UML ile Sistem Analizi Serisi:

[UML ile Sistem Analizi 1 – Giris](http://www.gelistir.org/blog/uml-ile-sistem-analizi-1)   
[UML ile Sistem Analizi 2 – Gereksinimler](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-2-gereksinimler)   
[UML ile Sistem Analizi 3 – Use Case Modelleme](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-3-use-case-modelleme)  
[UML ile Sistem Analizi 4 – Ekran Prototipleri (Mockup)](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-4-ekran-prototipleri-mockup)   
[UML ile Sistem Analizi 5 – Domain Modelleme](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-5-domain-modelleme)  
[UML ile Sistem Analizi 6 – Sequence Diyagramlar](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-6-sequence-diyagramlari) [UML ile Sistem Analizi 7 – Sonuç](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-7-sonuc)

Bir bakıyorsunuz ki en son yazdığınız yazıdan uzunca bir süre geçmiş! Vakit öyle hızlı ilerliyor ki bazen, yaptığınız şeylerle odaklanmış hayatınızı devam ettirirken elinizin altından geçen “asıl yapılması” gerekenleri unutuveriyorsunuz.

İşte tam bu anda “Asıl yapılması” gerekenleri konu edinen bir uygulama geliştirsek ve bunu da yazıya döksek nasıl olur diye düşündüm. Bu yazı dizisinde uygulama geliştirmek yerine sadece yazılım-sistem analizi ile ilgili detaylı bilgiler bulunacak. Belki, geliştirme süreci ayrı bir yazı dizisi olarak ileride açıklanabilir.

Uygulama oldukça basit olacak. Kişinin yapılacaklar listesini tutan bir görev listesi uygulaması…

İlk olarak, uygulama gereksinimleri (requirements) belirlenecek. Bu gereksinimleri belirlerken use case’ler oluşturulup bu case’lerin gereksinimlerle olan ilişkilerini kuracağız. Ekran prototipleri tasarlayacağız ve görsel olarak istenen özellikleri daha net ifade edebileceğiz. Ardından sistemin domain modelleri belirlenecek. Son olarak Sequence çizimlerini de oluşturarak bu sınıflar arasındaki ilişkileri dinamik bir şekilde inceleyeceğiz.

Profesyonel bir uygulama geliştirirken muhtemelen çok daha ayrıntılı süreçler, teknikler gerekir. Birçok kişinin aynı işte çalışması gerekir. Bazı işlerin ise paralel yürütülmesi gerekir. İş analistlerinden saha uzmanlarına (domain experts), proje yöneticilerinden yazılım mühendisleri ve bilgisayar mühendislerine, testçilerden programcılara kadar uzanan bir yelpazede teknik ekip gerekebilir. Aynı şekilde proje sahipleri, diğer yöneticiler kısacası doğrudan ya da dolaylı yoldan sistemden etkilenecek olan kişiler ve organizasyonlar (paydaşlar-stakeholder) sistemin geliştirilme sürecine bir şekilde katkıda bulunurlar.

Bu uygulama ise sadece bir kişi tarafından geliştirilecek. Proje sahibi de, projeyi analiz edip geliştirecek kişi de ben olduğum için, takdir edersiniz ki çok fazla profesyonellik beklememek lazım!

Ayrıca, tüm bu süreçlerde kullanılacak tekniklerin tam olarak bir standardı olmadığı için, projeden projeye, hatta kişiden kişiye bile değişiklik gösterebilir. O yüzden burada kullanılan teknikler herkese göre doğru olamayacağı gibi tam olarak yanlış da kabul edilmemelidir.

Samimi olması açısından uygulamamıza bir de isim verelim. Yapılacak listesi uygulaması dediğimize göre yaratıcılığımızın da sınırlarını zorlayarak “YapLIST” olsun diyelim :)

Sonraki yazıda, “YapLIST” uygulamamızın kapsamı ve neleri gerçekleştirmesi gerektiği, yani gereksinimleri belirleyeceğiz. Ayrıca bir proje planı da oluşturup, yukarıdaki aşamaları da kendi içinde alt aşamalara böleceğiz.

Bu süreçte, istediğiniz gibi yorumda bulunabilir, katkı yapabilirsiniz.

**Not:** Yazılarda, çeşitli UML (Unified Modeling Language) diyagramları kullanılmaktadır. Yazılarda kullanılan UML ile ilgili kısa bir açıklama verilmesine rağmen, bazı diyagramları yorumlarken daha detaylı bir UML bilgisine ihtiyaç duyulabilir.

Son olarak UML ile ilgili birkaç kaynak verelim (Linkler yeni sayfada açılır):

Önceki bölümde, geliştirilecek uygulama ile ilgili ufak bir giriş yapılmıştı. Bu bölümde ise uygulamamızın neleri gerçekleştireceğini, hangi özellikleri olacağını belirleyeceğiz. Kısacası uygulamamızın gereksinimlerini netleştireceğiz.

Gereksinimler belirlenirken, müşteriler ve ilgili sahada uzman kişilerle beraber çalışılması gerekebilir. Bu kişilerle beraber, sistemin hangi özelliklere sahip olması gerektiği belirlenmelidir. Birlikte çalışan insanlar farklı özelliklere ve bakış açılarına sahip oldukları için, herkesin ortak anlayabileceği bir tarzda iletişim kurmak önemlidir. Sade, basit diyagramlar (iş akış diyagramları, use case diyagramları) kullanmak bir seçenek olabilir. Yine benzer şekilde, sistemin ekran görüntülerinin basit bir şekilde (mockup – wireframing) tasarlanması da müşterinin, uygulamanın neye benzeyeceğini ile ilgili fikir sahibi olması bakımından oldukça işe yarayabilir.

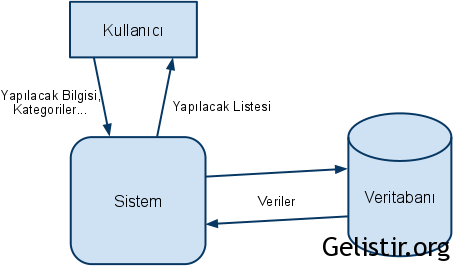
Gereksinimler belirlendikçe, günlük kullanılan dil ile liste şekilde yazmak gerekiyor.

Projemizin çıkış noktası ise bu gereksinim listesidir. Aşağıda projemizin hangi gereksinimleri karşılaması gerektiği listelenmiştir.

1. REQ01 Kayıtlı Olmayan Kişi Sisteme Kayıt Olabilmelidir
2. REQ02 Kullanıcı Sisteme Giriş Yapabilmelidir
3. REQ03 Kullanıcı Çıkış Yapabilmelidir
4. REQ04 Kullanıcı Yapılacak Ekleyebilmelidir
5. REQ05 Kullanıcı Yapılacak Değiştirebilmelidir
6. REQ06 Kullanıcı Yapılacak Silebilmelidir
7. REQ07 Kullanıcı Yapılacakları Listeleyebilmelidir
8. REQ08 Kullanıcı Yapılacakları Arayabilmelidir
9. REQ09 Kullanıcı Yapılacak Listesini Sıralayabilmelidir
10. REQ10 Kullanıcı Kategorileri Yönetebilmelidir
11. REQ11 Kullanıcı Yapılacak’a Kategoriler Ekleyebilmelidir
12. REQ12 Kullanıcı Yapılacak’a Tarih Ekleyebilmelidir
13. REQ13 Kullanıcı Yapılacak’a Öncelik Tanımlayabilmelidir

Her bir gereksinimi belirlemek için basit bir isimlendirme yaptık. Tüm gereksinimler REQ (requirement = gereksinim) ile başlamaktadır. Daha sonra numara verilmiştir. Dikkat edersiniz her bir gereksinimde “meli, malı” ekleri vardır. Sistemimiz bu özelliklere sahip olmalıdır ya da başka bir şekilde açıklayacak olursak; “bu işlemler gereklidir ve geliştireceğimiz uygulama bu işlemleri gerçekleştirebilmelidir” şeklinde yorumlayabiliriz. Bir başka ifade de “bunlar sistem tarafından karşılanmalıdır” olabilir. İngilizcede de gerekesinim raporlarında “meli, malı” anlamında genellikle “shall” kelimesi kullanılmaktadır.

Uzunca tartışmalardan (!) sonra sistemi, bir context diyagramı ile şu şekilde özetleyebiliriz:

[](http://www.gelistir.org/wp-content/uploads/2011/01/context2.png)

Görüldüğü gibi sadece tek çeşit bir kullanıcı var uygulamamızı kullanacak. Bu kişi uygulamamız yardımıyla yapılacaklarını (todo) tanımlayacak, oluşturduğu bu yapılacakları listeleyip, düzenleyebilecek. Kayıtlar bir veritabanında tutulacak.

Aslında bizim sistemimizin gereksinimleri fonsiyonel gereksinimler (functinonal requirements) kategorisi adı altında değerlendirilmelidir. Fonksiyonel gereksinimlerine ek olarak sistem gereksinimleri (non-functional requirements) de gereksinim analizinin olmazsa olmazıdır.

Sistem gereksinimleri, güvenlik, performans, sorunların giderilmesi süresi, kullanım kolaylığı (daha fazla örnek için [şu sayfaya](http://en.wikipedia.org/wiki/Non-functional_requirement) bakabilirsiniz – İngilizce) gibi birçok özelliğin somut bir şekilde belirtilmesini gerektirir. Sistem gereksinimleri proje alıcısının ihtiyaçlarına göre belirlenir. Ayrıca sistem gereksinim konuları geliştirilecek olan uygulamaya göre değişiklik gösterirler. Örneğin, bir bankacılık uygulamasında güvenlik çok daha ön plandayken performans nispeten daha az önemli kabul edilebilir. Yine bir uçak yazılımında en önemli kriter sistemin tepki süresi olabilir. İdealde tüm sistemlerin hızlı, güvenli, hiçbir sorun çıkarmaması istenir ama bu gereksinimlerin her birinin iş gücü ve zaman maliyeti getirdiği de unutulmamalıdır.

Bizim uygulamamızda sistem gereksinimleri açısından herhangi bir gereklilik olmayacak. Örnek vermek açısından şu tip gereksinimlerini karşılamamız gerekebilirdi:

1. Aynı anda 100 kişi sistemi sorunsuzca kullanabilmelidir.  
2. Herhangi bir sorun olduğunda bu sorunun ciddiyetine göre: A seviyesi için 3 gün, B seviyesi için 2 gün, C seviyesi için 1 gün içinde sorun giderilmelidir.  
3. Yapılacakları, sadece kişinin kendisi görüntüleyebilmelidir.  
4. …

Bir proje geliştirme sürecinde, gereksinimler tek bir seferde belirlenemez ve sürekli değişiklik gösterirler. Yazılım geliştirme sürecindeki başarısızlıkların en büyük sebeplerinden ikisinin gereksinimlerin doğru belirlenememesi ve iletişim problemleri olduğunu herkes söylemektedir. Sistemin değişen taleplere ve gereksinimlere göre daha dayanıklı olması için esnek tasarlanması gerekir. Hızlı bir şekilde minimum işleve sahip; fakat çalışan bir sistem geliştirip, müşteriyle sürekli veri alışverişi içerisinde olmak, projenin anahtarının tesliminin yapılabilmesi için oldukça önemlidir. Zaten sıklıkla adı geçen çeşitli çevik (Agile) yazılım geliştirme tekniklerinin özünde de bu vardır.

Sonraki yazıda, use case diyagramı oluşturulacak ve bu use caselerin gereksinimlerimizi nasıl karşılayacağını belirleyeceğiz.

Bir önceki yazımızda sistem gereksinimlerinden bahsetmiştik. Bu yazıda çözümlememize Use Case’ler (kullanım durumları) ile kaldığımız yerden devam edeceğiz.

**Use Case Nedir?**

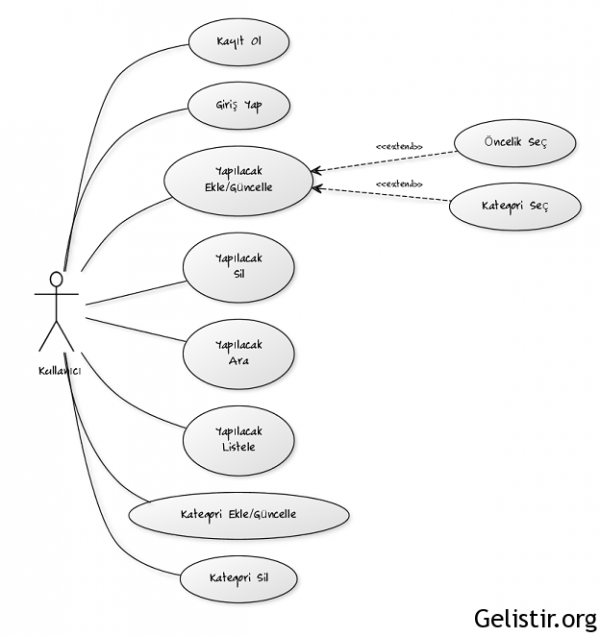
Use case ya da Türkçe’de “Kullanım Durumu” diye ifade edilen kavram, sistemin sahip olması gereken ve etkileşime girildiğinde dış dünyaya sağladığı işlevsellik olarak tanımlanabilir. Bir anlamda sistemimizin amaçlarını belirlememize yardımcı olur.

Use case’ler, diyagram ve tablo olmak üzere iki biçimde gösterilirler. Diyagram versiyonu standard UML diyagramlarından birisi olup, görece daha az ayrıntıyla ilgilenir ve farklı teknik bilgi düzeyine sahip kişilerin sistemi anlamasına yardımcı olur. Yazı versiyonu ise daha çok raporlamalarda teknik bilgi amaçlı kullanılır ve çok daha fazla ayrıntı içerebilir.

**Use Case Diyagramları**

Yazılım geliştirme süreçlerinin hemen her aşamasında kullanılan, özellikle sistemimizin fonksiyonalitelerini belirlememize yardımcı olan popüler UML diyagramlarından birisidir. Bir use case diyagramı çeşitli çöp adamlardan, oval şekillerden ve bu çöp adamlar ile oval şekiller arasındaki ilişkiyi gösteren çizgilerden oluşur (oval şekiller ve çöp adamların da kendi aralarında ilişkileri olabilir). Use case diyagramlarında bir aktör ve bu aktörün gerçekleştirdiği işlerden başka kapsamlı bir bilgi yoktur.

Kendi uygulamamıza dönecek olursak, “YapLIST” projemizdeki use case diyagramımız şu şekildedir:

[](http://www.gelistir.org/wp-content/uploads/2011/02/830814c2.png)

Çöp adam olarak görünen nesne “aktör” olarak isimlendirilir ve bir role denk gelir. Aynı kişi (örneğin Ahmet) farklı rollere sahip olabilir. Sistemi kullanacak ya da sistem ile etkileşim halinde olan sistem dışındaki kişiler/sistemler/cihazlar vs. aktör olabilirler. Yani sadece insanları aktör olarak değerlendirmemek gerekir.

Oval şeklinde görülenler ise use case ya da kullanım durumu dediğimiz aktörün sistem ile etkileşim halinde olan sistemin gerçekleştirebileceği işlevsellikleri ifade eder. Use case’ler yazılırken uyulması gereken en önemli kurallardan birisi, use case’lerin fiil cümlesi olarak yazılması gerektiğidir.  
Yukaridaki diyagramı yorumlayacak olursak, kullanıcı uygulamamız ile şu işlemleri gerçekleştirebilir:

* Kayıt olabilir.
* Sisteme giriş yapabilir.
* Yapılacak oluşturabilir/ güncelleyebilir. Yapılacak oluştururken veya güncellerken “isterse” kategori seçebilir, öncelik belirleyebilir. “İstemezse” kategori ve öncelik seçmek zorunda değildir. Use case’deki “extends” bizim örneğimiz için bunu ifade eder (ki bu özellikle çok karıştırılan bir konudur). “extends” eden use case, “extends” edilen use case’e dahil olabilir, şayet belli kriterler sağlanırsa.
* Oluşturduğu yapılacak listesinde arama yapabilir.
* Oluşturduğu yapılacakları listeleyebilir.
* Yapılacaklardan istediklerini silebilir.
* Kategori oluşturabilir, oluşturduğu kategorileri güncelleyebilir ve silebilir.

Kullanım durumları ihtiyaca göre daha basit veya daha ayrıntlı olabilirler. Bizim örneğimizde “Öncelik Seç” ya da “Kategori Seç” diye gösterdiğimiz kullanım durumları ayrı bir use case olarak gösterilmeyebilirlerdi.

**Senaryolar**

Bir use case’i, aktör ile sistem arasında etkileşim sonucu gerçekleştirilen iş olarak tanımlamıştık. Bu etkileşim, bir tenis maçında topun gidip gelmesi gibi aktör ve sistem arasında gidip gelir. Bu etkileşime teknik olarak senaryo ismi verilir. Tıpkı bir film senaryosu gibi, her bir use case’e ait senaryolar vardır. Bir use case bir “temel senaryo” (Basic flow), birden fazla alternatif senaryo (alternate/alternative flow) ve hata senaryolarından (exceptional flow) oluşur. Temel akış senaryosu hiç bir sorun olmadan gerçekleşen senaryodur.

Şimdi, “Kayıt Ol” kullanım durumuna ait temel senaryomuzu yazalım:

“*Sistem kullanıcıya kayıt sayfasını gösterir. Kullanıcı; isim, email, kullanıcı adı ve şifre bilgilerini girererek “kaydet” butonuna tıklar. Sistem tüm verileri kontrol ederek doğrular. Sistem, kullanıcıyı anasayfaya yönlendirir.*”

Görüldüğü gibi herşey yolunda giderken ve tüm veriler düzgün girildiğinde senaryomuzun nasıl işletileceği oldukça nettir; fakat işler her zaman yolunda gitmez. Örnek vermek gerekirse, kişinin emaili zaten sisteme kayıtlı olabilir. Şifre boş bırakılmış olabilir. Bu durumda alternatif senaryo ile devam edilir. Yukarıdaki senaryoda “sistem tüm verileri kontrol ederek doğrular” adımına alternatif bir adım gerçekleştirilir. Email’in sistemde kayıtlı olması durumunda işletilecek alternatif senaryo şu şekilde olabilir:

*“Sistem kullanıcıya kayıt sayfasını gösterir. Kullanıcı; isim, email, kullanıcı adı ve şifre bilgilerini girererek “kaydet” butonuna tıklar. Sistem emailin zaten veritabanında olduğunu tespit eder. Sistem, kullanıcıya hata mesajını iletir ve kullanıcıyı kayıt sayfasında yönlendirir.”*

Şifrenin boş olması ya da uygun olmaması durumunda şöyle bir senaryoya sahip olabiliriz:

*“Sistem kullanıcıya kayıt sayfasını gösterir. Kullanıcı; isim, email, kullanıcı adı ve şifre bilgilerini girererek “kaydet” butonuna tıklar. Sistem şifreyi doğrulayamaz. Sistem, kullanıcıya hata mesajını iletir ve kullanıcıyı kayıt sayfasında yönlendirir.”*

Kullanıcıdan kaynaklanan bir eksiklik olduğunda, alternatif adımlar devreye girer ve etkileşim belli bir yerden devam eder.

Hata senaryoları use case’in bir problemler karşılaştığını ve akışın kesilmesi gerektiği durumlarda işletililirler.

**Not:** Hata senaryoları ile alternatif senaryolar, hala tartışma konusu. Kimisi bizim yukarıdaki alternatif senaryoları hata senaryosu olarak yorumluyor. Hata senaryosunda akış sorun olduğu anda kesilir ve devam etmez; ama alternatif senaryosunda akış başka adımlardan başlayarak devam eder.

**Use Case’ler**

Giriş kısmında bahsedildiği gibi use caseleri daha ayrıntılı ve döküman formatında yazmak mümkün. Özellikle büyük projelerde, her bir use case dökümante edilirken bu şablon daha çok kullanılmaktadır. Her bir use case; zorluk derecesi, hangi aktör(ler) ile ilişkili olduğu, öncelik, bu use case’in gerçekleşmesi için gerekli ön koşullar (preconditions), bu use case, temel senaryo, alternatif senaryolar ve projeden projeye değişen tonla bilgi içerebilir.  
Kayıt ol use case’ini şu şekilde dökümante edebiliriz:

|  |  |
| --- | --- |
| Use Case Adı | UC1- Kayıt Ol |
| İlgili Aktörler | Kayıtlı olmayan kullanıcı |
| Amaç | Kullanıcın sisteme kayıt olmasını sağlamak |
| Açıklama | Sistemde yapılacak ekleme/çıkarma gib işlemler yapabilmek için, kayıtlı bir kullanıcıya ihtiyaç vardır. Bu use case’de kayıtlı olmayan kullanıcıların, sisteme kayıt olması işlemi ele alınmaktadır. |
| Zorluk | Orta |
| Tetik | Kullanıcının kayıt sayfasına girmesi |
| Ön Gereksinimler | Kullanıcının kayıt sayfasında bulunması |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem kullanıcıya kayıt sayfasını gösterir. 2. Kullanıcı; isim, email, kullanıcı adı ve şifre bilgilerini girer. 3. Kullanıcı “kaydet” butonuna tıklar. 4. Sistem tüm verileri kontrol ederek doğrular. 5. Sistem, kullanıcıyı anasayfaya yönlendirir. |
| Alternatif Senaryo(lar) | 4a. Sistem emailin sistemde kayıtlı olduğunu tespit eder ve kullanıcıyı 1. adıma yönlendirir. 4b. Sistem şifreyi doğrulayamaz ve kullanıcıyı 1. adıma yönlendirir. |

Dikkat edilirse, alternatif senaryodaki adımlar, temel senaryodaki 4. adıma alternatif olarak işletiliyorlar. “Yapılacak Sil” kullanım durumu ise şu şekildedir.

|  |  |
| --- | --- |
| Use Case Adı | UC -4 Yapılacak Sil |
| İlgili Aktörler | Kayıtlı kullanıcı |
| Amaç | Kullanıcın kendi oluşturduğu bir yapılacağı silmek. |
| Açıklama | Kullanıcı daha önce oluşturduğu bir görevi seçip silmek isteyebilir. Bu kullanım durumu, kullanıcının yapılacağı silmesi ile ilgili bilgileri içerir. |
| Zorluk | Kolay |
| Tetik | Kullanıcı listeden bir yapılacak seçer ve “sil” butonuna basar. |
| Ön Gereksinimler | Kullanıcının sisteme giriş yapmış olması ve yapılacak oluşturmuş olması gerekir. |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem kullanıcıya yapılacak listesini gösterir. 2. Kullanıcı yapılacaklardan birisini seçer. 3. Kullanıcı “yapılacak sil” butonuna basar. 4. Sistem “emin misiniz?” diyerek teyit alır. 5. Kullanıcı, “evet” butonuna tıklar. 6. Sistem yapılacak’ı siler. 7. Sistem yapılacak listesini gösterir. |
| Alternatif Senaryo(lar) | 5a1.Kullanıcı “hayır” butonuna tıklar. 5a2. Sistem kullanıcıyı yapılacak listesi sayfasına yönlendirir. |
| Hata Senaryosu | 6a1 Sistem silinecek “yapılacak’ı” bulamaz. 6a2. Sistem kullanıcıyı yapılacak listesi sayfasına yönlendirir. |

**Sonuç**

Use case diyagramları belki de en çok kullanılan UML diyagramı olup, her türlü uygulama geliştirme projesinde, bu sürecin her aşamasında ve her detayda kullanılabilen sistemin daha doğru çözümlenmesine katkıda bulunan önemi yadsınamaz teknolojilerden birisidir.

Bu yazıda, kullanım durumlarının ne olduğuna, nasıl kullanılması gerektiğine değindik. Uygulamamızı use case diyagramı kullanarak analiz etmeye çalıştık. Sonraki yazıda daha ayrıntılı olarak ekranların şablon (mockup) tasarımlarıyla beraber analizimize devam edeceğiz.

Son olarak use case oluştururken şunlara dikkat etmeliyiz:

* Aktör’lerin sistem dışından olmasına dikkat etmelisiniz.
* Her bir use case’e özgü bir isim verilmelidir.
* Kesinlikle mükemmeliyetçilikten uzak durulmalıdır. Önce basit olarak başlayıp, sonra üzerine koyarak gitmek, gereksinimler değiştiğinde use case’leri güncellemek en mantıklısıdır.
* İsimlendirmelere dikkat edilmelidir. Use case isimleri fiil cümlesi olmalıdır.
* Gereğinden fazla karmaşık, kalabalık olmamalıdır.
* Tablo biçiminde gösterilecek kullanım durumu da basit olmalıdır.

Sonraki yazıda görüşmek dileğiyle…

Bir önceki yazıda, analizimize use case’ler ile başlamıştık. Bu yazıda use case’lere kaldığımız yerden devam edeceğiz. YapLIST uygulamamıza ait önceden belirlenen her bir use case ve bu use case’ lere ait ekran görüntülerine ait prototipler oluşturacağız ki hem çözümleyiciler istenen gereksinimlerin nasıl olduğunu daha net öğrenebilsin, hem de proje sahibi derdini daha rahat açıklayabilsin.

**Mockup Nedir?**

Prototipler: Görsel olarak hızlıca menüler, butonlar, çeşitli bileşenler tasarlayıp sistemi anlamaya, dökümante etmeye yardımcı olur. Hatta farklı ekranlar yapabilir ve bu ekranlar arasındaki etkileşimi bile gösterebiliriz. Detayına göre 3 çeşit ekran modelleme var diyebiliriz: ayrıntılı (high fidelity), orta (medium fidelity) ve ayrıntısız (low fidelity). Biz analizimizde, ayrıntısız ekran prototiplerine yer vereceğiz.

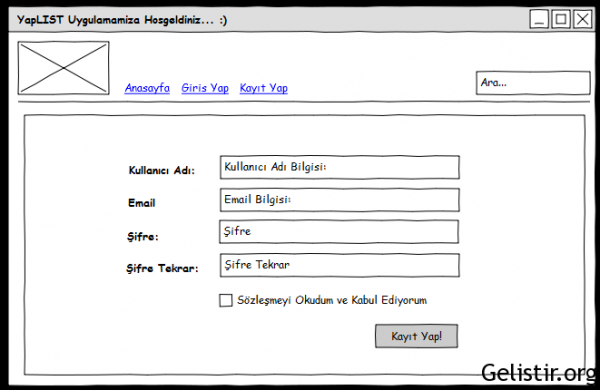
Ekran taslakları oluştururken, kullanabileceğimiz çeşitli araçlar mevcut. Kolayca ekranlar, menüler, bileşenler oluşturarak prototipler oluşturmamızı sağlıyorlar. Bazıları o kadar çok gelişmiş ki, butona basıldığında hangi işlemlerin yapılması gerektiği gibi interaktif işlemleri bile belirleyebiliyoruz.

Buradaki görüntüler ise görece basit sayılabilecek Firefox’a ait plugin olan “Pencil” ile tasarlandı.

**Use Case’ler**

Use case’ler ve ilgili ekran prototipleri şu şekildedir:

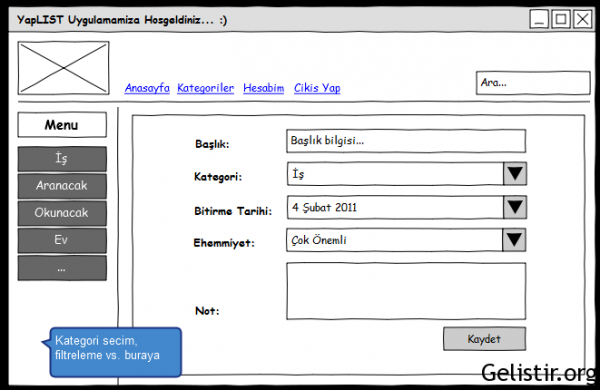
|  |  |
| --- | --- |
| Use Case Adı | UC1 – Kayıt Ol |
| İlgili Aktörler | Kayıtlı olmayan kullanıcı |
| Amaç | Kullanıcının sisteme kayıt olmasını sağlamak |
| Açıklama | Sistemde yapılacak ekleme/çıkarma gib işlemler yapabilmek için, kayıtlı bir kullanıcıya ihtiyaç vardır. Bu use case’de kayıtlı olmayan kullanıcıların, sisteme kayıt olması işlemi ele alınmaktadır. |
| Zorluk | Orta |
| Tetik | Kullanıcının kayıt sayfasına girmesi |
| Ön Gereksinimler | Kullanıcının kayıt sayfasında bulunması |
| Temel Akış Senaryosu Basic Flow Scenario) | 1. Sistem kullanıcıya kayıt sayfasını gösterir. 2. Kullanıcı; isim, email, kullanıcı adı ve şifre bilgilerini girer. 3. Kullanıcı “kaydet” butonuna tıklar. 4. Sistem tüm verileri kontrol ederek doğrular. 5. Sistem, kullanıcıyı anasayfaya yönlendirir.” |
| Alternatif Senaryo(lar) | 4a. Sistem emailin sistemde kayıtlı olduğunu tespit eder ve kullanıcıyı 1. adıma yönlendirir. 4b. Sistem şifreyi doğrulayamaz ve kullanıcıyı 1. Adıma yönlendirir. |

[](http://www.gelistir.org/wp-content/uploads/2011/02/Hesap-Olustur.png)

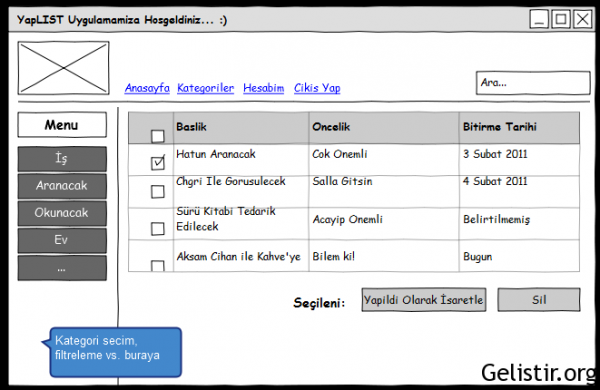
|  |  |
| --- | --- |
| Use Case Adı | UC2 – Giriş Yap |
| İlgili Aktörler | Kayılı kullanıcı |
| Amaç | Kullanıcın sisteme giriş yapmasını sağlamak |
| Açıklama | Sistemde yapılacak ekleme/çıkarma gibi işlemler yapabilmek için, kayıtlı bir kullanıcıya ihtiyaç vardır. Bu use case’de kayıt yapmış kullanıcıların, sisteme giriş işlemi ele alınmaktadır. |
| Zorluk | Basit |
| Tetik | Kullanıcının giriş sayfasına girmesi. |
| Ön Gereksinimler | Kullanıcının giriş sayfasında bulunması |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem kullanıcıya giriş sayfasını gösterir. 2. Kullanıcı; kullanıcı adı ve şifre bilgilerini girer. 3. Kullanıcı “giriş yap” butonuna tıklar. 4. Sistem tüm verileri kontrol ederek doğrular. 5. Sistem, kullanıcıyı anasayfaya yönlendirir. |
| Alternatif Senaryo(lar) | 4a. Sistem emailin sistemde kayıtlı olmadığını tespit eder ve kullanıcıyı 1. adıma yönlendirir.  4b. Sistem şifreyi doğrulayamaz ve kullanıcıyı 1. Adıma yönlendirir. |

[](http://www.gelistir.org/wp-content/uploads/2011/02/Giris-Yap.png)

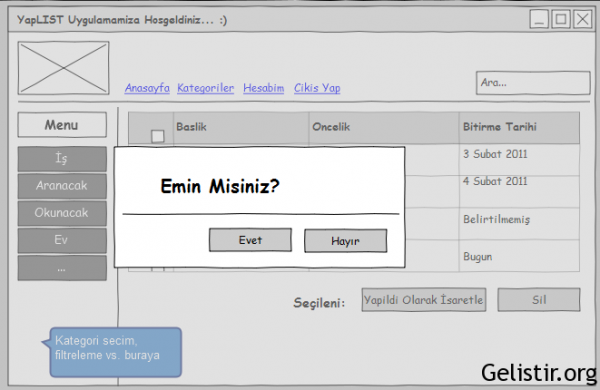
|  |  |
| --- | --- |
| Use Case Adı | UC3 – Yapılacak Ekle/Güncelle |
| İlgili Aktörler | Kayıtlı kullanıcı |
| Amaç | Yapılacak oluşturmak |
| Açıklama | Uygulamamızın amacı, kullanıcının kendi ihtiyaçlarına göre yapılacak oluşturması. Bu use case’de yapılacak oluşturma işlemi yapılır. |
| Zorluk | Zor |
| Tetik | Kullanıcı “yapılacak oluştur” butonuna basar. |
| Ön Gereksinimler | Kullanıcının sisteme giriş yapmış olması ve yapılacak oluşturma sayfasına yönlenmiş olması. |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem kullanıcıya yapılacak oluşturma sayfasını  gösterir. 2. Kullanıcı başlık bilgisi girer. 3. Kullanıcı kategori seçebilir. 4. Kullanıcı öncelik/önemlilik bilgisi seçebilir. 5. Kullanıcı, bitirme tarihi seçer. 6. Kullanıcı açıklama girer. 7. Kullanıcı “kaydet” butonuna basar. 8. Sistem bilgileri doğrular 9. Sistem, kullanıcıyı yapılacak listesi sayfasına yönlendirir. |
| Alternatif Senaryo(lar) | 8a1.Başlık boş bırakılmıştır. 8a2. Kullanıcı, hata mesajı ile uyarılır ve yapılacak oluşturma sayfasına yönlendirilir. 8b1. Bitirme tarihi önceki bir tarih olarak seçilmiştir. 8b2. Kullanıcı, hata mesajı ile uyarılır ve yapılacak oluşturma sayfasına yönlendirilir. |

[](http://www.gelistir.org/wp-content/uploads/2011/02/Gorev-Ekleme-e1296747891942.png)

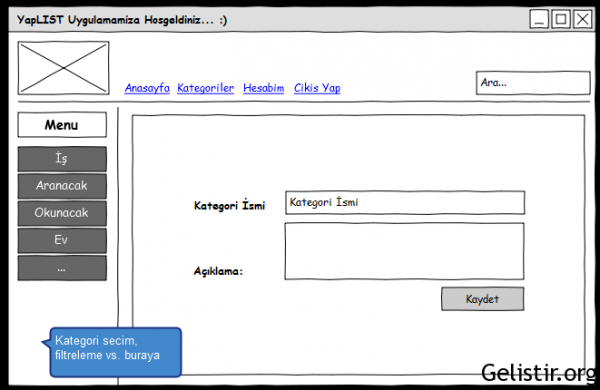
|  |  |
| --- | --- |
| Use Case Adı | UC4 – Yapılacak Listele |
| İlgili Aktörler | Kayıtlı kullanıcı |
| Amaç | Yapılacakları listelemek |
| Açıklama | Kullanıcı, oluşturmuş olduğu yapılacakları listelemek isteyebilir. Bu use case bu işlemi ele almaktadır. |
| Zorluk | Basit |
| Tetik | Kullanıcının giriş yapması. |
| Ön Gereksinimler | Kullanıcının kayıtlı olması |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem kullanıcıyı yapılacakların bulunduğu sayfaya yönlendirilir. |
| Alternatif Senaryo(lar) | Yok |

[](http://www.gelistir.org/wp-content/uploads/2011/02/Gorev-Listesi-e1296747822804.png)

|  |  |
| --- | --- |
| Use Case Adı | UC5 – Yapılacak Sil |
| İlgili Aktörler | Kayıtlı kullanıcı |
| Amaç | Kullanıcın kendi oluşturduğu bir yapılacağı silmek. |
| Açıklama | Kullanıcı daha önce oluşturduğu bir görevi seçip silmek isteyebilir. Bu kullanım durumu, kullanıcının yapılacağı silmesi ile ilgili bilgileri içerir. |
| Zorluk | Kolay |
| Tetik | Kullanıcı listeden bir yapılacak seçer ve “sil” butonuna basar. |
| Ön Gereksinimler | Kullanıcının sisteme giriş yapmış olması ve yapılacak oluşturmuş olması gerekir. |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem kullanıcıya yapılacak listesini gösterir. 2. Kullanıcı yapılacaklardan birisini seçer. 3. Kullanıcı “yapılacak sil” butonuna basar. 4. Sistem “emin misiniz?” diyerek teyit alır. 5. Kullanıcı, “evet” butonuna tıklar. 6. Sistem yapılacak’ı siler 7. Sistem yapılacak listesini gösterir |
| Alternatif Senaryo(lar) | 5a1.Kullanıcı “hayır” butonuna tıklar.  5a2. Sistem kullanıcıyı yapılacak listesi sayfasına yönlendirir. |
| Hata Senaryosu | 6a1 Sistem silinecek “yapılacakı” bulamaz.  6a2. Sistem kullanıcıyı yapılacak listesi sayfasına yönlendirir. |

[](http://www.gelistir.org/wp-content/uploads/2011/02/Gorev-Sil.png)

|  |  |
| --- | --- |
| Use Case Adı | UC6-Kategori Ekle/Güncelle |
| İlgili Aktörler | Kayıtlı kullanıcı |
| Amaç | Kategori oluşturmak |
| Açıklama | Kullanıcı, yapılacakları çeşitli kategorilere göre düzenleyebilir. Bu use case’de kategori oluşturma ve güncelleme ele alınmaktadır. |
| Zorluk | Orta |
| Tetik | Kullanıcının “kategori oluştur” butonuna basması veya var olan kategoriyi “güncelle” butonuna basması. |
| Ön Gereksinimler | Kullanıcının giriş yapması. |
| Temel Akış Senaryosu (Basic Flow Scenario) | 1. Sistem, kullanıcıyı kategori oluşturma/güncelleme sayfasına yönlendirir. 2. Kullanıcı, kategori ismi girer. 3. Kullanıcı açıklama girer. 4. Kullanıcı “kaydet” butonuna basar. 5. Sistem, verileri doğrular. 6. Sistem, kullanıcıyı, kategori listesi sayfasına yönlendirir |
| Alternatif Senaryo(lar) | 5a1.Kategori ismi boş bırakılmıştır. 5a2. Kullanıcı hata mesajı ile uyarılır ve kategori oluşturma sayfasına yönlendirilir. |

[](http://www.gelistir.org/wp-content/uploads/2011/02/Kategori-Olu%C5%9Ftur-e1296747868581.png)

**Sonuç**

Ekran prototipleri oluşturarak hem analizi yapan ekibin sistemi anlamasına, hem de proje sahibinin derdini daha kolay anlatmasına katkıda bulunmuş oluyoruz.  “Görselleştirdikten” sonra,  uygulamamızın ekranlarının neye benzeyeceği konusunda bir kademe daha ilerlemiş olduk.

Dikkat edilirse, şu ana kadar sadece çözümleme yaptık. Sistemin hangi özelliklere sahip olması gerektiğini belirledik. Sonraki yazıdan itibaren, kullanım durumlarının “nasıl” gerçekleştirileceğini inceleyeceğiz.

**Giriş**

Bir önceki yazıda, ekran prototiplerini oluşturmuştuk ve artık sistem gereksinimlerimizi belirleyerek çözümlememizi büyük oranda halletmiştik. Bu yazıdan itibaren farklı bir perspektiften bakarak analize devam edeceğiz.

İlk kısımda domain modeli ile ilgili genel bilgi bulunuyor. Ardından ihtiyacımız kadar UML bilgisi edindikten sonra uygulamamızdaki domainleri inceleyeceğiz.

**Domain Model**

Domain modeli olarak isimlendirilen bu model, projenin kullanılacağı uygulama sahası ile ilgili modellerin tespit edilmesi ve tespit edilen bu modellerin birbirleriyle olan ilişkilerini belirlememize yardımcı olur.

Her proje sahasının kendine göre farklı modelleri olacaktır. Örneğin insan kaynakları alanında bir uygulama geliştireceksek, “çalışan”, “maaş”, “prim”, “izin” gibi kavramlara ihtiyaç duyarken, stok ile ilgili bir projede “malzeme”, “depo” gibi domain bilgisine ihtiyaç duyulur. Domainlerin belirlenmesi ile bir anlamda projenin “sözlüğü” de çıkarılmış oluyor. Bu yazıda domain, domain modeli, domain objesi ya da nesne isimleriyle kullanılan tüm kavramlar yukarıda belirtilen örneklerdeki anlamda kullanılmıştır.

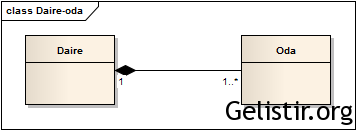
Domain modellerken UML sınıf diyagramlarından faydalanırız; fakat domain model diyagramları, sınıf diyagramları ile karıştırılmamalıdır. Sınıf diyagramlarına göre daha az ayrıntı içerirler. Domain modelinin veritabanı modelleriyle de karşılaştırılmaması gerekir. Geliştirme sürecine göre sınıflar ve veritabanı tabloları domain modeli baz alınarak tasarlanabilir.

**Biraz da UML Bilgisi**

Domain’ler arasındaki ilişkiyi gösterebilmek için UML bilgisine ihtiyacımız var. UML’de kullanılan sınıflar arası ilişkilerden birisi sahiplik ilişkisidir. Yukarıdaki açıklamadan örnek verecek olursak: kullanıcı yapılacaklara sahiptir. Bunu UML’de kullanıcı ve yapılacak arasında bir çizgi çizerek ifade ederiz.

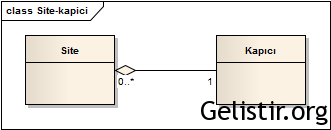
Sınıf diyagramlarında iki çeşit sahiplik ilişkisi bulunur. Bunlara “Composition” ve “Aggregation” denilmektedir. Composition’da tam sahiplik, aggregation’da ise kısmi sahiplik vardır gibi düşünebiliriz.

Composition’da parça bütün ilişkisi vardır. İçi dolu baklava dilimi ile gösterilirler. Parçalar bütünden ayrı düşünülemez. Örnek vermek gerekirse bir daire ve bu dairelere ait odaları düşünebiliriz. Odalar bir eve aittir. Odalar ve diğer objeler (örneğin kapılar) tek başına anlamlı değiller. Bu parçalar birleşerek bütünü (daire) oluştururlar. Eğer ev olmazsa odaların da anlamı yoktur, dolayısıyla odalar da yoktur. Daire-oda ilişkisini UML’de şu şekilde gösteririz:

[](http://www.gelistir.org/wp-content/uploads/2011/02/Daire-oda.png)

Daire ve Oda arasındaki çizginin iki ucunda bulunan sayılar, ilişkinin sayı miktarı ile ilgili bilgi verirler. Diyagramda, “Daire” tarafında 1, “Oda” tarafında 1..\* ifadesi görünür. Bu ilişki “bir daire 1 veya daha fazla oda’ya sahiptir” diye okunur. Oda tarafından düşünecek olursak; 1 oda bir dairede bulunabilir diye okuyabiliriz.

Aggragation’da ise yine parça bütün ilişkisi vardır; fakat iki obje birbirinden bağımsız işleve sahip olmaya devam edebilir. İçi boş baklava dili ile gösterilirler. Aggregation için Site-kapıcı örneği verelim. Site kendisine ait bir kapıcıya sahip olabilir. Fakat site olmadığı zaman kapıcı var olmaya devam edecektir.

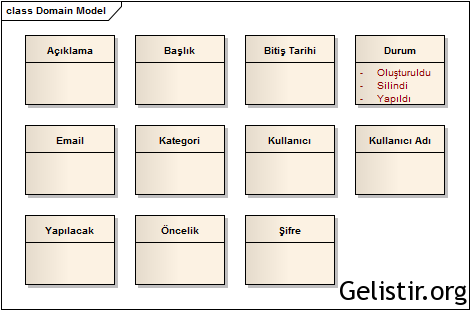
[](http://www.gelistir.org/wp-content/uploads/2011/02/Site-kapici.png)

**YapLIST Domain Modeli**

Gerçek bir projede farklı kategoriler altında binlerce domain model olacaktır. Bizim uygulamamız oldukça basit olduğu için birkaç domain objesi ile işimizi görüyoruz.

Uzatmadan uygulamamıza dönecek olursak: domain objeleri, gereksinimlerden ve tüm bu gereksinim analizi sürecindeki etkinliklerden çıkarılabilir. Kullanım durumları yazılırken senaryolardaki adımlar, kullanılacak domain’ler ile ilgili önemli bilgiler sunar.

Biz de daha önceki yazılardan, kullanılacak domain sınıflarımızın şunlar olduğunu belirledik:

[](http://www.gelistir.org/wp-content/uploads/2011/02/Domainler.png)

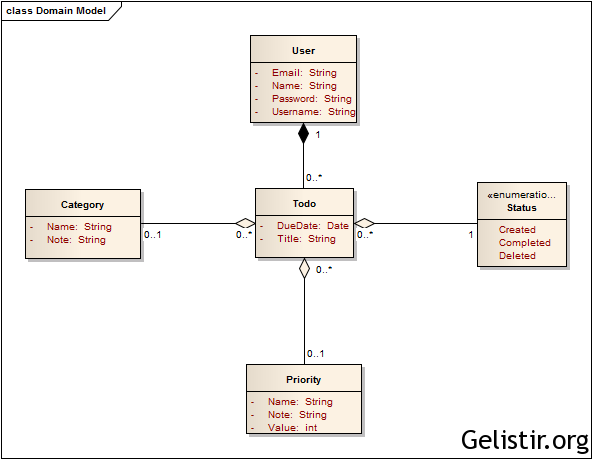
Gerçek bir projeye benzetmek ve biraz da artistlik yapmak amacıyla domain modelimizi İngilizce yapmaya karar verdik. O yüzden yukarıdaki diyagrama ait domainlerin İngilizce’lerini de belirleyelim.

|  |  |
| --- | --- |
| Domain Objesi | İngilizce Karşılığı |
| Açıklama | Note |
| Başlık | Title |
| Bitiş Tarihi | Due Date |
| Durum (Oluşturuldu, silindi, yapıldı) | Status (Created, deleted,completed) |
| Email | Email |
| Kategori | Category |
| Kullanıcı | User |
| Kullanıcı Adı | Username |
| Öncelik | Priority |
| Şifre | Password |
| Yapılacak | Todo |

Sesli düşünecek olursak:

* Kullanıcılar sisteme kayıt olacak. Kayıtlı olan bu kullanıcılara ait yapılacak(lar) olabilir.
* Her bir yapılacak için; başlık, açıklama zorunlu.
* Her yapılacak’a ait bir durum vardır. Bir yapılacak’ın durumu oluşturuldu, silindi ya da yapıldı seçeneklerinden birisi olabilir.
* Yapılacaklar ayrıca kategori ve öncelik içerebilir.
* Biraz daha inceleyince, başlık ve açıklama bilgisinin ayrı bir obje değil yapılacak’a ait bir öznitelik (attribute) olduğunu farkediyoruz. Aynı şekilde kullanıcı adı, email, şifre’nin de ayrı birer obje değil kullanıcıya ait bir özellik olduğunu belirledik. Ayrıca her bir yapılacak’a ait bitiş zamanı bilgisini de tutumamız gerektiğini farkettik.

Kendi uygulamamıza dönecek olursak, domain modelimiz şu şekilde olur:

[](http://www.gelistir.org/wp-content/uploads/2011/02/Domain-Model.png)

Yukarıdaki diyagramı teker teker yorumlayacak olursak:  
• Kullanıcı (user) 0 veya daha fazla yapılacak (todo) oluşturabilir.  
• Yapılacak sadece 1 kullanıcıya aittir.  
• Yapılacak’a ait (todo) 1 durum (status) vardır.  
• Aynı durum (örneğin tamamlandı), 1’den fazla yapılacak’ta bulunabilir.  
• Yapılacak’a ait öncelik belirlenebilir. Belirlenirse en fazla 1 tane önceliği olur.  
• Aynı öncelik 1’den fazla yapılacak’ta olabilir  
• Yapılacak’a ait en fazla 1 tane kategori olabilir. Kategori seçmek de zorunlu değildir.  
• 1 kategori 1’den fazla yapılacak’ta olabilir.

**Sonuç**

Bu yazıda, analiz-geliştirme sürecine farklı bir perspektiften baktık. Domain’lerin belirlenmesi sayesinde veritabanı tabloları ve Hibernate gibi kütüphanelerde kullanılan ve tablolara tekabül eden varlık (entity) sınıflarını çıkartabiliriz.

Sonraki yazıda sistem mimarisini detaylandırıp, farklı tür sınıflar arasındaki ilişkiyi ele alan dinamik modeller üzerinde çalışarak, kaldığımız yerden devam edeceğiz.

### Giriş

Bir önceki yazıda domain modellerimizi belirlemiştik. Bu yazıda ise sequence diyagramları ile kaldığımız yerden devam edeceğiz. Sequence diyagramlarında, birden fazla obje etkileşim halinde olur.

Yazıda ilk önce sequence diyagramları ve burada kullanacağımız kavramlar ile ilgili açıklamalar bulunmaktadır. Ardından projemize ait her bir use case’in sequence diyagramı inceleniyor.

### Sequence Diyagramı Nedir?

Sequence diyagramı: bir işin (kullanım durumu ya da use case) gerçekleşmesi için ilgili objelerin aralarında “sırasıyla” hangi metodların çağrılması gerektiğini modellemize yardımcı olur. Oldukça sık kullanılan dinamik UML modellerinden birisidir.

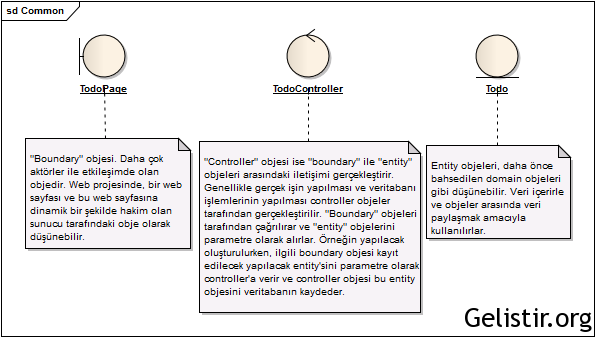
Diyagram okunurken, yukarıdan aşağıya doğru bir zaman çizgisi varmış gibi düşünülür.

Bizim yazımızdaki sequence diyagramlarını yorumlayabilmek için standart sequence diyagramlarında kullanılan kavramlara ek olarak bir iki tanesin daha bilmeye ihtiyacımız var. “Boundary”, “Control” ve “Entity” olarak isimlendirilen objeler ICONIX(\*) gibi çözümleme ve analiz tekniklerinde kullanılmaktadır. Boundary, dış dünyadaki aktörlerin ilk etkileşime girdiği objelerdir. İlgili komuta göre control objeleri, boundary objelerinden istenen işleme göre entity objelerini de kullanarak çalışırlar. İşlem sonuçlarını, dış dünyadaki aktörere gösteren ya da yönlendiren obje yine boundary objeleridir.

Entity-Boundary-Control, MVC (Model-View-Controller) tasarım desenine benzer. Aralarındaki ilişkiyi; Boundary-view, entity-model ve control-controller gibi düşünebiliriz. Bu objeler kullanılarak gerçekleştirilen bir işlem genelde şu şekilde olur:

1. İlgili sayfa veya form kullanıcıya gösterilir.  
2. Kullanıcı (aktör), bir işin yapılması için ilgili linke/butona tıklar.  
3. Control objesine ait metodlar çalıştırılır.  
4. Duruma göre entity objelerinde bulunan veriler de kullanılır. Entity’lerden gelen veriler ile birlikte, istenen işlemler (veritabanına yazma, hesap-kitap vs.) control tarafından gerçekleştirilir.  
5. Tüm süreç sonunda bulunan sonuç ya da istenen bilgi boundary objesi tarafından kullanıcıya (aktör) gösterilir.

Entity-Boundary-Control objelerini ve açıklamalarını görsel olarak şu şekilde ifade edebiliriz:

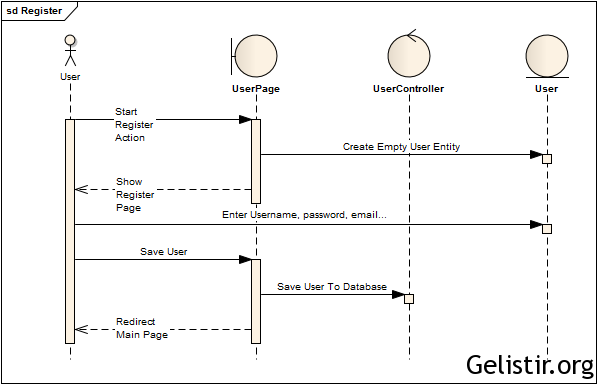
[](http://www.gelistir.org/wp-content/uploads/2011/02/Common1.png)

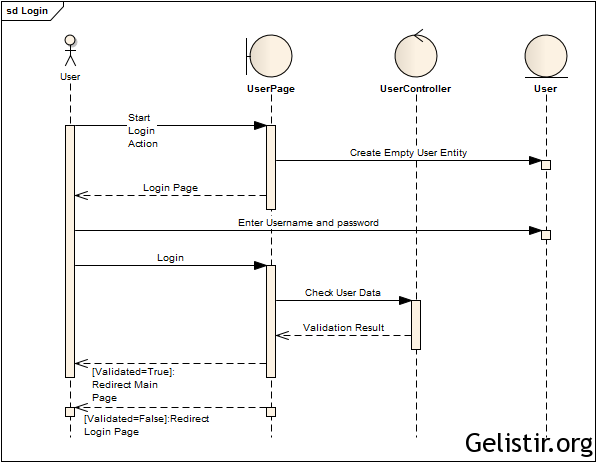
### Uygulamamıza Ait Sequence Diyagramları

Bu bölümde, önceki yazılarda belirlediğimiz use case’ler için sequence diyagramlarını vereceğiz. Açıklamalar Türkçe fakat diyaglardaki ifadeler İngilizce olarak gösterilmektedir.

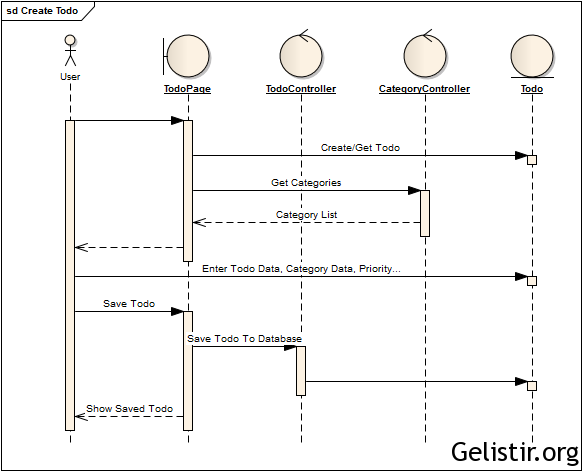
**Kayıt Ol**

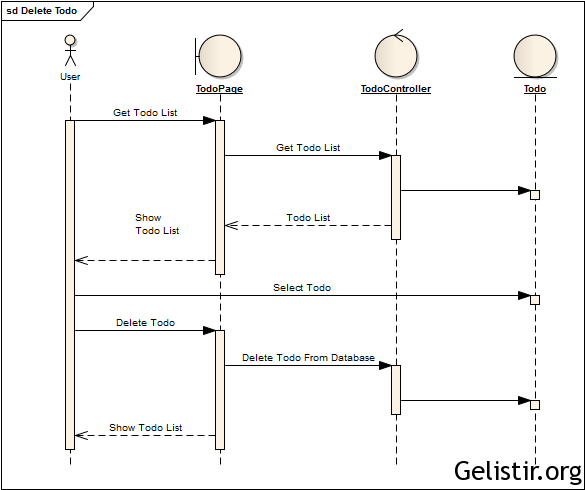
Kayıt olma işlemi için kullanıcının kayıt ol sayfasına yönlendirildiğini varsayıyoruz. Kayıt olma sayfasında yeni bir *User* objesi oluşturuluyor. Bu sayfada: kullanıcı adı, şifre gibi bilgiler bu *User* objesi içine dolduruluyor ve kullanıcı, “Kayıt ol” butonuna basıyor. Bu işlemi de *UserPage* boundary objesi karşılayıp, *UserController* objesine, kullanıcıyı kaydetmesi için mesaj gönderiyor. Tüm bu işlemlerin sonucunda kullanıcı ilgili sayfaya yönlendiriliyor.

[](http://www.gelistir.org/wp-content/uploads/2011/02/Register.png)

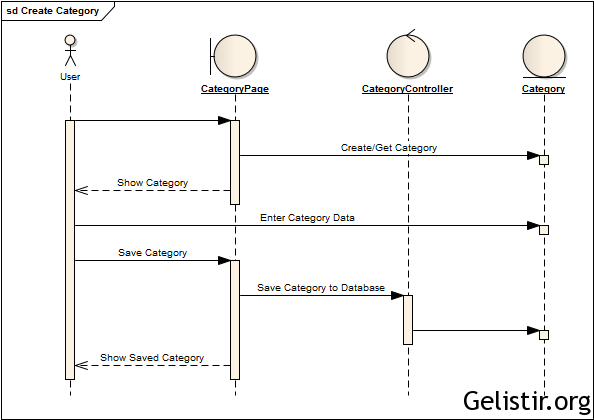
**Giriş Yap**  
Kullanıcı giriş sayfasına girer, kullanıcı adı ve şifre bilgilerini girer. Kullanıcı doğrulanır ve anasayaya yönlendirilir. Eğer doğrulanamazsa tekrar giriş sayfasına yönlendirilir. Doğrulanıp, doğrulanamadığını *[Validated=True veya False]* mesajı ile ifade ediyoruz. Sequence diyagramında bunun için başka gösterim yöntemleri de var; fakat bu durum için gerekli değil.  
[](http://www.gelistir.org/wp-content/uploads/2011/02/Login.png)

**Yapılacak Oluştur/Güncelle**  
Kullanıcı ilgili sayfaya yönlendirilir. Yapılacak ile ilgili verileri girer (başlık, tarih, kategori, öncelik vs.) ve kaydet butonuna basar. *TodoController* tarafından veritabanına kaydedilir ve kaydedilen yapılacak kullanıcıya *TodoPage* boundary objesi tarafından gösterilir.

[](http://www.gelistir.org/wp-content/uploads/2011/02/Create-Todo.png)

**Yapılacak Sil**  
Kullanıcı yapılacak seçer ve “sil” butonuna basar. Silinme sonrası kullanıcı yapılacak listesinin bulunduğu sayfaya yönlendilir.  
[](http://www.gelistir.org/wp-content/uploads/2011/02/Delete-Todo.png)

**Category Oluştur/Güncelle**  
Kategori oluşturma işlemi de “Yapılacak Oluştur/güncelle” kısmındaki örneğe benzer şekilde gerçekleşir.

[](http://www.gelistir.org/wp-content/uploads/2011/02/Create-Category.png)

### Sonuç

Sequence diyagramları ile nesnelerimizi/bileşenlerimizi büyük oranda belirlemiş olduk. Bu işlemden sonra uygulamada kullanacağımız sınıflarımızı belirleyip kodlama aşamasına geçebilirdik; fakat amacımız analiz olduğu için yazı serimizi burada sonlandırıyoruz.

Bir sonraki yazıda tecrübelerimizi paylaşarak yazı dizimizi kapatacağız.

Son yazımızda da görüşmek dileğiyle…

[15/02/2011](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%e2%80%93-7-sonuc/) by [Türkdoğan Taşdelen](http://www.gelistir.org/blog/author/admin/) in [UML](http://www.gelistir.org/category/uml/), [Yazılım Mühendisliği](http://www.gelistir.org/category/yazilim-muhendisligi/) | [Yorum Yap](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%e2%80%93-7-sonuc/#respond)

UML ile Sistem Analizi Serisi:

[UML ile Sistem Analizi 1 – Giris](http://www.gelistir.org/blog/uml-ile-sistem-analizi-1)   
[UML ile Sistem Analizi 2 – Gereksinimler](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-2-gereksinimler)   
[UML ile Sistem Analizi 3 – Use Case Modelleme](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-3-use-case-modelleme)  
[UML ile Sistem Analizi 4 – Ekran Prototipleri (Mockup)](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-4-ekran-prototipleri-mockup)   
[UML ile Sistem Analizi 5 – Domain Modelleme](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-5-domain-modelleme)  
[UML ile Sistem Analizi 6 – Sequence Diyagramlar](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-6-sequence-diyagramlari) [UML ile Sistem Analizi 7 – Sonuç](http://www.gelistir.org/blog/uml-ile-sistem-analizi-%E2%80%93-7-sonuc)

UML ile sistem analizi yazı serimizin sonuna gelmiş bulunuyoruz. Basit bir görev listesi uygulamasına ait gereksinimleri belirleyip sistemi analiz ettik. Hatta analiz etmekle kalmayıp, tasarımını da çıkarttık.

Projenin “ne” lerini ve “nasıl” larını belirlemiş olduk. Bundan sonraki süreç, uygulamayı geliştirmeye başlamak…

Uygulama geliştirme sürecinde ihtiyaçlar ve talepler değişebilir. Eğer uygun metodolojiler kullanılırsa, gereksinimlerde ya da tasarımdaki değişikliklerin kod geliştirme sürecini etkilememesi sağlanabilir.

Bizim adı geçen uygulamamız (yapLIST) oldukça basit olduğu için böylesine kapsamlı bir analiz sürecine ihtiyaç duyulmayabilirdi. Daha “sade” bir başlangıç yapıp, geliştirmeye hızlıca geçilebilir ve gelen geribildirimlere göre proje uyarlanabilirdi.

Daha büyük ölçekli uygulamalar içinse tabii ki daha ayrıntılı bir analiz süreci gerekecektir. İş akış bilgilerinden, hemen tüm UML diyagramlarına kadar farklı birçok modelleme tekniğinden faydalanılabilir.

Sonuç olarak analiz süreci ve bu süreçte kullanılacak teknik ve teknolojiler tamamen ihtiyaçlara göre değişir. Herhangi bir standard tam olarak mümkün olmadığı için karar vericilerin insiyatifine kalmış bir konudur.

Okuyan herkese faydalı olması dileklerimizle…