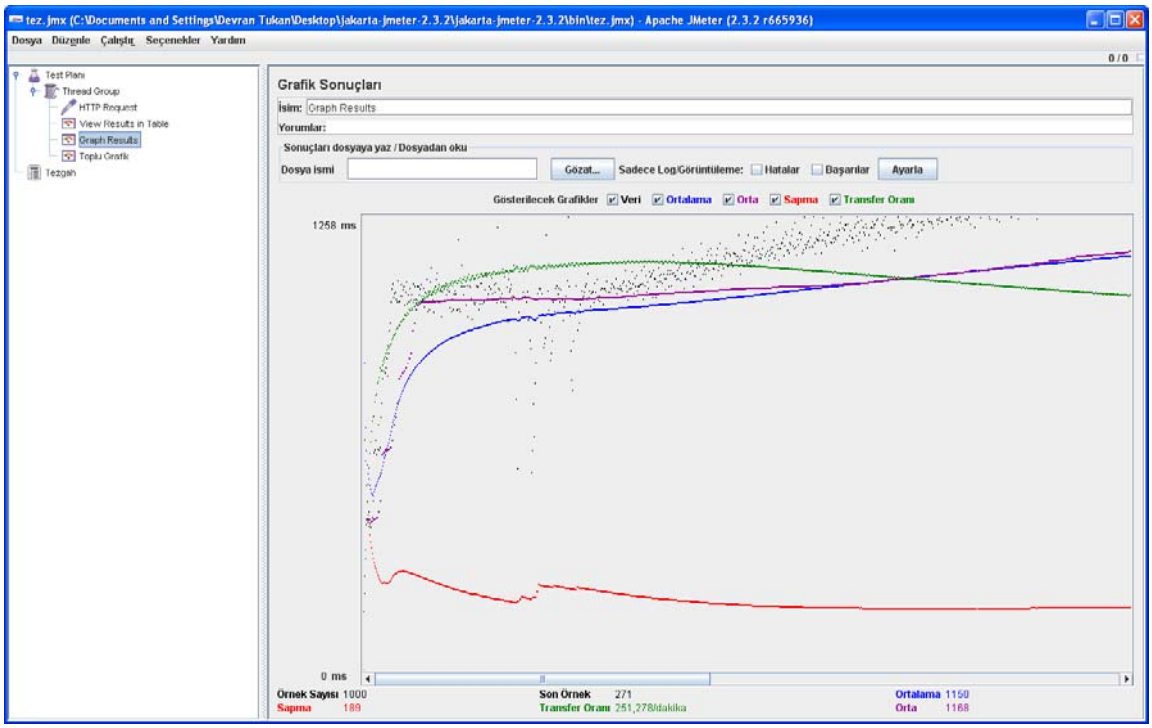
Transfer oranı: 241.084 örnek/dakika

Maximum erişim Süresi: 1329 ms

Ortalama Erişim Süresi: 1200ms

8.3.1.2. Apc Kapalı + Cache Açık

Bu test sırasında elde edilen değerler 1000 örnek gönderimi sonucu APC kapalı ve uygulama önbelleği açık durumdadır.



5 kullanıcı yoğunluğu ve kullanıcı başına 200 örneğin simule edildiği Jmeter testi sonuçlarına göre:

1000 örnek için

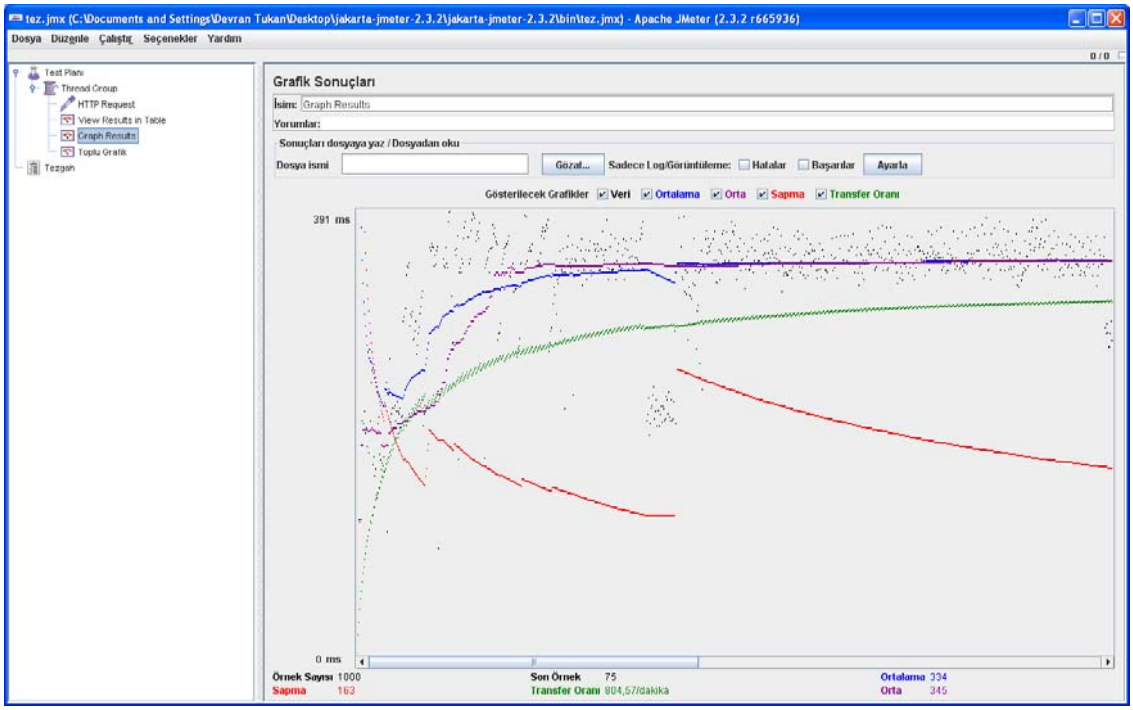
Transfer oranı: 251,278 örnek/dakika

Maximum erişim Süresi: 1258 ms

Ortalama Erişim Süresi: 1150ms

8.3.1.3. Apc Açık + Cache Kapalı

Bu test sırasında elde edilen değerler 1000 örnek gönderimi sonucu APC açık ve uygulama önbelleği kapalı durumdadır.



5 kullanıcı yoğunluğu ve kullanıcı başına 200 örneğin simule edildiği Jmeter testi sonuçlarına göre:

1000 örnek için

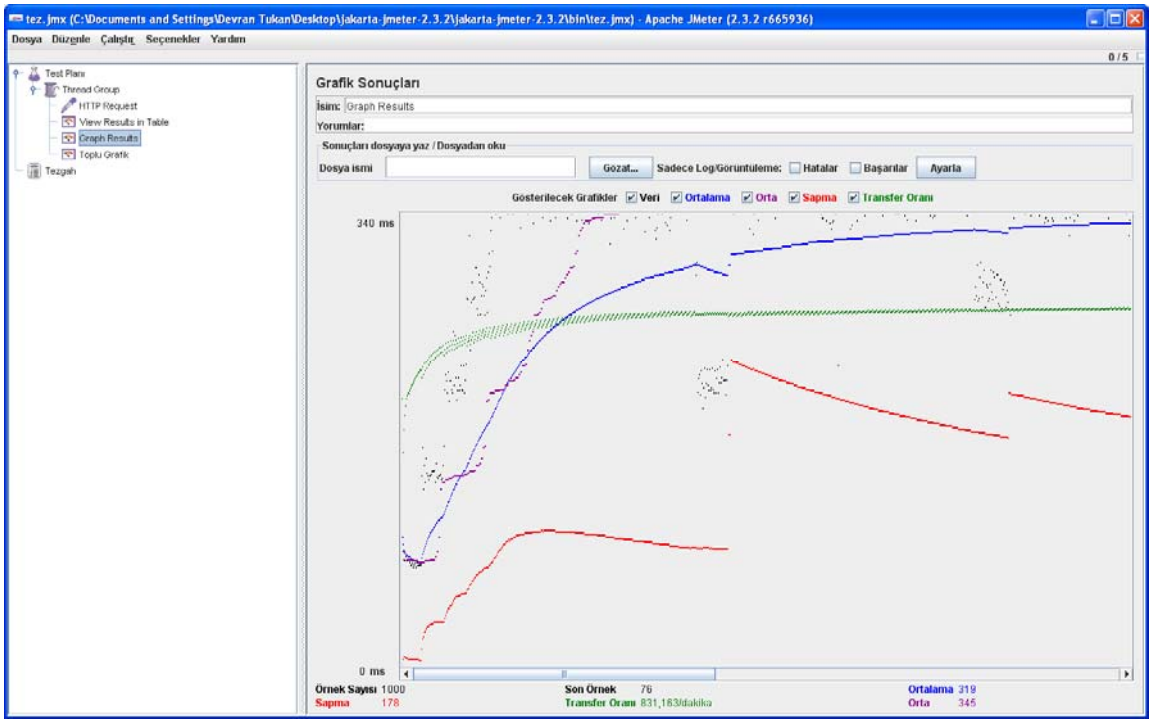
Transfer oranı: 804,57 örnek/dakika

Maximum erişim Süresi: 391 ms

Ortalama Erişim Süresi: 334ms

8.3.1.4. Apc Açık + Cache Açık

Bu test sırasında elde edilen değerler 1000 örnek gönderimi sonucu APC açık ve uygulama önbelleği açık durumdadır.



5 kullanıcı yoğunluğu ve kullanıcı başına 200 örneğin simule edildiği Jmeter testi sonuçlarına göre:

1000 örnek için

Transfer oranı: 804,57 örnek/dakika

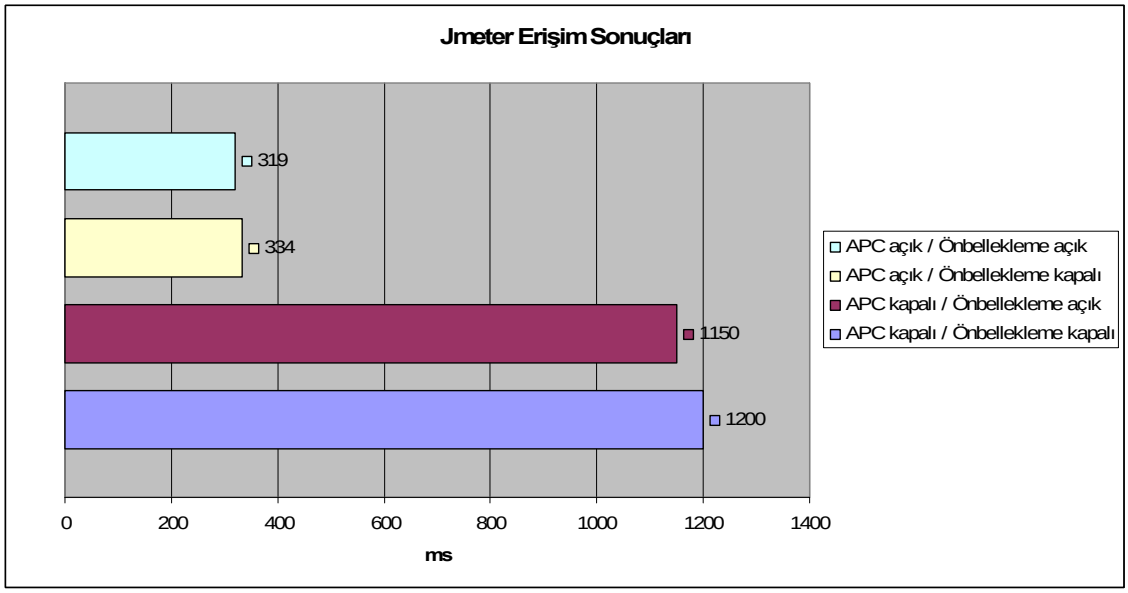
Maximum erişim Süresi: 391 ms

Ortalama Erişim Süresi: 334ms

8.3.1.5. Jmeter Test Sonuçları

Şekil 8.18'de erişim sonuçları incelendiğinde:

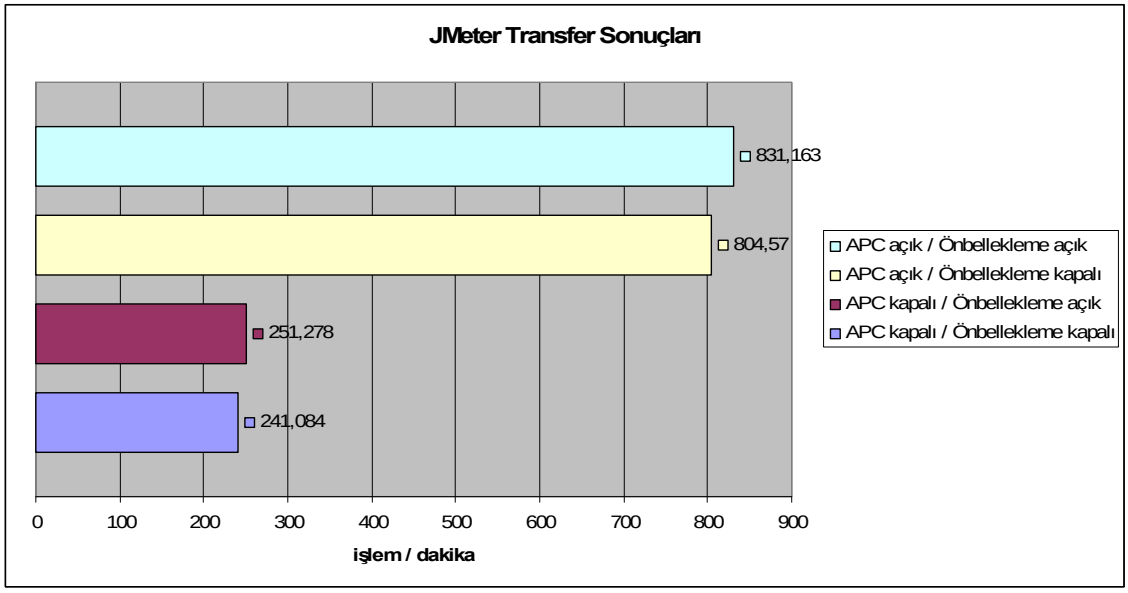
Uygulama seviyesindeki önbellekleme açık ve Apc kapali iken erişim süresinde %4,73 performans artışı gözlemlenmiştir. Apc aktif hale getirildiğinde ise erişim süresindeki performans artışı %4,43 olarak gözlemlenmiştir.



Şekil 8.18 - Jmeter erişim sonuçları

Şekil 8.19'da ise Jmeter transfer sonuçları yeralmaktadır.

Uygulama seviyesindeki önbellekleme açık ve Apc kapali iken dakikadaki örnek transfer sayısı %4,14 olarak gözlemlenmiştir. Apc aktif hale getirildiğinde ise bu süre %3,35 olarak tesbit edilmiştir.



Şekil 8.19 - Jmeter transfer sonuçları

8.3.2. Apachebench testi

Uygulama performansını değerlendirmek açısından yaptığımız Apachebench testinde 2338 byte veri içeren ve XML formatında çıktı veren ve de son girilen 10 haberi gösteren bir aksiyon kullanıldı. 200 örnek paket 5 kullanıcı simule edilerek sunucuya gönderilmiştir.

8.3.2.1.Apc Pasif - Cache Kapalı

Bu testte sunucuda herhangi önbellekleme yapılmadan baz sunucu performansı test edilmiştir.

root@devil:/var/www/devosphere# ab -n 200 -c 5
http://192.168.10.100:333/feed/lastposts
This is ApacheBench, Version 2.0.40-dev <$Revision: 1.146 $> apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright 2006 The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.10.100 (be patient)
Completed 100 requests
Finished 200 requests

Server Software: Apache/2.2.8

```
Server Hostname:        192.168.10.100
Server Port:            333

Document Path:          /feed/lastposts
Document Length:        2338 bytes

Concurrency Level:      5
Time taken for tests:   44.144973 seconds
Complete requests:      200
Failed requests:        0
Write errors:           0
Total transferred:      556000 bytes
HTML transferred:       467600 bytes
Requests per second:    4.53 [#/sec] (mean)
Time per request:       1103.624 [ms] (mean)
Time per request:       220.725 [ms] (mean, across all concurrent requests)
Transfer rate:          12.28 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:   868 1100  53.1   1090    1362
Waiting:      867 1097  53.6   1089    1362
Total:        868 1100  53.1   1090    1362

Percentage of the requests served within a certain time (ms)
  50%   1090
  66%   1119
  75%   1137
  80%   1149
  90%   1170
  95%   1180
  98%   1205
  99%   1246
 100%   1362 (longest request)
```

8.3.2.2. Apc Pasif - Cache Açık

Bu testte kullanılan aksiyon önbelleklenmiş fakat sunucu önbelleklemesi pasif durumdadır.

```
root@devil:/var/www/devosphere# ab -n 200 -c 5
http://192.168.10.100:333/feed/lastposts
This is ApacheBench, Version 2.0.40-dev <$Revision: 1.146 $> apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright 2006 The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking 192.168.10.100 (be patient)
Completed 100 requests
Finished 200 requests


Server Software:        Apache/2.2.8
Server Hostname:        192.168.10.100
Server Port:            333

Document Path:          /feed/lastposts
Document Length:        2338 bytes

Concurrency Level:      5
Time taken for tests:   43.47685 seconds
Complete requests:      200
Failed requests:        0
Write errors:           0
Total transferred:      556000 bytes
HTML transferred:       467600 bytes
Requests per second:    4.65 [#/sec] (mean)
Time per request:       1076.192 [ms] (mean)
Time per request:       215.238 [ms] (mean, across all concurrent requests)
Transfer rate:          12.59 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   7.5      0      78
Processing:   680 1071  43.9   1076    1259
Waiting:      679 1067  43.6   1071    1246
Total:        680 1072  43.8   1076    1259

Percentage of the requests served within a certain time (ms)
  50%   1076
  66%   1089
  75%   1090
  80%   1096
  90%   1109
  95%   1119
  98%   1140
  99%   1200
 100%   1259 (longest request)
```

8.3.2.3. Apc Aktif - Cache Kapalı

Bu testte kullanılan sunucu önbelleklemesi aktif hale getirilmiş fakat kod üzerinde herhangi bir önbellekleme yapılmamıştır, pasif durumdadır.

```
root@devil:/var/www/devosphere# ab -n 200 -c 5
http://192.168.10.100:333/feed/lastposts
This is ApacheBench, Version 2.0.40-dev <$Revision: 1.146 $> apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright 2006 The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.10.100 (be patient)
Completed 100 requests
Finished 200 requests

Server Software:        Apache/2.2.8
Server Hostname:        192.168.10.100
Server Port:            333

Document Path:          /feed/lastposts
Document Length:        364 bytes

Concurrency Level:      5
Time taken for tests:   19.454406 seconds
Complete requests:      200
Failed requests:        199
   (Connect: 0, Length: 199, Exceptions: 0)
Write errors:           0
Total transferred:      551032 bytes
HTML transferred:       463288 bytes
Requests per second:    10.28 [#/sec] (mean)
Time per request:       486.360 [ms] (mean)
Time per request:       97.272 [ms] (mean, across all concurrent requests)
Transfer rate:          27.65 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:   119  484 853.9    328    6534
Waiting:        0  459 816.6    320    6503
Total:        119  484 853.9    328    6534

Percentage of the requests served within a certain time (ms)
  50%    328
  66%    329
  75%    337
  80%    340
  90%    360
  95%    619
  98%   3976
  99%   6474
 100%   6534 (longest request)
```

8.3.2.4. Apc Aktif - Cache Açık

Bu testte kullanılan sunucu önbelleklemesi aktif hale getirilmiş; kod üzerinde önbellekleme yapılmıştır, Çerçeve model kullanımındaki optimum neticeler gözlemlenmiştir.

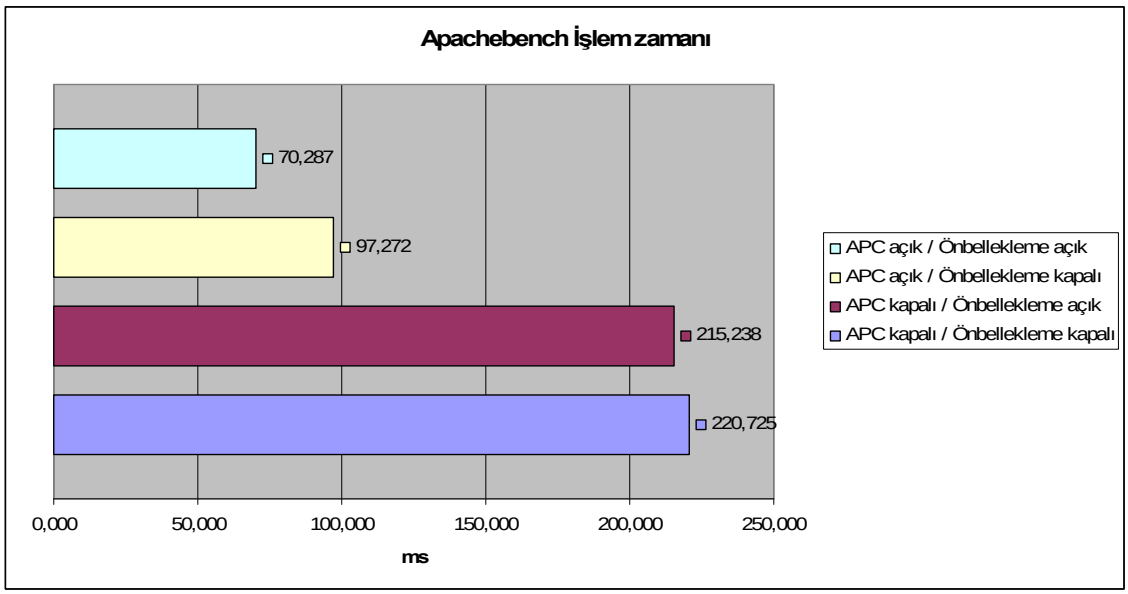
```
root@devil:/var/www/devosphere# ab -n 200 -c 5
http://192.168.10.100:333/feed/lastposts
This is ApacheBench, Version 2.0.40-dev <$Revision: 1.146 $> apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright 2006 The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.10.100 (be patient)
Completed 100 requests
Finished 200 requests


Server Software:        Apache/2.2.8
Server Hostname:        192.168.10.100
Server Port:            333

Document Path:          /feed/lastposts
Document Length:        2338 bytes

Concurrency Level:      5
Time taken for tests:   14.57341 seconds
Complete requests:      200
Failed requests:        0
Write errors:           0
Total transferred:      556000 bytes
HTML transferred:       467600 bytes
Requests per second:    14.23 [#/sec] (mean)
Time per request:       351.434 [ms] (mean)
Time per request:       70.287 [ms] (mean, across all concurrent requests)
Transfer rate:          38.56 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   7.6      0      79
Processing:   151  348 123.3    328    1120
Waiting:      151  343 120.5    324    1100
Total:        151  349 129.1    328    1140

Percentage of the requests served within a certain time (ms)
  50%    328
  66%    330
```

75% 336
80% 339
90% 367
95% 480
98% 1101
99% 1130
100% 1140 (longest request)

8.3.2.4. Apachebench Sonuçları

ab -n 200 -c 5 http://192.168.10.100:333/feed/lastposts komutu çalıştırılarak yapılan testleri neticeleri incelendiğinde (Şekil 2.20 ve Şekil 8.21)

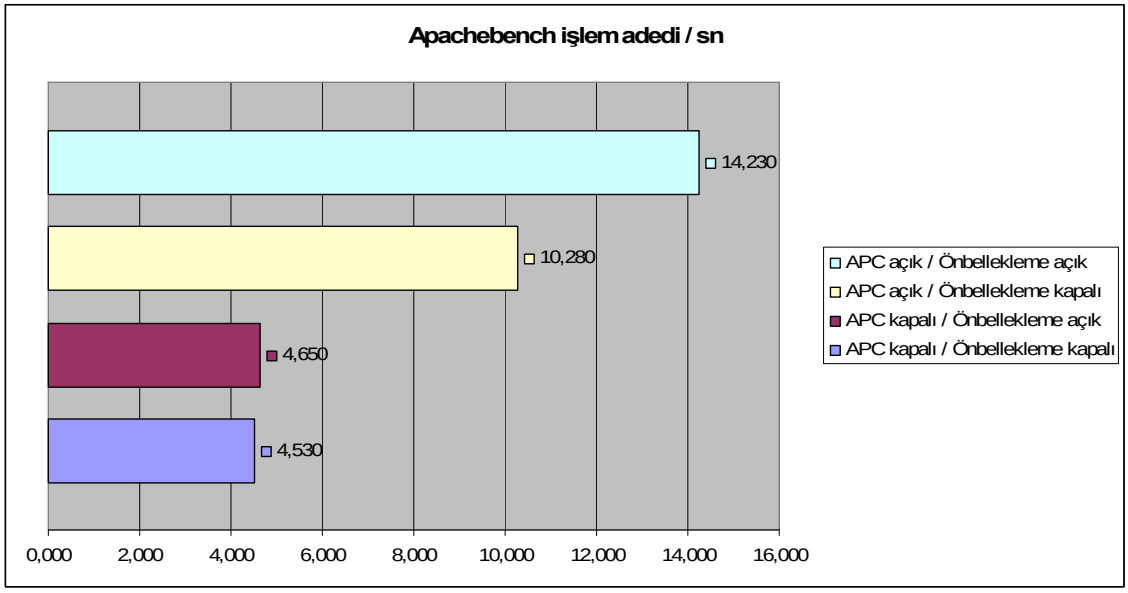
Uygulama seviyesindeki önbellekleme açık ve Apc kapali iken işlem zamanında %2,5 performans artışı gözlemlenmiştir. Apc aktif hale getirildiğinde ise işlem zamanındaki performans artışı %28 olarak gözlemlenmiştir



Şekil 8.20 - Apachebench işlem zamanı

Şekil 8.21'de ise Apachebench işlem adeti /sn testi sonuçları yeralmaktadır.

Uygulama seviyesindeki önbellekleme açık ve Apc kapalı iken saniyedeki işlem sayısındaki artış %2,64 olarak gözlemlenmiştir. Apc aktif hale getirildiğinde ise bu artış %38,42 olarak tesbit edilmiştir.



Şekil 8.21 - Apachebench işlem adedi / sn

Testlerin sonuçları göstermektedir ki yazılım çerçeve modellerinin kullanımı hem bir web uygulamasının performansına direk etki eden konularda (Veri erişimi, sunucu yükü dağıtımı, önbellekleme) birbirleri ile uyumlu komponentler kullanılması ile optimum sistemleri uygulama geliştiricilere hazır olarak sunmaktadır. Bu tarz sistemlerin kurulumu ve konfigurasyon kolaylığı sayesinde proje yönetimine önemli hızlar kazandırmaktadır. Tamamen çerçeve modellere yönelik gerek geliştiricileri gerek kullanıcı katılımı sayesinde geliştirilmiş ve dökümante edilmiş plug-in ler sayesinde uygulama geliştirme süresi oldukça hızlanmakta, geliştirme maliyetleri kısılmaktadır. Örnek olarak ileri derecede güvenli Kullanıcı, grup, izin bazlı bir kullanıcı yönetim sistemi ortalama iki gün içerisinde uygulanabilmektedir. Yazılım çerçeve modelleri oldukça kolay okunabilir, kolay düzenlenebilir, tekrardan kullanılabilir bir kod sistematiğine sahiptir. Yazılmış fonksiyonlar ve kütüphaneler daha sonraki projelerde sorunsuzca kullanılabilmektedir. Bu özelliği ile geliştiriciler agile development metodolojilerini (DRY,KISS veya XP) uygulayabilmektedir. Standart SDLC metodolojisi kullanılan projeler süre uzunluğu, kaynak gereksinimi ve yönetimsel eksiklilerden dolayı ülkemizde ancak %50 başarı ve memnuniyet ile tamamlanabilmektedir. Avrupa ve Amerika'daki büyük şirketler de bu dönemi yaşamışlardır. Daha verimli projeler üretmek üzere çeşitli yöntemler denemişler ve çoğu şirket yönetimde ve uygulamada en başarılı buldukları "Agile" (çevik) yazılım

metodolojisini benimsemişlerdir. Bu metodoloji sayesinde, artan verimlilik ve esneklik doğrultusunda projelerin kalitesi artmış ve başarı oranlarını %80' lere çıkartmayı başarmışlar ve ayrıca endüstrinin ihtiyaçlarına hızlı bir şekilde cevap verebilir olmuşlardır.

SONUÇLAR VE TARTIŞMA

Çerçeve modellerin kullanımı uygulama geliştiriciliğin en iyi örneklerini kullandıklarından ve hızlı uygulama geliştirme tekniklerini (RAD) desteklediklerinden Proje yönetimine önemli kolaylık katmaktadır. Uygulamalarda kaynakların doğru yönetilip analiz edilmesi yazılım çerçeve modelleri kullanılarak mümkün olabilmekte ve harici komponentlerin entegre edilmeye hazır bekliyor oluşu proje yönetiminin yükünü hafifletmektedir.

Çerçeve modeller kodda standartlık sunar. Standartlara uygun olarak yazılmış kodlar tekrardan kullanılabilir niteliğe sahip olmaktadır. Bu niteliğin efektif kullanımı proje süresini kısaltmakta ve yazılımın maliyetinde önemli bir düşüşe yol açmaktadır.

Çerçeve modellerin gelişimi ile güvenlik, performans, ajax kullanımı gibi uygulama geliştirmenin en can alıcı kısımlarında uygulama geliştiricilere önemli kazanımları beraberinde getirmektedir.

Bölüm 5 de Symfony ile Orm kullanımı incelenmiş ve SQL sorguları ile düzenlenen test platformlarında testler yapılmıştır. ORM ile veri alışverişi incelendiğinde uygulamalarında soyutlama katmanı kullanmanın önemi belirtilmiştir. Uygulama geliştirmede artık veritabanı seçimi ve uygulamaların ilerideki veritabanı değişiklerinde karşılasacağı güçlükler Symfony'de ORM kullanımı ile giderilmiştir. Artık tamamen veri tabanı bağımsız uygulamalar geliştirilebilmektedir.

ORM veri erişiminde uygulama geliştiricilere önemli katkılar sunmaktadır. Nesne tabanlı erişim ile web uygulamasındaki güvenlik noktaları katmanlanarak güvenlik alt yapısının kurulması basite indirgenmiş. Ve kod sayısında önemli bir azalmayı beraberinde getirmektedir.

Symfony'deki iki ORM sistemi incelendiğinde Propel'in geliştiricilere Doctrine oranla daha fazla performans sağladığı test sonuçlarında gözlemlenmiştir. Fakat Doctrine Aktif Kayıt tasarım örüntüsünün kullanımı ile uyumu nedeniyle birçok uygulama geliştirici

tarafından desteklenmektedir. Yakın gelecekte performans sorunları giderildiğinde Propel'e ciddi bir rakip olabilecektir.

Bu çalışmada web uygulamalarında önbellek kullanımının önemi hakkında çalışmalar yapılmıştır. Bilindiği üzere önbellekleme yerinde kullanıldığında sunucu üzerindeki yoğunluğu azaltmaktadır. Fakat uygulama geliştirme sırasında önbellekleme ile büyük problemlerle karşılabilmektedir. Kompleks uygulamalarda efektif önbellekleme kullanımı karmaşık yapılara geçilmesine, üretilen kodun okunabilirlik ve karmaşıklığını arttırmaktadır. Çalışmada Symfony ile standart Php ile karşılaşılabilecek problemlerle nasıl çözüme ulaşıldığı örnek uygulamalarla gösterilmiştir. Symfony önbellekleme sistemini modul ve aksiyon bazlı yapılabilmesine ve harici ayar dosyasında yml formatında saklayabilmesi önbellekleme yönetimini kolay yapılandırılabilir kılmaktadır.

Önbellekleme testlerinin sonuçlarına göre, uygulama seviyesindeki önbellekleme açık ve Apc kapalı iken saniyedeki işlem sayısındaki artış %2,64 olarak gözlemlenmiştir. Apc aktif hale getirildiğinde ise bu artış %38,42 olarak tesbit edilmiştir.

Uygulama seviyesindeki önbellekleme açık ve Apc kapali iken işlem zamanında %2,5 performans artışı gözlemlenmiştir. Apc aktif hale getirildiğinde ise işlem zamanındaki performans artışı %28 olarak gözlemlenmiştir

Basit yapılandırma ayarları yardımı ile Symfony'de bir aksiyonda önbellekleme kullanımı yaklaşık %5 performans artışı elde edilmiştir. Bununla beraber symfony ile uyumlu APC önbelleklemenin aktif hale getirilmesi ile ve daha karmaşık önbellekleme sistemleri kullanımı ile birlikte kazanım miktarının işlem zamanında %312 gözlemlenmiştir. Test platformunda elde edilen sonuçların yerel bir ağ üzerinden yapıldığı ve gerçek internet ortamında kazanımların daha da artacağı göz önünde bulundurulmalıdır.

Bölüm 4'de ise standart ajax uygulamalarında Symfony kullanımının kodun kolaylık seviyesinde ve okunabilirliğinde nasıl artışa sebebiyet verildiği gösterilmiştir. Symfony'de ki Ajax kullanım kolaylığı günümüz zengin internet uygulamalarında

çerçeve model kullanımının başlıca sebeblerden biri olmuştur. İnternetin kurulduğu günden bu güne bir internet devi olan yahoo.com günümüzün zengin internet uygulamalarında örnek niteliği taşıyan flickr.com ve delicious.com'u Symfony ile geliştirmiş ve yeni nesil uygulamalarını Symfony ile geliştireceğini duyurmuştur. Prototype ve script.aculo.us 'in başarılı kod yardımcıları eşliğinde uygulanması uygulama geliştiricilere zor uygulamaları rahatlıkla yapabilmelerine olanak tanımaktadır.

Symfony Mvc katmanlaması ile XSS/CSRF ve Sql Injection'dan korunmaktadır. Ve sade yazılmış yapılandırma dosyaları yardımı ile veritabani ile kullanıcılar arasındaki iletişim kolaylıkla denetlenebilmektedir.

Sonuç olarak, Çerçeve Modeller web yazılımının kalitesini arttıran bir çok unsuru entegre edilmiş halde barındıran ve üretilen yazılımın kalitesini uygulama geliştiricilere belli bir çerçevenin sınırları içerisinde kalmayı sağlayarak zorluk seviyesi yüksek kurumsal uygulamaları yalın, sade, anlaşılabilir kod kullanımı ile yazılmasını sağlayarak uygulamaya değer katmaktadır.

KAYNAKÇA

Anh, N. & Guarnieri, F. & Greene, D. & Shirley J. & Evans, F., *Automatically Hardening Web Applications Using Precise Tainting*, Department of Computer Science, University of Virginia, 151 Engineer's Way, Charlottesville, VA 22904-4740, USA

Da Silva, E.Q. & de Abreu Moreira, D., *Developing customizable Web-based educational applications through a component-based framework*

Deacon, J., *Model-View-Controller (MVC) Architecture*

Eiiti, H., *MVC dance: connecting software development and corporeality from agile process and pattern language perspectives*
Creating, Connecting and Collaborating through Computing, 2004. Proceedings. Second International Conference on
29-30 Jan. 2004 Page(s):174 - 176
Digital Object Identifier 10.1109/C5.2004.1314388

Fowler, M., *Patterns of Enterprise Application Architecture*, Publisher: Addison-Wesley Professional; 1st edition (November 5, 2002) Language: English ISBN-10: 0321127420

Gallego-Carrillo, M. & García-Alcaide, I., Montalvo-Herranz, S., *Applying hierarchical mvc architecture to high interactive web applications.*

Gan, Z. & Wei, D. & Varadharajan, V., *Evaluating the performance and scalability of Web application systems*
Information Technology and Applications, 2005. ICITA 2005. Third International Conference on
Volume 1, 4-7 July 2005 Page(s):111 - 114 vol.1
Digital Object Identifier 10.1109/ICITA.2005.131

Holz, T. & Marechal, S. & Raynal, F., *New threats and attacks on the World Wide Web* Security & Privacy Magazine, IEEE
Volume 4, Issue 2, March-April 2006 Page(s):72 - 75
Digital Object Identifier 10.1109/MSP.2006.46

Leff, A. & Rayfield, J., *Web-Application Development Using the ModelViewController Design Pattern*, IBM T. J. Watson Research Center
Livshits, B. & Erlingsson, U., Using *Web Application Construction Frameworks to Protect Against Code Injection Attacks,* Microsoft Research
Nenad, J. & Kırda, E. & Kruegel, C., *Preventing Cross Site Request Forgery Attacks*

Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on
22-26 Aug. 2005 Page(s):6 pp.
Digital Object Identifier 10.1109/NWESP.2005.29
Orenstein, J., *Supporting Retrievals and Updates in/an Object relational mapping system*, Novera Software

OWASP, *http://www.owasp.org/asac/input_validation/sql.shtml*

Ping, Y. & Kontogiannis, K. & Lau, K., *Transforming legacy Web applications to the MVC architecture*, Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on, 19-21 Sept. 2003, Page(s): 133 - 142
Digital Object Identifier 10.1109/STEP.2003.35

Potencier, F. & Zaninotto, F. *Definitive guide to Symfony* Publisher: Apress (January 22, 2007) Language: English ISBN-10: 1590597869 ISBN-13: 978-1590597866

Pree , W., *Design Patterns for Object-Oriented Software Development, Johannes Kepler University Linz* ,Hermann Sikora RACON Software Inc.
Redol, J. & Simoes, D. & Carvalho, A. & Pascoa, H. & Coelho, J. & Grave, P. Luis, R. & Horta, N. *VIANET-a new Web framework for distance learning*

Advanced Learning Technologies, 2003. Proceedings. The 3rd IEEE International Conference on 9-11 July 2003 Page(s):258 – 259

Securecomm and Workshops, 2006
Aug. 28 2006-Sept. 1 2006 Page(s):1 - 10
Digital Object Identifier 10.1109/SECCOMW.2006.359531
Selfa, D. & Carrillo, M., Del Rocio Boone, M., *A Database and Web Application Based on MVC Architecture* Electronics, Communications and Computers, 2006. CONIELECOMP 2006. 16th International Conference on
27-01 Feb. 2006 Page(s):48 - 48
Digital Object Identifier 10.1109/CONIELECOMP.2006.6

Shan, T. C. & Hua W. W. , *Taxonomy of Java Web Application Frameworks*, e-Business Engineering, 2006. ICEBE '06.

SPI Dynamics Sql Injection Whitepaper,
http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf

SQL Injection In SQL Server Applications,
http://www.nextgenss.com/papers/advanced_sql_injection.pdf

SQLSecurity.com, *http://www.sqlsecurity.com/faq-inj.asp*
Tao, Y., *Component- vs. application-level MVC architecture*
Frontiers in Education, 2002. FIE 2002. 32nd Annual
Volume 1, 6-9 Nov. 2002 Page(s):T2G-7 - T2G-10 vol.1
Digital Object Identifier 10.1109/FIE.2002.1157950

Wojciechowski, J. & Sakowicz, B. & Dura, K. & Napieralski, A., *MVC model, struts framework and file upload issues in web applications based on J2EE platform*, Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004. Proceedings of the International Conference

24-28 Feb. 2004 Page(s):342 - 345
Digital Object Identifier 10.1109/TCSET.2004.1365980