TÜRKÇE ÖZET

Acing the System Design Interview

Zhiyong Tan

Forewords by Anthony Asta and Michael Elder

MANNING

Kitap şu bölümlerden oluşmaktadır:

5.5 İşlem denetleyici
5.6 Saga
Bölüm 6: İşlevsel Bölümleme için Ortak Hizmetler
6.1 Çeşitli servislerin yaygın işlevsellikleri
6.2 Hizmet ağı/desen
6.3 Meta veri servisi
6.4 Servis keşfi
6.5 Kütüphane mi yoksa hizmet mi?
6.6 Ortak API paradigmaları
Bölüm 7: Craigslist Tasarımı
7.1 Kullanıcı hikayeleri ve gereksinimler
7.2 API
7.3 SQL veritabanı şeması
7.4 İlk yüksek seviyeli mimari
.**Bölüm 8**: Oran Sınırlama Servisi Tasarımı
8.1 Oran sınırlama servisine alternatifler
8.2 Oran sınırlama ne zaman yapılmaz?
8.3 İşlevsel gereksinimler
8.4 İşlevsel olmayan gereksinimler
8.5 Kullanıcı hikayeleri ve gerekli servis bileşenleri
8.6 Yüksek seviyeli mimari
8.7 Durum bilgili yaklaşım/bölümlendirme
8.8 Her hosta tüm sayaçları kaydetme
8.9 Oran sınırlama algoritmaları
8.10 Yan arabirim modeli kullanma
8.11 Loglama, izleme ve uyarı verme
Bölüm 9: Bildirim/Uyarı Servisi Tasarımı
9.1 İşlevsel gereksinimler
9.2 İşlevsel olmayan gereksinimler
9.3 İlk yüksek seviyeli mimari
9.4 Nesne deposu: Bildirim yapılandırma ve gönderme
9.5 Bildirim şablonları
9.6 Planlanmış bildirimler
9.7 Bildirim alıcı grupları
9.8 İptal isteklerinin işlenmesi
9.9 Başarısız gönderimlerin işlenmesi
9.10 Müşteri tarafındaki tekrarlı bildirimlerle ilgili dikkat edilmesi gerekenler
9.11 Öncelik
9.12 Arama
9.13 İzleme ve uyarı verme
Bölüm 10: Veritabanı Toplu Denetim Servisi Tasarımı
10.1 Neden denetim gereklidir?
10.2 SQL sorgusunun sonucuna koşullu ifadeyle doğrulama tanımlama
10.3 Basit SQL toplu denetim servisi

10.4 Gereksinimler
10.5 Yüksek seviyeli mimari
10.6 Veritabanı sorguları üzerindeki kısıtlamalar
10.7 Aynı anda çok fazla sorgu çalıştırmayı önleme
Bölüm 11: Otomatik Tamamlama/Typeahead Tasarımı
11.1 Otomatik tamamlama kullanımları
11.2 Arama vs. otomatik tamamlama
11.3 İşlevsel gereksinimler
11.4 İşlevsel olmayan gereksinimler
11.5 Yüksek seviyeli mimari planlaması
11.6 Ağırlıklı trie yaklaşımı ve ilk yüksek seviyeli mimari
11.7 Detaylı uygulama
11.8 Örnekleme yaklaşımı
11.9 Depolama gereksinimlerinin işlenmesi
Bölüm 12: Flickr Tasarımı
12.1 Kullanıcı hikayeleri ve işlevsel gereksinimler
12.2 İşlevsel olmayan gereksinimler
12.3 Yüksek seviyeli mimari
12.4 SQL şeması
12.5 CDN üzerinde dizin ve dosyaların düzenlenmesi
12.6 Fotoğraf yükleme
12.7 Görselleri ve verileri indirme
12.8 İzleme ve uyarı verme
Bölüm 13: İçerik Dağıtım Ağı (CDN) Tasarımı
13.1 CDN'in avantajları ve dezavantajları
13.2 Gereksinimler
13.3 CDN kimlik doğrulama ve yetkilendirme
13.4 Yüksek seviyeli mimari
13.5 Depolama servisi
13.6 Yaygın işlemler
13.7 Önbellek geçersiz kılma
13.8 İzleme ve uyarı verme
Bölüm 14: Mesajlaşma Uygulaması Tasarımı
14.1 Gereksinimler
14.2 İlk düşünceler
14.3 Yüksek seviyeli tasarım
14.4 Bağlantı servisi
14.5 Mesaj gönderme servisi
14.6 Mesaj servisi
14.7 Mesaj gönderim adımları
14.8 Arama
14.9 İzleme ve uyarı verme
Bölüm 15: Airbnb Tasarımı
15.1 Gereksinimler

Bölüm 1: Sistem Tasarım Kavramlarına Giriş

1.1 Karar Noktaları

Sistem tasarımı, mülakatlarda oldukça önemli bir aşamadır. Bu süreçte adayın teknik yeterliliklerinin yanı sıra, farklı tasarım seçeneklerini değerlendirip uygun olanı seçme becerisi de sınanır. Sistem tasarımı, belirli bir çözümün tüm yönlerini göz önüne alarak ticari ve teknik gereksinimler doğrultusunda kararlar almayı içerir. Bu bölümde, bir sistem tasarım mülakatının neden önemli olduğu ve bu mülakatlarda hangi faktörlerin dikkate alınması gerektiği üzerinde durulmaktadır.

1.2 Kitabı Okumanız Gerekli mi?

Bu kitabın, sistem tasarım mülakatlarına hazırlanan mühendisler için önemli bir kaynak olduğu anlatılmaktadır. Eğer yazılım mühendisliği alanında kariyer yapıyorsanız ve sistem tasarımı mülakatlarına katılmayı planlıyorsanız, bu kitabın size fayda sağlayacağı ifade edilmektedir. Kitap, özellikle orta ve ileri düzeyde teknik bilgiye sahip okuyuculara yönelik hazırlanmıştır.

1.3 Kitabın Genel Bakışı

Kitap, iki ana bölümden oluşur. İlk bölüm, sistem tasarımıyla ilgili kavramların derinlemesine açıklamalarını içerirken, ikinci bölümde ise pratik sistem tasarımı mülakat örneklerine yer verilmiştir. Her iki bölümde de gerçek dünya örnekleri ve sistem tasarımı süreçlerinde karşılaşılabilecek zorluklar ve bu zorluklarla başa çıkma yolları anlatılmaktadır.

1.4 Sistemin Çeşitli Servislerini Ölçeklendirme

Bir sistem tasarımının temel amaçlarından biri, sistemin çeşitli servislerini ölçeklendirerek yük altındaki performansını artırmaktır. Bu bölümde, bir uygulamanın başlangıçta küçük bir dağıtım yapısına sahip olacağı ve zamanla kullanıcı sayısı arttıkça bu yapının ölçeklenmesi gerektiği anlatılmaktadır. Sistemin ölçeklenebilirliği, kullanıcı sayısındaki ve isteklerdeki artışlara maliyet etkin ve kolay bir şekilde yanıt verebilecek şekilde tasarlanmalıdır. Bu süreçte kullanılan araçlar ve yöntemler de detaylı olarak ele alınmaktadır, örneğin: GeoDNS, önbellekleme, içerik dağıtım ağları (CDN), yatay ölçeklendirme, sürekli entegrasyon ve sürekli teslimat (CI/CD).
Bu bölüm, sistem tasarımında karşılaşılan temel kavramların anlaşılması ve nasıl ölçeklendirme yapılabileceği konularında rehberlik sunar.

Bölüm 2: Tipik Bir Sistem Tasarım Mülakatının Akışı

2.1 Gereksinimlerin Netleştirilmesi

Sistem tasarımı mülakatlarının başlangıcında, sorunun ne olduğu tam olarak anlaşılmalı ve gereksinimler netleştirilmelidir. Mülakat sırasında adaya genellikle çok genel bir soru sorulur, bu yüzden adayın ilk görevi, sistemin işlevsel ve işlevsel olmayan gereksinimlerini netleştirmek ve soruyu detaylandırmaktır. Bu aşamada, ölçeklenebilirlik, performans, kullanılabilirlik ve güvenlik gibi önemli noktalar üzerine sorular sorulmalı ve taleplerin ne olduğu konusunda net bir tablo oluşturulmalıdır.

2.2 API Taslağının Hazırlanması

Bir sistemin dış dünya ile nasıl iletişim kuracağı önemli bir sorundur. Bu nedenle, mülakatta genellikle bir API taslağı oluşturmanız beklenir. API taslağı, sistemin sunacağı fonksiyonların ve servislerin tanımını içermelidir. Sık kullanılan API uç noktaları (endpoints), veri giriş/çıkışları ve bu verilerin nasıl işleneceği bu aşamada belirlenir.

2.3 Kullanıcılar ve Veriler Arasındaki Bağlantılar

Bu adımda, sistemin kullanıcılar ile veriler arasındaki ilişkileri nasıl yöneteceği tartışılır. Örneğin, kullanıcıların sisteme nasıl bağlanacağı, hangi işlemleri yapacağı ve bu işlemlerle hangi verilerin yönetileceği detaylandırılır. Kullanıcıdan gelen isteklerin nasıl işleneceği, sistemin nasıl yanıt vereceği, verilerin nasıl okunup yazılacağı gibi konular ele alınır.

2.4 Veri Modeli Tasarımı

Veri modelinin nasıl tasarlanacağı, özellikle büyük ve karmaşık sistemlerde çok önemlidir. Verilerin nasıl organize edileceği, hangi veri tabanı yapılarının kullanılacağı, tablolar ve ilişkilerin nasıl yapılandırılacağı bu aşamada planlanır. Ayrıca, farklı hizmetlerin aynı veritabanını paylaştığında ortaya çıkabilecek sorunlar ve bu sorunların nasıl çözülebileceği üzerine tartışmalar yapılır.

2.5 Loglama, İzleme ve Uyarılar

Sistemin operasyonel anlamda sürdürülebilir olması için loglama, izleme ve uyarı mekanizmaları tasarlanmalıdır. Loglama, sistemde neler olup bittiğini takip etmek için gereklidir. İzleme (monitoring), sistemin performansını, kullanılabilirliğini ve sağlığını sürekli izlemek anlamına

gelirken, uyarılar (alerting) belirli olaylar gerçekleştiğinde harekete geçilmesini sağlar. Bu süreçler, sistemin kararlı bir şekilde çalışmasını sağlamak için kritik öneme sahiptir.

2.6 Arama Çubuğu

Eğer sistemde bir arama fonksiyonu varsa, bu özelliğin nasıl uygulanacağı üzerinde durulur. Arama motoru olarak hangi teknolojinin kullanılacağı (örneğin Elasticsearch), arama verilerinin nasıl organize edileceği ve arama sonuçlarının nasıl getirileceği mülakat sırasında tartışılan konular arasındadır.

2.7 Diğer Tartışmalar

Bu aşamada, sistemin bakımının nasıl yapılacağı, yeni özelliklerin nasıl eklenebileceği ve gelecekteki olası ihtiyaçlara nasıl adapte edilebileceği gibi konular tartışılır. Ayrıca, kullanıcı deneyimini geliştirmek için neler yapılabileceği, sınır durumlar (edge cases) ve yeni kısıtlar hakkında düşünceler paylaşılır. Bulut teknolojilerinin kullanımı ve sistemin bulut-yerel (cloud-native) olması da bu kapsamda ele alınabilir.

2.8 Mülakat Sonrası Yansıma ve Değerlendirme

Mülakat sona erdikten hemen sonra, adayın kendi tasarımını değerlendirmesi beklenir. Bu değerlendirme sürecinde, mülakat sırasında tartışılmamış detaylar üzerine düşünmek ve iyileştirilebilecek noktaları belirlemek önemlidir. Ayrıca, mülakatta verilen geribildirimler ve yapılacak geliştirmeler üzerinde de durulmalıdır. Bu, gelecekteki mülakatlar için daha hazırlıklı olmayı sağlar.

Bu bölüm, tipik bir sistem tasarım mülakatının nasıl ilerlediği ve hangi konuların ele alındığı konusunda adaylara rehberlik eder.

Bölüm 3: İşlevsel Olmayan Gereksinimler

İşlevsel olmayan gereksinimler, bir sistemin nasıl çalışacağını belirleyen gereksinimlerdir ve sistemin işlevselliği kadar önemlidir. Bu bölümde, sistem tasarımında dikkate alınması gereken işlevsel olmayan gereksinimlere odaklanılır.

3.1 Ölçeklenebilirlik

Bir sistemin ölçeklenebilir olması, artan kullanıcı ve işlem sayısına kolayca uyum sağlayabilmesi anlamına gelir. Sistemin artan taleplere cevap verebilmesi için yatay ve dikey ölçeklendirme stratejilerinin kullanılması gerekir. Ayrıca, servislerin dağıtık yapıda olması ve sistemin yeni kaynaklarla donatılabilir olması, ölçeklenebilirliği arttıran faktörlerdendir.

3.2 Kullanılabilirlik

Kullanılabilirlik, bir sistemin ne kadar süre boyunca erişilebilir ve çalışır durumda olacağını ifade eder. Sistemin yüksek kullanılabilirlik sunması için sürekli izleme, otomatik iyileştirme (self-healing) mekanizmaları ve hata toleransı gibi stratejiler kullanılır. Ayrıca, sistemin kesintisiz hizmet verebilmesi için yük dengeleme ve yedekli sistemler gibi çözümler de ele alınmalıdır.

3.3 Hata Toleransı

Hata toleransı, sistemin herhangi bir bileşeninin arızalanması durumunda çalışmaya devam edebilme yeteneğidir. Hata toleransını sağlamak için replikasyon, yedeklilik ve hata yönetim stratejileri kullanılır. Örneğin, servislerin yedekli olması ve veri replikasyon teknikleri ile arızalara karşı dayanıklılık sağlanır.

3.4 Performans ve Gecikme Süresi

Sistem tasarımında performans, sistemin hızlı bir şekilde yanıt verebilmesi anlamına gelirken, gecikme süresi, bu yanıtların ne kadar hızlı olduğunu ifade eder. Sistemlerin yüksek performans sunabilmesi için optimize edilmiş veri yapıları, verimli algoritmalar, önbellekleme ve uygun veri tabanı çözümleri kullanılır. Ayrıca, ağ gecikmeleri ve işlem süreleri de dikkate alınmalıdır.

3.5 Tutarlılık

Tutarlılık, bir sistemdeki tüm verilerin güncel ve uyumlu olmasını sağlar. Dağıtık sistemlerde tutarlılığın korunması özellikle zor olabilir. Bu nedenle, CAP teoremi çerçevesinde sistemin tutarlılık ile kullanılabilirlik veya bölünebilirlik arasında dengelenmesi gerekebilir. Dağıtık veritabanları için tutarlılığı sağlamak, zaman damgaları ve koordinasyon servisleri gibi çözümler gerektirebilir.

3.6 Doğruluk

Sistemin doğruluğu, ürettiği sonuçların ve işlediği verilerin doğru olmasıyla ilgilidir. Doğruluk, özellikle finansal veya sağlık hizmetleri gibi alanlarda kritik öneme sahiptir. Doğru sonuçlar üretmek için veri işleme süreçlerinde doğrulama mekanizmaları ve denetimler kullanılır.

3.7 Karmaşıklık ve Bakım

Bir sistemin ne kadar karmaşık olduğu, onun ne kadar kolay veya zor bakım yapılabilir olduğunu etkiler. Sistem tasarımında karmaşıklığın azaltılması, kodun ve mimarinin modüler, esnek ve anlaşılır olmasını sağlar. Bu sayede sistemin bakım süreci kolaylaşır ve yeni özellikler eklemek veya hataları düzeltmek daha hızlı hale gelir.

3.8 Maliyet

Sistem tasarımı yapılırken maliyet önemli bir faktördür. Hem donanım hem de yazılım kaynaklarının maliyetleri göz önünde bulundurulmalıdır. Aynı zamanda, bakım maliyetleri, enerji tüketimi ve kullanılan bulut hizmetlerinin ücretlendirme modelleri de dikkate alınmalıdır. Maliyet optimizasyonu için kaynakların verimli kullanılması ve gereksiz harcamaların önüne geçilmesi gerekir.

3.9 Güvenlik

Sistem tasarımında güvenlik, veri ve sistem bileşenlerinin yetkisiz erişimlerden korunmasını sağlar. Güvenli bir sistem tasarımı için kimlik doğrulama, yetkilendirme, veri şifreleme, güvenlik duvarları ve diğer güvenlik protokollerinin entegre edilmesi gereklidir. Ayrıca, güvenlik denetimleri ve güvenlik açıklarının izlenmesi gibi sürekli bir izleme süreci de şarttır.

3.10 Gizlilik

Gizlilik, sistemin kullanıcıların kişisel ve hassas verilerini nasıl koruduğunu ele alır. Veri güvenliği politikaları ve veri koruma yasalarına uyum sağlanması bu konuda önemlidir. Veri şifreleme, anonimleştirme ve veri erişim politikaları gibi çözümlerle kullanıcıların verilerinin korunması sağlanır.

3.11 Bulut Yerli Çözümler

Bulut tabanlı çözümler, modern sistemlerde yaygın olarak kullanılmaktadır. Bulut yerli çözümler, sistemin bulut ortamında çalışmak üzere optimize edilmesini içerir. Bu, sistemin otomatik olarak ölçeklenebilir, esnek ve maliyet etkin olmasını sağlar. Ayrıca, bulut yerli sistemlerde yedeklilik, otomatik iyileşme ve dağıtık yapı doğal olarak sağlanır.

Bu bölüm, sistem tasarımı sırasında dikkate alınması gereken işlevsel olmayan gereksinimlerin detaylı bir şekilde ele alındığı ve bu gereksinimlerin sistemin başarısı üzerindeki kritik rolünü vurgulamaktadır.

Bölüm 4: Veritabanlarını Ölçeklendirme

4.1 Depolama Servislerine Genel Bakış

Depolama hizmetleri, bir sistemin büyüklüğü ve veri miktarı arttıkça kritik hale gelir. Depolama servisleri, verilerin güvenli ve verimli bir şekilde saklanmasını sağlamak için farklı çözümler sunar. Bu bölümde, veritabanlarının farklı türleri, veri depolama yapıları ve bu yapıların nasıl ölçeklenebileceği tartışılmaktadır. Veritabanı türleri arasında ilişkisel veritabanları (RDBMS), NoSQL veritabanları ve dağıtık depolama çözümleri bulunmaktadır.

4.2 Veritabanları Ne Zaman Kullanılmalı?

Bir sistemin ihtiyaçlarına bağlı olarak, veritabanlarının ne zaman ve hangi koşullarda kullanılacağı karar verilmelidir. Veritabanı seçimi, verinin büyüklüğüne, veri yapısına, işleme hızı gereksinimlerine ve erişim sıklığına göre yapılır. Ayrıca, performans ve ölçeklenebilirlik açısından hangi veritabanı türünün en uygun olduğuna karar verilmelidir. Bu bölüm, ilişkisel ve NoSQL veritabanlarının ne zaman tercih edileceği konusunda rehberlik sunar.

4.3 Replikasyon

Replikasyon, verilerin birden fazla sunucuda çoğaltılması işlemidir. Bu sayede, sistemin hata toleransı ve kullanılabilirliği artırılır. Replikasyon, verilerin güvenli bir şekilde saklanmasını sağlamak için yaygın bir yöntemdir. Tek liderli replikasyon, çok liderli replikasyon ve lider olmayan replikasyon gibi farklı replikasyon stratejileri ele alınır. Bu bölümde, replikasyonun nasıl yapılacağı ve çeşitli replikasyon stratejileri arasındaki farklar açıklanır.

4.4 Shard'lanmış Veritabanları ile Kapasiteyi Artırma

Shard'lama, büyük bir veritabanını daha küçük parçalara bölerek kapasiteyi artırma işlemidir. Bu işlem sayesinde, veri tabanının performansı artırılır ve yüksek hacimli veri talepleri daha verimli bir şekilde karşılanır. Shard'lanmış veritabanlarının nasıl yapılandırılacağı ve shard'lama stratejileri bu bölümde detaylandırılır. Ayrıca, shard'lama işleminin getirdiği potansiyel zorluklar ve bu zorlukların nasıl aşılabileceği de tartışılmaktadır.

4.5 Olayları Toplama

Olay toplama, sistemde gerçekleşen olayların kaydedilmesi ve bu olayların bir araya getirilerek analiz edilmesi işlemidir. Verilerin nasıl toplandığı, işlendiği ve depolandığı bu bölümde ele alınır. Ayrıca, tek aşamalı ve çok aşamalı olay toplama süreçleri incelenir. Olayların doğru bir şekilde toplanması, sistemin genel performansı ve verilerin analizi açısından büyük önem taşır.

4.6 Toplu ve Akışlı ETL

ETL (Extract, Transform, Load), verilerin bir kaynaktan çıkarılması, dönüştürülmesi ve hedef bir sisteme yüklenmesi sürecidir. Bu süreç, veri ambarları ve büyük veri analizleri için kritik bir öneme sahiptir. Toplu ETL, verilerin belirli aralıklarla toplu olarak işlenmesini sağlarken, akışlı ETL, verilerin sürekli olarak işlenmesini sağlar. Bu bölüm, toplu ve akışlı ETL süreçlerinin nasıl çalıştığını ve hangi durumlarda kullanılacağını açıklar.

4.7 Denormalizasyon

Denormalizasyon, veritabanı performansını artırmak için normalleştirilmiş veri yapılarının kısmen veya tamamen çözülmesi işlemidir. Bu süreçte, veritabanında daha az sorgu yapmak ve veri erişimini hızlandırmak amacıyla aynı verinin birden fazla yerde tutulması sağlanır. Denormalizasyonun getirdiği avantajlar ve dezavantajlar bu bölümde ele alınır. Ayrıca, denormalizasyonun hangi durumlarda faydalı olacağı ve uygulanma yolları üzerinde durulmaktadır.

4.8 Önbellekleme

Önbellekleme, sıkça kullanılan verilerin bellekte tutulmasını sağlayarak veri erişim süresini azaltır ve veritabanı üzerindeki yükü hafifletir. Önbellekleme stratejileri, okuma ve yazma işlemleri için farklı yöntemler sunar. Bu bölümde, veritabanı sistemlerinde önbellekleme işlemlerinin nasıl yapılacağı ve hangi stratejilerin kullanılacağı tartışılmaktadır. Ayrıca, önbellek geçersiz kılma ve önbellek ısınması gibi konulara da değinilir.
Bu bölümde, veritabanlarının nasıl ölçeklendirileceği ve bu süreçte hangi yöntemlerin kullanılacağı detaylı bir şekilde ele alınmaktadır. Veritabanı yönetimi ve veri işleme teknikleri hakkında kapsamlı bilgi sağlanmaktadır.

Bölüm 5: Dağıtık İşlemler

5.1 Olay Tabanlı Mimari (EDA)

Olay tabanlı mimari, sistemin olaylar üzerinden iletişim kurduğu bir tasarım yaklaşımıdır. Bu mimaride, sistemde gerçekleşen her önemli olay (örneğin bir kullanıcı işlemi) bir mesaj olarak yayımlanır ve bu mesaj, bu olayla ilgilenen diğer servisler tarafından işlenir. Bu bölümde, olay tabanlı mimarinin nasıl çalıştığı ve sistemin modülerliğini ve esnekliğini nasıl artırdığı açıklanmaktadır. Olay tabanlı mimari, özellikle yüksek düzeyde ölçeklenebilirlik ve gevşek bağlılık gerektiren sistemlerde kullanılır.

5.2 Olay Kaynaklandırma (Event Sourcing)

Olay kaynaklandırma, bir sistemin durumunu doğrudan güncellemeler yerine olaylar yoluyla yönetme yaklaşımıdır. Bu modelde, her durum değişikliği bir olay olarak kaydedilir ve sistemin mevcut durumu bu olaylar serisinin uygulanmasıyla elde edilir. Olay kaynaklandırma, veritabanında güncel durumu değil, olayları saklayarak geçmişteki tüm işlemlerin izlenebilir olmasını sağlar. Bu yaklaşım, geçmiş veri ve işlemler üzerinde analiz yapmayı ve hataları geri almayı kolaylaştırır.

5.3 Veri Yakalama (CDC)

Veri Yakalama (Change Data Capture - CDC), veritabanında yapılan değişikliklerin izlenmesi ve bu değişikliklerin gerçek zamanlı olarak diğer sistemlerle paylaşılması yöntemidir. CDC, özellikle veritabanı güncellemelerini takip etmek ve bu güncellemeleri olay tabanlı sistemlere entegre etmek için kullanılır. Bu bölümde, CDC'nin nasıl çalıştığı ve hangi durumlarda faydalı olduğu üzerinde durulur.

5.4 Olay Kaynaklandırma ve CDC Karşılaştırması

Olay kaynaklandırma ve CDC, veri güncellemelerini izlemek için kullanılan iki farklı yaklaşımdır. Bu bölümde, her iki yöntemin karşılaştırması yapılır. Olay kaynaklandırma, tüm sistemin durumunu olaylar üzerinden yönetirken, CDC sadece veri tabanındaki değişiklikleri izler. Her iki yaklaşımın avantajları, dezavantajları ve hangi durumlarda kullanılacakları detaylandırılır. Örneğin, olay kaynaklandırma, sistemde tam bir olay akışı sağlayarak daha geniş bir kapsam sunarken, CDC daha basit ve veritabanı odaklı bir çözüm sunar.

5.5 İşlem Denetleyici (Transaction Supervisor)

Dağıtık sistemlerde, birden fazla hizmet arasında işlem yönetimi karmaşık olabilir. İşlem denetleyici, bu işlemleri yönetmek ve koordine etmek için kullanılan bir mekanizmadır. Bu mekanizma, özellikle birden fazla veri kaynağına yazılması gereken işlemleri düzenler ve işlemlerin başarılı bir şekilde tamamlanmasını sağlar. Ayrıca, hatalı işlemleri geri alabilme (rollback) gibi yetenekler sunar.

5.6 Saga

Saga deseni, uzun süren dağıtık işlemler için kullanılan bir yöntemdir. Saga, bir dizi yerel işlemden oluşur ve her bir işlem bağımsız olarak başarılı olmalıdır. Eğer bir işlem başarısız olursa, işlemi geri almak için telafi edici işlemler (compensation transactions) devreye girer. Saga, dağıtık sistemlerde tutarlılığı ve işlem yönetimini sağlamak için popüler bir yöntemdir. Saga'nın koreografi ve orkestrasyon olarak iki farklı yaklaşımı vardır ve bu bölümde her iki yaklaşımın nasıl çalıştığı ve hangi durumlarda tercih edilmesi gerektiği açıklanır.
Bu bölümde, dağıtık sistemlerde işlemleri yönetmek için kullanılan çeşitli yaklaşımlar ve desenler ele alınır. Olay tabanlı mimari, olay kaynaklandırma, CDC, işlem denetleyici ve Saga gibi yöntemler, dağıtık işlemlerle başa çıkma yolları sunar.

Bölüm 6: İşlevsel Bölümleme için Ortak Hizmetler
Bu bölümde, sistem tasarımında yaygın olarak kullanılan hizmetlerin işlevsel bölümlemesi ve bu hizmetlerin ortak kullanımını sağlayan mimari desenler ele alınmaktadır. Modern sistemlerde, çeşitli servisler arasında işlevselliklerin paylaşımı önemli bir tasarım unsurudur.
6.1 Çeşitli Servislerin Yaygın İşlevsellikleri
Bir sistemde kullanılan farklı servislerin çoğu, ortak işlevselliklere sahiptir. Örneğin, kimlik doğrulama, yetkilendirme, hata yönetimi, izleme ve loglama gibi işlevler birçok servis tarafından kullanılır. Bu işlevselliklerin merkezi olarak yönetilmesi ve tüm servislere ortak bir şekilde sunulması, sistemin modülerliğini ve bakım kolaylığını artırır. Bu bölümde, bu yaygın işlevselliklerin nasıl tasarlanacağı ve yönetileceği tartışılmaktadır.
6.2 Hizmet Ağı/Desen (Service Mesh)
Hizmet ağı, mikroservisler arasında iletişimi yönetmek için kullanılan bir mimari yaklaşımdır. Servisler arasındaki ağ trafiğini izleyen ve yöneten bir katman sunar. Bu yapı, hizmetler arasındaki iletişimin güvenliğini, izlenebilirliğini ve hata toleransını artırır. Bu bölümde, hizmet ağlarının nasıl çalıştığı, veri trafiğini nasıl yönettiği ve sistem üzerindeki etkileri detaylandırılır. Ayrıca, hizmet ağıyla ilgili bazı popüler araçlar ve desenler incelenir.
6.3 Meta Veri Servisi
Meta veri servisi, sistemdeki diğer servislerin yapılandırma, durum ve diğer operasyonel bilgilerini depolamak ve yönetmek için kullanılan bir hizmettir. Bu hizmet, servislerin durumunu takip eder ve gerekirse diğer servisler tarafından kullanılabilir yapılandırma bilgilerini sunar. Meta veri servisi, özellikle dinamik ve sürekli değişen ortamlarda servislerin güncel kalmasını sağlar.
6.4 Servis Keşfi (Service Discovery)
Servis keşfi, bir sistemdeki servislerin birbirlerini bulabilmesini ve iletişim kurabilmesini sağlayan bir mekanizmadır. Dinamik ölçeklendirme ve sürekli dağıtım süreçlerinde, servislerin konumları ve adresleri sık sık değişebilir. Servis keşfi, bu değişiklikleri izler ve servislerin doğru adreslerle iletişim kurmasını sağlar. Bu bölümde, merkezi ve dağıtık servis keşfi yaklaşımları ele alınır ve hangi durumlarda hangi yöntemlerin kullanılması gerektiği üzerinde durulur.
6.5 Kütüphane mi Yoksa Hizmet mi?
Bir işlevselliği bir hizmet olarak mı yoksa bir kütüphane olarak mı sunmak gerektiği, sistem tasarımında kritik bir karardır. Kütüphaneler, uygulamaya dahil edilirken, hizmetler dışardan çağrılır. Bu iki yaklaşım arasında seçim yapılırken gecikme süreleri, dağıtım zorlukları, ölçeklenebilirlik ve esneklik gibi faktörler dikkate alınmalıdır. Bu bölümde, kütüphane ve hizmet

kullanımı arasındaki farklar ve hangi durumlarda hangi yaklaşımın tercih edilmesi gerektiği açıklanmaktadır.

6.6 Ortak API Paradigmaları

API'ler, sistemlerin birbirleriyle nasıl iletişim kurduğunu tanımlar. Bu bölümde, yaygın kullanılan API paradigmaları olan REST, RPC (Remote Procedure Call), GraphQL ve WebSocket gibi yaklaşımlar incelenir. Her bir paradigmaya ait avantajlar, dezavantajlar ve kullanım durumları detaylandırılır. Ayrıca, bu API'lerin performans, güvenlik ve esneklik açısından nasıl tasarlanabileceği de ele alınır.

Bu bölüm, işlevsel bölümleme ve ortak hizmetler ile ilgili temel tasarım yaklaşımlarını ve desenlerini inceleyerek, sistem tasarımının ölçeklenebilir ve sürdürülebilir olmasını sağlayacak yollar sunmaktadır.

Bölüm 7: Craigslist Tasarımı

Bu bölümde, bir ilan ve ticaret platformu olan Craigslist'in mimari tasarımı ele alınmaktadır. Craigslist gibi bir sistem, kullanıcıların ilan oluşturup görüntülemesine olanak tanıyan geniş bir veri tabanı ve işlem hacmine sahip, yüksek trafikli bir platformdur. Bu bölümde, böyle bir platformun gereksinimleri ve tasarım süreçleri detaylı olarak incelenir.

7.1 Kullanıcı Hikayeleri ve Gereksinimler

Craigslist tasarımında, farklı kullanıcı tiplerinin (ilan verenler, alıcılar, yöneticiler) ihtiyaçları göz önünde bulundurularak kullanıcı hikayeleri oluşturulmalıdır. Kullanıcılar, yeni ilanlar ekleyebilmeli, mevcut ilanları arayabilmeli ve ilgilendikleri ilanlarla etkileşimde bulunabilmelidir. Aynı zamanda, yöneticiler platformda kullanıcıları ve içerikleri yönetebilmelidir. Bu bölümde, platformun temel işlevsel gereksinimleri ve kullanıcı hikayeleri ele alınmaktadır.

7.2 API

Craigslist gibi bir platformda, API tasarımı, kullanıcıların sistemle nasıl etkileşim kuracağını belirler. API uç noktaları, kullanıcıların ilan ekleme, ilan arama ve mevcut ilanları güncelleme gibi işlemleri gerçekleştirmesine olanak tanır. API tasarımı sırasında performans, güvenlik ve esneklik gibi faktörler dikkate alınmalıdır. Bu bölümde, Craigslist'in API taslağı üzerinde durulur ve hangi API uç noktalarının oluşturulması gerektiği açıklanır.

7.3 SQL Veritabanı Şeması

Craigslist gibi bir platform, büyük miktarda yapılandırılmış veri yönetir. Veritabanı tasarımı, ilanların depolanmasını, kategorilere ayrılmasını ve hızlı bir şekilde erişilebilmesini sağlamalıdır. SQL veritabanı kullanarak, ilanlar, kullanıcılar, kategoriler ve konumlar gibi temel varlıkların yönetimi için şema oluşturulmalıdır. Bu bölümde, Craigslist için önerilen SQL veritabanı şeması açıklanmaktadır.

7.4 İlk Yüksek Seviyeli Mimari

Başlangıç aşamasında, Craigslist'in temel mimarisi tasarlanırken, platformun geniş bir kullanıcı tabanını ve veri hacmini nasıl yöneteceği göz önünde bulundurulmalıdır. Yüksek seviyeli mimari, istemci-sunucu modeliyle oluşturulur ve kullanıcı istekleri merkezi bir sunucuya yönlendirilir. Bu bölümde, platformun başlangıçtaki temel bileşenleri ve hizmetleri hakkında bilgi verilir.

7.5 Monolitik Mimari

Başlangıçta, Craigslist'in monolitik bir mimariyle kurulması önerilir. Monolitik mimari, tüm işlevlerin tek bir uygulamada toplandığı, yönetimi ve dağıtımı kolay bir yapıdır. Ancak,

ölçeklenebilirlik ve esneklik açısından zamanla mikroservis mimarisine geçiş gerekebilir. Bu bölümde, monolitik mimarinin avantajları ve dezavantajları ele alınır.

7.6 SQL Veritabanı ve Nesne Deposu Kullanımı

SQL veritabanı ile ilan verilerinin yanı sıra, büyük medya dosyalarının (resim ve videolar) saklanması için nesne depolarının kullanılması gerekebilir. Bu bölümde, ilan verilerinin ilişkisel veritabanında saklanması ve medya dosyalarının nesne depolarında yönetilmesi konusu tartışılır.

7.7 Migrations (Veritabanı Geçişleri) Sorunları

Veritabanı yapısında büyük değişiklikler gerektiğinde, veritabanı geçişleri (migrations) zorlu olabilir. Özellikle, ilan ve kullanıcı verilerinin çok büyük olduğu durumlarda, bu geçişlerin dikkatlice planlanması gerekir. Bu bölümde, geçiş işlemlerinin yönetimi ve potansiyel zorlukları ele alınır.

7.8 İlanları Yazma ve Okuma

Craigslist platformunda, kullanıcıların ilan ekleme ve bu ilanları görüntüleme işlemleri merkezi bir rol oynar. Bu bölümde, ilanların nasıl yazılacağı ve kullanıcıların bu ilanları nasıl okuyacağına dair süreçler detaylandırılır. Ayrıca, bu işlemlerin optimize edilmesi için önerilen teknikler ve veri işleme süreçleri tartışılır.

7.9 İşlevsel Bölümleme

İşlevsel bölümleme, Craigslist gibi büyük bir sistemde farklı hizmetlerin bölünmesini ve her hizmetin ayrı bir bileşen olarak yönetilmesini içerir. Örneğin, ilan yönetimi, kullanıcı yönetimi ve arama işlevleri gibi temel bileşenler farklı mikroservisler olarak ayrılabilir. Bu bölümde, işlevsel bölümlemenin avantajları ve uygulama yolları anlatılmaktadır.

7.10 Önbellekleme

Önbellekleme, sıkça erişilen verilerin hızlı bir şekilde sunulmasını sağlar. Craigslist gibi bir platformda, ilanların ve kullanıcı verilerinin önbelleğe alınması, sistem performansını büyük ölçüde artırabilir. Bu bölümde, hangi verilerin önbelleğe alınacağı ve önbellekleme stratejileri açıklanmaktadır.

7.11 CDN (İçerik Dağıtım Ağı) Kullanımı

Craigslist, statik içerikleri (resim, video vb.) hızlı bir şekilde sunmak için CDN'ler kullanabilir. Bu, özellikle geniş bir kullanıcı tabanına sahip platformlarda, içeriklerin düşük gecikme süresiyle sunulmasını sağlar. Bu bölümde, CDN'in nasıl entegre edileceği ve performansa olan etkileri ele alınır.

7.12 SQL Kümesi ile Okuma İşlemlerini Ölçeklendirme

Craigslist gibi bir platformda, okuma işlemleri çok yoğun olabilir. Bu bölümde, SQL kümesi (SQL cluster) kullanarak okuma işlemlerinin nasıl ölçeklendirileceği ve performansın nasıl artırılacağı tartışılır.

7.13 Yazma İşlem Hacmini Ölçeklendirme

Yazma işlemleri genellikle daha karmaşıktır ve veritabanının yoğun bir şekilde kullanıldığı işlemlerdir. Bu bölümde, yazma işlemlerini ölçeklendirmenin yolları ve veritabanı performansını artırmak için yapılabilecek optimizasyonlar anlatılmaktadır.

7.14 E-posta Servisi

Craigslist gibi bir platformda, kullanıcıların e-posta bildirimleri alması ve etkileşimlerinin izlenmesi için bir e-posta servisi gereklidir. Bu bölümde, e-posta servisi tasarımının nasıl yapılacağı ve sistemle nasıl entegre edileceği anlatılmaktadır.

7.15 Arama

Craigslist, geniş bir veri yelpazesi sunduğundan etkili bir arama motoru kritik öneme sahiptir. Bu bölümde, arama çubuğunun nasıl entegre edileceği, arama algoritmaları ve kullanılan teknolojiler (örneğin Elasticsearch) ele alınmaktadır.

7.16 Eski İlanların Kaldırılması

Craigslist gibi bir platformda, eski ilanların kaldırılması veya arşivlenmesi gerekir. Bu bölümde, eski ilanların nasıl yönetileceği ve veri tabanındaki gereksiz verilerin nasıl temizleneceği üzerine odaklanılır.

7.17 İzleme ve Uyarılar

Bir sistemin sağlıklı çalışmasını sağlamak için izleme ve uyarı mekanizmaları kritik öneme sahiptir. Bu bölümde, Craigslist sisteminde izleme ve uyarı hizmetlerinin nasıl entegre edileceği, hangi metriklerin izleneceği ve olası sorunların nasıl ele alınacağı açıklanmaktadır.

7.18 Mimari Tartışmaların Özeti

Bu bölümde, Craigslist platformunun mimarisi ile ilgili yapılan tüm tartışmaların bir özeti verilir. Tasarım kararları, karşılaşılan zorluklar ve çözümler değerlendirilir.

7.19 Diğer Olası Tartışma Konuları

Craigslist tasarımı ile ilgili başka tartışma konuları da ele alınabilir. Bu bölümde, raporlama, kademeli bozulma (graceful degradation), karmaşıklık yönetimi, kategori ve etiket kullanımı gibi konular tartışılır.

Craigslist gibi geniş çaplı bir sistemin tasarımı, ölçeklenebilirlik, esneklik ve performans açısından dikkatli bir planlama gerektirir. Bu bölümde, Craigslist'in temel mimarisi, veri yönetimi stratejileri ve teknik altyapısı detaylı bir şekilde ele alınır.

Bölüm 8: Oran Sınırlama Servisi Tasarımı

Oran sınırlama (rate limiting), bir sistemin kaynaklarına gelen isteklerin belirli bir hızda sınırlandırılmasını sağlar. Bu, özellikle sistemin aşırı yüklenmesini engellemek, kötüye kullanımı önlemek ve hizmet kalitesini sürdürmek için kritik bir işlemdir. Bu bölümde, oran sınırlama servisi tasarımının detayları ele alınmaktadır.

8.1 Oran Sınırlama Servisine Alternatifler

Oran sınırlama servisi yerine kullanılabilecek alternatif çözümler, sistemin performansını ve güvenliğini artırabilir. Alternatifler arasında, sistemdeki kaynakların otomatik olarak ölçeklendirilmesi, daha gelişmiş trafik şekillendirme (traffic shaping) yöntemleri veya önbellekleme teknikleri bulunmaktadır. Bu bölümde, oran sınırlamaya alternatif olabilecek bu çözümler ve kullanım alanları incelenir.

8.2 Oran Sınırlama Ne Zaman Yapılmaz?

Oran sınırlamanın her zaman gerekli olmadığı senaryolar da vardır. Örneğin, düşük trafikli sistemlerde oran sınırlama kullanmak gereksiz olabilir veya kritik kullanıcı işlemleri sırasında oran sınırlama devre dışı bırakılabilir. Bu bölümde, oran sınırlamanın kullanılmaması gereken durumlar ve bu gibi senaryoların nasıl yönetileceği açıklanmaktadır.

8.3 İşlevsel Gereksinimler

Oran sınırlama servisinin temel işlevsel gereksinimleri belirlenir. Servisin, belirli zaman dilimlerinde gelen istekleri sayması, belirlenen limitlere göre bu istekleri engellemesi ve kullanıcıya uygun yanıtlar vermesi gerekir. Ayrıca, limit aşımında hangi tür geri bildirimlerin (örneğin, 429 Too Many Requests HTTP yanıtı) verileceği kararlaştırılmalıdır.

8.4 İşlevsel Olmayan Gereksinimler

Oran sınırlama servisinin performans, kullanılabilirlik, güvenilirlik ve ölçeklenebilirlik gibi işlevsel olmayan gereksinimleri de önemlidir. Servisin, sistemin geri kalanıyla uyumlu, esnek ve yüksek yük altında da çalışabilir olması gerekmektedir. Bu bölümde, işlevsel olmayan gereksinimlerin nasıl karşılanacağı ve hangi metriklerin izleneceği tartışılır.

8.5 Kullanıcı Hikayeleri ve Gerekli Servis Bileşenleri

Kullanıcı hikayeleri, oran sınırlama servisi tasarımında yol gösterici bir unsurdur. Farklı kullanıcıların nasıl etkileşimde bulunacağı ve servisin nasıl yanıt vereceği bu hikayelerle belirlenir. Örneğin, bir kullanıcı belirli bir zaman dilimi içinde çok fazla istek gönderdiğinde ne olacağı ve bu isteklerin nasıl yönetileceği açıklanır. Ayrıca, bu işlevsellikleri sağlamak için gerekli olan servis bileşenleri tanımlanır.

8.6 Yüksek Seviyeli Mimari

Oran sınırlama servisinin yüksek seviyeli mimarisi, servisin temel bileşenlerini ve bu bileşenler arasındaki etkileşimi tanımlar. Bu bölümde, oran sınırlama servisinin genel yapısı, istemci ve sunucu arasındaki iletişim, merkezi ya da dağıtık veri saklama yöntemleri ve oran sınırlama kurallarının uygulanması gibi konular ele alınır.

8.7 Durum Bilgili Yaklaşım/Bölümlendirme

Durum bilgili bir oran sınırlama servisi, kullanıcının önceki isteklerine dayanarak kararlar alır. Bu, kullanıcının gönderdiği her isteğin izlenmesi ve oran sınırlamanın buna göre uygulanması anlamına gelir. Bu bölümde, durum bilgili sistemlerin nasıl çalıştığı ve duruma dayalı oran sınırlamanın nasıl uygulanabileceği açıklanır. Ayrıca, bu yaklaşımın veri bölümlendirme (sharding) stratejileriyle nasıl entegre edileceği üzerinde durulur.

8.8 Her Hosta Tüm Sayaçları Kaydetme

Oran sınırlama sırasında, her bir istemciye ait isteklerin sayaçları her host üzerinde tutulabilir. Bu bölümde, bu yaklaşımın avantajları ve zorlukları tartışılır. Merkezi veya dağıtık sayaç yönetimi ile oran sınırlama verilerinin doğru bir şekilde nasıl yönetileceği ele alınır. Ayrıca, farklı sunucular arasında sayaçların nasıl senkronize edileceği anlatılır.

8.9 Oran Sınırlama Algoritmaları

Oran sınırlama servisini tasarlarken kullanılabilecek birkaç yaygın algoritma bulunmaktadır. Bu bölümde, **Token Bucket**, **Leaky Bucket**, **Fixed Window** ve **Sliding Window** gibi oran sınırlama algoritmaları ele alınır. Her algoritmanın nasıl çalıştığı, avantajları ve dezavantajları açıklanır. Ayrıca, belirli kullanım senaryoları için hangi algoritmanın en uygun olduğuna dair rehberlik sunulur.

8.10 Yan Arabirim Modeli Kullanma

Oran sınırlama servisi, mikroservis mimarisinde bir yan arabirim (sidecar) modeliyle uygulanabilir. Bu model, oran sınırlama işlemini servislerin yanına yerleştirilen bir bileşen aracılığıyla gerçekleştirir. Yan arabirim modeli, servisin diğer işlevlerinden ayrılarak bağımsız bir oran sınırlama modülü oluşturulmasını sağlar. Bu bölümde, yan arabirim modelinin nasıl çalıştığı ve hangi senaryolarda tercih edilmesi gerektiği anlatılmaktadır.

8.11 Loglama, İzleme ve Uyarı Verme

Oran sınırlama servisinin izlenmesi ve yönetilmesi için loglama ve izleme mekanizmaları kritik öneme sahiptir. Sistem üzerindeki oran sınırlama kararlarının kayıt altına alınması ve potansiyel sorunların tespit edilmesi, sistemin genel sağlığını izlemek için önemlidir. Bu bölümde, loglama,

izleme ve uyarı verme stratejileri ele alınır. Ayrıca, oran sınırlama servisinin nasıl optimize edileceği ve olası sorunlara nasıl hızlı bir şekilde müdahale edileceği anlatılır.
Bu bölüm, oran sınırlama servisinin nasıl tasarlanacağını ve bu servisin yüksek performanslı bir sistemde nasıl entegre edileceğini detaylandırır. Oran sınırlama, kaynakların korunması ve hizmet kalitesinin sürdürülmesi açısından kritik bir role sahiptir.

Bölüm 9: Bildirim/Uyarı Servisi Tasarımı
Bu bölümde, bildirim ve uyarı servislerinin tasarımı ele alınır. Bildirim servisleri, kullanıcıları belirli olaylar hakkında bilgilendirmek için kritik öneme sahiptir. Bu servisin tasarımı, hem işlevsel hem de işlevsel olmayan gereksinimlere uygun olarak yapılandırılmalıdır.
9.1 İşlevsel Gereksinimler
Bildirim/uayrı servisi, kullanıcılara çeşitli kanallar aracılığıyla (e-posta, SMS, push bildirimleri, vb.) mesajlar gönderebilmelidir. Sistem, bildirimlerin kimlere ve hangi koşullarda gönderileceğini belirlemelidir. Ayrıca, bildirimlerin gönderilmesi gerektiğinde doğru zamanlamayla çalışması ve kullanıcının bildirim tercihlerini dikkate alması gerekmektedir.
9.2 İşlevsel Olmayan Gereksinimler
Servisin performansı, güvenilirliği ve ölçeklenebilirliği, işlevsel olmayan gereksinimlerin başında gelir. Yüksek hacimli bildirim taleplerine yanıt verebilmesi, kullanıcı deneyimini olumsuz etkilememesi ve güvenli bir şekilde çalışması önemlidir. Ayrıca, bildirimlerin kesintisiz bir şekilde iletilmesi ve sistem hatalarına karşı dayanıklı olması gerekmektedir.
9.3 İlk Yüksek Seviyeli Mimari
Bildirim/uayrı servisi tasarımının temel mimarisi belirlenir. Bu, bildirim gönderim süreçlerini yöneten bir ana bileşen, çeşitli mesajlaşma kanallarını yöneten bileşenler ve geri bildirimleri işleyen bir yapı içerir. Bu mimarinin esnek ve modüler olması, farklı kanalların entegre edilebilmesini sağlar. Merkezi bir bildirim yönetim sistemi ile çeşitli kullanıcı gruplarına hitap eden esnek bir yapı önerilir.
9.4 Nesne Deposu: Bildirim Yapılandırma ve Gönderme
Bildirim yapılandırmaları, kullanıcının aldığı bildirimlerin içeriğini, biçimini ve zamanlamasını belirler. Nesne deposu, bu yapılandırmaların saklanmasını ve gerektiğinde hızlıca erişilmesini sağlar. Kullanıcıların hangi bildirimleri almak istediklerini belirtmeleri ve sistemin buna göre hareket etmesi bu yapı ile sağlanır.
9.5 Bildirim Şablonları
Bildirim şablonları, belirli olaylar için önceden tanımlanmış mesajlar içerir. Farklı olaylar için özelleştirilmiş şablonlar oluşturulabilir ve bu şablonlar dinamik içerikle doldurulabilir. Şablonların kullanımı, sistemin esnekliğini artırır ve tutarlı mesajlar sunar. Bu bölümde, bildirim şablonlarının nasıl tasarlanacağı ve yönetileceği anlatılır.
9.6 Planlanmış Bildirimler
Bazen, bildirimlerin belirli bir zaman diliminde gönderilmesi gerekebilir. Planlanmış bildirimler, belirli bir olaydan sonra veya belirli bir tarihte kullanıcılara iletilecek bildirimleri yönetir. Bu bölümde, zamanlanmış görevlerin nasıl yapılandırılacağı ve sistemde nasıl yönetileceği ele alınır.
9.7 Bildirim Alıcı Grupları
Bildirimler genellikle belirli kullanıcı gruplarına yönelik olarak gönderilir. Alıcı grupları, kullanıcıların segmentlere ayrılması ve bu segmentlere özel bildirimlerin gönderilmesi için

kullanılır. Örneğin, sadece premium kullanıcılara veya belirli bir coğrafi bölgede bulunan kullanıcılara bildirim göndermek mümkündür. Bu bölümde, alıcı gruplarının nasıl oluşturulacağı ve yönetileceği açıklanır.

9.8 İptal İsteklerinin İşlenmesi

Kullanıcılar, belirli bildirimlerin iptal edilmesini talep edebilirler. Bu iptal istekleri, sistemde hızlıca işlenmeli ve kullanıcının bildirim tercihleri anında güncellenmelidir. Bu bölümde, iptal isteklerinin nasıl yönetileceği ve bu sürecin kullanıcı dostu hale getirilmesi için yapılması gerekenler tartışılır.

9.9 Başarısız Gönderimlerin İşlenmesi

Bazen bildirimler çeşitli sebeplerden dolayı kullanıcıya ulaşamayabilir. Başarısız gönderimlerin kaydedilmesi, izlenmesi ve gerektiğinde tekrar denenmesi önemlidir. Bu bölümde, başarısız bildirimlerin nasıl yönetileceği ve sistemin hata toleransını artırmak için hangi stratejilerin kullanılabileceği anlatılır.

9.10 Müşteri Tarafındaki Tekrarlı Bildirimlerle İlgili Dikkat Edilmesi Gerekenler

Kullanıcılar, aynı olay için birden fazla bildirim alabilir ve bu tekrarlı bildirimler rahatsızlık yaratabilir. Tekrarlı bildirimlerin önlenmesi için kullanıcı tarafında kontrol mekanizmaları uygulanmalıdır. Bu bölümde, tekrarlanan bildirimlerin nasıl engelleneceği ve kullanıcı deneyiminin nasıl iyileştirileceği üzerine çözüm önerileri sunulur.

9.11 Öncelik

Bildirimlerin önceliklendirilmesi, hangi bildirimlerin önce gönderileceğini ve hangi bildirimlerin daha kritik olduğunu belirler. Acil bildirimlerin hızlıca işlenmesi ve düşük öncelikli bildirimlerin uygun zamanlarda iletilmesi gerekir. Bu bölümde, bildirimlerin nasıl önceliklendirileceği ve hangi kriterlere göre gönderileceği açıklanmaktadır.

9.12 Arama

Bildirimlerin geçmişi, kullanıcıların ne tür bildirimler aldığının izlenmesi ve yönetilmesi için arşivlenir. Bildirimlerin arşivlenmesi ve bu arşivde arama yapma yeteneği, sistem yöneticilerinin geçmiş bildirimleri incelemesine ve analiz yapmasına olanak tanır. Bu bölümde, bildirimlerin nasıl arşivleneceği ve arama işlevinin nasıl tasarlanacağı üzerinde durulur.

9.13 İzleme ve Uyarı Verme

Bildirim servisinin performansı ve durumu sürekli izlenmeli ve gerektiğinde uyarılar verilmelidir. Özellikle başarısız bildirimler, gecikmeler veya sistem hataları gibi sorunların zamanında tespit edilmesi için izleme ve uyarı mekanizmaları kritik öneme sahiptir. Bu bölümde, hangi metriklerin izleneceği ve sorunların nasıl tespit edileceği ele alınır.

Bu bölümde, bildirim ve uyarı servislerinin tasarımına yönelik detaylı bir yaklaşım sunulmaktadır. Bildirimlerin nasıl oluşturulacağı, yönetileceği ve kullanıcıya iletileceği gibi konular detaylandırılmıştır.

Bölüm 10: Veritabanı Toplu Denetim Servisi Tasarımı

Bu bölümde, veritabanı toplu denetim servisi (bulk auditing) tasarımı ele alınır. Büyük veritabanları genellikle denetim süreçlerine ihtiyaç duyar, özellikle veri güvenliği, bütünlüğü ve düzenliliği sağlamak için. Bu süreç, veritabanındaki verilerin doğruluğunu ve sistemin düzgün çalışıp çalışmadığını izlemek için kullanılır.

10.1 Neden Denetim Gereklidir?

Veritabanı denetimi, sistemin zaman içinde nasıl değiştiğini anlamak ve olası hataları veya yanlışlıkları tespit etmek için önemlidir. Özellikle büyük sistemlerde, veri tutarlılığı sağlamak ve güvenlik ihlallerini tespit etmek için denetim hayati bir rol oynar. Bu bölümde, veritabanı denetiminin neden gerekli olduğu ve hangi durumlarda zorunlu hale geldiği açıklanır.

10.2 SQL Sorgusunun Sonucuna Koşullu İfadeyle Doğrulama Tanımlama

Denetim sürecinde SQL sorgularının sonucuna göre doğrulama yapılması gerekebilir. Örneğin, belirli bir sorgu sonucu beklenmedik bir veri döndürüyorsa bu durumda bir uyarı verilmesi veya bir işlem yapılması gerekir. Koşullu ifadeler kullanılarak, sonuçlara göre denetim süreçleri daha dinamik hale getirilebilir.

10.3 Basit SQL Toplu Denetim Servisi

Toplu denetim servisi, bir veritabanındaki tüm verilerin veya seçilen bir alt kümenin denetlenmesi işlemidir. Bu süreçte, verilerin önceden belirlenmiş kurallara göre kontrol edilmesi ve bu kuralların ihlal edilip edilmediğinin tespit edilmesi sağlanır. Bu bölümde, basit bir SQL toplu denetim servisinin nasıl çalışacağı ve hangi yöntemlerle tasarlanabileceği anlatılır.

10.4 Gereksinimler

Veritabanı toplu denetim servisi tasarlanırken, işlevsel ve işlevsel olmayan gereksinimler belirlenir. İşlevsel gereksinimler arasında hangi verilerin denetleneceği, nasıl raporlanacağı ve hangi işlemlerin gerçekleştirileceği yer alır. İşlevsel olmayan gereksinimler ise performans, ölçeklenebilirlik ve hata toleransı gibi konuları kapsar.

10.5 Yüksek Seviyeli Mimari

Denetim servisi, veritabanına sürekli erişim sağlayan ve verilerin düzenli olarak kontrol edilmesini sağlayan bir yapı olmalıdır. Bu yüksek seviyeli mimari, verilerin toplanması, denetim kurallarının uygulanması ve sonuçların raporlanması adımlarını içerir. Bu bölümde, toplu denetim servisi için önerilen mimari yapının nasıl tasarlanacağı anlatılır.

10.6 Veritabanı Sorguları Üzerindeki Kısıtlamalar

Toplu denetim sırasında, veritabanı sorgularının boyutu ve karmaşıklığı nedeniyle performans sorunları ortaya çıkabilir. Bu bölümde, veritabanı sorgularının nasıl optimize edileceği ve performansın nasıl iyileştirileceği üzerine stratejiler sunulur. Sorgu optimizasyonu ve gereksiz veri yükünden kaçınma yöntemleri açıklanır.

10.7 Aynı Anda Çok Fazla Sorgu Çalıştırmayı Önleme

Toplu denetim işlemleri sırasında, aynı anda çok fazla sorgu çalıştırmak veritabanı üzerinde ciddi bir yük oluşturabilir ve performans düşüşüne neden olabilir. Bu bölümde, aynı anda çalıştırılan sorgu sayısının nasıl kontrol edileceği ve sistemin aşırı yüklenmesini önlemek için hangi stratejilerin kullanılacağı anlatılır.

Bu bölümde, veritabanı denetim sürecinin nasıl tasarlanacağı ve yönetileceği ele alınmıştır. Denetim servisi, büyük sistemlerde veri güvenliği ve bütünlüğünü sağlamak için kritik öneme sahiptir ve bu servisin tasarımı, performans ve verimlilik göz önünde bulundurularak yapılmalıdır.

Bölüm 11: Otomatik Tamamlama/Typeahead Tasarımı

Bu bölümde, kullanıcı deneyimini geliştiren bir özellik olan otomatik tamamlama (autocomplete) veya typeahead tasarımı incelenir. Bu işlev, kullanıcının bir metin kutusuna veri girişi yaparken

sistemin kullanıcının niyetini tahmin ederek öneriler sunmasını sağlar. Otomatik tamamlama, arama çubuğu, form doldurma ve çeşitli kullanıcı etkileşimlerinde kullanılır.

11.1 Otomatik Tamamlama Kullanımları

Otomatik tamamlama, özellikle arama motorları ve form doldurma işlemlerinde yaygın olarak kullanılır. Kullanıcılar, yalnızca birkaç karakter girerek ilgili sonuçları hızlı bir şekilde görme olanağına sahip olur. Bu bölümde, otomatik tamamlama işlevinin farklı kullanım alanları ve kullanıcı deneyimi üzerindeki etkisi açıklanır.

11.2 Arama vs. Otomatik Tamamlama

Otomatik tamamlama ve arama işlevleri genellikle birlikte çalışır, ancak farklı kullanım alanlarına sahiptir. Arama, kullanıcının tam sorgusunu işlerken, otomatik tamamlama, kullanıcı sorgusunu tamamlamadan önce öneriler sunar. Bu bölümde, bu iki işlev arasındaki farklar ve otomatik tamamlama işlevinin arama sistemleriyle nasıl entegre edilebileceği anlatılır.

11.3 İşlevsel Gereksinimler

Otomatik tamamlama işlevi, kullanıcının girdiği verileri hızlı ve doğru bir şekilde tahmin ederek önerilerde bulunmalıdır. Ayrıca, önerilerin uygun ve güncel olması gerekir. Bu bölümde, otomatik tamamlama için temel işlevsel gereksinimler belirlenir. Bu gereksinimler arasında hız, doğru öneriler, önerilerin sıralanması ve kullanıcı girdisine göre özelleştirilebilme yer alır.

11.4 İşlevsel Olmayan Gereksinimler

İşlevsel olmayan gereksinimler, sistemin performansı, ölçeklenebilirliği ve yanıt süreleri ile ilgilidir. Kullanıcı girdilerine anında geri bildirim sağlanması kritik öneme sahiptir. Özellikle büyük veri kümelerinde hızlı yanıt verebilmek için sistemin optimize edilmesi gerekir. Bu bölümde, otomatik tamamlama sisteminin performans, ölçeklenebilirlik ve hata toleransı gibi işlevsel olmayan gereksinimleri ele alınır.

11.5 Yüksek Seviyeli Mimari Planlaması

Otomatik tamamlama işlevi için yüksek seviyeli mimari, kullanıcı isteklerinin nasıl işleneceğini ve önerilerin nasıl sunulacağını belirler. Veriler genellikle bir dizinleme sistemi veya önbellek kullanılarak yönetilir. Bu bölümde, otomatik tamamlama işlevi için önerilen mimari yapı açıklanır ve veri depolama, sorgulama ve öneri süreçlerinin nasıl yönetileceği anlatılır.

11.6 Ağırlıklı Trie Yaklaşımı ve İlk Yüksek Seviyeli Mimari

Otomatik tamamlama sistemlerinde sıklıkla kullanılan veri yapılarından biri Trie (ön ek ağacı) veri yapısıdır. Ağırlıklı Trie, her düğümde olası sonuçların popülerliğine göre sıralama yaparak kullanıcıya en uygun sonuçları sunar. Bu bölümde, ağırlıklı Trie veri yapısının otomatik tamamlama sistemlerinde nasıl kullanılacağı ve yüksek seviyeli mimari ile nasıl entegre edileceği anlatılır.

11.7 Detaylı Uygulama

Otomatik tamamlama işlevinin detaylı uygulaması, kullanıcı girdilerine hızlı yanıt verebilmek için veri yapılarının nasıl optimize edileceğini ele alır. Verilerin nasıl indeksleneceği, önbellekleme stratejileri ve kullanıcı girdileriyle öneriler arasındaki ilişki bu bölümde detaylandırılır. Ayrıca, algoritmaların nasıl uygulanacağı ve veri sorgulama süreçleri hakkında bilgiler verilir.

11.8 Örnekleme Yaklaşımı

Veri kümesinin çok büyük olduğu durumlarda, öneriler sunarken örnekleme yaklaşımı kullanılarak performans artırılabilir. Bu yöntemle, kullanıcının tüm veri kümesi üzerinde işlem yapması yerine belirli bir örneklem üzerinden tahminler yürütülür. Bu bölümde, örnekleme yaklaşımının nasıl uygulanacağı ve hangi durumlarda tercih edileceği açıklanır.

11.9 Depolama Gereksinimlerinin İşlenmesi

Otomatik tamamlama işlevi için kullanılan verilerin depolanması, veri kümesinin büyüklüğüne ve sistemin performans gereksinimlerine bağlıdır. Depolama gereksinimlerinin yönetimi, büyük veri kümeleriyle çalışırken kritik öneme sahiptir. Bu bölümde, verilerin nasıl saklanacağı, hangi veri yapılarının kullanılacağı ve veritabanı seçimi hakkında bilgi verilir.

Bu bölüm, otomatik tamamlama işlevinin nasıl tasarlanacağı ve geliştirileceği hakkında kapsamlı bilgi sunar. Kullanıcı deneyimini geliştiren bu özellik, özellikle büyük veri kümeleriyle çalışırken dikkatlice tasarlanmalıdır. Otomatik tamamlama sistemi, hız, doğruluk ve ölçeklenebilirlik açısından optimize edilmelidir.

Bölüm 12: Flickr Tasarımı

Bu bölümde, bir fotoğraf paylaşım ve saklama platformu olan Flickr'ın sistem tasarımı incelenmektedir. Flickr, büyük bir kullanıcı kitlesi tarafından kullanılan, yüksek hacimli veri (fotoğraf, video) barındıran ve hızlı erişim sunan bir sistemdir. Bu bölümde, böyle bir platformun işlevsel ve teknik gereksinimleri, veri yönetimi stratejileri ve yüksek seviyeli mimarisi ele alınmaktadır.

12.1 Kullanıcı Hikayeleri ve İşlevsel Gereksinimler

Flickr gibi bir sistemin temel işlevsel gereksinimleri, kullanıcıların fotoğraf ve video yüklemesine, bunları paylaşmasına ve aramasına olanak tanımaktır. Kullanıcılar ayrıca fotoğraflarını albümler halinde düzenleyebilmelidir. Bu bölümde, Flickr'ın işlevsel gereksinimleri kullanıcı hikayeleri üzerinden açıklanmaktadır. Örneğin, bir kullanıcı yeni bir fotoğraf yükler, bir başka kullanıcı bu fotoğrafı arar ve albümünde görüntüler.

12.2 İşlevsel Olmayan Gereksinimler

Flickr'ın işlevsel olmayan gereksinimleri, özellikle performans, ölçeklenebilirlik, hata toleransı ve kullanılabilirlik gibi konuları içerir. Kullanıcıların fotoğraflarına hızlı erişim sağlayabilmek için sistemin yüksek performans göstermesi gerekir. Ayrıca, büyük veri hacimlerini yönetebilmesi için depolama çözümleri, fotoğraf ve video verilerinin güvenli bir şekilde saklanması ve yedeklenmesi gibi konular ele alınır.

12.3 Yüksek Seviyeli Mimari

Flickr'ın yüksek seviyeli mimarisi, kullanıcıların yüklediği büyük boyutlu medya dosyalarını hızlı ve verimli bir şekilde yönetmeyi amaçlar. Dağıtık bir sistem kullanarak veri depolama, işleme ve kullanıcı etkileşimlerini yönetmek için çeşitli servisler devreye girer. Bu bölümde, sistemin temel bileşenleri ve bileşenler arasındaki etkileşimler detaylandırılır.

12.4 SQL Şeması

Flickr, hem ilişkisel hem de NoSQL veritabanlarını kullanarak hem kullanıcı bilgilerini hem de medya içeriklerini yönetir. Bu bölümde, Flickr için önerilen SQL veritabanı şeması üzerinde durulmaktadır. Fotoğraflar, kullanıcılar, albümler ve etiketler arasındaki ilişkiler veri modellemesi ile gösterilir. Veritabanı şeması, kullanıcıların arama, filtreleme ve fotoğraf görüntüleme işlemlerini optimize edecek şekilde tasarlanmalıdır.

12.5 CDN Üzerinde Dizin ve Dosyaların Düzenlenmesi

Flickr gibi bir platformda, fotoğraf ve video gibi büyük medya dosyalarının hızlı bir şekilde kullanıcılara sunulması için içerik dağıtım ağı (CDN) kullanılır. Bu bölümde, CDN'nin nasıl entegre edileceği ve medya dosyalarının dizinleme ve düzenleme süreçlerinin nasıl yönetileceği

anlatılmaktadır. Dosyaların coğrafi olarak dağıtılması, kullanıcı deneyimini artırmak için önemlidir.

12.6 Fotoğraf Yükleme

Kullanıcıların fotoğraflarını yükleme süreci, verimli ve hızlı bir şekilde yapılmalıdır. Bu bölümde, fotoğraf yükleme işleminin nasıl optimize edileceği, arka planda işlenecek süreçler ve yükleme sırasında hangi tekniklerin kullanılacağı anlatılır. Örneğin, fotoğraflar yüklenirken arka planda sıkıştırma, boyutlandırma ve meta veri işlemleri yapılabilir.

12.7 Görselleri ve Verileri İndirme

Flickr'daki fotoğraf ve video içeriklerinin indirilmesi, kullanıcıların yüksek kaliteli görsellere hızlı erişimini sağlamalıdır. Bu bölümde, indirme işlemlerinin nasıl optimize edileceği ve kullanıcıların büyük medya dosyalarını etkili bir şekilde nasıl indirebileceği açıklanır. CDN entegrasyonu ve önbellekleme gibi yöntemler, indirme işlemlerini hızlandırmada yardımcı olabilir.

12.8 İzleme ve Uyarı Verme

Flickr gibi büyük bir sistemin stabil çalışması için izleme ve uyarı mekanizmaları kritik öneme sahiptir. Bu bölümde, sistemin nasıl izleneceği, hangi metriklerin takip edileceği ve olası sorunların nasıl tespit edilip çözüleceği tartışılmaktadır. Ayrıca, performans düşüşleri veya hata oranlarındaki artış gibi durumlarda otomatik uyarı sistemlerinin devreye girmesi gerekmektedir. Flickr tasarımı, büyük miktarda medya içeriğinin depolanması, yönetilmesi ve hızlı bir şekilde erişilmesi gereken sistemler için örnek bir yaklaşımdır. Bu bölümde, Flickr'ın mimarisi, veri yönetimi stratejileri ve performans gereksinimleri detaylı bir şekilde ele alınmıştır.

Bölüm 13: İçerik Dağıtım Ağı (CDN) Tasarımı

Bu bölümde, bir İçerik Dağıtım Ağı'nın (CDN) nasıl tasarlanacağı ve kullanılacağı ele alınmaktadır. CDN, özellikle büyük ölçekli sistemlerde, medya dosyalarının ve diğer statik içeriklerin kullanıcıya hızlı ve verimli bir şekilde sunulmasını sağlar. Kullanıcı deneyimini geliştirmek ve sistem üzerindeki yükü hafifletmek için CDN çözümleri kritik bir role sahiptir.

13.1 CDN'in Avantajları ve Dezavantajları

CDN, içeriği kullanıcılara coğrafi olarak en yakın sunucudan sunarak gecikme süresini azaltır ve performansı artırır. CDN kullanmanın en önemli avantajları arasında düşük gecikme süreleri, bant genişliği tasarrufu ve sistem yükünün dengelenmesi yer alır. Ancak, CDN'nin maliyeti ve bazı durumlarda içerik senkronizasyonu gibi dezavantajları da bulunmaktadır. Bu bölümde, CDN'nin avantajları ve dezavantajları detaylı bir şekilde incelenir.

13.2 Gereksinimler

CDN tasarımı yapılırken, sistemin hangi içerikleri dağıtacağı ve bu içeriklerin nasıl yönetileceği belirlenmelidir. İçerik türüne göre, statik dosyalar (örneğin resimler, CSS, JavaScript) ve dinamik içerikler için farklı çözümler geliştirilebilir. Bu bölümde, CDN kullanımının temel işlevsel gereksinimleri ele alınır ve hangi içeriklerin CDN üzerinden dağıtılacağı planlanır.

13.3 CDN Kimlik Doğrulama ve Yetkilendirme

Bazı içerikler, yalnızca yetkili kullanıcılar tarafından erişilebilecek şekilde yapılandırılmalıdır. Bu nedenle, CDN'in kimlik doğrulama ve yetkilendirme süreçleri de tasarlanmalıdır. Bu bölümde, CDN üzerindeki içeriklere erişimin nasıl kontrol edileceği ve hangi yetkilendirme mekanizmalarının kullanılacağı tartışılır. Örneğin, erişim belirteçleri (token-based access) ve oturum yönetimi kullanılarak yetkili kullanıcıların içeriklere erişimi sağlanabilir.

13.4 Yüksek Seviyeli Mimari
CDN'in yüksek seviyeli mimarisi, kullanıcı isteklerinin en yakın sunucuya yönlendirilmesi, içeriklerin uygun şekilde önbelleğe alınması ve güncellenmesi süreçlerini içerir. Bu bölümde, CDN tasarımında kullanılan ana bileşenler ve bu bileşenler arasındaki etkileşimler açıklanır. CDN mimarisinin ölçeklenebilir, hızlı ve güvenilir olması kritik öneme sahiptir.
13.5 Depolama Servisi
CDN, büyük miktarda statik içeriği barındırmak zorundadır. Depolama servisleri, bu içeriklerin güvenli bir şekilde saklanması ve yönetilmesi için kullanılır. Bu bölümde, depolama servislerinin nasıl yapılandırılacağı ve hangi veri yönetim stratejilerinin kullanılacağı anlatılır. Ayrıca, CDN ile depolama servislerinin entegrasyonu ele alınır.
13.6 Yaygın İşlemler
CDN tasarımı, içerik yükleme, önbelleğe alma, önbellek geçersiz kılma ve içerik güncelleme gibi işlemler için optimize edilmelidir. Bu bölümde, CDN üzerinde gerçekleştirilen yaygın işlemler detaylandırılır. Önbellek geçerlilik süreleri, içerik güncellemeleri ve kullanıcı isteklerine hızlı yanıt verme gibi konular üzerinde durulmaktadır.
13.7 Önbellek Geçersiz Kılma
CDN, statik içerikleri uzun süre önbellekte tutabilir, ancak içerikler güncellendiğinde eski verilerin geçersiz kılınması ve yeni içeriklerin sunulması gerekir. Önbellek geçersiz kılma, CDN sistemlerinde önemli bir süreçtir. Bu bölümde, içeriklerin nasıl önbellekten kaldırılacağı ve güncel içeriklerin nasıl sunulacağı açıklanır. Ayrıca, önbellek geçersiz kılma stratejileri üzerinde durulur.
13.8 İzleme ve Uyarı Verme
CDN'nin performansı sürekli olarak izlenmeli ve olası sorunlara karşı uyarılar oluşturulmalıdır. Bu bölümde, CDN'nin nasıl izleneceği ve hangi metriklerin takip edilmesi gerektiği açıklanır. Ayrıca, CDN performansını optimize etmek ve olası hataları erken tespit etmek için kullanılacak uyarı mekanizmaları tartışılır.
Bu bölüm, bir CDN'in nasıl tasarlanacağı, yönetileceği ve optimize edileceği konusunda rehberlik sunar. CDN'ler, yüksek performanslı ve ölçeklenebilir sistemler oluşturmak için kritik bir rol oynar ve doğru tasarlandığında kullanıcı deneyimini büyük ölçüde iyileştirir.

Bölüm 14: Mesajlaşma Uygulaması Tasarımı
Bu bölümde, bir mesajlaşma uygulamasının sistem tasarımı ele alınmaktadır. Mesajlaşma uygulamaları, gerçek zamanlı iletişim sağladıkları için yüksek performans, düşük gecikme ve güvenilirlik gibi gereksinimlere sahiptir. Bu tür uygulamaların tasarımı, hem kullanıcı deneyimi hem de sistem verimliliği açısından kritik öneme sahiptir.
14.1 Gereksinimler
Bir mesajlaşma uygulaması tasarlanırken, temel işlevsel gereksinimler şunlardır: kullanıcılar arasında mesajların hızlı ve güvenilir bir şekilde iletilmesi, grup sohbetleri, mesajların senkronize edilmesi, mesaj geçmişi ve medya dosyalarının gönderilmesi. Bu bölümde, mesajlaşma uygulamasının işlevsel ve işlevsel olmayan gereksinimleri detaylandırılmaktadır. Performans, ölçeklenebilirlik ve güvenlik ön plandadır.
14.2 İlk Düşünceler

Bir mesajlaşma uygulaması tasarlarken, düşük gecikme süreleri sağlamak, mesajları güvenilir bir şekilde iletmek ve mesajların sistemde doğru bir sırayla ulaşmasını garanti etmek gerekir. Bu bölümde, mesajların nasıl işleneceği ve uygulamanın mimarisi hakkında başlangıç fikirleri tartışılmaktadır. Uygulamanın ölçeklenebilirliği ve kullanıcı sayısına göre nasıl büyüyebileceği ele alınır.

14.3 Yüksek Seviyeli Tasarım

Mesajlaşma uygulamasının yüksek seviyeli tasarımı, istemci ve sunucu arasındaki iletişim modeline dayanır. Kullanıcı istekleri, sunucuya iletilir ve sunucu mesajları diğer kullanıcılara yönlendirir. Bu bölümde, merkezi bir mesaj yönlendirme sistemi tasarımı ve istemci-sunucu etkileşimi incelenir. Ayrıca, farklı iletişim protokolleri (örneğin, HTTP, WebSockets) tartışılır.

14.4 Bağlantı Servisi

Mesajlaşma uygulamalarında sürekli bağlantının sağlanması gereklidir. Kullanıcılar uygulamada çevrimiçi olduğu sürece bağlantı açık olmalı ve sunucu ile sürekli iletişimde olmalıdır. Bu bölümde, kullanıcı bağlantılarını yönetmek için gereken altyapı ve teknikler (örneğin, uzun süreli bağlantılar, WebSocket) açıklanır. Ayrıca, bağlantı kopması durumunda yeniden bağlanma stratejileri ele alınır.

14.5 Mesaj Gönderme Servisi

Mesaj gönderme servisi, kullanıcıların mesajlarını alıcılara iletmekle sorumludur. Bu bölümde, mesajların gönderilmesi, mesajların sıraya alınması ve mesajın alıcılara ulaştığından emin olunması gibi süreçler ele alınır. Mesaj teslimi garantisi sağlamak için kullanılan teknikler (örneğin, kuyruklama ve mesajın başarılı bir şekilde teslim edildiğine dair geri bildirim) tartışılır.

14.6 Mesaj Servisi

Mesaj servisi, mesajların sunucu tarafında işlenmesini ve depolanmasını sağlar. Bu bölümde, mesajların nasıl depolanacağı, nasıl sıralanacağı ve nasıl kullanılabilir hale getirileceği üzerinde durulur. Mesaj geçmişinin yönetimi, grup mesajlarının işlenmesi ve medya içeriklerinin depolanması gibi konular da ele alınır.

14.7 Mesaj Gönderim Adımları

Mesajların gönderim adımları, istemci ve sunucu arasındaki işlemleri kapsar. Mesajın gönderilmesi, kuyrukta işlenmesi, alıcıya iletilmesi ve alıcının mesajı aldığına dair onayının geri gönderilmesi süreci bu bölümde anlatılır. Mesajın teslim edilmesi ve iletilen mesajların başarılı bir şekilde alındığından emin olunması için gereken adımlar detaylandırılır.

14.8 Arama

Mesajlaşma uygulamalarında, kullanıcıların mesaj geçmişinde arama yapabilmesi önemli bir özelliktir. Bu bölümde, mesajların nasıl aranacağı, hangi veri yapılarının kullanılacağı ve arama sonuçlarının nasıl sunulacağı anlatılır. Ayrıca, performansı artırmak için kullanılabilecek arama indeksleme teknikleri tartışılır.

14.9 İzleme ve Uyarı Verme

Mesajlaşma uygulamalarında, sistemin sağlığını izlemek ve olası sorunlara karşı uyarılar vermek önemlidir. Bu bölümde, uygulamanın performansını ve mesaj teslimat başarısını izlemek için kullanılacak metrikler ve izleme araçları tanıtılır. Olası gecikmeler, başarısız mesaj gönderimleri veya bağlantı sorunları gibi durumlara karşı uyarı mekanizmalarının nasıl devreye alınacağı açıklanır.

Mesajlaşma uygulamalarının tasarımında düşük gecikme süresi, yüksek performans, ölçeklenebilirlik ve güvenilirlik kritik öneme sahiptir. Bu bölümde, mesaj gönderimi ve

alınmasıyla ilgili tüm süreçler, bağlantı yönetimi ve mesaj işleme adımları detaylı bir şekilde ele alınmıştır.

Bölüm 15: Airbnb Tasarımı

Bu bölümde, bir çevrimiçi kiralama platformu olan Airbnb'nin sistem tasarımı incelenmektedir. Airbnb, kullanıcıların konaklama hizmeti sunabilmesini ve bu hizmetlerden yararlanabilmesini sağlayan, geniş çaplı veri yönetimi, kullanıcı etkileşimi ve işlem süreçlerini kapsayan bir platformdur. Sistem tasarımı, yüksek kullanılabilirlik, güvenlik ve ölçeklenebilirlik gibi önemli gereksinimlere sahiptir.

15.1 Gereksinimler

Airbnb tasarımında, kullanıcıların konaklama yerleri listelemesi, bu yerler arasında arama yapabilmesi, rezervasyon işlemleri gerçekleştirebilmesi ve ödemeleri yapabilmesi gibi temel işlevsel gereksinimler belirlenmelidir. Bu bölümde, Airbnb'nin işlevsel gereksinimleri, kullanıcıların gerçekleştirdiği işlemler ve sistemin desteklemesi gereken temel özellikler tartışılmaktadır. Ayrıca, işlevsel olmayan gereksinimler (performans, güvenlik, ölçeklenebilirlik) de ele alınır.

15.2 Tasarım Kararları

Airbnb gibi bir platformun tasarımında, kullanıcı deneyimi, sistemin hız ve güvenliği ön planda tutulmalıdır. Bu bölümde, Airbnb sisteminin tasarımıyla ilgili alınacak kritik kararlar, veri yönetimi, kullanıcı arayüzü ve işlem yönetimi açısından tartışılmaktadır. Tasarım kararları, veritabanı seçimi, işlem işleme, kullanıcı yönetimi ve arama algoritmalarını içermektedir.

15.3 Yüksek Seviyeli Mimari

Airbnb'nin yüksek seviyeli mimarisi, kullanıcıların sorunsuz bir şekilde konaklama yeri araması, rezervasyon yapması ve ödeme işlemlerini tamamlamasına olanak tanır. Bu bölümde, sistemin temel bileşenleri ve servislerinin nasıl organize edileceği, mikroservis mimarisi, API yönetimi ve ölçeklenebilirlik konularına odaklanılır. Airbnb'nin global kullanıcı kitlesine hizmet edebilmesi için kullanılan dağıtık sistemlerin ve veri yönetim stratejilerinin nasıl çalıştığı açıklanır.

15.4 İşlevsel Bölümleme

İşlevsel bölümleme, Airbnb'nin farklı hizmetleri (konaklama listeleme, rezervasyon, ödeme, kullanıcı yönetimi) arasında işlevlerin dağıtılmasını sağlar. Bu bölümde, Airbnb'nin her bir ana işlevi için ayrı mikroservisler oluşturulması ve bu servislerin birbirleriyle nasıl etkileşim kuracağı anlatılmaktadır. Servisler arasında sağlanan esneklik, sistemin büyüdükçe ölçeklenmesini kolaylaştırır.

15.5 Liste Oluşturma veya Güncelleme

Kullanıcılar, Airbnb platformunda konaklama yerlerini listeleyebilir veya mevcut listelerini güncelleyebilirler. Bu süreçte, kullanıcılardan alınan bilgilerin doğrulanması, fotoğraf yükleme, konum belirleme ve fiyatlandırma gibi işlemler yer alır. Bu bölümde, listeleme sürecinin nasıl tasarlanacağı ve kullanıcı dostu bir arayüzün nasıl oluşturulacağı tartışılır. Veritabanında bu verilerin nasıl saklanacağı ve güncellemelerin nasıl yönetileceği de ele alınır.

15.6 Onay Servisi

Rezervasyon işlemleri sırasında, rezervasyon talebi ve onay süreçleri kritik bir rol oynar. Bu bölümde, rezervasyon taleplerinin nasıl işleneceği, onay süreçlerinin nasıl yürütüleceği ve

kullanıcıların rezervasyon durumları hakkında nasıl bilgilendirileceği anlatılmaktadır. Ayrıca, rezervasyonların iptali ve iade süreçlerinin nasıl yönetileceği tartışılır.

15.7 Rezervasyon Servisi

Rezervasyon servisi, kullanıcıların platform üzerindeki konaklama yerlerini rezerve edebilmesini sağlar. Bu servisin doğru ve güvenli çalışması, kullanıcı deneyimini doğrudan etkiler. Bu bölümde, rezervasyonların nasıl işlendiği, sistemde nasıl saklandığı ve ödeme entegrasyonları ile nasıl ilişkilendirildiği açıklanmaktadır. Rezervasyon işlemlerinin hatasız gerçekleşmesi için kullanılan stratejiler de ele alınır.

15.8 Kullanılabilirlik Servisi

Kullanılabilirlik servisi, kullanıcıların arama sonuçlarında gösterilen konaklama yerlerinin uygunluk durumunu kontrol eder. Konaklama yerinin müsaitlik durumu, tarih seçimi ve rezervasyon talepleriyle senkronize edilmelidir. Bu bölümde, kullanılabilirlik durumunun nasıl kontrol edileceği, verilerin nasıl güncelleneceği ve sistemin bu verilere göre nasıl yanıt vereceği tartışılır.

15.9 İzleme ve Uyarı Verme

Airbnb gibi geniş çaplı bir platformda, sistemin sağlığını ve performansını izlemek önemlidir. Bu bölümde, rezervasyon süreçlerinde, liste güncellemelerinde veya ödeme işlemlerinde ortaya çıkabilecek hataların nasıl izleneceği ve sistem performansının nasıl takip edileceği anlatılmaktadır. Ayrıca, sistemde oluşabilecek sorunlar için uyarı mekanizmalarının nasıl devreye gireceği ve sorunların hızlı bir şekilde çözüme kavuşturulacağı açıklanır.

Bu bölüm, Airbnb gibi büyük bir kiralama platformunun tasarımında dikkate alınması gereken önemli sistem gereksinimlerini ve mimari stratejileri ele almaktadır. Rezervasyon, kullanılabilirlik, ödeme ve listeleme süreçlerinin her biri ayrı birer mikroservis olarak ele alınarak, ölçeklenebilir ve esnek bir yapı oluşturulması sağlanır.

Bölüm 16: Haber Akışı Tasarımı

Bu bölümde, kullanıcıların ilgi alanlarına göre düzenlenen içeriklerin sunulduğu bir haber akışı (news feed) sisteminin tasarımı incelenmektedir. Haber akışı, sosyal medya platformlarında yaygın olarak kullanılan bir özelliktir ve kullanıcıların en son içeriklerle etkileşimini sağlar. Haber akışının tasarımı, kullanıcı deneyimini geliştirmek ve kişiselleştirilmiş içerik sunmak açısından kritik bir rol oynar.

16.1 Gereksinimler

Haber akışı tasarımında temel gereksinimler, kullanıcılara en güncel ve ilgili içeriği hızlı bir şekilde sunmak, içerik sıralamasını optimize etmek ve kişiselleştirilmiş öneriler sunmaktır. Bu bölümde, bir haber akışının işlevsel gereksinimleri açıklanır. Kullanıcıların beğenilerine, takip ettikleri kişilere veya ilgilendikleri konulara göre içeriklerin nasıl sıralanacağı tartışılır. Aynı zamanda performans, ölçeklenebilirlik ve güvenlik gibi işlevsel olmayan gereksinimler de ele alınır.

16.2 Yüksek Seviyeli Mimari

Haber akışı sistemi, kullanıcıların tercihleri ve etkileşimlerine göre özelleştirilmiş içerik sunar. Bu mimaride, kullanıcı verilerini işleyen ve içeriği sıralayan algoritmalar kullanılır. Bu bölümde, haber akışının yüksek seviyeli mimarisi açıklanır. Veri depolama, içerik filtreleme ve sıralama algoritmalarının nasıl çalıştığı ve istemci-sunucu arasındaki iletişim modelleri ele alınır.

16.3 Haber Akışını Önceden Hazırlama
Haber akışı, kullanıcıların talep etmesinden önce hazırlıklı olmalıdır. Önceden hazırlanmış içerik, kullanıcının akışını hızlandırmak ve sistemin performansını artırmak için gereklidir. Bu bölümde, haber akışını önceden hazırlamanın yolları ve bu yaklaşımın sistem performansı üzerindeki etkileri açıklanır. Ayrıca, önceden hazırlanmış içeriklerin nasıl güncelleneceği ve kullanıcılara hızlıca sunulacağı ele alınır.
16.4 Doğrulama ve İçerik Denetleme
Haber akışında sunulan içeriklerin doğruluğu ve güvenliği önemlidir. Bu bölümde, sahte içeriklerin ve yanıltıcı bilgilerin nasıl filtreleneceği ve doğrulanacağı anlatılır. Ayrıca, içeriklerin kullanıcı güvenliğini tehdit etmeyecek şekilde denetlenmesi için uygulanacak yöntemler ve araçlar üzerinde durulur. İçerik denetimi, otomatik ve manuel yöntemlerle yapılabilir.
16.5 Görsel ve Metin Servis Etme
Haber akışında sunulan içeriklerin hem görsel hem de metin tabanlı olması gerektiği için, bu içeriklerin nasıl hızlı ve etkili bir şekilde sunulacağı önemlidir. Bu bölümde, görsellerin ve metinlerin kullanıcıya nasıl sunulacağı, CDN kullanımı ve önbellekleme gibi performansı artırıcı teknikler tartışılır. Ayrıca, medya dosyalarının optimize edilmesi ve doğru formatta sunulması konusu ele alınır.
16.6 Diğer Olası Tartışma Noktaları
Haber akışı tasarımında başka tartışma konuları da ortaya çıkabilir. Bu bölümde, kullanıcı gizliliği, kişiselleştirme algoritmalarının şeffaflığı, içeriklerin kullanıcı etkileşimine göre sıralanması ve spam içeriklerin engellenmesi gibi konular ele alınır. Ayrıca, haber akışının nasıl test edileceği ve performans optimizasyonu yapılacağı anlatılır.
Haber akışı sistemi, kullanıcıların ilgilendiği içerikleri doğru zamanda sunmak için karmaşık algoritmalar ve veri yönetimi stratejileri gerektirir. Bu bölümde, haber akışının nasıl tasarlanacağı, kullanıcı etkileşimlerinin nasıl yönetileceği ve sistemin ölçeklenebilirliği üzerine kapsamlı bilgiler sunulmuştur.

Bölüm 17: Amazon’daki İlk 10 Ürünün Dashboard Tasarımı
Bu bölümde, Amazon gibi geniş bir e-ticaret platformunda en çok satan ürünlerin takip edilmesini sağlayan bir dashboard'un (gösterge paneli) tasarımı ele alınmaktadır. Bu tür bir dashboard, şirketin ürün performansını izlemek, en çok satan ürünleri belirlemek ve satış trendlerini analiz etmek için kullanılır. Sistemin hızlı, güvenilir ve ölçeklenebilir olması gerekmektedir.
17.1 Gereksinimler
Dashboard tasarımının temel gereksinimleri arasında, Amazon’daki en çok satan 10 ürünün gerçek zamanlı olarak gösterilmesi, kullanıcıların bu verileri filtreleyebilmesi ve bu ürünlerin performansına dair istatistiklerin sunulması yer alır. Bu bölümde, dashboard'un işlevsel gereksinimleri ve işlevsel olmayan gereksinimleri (performans, güvenilirlik, kullanılabilirlik) tanımlanır. Ayrıca, kullanıcıların veri görselleştirmesi ve arayüz ihtiyaçları da ele alınır.
17.2 İlk Düşünceler
Dashboard tasarımı yapılırken, verilerin nasıl toplanacağı, nasıl saklanacağı ve nasıl hızlı bir şekilde gösterileceği konusunda kritik kararlar alınmalıdır. Bu bölümde, verilerin gerçek zamanlı veya yakın gerçek zamanlı olarak nasıl işleneceği ve hangi veri yapılarının kullanılacağı tartışılır.

Ayrıca, sistemin nasıl ölçekleneceği ve gelecekteki ihtiyaçlara nasıl uyum sağlayacağı üzerine düşünceler paylaşılır.

17.3 Yüksek Seviyeli Mimari

Bu bölümde, en çok satan 10 ürünü gösteren bir dashboard'un yüksek seviyeli mimarisi tanımlanır. Dashboard'un gerçek zamanlı olarak veri toplama, veri işleme ve kullanıcı arayüzünde sunma aşamaları anlatılır. Mikroservis mimarisi, API kullanımı, veri depolama çözümleri (örneğin, NoSQL veritabanları) ve önbellekleme gibi bileşenler bu mimarinin bir parçasıdır. Ayrıca, verilerin hızlı ve güvenilir bir şekilde gösterilmesi için veri akış sistemleri ve önbellekleme stratejileri ele alınır.

17.4 Toplama Servisi

Dashboard için gerekli olan veriler, farklı kaynaklardan toplanmalıdır. Toplama servisi, satış verilerini Amazon'un veri tabanından çekerek en çok satan ürünleri belirler. Bu bölümde, toplama servisi için kullanılan veri kaynaklarının nasıl entegre edileceği, veri alım sıklığı ve veri doğruluğu gibi konular üzerinde durulmaktadır.

17.5 Toplu İş Hattı

Toplu iş hattı, büyük miktarda verinin toplu olarak işlenmesini sağlayan bir süreçtir. Dashboard'da gösterilen verilerin sürekli güncel kalması için toplu işleme yapılabilir. Bu bölümde, toplu iş hattının nasıl yapılandırılacağı ve verilerin hangi sırayla işleneceği açıklanır. Ayrıca, iş hattının performansını artırmak için kullanılan yöntemler (örneğin, paralel işleme) tartışılır.

17.6 Akış Hattı

Akış hattı, verilerin gerçek zamanlı olarak işlenmesi ve dashboard'a anında yansıtılması amacıyla kullanılır. Özellikle e-ticaret platformlarında, verilerin hızla güncellenmesi önemlidir. Bu bölümde, veri akış hattının nasıl tasarlanacağı, gerçek zamanlı veri işleme sistemleri (örneğin, Apache Kafka) ve bu sistemlerin nasıl entegre edileceği anlatılır.

17.7 Yaklaşım

Bu bölümde, dashboard tasarımı için kullanılan genel yaklaşım açıklanmaktadır. Verilerin nasıl işleneceği, hangi metriklerin izleneceği ve dashboard'da hangi bilgilerin sunulacağı ele alınır. Kullanıcıların performans verilerini analiz edebilmesi için veri görselleştirme teknikleri (grafikler, tablolar, metrikler) detaylandırılır. Ayrıca, kullanıcı deneyimini iyileştirmek için dashboard'un nasıl optimize edileceği anlatılır.

17.8 Lambda Mimarisi ile Dashboard

Lambda mimarisi, hem toplu işleme hem de akış işleme yaklaşımlarını birleştiren bir mimaridir. Bu mimari, hem geçmiş verileri işlemek hem de gerçek zamanlı veri akışını yönetmek için kullanılır. Bu bölümde, Lambda mimarisinin dashboard tasarımında nasıl kullanılabileceği ve en çok satan ürünlerin nasıl gerçek zamanlı olarak takip edilebileceği anlatılır. Lambda mimarisi, büyük veri kümelerini yönetmek ve düşük gecikme süreleriyle gerçek zamanlı sonuçlar üretmek için uygundur.

Bu bölümde, Amazon'daki en çok satan 10 ürünün takip edilmesini sağlayan bir dashboard'un tasarımı için gerekli olan mimari ve teknik çözümler detaylı bir şekilde ele alınmıştır. Verilerin gerçek zamanlı işlenmesi, ölçeklenebilirlik ve kullanıcıya hızlı ve etkili bir arayüz sağlamak bu sistemin temel hedefleridir.

Ek A: Monolitler vs. Mikroservisler

Bu ek bölümde, monolitik mimari ile mikroservis mimarisi arasındaki farklar ve her bir mimarinin avantajları ve dezavantajları incelenmektedir.

- **Monolitik Mimari**: Tüm işlevlerin tek bir uygulamada toplandığı bir yapıdır. Geliştirmesi ve yönetimi başlangıçta daha kolay olabilir, ancak uygulama büyüdükçe karmaşıklaşır ve bakım zorlaşır.
    - **Avantajları**: Basit dağıtım, tüm sistemin tek bir yerde yönetilebilmesi, başlangıç maliyetlerinin düşük olması.
    - **Dezavantajları**: Büyük ve karmaşık uygulamalarda değişiklik yapmanın zor olması, ölçeklenebilirliğin sınırlı olması.
- **Mikroservis Mimari**: Uygulama, bağımsız olarak dağıtılabilen ve yönetilebilen küçük hizmetlerden oluşur. Her bir servis farklı bir işlevi yerine getirir.
    - **Avantajları**: Daha esnek ve ölçeklenebilir, bağımsız hizmetlerin yönetimi ve güncellenmesi kolay, esnek teknoloji kullanımı.
    - **Dezavantajları**: Dağıtım ve yönetim karmaşıklığı, hizmetler arasındaki iletişim maliyeti ve potansiyel performans sorunları.

Her iki mimarinin de kullanım senaryoları bulunmaktadır, bu bölümde hangi durumlarda hangi mimarinin tercih edilmesi gerektiği tartışılır.

Ek B: OAuth 2.0 Yetkilendirme ve OpenID Connect Kimlik Doğrulama

OAuth 2.0, bir kullanıcının kaynaklarına üçüncü taraf bir uygulama tarafından erişim izni vermesi için kullanılan bir yetkilendirme protokolüdür. OpenID Connect ise, OAuth 2.0 protokolünün üzerine inşa edilmiş bir kimlik doğrulama katmanıdır.

- **OAuth 2.0**: Yetkilendirme için kullanılır. Bir kullanıcının, belirli bir kaynağa (örneğin, Google Drive'daki dosyalar) erişmesine izin veren jetonlar (tokens) üretir.
    - **Akışlar (Flows)**: Yetkilendirme kodu, kullanıcı kimlik doğrulaması ve token alımı gibi süreçleri içerir.
    - **Access Tokens**: Erişim izni sağlamak için kullanılan geçici jetonlardır.
- **OpenID Connect**: Kimlik doğrulama ekleyerek OAuth 2.0'ı genişletir. Kullanıcıların kimlik bilgilerini güvenilir bir kimlik sağlayıcı üzerinden doğrulamalarına olanak tanır.

Bu bölümde, OAuth 2.0 ve OpenID Connect protokollerinin nasıl çalıştığı ve modern uygulamalarda nasıl entegre edileceği ele alınmaktadır.

Ek C: C4 Modeli

C4 Modeli, yazılım mimarisini görselleştirmek için kullanılan bir çerçevedir. "C4", dört ana seviyeyi ifade eder: Context, Container, Component ve Code.

- **Context**: Sistemin genel çerçevesini ve sistemin diğer sistemlerle nasıl etkileşimde bulunduğunu gösterir.
- **Container**: Uygulamanın ana bileşenlerini (örneğin, veritabanları, web sunucuları) ve bunların birbirleriyle nasıl etkileşime girdiğini açıklar.
- **Component**: Her bir container'ın içindeki bileşenleri ve bu bileşenlerin nasıl etkileşimde bulunduğunu gösterir.
- **Code**: Her bir bileşenin kod seviyesindeki detaylarını açıklar.

Bu model, karmaşık sistemlerin anlaşılmasını ve belgelenmesini kolaylaştırır. Bu ek bölümde, C4 Modeli'nin nasıl kullanılacağı ve uygulamaların bu modelle nasıl daha iyi anlaşılabileceği anlatılmaktadır.

Ek D: İki Aşamalı İşlem (2PC)

İki aşamalı işlem (2-Phase Commit, 2PC), bir dağıtık sistemde işlemlerin atomik olarak gerçekleştirilmesini sağlamak için kullanılan bir protokoldür. Bu protokol, işlemlerin tutarlılığını koruyarak sistemde birden fazla kaynak arasında işlem yapılmasını sağlar.

- **1. Aşama - Hazırlık (Prepare)**: Koordinatör, tüm katılımcılardan işlemi tamamlamaya hazır olup olmadıklarını sorar. Eğer tüm katılımcılar "hazır" yanıtını verirse, bir sonraki aşamaya geçilir.
- **2. Aşama - Taahhüt (Commit)**: Eğer tüm katılımcılar hazırsa, koordinatör işlemi tamamlar ve her katılımcıya "taahhüt et" komutunu gönderir. Herhangi bir katılımcı başarısız olursa, işlem geri alınır (rollback).

Bu bölümde, iki aşamalı işlem protokolünün nasıl çalıştığı, avantajları ve dezavantajları anlatılmaktadır. Özellikle dağıtık sistemlerde tutarlılığı korumak için bu protokol sıkça kullanılır, ancak sistem üzerindeki yük ve gecikme süreleri gibi dezavantajları da vardır.

Bu ek bölümler, sistem tasarımında kullanılan farklı yaklaşımlar, mimari desenler ve teknik çözümleri kapsamlı bir şekilde ele alarak, okuyucunun yazılım mimarisi ve sistem tasarımı konularında derinlemesine bilgi sahibi olmasını sağlar.