1-MVC kavramını açıklar mısınız ? Neden ihtiyaç duyuluyor. Java'da nasıl kurgulanıyor. Object Oriented katmanları nelerdir ?

MVC programlama (Model View Controller) , Spagetti programlamaya göre çok daha avantajlı ve kolay kullanıma sahip bir programlama yapısıdır. Spagetti kodlama dediğimiz şey, bütün kodların tek bir sayfa da yer alması anlamına gelir. Yani Java kodlarının, database, bağlantılarının, database sorgularının ve hatta HTML , CSS3 dosyalarının tamamen bir sayfada olmasına Spaggetti kodlama denir.
Fakat MVC yapısında ise "Arayüz" ve "Mantık" kavramları tamamen birbirlerinden ayrılmaktadır. MVC deseni, 3 katmandan oluşmaktadır ve katmanları birbirinden bağımsız (birbirini etkilemeden) çalışmaktadır. Bu sebeple çoğunlukla büyük çaplı projelerde projelerin yönetiminin ve kontrolünün daha rahat sağlanabilmesi için tercih edilmektedir. MVC ile geliştirilen projelerde projenin detaylarına göre birçok kişi eş zamanlı olarak kolaylıkla çalışabilmektedir.

Model, MVC'de projenin database bağlantılarının olduğu, veri erişim (data access) işlemlerini gerçekleştirdiğimiz bölümdür.
View, MVC'de projenin arayüzlerinin oluşturulduğu client'ın da gözlemleyebildiği, HTML,CSS,JS,REACT JS kodlarının bulunduğu katmanı temsil eder.
Controller katmanı ise MVC'de projenin iç süreçlerini kontrol eden, endpointler ile @Getter @Setter anotasyanlarını kullanarak "Mantık" operatörlerini yazdığımız bölümdür.

OOP dört temel prensibe sahiptir.

- Abstarction (Soyutlama)
- Encapsulation (Kapsülleme-Gizleme)
- Inheritence (Kalıtım)
- Polimorphisim(Çok biçimlilik)

Abstraction, detayların ve karmaşıklığın azaltılması anlamına gelmektedir. Bir nesnenin neleri içermesi gerektiğine odaklanmayı ve önemli bilgileri gösterirken istenmeyen ayrıntıları gizlemeyi amaçlar. Nesnenin program için ihtiyaç duyulan özelliklerin arka plan ayrıntıları içermeksizin ifade ediliş biçimidir. Abstract class içerisinde ise sadece abstract olarak belirtilen metotlar extend eden class tarafından doldurulmalıdır. Abstraction'a klasik bir örnek verecek olursak eğer elimizde Telefon adlı bir interface class varsa, bu class'ı implemente eden classlar da telefon türleri olabilir.

Encapsulation, bir class içerisindeki tüm verilerin diğer classlar tarafından doğrudan erişimini kısıtlamak için kullanılır. Bu veriler ile işlem yapabilmek için sadece getter ve ,setter methodlarına ihtiyaç vardır. Bir nesnede dışarıdan görünmeyen bir özelliğiniz varsa ve erişimi sağlayan yöntemlerle bir araya getirirseniz, belirli bilgileri gizleyebilir ve nesnenin iç durumuna erişimi kontrol edebilirsiniz.

Inheritance , bir parent class içerisindeki özelliklerin child class içerisinde kullanılmasına denir.
Yazılan bir base sınıf bir başka sınıf tarafından miras alınabilir. Bu işlem yapıldığı zaman temel alınan sınıfın tüm özellikleri yeni sınıfa aktarılır.

Taban sınıfı olarak "Hayvan" adında bir sınıf düşünelim. Bu sınıfın özellikleri türü, yaşam alanı ve beslenme şekli olsun.

```
public class Hayvan {
    private String tur;
    private String yasam_alani;
    private String beslenme_sekli;

    public Hayvan(String tur, String yasam_alani, String beslenme_sekli) {
        this.tur = tur;
        this.yasam_alani = yasam_alani;
        this.beslenme_sekli = beslenme_sekli;
    }

    public void bilgileriGoster() {
        System.out.println("Tür: " + tur);
        System.out.println("Yaşam Alanı: " + yasam_alani);
        System.out.println("Beslenme Şekli: " + beslenme_sekli);
    }
}
```

Şimdi, Hayvan sınıfından türeyen "Kedi" adında bir alt sınıf düşünelim. Bu alt sınıfın, Hayvan sınıfındaki özelliklere ek olarak kediye özgü özellikleri (tüy rengi, pençe uzunluğu vb.) olsun.

```
public class Kedi extends Hayvan {
    private String tuy_rengi;
    private int pence_uzunlugu;

    public Kedi(String tur, String yasam_alani, String beslenme_sekli, String tuy_rengi, int pence_uzunlugu) {
        super(tur, yasam_alani, beslenme_sekli);
        this.tuy_rengi = tuy_rengi;
        this.pence_uzunlugu = pence_uzunlugu;
    }
```

```
    public void kediBilgileriniGoster() {
        bilgileriGoster();
        System.out.println("Tüy Rengi: " + tuy_rengi);
        System.out.println("Pençe Uzunluğu: " + pence_uzunlugu);
    }
}
```

Bu örnekte, "Kedi" sınıfı "Hayvan" sınıfından türemiştir ve Hayvan sınıfındaki özelliklere ek olarak Kedi sınıfına özgü özellikleri barındırmaktadır. Kedi sınıfı, Hayvan sınıfının tüm özelliklerine ve metotlarına da sahiptir.

Polymorphism (Çok Biçimlilik) ise genel olarak, birçok biçimde görünme yeteneğidir. OOP(Nesneye yönelik programlama)'da, bir programlama dilinin veri türüne veya sınıfına bağlı olarak nesneleri farklı şekilde işleme yeteneğini ifade eder. Daha spesifik olarak, türetilmiş sınıflar için yöntemleri yeniden tanımlama yeteneğidir. Bu prensibin en iyi şekilde anlaşılabilmesi için overriding ve overloading kavramlarının anlaşılması gerekmektedir. Örneğin extend edilen bir parent class içerisindeki obje referansı, extend eden child classlar içerisinde farklı şekillerde kullanılabilir. Yani parent class içerisindeki obje referanslarının child classlar içerisinde farklı şekillerde kullanılmasına polymorphism denir.

2- Birbirinden bağımsız iki platformun birbiriyle haberleşmesi nasıl sağlanabilir. Örneğin, X platformu Java ile yazılmış olsun, Y platform u C# ile. Bu iki platformun bir biri ile iletişim halinde request-response ilişkisi kurması gerekiyor. Bu yapıyı nasıl sağlarız?

Bu soruya öncelikle Rest ve Restful Service kavramlarını açıklayarak başlamak istiyorum.
REST, servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir veri transfer yöntemidir. HTTP üzerinde çalışır ve diğer alternatiflere göre daha basittir, minimum içerikle veri alıp gönderdiği için de daha hızlıdır. İstemci ve sunucu arasında XML veya JSON verilerini taşıyarak uygulamaların haberleşmesini sağlar. REST mimarisini kullanan servislere ise RESTful servis denir.

Restful servisler, birbirinden bağımsız iki platformun haberleşmesi için kullanılan bir protokol olarak kabul edilir, HTTP protokolünü kullanarak bir platformun kaynaklarına erişim sağlayarak, diğer platformların bu kaynaklar üzerinde işlem yapmasına izin verir.
Bu protokolde, bir platform (istemci) HTTP protokolünü kullanarak diğer platformdaki (sunucu) kaynaklarına istek gönderir. Sunucu, isteği işler ve istemciye bir cevap verir. Bu cevap, bir JSON, XML veya HTML gibi belirli bir veri formatında olabilir.
Restful servisler, tüm verilerin bir URI üzerinden erişilebilir olduğu ve her kaynağın benzersiz bir URI'sinin olduğu bir yapıda tasarlanmıştır. Bu sayede, istemciler bir kaynağa erişmek için URI'yi kullanabilir ve sunucu da istemcilerin isteklerini doğru bir şekilde işleyebilir. Özetle, Restful servisler, birbirinden bağımsız iki platform arasındaki haberleşme için kullanılan bir protokoldür ve HTTP protokolü üzerine inşa edilir. Bu servisler, kaynaklara URI üzerinden erişim sağlar ve verileri belirli bir veri formatında (JSON, XML vb.) sunar.

Sorudaki örneğimize dönecek olursak, dışardaki farklı tiplerde projelerin bu database'e erişebilmeleri için bizim database'imizi farklı tiplerdeki projeler ile paylaşmamıza gerek yoktur. Burada database'imizde işlem yapılabilmesi için tüm erişim kodlarımızı kendi uygulamamızda Controller yada Service içerisine yazabilir, diğer cihazlara uygun erişim haklarını buradan verebiliriz. Yani Database'imize erişimi direkt olarak firmaya vermek yerine, sadece bir adress bilgisi vererek erişimi kontrol altına alabiliriz. Burada kaynak alışverişi, tüm uygulamaların anlayabileceği bir Json yapısına döndürülerek sağlanır. Manuel testing için kullandığımız Postman Api de tam olarak bu mantıkla çalışmaktadır.

3- Bir web sayfasında ekran sürekli Backend' den veya bir başka yapı tarafından güncelleniyor. Siz, web sayfasını refresh etmeden bu güncel bilgiyi anlık ekrana nasıl yansıtırsınız.

Bu işlemi yapabilmek için web socket, socket Io yapılarını kullanabiliriz. Web socket dediğimiz yapı , çift yönlü bir iletişim sistemidir. Bu protokol, istemci ve sunucu arasında gerçek zamanlı veri alışverişi sağlar.

Ekran refresh etme ise bir web sayfasının, kullanıcının yaptığı işlemlere veya veri değişikliklerine anlık olarak tepki vermesi için güncellenmesidir. Web socket, bu işlemi daha hızlı ve verimli bir şekilde gerçekleştirmek için kullanılır.

Web socket ile ekran refresh etme, istemci tarafında bir web sayfası açıldığında, sunucudaki verilerin anlık olarak takip edilmesini ve sayfada herhangi bir değişiklik olduğunda bu değişikliğin hızlı bir şekilde istemci tarafında gösterilmesini sağlar.

Örneğin, bir online oyun sayfasında, rakip oyuncunun hareketlerinin anlık olarak görüntülenmesi için web socket kullanılabilir. Bu sayede, rakip oyuncunun hareketleri hızlı bir şekilde istemci tarafında görüntülenebilir ve oyun daha akıcı bir şekilde oynanabilir.

Temel adımlarla bu yapıyı teknik olarak nasıl kullanacağımıza geçecek olursak eğer ; öncelikle pom.xml dosyamızın içerisinde gerekli dependency dosyalarımızı eklemeliyiz. Daha sonra ise bu Configuration dosyamızda sunucumuzun ve client'ımızın dinlediği kodları yazarız. Daha sonra yapmamız gereken her şey Controller katmanında yer almaktaö Controller kısmında gerekli katmanlar çağırılıe ve gerekli işlemler yapılır.

4- Bir for döngüsü ile aşağıdaki çıktıyı yazar mısınız?

```
public class Star {

   public static void main(String[]  args) {

      int i,k;

      int star=1;

      System.out.println("*");
```

```
      for(k=1;  k<6; k++) {
         for (i=0; i <star; i++) {
            System.out.print("**");
         }
         System.out.println("");
         star++;
      }
   }
}
```

5- Firmada çalışman için sana remote bir linux server verildi. Elinde ip adresi port bilgisi kullanıcı adı ve şifren var. Server a erişimi nasıl test edersin, Server'a nasıl erişirsin, Server'a nasıl dosya atarsın, Serverdan nasıl dosya çekersin?

Bu işlemi gerçekleştirmek için öncelikle SSH komutunun kullanılması gerekir.

Linux sunucuda SSH (Secure Shell) ve SCP (Secure Copy) komutları, uzak sunuculara erişmek, dosyaları kopyalamak ve yönetmek için kullanılır. İşlevleri aşağıdaki gibi özetlenebilir:

SSH komutu, uzak sunuculara güvenli bir şekilde erişim sağlar. Bu komut, bir kullanıcı adı ve parolası ile uzak sunucuya bağlanmanızı sağlar. Ayrıca, uzak sunucuda bir komut çalıştırmak veya uzaktaki bir dosyayı görüntülemek için de kullanılabilir.

SCP komutu, dosyaları bir sunucudan diğerine kopyalamak için kullanılır. Bu komut, dosyaları güvenli bir şekilde kopyalamak için SSH protokolünü kullanır. Örneğin, bir sunucudaki dosyaları yerel bilgisayara kopyalamak veya yerel bilgisayardaki dosyaları bir sunucuya kopyalamak için kullanılabilir.

Temel kullanımlarından kısa bir örnek verecek olursak eğer,
SSH kullanımı: ssh kullaniciadi@uzak-sunucu-ip komutu ile uzak sunucuya bağlanabilirsiniz. Komutu çalıştırdıktan sonra kullanıcı adınızı ve şifrenizi girerek sunucuya erişebilirsiniz.

SCP kullanımı: scp kaynak-dosya hedef-dizin komutu ile dosya kopyalayabilirsiniz. Örneğin, scp dosya.txt kullaniciadi@uzak-sunucu-ip:/home/kullaniciadi/ komutu ile yerel bilgisayarınızdaki "dosya.txt" dosyasını "uzak-sunucu-ip" adresli sunucuya "/home/kullaniciadi/" dizinine kopyalayabilirsiniz.

Bu komutların daha gelişmiş kullanımları da mümkündür, ancak temel kullanımları bu şekildedir.

6- Local database kurulumu (mysql, postgresql veya herhangi bir database)
- Java spring uygulaması ayağa kaldırılması,
- İki adet tablo yer alamalı ve bu tabloların birbirleriyle bağı olmalıdır. (Örn: şirket ve çalışan
gibi),
- Java spring uygulamasında ekleme,silme,güncelleme,listeleme gibi servisler yer almalıdır ve
responseda yapılan işlem detayı return edilmelidir.
- Ekleme,silme,güncelleme,listeleme işlemlerini postman vb ile işlem yapılabilmelidir.

Öncelikle projenin tanımı, amacı ve katmanların yapısı ile ilgili bilgiler vermek ardından tüm bu adımların uygulandığı kodları adım adım açıklamaları ile göstermek istiyorum.

Proje İsmi : LibraryApp

Java kullanılarak kurgulanan proje, OOP prensipleri dikkate alınarak düzenlenmiştir.
Proje Spring Boot kapsamında katmanlı mimari kullanarak tasarlanmıştır. Bu mimari; projenin daha sürdürülebilir, test edilebilir ve genişletilebilir olmasını sağlamakla birlikte aynı zamanda farklı görevleri yerine getiren bileşenlerin birbirinden ayrılmasına olanak tanır. Daha kolay bir yönetim ve ölçeklendirme sağlar.

Projenin temel katmanlarından bahsedecek olursak;

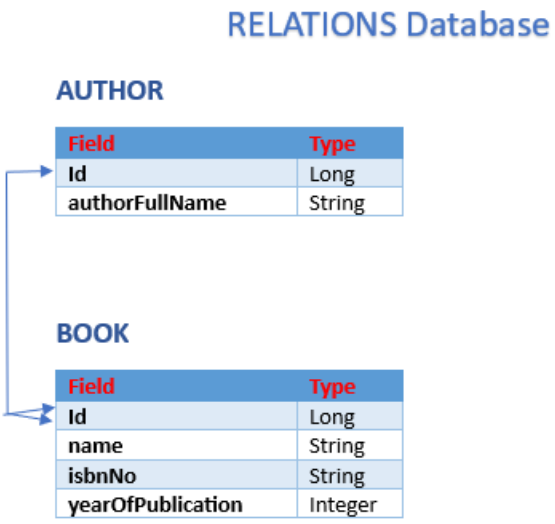
1) Entity Katmanı: Bu katman, veritabanında saklanan verilerin sınıflarını içerir. Bu sınıflar genellikle ORM (Object Relational Mapping) teknolojisi kullanılarak oluşturulur.

2) Controller Katmanı: Bu katman, gelen HTTP isteklerini ele alır ve bunları uygun servis katmanına yönlendirir. Ayrıca, yanıt olarak gönderilecek verileri oluşturur ve HTTP yanıtını hazırlar. Bu katman genellikle RESTful servisler için kullanılır.

3) Service Katmanı: Bu katman, iş mantığı işlemlerinin yapıldığı bileşenleri içerir. Controller katmanından gelen istekler buraya yönlendirilir ve buradaki sınıflar, işlemleri yürütmek için gerekli olan verileri işleyerek ilgili iş mantığı işlemlerini yerine getirir.

4) Repository Katmanı: Bu katman, veritabanı işlemlerini yerine getiren bileşenleri içerir. Veritabanına erişim için gerekli olan sınıfların yer aldığı katmandır ve genellikle Spring Data JPA ile oluşturulur.

Projede Author ve Book olmak üzere iki Entity Class ve bu classların Controller, Service , Repository,Mapper katmanları bulunmaktadır. Projenin veritabanı işlemleri PostgreSQL veritabanı kullanarak gerçekleştirildi. Book ve Author bilgilerinin tutulması için "tbl_book" ve "tbl_author" isimli iki farklı tablo oluşturuldu. Hibernate ile tabloların birbiriyle etkileşim içerisinde olması sağlanıldı. JPQL ile sorgulamalar yapıldı. Manuel testler için PostmanAPI kullanıldı.



Author Katmanı Kapsamında Yapılan işlemler:

- Gerekli anotasyonlar konuldu ve injectionlar yapıldı.
- Post Mapping ile yazar kaydı yapıldı.
- Get Mapping ile belirli Id'ye sahip yazar sorgulaması yapıldı ve DTO olarak response edildi.
- Get Mapping ile veritabanındaki bütün yazarların sorgulaması yapıldı ve DTO olarak response edildi.
- Get Mapping ile bütün yazarların Pageable yapısında sorgulaması yapıldı ve DTO olarak response edildi.
- Put Mapping ile yazar güncellenmesi yapıldı.
- Delete Mapping ile belirli verinin silinmesi işlemi yapıldı.

Book Katmanı Kapsamında Yapılan işlemler:

- Gerekli anotasyonlar konuldu ve injectionlar yapıldı.
- Post Mapping ile kitap kaydı yapıldı.
- Get Mapping ile belirli Id'ye sahip kitap sorgulaması yapıldı ve DTO olarak response edildi.
- Get Mapping ile kitap ismiyle kitap sorgulaması yapıldı ve DTO olarak response edildi.
- Get Mapping ile kitap türüne göre kitap sorgulaması yapıldı ve DTO olarak response edildi.
- Get Mapping ile kitabın isminden, yazar isminden ve kitabın türünden oluşan özet bir sorgulama işlemi yapıldı.
- Get Mapping ile veritabanındaki bütün kitapların sorgulaması yapıldı ve DTO olarak response edildi.
- Get Mapping ile bütün kitapların Pageable yapısında sorgulaması yapıldı ve DTO olarak response edildi.
- Put Mapping ile kitap güncellenmesi yapıldı.
- Delete Mapping ile belirli verinin silinmesi işlemi yapıldı.

1) Connection To DB

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/LibraryApp
    username: db_user
    password: db_password
```

PostgreSql içerisinde oluşturmuş olduğum database yol haritamı application.yml dosyamın içerisinde yukarıda belirttiğim gibi eklemeliyim.

2) Creating To Tabel (Model)

Daha sonra hibernate injection yardımı ile domain classlarım ile database'imin içerisindeki table'larımı oluşturdum.

```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor

@Entity
@Table(name = "tbl_author")
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Setter(AccessLevel.NONE)
    private Long id;


    @Column(nullable = false, length = 50)
    private String authorFullName;

    @JsonIgnore
    @OneToMany(mappedBy = "author")
    private List<Book> books = new ArrayList<>();
}
```

@Entitiy annotation yardımı ile databasede oluşturduğum table'lara, yukarıda belirtmiş olduğum Author entity class örnek olarak verilebilir. Bunun ile benzer olarak aynı şekilde Book entity class da gösterilebilir.

4) Controller

Controller classları içerisinde Author ve Book ile ilgili yukarıda bahsettiğim tüm işlemlerimi gerçekleştirdim. Controller katmanı bizim client'ımıza response gönderdiğimiz katmandır. İşlemlerimizin gerçekleşmesi için gerekli anotasyonlar konuldu ve injectionlar yapıldı.

```java
@RestController
@RequestMapping("/book")
public class BookController {


    10 usages
    private final BookService bookService;

    Beyza KESER
    public BookController(BookService bookService) { this.bookService = bookService; }
```

Book ile benzer olarak aynı şekilde Author entity classını da gösterebiliriz.

```java
@RestController
@RequestMapping("/author")
public class AuthorController {

    7 usages
    private final AuthorService authorService;

    Beyza KESER
    public AuthorController(AuthorService authorService) { this.authorService = authorService; }
```

4)Service

```
@Service
public class BookService {

    10 usages
    private final BookRepository bookRepository;
    3 usages
    private final AuthorService authorService;

    7 usages
    private final BookMapper bookMapper;

    Beyza KESER
    public BookService(BookRepository bookRepository, AuthorService authorService, BookMapper
        this.bookRepository = bookRepository;
        this.authorService = authorService;
        this.bookMapper = bookMapper;
    }
```

Controller dan gelen tüm görevleri service katmanı içerisinde gerçekleştiririz. Service katmanımızdan ya işlem gerçekleştirilir ve client'a bir response döner ya da service içerisinden data controller katmanına return edilir. (void, return).

5) Repository and Map

```
3 usages  Beyza KESER
@Mapper(componentModel = "spring")
public interface BookMapper {

    2 usages  Beyza KESER
    BookDTO bookToBookDTO(Book book);
    3 usages  Beyza KESER
    List<BookDTO> map(List<Book> books);

    1 usage  Beyza KESER
    Book bookRequestToBook(BookRequest bookRequest);
}
```

```java
@Repository
public interface BookRepository extends JpaRepository<Book,Long> {

    1 usage  Beyza KESER
    List<Book> findByName(String name);

    1 usage  Beyza KESER
    List<Book> findByType(String type);

    1 usage  Beyza KESER
    @Query("SELECT b.name, b.author.authorFullName, b.type FROM Book b")
    List<Object> bookSummary();
}
```

Controller ve service katmanlarımızda bu işlemlerin gerçekleştirilebilemesi için repository ve map interfacelerine ihtiyacımız vardır. Burada kendilerini implemente eden classlar ile verileri repository katmanına en uygun şekilde göndeririz. Book olarak çektiğimiz verileri client a gönderirken BookDTO formatında göndeririz. Burada mapper interfacine ihtiyaç duyarız. Repository ile bağlantı sağlarken de repository interface'ini JpaRepository extend ederek tüm crud operationları'nı gerçekleştiririz.

Projeyle, katmanlarıyla ve projede yapılan işlemlerle ilgili bütün detayları paylaştığım Github bağlantısından bulabilirsiniz.

7- Apache Solr servisine yazılacak bir query örneği Apache Solr kullanılan sql programlarından daha farklı runtime bir database. Solr a hali hazırda kayıtlı bir alan olduğunudüşünelim. Alanın ismi “updatedAt” long tipinde tutulan bir alan. Ben 2020 Ocak ayından sonraki verileri getir dediğimde solr a nasıl bir query yazılmalı. http://example?query=kısmını nasıl doldurmalıyım?

Apache Solr, bir veritabanı yönetim sistemi değildir. Apache Solr, açık kaynaklı bir arama platformudur. Solr, web uygulamaları, büyük ölçekli veri depolama, e-ticaret siteleri ve daha birçok alanda kullanılabilir. Ayrıca Solr, isteğe bağlı ölçeklenebilirlik özellikleri ve yüksek performanslı arama hızı gibi avantajlara da sahiptir. RESTful API'si sayesinde, diğer uygulamalarla kolayca entegre edilebilir.

Solr'a bir query göndermek için genellikle HTTP GET isteği kullanılır ve istek URL'sinde "q" parametresi ile query belirtilir. Bu örnekte, "updatedAt" adlı bir alanımız var ve Ocak 2020'den sonraki verileri getirmek istiyoruz.

Solr'da tarih alanları için "DateRangeField" tipi kullanılır. Bu tip alanlar, verileri tarih ve saat bilgileri ile indeksler ve tarihe göre sıralama yapılabilmesini sağlar.

Örneğin, Ocak 2020'den sonraki verileri getirmek için aşağıdaki query kullanılabilir:

q=updatedAt:[2020-01-01T00:00:00Z TO *]

Bu query, "updatedAt" alanı için 1 Ocak 2020'den sonraki tarihleri içeren belgeleri getirecektir. "T00:00:00Z" ifadesi belirtilerek, saati sıfırlanmış bir tarih belirtilir. "* " karakteri ise herhangi bir saat ve dakika belirttiğimizi gösterir.