**1. Spring dışında dependency injection için kullanabileceğimiz framework’ler / kütüphaneler nelerdir ? (Herhangi bir programlama dili için olabilir.)**

1-) .Net için: LightInject , Ninject

2-) Java için : Google Guice, Apache Tapestry, HiveMind,

3-) JavaScript için : İnversify

4-) Php : Symfony, Laravel

**2. @SpringBootApplication anotasyonu hangi anotasyonları kapsamaktadır ? Görevlerini kısaca açıklayınız.**

@EnableAutoConfiguration: Sınıf dosyalarında bulunan jarları ve bizim tarafımızdan oluşturulan @bean’leri tarar, tanımlar ve kaydeder.

@ComponentScan: Spring’in uygulamamamızda ki configurations, controllers, services gibi componentlerin taranmasını sağlar.

@ Configuration: bir sınıfın bir veya daha fazla @Bean yöntem bildirdiğini ve çalışma zamanında bu çekirdekler için bean tanımları ve hizmet istekleri oluşturmak için Spring container tarafından işlenebileceğini gösterir.

**3.@Primary, @Qualifier anotasyonlarının kullanım amaçlarını açıklayınız.**

Aynı ara yüzden kalıtan iki tane bean varsa @Qualifier ile belirtilen isme sahip bean'e erişebilmeyi sağlar. @Autowired ile birlikte kullanılır.

**Aynı türden birden fazla bean olduğunda bir bean’i daha fazla tercih vermek için @Primary kullanırız.**

**5. Aspect Oriented Programlama nedir? Avantajları ve dezavantajları nelerdir?**

Yapılacak işlerin katmanlara ayrılarak bölünmesidir. Örneğin veritabanı işlemlerini farklı bir class da, controller işlemlerini de farklı bir class da yazılmasıdır. Bu işlemlerin kodları birlikte bir metodun içinde de yazılabilir fakat böyle yapıldığında tasarım prensiblrine uymaz ve kod karmaşası olabilir. Aynı zamanda bir çok kodu tekrar tekrar aynı yerlerde yazılması gerekebilir.

Object Oriented Programming’de de olduğu üzere oluşturmuş olduğumuz sınıfları, her sınıfın kendi sorumluluğunu yerine getirmesi (Single Responsibility) gibi prensiplere uyarak geliştirdiğimizde oluşan kodun okunabilirliğini (Readability), tekrar kullanabilirliğini (Reusability) ve genişletilebilirliğini (Extensibility) sağlamaktayız. AOP ise bize bu anlamda kodun okunabilirliğini ve tekrar kullanılabilirliğini en üst seviyede sağlamayı amaçlamaktadır

Object Oriented Programming’de soyutlamaya (Abstraction) giderekte kodun tekrar kullanılabilirliğini, okunabilirliğini ve genişletilebilirliğini ve sınıfların birbirine sıkı sıkıya bağlı (Tightly Coupled) olmamasını sağlayabiliriz ama bir noktaya kadar. O nokta geldiğinde ise AOP kesişen ilgilerimizi ele alarak onları tamamen bağımsız (Independent) hale getirmektedir. Bu sayede her bir kesişen ilgimiz, kendi içerisinde bağımsız olarak geliştirilebilir ve tekrar kullanılabilir hale gelmektedir.

**6. SOLID prensiplerini kısaca açıklayınız. Sizce her koşulda bu prensipler çerçevesinde mi kod yazılmalıdır? Neden?**

SOLİD prensipleri yazılım geliştirme konusunda sürekli kullanılmalıdır. Takım halinde çalışırken SOLİD prensipleri sayesinde Agile metodolojisine uyarak yazlım geliştirmiş olursunuz. Kodunuzu kısaltmaya ve daha anlamlı olmasını sağlar. Ayrıca projenizin ileride bağımlılık sorunları yaşamamasını sağlar.

Solid bir yazılım geliştiricinin esnek ve gelişmeye açık object oriented programlama yaparken uyması gereken kuralların bir araya getirildiği bir prensiptir. Dünya standartlarında yazılım geliştirmek için uymamız gereken bu prensipler 5 başlıkta ele alınıyor. Single Responsibility Principle Open/Closed Principle Liskov ‘s Substitution Principle Interface Segregation Principle Dependency Inversion Principle

**Single Responsibility Principle:** Tek sorumluluk anlamına gelen bu kuralın amacı projede bir değişiklik yapılmak istendiğinde buna bağlı olarak nelerin etkileneceği düşüncesinden kurtulmak ve özgürce isteğimiz geliştirmeyi yapabilmemize olanak sağlamaktır. Her bir method sadece kendisine verilen işi yapar aynı anda birden fazla modeli update etmez, örnek vermek gerekirse bir dizi işi yapan bir fonksiyon yazmaktansa tüm gereksinimleri parçalara ayırıp bağımsız fonksiyonlar ile yapmayı gerektiren bir kuraldır. Böylece zaman içerisinde geliştirme yaparken etkilenecek diğer aşamaları gözden kaçırmanız gibi bir risk oluşmaz.

**Open/Closed Principle:** Açık kapalı prensibi projemizdeki nesnelerin geliştirmeye açık ama değişime kapalı olmaları anlamına gelmektedir. Oluşturduğunuz nesneler zaman içerisinde ek özellikler kazanabilir genişlemeye açık olurlar bu normal bir yazılım projesinde kaçınılmaz bir durumdur. Ama temel nesne değişime kapalı tutulmalıdır.

**Liskov ‘s Substitution Principle:** Yerine geçme prensibi kalıtım alarak türeyen sınıfların kalıtım aldıkları nesnenin tüm özellikleri kullanmalı ve sonrasında kendi özelliklerini barındırmasını hedefleyen bir prensiptir eğer nesne kalıtım aldığı objenin tüm özelliklerini kullanmaz ise ortaya gereksiz kod yığınları oluşur ve sonrasında kalıtım alınan objenin diğerlerinden ayrılması için if else bloklarına ihtiyaç olur ve bu durum son derece verimsiz bir yazılıma sebep olur.

**Interface Segregation Principle:** Arayüz ayırım prensibi, bir arayüze gerektiğinden fazla yetenek eklenmemesi gerektiğini söyler. Nesnelerin ihtiyaç duymadıkları fonksiyonların Interface’lerinden münkün olduğunca ayrıştırılmasıdır.

Dependency Inversion Principle: Bağımlılığın ters çevirilmesi ilkesine göre üst seviye sınıflar, modüller, methodlar vs. alt seviyeli sınıflara bağımlı olmamalıdır. Alt sınıflarda yapılan değişiklikler üst sınıfları etkilememelidir. Yüksek seviyeli sınıflar, düşük seviyeli sınıflara bağlı olmamalı, her ikisi de soyut kavramlara bağlı olmalıdır.

**7. Swagger nedir, ne amaçla kullanılmaktadır?**

Swagger, API'lerinizin yapısını tanımlamanıza olanak tanır. Swagger ile API'nizin yapısını okuyarak, güzel ve etkileşimli API belgelerini otomatik olarak oluşturabiliriz. Ayrıca API'niz için birçok dilde otomatik olarak istemci kitaplıkları oluşturabilir ve otomatik test gibi diğer olasılıkları sağlayabilirsiniz. Swagger, API'nizden tüm API'nizin ayrıntılı bir açıklamasını içeren bir YAML veya JSON döndürür. Bu dosya, esasen, API'nizin OpenAPI Spesifikasyonuna uyan bir kaynak listesidir. Spesifikasyon, aşağıdaki gibi bilgileri eklemenizi ister:

API'nizin desteklediği tüm işlemler nelerdir?

API'nizin parametreleri nelerdir ve ne döndürür?

API'nizin biraz yetkilendirmeye ihtiyacı var mı?

Swagger ile bu bilgileri kolayca oluşturabilirsiniz.

Oluşturulan Apinin hem insanların hem de makinelerin anlayacağı formatta ortak bir dilde tanımlanmaya ihtiyacı vardır. Swagger Open Api standartlarına uyarak bu tanımlamayı yapmamızı sağlar.

8. Richardson Maturity Model’i seviyeleriyle birlikte açıklayınız.

Richardson Olgunluk Modeli, API'nizi REST kısıtlamalarına göre derecelendirmenin bir yoludur. API'niz bu kısıtlamalara ne kadar iyi uyarsa, puanı o kadar yüksek olur.

**Level-0: REST bir taşıma protokolü olarak değerlendirip kullanmaktır. HTTP bir aktarım için kullanılır fakat hiçbir web mekanizması kullanılmaz. Remote Procedure Call yaparak uzak sistemdeki bir metodu çağırmaktan farksızdır.**

**Level-1: URl üzerinden bir sorgu gönderimidir. Bu seviyede bir çok url kullanılır. Bu url’ler üzerinden istenen kaynağa erişilir.**

**Level-2: API'nin ölçeklenebilirlik ve hatalarla başa çıkmak için protokol özelliklerini kullanması gerektiğini gösterir.GET, POST, DELETE protokollerinin Api’de birlikte kullanıldığı seviyedir. Aynı zamanda bu metotlarla birlikte http durum kodları da kullanılır.,**

**Level-3: En üst düzey olan 3. Düzey, API'nizin istemcilere yönelik olanaklarını keşfetmek için HATEOAS'ı kullanır. HATEOAS REST uygulamasının herhangi bir uç noktasına istek gönderdiğimizde sonuç olarak sadece kaynakla ilgili bilgilerin yanında bu kaynakla ilgili diğer kaynakların, yapılacak işlem ve alınabilecek bilgilerin listesini/bağlantılarını da çoğu zaman beraberinde göndermesidir.**

**9.URL, URI, URN kavramlarını bir örnek üzerinden açıklayınız.**

**URI: Hem URL'leri hemde URN'leri kapsar.**

**URL:** **Uniform Resource Locator (URL), kaynağın İnternet'teki adresine atıfta bulunan bir karakter dizisidir. URL, nesnenin bulunabileceği konuma ulaşmaya yardımcı olur. Temel olarak internette yayınlanmış ve paylaşılan dosya, video, belge, resim, web sayfası vb.'nin web adresi veya konumudur.**

Örneğin; **makalleler.com/makale/12 Burda makaleler sitesindeki 12. Makalenin konumu verilmiştir. Bu konuma giderek bu kaynağa erişmiş olursunuz. Bu konum sadece 12. Makaleye özeldir.**

**URN: URN kaynağın adını veren bir karakter dizisidir. Gerçek hayatta olduğu gibi, İnternetteki iki nesne de aynı ada sahip olabilir. URN, tanımlanmış bir alan veya ad alanı içindeki bir kaynağa benzersiz bir kimlik verir.**

**mail:halil9819@gmail.com**

**10. Idempotency nedir ? Hangi HTTP metotları idempotent’ tir ?**

Sunucunun bir kaynak üzerinde bir eylemi birden çok kez oluşturması veya tamamlaması endişesi olmadan istediğiniz sayıda bağımsız çağrı yapabilmesidir.  İstemci 5xxSunucu kimliği sakladığı sürece , ağ zaman aşımları veya HTTP durum kodlarıyla başarısız olan bağımsız çağrıları yeniden deneyebilir. Idempotency, istek yüklerini yanıt yükleriyle ilişkilendirmenize, yinelenen istekleri ortadan kaldırmanıza ve başarısız istekleri veya belirsiz yanıtları olan istekleri yeniden denemenize olanak tanır.

Options, Get, Head, Put, Delete Http metodları idempotent’tir.

**11.RFC (Request For Comment) neyi ifade etmektedir ? HTTP hangi RFC dokümanında açıklanmıştır ? Bu dokümanda HTTP hakkında ne tür bilgiler yer almaktadır ?**

RFC'ler (Yorum Talebi), günümüzün modern bilgisayar ağlarının ve internetin temel yapı taşlarıdır. İnternet için neredeyse tüm ağ protokolleri RFC'lere dayalı olarak oluşturulmuştur.

Bazı RFC'ler bilgi amaçlıdır ve diğerleri yayınlanmış İnternet standartlarıdır. RFC'nin son versiyonu standart hale gelir ve bir sayı ile yayınlanır. Nihai sürüm için başka yorum veya değişikliğe izin verilmez. Değişikliklere yalnızca önceki RFC'lerin yerini alan sonraki RFC'ler aracılığıyla izin verilir.

"Standart" ağ protokolleri tescilli olarak kabul edilmez. Herhangi bir satıcı, tanımlanmış RFC standardına dayalı olarak uygulama yazılımı veya sürücüleri geliştirebilir. Bu nedenle RFC, platformlar arası ağ iletişimi için güçlü bir temel sağlar.

Tüm standart ağ protokolleri (HTTP, FTP, SMTP, TCP, UDP, IP vb.) RFC olarak tanımlanır. Bireyler, ağ standartlarını veya ağ protokollerini tasarlamaya ve geliştirmeye yardımcı olmak için IETF çalışma gruplarına katılabilir.

HTTP 7230 numaralı RFC dökümanında tanımlanmıştır. Mesajlar ve yönlendirme hakkında bilgiler verir.