Spring dışında dependency injection için kullanabileceğimiz framework'ler / kütüphaneler nelerdir ?

Google Guice (Java)

Dagger (Java & Android)

Castle Windsor (.NET)

Unity (.NET)

@SpringBootApplication anotasyonu hangi anotasyonları kapsamaktadır?

@Configuration : Java tabanlı configuration işlemi yapan bir anotasyondur.

@EnableAutoConfiguration : Projeye dahil edilen komponentlerin otomatik taranmasını sağlar.

@ComponentScan : Varsayılan konfigürasyonların otomatik gerçekleşmesini sağlar.

@Primary, @Qualifier anotasyonlarının kullanım amaçlarını açıklayınız.

@Primary : Aynı türden birden fazla bean olduğunda, öncelik vermek istediğimiz bean için bu anotasyon kullanılır.

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

Convention over configuration kavramını seçtiğiniz bir örnek üzerinden açıklayınız.

"Uygulama ayarlarını ayrı bir dosyada değil de kod içinde yazmak." Anlamına gelen bir kavramdır.

Aspect Oriented Programlama nedir ? Avantajları ve dezavantajları nelerdir ?

AOP, yazılımın karmaşıklığını azaltmaya, modülariteyi artırmaya yarayan bir yaklaşım biçimidir. Buradaki modülariteden kasıt uygulama süresince sistemin birçok bölümünde kullanılan, fonksiyonel olmayan kodun yani kesişen ilgilerin ufak ufak parçalara ayrılmasıdır (**Seperation of Cross Cutting Concerns**)**.** Bu sayede uygulama genelinde kullanılacak olan yapıları, sistemden soyutlamış olup enkapsüle de ederek birçok yerde kullanılmasını sağlar. Genel olarak AOP bir sorunu çözmektense var olan sistemin daha güzel bir hale getirilmesini de sağlamaya yardımcı olur denilebilir.

İçi içe yazılı tekrar edenler kod bloklarından kurtarır.

Temiz ve anlaşılır kod sağlar.

Yazmış olduğumuz kodları daha abstract hale getirerek modülerliğini arttırır.

Bakım ve geliştirme maliyetlerini azaltır.

Uygulamamızı daha yönetilebilir ve daha esnek hale getirilir.

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

1. Single Responsibility Principle

"Her sınıfın veya metodun tek bir sorumluluğu olmalı". Projelerde bu önem çoğu zaman ihmal edilir ve bir metodun içerisine birden fazla işi yapan kodlar yazılır. Bu da ileriki zamanlarda kodun bakımını zorlaştırır. Çünkü anlaması zor, okuması zor ve geliştirme maliyeti fazla olan bir kod ortaya çıkar. "Spagetti kod" ile baş başa kalırız.

2. Open/Closed Principle

Sınıflarımız veya metotlarımızı oluştururken ileride olabilecek yeni istekler ve gelişmeleri de öngörerek tasarlamamız gerekir. Projemizde oluşabilecek yeni istek ve ihtiyaçlar sonucunda yapacağımız geliştirmeler, projemizdeki diğer sistemleri etkilememeli ve herhangi bir değişikliğe sebebiyet vermemelidir.

3. Liskov 's Substitution Principle

Alt sınıflar miras aldığı üst sınıfın bütün özelliklerini kullanmalı, alt sınıflarda oluşturulan nesneler üst sınıfların nesneleriyle yer değiştirdiklerinde aynı davranışı göstermeli ve herhangi bir kullanılmayan özellik olmamalı.

4. Interface Segregation Principle

Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturmalıyız.

5. Dependency Inversion Principle

Sınıflar arası bağımlılıklar olabildiğince az olmalıdır özellikle üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır.

Swagger nedir, ne amaçla kullanılmaktadır ?

Swagger API, yazılım geliştiricilerin RESTful web hizmetlerini tasarlamasına, oluşturmasına, belgelemesine ve kullanmasına yardımcı olan geniş bir araç ekosistemi tarafından desteklenen açık kaynaklı bir yazılım framework'udur.

Richardson Maturity Model'i seviyeleriyle birlikte açıklayınız.

Richardson Maturity Model Rest API'lerimizin hangi seviyede olgun olduğunu gösteren bir **olgunluk seviye**sidir. RMM 4 seviyeden oluşmaktadır ve 0'dan 3'e başlayan seviyeler, yukarı doğru çıktıkca daha etkin kullanıldığını ifade etmektedir.

Level 0: Swamp of POX

Servisimizdeki sadece tek metod üzerinden POST olarak erişebildiğimiz seviyedir. Bu seviye transfer protokolü olarak da adlandırılmaktadır.

Level 1: Resources

Servimizdeki URI üzerinden bir metot üzerinden erişilebildiğimiz seviyedir. Örnek URI http://localhost/students/1

Level 2: HTTP verbs

POST, PUT, GET ve DELETE metotları çağırdığımız seviyedir.

Level 3: Hypermedia controls

Servislerimizde URI'nin istek gönderip cevaba göre tekrar URI'ye istek yapabildiğimiz bir seviyedir. Servis üzerinden gelen cevaba göre akıştaki davranışı görebiliriz. HATEOAS kullandığımız seviyedir.

URL, URI, URN kavramlarını bir örnek üzerinden açıklayınız.

URI (Uniform Resource Identifier), kaynağa tam olarak tanımlayıcı (identifier) ile işaret eden (belge, resim, dosya vb.) ve bu işaretleme için standart bir formata sahip karakter dizgisidir. URL kapsamında tutulan alt tanımlardır. Bir sayfa, görsel, dosya vb. **URI** için örnek olabilir; http://patikabootcampornekodevsitesi.com/bootcamp.png

URL (Uniform Resource Locator), Tekdüzen Kaynak Bulucu ya da Kaynak Konumlayıcı şeklinde ifade edilebilir. İnternet aracılığıyla erişişebilecek kaynakların (dosyalar, dökümanlar vb.) konumu URL ile ifade edilir. URL teknik olarak URI'nın başlangıç kısmını oluşturur. Yapısal olarak, URL'in ardından ise URN gelmektedir. Ancak, bir konum belirtilmek istendiğinde çoğu durumda URI yerine URL ifadesi kullanılmaktadır. URI sözdizimi (syntax) şöyledir:

scheme://domain:port/path?query_string#fragment_id

URN (Uniform Resource Name)

Bir kaynağı benzersiz ve kalıcı bir adla tanımlar, ancak bunu İnternet'te nasıl bulunacağının söylenmesi gerekmez. Belgeleri tanımlamakla sınırlı değildirler. Hatta, URN'ler fikirleri ve kavramları tanımlayabilir. Bir URN genelde urn: prefix'i ile başlar.

Idempotency nedir ? Hangi HTTP metotları idempotent' tir ?

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