Mülakat Soruları

▼ .NET Core MVC ne işe yarar?

- Model-View-Controller (Model-Görünüm-Kontrolcü) desenini takip eder ve web uygulamalarının kodunun daha düzenli, sürdürülebilir ve test edilebilir olmasını sağlar.
  - Model, veri ve iş mantığını temsil eder.
  - View, kullanıcı arayüzünü gösterir.
  - Controller, Model ve View arasında iletişimi sağlar.

▼ Entity Framework Core ne işe yarar?

- Entity Framework Core, .NET tabanlı uygulamalarda veritabanı işlemlerini kolaylaştıran bir ORM çerçevesidir. Veritabanı tablolarını .NET nesneleriyle eşleştirerek veritabanı işlemlerini nesne tabanlı bir şekilde yönetmeyi sağlar. LINQ kullanarak sorguların yazılmasını ve verilerin çekilmesini kolaylaştırır. Aynı kodu farklı veritabanlarıyla kullanabilme imkanı sunar ve veritabanı şeması yönetimini kolaylaştırır.

▼ ORM nedir, hangi araçlar vardır?

- ORM (Object-Relational Mapping), nesne yönelimli programlama dillerinde veritabanı tablolarını nesnelerle eşleştirerek verileri nesne olarak ele almamızı

sağlayan bir tekniktir.Popüler ORM araçları şunlardır:

- Entity Framework (EF): .NET platformu için kullanılır.
- Hibernate: Java tabanlı uygulamalarda yaygın olarak kullanılır.
- Django ORM: Python tabanlı Django web çerçevesinde yerleşik olarak bulunur.
- SQLAlchemy: Python'da sıkça kullanılan ORM aracıdır.
  ORM, veritabanı işlemlerini daha basit ve taşınabilir hale getirir, böylece geliştiriciler veritabanı ile ilgili detaylara daha az odaklanabilir.

▼ WCF, Restful API nedir, farkları nelerdir?

- WCF (Windows Communication Foundation):
  - WCF, Microsoft tarafından .NET platformu için geliştirilmiş bir iletişim çerçevesidir.
  - Servis tabanlı mimariye odaklanır ve farklı sistemler arasında iletişim kurmak için kullanılır.
  - SOAP (Simple Object Access Protocol) tabanlı web servislerinin geliştirilmesine ve kullanılmasına olanak tanır.
  - Uzun süreli uygulama desteği ve karmaşık senaryolar için daha uygundur.
- RESTful API (Representational State Transfer):
  - RESTful API, HTTP protokolünü kullanarak web servislerini tasarlamak için popüler bir yöntemdir.
  - Verileri temsil eden kaynakları (resource) benzersiz URI'lar ile tanımlar ve HTTP metodları (GET, POST, PUT, DELETE) ile bu kaynaklara erişir.
  - Platform ve dil bağımsızdır, farklı istemci cihazlarla kolayca kullanılabilir.
  - Daha basit ve hafif bir yapıya sahip olduğu için özellikle web ve mobil uygulamalar için tercih edilir.

Farkları:

- WCF, servis tabanlı ve SOAP protokolünü kullanırken; RESTful API, HTTP tabanlı ve REST prensiplerini takip eder.

- WCF, daha çok .NET platformuna özgüdür ve daha kapsamlı işlevsellik sunar. RESTful API ise daha basit ve genellikle platform ve dil bağımsızdır.
- WCF, WS-* (Web Services) standartlarına uyar ve güvenlik, oturum yönetimi gibi karmaşık senaryoları desteklerken; RESTful API, daha hafif ve basit olduğu için sadece temel HTTP fonksiyonlarını kullanır.

Genel olarak, WCF genellikle iş kurallarına ve karmaşık senaryolara sahip büyük ölçekli uygulamalarda kullanılırken, RESTful API daha basit ve hafif projeler, özellikle web ve mobil uygulamalar için tercih edilir.

▼ N katmanlı mimaride hangi katmanları kullandınız?

- Sunum Katmanı (Presentation Layer)
  - Kullanıcı arayüzünü temsil eder.
  - Kullanıcının uygulama ile etkileşime geçtiği yerdir.
  - Veri girişi ve çıkışı gibi kullanıcı ile ilgili işlemleri yönetir.
  - Web uygulamalarında görüntülenen sayfalar, formlar ve görsel bileşenler bu katmanda yer alır.
- İş Katmanı (Business Layer) (BLL)
  - İş mantığını ve iş kurallarını yönetir.
  - Verilerin işlenmesi, doğrulanması ve işlenmesi gibi işlemleri gerçekleştirir.
  - Veri tabanından alınan bilgilerin işlenerek hazırlanması ve sunum katmanına aktarılması
  - işlemleri bu katmanda yapılır.
- Veri Erişim Katmanı (Data Access Layer) (DAL)
  - Veri tabanıyla iletişim sağlar.
  - Veri tabanına veri ekleme, çekme, güncelleme ve silme işlemlerini gerçekleştirir.
  - Veri erişim yöntemlerini ve veri tabanı sorgularını içerir.
  - İş katmanından gelen verileri veritabanına uygun formatta dönüştürür.

▼ Repositorypattern ne işe yarar?

- Repository Pattern, veritabanı işlemlerini soyutlama ve kolay yönetim için kullanılan bir tasarım desenidir. Bu desen, veritabanı iletişimi için kullanılan kodu bir arayüze sarmalar ve iş mantığından ayrı tutarak, veritabanı bağımlılığını azaltmayı hedefler.
- Repository Pattern, veritabanı bağımlılığını azaltarak, kodun daha esnek, bakımı daha kolay ve test edilebilir bir hale gelmesini sağlar. Bu sayede, yazılım projelerinin yönetimi ve geliştirilmesi kolaylaşır.

▼ Factory pattern ne işe yarar?

- Factory pattern, nesne yaratma süreçlerini soyutlamak ve istemci kodu ile somut nesneler arasındaki bağı azaltmak için kullanılan bir tasarım desenidir.
    - nesne yaratma işlemlerini soyut bir sınıf üzerinden yönetmeyi sağlar.Bu sayede istemci kodu, nesnelerin somut sınıflarını bilmek zorunda kalmaz ve sadece soyut sınıf ile iletişim kurar.
    - Nesne yaratma süreçleri, bir fabrika sınıfında (Factory) merkezi olarak kontrol edilir ve gerekli somut nesneleri döndürür.
    - Factory pattern, kodun daha esnek, sürdürülebilir ve yeniden kullanılabilir olmasına katkı sağlar.

▼ Singleton pattern ne işe yarar?

- Singleton Pattern, tasarım desenlerinden biridir ve sadece bir örneği (instance) olan bir sınıfın oluşturulmasını sağlar. Bu desen, bir sınıfın yalnızca bir kez oluşturulmasını ve tüm uygulama boyunca aynı örneğin kullanılmasını sağlayarak, tek bir noktadan veri paylaşımını ve erişimini kolaylaştırır.

Singleton Pattern'in işlevleri:

- Tek bir örnek: Singleton, belirli bir sınıfın yalnızca bir tane örneğinin oluşturulmasını garanti eder. Bu örnek, uygulama boyunca kullanılabilir ve birden çok yerde aynı nesneye erişimi sağlar.
- Global erişim: Singleton, tüm uygulama boyunca tek bir noktadan erişilebilir. Bu sayede, veri paylaşımı ve erişimi kolaylaştırır ve veri tutma işlemleri daha düzenli bir şekilde yapılabilir.
- Kaynak tasarrufu: Singleton, her defasında yeni bir örnek oluşturmak yerine, var olan örneği kullanarak kaynak kullanımını optimize eder.

Singleton Pattern'in kullanım alanları:

- Yapılandırma dosyaları: Uygulama yapılandırması için kullanılan bir Singleton örneği, yapılandırma dosyasının tüm uygulama boyunca tek bir yerden erişilmesini sağlar.
- Günlük kaydı (logging): Uygulama günlüklerinin tek bir nesne üzerinden tutulması ve kaydedilmesi için Singleton kullanılabilir.
- Veritabanı bağlantıları: Bir veritabanı bağlantısının tek bir örneği üzerinden tüm uygulama boyunca veritabanına erişimi sağlamak için Singleton tercih edilebilir.
- Singleton Pattern, özellikle küçük ve orta ölçekli uygulamalarda kullanışlı olabilir. Ancak, gereğinden fazla kullanıldığında karmaşık ve bağımlılıklara yol açabileceği için dikkatli bir şekilde uygulanmalıdır.

▼ Solid prensipleri nelerdir?

- SOLID prensipleri, temiz kod yazma, değişikliklere karşı dirençli ve yeniden kullanılabilir uygulamalar geliştirme konusunda önemli yönergeler sunan beş temel prensiptir .
    - Single Responsibility Principle (SRP) - Tek Sorumluluk Prensibi:
      Her sınıfın, yalnızca bir tek sorumluluğu olmalıdır. Bir sınıfın değişebilecek nedenlerinin sadece bir tanesi olmalıdır. Bu sayede sınıflar daha basit, sürdürülebilir ve yeniden kullanılabilir olur.
    - Open/Closed Principle (OCP) - Açık/Kapalı Prensibi:
      Mevcut kodu değiştirmek yerine, yeni davranışlar eklemek için tasarım yapılmalıdır. Modüller açık olmalı (yeni davranışlar eklemeye izin verir), ancak değiştirilemez olmalıdır. Böylece değişikliklerin mevcut kodu etkilemesi engellenir.
    - Liskov Substitution Principle (LSP) - Liskov Yerine Geçme Prensibi:
      Alt sınıflar, üst sınıfların yerine geçebilmelidir. Yani, üst sınıfın kullandığı herhangi bir nesnenin yerine, alt sınıfın nesnesi geçebilmelidir. Böylece kodlar, alt sınıflar tarafından genişletilebilir ve değiştirilebilir.
    - Interface Segregation Principle (ISP) - Arayüz Ayrımı Prensibi:
      İstemci, kullandığı herhangi bir arayüzü, ihtiyacı olmayan metodlar nedeniyle zorlamamalıdır. Bu nedenle, arayüzlerin daha küçük ve daha spesifik olması tercih edilir.

    - Dependency Inversion Principle (DIP) - Bağımlılıkları Tersine Çevirme Prensibi:
      Üst seviye modüller, alt seviye modüllere bağlı olmamalıdır. Bu prensibe göre, soyutlamalara ve arayüzlere dayalı bağımlılıklar kullanılarak, yüksek düzey modüller düşük düzey modüllere bağımlı olacak şekilde tasarlanmalıdır.

▼ Dependency injection nedir, nasıl kullanılır?

- Dependency Injection (Bağımlılık Enjeksiyonu), bir yazılım bileşeninin diğer bileşenlerin oluşturulması veya kullanılması için doğrudan bağımlı olmaktan çıkarılıp, dışarıdan verilerek kullanılmasını sağlayan bir tasarım desenidir. Bu sayede, bileşenler arasındaki bağımlılık azalır ve uygulama daha esnek, kolay test edilebilir ve bakımı daha kolay hale gelir.

Dependency Injection nasıl kullanılır?

- Dependency Injection, bir bileşenin gereksinim duyduğu diğer bileşenlerin nesnelerini dışarıdan almasıyla gerçekleştirilir. Bunun için genellikle üç yöntem kullanılır:
- Constructor Injection: Bağımlılık enjeksiyonunu, bileşenin constructor (kurucu) metodu aracılığıyla gerçekleştirir. Böylece, bileşenin nesnesi oluşturulurken bağımlı olduğu diğer nesneler de constructor parametreleri olarak geçirilir.

```
public class MyService
{
  private readonly IDependency dependency;

  public MyService(IDependency dependency)
  {
    this.dependency = dependency;
  }

// ... diğer metotlar ...
}
```

- Property Injection: Bağımlılık enjeksiyonu, bileşenin özelliklerine diğer bileşenlerin nesnelerinin atanmasıyla yapılır.

```
public class MyService
{
```

```
  public IDependency Dependency { get; set; }

// ... diğer metotlar ...
}
```

- Method Injection: Bağımlılık enjeksiyonu, bileşenin bir metodu aracılığıyla gerçekleştirilir. Bu yöntem, bir metot parametresi olarak diğer bileşenin nesnesini alır.

```
public class MyService
{
   public void DoSomething(IDependency dependency)
{
   // dependency kullanımı...
}

   // ... diğer metotlar ...
}
```

Dependency Injection, uygulamayı daha esnek ve test edilebilir hale getirir, kodun bağımlılıklarını azaltır ve kodun yeniden kullanılabilirliğini arttırır. Ayrıca, uygulamadaki bileşenlerin değiştirilmesini ve güncellenmesini kolaylaştırır. Bu nedenle, modern yazılım geliştirme yöntemlerinde yaygın bir şekilde kullanılan önemli bir tasarım desenidir.

▼ IOC containerlar ne işe yarar, popüler olanları hangileridir?

- IOC (Inversion of Control) containerlar, yazılım geliştirme sürecinde bağımlılıkları yönetmek ve nesnelerin oluşturulmasını, bağımlılıklarının çözülmesini ve yönetilmesini otomatikleştirmek için kullanılan araçlardır.
- Bu containerlar, Dependency Injection (Bağımlılık Enjeksiyonu) prensibini uygulayarak, bileşenler arasındaki bağımlılıkları kod içinde belirli bir yerde (container) toplar ve bu bağımlılıkları otomatik olarak çözer. Böylece kodun daha sürdürülebilir, test edilebilir ve esnek olmasını sağlar.

Popüler IOC containerlardan bazıları şunlardır:

1. Unity: Microsoft tarafından geliştirilen ve .NET tabanlı uygulamalarda kullanılan popüler bir IOC containerdır.

2. Ninject: .NET platformunda yaygın olarak kullanılan, açık kaynaklı ve hafif bir IOC containerdır.
3. Autofac: Açık kaynaklı ve özellikle .NET Core uygulamalarında tercih edilen bir IOC containerdır.
4. Spring Framework: Java tabanlı uygulamalarda yaygın olarak kullanılan IOC containerlardan biridir.
5. Dagger: Android uygulamaları için Google tarafından geliştirilmiş hızlı ve etkili bir IOC containerdır.

Bu IOC containerlar, bağımlılıkları yönetmek ve Dependency Injection prensibini uygulamak için geliştiricilere kolaylık sağlar. Projenizin gereksinimlerine ve kullandığınız platforma uygun bir IOC container seçerek, kodunuzu daha esnek ve yönetilebilir hale getirebilirsiniz.

▼ SP vs View vs Function arasındaki farklar nelerdir?

- SP (Stored Procedure), View ve Function (Fonksiyon) veritabanlarında kullanılan farklı yapı ve amaçlara sahip nesnelerdir. İşte bu üçünün temel farkları:

Stored Procedure (SP):

- SP, veritabanında saklanan ve adlandırılmış bir dizi SQL ifadesinden oluşan bir programdır.
  Veritabanında karmaşık işlemleri gerçekleştirmek için kullanılır.
- SP, birden çok SQL sorgusunu içerebilir ve bu sorguları çalıştırabilir.
- SP, veritabanında tutulan bir araştırma veya raporlama işlemi gibi sık sık tekrar eden işlemleri birleştirmek için kullanılabilir.

View:

- View, veritabanında saklanan ve adlandırılmış bir SQL sorgusundan oluşan, sanal bir tablodur.
- View, bir veya daha fazla tablodan veri çekerek, bu verileri mantıksal olarak gruplandırır ve belirli bir amaca yönelik bir görünüm sunar.
- View'lar, karmaşık sorguları basitleştirmek ve verileri daha anlamlı bir şekilde sunmak için kullanılır.

- View'lar, uygulama tarafından tablolara erişim yerine kullanılabilir ve veritabanında güncellenemezler (read-only).

Function (Fonksiyon):

- Function, bir dizi SQL ifadesinden oluşan ve belirli bir işlevi yerine getiren, geri dönüş değeri olan bir işlem bloğudur.
- Function, veritabanında belirli işlemleri gerçekleştirmek ve sonuçları döndürmek için kullanılır.
- Function, Stored Procedure'den farklı olarak, bir değeri geri döndürür ve başka bir sorgunun sonucunda bir değeri kullanmak için kullanılabilir.
- Function'lar, veritabanında belirli hesaplamaları yapmak veya verileri işlemek için kullanılabilir.

Özetle, Stored Procedure, karmaşık işlemleri gerçekleştirmek için kullanılırken, View, verileri daha anlamlı bir şekilde sunmak için ve Function, belirli hesaplamaları yapmak veya işlemek için kullanılır. Her birinin farklı bir amacı ve kullanım alanı vardır ve veritabanı tasarımında doğru seçim yapmak önemlidir.

▼ Trigger ne işe yarar?

- Trigger, veritabanında belirli olaylar gerçekleştiğinde otomatik olarak çalışan bir veritabanı nesnesidir.
- Trigger'lar, veri bütünlüğünü sağlamak, veri denetimi yapmak, veri izlemek ve iş akışı kontrolü gibi işlevleri yerine getirmek için kullanılır. Ancak, dikkatli bir şekilde kullanılmalıdır çünkü yanlış tasarlanmış trigger'lar veritabanı performansını etkileyebilir ve beklenmeyen sonuçlara yol açabilir.

▼ İndex nedir? Ne sağlar?

- Index, veritabanında belirli sütunlara göre düzenlenen bir yapıdır. Bu düzenleme sayesinde veri erişimi hızlanır ve sorgu performansı artar. Ancak, ekstra alan kaplama ve güncelleme yavaşlığı gibi dezavantajları da vardır.
- Index'ler, doğru kullanıldığında veritabanı performansını önemli ölçüde artırabilir, ancak gereksiz veya yanlış tasarlanmış Index'ler veri tabanı performansını olumsuz etkileyebilir.

▼ JS kütüphanesini kullanan teknolojiler hangileridir?

▼ Swot analiziniz, güçlü yanlarınız, zayıf yanlarınız?

▼ Takımçalışmasına yatkın mısınız?

▼ Projede yaşadığınız en büyük sıkıntı neydi? Nasıl çözdünüz?

▼ Literatürü hangi sitelerden takip ediyorsunuz?

▼ TFS, Jira, Git ne işe yarar?

- TFS (Team Foundation Server), Jira ve Git, yazılım geliştirme süreçlerini yönetmek ve yazılım projelerini takip etmek için kullanılan üç farklı araçtır.

TFS (Team Foundation Server):

- TFS, Microsoft tarafından geliştirilen ve özellikle Microsoft tabanlı yazılım geliştirme süreçlerine yönelik olarak tasarlanmış bir işbirliği ve kod yönetim platformudur. TFS, bir dizi aracı içerir ve aşağıdaki işlevleri sağlar:
  - Sürüm kontrolü: Kodların sürüm kontrolünü sağlar, böylece ekip üyeleri kodları güvenli bir şekilde paylaşabilir, sürümlere göre çalışabilir ve değişiklikleri geri alabilir.
    Proje ve iş görevleri yönetimi: Ekip üyeleri arasında iş görevlerini tanımlamak, takip etmek ve atamak için kullanılır.
  - Takım işbirliği: Ekip üyeleri, paylaşılan kod, belgeler ve diğer projeler aracılığıyla birlikte çalışabilir ve iletişim kurabilir.
    Entegre raporlama: Proje ilerlemesini, hataları ve diğer metrikleri takip etmek için raporlama özelliklerine sahiptir.

Jira:

- Jira, Atlassian tarafından geliştirilen popüler bir proje yönetim ve iş takip aracıdır. Genellikle çevik (agile) yazılım geliştirme süreçlerine odaklanır ve esnek konfigürasyon seçenekleri sunar. Jira'nın temel işlevleri şunlardır:
  - Görev yönetimi: Proje görevlerini tanımlamak, atamak, takip etmek ve raporlamak için kullanılır.
  - Hata yönetimi: Yazılım hatalarını takip etmek ve çözmek için kullanılır.
    Proje takibi: Proje ilerlemesini görsel panolar, raporlar ve gösterge tablolarıyla takip etmeye olanak tanır.

    - Entegre işbirliği: Ekip üyeleri, belgeleri ve paylaşılan bilgileri birlikte çalışmak için kolayca paylaşabilir.

Git:

- Git, dağıtık bir sürüm kontrol sistemi olarak bilinir ve yazılım projelerinde kod yönetimi için kullanılır. Git'in temel işlevleri şunlardır:
    - Sürüm kontrolü: Kodların değişikliklerini takip eder, versiyonlarını yönetir ve geliştiricilere kodlar üzerinde kolayca çalışma ve değişikliklerini geri alma imkanı sağlar.
    - Dal (branch) yönetimi: Paralel kod geliştirme için dallar oluşturulmasına ve birleştirilmesine olanak tanır.
    - Dağıtık yapı: Merkezi bir sunucuya bağımlı olmaksızın tüm geliştiricilerin yerel olarak çalışmasına izin verir.
    - Topluluk desteği: Git, büyük bir açık kaynaklı topluluk tarafından desteklenir ve geniş bir ekosistem sunar.

Özetle, TFS, Jira ve Git, yazılım geliştirme süreçlerini desteklemek ve yazılım projelerini yönetmek için kullanılan farklı araçlardır. TFS, Microsoft tabanlı geliştirme için özel olarak tasarlanmıştır, Jira genellikle çevik yazılım geliştirme için tercih edilirken, Git ise sürüm kontrolü ve kod yönetimi için yaygın olarak kullanılır.

▼ SOA nedir?

SOA, "Service-Oriented Architecture" (Hizmet Odaklı Mimari) kavramının kısaltmasıdır. SOA, yazılım uygulamalarını modüler ve bağımsız hizmetler olarak tasarlamayı ve bu hizmetlerin birbiriyle haberleşerek işlevselliği sağlamasını temel alan bir mimari yaklaşımıdır.

Kısaca açıklamak gerekirse:

- SOA, bir uygulamayı birçok bağımsız hizmete bölmeyi ve bu hizmetlerin iş mantığına odaklanarak geliştirilmesini sağlar.
- Her hizmet, belirli bir işlevi gerçekleştiren ve genellikle standart bir arabirim (API) ile diğer hizmetlerle iletişim kuran birimdir.
- Hizmetler, genellikle kendi başlarına çalışabilir, değiştirilebilir ve ölçeklenebilir olmalıdır.

- SOA, esneklik, sürdürülebilirlik ve hızlı geliştirme gibi avantajlar sağlayarak iş gereksinimlerine daha iyi uyum sağlayan bir mimari sunar.
- Bu mimari, farklı platformlar ve diller arasında uyumluluk sağlama ve sistemlerin birleştirilmesi gibi entegrasyon ihtiyaçlarını karşılamak için de yaygın olarak kullanılır.

SOA, büyük ve karmaşık uygulamaların geliştirilmesinde, iş süreçlerini esnek ve modüler bir şekilde yönetmede ve sistemlerin birbiriyle entegre edilmesinde etkili bir yaklaşımdır.

▼ Microservice ve monolotik arasındaki farklar nelerdir?

Microservice (Mikro Hizmet) ve Monolitik mimari, yazılım uygulamalarını farklı şekillerde organize eden iki farklı yaklaşımdır. İşte Microservice ve Monolitik mimari arasındaki temel farklar:

Monolitik Mimari:

- Monolitik mimaride, tüm uygulama bileşenleri tek bir büyük ve bütünleşik bir yapıda yer alır.
  Uygulama, tek bir büyük kod tabanında geliştirilir ve dağıtılır.
- Bütünleşik yapısı nedeniyle, kodun bakımı, testi ve dağıtımı biraz daha karmaşıktır.
  Tek bir hata veya sorun, tüm uygulamayı etkileyebilir.
- Büyüdükçe ve karmaşıklaştıkça, kod tabanının yönetimi ve geliştirilmesi zorlaşabilir.

Microservice Mimari:

- Microservice mimarisinde, uygulama farklı ve bağımsız hizmetlere bölünür. Her hizmet, belirli bir işlevselliği temsil eder ve kendi veritabanına sahip olabilir.
- Hizmetler, birbirleriyle API'ler aracılığıyla iletişim kurarlar. Bu sayede her hizmet, bağımsız olarak geliştirilebilir, dağıtılabilir ve ölçeklendirilebilir.
- Her hizmetin kendi kod tabanı ve veritabanı olduğu için, hata veya sorunlar bir hizmeti etkiler ve diğer hizmetler etkilenmez.
- Geliştirme ve dağıtım, hizmetleri bağımsız olarak yönetilebilir hale getirir, böylece ekipler daha hızlı ve etkili bir şekilde çalışabilir.

- Karmaşık sistemlerin modüler hale gelmesini sağlar ve yeni özellikler eklemeyi ve mevcut hizmetleri güncellemeyi daha kolay hale getirir.

Genel olarak, Monolitik mimari basit ve küçük uygulamalarda tercih edilebilirken, Microservice mimari, büyük, karmaşık ve ölçeklenebilir uygulamalarda tercih edilir. Ancak, her mimarinin kendi avantajları ve dezavantajları vardır ve kullanılacak mimari, uygulamanın gereksinimleri ve boyutu göz önünde bulundurularak belirlenmelidir.

▼ Docker, kubernates ne işe yarar?

Docker ve Kubernetes, yazılım uygulamalarının konteyner tabanlı dağıtımını ve yönetimini kolaylaştıran popüler açık kaynaklı teknolojilerdir.

Docker:

- Docker, uygulamaların yazılım konteynerleri içinde çalıştırılmasını ve dağıtılmasını sağlayan bir konteynerleme platformudur. Konteynerler, uygulamaların ve tüm bağımlılıklarının izole bir ortamda çalışmasını sağlayan hafif ve taşınabilir bir yapıdır. Docker, uygulamaları donanımdan bağımsız olarak çalıştırabilir ve herhangi bir ortamda tutarlı bir şekilde çalışmalarını sağlar. Böylece uygulamaların hızlı bir şekilde dağıtılması, ölçeklendirilmesi ve güncellenmesi kolaylaşır. Yazılım geliştirme, test ve dağıtım süreçlerini optimize eder ve sunucu kaynaklarını daha verimli kullanmanıza olanak tanır.

Kubernetes:

- Kubernetes, kısaca K8s olarak da bilinir, Docker konteynerlerini otomatik olarak yönetmek ve orkestrasyonunu sağlamak için kullanılan açık kaynaklı bir platformdur. Kubernetes, birden çok konteynerin yönetimi için güçlü bir araçtır ve yüksek ölçekli uygulamaların, mikro hizmet mimarilerinin ve dağıtık sistemlerin kolay yönetimi için kullanılır. Kubernetes, konteynerlerin hızlı bir şekilde ölçeklenmesini, dağıtılmasını ve izlenmesini sağlar. Ayrıca otomatik hata toleransı ve yedekleme gibi özellikler sunarak yüksek kullanılabilirlik sağlar.

Özetle, Docker konteynerleri uygulamaların taşınabilir ve izole bir şekilde çalışmasını sağlarken, Kubernetes bu konteynerleri otomatik olarak yöneterek, ölçeklendirerek ve izleyerek uygulamaların daha verimli ve güvenilir bir şekilde çalışmasını sağlar. Bu iki teknoloji birlikte, modern yazılım uygulamalarının hızlı bir şekilde dağıtılmasını ve yönetilmesini sağlar.

▼ Mongodb ne işe yarar? Hangi firmalar kullanır?

MongoDB, açık kaynaklı bir NoSQL veritabanı yönetim sistemidir. NoSQL (Not Only SQL) terimi, ilişkisel veritabanlarından farklı olarak yapılandırılmamış ve daha esnek veri modellerine sahip olan veritabanı sistemlerini tanımlar.

MongoDB'nin temel işlevleri şunlardır:

- Doküman Tabanlı Veritabanı: MongoDB, JSON benzeri BSON formatında dokümanları kullanır. Her bir doküman, ilişkisel veritabanlarında bir satıra karşılık gelir ve verilerin daha yapılandırılmamış bir şekilde saklanmasını sağlar.
- Yüksek Ölçeklenebilirlik: MongoDB, dağıtık veritabanı mimarisi ile yüksek ölçeklenebilirlik sunar. Büyük miktarda veriyi kolayca depolayabilir ve işleyebilir.
- Esneklik: Veri şemasının esnek olması, MongoDB'yi farklı ve değişen veri yapılarına uygun hale getirir. Bu, geliştirme süreçlerini hızlandırır ve yeni gereksinimlere hızlı bir şekilde cevap verilmesini sağlar.
- Yüksek Performans: Veri tabanı yönetiminin hızlı ve etkin olması, uygulamaların performansını artırır.
- Harita/Kısmi Dizinleme: MongoDB, verileri daha hızlı sorgulamak ve indekslemek için harita ve kısmi dizinleme işlevlerini destekler.

MongoDB kullanımı, özellikle büyük ve veri yoğunluğu olan projelerde tercih edilir. İşte MongoDB kullanıcılarından bazıları:

- Uber
- Facebook
- Google
- Adobe
- eBay
- Cisco
- IBM
- Twitter
- LinkedIn

- Nokia

Bu firmalar, MongoDB'nin esnekliği ve ölçeklenebilirliği sayesinde büyük veri işleme ve yönetme ihtiyaçlarını karşılamak için bu veritabanı sistemini tercih ederler.

▼ TDD nedir?

TDD (Test Driven Development), yazılım geliştirme sürecinde bir test odaklı yaklaşımı temsil eden bir yazılım geliştirme yöntemidir. TDD, yazılımın test edilebilir, güvenilir ve sürdürülebilir olmasını sağlamak için testleri önceden yazarak kod geliştirme sürecini yönetir. Temel adımları şu şekildedir:

- Test Yazma (Test First Approach): Yazılım geliştirme süreci, önce testlerin yazılmasıyla başlar. Kod henüz yazılmamış olsa bile, geliştirici, yazılacak kodun davranışını ve özelliklerini tanımlayan testleri oluşturur.
- Test Başarısızlığı (Red): Henüz yazılmamış olan kod nedeniyle, oluşturulan testler başarısız olur (kırmızı durumda). Bu, beklentilerin henüz karşılanmadığı anlamına gelir.
- Kod Yazma (Green): Şimdi, yazılacak kodu, oluşturulan testleri başarılı kılmak için yazılır. Yani, testlerin başarılı olmasını sağlamak için minimum kod yazılır (sadece yeterli düzeyde). Bu aşamada kod kırmızıdan yeşile döner.
- Kod İyileştirme (Refactor): Başarılı testlere sahip olduğunuzda, kodu optimize edebilir ve iyileştirebilirsiniz. Bu, daha temiz, daha düzenli ve daha sürdürülebilir kod yazmanızı sağlar.
- Bu adımlar sürekli tekrar edilir, her yeni özellik eklenirken ve her kod değişikliğinde testlerin başarılı olması sağlanır. Böylece, sürekli olarak kod kalitesi artırılır ve uygulamanın güvenilirliği sağlanmış olur.

TDD'nin avantajları şunlardır:

- Yazılım hatalarının erken tespiti ve giderilmesi.
- Kod kalitesinin artırılması ve sürdürülebilirlik sağlanması.
- Kodun doğruluğunun ve istikrarının artması.
- Yeniden kullanılabilir ve modüler kod yapısının oluşturulması.
- Kod değişiklikleri ve iyileştirmelerin daha az riskli ve daha kontrollü olması.

- TDD, yazılım geliştirme süreçlerini daha disiplinli ve güvenilir hale getirirken, başlangıçta biraz daha fazla çaba ve zaman gerektirebilir. Ancak, uzun vadede yazılım projelerinin başarısını artırır ve daha az hata ile daha sağlam bir temel oluşturur.

▼ Azure, AWS hakkında neler biliyorsunuz?

Azure ve AWS, dünyanın en büyük ve en yaygın bulut hizmet sağlayıcılarıdır. Her ikisi de farklı şirketler tarafından sunulsa da, bulut bilişim alanında bir dizi hizmet ve çözüm sunarak benzer amaçları paylaşırlar.

Azure (Microsoft Azure):

- Microsoft tarafından geliştirilen ve sunulan bir bulut hizmet platformudur.
  Küresel çapta veri merkezleri üzerinden bulut hizmetleri sağlar.
  IaaS (Altyapı olarak Hizmet), PaaS (Platform olarak Hizmet) ve SaaS (Yazılım olarak Hizmet) çözümleri sunar.
- Microsoft ürünleri ve teknolojileriyle entegrasyonu sağlar.
  .NET, C#, Visual Studio gibi Microsoft teknolojilerine özel çözümler sunar.
  Geniş bir hizmet yelpazesine sahiptir, veritabanı hizmetleri, yapay zeka, büyük veri işleme, sanallaştırma, web uygulamaları ve daha fazlasını içerir.
  Büyük işletmelerden KOBİ'lere kadar farklı ölçeklerdeki müşterilere hizmet verir.

AWS (Amazon Web Services):

- Amazon tarafından geliştirilen ve sunulan bir bulut hizmet platformudur.
  Dünyanın dört bir yanındaki veri merkezlerinde bulut hizmetleri sağlar.
- Öncü ve en büyük bulut hizmet sağlayıcısı olarak kabul edilir.
- Geniş bir hizmet yelpazesine sahiptir, depolama, veritabanı, yapay zeka, analitik, sunucular, IoT ve daha fazlasını içerir.
- IaaS, PaaS ve SaaS çözümleri sunar.
- AWS Lambda gibi serverless hizmetleri, kolay ölçeklenebilirlik ve esneklik sağlar.
  Çeşitli sektörlerdeki müşterilere hizmet verir ve başta internet tabanlı şirketler olmak üzere birçok büyük şirket tarafından tercih edilir.

Her iki platform da güçlü ve güvenilir bulut hizmetleri sunar ve müşterilere farklı ihtiyaçlarına uygun çözümler sunar. Seçim, uygulamanızın gereksinimlerine ve

varolan altyapınıza bağlı olarak yapılmalıdır. İşletmeler, hizmet özelliklerini, maliyetleri, performansı ve desteklenen teknolojileri göz önünde bulundurarak en uygun bulut hizmet sağlayıcısını seçmelidir.

▼ CI/CD hakkında neler biliyorsunuz?

CI/CD (Continuous Integration/Continuous Deployment), yazılım geliştirme süreçlerinde otomasyon ve sürekli olarak entegrasyon ve dağıtımı sağlayan bir dizi uygulama ve yöntemler bütünüdür. CI/CD, yazılım projelerinin hızlı bir şekilde geliştirilmesini, test edilmesini ve dağıtılmasını kolaylaştırır.

Continuous Integration (Sürekli Entegrasyon):

- Geliştiricilerin yazılım kodlarını sürekli olarak birleştirip, paylaşması ve test etmesini sağlar.
  Kodlar sürekli bir merkezi havuzda toplanır ve sık sık birleştirilir, böylece çakışmalar ve uyumsuzluklar önlenir.
- Otomatik yapı ve test süreçleriyle hataların erken tespiti ve düzeltilmesi desteklenir.
  Entegrasyon hatalarının ve uyumsuzlukların azalmasını sağlar.
  Continuous Deployment (Sürekli Dağıtım):
- Otomatik yapılan testler başarılı olduğunda, yazılımın canlı ortama sürekli olarak dağıtılmasını sağlar.
- Değişikliklerin kullanıcıya en kısa sürede sunulmasını sağlar.
- Sürekli dağıtım, geliştirme ve operasyon ekipleri arasındaki işbirliğini artırır.
- Hızlı geri dönüşler ve geri alma imkanı sunar.

CI/CD'nin avantajları şunlardır:

- Hızlı ve sık sık kod teslimatı.
- Hataların ve uyumsuzlukların erken tespiti ve düzeltilmesi.
- Tekrar eden ve zaman alıcı işlemlerin otomatikleştirilmesi.
- Verimlilik ve işbirliğini artırır.
- Kullanıcılara sürekli güncel ve kaliteli yazılım sunulması.

CI/CD, modern yazılım geliştirme süreçlerinde önemli bir rol oynar ve Agile ve DevOps yaklaşımları ile birlikte kullanılır. Otomasyon ve sürekli test etme işlemleri sayesinde yazılımın kalitesi artar, müşteri memnuniyeti sağlanır ve rekabet avantajı elde edilir.

▼ Unit test hakkında neler biliyorsunuz?

Unit test, yazılım geliştirme sürecindeki test türlerinden biridir ve belirli birimleri (fonksiyonlar, sınıflar veya metodlar gibi) izole etmek ve test etmek amacıyla kullanılır. Bu testler, uygulamanın her bir bileşeninin (birim) istenen davranışları doğru bir şekilde sergilediğini doğrulamak için otomatik olarak oluşturulan test kodlarıdır.

Unit testlerin temel özellikleri şunlardır:

- İzole Testler: Unit testler, sadece belirli birimleri (bir fonksiyon, metot veya sınıfı) izole ederek, diğer bileşenlerle etkileşimini en aza indirir. Böylece testin sonuçları, ilgili birimin hatalarını ve eksikliklerini açıkça ortaya koyar.
- Otomatik Testler: Unit testler otomatik olarak yürütülen testlerdir. Geliştiriciler, testleri yazarken manuel olarak test işlemleri gerçekleştirmek zorunda değildir. Bu, sürekli entegrasyon ve sürekli dağıtım (CI/CD) süreçlerinde testlerin kolayca entegre edilmesini sağlar.
- Küçük Ölçekli Testler: Unit testler, yazılımın en küçük parçaları olan birimlerin test edilmesine odaklanır. Birimler, bağımsız olarak test edilebilir ve hataların kök nedenlerini belirlemede yardımcı olur.
- Tekrarlanabilirlik: Unit testler, sürekli olarak tekrarlanabilir ve güvenilir sonuçlar üretir. Aynı test her zaman aynı sonuçları vermeli ve kod üzerinde yapılan değişikliklerin hataları ortaya çıkarması sağlanır.

Unit testlerin faydaları şunlardır:

- Hataların erken tespiti: Unit testler, hataları geliştirme sürecinin erken aşamalarında tespit etmeye yardımcı olur, böylece maliyetli hata düzeltme süreçleri azaltılır.
- Güvenilirlik: Her birimin (bir fonksiyon, metot veya sınıfın) doğruluğu ve işlevselliği test edildiği için kodun genel güvenilirliği artar.

- Dökümantasyon: Unit testler, birimlerin nasıl çalıştığı hakkında belgelenmiş bilgiler sunar.
- Refaktoring (Kod Yeniden Düzenleme) Desteği: Kodda yapılacak değişikliklerin, mevcut testlerin etkilenip etkilenmediğini anlamak için unit testler kullanılır.

Unit testler, yazılım kalitesini artıran ve güvenilir bir kod tabanı oluşturan önemli bir geliştirme uygulamasıdır.

▼ RabbitMQ nedir?

RabbitMQ, açık kaynaklı bir mesaj sırası (message queue) yazılımıdır. Mesaj sırası, yazılım uygulamalarının birbiriyle veri alışverişini yapmasına yardımcı olan bir iletişim mekanizmasıdır. RabbitMQ, popüler bir mesaj sırası uygulaması olup, geliştiricilere ölçeklenebilir ve güvenilir bir şekilde mesaj tabanlı sistemler inşa etmelerine olanak tanır.

RabbitMQ, özellikle dağıtık ve mikro hizmet mimarileri için uygun olan bir mesajlaşma aracıdır. Birçok uygulama ve sistem arasında asenkron iletişimi kolaylaştırır ve farklı platformlar arasında veri alışverişini sağlar.

Anahtar özellikleri şunlardır:

- Mesaj Kuyruğu: İletişim kurmak isteyen uygulamalar, mesajları RabbitMQ üzerinden bir kuyruğa gönderir ve hedef uygulama bu kuyruktan mesajları alır. Bu, gönderen ve alıcı arasında bağımsız bir iletişim sağlar.
- Asenkron İletişim: RabbitMQ, mesaj tabanlı iletişim sağlayarak gönderen ve alıcı uygulamaların zaman açısından bağımsız olarak çalışmasını sağlar. Gönderilen bir mesaj anında işlenmek zorunda değildir ve hedef uygulama hazır olduğunda mesajı alır.
- Yayın-Abone Modeli: RabbitMQ, yayın-abone (publish-subscribe) modelini destekler. Bir gönderici, mesajları birden çok alıcıya yayınlayabilir ve alıcılar bu mesajları dinlemek için uygun kuyruklara abone olabilirler.
- Çoklu Protokol Desteği: RabbitMQ, farklı platformlar ve diller arasında iletişim kurmak için popüler mesajlaşma protokolleri olan AMQP (Advanced Message Queuing Protocol), MQTT, STOMP ve HTTP gibi protokolleri destekler.
- Yüksek Ölçeklenebilirlik: RabbitMQ, yüksek iş yüklerini ve yoğun trafiği kolayca işleyebilecek şekilde ölçeklenebilir bir yapıya sahiptir.

RabbitMQ, dağıtık sistemler, mikro hizmet mimarileri ve büyük ölçekli uygulamalar için veri alışverişi ve iletişim için güvenilir ve etkili bir araçtır.

▼ Access Modifier’lar nelerdir? Ne işe yararlar?

Access Modifier'lar (Erişim Belirleyiciler), nesne yönelimli programlama dillerinde sınıfların, üyelerin ve metotların erişilebilirlik düzeyini belirleyen anahtar kelimelerdir. Bu belirleyiciler, bir sınıfın içindeki üyelerin diğer sınıflar veya kod parçacıkları tarafından nasıl erişilebileceğini kontrol eder. Dört temel access modifier bulunmaktadır:

- Public (Herkese Açık): Public access modifier, sınıfın her yerinden ve diğer sınıflardan erişime izin verir. Bir sınıf üyesi public olarak tanımlandığında, o üyeye diğer sınıflardan rahatlıkla erişilebilir.
- Private (Özel): Private access modifier, sadece tanımlandığı sınıf içerisinden erişime izin verir. Başka hiçbir sınıf veya kod parçacığı, private olarak tanımlanan üyelere doğrudan erişemez.
- Protected (Korumalı): Protected access modifier, tanımlandığı sınıf içinden ve bu sınıftan türetilmiş alt sınıflardan erişime izin verir. Alt sınıflar, korumalı üyelere erişebilir ve kullanabilirler, ancak sınıfın dışındaki diğer kod parçacıkları erişemez.
- Default (Package-private, Erişim Belirtilmemiş): Java gibi bazı dillerde, bir üye için access modifier belirtilmediğinde, otomatik olarak default olarak kabul edilir. Bu durumda, aynı paket içerisinde bulunan diğer sınıflardan erişime izin verilirken, paket dışındaki sınıfların erişmesine izin verilmez.

Access modifier'ların temel işlevleri şunlardır:

- Erişim Kontrolü: Access modifier'lar, üyelerin diğer sınıflar veya kod blokları tarafından nasıl erişilebileceğini kontrol eder ve izin verilen sınırlar içinde üyelerin kullanılmasını sağlar.
- Bilgi Gizleme (Encapsulation): Private ve korumalı access modifier'lar, sınıfın iç yapısını gizleyerek, sınıfın daha güvenli ve daha az bağımlı olmasını sağlar. Bu, güvenli ve sürdürülebilir bir kod tasarımı sağlamak için önemlidir.
- Kalıtım (Inheritance): Protected access modifier, kalıtım ile ilişkili olarak, alt sınıfların bazı üyelere erişebilmesini sağlar, böylece kodun yeniden kullanımını ve genişletilebilirliğini kolaylaştırır.

Access modifier'lar, nesne yönelimli programlamada güçlü bir mekanizma sağlayarak, kodun doğru şekilde çalışmasını ve uygulamanın güvenliğini sağlar. Doğru access modifier kullanımı, yazılımın kalitesini ve bakım kolaylığını artırır.

▼ Abstract class ve interface farklar nelerdir?

Abstract class ve interface, nesne yönelimli programlamada farklı amaçlar için kullanılan iki farklı yapıdır. İşte abstract class ve interface arasındaki temel farklar:

Abstract Class:

- Abstract class, soyut sınıf olarak da adlandırılır ve bir sınıfın örneğinin doğrudan oluşturulamayacağı, ancak diğer sınıflar tarafından miras alınarak kullanılabileceği bir sınıf türüdür.
- Abstract class, içinde soyut (abstract) ve somut (concrete) metotları içerebilir. Soyut metotlar, sadece imza (signature) tanımı olan, gövdesi olmayan metotlardır ve alt sınıflar tarafından uygulanmalıdır. Somut metotlar ise normal metotlardır ve zaten bir gövdeye sahiptir.
  Bir sınıf yalnızca bir abstract class'ı miras alabilir, bu nedenle tek kalıtım kullanılabilir.

Interface:

- Interface, arayüz olarak da adlandırılır ve bir sınıfın işlevselliğini tanımlayan, ancak kendisi bir kod bloğu içermeyen yapılardır.
- Interface, yalnızca soyut metotlardan ve sınırsız sayıda sabit değişken (constant) tanımından oluşur. Soyut metotlar, arayüzde tanımlanır ancak gövdeleri yoktur. Bunun yerine, arayüzü uygulayan sınıflar bu metotları kendilerine özgü şekilde doldurmalıdır.
- Bir sınıf birden fazla interface'i uygulayabilir. Bu çoklu arayüz kullanımı, çoklu kalıtım sağlamak için kullanılır.

Özetle, abstract class, kısmen somut metotları içerebilen bir sınıf türüdür ve tek kalıtım kullanılabilirken, interface sadece soyut metotlar ve sabit değişkenler içeren yapılar olup, çoklu kalıtımı destekler. Abstract class, kodun tekrar kullanımı ve ortak işlevselliğin tanımlanması için kullanılırken, interface, farklı sınıfların bir arayüzü uygulayarak benzer işlevselliği sağlamasına ve polimorfizm (çok biçimlilik) özelliğini kullanmasına yardımcı olur.

▼ OOP prensipleri nelerdir?

Nesne Yönelimli Programlama (Object-Oriented Programming - OOP), yazılım geliştirme sürecinde kullanılan bir programlama paradigmasıdır. OOP prensipleri, kodun daha organize, bakımı kolay ve esnek olmasını sağlayan bir dizi temel kavram ve ilkedir. OOP prensipleri, SOLID adı altında birleştirilen beş ana ilke ve daha genel olarak şu şekilde sıralanabilir:

- Nesne Merkezli Programlama (Object-Oriented Programming - OOP): Programlama sürecinde nesneleri temel yapı taşları olarak kullanır. Nesneler, verileri (özellikler) ve bu verilere işlem yapabilen fonksiyonları (metotlar) içeren yapılardır.
- Encapsulation (Kapsülleme): Verilerin ve işlemlerin bir arada gruplandırılmasıdır. Bir nesnenin iç yapısı dışarıdan gizlenerek, erişim belirleyicileri (access modifier'lar) kullanılarak sadece belirli metotlar üzerinden verilere erişim sağlanır. Bu, nesnelerin dışarıya sadece belirli bir arayüz sunmasını ve verilerin korunmasını sağlar.
- Inheritance (Kalıtım): Bir sınıfın, başka bir sınıftan özellikler ve davranışlar miras alabilmesini sağlar. Kalıtım sayesinde, mevcut bir sınıfın özellikleri ve davranışları, yeni bir sınıf tarafından yeniden kullanılabilir veya genişletilebilir.
- Polymorphism (Çok Biçimlilik): Aynı adı taşıyan fakat farklı davranışlar sergileyebilen metotlardır. Bir nesnenin farklı türlerine ait metotlar, aynı arayüz ile çağrılabilir ve nesnenin türüne bağlı olarak farklı işlemler yapabilir.
- Abstraction (Soyutlama): Karmaşık sistemleri ve yapıları basit bir şekilde temsil eder. Sınıfların ve nesnelerin gereksinimlere uygun olarak soyutlanmasını ve sadece önemli bilgilerin dışarıya sunulmasını sağlar.

SOLID prensipleri, OOP prensiplerini daha spesifik hale getiren ve yazılım tasarımının güvenilir ve esnek olmasını sağlayan beş temel ilkedir:

- Single Responsibility Principle (Tek Sorumluluk Prensibi): Bir sınıfın yalnızca bir sorumluluğu olmalıdır.
- Open/Closed Principle (Açık/Kapalı Prensibi): Bir sınıfın davranışları, değişikliğe kapalı (kodu değiştirmeden genişletilebilir) olmalı, ancak yeni davranışlar eklemeye açık olmalıdır.

- Liskov Substitution Principle (Liskov Yerine Geçme Prensibi): Bir sınıf, onun üst sınıfının yerine kullanılabilmelidir.
- Interface Segregation Principle (Arayüz Ayrım Prensibi): Bir sınıf, kendi ihtiyaçlarına uygun ve gereksiz metotları içermeyen birden fazla küçük arayüzler kullanmalıdır.
- Dependency Inversion Principle (Bağımlılığı Tersine Çevirme Prensibi): Bağımlılıklar, soyutlamalar ve somutlamalarla tersine çevrilmelidir, böylece yüksek seviyeli sınıflar düşük seviyeli sınıflara bağlı olmaz.

Bu prensipler, kodun daha iyi yapılandırılmasını, değişikliklere daha esnek bir şekilde adapte olmasını ve daha kolay bakım yapılmasını sağlayarak yazılım tasarımının kalitesini artırır.

▼ Polymorphism nedir? Örnek veriniz.

Polymorphism (Çok Biçimlilik), nesne yönelimli programlamada kullanılan önemli bir kavramdır ve aynı adı taşıyan fakat farklı türe ait metotların, farklı nesnelerin üzerinde çağrılabilmesini ifade eder. Yani, bir metot, birden fazla türdeki nesne tarafından farklı davranışlar sergileyebilir.

Örnek olarak, geometrik şekilleri temsil eden bir uygulama düşünelim. Bu uygulamada "Shape" (Şekil) adında bir ana sınıfımız olsun ve bu sınıftan türeyen "Circle" (Daire) ve "Rectangle" (Dikdörtgen) adında alt sınıflarımız olsun.

```
class Shape {
void draw() {
System.out.println("Bir şekil çizildi.");
}
}

```csharp

```

class Circle extends Shape {
@Override
void draw() {
System.out.println("Bir daire çizildi.");
}
}

class Rectangle extends Shape {
@Override
```

```
void draw() {
System.out.println("Bir dikdörtgen çizildi.");
}
}
```

Şimdi, bu sınıfları kullanarak bir liste oluşturalım ve bu liste içinde hem daire hem de dikdörtgen nesneleri bulunduralım:

```
import java.util.ArrayList;
import java.util.List;

public class Main {
public static void main(String[] args) {
List<Shape> shapes = new ArrayList<>();
shapes.add(new Circle());
shapes.add(new Rectangle());

```
    // Polymorphism ile aynı metot ismi ile farklı davranışlar sergilenebilir.
    for (Shape shape : shapes) {
        shape.draw();
    }
}

```

}
```

Bu örnekte "shapes" listesi içinde hem "Circle" hem de "Rectangle" nesneleri bulunuyor. Ancak, her ikisi de "Shape" sınıfından türetilmiş, bu nedenle "Shape" sınıfındaki "draw" metodu, hem "Circle" hem de "Rectangle" nesneleri için kullanılabilir.

Çıktı:

- Copy code
- Bir daire çizildi.
- Bir dikdörtgen çizildi.

Görüldüğü gibi, "Shape" sınıfında bulunan "draw" metodu, her iki alt sınıf olan "Circle" ve "Rectangle" tarafından farklı şekilde uygulanarak farklı davranışlar sergilenmiştir. Bu, Java dilindeki dinamik bağlama (dynamic binding) mekanizması sayesinde gerçekleşir ve polymorphism özelliğinin gücünü gösterir. Polymorphism sayesinde, uygulamalarda kodun daha esnek ve genişletilebilir olması sağlanır.

▼ SQL constarints türleri nelerdir?

SQL'de constraints (kısıtlamalar), veritabanında saklanan verilerin belli kurallara uymasını sağlayan kural setleridir. Bu kısıtlamalar, veritabanının bütünlüğünü korumak, verilerin tutarlılığını sağlamak ve yanlış veya geçersiz veri girişlerini engellemek için kullanılır. SQL'de kullanılan temel constraints türleri şunlardır:

NOT NULL Constraint: Bu kısıtlama, bir sütunun değerinin NULL olamayacağını belirtir. Yani, bu sütuna her zaman bir değer girmek zorunludur.
Örnek:

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
Email VARCHAR(100) NOT NULL
);
```

UNIQUE Constraint: Bu kısıtlama, bir sütunda bulunan değerlerin benzersiz (unique) olması gerektiğini belirtir. Bir sütunda birden fazla aynı değer olamaz.
Örnek:

```
CREATE TABLE Products (
ProductID INT PRIMARY KEY,
ProductName VARCHAR(100) UNIQUE,
Price DECIMAL(10, 2)
);
```

PRIMARY KEY Constraint: Bu kısıtlama, bir sütunu benzersiz bir kimlik olarak işaretler ve tablodaki her bir satırın tekil olarak tanımlanmasını sağlar. Bir tabloda

yalnızca bir tane PRIMARY KEY olabilir.
Örnek:

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50)
);
```

FOREIGN KEY Constraint: Bu kısıtlama, iki tablo arasındaki ilişkiyi tanımlar. FOREIGN KEY, bir tablodaki sütunun diğer bir tablodaki PRIMARY KEY ile ilişkilendirilmesini sağlar. Böylece, bir tabloda yer alan bir değer, diğer tablodaki bir değeri temsil eder.
Örnek:

```
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
OrderDate DATE,
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

CHECK Constraint: Bu kısıtlama, bir sütunun belirli bir koşulu karşılamasını sağlar. Yani, sütunun değeri belirli bir şartı sağlamalıdır.
Örnek:

```
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT CHECK (Age >= 18)
);
```

SQL constraints'ler, veritabanında tutarlı ve güvenli veri yönetimi için çok önemli araçlardır. Bu kısıtlamalar, veri bütünlüğünü korumak ve hatalı veri girişlerini engellemek için kullanıldığında, veritabanının performansı ve güvenilirliği artırır.

▼ PK, FK, Composite Key kavramları nelerdir?

PK (Primary Key), FK (Foreign Key) ve Composite Key, veritabanı tasarımında kullanılan önemli kavramlardır. İşlevleri ve kullanım amaçları aşağıda açıklanmıştır:

- Primary Key (PK - Birincil Anahtar):
  Primary Key, bir veritabanı tablosundaki her bir satırın benzersiz bir şekilde tanımlanmasını sağlayan bir sütundur. Bir tabloda yalnızca bir adet Primary Key bulunabilir. Veritabanında tutulan verilerin her biri için bir benzersiz anahtar değeri belirlemek ve bu anahtar üzerinden satırlara erişmek için kullanılır. Primary Key, veri bütünlüğünü korumak ve verilerin tutarlılığını sağlamak için önemlidir.
  Örnek:

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Email VARCHAR(100)
);
```

- Foreign Key (FK - Yabancı Anahtar):
  Foreign Key, bir tablodaki sütunun diğer bir tablodaki Primary Key ile ilişkilendirilmesini sağlar. Yani, bir tabloda yer alan bir değer, diğer tablodaki bir değeri temsil eder. Foreign Key kullanımı, tablolar arasında ilişki kurmak ve bu ilişki üzerinden veri alışverişi yapmak için kullanılır. Bir tabloda Foreign Key, diğer bir tablodaki Primary Key ile aynı veri türünde ve boyutta olmalıdır.
  Örnek:

```
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
OrderDate DATE,
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

- Composite Key (Bileşik Anahtar):
  Composite Key, bir tabloda iki veya daha fazla sütunun birleştirilerek oluşturduğu birincil anahtardır. Yani, tek bir sütun değil, birden fazla sütunun kombinasyonu ile oluşturulan anahtardır. Bileşik anahtar, tek bir sütunun tek başına benzersiz olmadığı durumlarda kullanılır.
  Örnek:

  ```
  CREATE TABLE OrderDetails (
  OrderID INT,
  ProductID INT,
  Quantity INT,
  PRIMARY KEY (OrderID, ProductID)
  );
  ```

  Yukarıdaki örnekte "OrderDetails" tablosunda, "OrderID" ve "ProductID" sütunlarından oluşan bir Composite Key tanımlanmıştır. Böylece, her satırın benzersiz bir şekilde tanımlandığı ve aynı ürünün birden fazla siparişte yer alabileceği bir yapı oluşturulmuştur.

PK, FK ve Composite Key, veritabanı tasarımında veri bütünlüğü ve ilişkisel yapıların sağlanmasında önemli rol oynarlar. Doğru ve etkili bir şekilde kullanıldığında, veritabanının performansını artırır ve verilerin güvenilirliğini sağlar.

▼ Katmanlı mimari nedir? Ne amaçla kullanılır? Genel kabul görmüş katmanlar nelerdir, ne iş yaparlar?

▼ Agile-Scrum vs Waterfall hakkında neler biliyorsunuz?

Agile-Scrum ve Waterfall, yazılım geliştirme süreçlerinde kullanılan iki farklı proje yönetimi ve geliştirme metodolojisidir. Her ikisi de farklı yaklaşımlar sunar ve belirli durumlarda tercih edilebilirler. İşte Agile-Scrum ve Waterfall hakkında temel bilgiler:

- Waterfall (Çeşme) Metodolojisi:
  - Waterfall, geleneksel ve lineer bir proje yönetim metodolojisidir. Projeyi aşamalı ve sıralı bir şekilde ilerletir. Her bir aşama, bir sonraki aşamayı başlatmak için tamamlanmalıdır. Sırasıyla şu aşamaları içerir: Gereksinim Analizi, Tasarım, Geliştirme, Test ve Dağıtım.

    Avantajları:

        - Basit ve anlaşılması kolay bir yapıya sahiptir.
          İlerleme takibi ve proje planlaması daha kolaydır.
          Dezavantajları:
        - Esneklik eksikliği: Proje sürecine değişiklikler eklemek zor ve maliyetlidir.
        - Müşteri geri bildirimini geç alır ve sonuçları görmek için proje tamamlanmasını beklemelidir.
        - Risk yönetimi zor olabilir.
- Agile-Scrum:
    - Agile, esnek ve dönüşümsel bir proje yönetim ve geliştirme yaklaşımıdır. Scrum ise Agile yöntemlerden biridir ve projenin belirli aralıklarla iterasyonlar halinde ilerlediği bir yapıya sahiptir.
    - Her iterasyon, bir öncekinden elde edilen geri bildirimlerle şekillenir ve önceliklendirilen iş öğeleri tamamlanır.

      Avantajları:
        - Esneklik: Proje süreci boyunca gereksinimler değişebilir ve projeye yansıtılabilir.
        - Daha sık müşteri geri bildirimi alınır ve projenin gidişatı hızla değiştirilebilir.
        - Takım işbirliğini ve iletişimini teşvik eder.

      Dezavantajları:
        - Tamamen esnek olması, bazı durumlarda daha disipline ve planlamaya ihtiyaç duyulmasına neden olabilir.
        - Proje ilerlemesini bazı organizasyonlar için zor takip edilebilir hale getirebilir.

  Hangi yöntemin kullanılacağı, projenin gereksinimleri, zaman çizelgesi, müşteri ihtiyaçları ve takım yapısına bağlıdır. Küçük ve hızlı değişikliklere ihtiyaç duyulan projeler için Agile-Scrum tercih edilirken, büyük, karmaşık ve sabit gereksinimlere sahip projeler için Waterfall yaklaşımı daha uygun olabilir.

Günümüzde Agile-Scrum, hızla yaygınlaşan ve çeviklik gerektiren yazılım geliştirme projelerinde sıkça kullanılan bir metodoloji haline gelmiştir.

▼ SDLC nedir, detaylıca açıklar mısınız?

SDLC, "Software Development Life Cycle"ın kısaltmasıdır ve yazılım geliştirme sürecini ifade eder. Bu süreç, bir yazılım projesinin planlamadan başlayıp dağıtıma kadar geçen tüm aşamaları içerir. SDLC, yazılımın kaliteli, verimli ve uygun maliyetli bir şekilde geliştirilmesini sağlamak için standart bir yaklaşım sunar. Genellikle aşağıdaki temel adımları içerir:

- Gereksinim Analizi: Proje ekibi, müşteri ve paydaşlarla bir araya gelerek yazılımın gereksinimlerini anlamak için detaylı bir analiz yapar. Gereksinimler, işlevsel ve teknik olarak belirlenir. Bu adım, projenin başarıyla tamamlanması için en önemli aşamalardan biridir.
- Tasarım: Gereksinim analizi sonrasında, yazılımın tasarımı yapılır. Bu aşamada, sistem mimarisi, veri tabloları, arayüzler ve algoritma gibi unsurlar belirlenir. Tasarım aşamasında, gereksinimlere uygun bir plan oluşturulur.
- Geliştirme: Yazılımın kodlama süreci bu aşamada gerçekleşir. Tasarım aşamasında belirlenen plana göre, yazılımın bileşenleri kodlanır. Geliştirme aşamasında, yazılımın modülleri bir araya getirilir ve test edilir.
- Test: Yazılımın hatalardan arındırılması ve işlevselliğinin doğrulanması için testler yapılır. Test aşamasında, yazılımın doğru ve beklenen sonuçları verdiği kontrol edilir. Bu adım, yazılımın güvenilirliği ve performansını sağlamak için kritik öneme sahiptir.
- Dağıtım: Yazılımın canlı ortamda kullanılmak üzere müşteriye teslim edildiği aşamadır. Bu aşamada, yazılımın kullanılabilir hale getirilmesi, kullanıcıların eğitilmesi ve gerektiğinde teknik destek sağlanması gibi süreçler gerçekleştirilir.
- Bakım ve Destek: Yazılımın kullanımda olduğu süre boyunca ihtiyaç duyulan bakım ve güncelleme işlemleri yapılır. Bu aşama, yazılımın sürekli olarak iyileştirilmesi ve güncel tutulmasını sağlar.

SDLC, yazılım geliştirme sürecinin düzenlenmesini ve yönetilmesini sağlayarak, projenin başarıya ulaşmasını ve hedeflenen sonuçların elde edilmesini kolaylaştırır. Her aşama, gerektiğinde geri dönüşler ve düzeltmeler yapılmasına izin vererek, müşteri ihtiyaçlarına daha iyi uyum sağlamayı ve yazılımın kalitesini artırmayı

amaçlar. Bu süreç, proje ekibi ve müşteriler arasında daha iyi bir iletişimi ve işbirliğini teşvik eder.

▼ VievBag vs ViewData vs TempData arasındaki farklar nelerdir?

▼ Session vs Cookie arasındaki farklar nelerdir?

Session ve Cookie, web uygulamalarında kullanılan veri depolama mekanizmalarıdır ve kullanıcıların bilgilerini tutmak için kullanılırlar. Ancak, çalışma prensipleri ve kullanım amaçları farklıdır. İşte Session ve Cookie arasındaki temel farklar:

- Session:
  - Session, sunucu tarafında tutulan verilerdir.
  - Her bir kullanıcı için sunucuda ayrı bir Session oluşturulur ve kullanıcının tarayıcısında bir Session kimliği (Session ID) saklanır.
  - Kullanıcının tarayıcısıyla sunucu arasında Session ID iletişimi yapılır ve sunucu, bu Session ID ile kullanıcının verilerini tanır ve yönetir.
  - Session'lar, varsayılan olarak sunucu belleğinde saklanır. Ancak, özel konfigürasyonlarla sunucu dışında (örneğin veritabanında) da depolanabilir.
  - Güvenlik açısından, kullanıcı bilgilerini tarayıcıda değil, sunucuda tutması nedeniyle Cookie'lere göre daha güvenli kabul edilir.
  - Oturum bazlı işlemler için idealdir. Yani, kullanıcı tarayıcısını kapattığında Session verileri otomatik olarak silinir.
    Örnek Kullanım:

```
// Session'a veri eklemek
Session["Username"] = "john_doe";

// Session'dan veri almak
string username = Session["Username"] as string;
```

- Cookie:
  - Cookie, kullanıcının tarayıcısında saklanan verilerdir.
  - Tarayıcı, web sunucusundan alınan bir Cookie'yi saklar ve aynı web sunucusuna yapılan sonraki isteklerde bu Cookie'yi sunucuyla birlikte

gönderir.

- Cookie'ler, sunucu tarafından tarayıcıya gönderilir ve tarayıcıda belirli bir süre boyunca (belirtilen bir süre veya tarayıcının kapatılmasına kadar) saklanır.
- Genellikle kullanıcı tercihlerini, dil seçeneklerini, oturum kimliklerini (Session ID gibi) veya reklam verilerini saklamak için kullanılırlar.
- Güvenlik açısından, tarayıcıda saklanmaları nedeniyle Session'lara göre daha az güvenli kabul edilir. Hassas verileri doğrudan Cookie'lere koymak, güvenlik açığı oluşturabilir.
  Örnek Kullanım:

```
// Cookie'ye veri eklemek
HttpCookie cookie = new HttpCookie("Username", "john_doe");
cookie.Expires = DateTime.Now.AddDays(7);
Response.Cookies.Add(cookie);

// Cookie'den veri almak
HttpCookie cookie = Request.Cookies["Username"];
string username = cookie?.Value;
```

Hangi veri depolama mekanizmasının kullanılacağı, kullanıcı bilgilerinin türüne, güvenlik ihtiyacına ve işlevselliğe bağlı olarak belirlenmelidir. Kullanıcının tarayıcısında saklanması gereken veriler için Cookie, sunucuda tutulması gereken veya oturum bazlı işlemler için Session tercih edilebilir.

▼ Eğitimde neler öğrendiniz?

▼ Kendinizde en başarılı gördüğünüz konular hangileri?

▼ .Net Forms neden kullanılır, bu konuda neleri öğrendiniz?

.NET Forms, Microsoft tarafından geliştirilen bir yazılım geliştirme teknolojisidir ve Windows tabanlı masaüstü uygulamaları oluşturmak için kullanılır. Bu teknoloji, Windows Presentation Foundation (WPF) ve Universal Windows Platform (UWP) gibi diğer modern Windows uygulama geliştirme teknolojilerinin öncesi olarak düşünülebilir.

.NET Forms, özellikle Windows işletim sistemine sahip bilgisayarlar için kullanıcı arayüzüne sahip uygulamalar geliştirmek isteyen geliştiriciler için popüler bir tercih

olmuştur. Bazı temel özellikleri ve nedenleri şunlardır:

- Kolaylık ve Hızlı Geliştirme: .NET Forms, görsel bir tasarım aracı olan Visual Studio gibi gelişmiş bir entegre geliştirme ortamıyla birlikte gelir. Bu sayede, görsel arayüzlerin tasarımı ve uygulama kodunun yazımı daha hızlı ve kolay bir şekilde yapılabilir.
- Zengin Kontrol Kütüphanesi: .NET Forms, birçok kullanıcı arabirimi kontrolü (button, textbox, label, datagridview vb.) içeren zengin bir kütüphaneye sahiptir. Bu kontrol kütüphanesi, farklı özelliklerle donatılmış uygulamaların kolayca geliştirilmesini sağlar.
- Geriye Dönük Uyumluluk: .NET Forms, birçok eski .NET uygulaması tarafından kullanılmış ve geliştirilmiştir. Bu nedenle, mevcut uygulamaların bakımı veya geliştirilmesi için hala kullanılmaya devam edilmektedir.
- Offline Uygulamalar: .NET Forms, internet bağlantısı olmadan çalışan masaüstü uygulamalarının geliştirilmesi için ideal bir seçenektir.

Öğrendiğim bazı temel noktalar şunlardır:

- .NET Forms'un Windows tabanlı masaüstü uygulamaları için kullanıcı arayüzü geliştirme yeteneklerini içerdiği ve hızlı bir şekilde uygulama geliştirme sağladığı.
- Geliştirme için Visual Studio IDE'nin tercih edildiği.
- Mevcut Windows uygulamalarının bakımı veya geliştirilmesi için hala kullanıldığı.
- Ancak, modern uygulamalar için Windows Presentation Foundation (WPF) veya Universal Windows Platform (UWP) gibi teknolojilerin daha tercih edilebilir olduğu, çünkü daha fazla özellik ve esneklik sunarlar.

▼ Value Type vs Referans Type arasındaki farklar nelerdir?

Value Type ve Referans Type (Reference Type), C# ve benzeri programlama dillerinde verilerin bellekte nasıl saklandığına bağlı olarak iki farklı türdür. İşte bu iki tür arasındaki temel farklar:

- Bellek Saklama Yöntemi:

    - Value Type: Value Type veriler, değerleri doğrudan belleğin yığın (stack) bölgesinde saklanır. Bu nedenle, değer türü verilerin değerleri doğrudan bellekte saklanır.
      Referans Type: Referans Type verilerin bellekte kendileri değil, verilerin bulunduğu bellek adresleri (referansları) saklanır. Bu nedenle, referans türü verilerin değerleri, bellekte verilere gösteren referanslarla belirtilir.

- Davranış ve Kopyalama:
    - Value Type: Value Type veriler, başka bir değişkene atandığında, değerleri bağımsız olarak kopyalanır. Yani, değer tipi değişkenler arasında yapılan atamalar, birbirlerinden bağımsız kopyalar oluşturur.
    - Referans Type: Referans Type veriler, başka bir değişkene atandığında, verilerin referansları (bellek adresleri) kopyalanır. Yani, referans türü değişkenler arasında yapılan atamalar, aynı bellek bloğunu paylaşan farklı isimler oluşturur. Bu nedenle, bir değişken değiştirildiğinde, diğer değişkenler de etkilenir.
- İkilik Karşılaştırma:
    - Value Type: Value Type verilerin eşitliği, değerlerine göre karşılaştırılır. İki değer aynı olduğunda, değer tipli değişkenler eşittir.
    - Referans Type: Referans Type verilerin eşitliği, bellek adreslerine göre karşılaştırılır. Yani, iki referans tipi değişkeni, aynı bellek adresine işaret ediyorsa eşittir. Ancak veriler aynı olsa bile, farklı bellek adreslerine işaret eden referans türü değişkenleri eşit değildir.
    - Value Type örnekleri: int, float, double, bool, char, struct vb.
      Referans Type örnekleri: class, string, object, array vb.

  Özetle, Value Type veriler, değerleri doğrudan bellekte saklanır ve kopyalanırken bağımsız kopyalar oluşturur. Referans Type veriler ise bellekte verilerin adresleri saklanır ve kopyalanırken aynı bellek bloğunu paylaşan farklı isimler oluşturur. Bu farklar, programlama dillerinde verilerin yönetimi ve davranışı üzerinde önemli etkilere sahiptir.

▼ Garbage Collector ne işe yarar?

Garbage Collector (Çöp Toplayıcı), programlama dillerinde, özellikle .NET platformunda ve Java gibi dillerde, bellek yönetiminin otomatik olarak yapılmasını

sağlayan bir mekanizmadır. Bu mekanizma, uygulamaların bellekte kullanılmayan ve artık referansı bulunmayan nesneleri tespit ederek ve bellekten kaldırarak, bellek sızıntılarını önler ve bellek kullanımını optimize eder.

▼ RAM'in çalışma prensibi nedir?

RAM (Random Access Memory), bilgisayarın geçici veri depolama birimidir ve çalışma belleği olarak da bilinir. RAM, bilgisayara yüklenen programların ve verilerin anlık olarak işlem için tutulduğu bir hafıza türüdür. Bilgisayarın işlem gücüne ve performansına büyük ölçüde etki eder.
RAM, bilgisayarın performansı için kritik bir rol oynar. Yüksek kapasiteli ve hızlı RAM, daha fazla programı ve veriyi aynı anda çalıştırmak ve işlemek için bilgisayara daha fazla güç sağlar. Bu nedenle, daha karmaşık ve yoğun bilgi işlem ihtiyaçları olan uygulamalar, büyük miktarda RAM kullanımını gerektirir.

▼ Stack– Heap kavramları nedir?

Stack ve Heap, programlama dillerinde bellek yönetimi için kullanılan iki önemli bellek bölgesidir. Bu iki bölge, farklı veri türlerinin ve değişkenlerin nasıl bellekte saklandığına ve erişildiğine bağlı olarak çalışır.

- Stack (Yığın):
  - Stack, bir veri yapısıdır ve bilgisayarın çalışma belleğinin bir parçasıdır.
  - Stack, belleğin en hızlı bölgesidir ve doğrudan işlemci tarafından yönetilir.
  - Stack, otomatik bellek yönetimi kullanır ve değişkenlerin hayat döngüsü (scope) ile doğrudan ilişkilidir.
  - Yerel değişkenler (local variables), metot çağrıları (method calls) ve fonksiyon parametreleri gibi kısa ömürlü veriler stack belleğinde saklanır.
  - Yeni bir metot çağrıldığında, o metoda ait değişkenler ve çalışma bilgileri (geri dönüş adresi, geçici değerler vb.) stack belleğinde oluşturulur.
  - Stack, Last In First Out (LIFO) yani son giren, ilk çıkan mantığı ile çalışır.
- Heap (Öbek):
  - Heap, programın çalışma zamanında dinamik olarak oluşturulan verilerin (nesnelerin) ve veri yapılarının saklandığı bellek bölgesidir.

- Heap, bilgisayarın çalışma belleğinin daha büyük bir bölgesidir ve işlemci tarafından doğrudan yönetilmez.
- Heap belleği, manuel olarak yönetilmelidir (örneğin, bellek tahsisi ve bellek serbest bırakma işlemleri).
- Uzun ömürlü veriler (global variables, static variables) ve dinamik olarak oluşturulan nesneler heap belleğinde saklanır.
- Programın çalışma zamanında, new anahtar kelimesiyle dinamik olarak oluşturulan nesneler heap belleğinde yer kaplar ve bellek serbest bırakılmadan kalırlar.
- Heap, veri bloklarının herhangi bir sıra ile yerleştirilebileceği bir alandır ve bu nedenle işlemci tarafından daha yavaş erişilir.

Özetle, Stack belleği kısa ömürlü ve otomatik olarak yönetilen verileri saklarken, Heap belleği uzun ömürlü ve manuel olarak yönetilen dinamik verileri saklar. Programlama dillerinde, değişkenlerin, nesnelerin ve veri yapılarının doğru bellek bölgesine yerleştirilmesi ve yönetilmesi önemlidir, çünkü yanlış kullanım bellek sızıntılarına (memory leaks) veya program hatalarına neden olabilir.

▼ Yazılım yaparken nelere dikkat edersin?

Yazılım geliştirme metodolojileri, bir yazılım projesini planlama, tasarım, uygulama ve dağıtma süreçlerini organize etmek ve yönetmek için kullanılan çeşitli yaklaşımları ifade eder. Bu metodolojiler, yazılım geliştirme sürecini daha verimli, düzenli ve başarılı bir şekilde tamamlamayı amaçlar. İşte yaygın olarak kullanılan bazı yazılım geliştirme metodolojileri:

- Waterfall (Şelale):
  - Lineer bir yaklaşımdır ve aşamalar sırasıyla işlenir.
  - Her aşama tamamlandıktan sonra bir sonraki aşamaya geçilir.
  - Geri dönülmez bir süreçtir, değişiklikler zor ve maliyetli olabilir.
  - Küçük ve basit projelerde tercih edilir.
- Agile:
  - Esnek ve sürekli gelişim odaklı bir yaklaşımdır.
  - İteratif ve inkremental geliştirme yapar, kısa süreli döngüler halinde çalışır.

    - Müşteri geri bildirimine önem verir, değişikliklere açıktır.
    - Scrum, Kanban gibi çeşitli Agile yöntemleri vardır.
- Scrum:
    - Ekiplerin işbirliğini ve etkili yönetimi sağlamak için kullanılır.
    - Süreçleri kısa süreli Sprint'lerle (genellikle 2-4 hafta) yönetir.
    - Sprint başında hedef belirlenir, Sprint sonunda işlevsel bir ürün parçası elde edilir.
- Kanban:
    - Görsel bir yönetim sistemi kullanır ve iş akışını izlemek için tahta panolar kullanır.
    - İş öğeleri, görsel kartlarda takip edilir ve iş akışı sırasında kolayca değiştirilebilir.
- Lean:
    - İşlem verimliliği ve müşteri değeri odaklı bir yaklaşımdır.
    - Atıl kaynakları azaltmayı ve iş süreçlerini optimize etmeyi hedefler.
- Spiral:
    - Risk odaklı bir yaklaşımdır ve proje aşamaları döngüsel olarak tekrarlanır.
    - Her döngüde yeni özellikler eklenir ve riskler değerlendirilir.
- XP (Extreme Programming):
    - Yazılımın kalitesini artırmak ve müşteri geri bildirimine önem vermek için kullanılır.
    - Pair programming, sıkı test yazımı gibi uygulamaları içerir.
- DevOps:
    - Yazılım geliştirme ve işletme süreçlerinin entegrasyonunu sağlamak için kullanılır.
    - Uygulamanın hızlı bir şekilde dağıtımını ve güncellemelerin hızlı bir şekilde yapılmasını hedefler.

▼ Yazılım geliştirme metodolojileri nelerdir?

▼ Kendini güncel tutmak için hangi kaynakları ve kişileri takip ediyorsun?

▼ En son hangi makaleyi, kitabı okudun?

▼ En son hangi problemle karşılaştın?

▼ SP neden kullanırız?

Stored Procedure (SP), veritabanında önceden tanımlanmış ve derlenmiş bir SQL kod bloğudur. SP'ler, veritabanı yönetim sistemlerinde (DBMS) saklanır ve uygulama tarafından çağrılarak çalıştırılırlar. SP'ler, çeşitli nedenlerle kullanılır:

- Performans ve Optimizasyon: SP'ler, veritabanı tarafında derlenmiş olduğundan, sorguların performansını artırır. Aynı sorguların tekrar tekrar çalıştırılması yerine, SP'lerin çağrılması, sorgu maliyetini azaltır ve veritabanı üzerinde yükü azaltır.
- Güvenlik: SP'ler, veritabanına erişim için yetkilendirme ve güvenlik kontrolleri sağlar. Uygulamaların doğrudan tablolara erişmesine izin vermek yerine, SP'ler üzerinden güvenli bir arayüz sağlayarak, güvenlik açıklarını azaltır.
- Veritabanı Mantığı: SP'ler, veritabanı üzerinde kompleks işlemleri gerçekleştirmek için kullanılabilir. Birden çok tabloya veya veri kaynağına erişim gerektiren işlemler, SP'lerin yardımı ile veritabanı üzerinde yönetilebilir.
- Tekrar Kullanılabilirlik: SP'ler, farklı uygulamalar ve sorgular tarafından tekrar tekrar kullanılabilir. Bu, kod tekrarını azaltır ve bakımı kolaylaştırır.
- Veri Bütünlüğü: SP'ler, işlem mantığını veritabanı seviyesinde gerçekleştirerek, veri bütünlüğünü sağlar. Birden çok tabloya etkileyen işlemlerde, veri bütünlüğünün korunmasına yardımcı olur.
- Transaktif İşlemler: SP'ler, transaktif işlemleri destekler. Birden çok sorgunun bir arada çalıştığı, tüm işlemlerin başarılı olması veya hiçbirinin yapılmaması gibi durumlarda, SP'lerin kullanılması önemlidir.
- Veri İzleme ve Raporlama: SP'ler, belirli raporlar veya veri istatistikleri üretmek için kullanılabilir. Birden çok işlemi bir araya getirerek, karmaşık raporların üretilmesine yardımcı olur.

SP'ler, veritabanı yönetim sistemlerine özgüdür ve çeşitli avantajlar sunar. Ancak, her durumda SP kullanımı uygun olmayabilir. Performans, güvenlik ve kod karmaşıklığı gibi faktörler göz önünde bulundurularak, SP kullanımının doğru bir şekilde planlanması ve tasarlanması önemlidir.

▼ Readonly keyword’unu nerede kullanırız?

C# programlama dilinde readonly keyword'ünü kullanarak, salt okunur (değiştirilemez) alanlar (fields) tanımlayabiliriz. Bir readonly alan, sadece tanımlandığı yerde veya nesne oluşturulurken başlatılabilir ve daha sonra değeri değiştirilemez. Bu keyword, özellikle sabit değerlerin tanımlanması veya nesnenin yapısı içinde değişmeyecek değerlerin kullanılması gerektiği durumlarda kullanışlıdır.

- readonly keyword'ü kullanılan alanların önemli özellikleri şunlardır:
  - Tanımlandıkları yerde veya sınıfın constructor'ında başlatılabilirler.
  - Değeri sadece başlatma sırasında değiştirilebilir ve sonrasında değiştirilemez.
  - readonly alanların değeri, tanımlandığı sınıf içerisinde veya constructor'larda değiştirilebilir, ancak başka bir method veya property içerisinde değiştirilemez.
  - Sabit değerler için const keyword'ü kullanılabilirken, readonly alanlar runtime sırasında başlatılabilir, dolayısıyla dinamik olarak hesaplanan değerler için daha uygundur.
    İşte bir örnek:

```
public class MyClass
{
public readonly int MyReadOnlyField;

```
```
public MyClass(int value)
{
    MyReadOnlyField = value; // Sadece constructor'da değeri atanabilir.
}

public void SomeMethod()
{
    // MyReadOnlyField = 10; // Hata! Method içinde değeri değiştirilemez.
```

```
}
}
```

▼ Static vs Static olmayan metodlar arasındaki farklar nelerdir?

Static ve static olmayan (instance) metodlar arasındaki farklar şunlardır:

- Erişim ve Çağrı:
  - Static Metodlar: Static metodlar, sınıfın bir üyesi olmaktan ziyade sınıfın kendisine aittir. Sınıfın nesnesi oluşturulmasına gerek kalmadan doğrudan sınıf adıyla çağrılabilirler.
    Static Olmayan Metodlar: Static olmayan (instance) metodlar, sınıfın örneği (nesnesi) üzerinden çağrılır. Bu nedenle, önce bir nesne oluşturmak ve o nesne üzerinden metodları çağırmak gerekir.
- Bellek Kullanımı:
  - Static Metodlar: Static metodlar, sınıfın örneğine bağlı olmadıkları için nesneye ait bir bellek alanını kullanmazlar. Bellekte yalnızca bir kez tanımlanır ve tüm nesneler tarafından paylaşılırlar.
  - Static Olmayan Metodlar: Her örnek (nesne) için ayrı bir bellek alanı kullanırlar. Bu nedenle, her örnek için aynı metodun bir kopyası bellekte saklanır.
- Erişilebilirlik:
  - Static Metodlar: Static metodlar, sınıfın içindeki diğer static ve static olmayan üyeleri doğrudan erişebilirler.
  - Static Olmayan Metodlar: Static olmayan metodlar, sınıfın içindeki static ve static olmayan üyeleri doğrudan erişebilirler. Ayrıca, static olmayan metodlar bir nesne üzerinden çağrıldıkları için nesne üzerinde tanımlı olan alanlara ve diğer metodlara erişebilirler.
- this Anahtar Kelimesi:
  - Static Metodlar: Static metodlar içinde this anahtar kelimesi kullanılamaz, çünkü nesne bağlamına sahip değillerdir.

    - Static Olmayan Metodlar: Static olmayan metodlar içinde this anahtar kelimesi kullanılabilir. this, o anki örneğin (nesnenin) referansını ifade eder.

  Hangi tür metodu kullanacağınız, özellikle metodu kullanacağınız senaryoya ve veri paylaşımına bağlı olacaktır. Static metodlar genellikle yardımcı fonksiyonlar veya yardımcı işlemler için kullanılırken, static olmayan metodlar nesnenin durumuyla ilgili işlemler için tercih edilir.

▼ Sıralama algoritmaları hangileridir?

  Sıralama algoritmaları, bir dizi veya liste içindeki öğeleri belirli bir düzene göre sıralamak için kullanılan algoritmalardır. Temel sıralama algoritmaları şunlardır:

  - Bubble Sort (Kabarcık Sıralama): Komşu öğelerin karşılaştırılarak yer değiştirdiği basit bir sıralama algoritmasıdır. Veri seti küçük olduğunda etkili olabilir, ancak büyük listelerde performansı düşüktür.
  - Selection Sort (Seçim Sıralama): Dizideki en küçük elemanı bulup dizinin başına yerleştirir ve sıralanan bölümü genişletir. Basit ve anlaşılır bir algoritmadır, ancak büyük veri kümesi için verimsizdir.
  - Insertion Sort (Ekleme Sıralama): Listenin sıralı kısmına bir eleman ekleyerek ve doğru konumunu bulana kadar yer değiştirerek çalışır. Küçük veri kümesi için iyi bir performansa sahiptir.
  - Merge Sort (Birleştirme Sıralama): Birleştirme (merge) adımı kullanarak veri kümesini ikiye böler, her alt listeyi sıralar ve sonunda iki sıralı alt liste birleştirilir. Kararlı ve veri kümesi büyüklüğüne göre etkili bir algoritmadır.
  - Quick Sort (Hızlı Sıralama): Pivot elemanı seçip bu elemandan küçük olanları solunda, büyük olanları sağında olacak şekilde veri kümesini bölerek çalışır. Alt listeleri sıralayarak işlemi rekürsif olarak gerçekleştirir. Ortalama durumda hızlı ve etkili bir algoritmadır.
  - Heap Sort (Yığın Sıralama): Yığın veri yapısını kullanarak sıralama işlemini gerçekleştirir. Büyük veri kümesi için etkili bir algoritmadır ancak diğerlerine göre daha karmaşıktır.
  - Counting Sort (Sayma Sıralama): Sadece pozitif tam sayılardan oluşan veri kümesi için uygun olan bir sıralama algoritmasıdır. Elemanların tekrar sayısını kullanarak sıralamayı gerçekleştirir.

- Radix Sort (Radyal Sıralama): Verileri basamaklarını kullanarak sıralar. Genellikle çok büyük sayılar veya sıralanacak veri setinde farklı uzunluktaki anahtarlar bulunan durumlarda tercih edilir.

Bu sıralama algoritmaları arasında tercih, sıralanacak veri kümesinin büyüklüğüne, tipine ve verimlilik gereksinimlerine bağlı olarak yapılmalıdır. Büyük veri kümeleri için daha hızlı algoritmalar tercih edilirken, küçük veri kümeleri için daha basit ve anlaşılır algoritmalar yeterli olabilir.

▼ Hangi veri tiplerini kullandınız?

C# programlama dilinde yaygın olarak kullanılan veri tipleri şunlardır:

- Tam Sayı Veri Tipleri:
  - int: 32 bitlik işaretli tam sayılar. Örneğin: 10, -5, 0.
  - uint: 32 bitlik işaretsiz tam sayılar. Örneğin: 100, 0.
- Ondalık Sayı Veri Tipleri:
  - double: 64 bitlik çift hassasiyetli kayan noktalı sayılar. Örneğin: 3.14, -2.5.
  - float: 32 bitlik çift hassasiyetli kayan noktalı sayılar. Örneğin: 1.5f, -0.25f.
  - decimal: 128 bitlik ondalık sayılar. Finansal hesaplamalarda daha yüksek hassasiyet gerektiğinde kullanılır.
- Karakter Veri Tipleri:
  - char: 16 bitlik Unicode karakterler. Örneğin: 'A', 'b', '1'.
    Metin Veri Tipi:
  - string: Metin veya karakter dizilerini temsil eden veri tipi. Örneğin: "Hello, World!", "C# Programming".
    Mantıksal Veri Tipi:
  - bool: Sadece iki değer alabilen (true veya false) mantıksal veri tipi.
    Tarih ve Saat Veri Tipi:
  - DateTime: Tarih ve saat bilgisini temsil eden veri tipi.
    Dizi Veri Tipi:
  - int[], string[], vb.: Birden çok öğeyi aynı veri tipi altında saklamak için kullanılan veri yapısı.

Nesne Veri Tipi:

- object: Her türlü veriyi temsil edebilen en genel veri tipi.
  Enum Veri Tipi:
- enum: Bir dizi adlandırılmış sabit değeri temsil eden veri tipi.
  Yapı Veri Tipi:
- struct: Birden fazla veri üyesini tek bir yapı altında gruplamak için kullanılan veri tipi.
  Bu veri tipleri, C# dilinde kullanılan temel veri tiplerindendir ve C# programlamada verileri tanımlamak ve işlemek için kullanılırlar. Ayrıca, C# dilinde kullanıcı tarafından tanımlanabilen yapısal veri tipleri ve sınıf türleri de mevcuttur.

▼ Kuyruk yapıları hakkında neler biliyorsunuz?

Kuyruk (Queue), verilerin ilk giren ilk çıkar (First-In-First-Out - FIFO) mantığına göre sıralandığı bir veri yapısıdır. Bu yapı, elemanların sırayla eklenip çıkarıldığı bir sıra oluşturur. En eski eleman, kuyruğun önünden (başından) çıkarılırken, yeni elemanlar kuyruğun sonuna eklenir.

Kuyruk veri yapısının temel özellikleri şunlardır:

- Enqueue: Yeni bir elemanın kuyruğun sonuna eklenmesi işlemidir.
- Dequeue: Kuyruğun başındaki elemanın çıkarılması işlemidir.
- Peek: Kuyruğun başındaki elemana erişmek, ancak onu kuyruktan çıkarmadan görmek için kullanılır.
- Count: Kuyruktaki eleman sayısını verir.

Kuyruk veri yapısı, işlem sırasını yönetmek veya iş parçacıklarını düzenlemek gibi çeşitli senaryolarda kullanışlıdır. Örneğin, işlemci kuyrukları, işlem gönderme sistemleri, yazılım mühendisliğinde kullanılan iş parçacığı havuzları gibi durumlarda kuyrukların kullanımı yaygındır.

Kuyruk veri yapısı, verilerin eklenme ve çıkarma işlemlerinin hızlı olduğu ve FIFO sıralamasının önemli olduğu senaryolarda tercih edilir. Bu tür veri yapıları, verilerin düzenli ve sıralı bir şekilde işlenmesini sağlar.

▼ object reference not set to an instance of an object hatası neden alınır, nasıl çözeriz?

"Object reference not set to an instance of an object" hatası, C# veya diğer programlama dillerinde sıkça karşılaşılan bir hata türüdür. Bu hata, bir nesnenin başlatılmadan veya null olarak kabul edildiğinde, bu nesneye erişilmeye çalışıldığında ortaya çıkar.

Bu hatanın nedenleri şunlar olabilir:

- Null Referans: Bir nesne başlatılmadan veya null olarak atandıktan sonra, bu nesneye erişilmeye çalışıldığında hata oluşur. Örneğin:

```
string name = null;
Console.WriteLine(name.Length); // "Object reference not set to an instance of an obj
ect" hatası alınır.
```

Nesne Başlatılmamış: Bir nesne henüz başlatılmadan ona erişmeye çalışılırsa, bu hatayı alırsınız.

```
MyClass obj;
Console.WriteLine(obj.SomeProperty); // "Object reference not set to an instance of a
n object" hatası alınır.
```

Nesne İlkelleştirilmemiş: Nesne örneğinin bir yapılandırıcı veya başlatıcı ile başlatılmamış olması durumunda bu hata alınır.
Hatanın çözümü için aşağıdaki adımları takip edebilirsiniz:

Nesneyi Başlatın: Nesne örneğini yapıcı metot veya nesne başlatma ifadesiyle başlatın.

```
MyClass obj = new MyClass(); // MyClass sınıfınızın yapılandırıcısını çağırın.
```

Null Kontrolü: Eğer bir nesne null olabilecekse, ona erişmeden önce null kontrolü yapın.

```
if (obj != null)
{
Console.WriteLine(obj.SomeProperty);
}
```

Default Değerler: Değişkenleri veya nesneleri tanımlarken, varsayılan değerlerle başlatın.

```
string name = ""; // veya null yerine varsayılan bir değer atayın.
```

Hatanın kaynağına bağlı olarak bu adımları takip ederek "Object reference not set to an instance of an object" hatasını çözebilirsiniz. Özellikle null referanslara dikkat ederek ve nesneleri doğru şekilde başlatarak bu tür hataları önlemek önemlidir.

▼ Design pattern’lerden hangilerini biliyorsunuz?

- Singleton Pattern: Sadece bir örneği (instance) olan bir sınıfın oluşturulmasını sağlar ve bu örneğe global erişim imkanı sunar.
- Factory Pattern: Nesnelerin oluşturulmasını merkezi bir fabrika sınıfına devreder ve istemciye oluşturulmuş nesneleri döndürür.
- Observer Pattern: Nesneler arasında bağımlılıkları minimize ederek, bir nesnenin durumunda değişiklik olduğunda diğer nesnelerin buna tepki göstermesini sağlar.
- Strategy Pattern: Bir işlemi yapacak olan algoritmayı soyutlar ve çalışma zamanında farklı algoritmaların kullanılmasına olanak tanır.
- Decorator Pattern: Bir nesneyi sarmalayarak, dinamik olarak ek özellikler eklemeyi sağlar.
- Adapter Pattern: Bir nesnenin arabirimini farklı bir arabirime dönüştürmeyi sağlar, böylece uyumsuz sınıfların birlikte çalışmasını sağlar.
- Proxy Pattern: Gerçek nesneye olan erişimi kontrol ederek, onun yerine geçerek veya işlemleri yönlendirerek davranışı değiştirir.
- Template Method Pattern: Bir işlemin adımlarını soyutlar ve alt sınıfların bu adımları uygulayarak işlemi tamamlamasını sağlar.
- Composite Pattern: Nesneleri hiyerarşik bir ağaç yapısında organize ederek, tek bir nesne gibi davranmalarını sağlar.
- Command Pattern: İstemci ile bir komutu içeren nesneler arasında bağlantı kurar ve istemcinin bir komutu yürütmesini sağlar.

- Iterator Pattern: Bir koleksiyonun içindeki elemanlara sırasıyla erişmek için bir arabirim sağlar.
- Observer Pattern: Nesneler arasındaki bağımlılıkları minimize ederek, bir nesnenin durumunda değişiklik olduğunda diğer nesnelerin buna tepki göstermesini sağlar.

Bu, yalnızca bazı tasarım desenlerinin bir listesidir ve daha fazla tasarım deseni bulunmaktadır. Tasarım desenleri, yazılım geliştirme süreçlerinde tekrarlanan problemleri çözmek ve daha esnek ve sürdürülebilir yazılımlar oluşturmak için kullanılırlar.

▼ EF- Bir tabloya index neden tanımlanır, PK neden kullanırız?

EF (Entity Framework), bir ORM (Object-Relational Mapping) aracıdır ve veritabanı işlemlerini kolaylaştırmak için kullanılır. Bir tabloya index tanımlamak ve PK (Primary Key) kullanmak, veritabanı performansını artırmak ve verilere erişimi hızlandırmak için önemli rol oynar.

- Index (Dizin):
  - Bir veritabanı tablosundaki sütunlara veya sütun kombinasyonlarına veri erişimini hızlandırmak için oluşturulan bir yapıdır.
  - Index, verilerin fiziksel olarak sıralanmasını sağlar, bu da belirli bir sütuna veya sütun kombinasyonuna dayalı sorgularda performans artışı sağlar.
  - Index, bir tür veri yapısıdır ve veri tabanındaki sorguların daha hızlı çalışmasına yardımcı olur.
- Primary Key (PK):
  - Bir tablodaki benzersiz tanımlayıcıdır. Her tabloda yalnızca bir tane olabilir.
  - PK, tablodaki her kaydın benzersiz bir şekilde tanımlanmasını sağlar ve bir nesneye veya veriye benzersiz bir kimlik atar.
  - EF ve diğer ORM araçları, veri tabanındaki bir tablodaki nesneleri temsil ederken PK kullanır. Bu sayede her nesneye erişim ve güncelleme işlemleri için benzersiz bir tanımlayıcıya sahip olunur.
- Neden Index Kullanırız?

    - Veri tabanı tablolarında sık kullanılan sorguları hızlandırmak için index kullanılır. Özellikle büyük veri tablolarında sorgu performansı artırılır.
    - Sık sık sorgulanan sütunlarda index kullanarak, veri erişim hızını ve sorgu yanıtlarını optimize ederiz.
    - Index, belirli sorguların daha hızlı çalışmasını sağlar, ancak tabloya veri ekleme, güncelleme ve silme işlemlerinin biraz daha yavaş olmasına neden olabilir. Bu nedenle index oluştururken, tablonun kullanım senaryosuna göre dikkatli seçim yapmak önemlidir.
- Neden Primary Key Kullanırız?
    - Veri tabanında benzersiz tanımlama ve referans için PK kullanılır.
    - PK, bir nesnenin veya verinin veri tabanında tek bir kaydı temsil etmesini sağlar. Bu, verilere erişimi kolaylaştırır ve çakışmaları önler.
    - EF ve diğer ORM araçları, PK kullanarak veritabanındaki nesnelerin tekil kimliklerini belirler ve nesneler arasındaki ilişkileri tanımlar. Bu, veritabanı ilişkilerini yönetmeyi kolaylaştırır ve ilişkisel verileri daha etkin bir şekilde kullanmamıza olanak tanır.

▼ Index– PK arasındaki fark nedir?

- Index (Dizin):
    - Bir veritabanı tablosunda bir veya daha fazla sütuna veri erişimini hızlandırmak için oluşturulan bir yapıdır.
    - Tablodaki verilerin fiziksel olarak sıralanmasını ve hızlı arama işlemleri yapılmasını sağlar.
    - Tablodaki bir veya birden fazla sütuna index oluşturulabilir ve indexlerin amacı sorgu performansını artırmaktır.
    - Index, bir veri yapısıdır ve veri tabanındaki sorguların daha hızlı çalışmasına yardımcı olur.
    - Index, verilerin hızlı erişimi için yardımcı bir mekanizmadır ancak her tabloda olması zorunlu değildir.
- Primary Key (PK):

- Bir tablodaki her kaydın benzersiz bir şekilde tanımlanmasını sağlayan bir sütundur.
- Yalnızca bir PK olabilir ve her tabloda en az bir PK tanımlanmalıdır.
- PK, veri tabanındaki bir tablodaki nesneleri temsil ederken kullanılır ve her nesnenin benzersiz bir kimliği olmasını sağlar.
- PK, bir nesnenin veri tabanında tek bir kaydı temsil etmesini sağlar ve verilere erişimi kolaylaştırır.
- PK, bir mantık ve veri bütünlüğü unsuru olarak kullanılır ve veritabanındaki diğer tablolarla ilişkilendirme yapmak için referans olarak kullanılabilir.
- Özetle, index bir veri yapısıdır ve veri tabanındaki sorguların performansını artırmak için kullanılırken, PK bir mantık ve veri bütünlüğü unsuru olarak bir nesnenin benzersiz kimliğini belirlemek için kullanılır. Her tabloda yalnızca bir PK olabilirken, bir tabloda birden fazla index oluşturulabilir. PK'nın bir tür index olduğu düşünülebilir, ancak index ve PK farklı amaçlar için kullanılır.

▼ JS değişken tanımlarken let- var-const farkı nedir?

JavaScript'de değişken tanımlarken let, var, ve const anahtar kelimeleri farklı amaçlar için kullanılır ve aralarında bazı önemli farklar vardır:

- var:
  - ES5 (ECMAScript 5) ve öncesinde kullanılan değişken tanımlama anahtar kelimesidir.
  - Fonksiyon kapsamında (function scope) çalışır. Yani, tanımlanan bir değişken sadece içinde tanımlandığı fonksiyon içerisinde geçerlidir.
  - Eğer bir değişken, fonksiyon içinde var ile tanımlanmışsa, o değişken dışarıdan erişilemez.
  - Özellikle modern JavaScript kullanımında, let ve const anahtar kelimeleri tercih edilmektedir
- let:
  - ES6 (ECMAScript 2015) ile tanıtılan blok kapsamında (block scope) çalışan bir değişken tanımlama anahtar kelimesidir.

    - Bir blok içinde (örn. if, for, while) tanımlanan değişkenler, sadece o blok içerisinde geçerlidir ve dışarıdan erişilemez.
    - Aynı isimle tekrar tanımlama yapılabilir, ancak aynı kapsamdaki aynı isimli değişkenler arasında kavram kirliliği (hoisting) yaşanmaz.
- const:
    - ES6 ile tanıtılan, değeri bir kez atandıktan sonra değiştirilemeyen (sabit) bir değişken tanımlama anahtar kelimesidir.
    - const ile tanımlanan bir değişkenin değeri başlangıçta atandıktan sonra değiştirilemez.
    - Ancak, eğer bir nesne veya dizi ise, nesne veya dizi içindeki özellikler değiştirilebilir.

  Genel olarak, modern JavaScript projelerinde let ve const anahtar kelimeleri tercih edilmektedir. var anahtar kelimesi, işlevsel kapsamı ve kavram kirliliği gibi bazı sorunlara neden olabildiği için, mümkün olduğunca var yerine let veya const kullanılması önerilir.

▼ JS de == , === farkı nedir?

▼ JS’de API çağırdığında Json’a çevirirken ne yaparsın?

▼ Seni motive eden – demotive eden 3 unsur nelerdir?

▼ Core 5 vs Core 6 vs Core 7 vs Core 8 arasındaki farklar nelerdir?

▼ Middleware kavramı hakkında neler biliyorsunuz, neden kullanılır?

▼ Servis life time kavramı hakkında neler biliyorsunuz?

▼ HTTP– Get, Post metod farkları nelerdir?

▼ Get metodu ile müşteri bilgilerini gönderip Create işlemi yapabilir miyiz?

▼ Sizi hangi konuda eleştirirler?