Yazılım Mimarisi ve En Bilinen Mimari Modeller

İçindekiler gizle

Yazılım Mimarisi Nedir? (Software Architecture)

Yazılım mimarisi bir yazılımın inşası için gerekli olan yazılımın tüm yapılarını tanımlayan ve bunu üst seviyeden görebilmek için ihtiyaç duyulan plandır. Yazılım mimarisini tüm sistemin organizasyonu ve iskeleti olarak ifade edebiliriz.

Yazılım mimarisi sistemin nasıl davranması gerektiğini en üst düzeyde açıklayan sistematik bir yoldur. Yazılım mimarisi yazılımın fonksiyonlarını, tüm bileşenlerini, kısıtlamaları, yazılıma ait modüllerin özelliklerini ve bunlar arasındaki ilişkileri açıklar. Genellikle sistemin nasıl yapıldığıyla ilgili değil ne yapıldığıyla ilgilenir. Sistemin nasıl inşa edildiği kısmıyla yazılım tasarımı (software design) ilgilenir. Yazılım mimarisi için iki soyutlanmış düzeyden bahsedebiliriz:

1. İlk düzey sistemin kendi mimarisi ile ilgilidir. Bu düzey sistemin bileşenlerini mimari açıdan gösterir.
2. Diğer düzeyde ise genel olarak mimari; diğer sistemleri, programları ve bu programların bileşenlerini içeren karmaşık kurumsal sistemlerin mimarisini içerir.Burada üst seviyeden farklı birimler hatta farklı şirketler tarafından sahip olunan ve yönetilen dış sistemleri de görebiliriz.

Yazılım mimarisinin ne olduğuna dair birçok klasik ve modern tanımlar bulunmaktadır. Bunun için SEI'nin (Software Engineering Institute) birçok kaynaktan hazırlamış olduğu bu adresteki derleme klasik ve modern yazılım mimarisi ile ilgili bir çok faydalı tanımları içermektedir.

Mimari Model Nedir? (Architectural Pattern)

Mimari modeller (kalıplar veya stiller olarak da ifade edebiliriz) bilgiyi temsil etmenin, paylaşmanın ve yeniden kullanmanın bir yoludur. Mimari modeller de farklı ortamlarda denenmiş ve test edilmiş iyi tasarım deneyimlerinin stilize edilmiş tanımlarıdır.

Yazılım mimarisi bir sistemi oluşturan parçaların bir bütün halde nasıl tasarlandığını ve aralarındaki etkileşimin nasıl olduğunu açıklar. Yazılım sistemleri için yazılım ilişkisini ve oluşumunu gösteren kavrama ise **mimari model (architectural pattern, architectural model)** denir. Mimari modeller; roller, sorumluluklar ve bunların ilişkilerini tanımlayan kurallar ve yol haritaları ile ilişkilendirme de dahil olmak üzere bir dizi yapı sağlayan yeniden kullanılabilir bir çözüm olarak düşünülebilir.

Farklı ortamlarda test edilmiş mimari modeller, yüksek kaliteli bir yazılım sistemi sunmak, mimari sistemlerin anlaşılmasına dayalı olarak maliyeti tahmin etmek ve ayrıca bir projenin zaman çizelgesini hızlandırmak için altyapı sağlar. Temel olarak uygulamanın genel özelliklerini tanımlama, ürünün işlevselliğini artırma ve uygulama oluşturma süreçlerinin verimliliğini ve üretkenliğini artırma için kullanılır. Bu mimari modeller ne zaman yararlı olduklarına dair veya ne zaman yararlı olmadıklarına dair bilgi içermelidir. Modeller tablo ve grafik açıklamalar kullanılarak temsil edilebilir.

Yaygın Mimari Modeller

Yazılım mimarisi sistemin amacı göz önünde bulundurularak tasarlandığından, ortak sorunları çözmek için yeniden kullanıma, paylaşmaya ve temsil etmeye uygun mimari kalıplar geliştirilmiştir. Uygulama kapsamı ve kurumsal kapsam ile ilgili olarak üretkenliği ve verimliliği artıran çok sayıda mimari model vardır. Proje gecikmesine sebep olmaması için, yazılım gereksinimlerinize uygun olanı seçebilmeniz için mimari kalıpları ve bunların hangi uygulamalar için en uygun olduğunu iyi anlamak gerekir. Bu bölümde yaygın ve iyi bilinen mimari modelleri bulabilirsiniz:

Model-View-Controller (MVC) Yazılım Mimarisi

MVC nedir? İlk olarak 1970'lerde tanımlanan ve bir tasarım kalıbından çok bir uygulamanın mimarisini kapsayan **MVC** mimari modeli, yaygın olarak kullanılan mimarilerden biridir. MVC'nin temel amacı, iş mantığını ve verileri sunum detaylarından ayırarak bağımsız olarak değişebilmelerine olanak sağlamaktır. MVC, application logic (uygulama) ile presentation (sunum) kısımlarını ayrı katman olarak ayırır. Presentation da controller ve view olarak bölünür. Bu modelin uygulama mantığına karşılık geldiği MVC'nin bu 3 mantıksal bileşenlerini açıklayalım:

Model

Application logic (bazen domain layer, etki alanı katmanı olarak adlandırılır) mimarinin bu kısmı ile ilgilidir. 3 bileşenden en düşük seviye olarak kabul edilir. Temel fonksiyonların yer aldığı bölümdür. Ayrıca verilerin korunmasından sorumludur. Birçok uygulama, verileri depolamak için kalıcı bir veritabanı gibi depolama mekanizması kullanır. Kaynak yönetimi "Model" tarafından kapsanmaktadır. Verileri mantıksal olarak işler ve uygulama durumunu saklar.

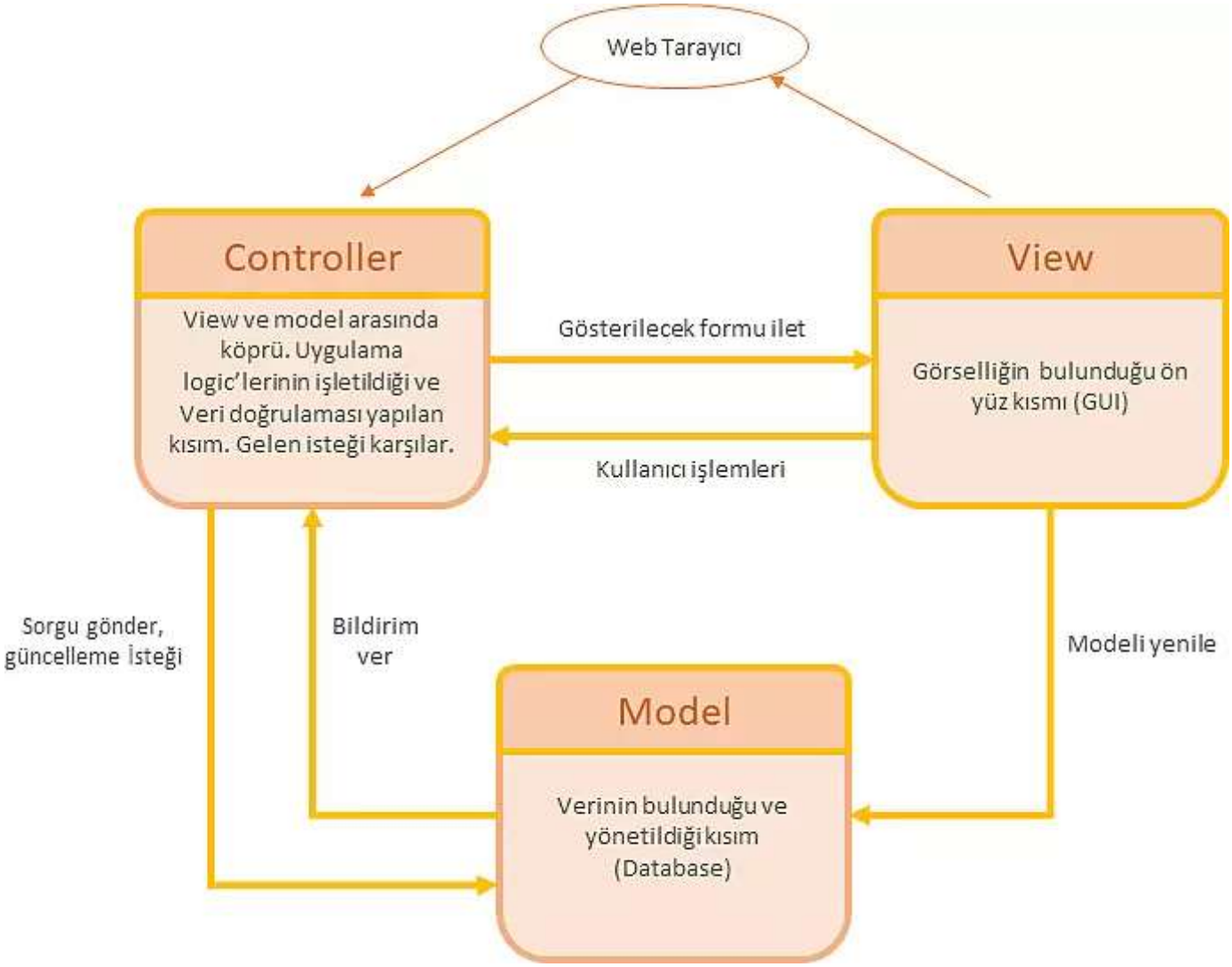
Controller

Bu bileşen, uygulama kullanıcılarının etkileşimleriyle ilgilenir. “View” ve “Model” arasındaki bağlantıyı sağlar. Veriler üzerindeki ve bu veriler üzerinde işlemleri yönetir. Başta kullanıcı eylemleri olmak üzere oluşan butona tıklama gibi event adı verilen işlemleri alır, işler, View ve Model’e iletir ve bunlara yanıt verir. Yani “Controller” kullanıcıların yaptığı istekleri gerçekleştirir. Model ile çalışarak görüntülenecek uygun görünümü seçer. View (arayüz) ve Model (application logic) arasında gerçekleşen tek etkileşim Controller aracılığıyla sağlanır.

View

Bu bileşen, uygulama kullanıcılarının etkileşimde bulunduğu bir uygulamanın kullanıcı önyüzünü hazırlamak için kullanılır. Modeldeki verilerin görünümünü etkileşim sağlanabilmesi için uygun bir forma, yani UI (user interface) olarak ifade ettiğimiz önyüze dönüştürmek için kullanılır.

Örnek olarak bu model, birçok web uygulamalarının etkileşim yönetiminin temelidir:

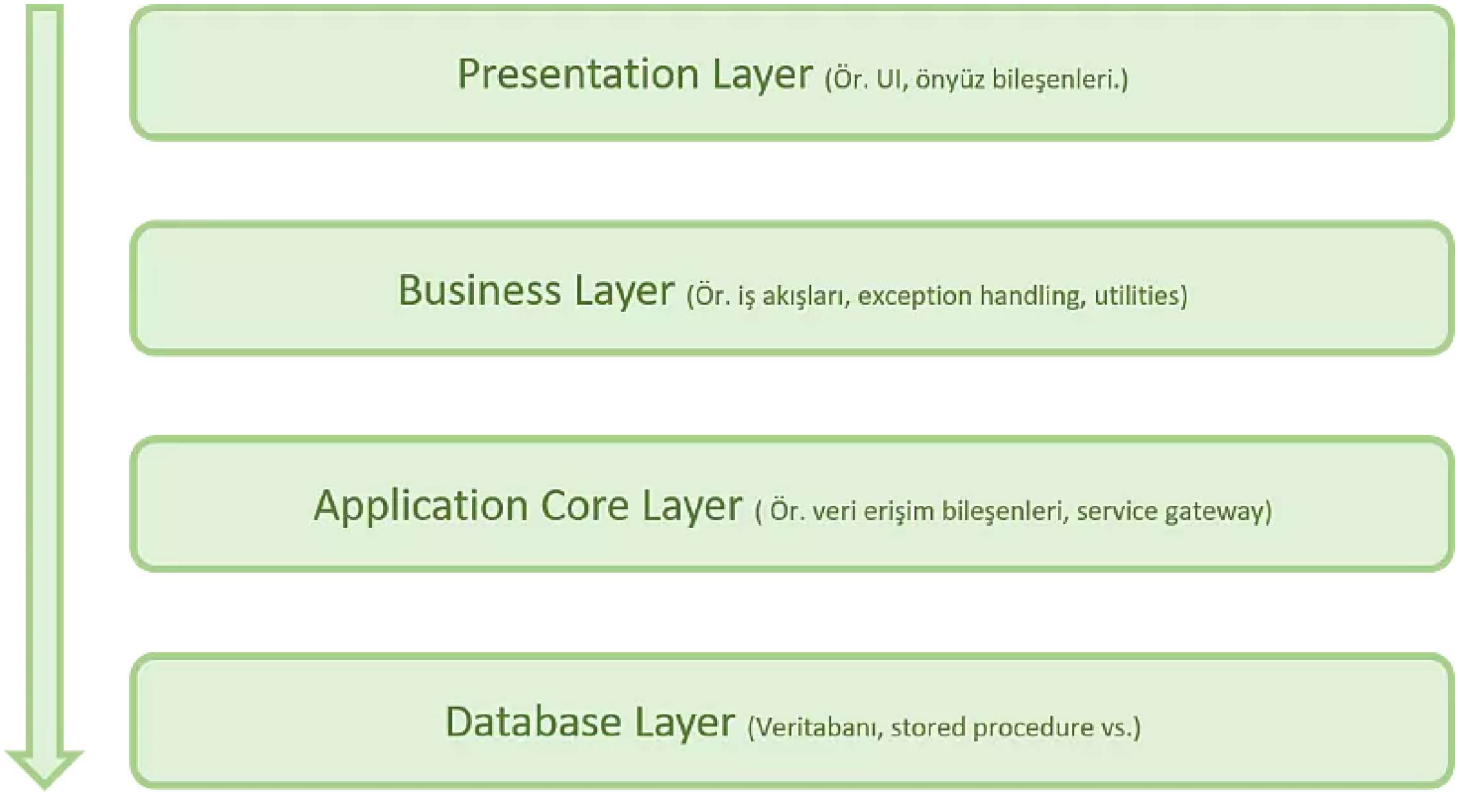


Bu üç bileşen, bağımsız olarak tasarlanabildikleri, geliştirilebildikleri ve konuşlandırılabildikleri için kullanımı kolaydır. Verilerin nasıl sunulacağından bağımsız olarak değişmesine olanak sağlar. Öte yandan, MVC mimarisindeki karmaşıklık yüksektir. Verileri görüntülemenin ve bunlarla etkileşim kurmanın birden çok yolu olduğunda kullanılabilir.

Katmanlı Yazılım Mimarisi (Layered Architecture)

Layered architecture yani katmanlı mimari model, bağımsız ve ayrı çalışma mantığı için en çok kullanılan alternatif yollarından biridir. Multitier veya n-tier olarak da ifade edilir. Farklı soyutlama düzeylerinde ayrılabilecek alt işlevsellik içeren uygulamalar için kullanışlıdır. Bu modelde her katman farklı bir işlevsellik sunar. En dış katman, verilerin sisteme girildiği yerdir. Genel olarak ilk katman presentation layer olarak dizayn edilir, sonrasında business layer, remote service layer, persistence layer, database layer gibi katmanlar olur.

Her katman bir üst katmana hizmet verir. Katmanlar, tek yönlü bir şekilde etkileşime girer; yani her katman kendi hizmeti için altındaki katman tarafından sunulan hizmete bağlıdır.



Uygulamada her bir işlevsellik farklı bir katman olarak ayarlanması ve bu katmanların ayrı ayrı kullanılması avantaj sağlar. Bu mimari stilde diğer katmanlardaki belirli değişikliklerden diğer katmanlar etkilenmez bundan dolayı yeniden düzenlemeyi kolaylaştırır. Bir katmanın ara yüzünün değişmesi sadece ona bitişik katmanı etkiler. Katmanların ayrılması, mantıksal olduğu kadar fiziksel de olabilir. Ara yüzü değişmediği sürece, bir katman başka bir eşdeğer katmanla değiştirilebilir.

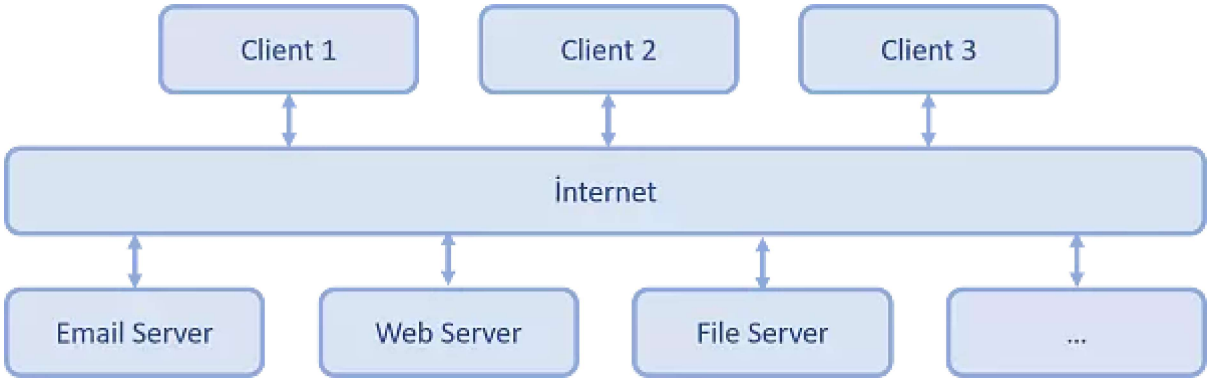
Pratikte, katmanlar arasında temiz bir ayrım sağlamak genellikle zordur ve üst düzey bir katmanın, hemen altındaki katmandan ziyade alt düzey katmanlarla doğrudan etkileşime girmesi gerekebilir. Her katmanda işlenirken bir hizmet talebinin birden çok düzeyde yorumlanması nedeniyle performans bir sorun olabilir.

Ne zaman kullanılır? Mevcut sistemlerin üzerine yeni özellikler eklenirken, yeni işlevsellik inşa ederken, her katmanın sorumluluğu ayrı ekiplere verildiğinde, çok seviyeli güvenlik gereklilik olduğunda kullanılır.

İstemci-Sunucu Yazılım Mimarisi (Client-Server Architecture)

Bu modelde esas olarak 2 ana bileşen vardır: İstemci (Client) ve Sunucu (Server). İstemci, isteği yapan sunucu ise isteği karşılayandır yani servisi sağlayandır. Bu modelde, en az bir bileşen sunucu rolünde, en az bir bileşen de istemci rolündedir. Bu iki bileşen, genellikle farklı katmanlarda olsalar da aynı altyapı üzerinde yer alabilirler.

İstemci, bunları işleyen ve yanıt veren sunucuya istek gönderir. Sunucu, bir istemciden gelen isteği kabul ettiğinde, istemciyle belirli bir protokol üzerinden bir bağlantı açar. Sunucular "stateful" veya "stateless" olabilir. Sunucu istemciden gelen isteklerin kaydını tutuyorsa bu kayda oturum denir.



Bu model verileri ve bunların işlendiğini gösteren bir dizi bileşen arasında nasıl dağıtıldığını gösteren dağıtılmış sistem modelidir. Paylaşılan veritabanının farklı lokasyonlardan, farklı client sistemlerden erişilmesi gerektiğinde ve sistem üzerindeki yük değişken olduğunda kullanılabilir.

Client-server yani istemci-sunucu modelinin en önemli avantajı dağıtık bir mimari olmasıdır. Sunucu tarafının sunduğu bir servis tüm istemciler tarafından kullanılabilir. Bu servis için de tüm servislerin sağlanmasına gerek yoktur. Bir diğer önemli avantajı sağlanan servislerin bağımsız ve birbirinden ayrı olmasıdır. Yeni bir sunucu eklemek ve sistemin geri kalanıyla entegre etmek ya da sistemin diğer bölümlerini etkilemeden sunucuları güncellemek kolaydır. Öte yandan dezavantajı olarak da servis, merkezi bir sunucu tarafından veriliyorsa "single point of failure" olarak ifade edilen durum oluşur yani sunucuda sorun olduğu takdirde bu hizmet sağlanamayacaktır. Performans sorunları, sisteme olduğu kadar ağa da bağlı olduğu için öngörülemeyebilir. Ayrıca sunucular farklı kuruluşlara aitse yönetim sorunları olabilir.

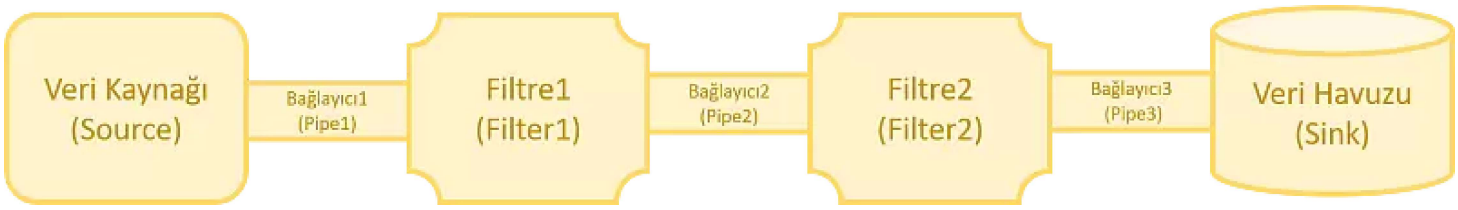
İstemciler, mevcut sunucuların adlarını ve sağladıkları servisleri bilmek zorunda kalabilir. Ancak sunucuların, istemcilerin kim olduğunu, sağladıkları servislere kaç istemcinin eriştiğini bilmesine gerek yoktur.

Boru & Filtre Yazılım Mimarisi (Pipe & Filter Architecture)

Pipe and Filter (boru ve filtre) mimarisi modeli, sıralı işleme yapan modüler ayrıştırması sayesinde çok güçlü ve sağlam bir mimari olarak kabul edilir. Bu mimaride ayrı ayrı filtreleme bileşenleri bulunur. Filtreler arasında da yerel bağlantı bulunur. Bu bağlantılara **pipe** denir. Her bileşenin bir dizi input'u ve output'u vardır. Her filtre kendisine gelen veriyi işler ve sıradaki filtreye aktarır. Filtreler verileri zenginleştirebilir, detayları gizleyebilir, başka bir gösterime

dönüştürebilir vb. Bu filtrelerden birinden diğerine verinin aktarılmasını sağlayan pipe'lar (bağlayıcılar) vardır.

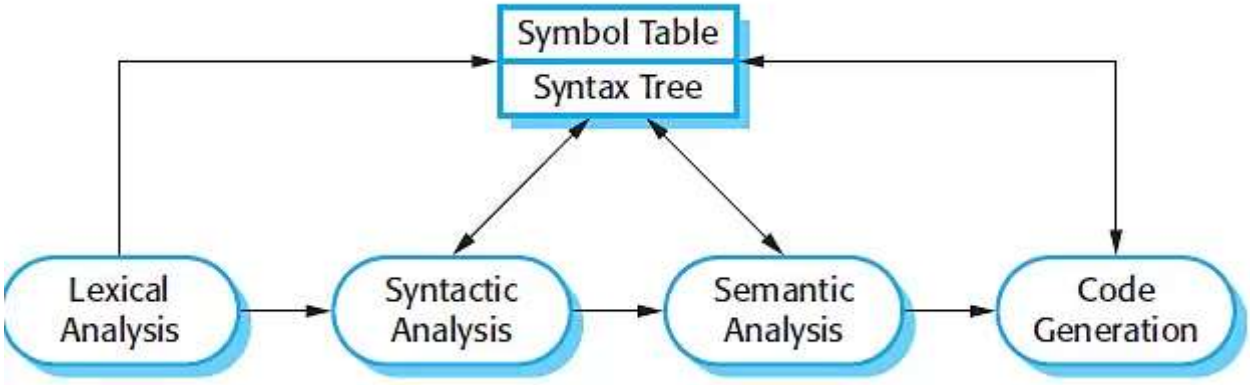
Veriler, bitişik filtreler arasındaki bağlayıcılardan geçirilir. Tüm filtreler aynı anda işler. Bu sayede farklı iş parçacıkları (threads) veya eşyordamlar (coroutines) aynı anda çalıştırılabilir veya filtreler farklı makinelerde yer alabilir. İşlemin sonunda veriler, **sink** adı verilen bir veri hedefine aktarılmış olur.



Bu mimari eşzamanlılık sağlar. Filtreler, bir diğerine bağlı olmadığı sürece aynı anda çalışabilmeleri sayesinde verilerin işlenmesi daha hızlıdır. Bu mimarinin en güzel özelliklerinden biri, aynı anda çalışan yeni filtre ve bağlayıcıların kolayca eklenebilmesi ve böylece hızlı bir şekilde büyütülebilmesidir. Ayrıca bağımsız filtreleri yeniden kullanmak kolaydır. Öte yandan bu mimari, filtreler arasında daha fazla etkileşime ihtiyaç duyan ve kullanıcıları ile anında geri bildirim veya etkileşim gerektiren sistemler için ideal değildir.

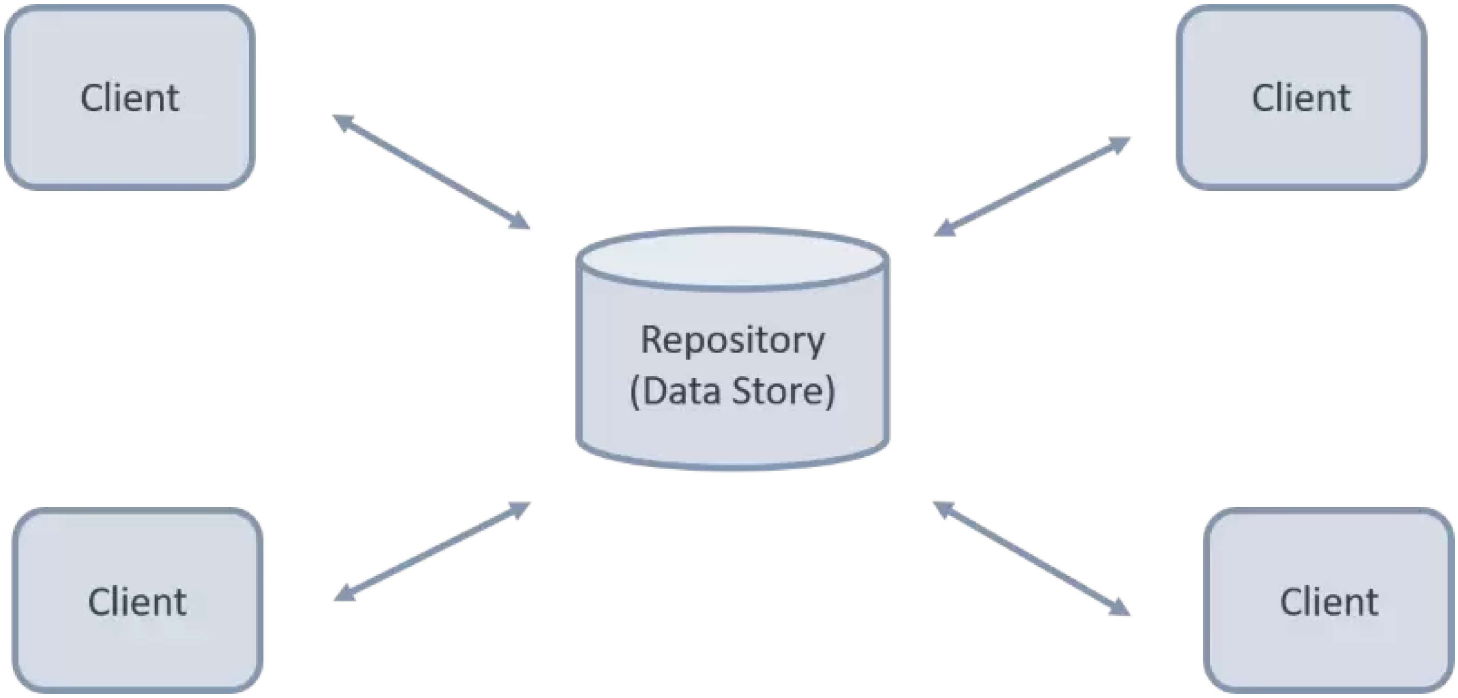
Bu mimari model, birden çok adım olarak ayrılabilen büyük süreçler için uygundur. En iyi bilinen örneği **UNIX shell'dir.** Görüntü işleme ve derleyiciler de bu mimari modelinin iyi bilinen ve yaygın olarak kullanılan alanlarıdır.

Ian Sommerville'dan (Sommerville, I. "Software Engineering 9th Edition) örnek bir pipe and filter yapısına sahip compiler mimarisi:



Depo Yazılım Mimarisi (Repository Architecture)

Bir sistemdeki tüm veriler, tüm sistem bileşenlerinin erişebildiği merkezi bir havuzda (repository'de) yönetilir. Bileşenler direkt etkileşime girmez yalnızca repository aracılığıyla doğrudan etkileşime girer. Repository mimarisi, merkezi bir veri yapısından (genellikle bir veritabanı) ve merkezi veri yapısı ile çalışan bağımsız bileşenlerden oluşur.



Bileşenler bağımsız olabilir, diğer bileşenlerin varlığını bilmeleri gerekmez. Bileşenler veri deposuna yani repository'ye veri ekleyebilir, buradan veri çekebilir. Bir bileşen tarafından yapılan değişiklikler tüm bileşenlere yayılabilir. Tüm veriler tek bir yerde olduğu için tutarlı bir şekilde yönetilebilir (örneğin, aynı anda yapılan yedeklemeler.)

Bazı dezavantajları olarak; Repository de **single point of failure** olduğu için repository'deki sorun tüm sistemi etkiler. Repository aracılığıyla tüm iletişimi organize etmede yetersizlikler olabilir. Ayrıca farklı makineler arasında dağıtmak zor olabilir.

Bu mimari model, uzun süre saklanması gereken büyük hacimli bilgilerin üretildiği bir sistem için kullanılmalıdır. Veritabanı yönetim sistemlerinde yaygın olarak kullanılır. Ayrıca, veri havuzuna veri eklenmesinin bir eylemi veya aracı tetiklediği veri odaklı sistemlerde de kullanabilirsiniz.

Servis Odaklı Mimari (Service Oriented Architecture)

Servis odaklı mimari, service oriented architecture ya da bilinen kısa ismiyle SOA ayrı bir yazı konusu. Service Oriented Architecture yazısında SOA'nın tüm temel kavramlarını, tarihçesini, prensiplerini, avantajlarını hatta mikroservis mimarisi (microservice architecture) ile aralarındaki farklarını bulabilirsiniz.

Mikroservis Mimarisi (Microservice Architecture, Microservices)

SOA'da olduğu gibi aynı şekilde mikroservis mimarisini de ayrı bir yazıda işliyoruz. Mikroservis uygulamaları ve monolitik uygulamaların detaylı açıklamalarını, mikroservis mimarisinin prensiplerini, avantajlarını ve teknoloji önerilerini kapsamlı açıklamalarıyla Microservices yazımızda bulabilirsiniz.