



T.C.

TOKAT GAZİOSMANPAŞA ÜNİVERSİTESİ
MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ

STAJ DEFTERİ

ÖĞRENCİNİN

ADI SOYADI :...İrem.....Çakmak.....
NUMARASI :.....203908028.....
BÖLÜMÜ / SINIFI :...Bilgisayar...Mühendisliği.../...4.Sınıf.....

T.C.
TOKAT GAZİOSMANPAŞA ÜNİVERSİTESİ
MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ

2023/ 2024 EĞİTİM ÖĞRETİM YILI

STAJ YAPAN ÖĞRENCİYE AİT BİLGİLER

ADI SOYADI :İrem Çakmak
NUMARASI :203908028
BÖLÜMÜ :Bilgisayar Mühendisliği
SINIFI :4.Sınıf

STAJ YAPILAN KURUM/KURULUŞA AİT BİLGİLER

ADI :...PTT...Bilişim...Daire...Başkanlığı.....
ADRESİ : CEPA AVM Yanı Mustafa Kemal Mahallesi 2151 Cd. 06510
Çankaya/ Ankara
TELEFON :..... 03122538918.....

Yukarıda kimliği yazılı öğrenci 07/10/2024 ile 04/11/2024 tarihleri arasında toplam

...20... işgünü staj çalışmasını yapmıştır.

İŞYERİ AMİRİ		
ADI VE SOYADI	ÜNVANI	İMZA-MÜHÜR

T.C.
TOKAT GAZİOSMANPAŞA ÜNİVERSİTESİ
MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ

GÜNLÜK STAJ DEVAM ÇİZELGESİ

GÜN	TARİH	YAPILAN İŞ	İMZA
1.	07/10/2024		
2.	08/10/2024		
3.	09/10/2024		
4.	10/10/2024		
5.	11/10/2024		
6.	14/10/2024		
7.	15/10/2024		
8.	16/10/2024		
9.	17/10/2024		
10.	18/10/2024		
11.	21/10/2024		
12.	23/10/2024		
13.	24/10/2024		
14.	25/10/2024		
15.	28/10/2024		
16.	30/10/2024		
17.	31/10/2024		
18.	01/11/2024		
19.	04/11/2024		
20.	05/11/2024		

Springboot nedir ve neden kullanılır?

Springboot , spring framework'ün uzantısıdır. Springboot web uygulamaları ,API ve mikroservisler geliştirmek için yapılır. Java tabanlı uygulamaları geliştirmek için kullanılır. Java tabanlı uygulamaları geliştirmek için basic anotasyonlar sunar. Tomcat, Jetty yerleşik sunucuları vardır bu yüzden de harici sunucuya ihtiyaç duyulmaz. Tomcat, java tabanlı web uygulamalarda kullanılan servlet container yazılımlardır. Java Servlet http protokolünü destekler. Springboot projelerde tomcat sunucuları entegre gelir. Jetty, java tabanlı web uygulamalarda kullanılan hafif ve modüler yapıda olan servlet container yazılımlardır. Java Servlet http protokolünü destekler. Performans esneklik isteyen projelerde ve mikro servis projelerinde kullanılır.

Özellik	Tomcat	Jetty
Performans	Orta (daha fazla bellek tüketebilir)	Hafif ve hızlı başlatılır
Kullanım Alanı	Kurumsal ve büyük web uygulamaları	Mikro servisler, IoT projeleri
Yerleşik Kullanım	Spring Boot ile varsayılan	Hafif mikro servislerde yaygın
Asenkron İşlem	Destek var ama sınırlı	Daha güçlü asenkron destek
Modülerlik	Daha az modüler	Çok modüler ve özelleştirilebilir

Kolay Test Edilebilirlik:Geliştirme sırasında kolayca JUnit veya Mockito gibi test araçlarıyla entegre olur.Spring Boot uygulamaları için özel olarak geliştirilmiş @SpringBootTest gibi anotasyonlar test sürecini kolaylaştırır.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
```

Pom.xml dosyası

Dependencies ihtiyaç duyduğu kütüphaneleri, frameworkleri , dışa bağımlılıkları ifade eder. pom.xml içinde bulunur.

Dışa Bağımlılıklar:

Lombok: Getter/Setter gibi kod tekrarlarını önlemek için ve

tekrar eden kodları otomatik olarak üretir.

PostgreSql: Veri tabanı kullanımı için.

Devtools: Geliştirme sürecini hızlandırmak için ek kolaylıklar sağlayan bir modüldür.

ONAYLAYAN YETKİLİ

ADI SOYADI

UNVANI

İMZA

TARİH:...../...../20.....	SAYFA NO:	
<p>Jar dosyası java arşivlerini çalıştırmak için arşiv formatıdır.</p> <p>Gradle ve Maven yapıları, bir projenin Spring Boot veya başka bir Java tabanlı framework kullandığını anlamamıza yardımcı olur. Maven'in pom.xml dosyası veya Gradle'in build.gradle dosyası, bu tür projelerde kullanılan bağımlılıkları (dependencies) ve yapılandırma ayarlarını barındırır.</p> <p>Spring Security, Java uygulamalarına güvenlik katmanı eklemek için kullanılan bir güvenlik çerçevesidir (framework). Uygulamalarınızda kimlik doğrulama (authentication) ve yetkilendirme (authorization) gibi işlemleri kolayca yapmanızı sağlar. Hem web tabanlı hem de API tabanlı uygulamalarda kullanıcıların erişimini güvence altına alır.</p> <p>Spring Framework ve Hibernate ORM birlikte kullanıldığında, Java tabanlı uygulamalar için güçlü bir veritabanı erişim katmanı sağlar. Spring + Hibernate entegrasyonu, veri işlemlerini daha esnek, modüler ve bakımı kolay hale getirir.</p> <p>Spring Data JPA, Java Persistence API (JPA) kullanarak veritabanı işlemlerini kolaylaştırmak için Spring Framework tarafından sağlanan bir modüldür. Veritabanı işlemleri için Hibernate gibi ORM araçlarının gücünü kullanır ve geliştiricilere CRUD (Create, Read, Update, Delete) işlemleri için güçlü ve kolay bir yapı sunar. Entity tablosuna karşılık gelen sınıftır. Hipernate ORM(Object Relational Mapping) araçlarıyla çalışarak veritabanı işlemlerini geliştirmeye olanak tanır. ORM class-db arasındaki ilişkiyi sağlar ve ORM işlemlerini kolaylaştıran frameworkdur.</p> <p>API Nedir?</p> <p>API (Application Programming Interface), yazılımların birbirleriyle iletişim kurmasını ve veri alışverişi yapmasını sağlayan bir arayüzdür. API'ler, farklı uygulama veya sistemlerin belirli kurallar çerçevesinde birbiriyle entegrasyonunu mümkün kılar.</p> <p>API Türleri:</p> <p>REST API (Representational State Transfer):</p> <ul style="list-style-type: none">-HTTP protokolü üzerinden çalışır.-JSON veya XML formatında veri gönderip alır.-Web servisleri arasında yaygın olarak kullanılır.-Örnek: GET /products ile ürünlerin listesi alınır. <p>SOAP API (Simple Object Access Protocol):</p> <ul style="list-style-type: none">-HTTP, SMTP gibi protokollerle çalışır.-Veri formatı olarak XML kullanır.-Daha ağır ve güvenlik açısından daha sıkı kurallara sahiptir. <p>GraphQL API:</p> <ul style="list-style-type: none">-İstemciye ihtiyaç duyduğu verileri spesifik olarak seçme imkanı sunar.-Daha esnek sorgu yapısı sağlar. <p>Open API (Açık API):</p> <ul style="list-style-type: none">-Herkese açık olup, dış geliştiriciler tarafından kullanılabilir. Örneğin, Google Maps API'si.		
ONAYLAYAN YETKİLİ		
ADI SOYADI	ÜNVANI	İMZA

API Türü	Açıklama	Örnekler	Kullanım Durumu
Database API	Veritabanıyla veri alışverişini sağlar	JDBC, ODBC	Veritabanı işlemleri (sorgulama, veri ekleme/silme vb.)
Operating System API	İşletim sisteminin işlevlerine erişim sağlar	POSIX, WinAPI	Dosya işlemleri, bellek yönetimi, cihaz erişimi
Remote API	Farklı bir sistemde çalışan hizmetlere ağ üzerinden erişim sağlar	SOAP, JSON-RPC	Dağıtık sistemlerde uzak hizmetlere erişim
Web API	İnternet üzerinden erişilebilen ve genellikle HTTP üzerinden çalışan API'ler	REST, GraphQL, SOAP	Web tabanlı hizmetlere veya mikro servis yapılarına erişim

API Nasıl Çalışır?

-Bir istemci (client), belirli bir API'ye istek (request) gönderir ve sunucu (server) bu isteğe yanıt (response) verir. Bu iletişim genelde HTTP protokolü üzerinden gerçekleşir.

ONAYLAYAN YETKİLİ

ADI SOYADI	ÜNVANI	İMZA

TARİH:...../...../20.....		SAYFA NO:
- SOAP ve REST Arasındaki Farklar:		
Özellik	SOAP API	REST API
Mimari Yapı	Katı bir protokol (SOAP standardına bağlı)	Esnek mimari stil (HTTP protokolü kullanır)
Veri Formatı	Yalnızca XML	JSON, XML, HTML, Text gibi çeşitli formatlar
Hız ve Performans	Daha yavaş, daha ağır	Daha hızlı ve hafif
HTTP Metotları	Kendi mesajlaşma yapısına sahiptir	GET, POST, PUT, DELETE gibi HTTP metotlarını kullanır
Güvenlik	WS-Security standartlarını kullanır, yüksek güvenlik sağlar	Daha az katı güvenlik, HTTPS ile güvenlik sağlanır
Durum Bağlılığı	Durum bilgisi taşıyabilir (stateful)	Stateless (istemcinin durumunu hatırlamaz)
Ölçeklenebilirlik	Daha az ölçeklenebilir	Yüksek ölçeklenebilir
Kullanım Alanları	Bankacılık, ödeme sistemleri gibi yüksek güvenlik gerektiren alanlar	Web ve mobil uygulamalar, IoT sistemleri
Hata Yönetimi	Daha detaylı ve standart hata mesajları	Hata yönetimi basit ve HTTP hata kodlarıyla yapılır
Esneklik	Katı ve kompleks	Daha esnek ve kullanımı kolay
İletişim Protokolü	HTTP, SMTP, TCP/IP gibi birden fazla protokolü destekler	Yalnızca HTTP kullanır
	Basit Nesne Erişim Protokolü (SOAP) , sistemler arasında mesajlaşma için kullanılan bir protokoldür.	"Temsili Durum Aktarımı" ilkesine dayanan bir mimari tarzıdır.
İletişim	SOAP API , yazılım mimarisinde iletişim arabirimleri olarak görev yapar ve uygulamalar arasında veri alışverişini sağlamak için kullanılan bir protokoldür.	İletişim arabirimleri tasarlamaya yönelik yazılım mimaridir.
ONAYLAYAN YETKİLİ		
ADI SOYADI	UNVANI	İMZA

TARİH:...../...../20.....		SAYFA NO:	
-SOAP API ve REST API'nin Benzerlikleri Tablosu			
Kriter	SOAP API	REST API	Benzerlik
Temel Amaç	Sistemler arası veri alışverişi	Sistemler arası veri alışverişi	İkisi de istemci-sunucu iletişimi sağlar
Protokol	HTTP, SMTP, TCP/IP destekler	HTTP üzerinden çalışır	İkisi de HTTP protokolünü destekler
Platform Bağımsızlığı	Farklı platform ve dillerle uyumludur	Farklı platform ve dillerle uyumludur	Her ikisi de farklı teknolojilerle entegre olabilir
Veri İşleme (CRUD)	SOAP Action ile veri işlemleri yapar	HTTP metotları (GET, POST vb.) kullanır	İkisi de CRUD işlemlerini destekler
Güvenlik	HTTPS, WS-Security gibi güvenlik katmanları	HTTPS, OAuth, JWT kullanabilir	İkisi de HTTPS ile güvenli iletişim sağlar
XML Desteği	Zorunlu olarak XML kullanır	JSON'un yanı sıra XML de destekler	Her ikisi de XML formatını kullanabilir
Hata Yönetimi	SOAP-Fault mesajları ile hata bildirir	HTTP hata kodları ile hata bildirir	Hata yönetimi için özel mekanizmaları vardır
Dağıtık Sistemlerde Kullanım	Evet	Evet	İkisi de dağıtık mimarilerde veri alışverişi sağlar
Dil Bağımsızlığı	Evet	Evet	Her ikisi de farklı dillerde geliştirilmiş sistemlerle çalışabilir
<ul style="list-style-type: none">REST API: Web servislerinin belirli kurallara dayanarak nasıl dayanarak nasıl tanımlanacağını söyleyen yazılım mimarisidir. Daha hafif ve esnek bir mimari stil sunar. JSON, XML, HTML gibi farklı veri formatlarını destekler. HTTP protokolü üzerine kurulu olup, modern web ve mobil uygulamalarda yaygındır. API türü olarak sistemler arasında veri alışverişi yapmayı sağlar.İnternet mekanizmasıdır.			
ONAYLAYAN YETKİLİ			
ADI SOYADI		ÜNVANI	İMZA

request → response

HTTP İstekleri

CREATE	→	Post
READ	→	Get
UPDATE	→	Put-Patch
DELETE	→	Delete

Anatasyonlar nedir?

Anotasyonlar, Java ve diğer programlama dillerinde, kodu daha açıklayıcı hale getirmek ve belirli işlevsellikleri sağlamak için etkili bir araçtır. Anotasyonlar, özellikle framework'lerde ve kütüphanelerde kodun daha temiz ve yönetilebilir olmasına yardımcı olur.

Not: Endpoint (uç nokta), bir API veya web uygulamasının dış dünyaya sunduğu erişim noktasıdır. İstemciler, endpoint'ler aracılığıyla belirli verilere veya işlemlere erişebilir. Her endpoint genellikle bir URL ve HTTP metodu (GET, POST, PUT, DELETE vb.) ile tanımlanır.

HTTP Yöntemi	Endpoint	Açıklama
GET	/api/books	Tüm kitapları listele
GET	/api/books/{id}	Belirli bir kitabı getir
POST	/api/books	Yeni bir kitap ekle
PUT	/api/books/{id}	Belirli bir kitabı güncelle
DELETE	/api/books/{id}	Belirli bir kitabı sil

@RestController web servislerini başlatmak için kullanılan anotasyondur.

@GetMapping http isteklerinden get isteğini başlatan anotasyondur.

@RequestMapping http isteklerindeki bir kontrolcü sınıfındaki belirli methoda yönlendirmek.

- localhost:8080/api
- @RequestMapping(path="api")
- method=RequestMethod.GetRead
- method=RequestMethod.GetPost
- method=RequestMethod.GetCreate
- method=RequestMethod.GetDelete

CREATE	→	Post (Data kaydetmek)	@PostMapping
READ	→	Get (okumak)	@GetMapping
UPDATE	→	Put(Güncellemek)	@PutMapping
UPDATE	→	Patch(Belirli olan dataları güncellemek)	@PatchMapping
DELETE	→	Delete(Silmek)	@DeleteMapping

ONAYLAYAN YETKİLİ

ADI SOYADI	UNVANI	İMZA

@PathVariable Http istekleri urlden dinamik olarak parametreleri için kullanılır.

@RequestParam Http isteğinden gelen parametreleri kontrolör methoduna bağlamak.

Required özelliği: Spring framework anotasyonudur , http isteğini belirli methodla ilişkilendirilir.

@RequestBody Verileri nesneye dönüştürmek için kullanılan anotasyondur.

@RequestHeader Http isteklerinde header bilgilerini almayı sağlar.

@Autowired Otomatik bağımlılık enjeksiyonunu gerçekleştirmek için kullanılır.

@Repository RestResource Spring data birlikte JPA yada başka veri erişim katmanını RESTFUL servisine dönüştürür.

@ManyToMany, @ManyToOne

@ManyToOne : Bir varlığın, başka bir varlıkla "çoktan bire" (many-to-one) ilişkide olduğunu belirtir.

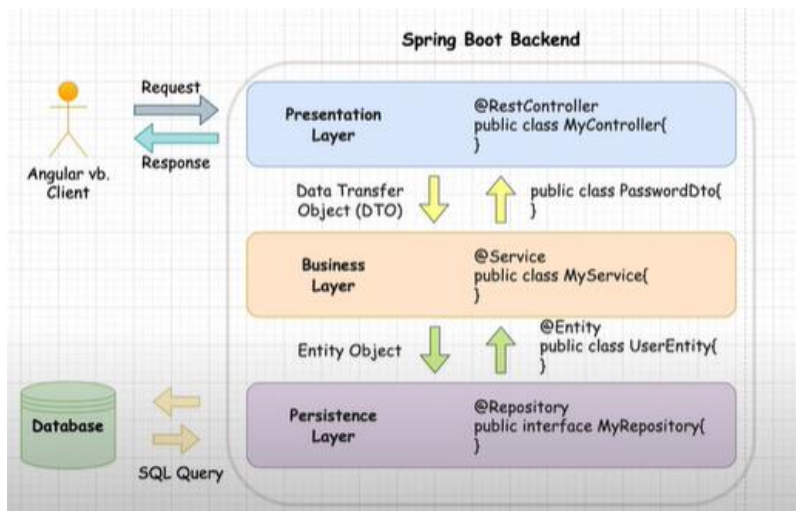
@ManyToMany : Bir varlığın, başka bir varlıkla "çoktan çoğa" (many-to-many) ilişkide olduğunu belirtir.

@JoinColumn(nullable = false) ifadesi, JPA'de bir tablo ile başka bir tablo arasında ilişki kurarken kullanılan bir özelliktir. @JoinColumn anotasyonu, iki tablo arasında bir foreign key (yabancı anahtar) ilişkilendirmesi yaparken hangi kolonun kullanılacağını belirtir. nullable = false ise bu yabancı anahtar kolonunun boş geçilemez olduğunu ifade eder.

Not:

Özellik	Amacı
@OneToMany(fetch = FetchType.EAGER)	İlişkili verilerin ne zaman yükleneceğini belirler (LAZY ya da EAGER).
@OneToMany(mappedBy = "user")	İlişkili verilerde ilişkiyi hangi varlığın yöneteceğini belirtir (ilişkinin sahibi olan tarafı).

- Katmanlı Mimariye giriş



ONAYLAYAN YETKİLİ

ADI SOYADI

UNVANI

İMZA

Katmanlı mimari, yazılımı mantıksal katmanlara ayırarak her katmanın belirli bir sorumluluğa sahip olmasını sağlar. Bu yapı, geliştirme ve bakım sürecini daha kolay ve anlaşılır hale getirir.

- **Presentation Layer (Sunum Katmanı):**

Kullanıcıdan gelen istekleri alır ve cevapları döner.

Genellikle @RestController olarak işaretlenir.

MyController sınıfı, kullanıcıdan gelen istekleri (örneğin, Angular gibi bir istemciden) alır ve işlenmesi için Service katmanına iletir.

Burada veri transferi için DTO (Data Transfer Object) sınıfları kullanılabilir. Örneğin, PasswordDto sınıfı, sunum katmanında kullanılan veriyi temsil eder.

- **Business Layer (İş Katmanı):**

İş mantığının bulunduğu katmandır.

@Service anotasyonu ile işaretlenen MyService sınıfı, iş kurallarını ve uygulama mantığını barındırır.

DTO sınıfından aldığı veriyi işler ve sonuçları Repository katmanına iletir ya da Controller katmanına geri gönderir.

- **Persistence Layer (Kalıcı Veri Katmanı):**

Veritabanı ile iletişimi sağlar.

@Repository ile işaretlenen MyRepository sınıfı, SQL sorguları veya ORM kullanarak veritabanına veri gönderme ve veri alma işlemlerini gerçekleştirir.

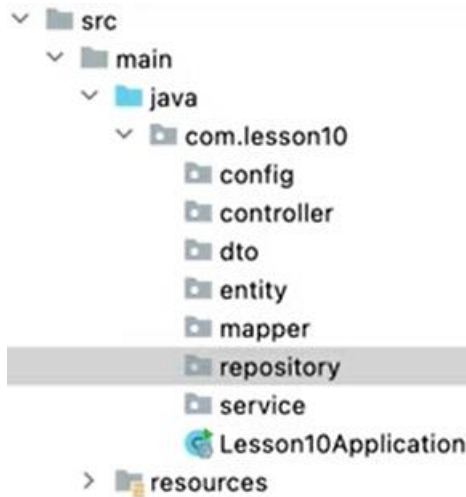
Entity sınıfı (UserEntity), veritabanındaki tabloları temsil eder ve nesne yapısında veriyi saklar.

- **Database (Veritabanı):**

Kalıcı veri saklanan yerdir.

Veriler Persistence Layer aracılığıyla buraya kaydedilir ve gerektiğinde sorgulanır.

-Not:



config: Uygulama yapılandırmalarının bulunduğu klasör.

controller: HTTP isteklerini alan ve yöneten katman.

dto: Veri Transfer Nesnelerini (Data Transfer Objects) içeren katman; veri taşımada kullanılır.

entity: Veritabanı tablolarını temsil eden nesneler burada tanımlanır.

mapper: Veri dönüşümlerini ve nesneler arası eşlemeleri yapan sınıfları içerir.

repository: Veritabanı işlemlerini gerçekleştiren sınıflar burada bulunur.

service: İş mantığını barındıran katman.

ONAYLAYAN YETKİLİ

ADI SOYADI

UNVANI

İMZA

Dependency Injection

Dependency Injection (Bağımlılık Enjeksiyonu), bir sınıfın bağımlılıklarını (başka bir sınıfın nesneleri) dışarıdan sağlama yöntemidir.

```
@Autowired
private firstClass FirstClass ;
```

Field injection Doğrudan sınıfın alanlarına enjekte edilmesi yöntemidir.
(private fields)

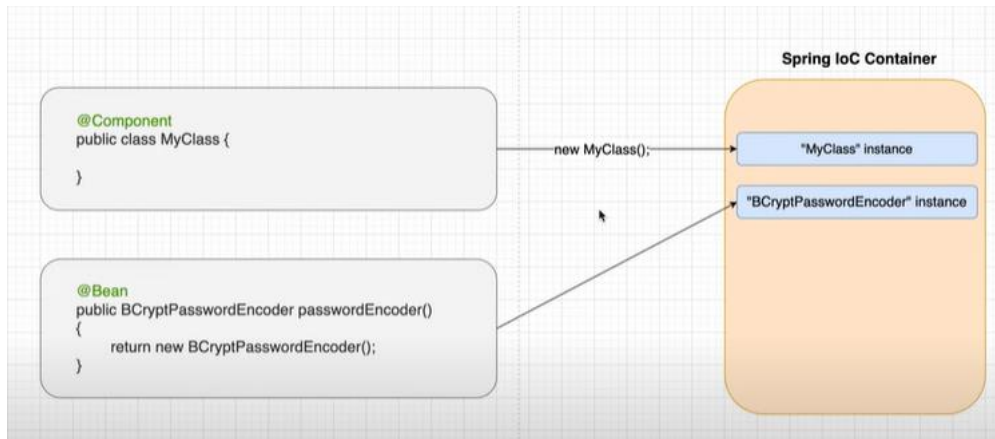
Constructon injection(Constructor yeni nesne oluşturulduğunda kullanırız.)

```
private final MyService myService; 3 usages

public Mycontroller(MyService myService) {
    this.myService = myService;
}
```

```
private secondClass SecondClass ; 2 usages
@Autowired
public void setSecondClass(secondClass secondClass) {
    SecondClass = secondClass;
}
```

Setter Injection, bir sınıfın ihtiyaç duyduğu bağımlılıkları sağlamak için kullanılan bir bağımlılık enjeksiyonu (dependency injection) yöntemidir.



@Component: Bu anotasyon, MyClass sınıfını Spring IoC Container'a bir bileşen (bean) olarak ekler. Spring, bu sınıfın bir örneğini otomatik olarak oluşturur ve yönetir.

@Bean: passwordEncoder isimli metot, Spring IoC Container'da bir BCryptPasswordEncoder nesnesi oluşturan bir bean tanımlar. @Bean anotasyonu, bu metodu Spring'e bir bean olarak kaydeder, böylece uygulama içinde bu nesneye ihtiyaç duyulduğunda, Spring bunu otomatik olarak sağlar.

ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

Inversion of Control (IoC) Container:

Yazılım geliştirmede bağımlılıkları yönetmek ve sınıflar arasındaki ilişkileri düzenlemek için kullanılan bir tasarım prensibidir. IoC Container, bir uygulamanın nesnelerini oluşturur, onların yaşam döngülerini yönetir ve diğer nesnelere olan bağımlılıklarını otomatik olarak enjekte eder.

Spring IoC Coytainer (Application Context Interface)
Temel interface

Inversion of Container IoC kontrolün tersine çevirilmesi anlamına gelir.

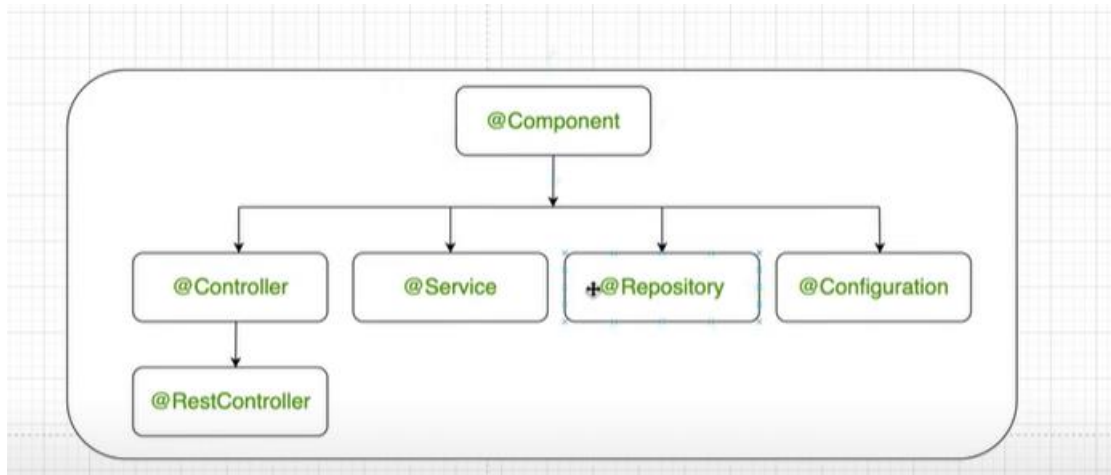
BeanFactory: Basit IoC Container implementasyonudur. Temel seviyede bağımlılık enjeksiyonu yapar.

ApplicationContext: BeanFactory'nin üstünde bir yapı sunar ve daha gelişmiş özellikler (örn. AOP desteği, mesajlaşma) sağlar.

@Qualifer Bağımlılık enjeksiyonunu @Bean ile nasıl enjekte edileceğini belirlemek

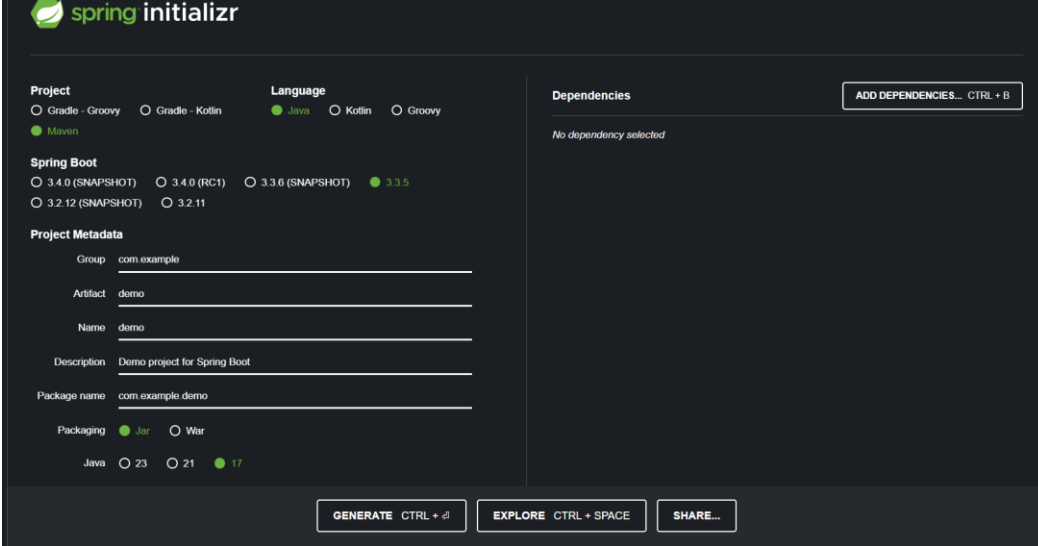
@Primary Öncelik sırasını belirlemek

@Lazy Bir @Bean 'nın tembel oluşturulmasını sağlar. İhtiyaç duyulduğunda
Spring Ioc Coytainer kısmına nesne oluşturur. Otomatik enjekte eder.

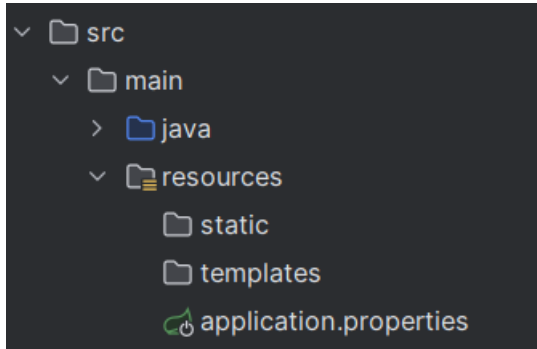
**ONAYLAYAN YETKİLİ****ADI SOYADI****UNVANI****İMZA**

-Springboot Katmanları

Spring Initializr, Spring Boot projelerini kolayca başlatmak ve oluşturmak için kullanılan bir araçtır. Web tabanlı bir arayüz sunarak geliştiricilerin ihtiyaçlarına göre bir Spring Boot projesini hızlıca import edip, çalışmaya başlamalarını sağlar. Böylece tüm bağımlılıkları manuel olarak eklemek yerine, birkaç adımda proje yapısı otomatik oluşturulur.



The image shows the Spring Initializr web interface. It has a dark theme. At the top left is the 'spring initializr' logo. Below it, there are sections for 'Project', 'Language', 'Spring Boot', 'Project Metadata', and 'Dependencies'. The 'Project' section has radio buttons for 'Gradle - Groovy', 'Gradle - Kotlin', and 'Maven' (selected). The 'Language' section has radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has radio buttons for '3.4.0 (SNAPSHOT)', '3.4.0 (RC1)', '3.3.6 (SNAPSHOT)', '3.3.5' (selected), and '3.2.12 (SNAPSHOT)'. The 'Project Metadata' section has input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo). There are also radio buttons for 'Packaging' (Jar selected, War) and 'Java' (23, 21, 17 selected). The 'Dependencies' section has a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.



application.properties Nedir?

Spring Boot'ta kullanılan adıyla application.properties, uygulamanın yapılandırma ayarlarını barındıran bir dosyadır. Spring Boot gibi çerçevelerde, uygulama ile ilgili birçok ayarın merkezi bir yerden yönetilmesi için kullanılır.

spring.jpa.properties.hibernate.dialect: Hibernate tarafından kullanılan SQL dialect'i belirlemek için kullanılır. Burada, PostgreSQL veritabanıyla uyumlu bir dialect belirtilmiştir.

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Hibernate'e PostgreSQL veritabanı komutlarını nasıl işleyeceğini bildirir. PostgreSQL'in SQL komutları ve özellikleri, diğer veritabanı türlerine göre farklılık gösterdiği için, PostgreSQLDialect kullanarak Hibernate'in PostgreSQL'e özgü SQL komutlarını üretmesi sağlanır.

ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

```
1  spring.application.name=ptt
2
3  spring.datasource.url=jdbc:postgresql://localhost:5432/project
4
5  server.port=3030
6  spring.datasource.username=postgres
7  spring.datasource.password=123456
8  spring.jpa.hibernate.ddl-auto=update
9  spring.datasource.driver-class-name=org.postgresql.Driver
10
```

update: Eğer tablolar yoksa oluşturur, ancak mevcut tabloları silmeden güncellemeler yapar. Geliştirme aşamasında kullanışlıdır.

create-drop: create gibi başlatırken veritabanını oluşturur, ancak uygulama kapanırken tüm veritabanı tablolarını siler.

none: Hibernate'e şema yönetimini devre dışı bırakır. Hibernate şema üzerinde hiçbir işlem yapmaz.

validate ayarı, Hibernate'in veritabanı şemasını kontrol etmesini sağlar, ancak veritabanında herhangi bir değişiklik yapmaz.

create ayarı, uygulama çalıştığında veritabanındaki mevcut tablo ve verileri siler ve yeniden oluşturur.

```
spring.jpa.hibernate.ddl-auto=|
spring.datasource.driver-c
```

- update (Update the schema if necessary)
- create (Create the schema and destroy previous data)
- create-drop (Create and then destroy the schema at the end of the...
- none (Disable DDL handling)
- validate (Validate the schema, make no changes to the database)

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards [Next Tip](#) ⋮

Spring Data JPA ve Entity Sınıfları:JPA (Java Persistence API), veritabanı işlemlerini Java sınıflarıyla yapmanı sağlar. Entity sınıfları ile tabloları modelleyebilirsin.

JPA REPOSITORY :Spring Data JPA'nın sağladığı ve veri erişim katmanını basitleştiren bir bileşendir.JPA Repository, genellikle veri tabanındaki bir tabloya denk gelen bir Entity sınıfıyla birlikte kullanılır.Aynı zamanda Rest Repositories bağımlılığı eklendiğinde db erişimi sağlamak için JPA repositories gibi yapıların otomatik olarak Restful web servislerine dönüştürülmesi sağlanır.

(Spring-boot-starter-data-rest)

Repository veri tabanları işlemleri için kullandığının arayüz sınıfıdır.

ONAYLAYAN YETKİLİ

ADI SOYADI	ÜNVANI	İMZA

TARİH:...../...../20.....		SAYFA NO:
<ul style="list-style-type: none">• public interface UserRepository extends JpaRepository<User, Long> <p>User, entity katmanını nasıl çalışacağını gösterir. Long, primary key veri tipi</p> <p>JPA Repository İçindeki Önemli Metotlar</p> <p>save(entity): Veriyi kaydeder veya günceller.</p> <p>findById(id): Belirtilen id'ye göre veri bulur.</p> <p>findAll(): Tüm kayıtları döner.</p> <p>deleteById(id): Belirtilen id'deki veriyi siler.</p> <p>count(): Toplam kayıt sayısını döner.</p> <p>Not:</p> <p>Bir sınıf, başka bir sınıfı extends anahtar kelimesi ile genişletir ve miras alır. Interface, çoklu kalıtım sağlamak veya birden fazla sınıfın ortak davranışlarını belirlemek için kullanılır. Java'da bir sınıf, bir veya daha fazla interface'in tanımladığı metodları kullanmak zorunda kalırsa, "implements" anahtar kelimesi ile interface'i uygular.</p> <p>Nasıl Çalışır:Spring Data JPA, Java sınıflarını (Entity'leri) otomatik olarak veritabanı tablolarına dönüştürür ve temel CRUD işlemlerini sağlar. Repository arayüzünü yazdığında, Spring Boot senin için bu arayüzün bir implementasyonunu (altyapı kodunu) otomatik olarak üretir.</p>		
ONAYLAYAN YETKİLİ		
ADI SOYADI	ÜNVANI	İMZA

TARİH:...../...../20.....		SAYFA NO:
<ul style="list-style-type: none">Veri Ekleme (Create): User user = new User("Irem", "irem@example.com"); userRepository.save(user);Tüm Kayıtları Getirme (Read): List<User> users = userRepository.findAll();ID'ye Göre Tek Kayıt Getirme: Optional<User> user = userRepository.findById(1L); user.ifPresent(System.out::println);Kayıt Güncelleme (Update): User user = userRepository.findById(1L).orElseThrow(); user.setName("Yeni İsim"); userRepository.save(user); // Güncellenmiş veriyi kaydetKayıt Silme (Delete): userRepository.deleteById(1L); <p>Not:</p> <p>Optional<TeslimSureleri>: Optional, bir değerin var olup olmadığını kontrol etmemize olanak tanır. Optional<TeslimSureleri> döndüren bir metot, TeslimSureleri nesnesinin null olma ihtimaline karşı önlem almak için kullanılır. Eğer TeslimSureleri nesnesi bulunamazsa, Optional.empty() döner, böylece NullPointerException'dan kaçınılmış olur.</p> <p>findBy: Bu kısım, arama yönteminin başlangıcını ifade eder.</p> <p>Service Katmanı Oluşturma:İş mantığını yöneten Service katmanı.Service katmanı (Hizmet Katmanı), yazılım mimarisinde çok önemli bir role sahiptir. Bu katman, uygulamanın iş kurallarını (business logic) içerir ve veritabanı işlemleri ile sunum (controller) katmanı arasında bir köprü görevi görür.</p>		
ONAYLAYAN YETKİLİ		
ADI SOYADI	ÜNVANI	İMZA

REST Controller Katmanı:API isteklerini işleyen Controller sınıfı.Controller katmanı, Spring Boot projelerinde web isteklerini (HTTP) alan ve cevaplayan katmandır. Uygulamanın frontend kısmıyla iletişim kurar (örn. kullanıcıdan gelen istekler) ve bu istekleri Service katmanına yönlendirir. Controller, sunum katmanının (presentation layer) bir parçası olup REST API işlemleri gibi görevleri üstlenir.

//src

|

|— /main

| |— /java

| | |— com.example.demo

| | |— User.java // Entity Sınıfı

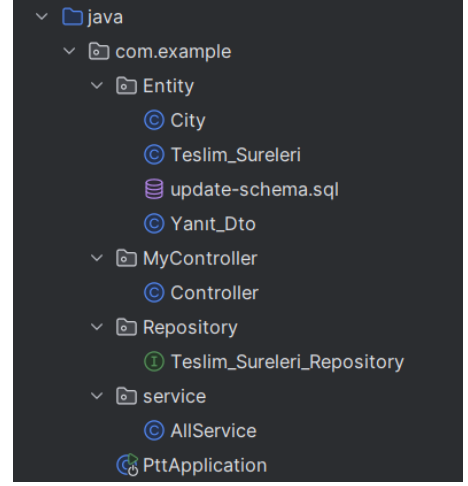
| | |— UserRepository.java // Repository Arayüzü

| | |— UserService.java // Service Katmanı

| | |— UserController.java // Controller Katmanı

| |— /resources

| |— application.properties // Konfigürasyon Dosyası



ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

TARİH:...../...../20.....		SAYFA NO:
<p>Unit Test (Birim Testi):Küçük ve bağımsız kod parçalarını, genellikle bir metod veya sınıf seviyesinde test eder.Amaç: Her bir fonksiyonun veya metodun doğru çalıştığını garanti altına almak.Hızlıdır ve iş kurallarına odaklanır. Veritabanı, servis veya API gibi dış bağımlılıklar mock nesnelerle izole edilir.</p> <p>Mocking Kullanımı: Mockito gibi araçlarla yapılır. Örneğin, UserService sınıfında UserRepository'yi mock'layarak test gerçekleştirilir.</p> <ul style="list-style-type: none">• @Test void shouldAddUserSuccessfully() { User user = new User(); user.setEmail("irem@example.com"); when(userRepository.findByEmail("irem@example.com")).thenReturn(null); when(userRepository.save(user)).thenReturn(user); User savedUser = userService.addUser(user); assertEquals("irem@example.com", savedUser.getEmail()); } <p>Mocking ile, dış sistemlere erişmeden metodun iş mantığı izole bir şekilde test edilir.</p> <p>Integration Test (Entegrasyon Testi):Farklı katmanların birlikte çalışmasını test eder. Örneğin, Controller, Service, ve Repository katmanları arasındaki uyumu doğrular.Gerçek sistemler ve veritabanı kullanılır, bu nedenle daha yavaştır.@SpringBootTest anotasyonu ile yapılır.</p> <ul style="list-style-type: none">• @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT) public class UserControllerIntegrationTest { @Autowired private TestRestTemplate restTemplate; @Test void shouldCreateUserSuccessfully() { User newUser = new User(); newUser.setName("Irem"); newUser.setEmail("irem@example.com"); ResponseEntity<User> response = restTemplate.postForEntity("/users", newUser, User.class); assertEquals(HttpStatus.OK, response.getStatusCode()); assertNotNull(response.getBody().getId()); } } <p>Bu test, tüm sistemin uyumlu çalıştığını doğrular ve gerçek veritabanı bağlantısı kullanır.</p>		
ONAYLAYAN YETKİLİ		
ADI SOYADI	ÜNVANI	İMZA

Fonksiyonel Test (Functional Testing):Yazılımın işlevlerinin gereksinimlere uygun olarak çalışıp çalışmadığını kontrol eder.Araçları Selenium, Postman, SoapUI gibi araçlarla yapılır.Örneğin, kullanıcı kayıt formunun tüm gerekli bilgileri alarak başarıyla kayıt yapmasını test eder.

Stres Testi (Stress Testing):Sistem aşırı yük altında çalıştığında nasıl davrandığını kontrol eder.Örneğin, e-ticaret sitesine çok sayıda kullanıcı giriş yaptığında yanıt süresi ve hata oranları gözlemlenir.

Ölçeklenebilirlik Testi (Scalability Testing):Sistemin artan iş yükü ve veri hacmi karşısında performansını koruyup koruyamadığını test eder.Kullanıcı sayısının veya veri hacminin artması durumunda performansın nasıl etkilendiği gözlemlenir.

Yük Testi (Load Testing):Sistemin beklenen normal yük altında nasıl performans gösterdiğini ölçer. Örneğin, bankacılık uygulamasında aynı anda 1000 kişinin işlem yapmasını simüle ederek performansı gözlemleriz.

Unit Test ile Integration Test Farkları

Özellik	Unit Test	Integration Test
Test Edilen Kapsam	Tekil metodlar veya sınıflar	Birden fazla katmanın birlikte çalışması
Bağımlılıklar	Mock nesneler kullanılır	Gerçek sistemler ve veritabanı kullanılır
Çalışma Hızı	Çok hızlı	Daha yavaş
Test Türü	İzole edilmiş iş mantığı testleri	Uygulama bütünlüğünü test eder
Örnek Kullanım	Service sınıfının bir metodunu test etmek	Controller, Service ve Repository uyumunu test etmek

Veritabanı Test Türleri

Fonksiyonel Testler: CRUD işlemlerinin doğru çalıştığını kontrol eder.

Performans Testleri: Sorguların hızlı ve optimize çalışıp çalışmadığını kontrol eder.

Veri Bütünlüğü Testleri: Verinin doğru ve eksiksiz işlendiğini doğrular.

Transaction Testleri: İşlemler arasında veri tutarlılığını kontrol eder.

ONAYLAYAN YETKİLİ

ADI SOYADI

UNVANI

İMZA

TARİH:...../...../20.....	SAYFA NO:	
<pre>• @SpringBootTest public class UserRepositoryTest { @Autowired private UserRepository userRepository; @Test void shouldSaveUser() { User user = new User(1L, "Irem", "irem@example.com"); userRepository.save(user); Optional<User> savedUser = userRepository.findById(1L); assertTrue(savedUser.isPresent()); assertEquals("Irem", savedUser.get().getName()); } }</pre>		
ONAYLAYAN YETKİLİ		
ADI SOYADI	ÜNVANI	İMZA

Proje—Ptt projesi: Ptt Kargo Varış günü projesi

Adım 1: PostgreSQLde Tablo oluşturma.



- CREATE TABLE *table_name* (
column1 datatype,
column2 datatype,
column3 datatype,
....
);

public
teslim_sureleri
gun integer
cut_off_time time without time zone
kabul_ili integer
varis_ili integer
gonderi_turu character varying(255)
id integer (IDENTITY)
varis_gunu character varying(255)

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
gun	integer			<input type="checkbox"/>	<input type="checkbox"/>	
cut_off_time	time without time zone			<input type="checkbox"/>	<input type="checkbox"/>	
kabul_ili	integer			<input type="checkbox"/>	<input type="checkbox"/>	
varis_ili	integer			<input type="checkbox"/>	<input type="checkbox"/>	
gonderi_turu	character varying	255		<input type="checkbox"/>	<input type="checkbox"/>	
id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
varis_gunu	character varying	255		<input type="checkbox"/>	<input type="checkbox"/>	

Adım 2: teslim_sureleri.csv içindeki verileri teslim_sureleri tablosuna import etme.

- teslim_sureleri.xlsx dosyasını teslim_sureleri.csv dosyasına dönüştürme.

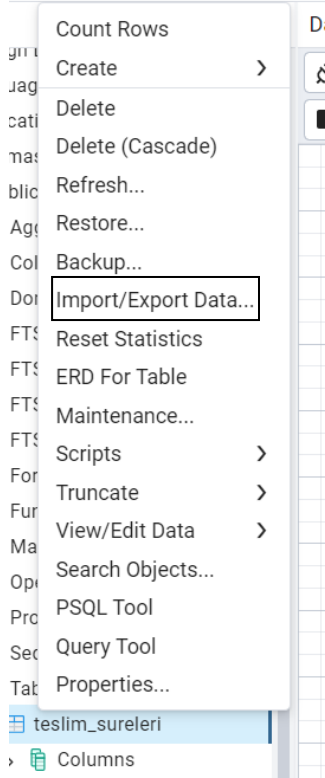
-  teslim_sureleri.xlsx
-  teslim_sureleri.csv

ONAYLAYAN YETKİLİ

ADI SOYADI

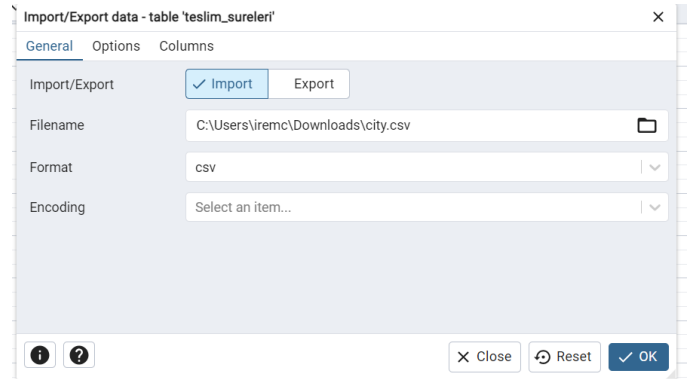
ÜNVANI

İMZA



teslim_sureleri.csv içindeki verilerin teslim_sureleri tablosuna import etmek için sağ click yapıyoruz; Import/Export Data... butonunu seçiyoruz.

- 1) Filename kısmından aktaracağımız teslim_sureleri.csv dosyasını seçiyoruz.



ONAYLAYAN YETKİLİ

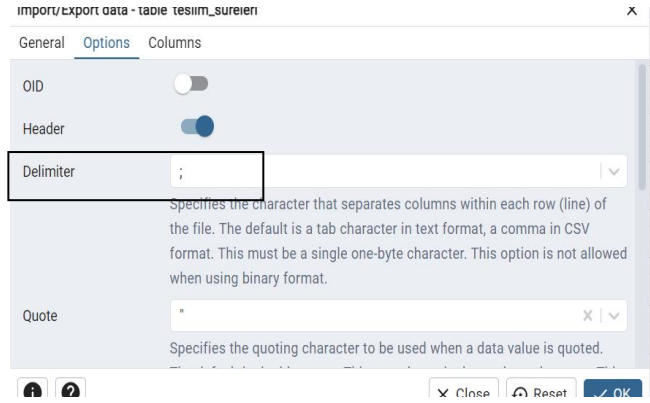
ADI SOYADI

ÜNVANI

İMZA

2) Options tabını seçiyoruz.

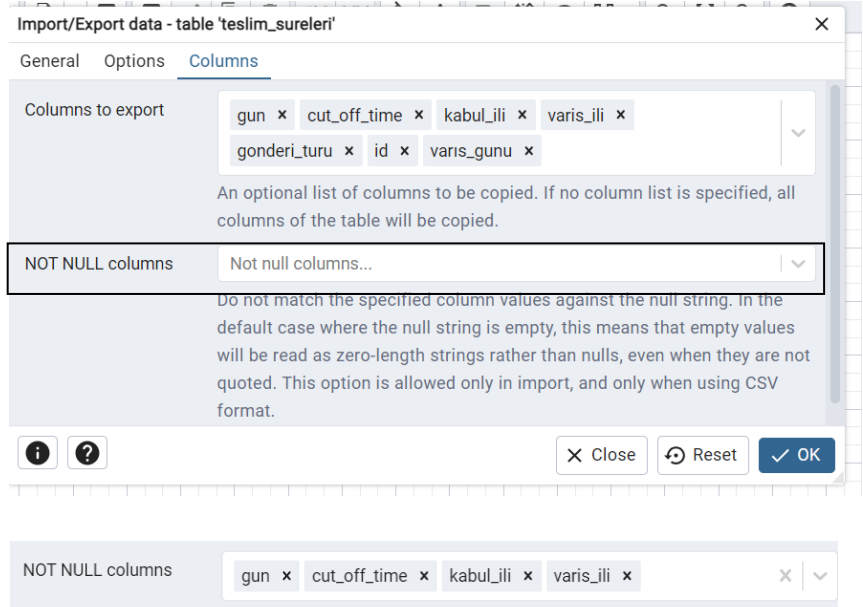
- Delimiter kısmını teslim_sureleri.csv dosyasında **1;17:30;1;2;APG** delimiter'in “;” olduğunu görüyoruz. Bu yüzden delimiter kısmını “;” seçiyoruz.



3)Columns tabını

seçiyoruz.

NOT NULL columns kısmını dataları teslim_sureleri columnna importlayacağımızı seçiyoruz. Ok butonuna basıyoruz.



ONAYLAYAN YETKİLİ

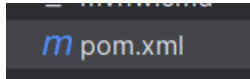
ADI SOYADI

ÜNVANI

İMZA

Not:

- Dependencies : Spring Boot projelerinde yaygın olarak kullanılan, belirli işlevler veya özellikler ekleyen kitaplık ve araçlardır.
Lombok, kodunuzu daha temiz ve kısa hale getirmek için kullanılan bir Java kütüphanesidir. Getter, Setter, toString, Equals, ve Constructor gibi metotları elle yazmanıza gerek kalmadan, yalnızca anotasyonlar (@Getter, @Setter, @ToString, @EqualsAndHashCode, @NoArgsConstructor, @AllArgsConstructor gibi) ile otomatik olarak oluşturulmasını sağlar.
Spring Data JPA, veri tabanı erişim işlemlerini kolaylaştıran bir kütüphanedir. JPA (Java Persistence API) ile Spring'in özelliklerini birleştirerek, veri tabanı işlemlerini basitleştirir. JpaRepository gibi arayüzleri kullanarak SQL sorgusu yazmadan, veri tabanı ile hızlıca etkileşime geçebiliriz.
PostgreSQL veri tabanına bağlantı sağlamak için gerekli olan JDBC (Java Database Connectivity) sürücüsüdür.
Spring Web, HTTP tabanlı web uygulamaları geliştirmenizi sağlayan bir modüldür. RESTful API'ler oluşturmak ve web sayfaları sunmak için kullanılır.



```
<java.version>17</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
</dependencies>
```

ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

Adım 4: PostgreSQL Bağlantısını Yapılandırma

application.properties dosyasında PostgreSQL bağlantısını yapılandırma:

```

application.properties x
1  spring.application.name=ptt
2
3  spring.datasource.url=jdbc:postgresql://localhost:5432/project
4
5  server.port=3030
6  spring.datasource.username=postgres
7  spring.datasource.password=123456
8  spring.jpa.hibernate.ddl-auto=update
9  spring.datasource.driver-class-name=org.postgresql.Driver
10

```

- spring.application.name=ptt

Bu ayar, uygulamanıza bir isim vermek için kullanılır. Burada uygulamanın adı "ptt" olarak belirlenmiş. Bu, uygulama loglarında ve diğer Spring modüllerinde uygulama adının görüntülenmesini sağlar.

- spring.datasource.url=jdbc:postgresql://localhost:5432/project:

Bu ayar, PostgreSQL veri tabanına bağlanmak için kullanılan URL'yi belirtir.

- jdbc:postgresql://localhost:5432/project ifadesi:

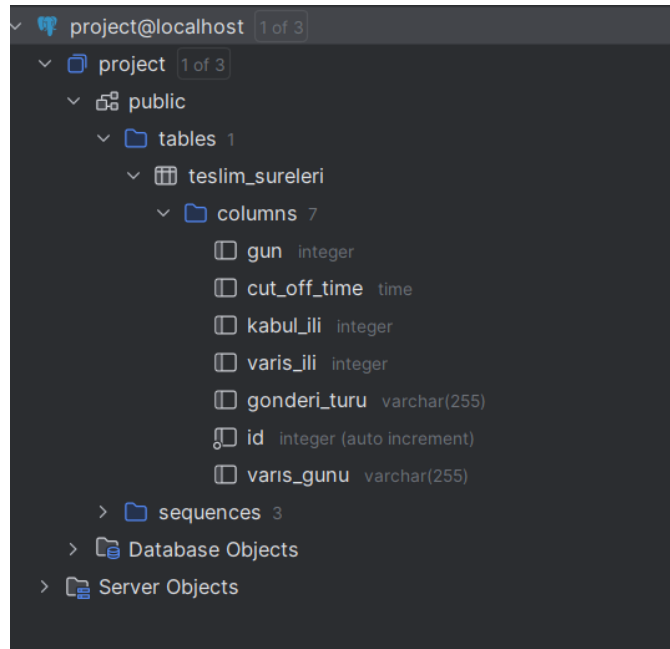
jdbc:postgresql://: JDBC bağlantısı için PostgreSQL protokolü.

localhost: Veri tabanı sunucusunun adresi.

localhost kendi bilgisayarınızda çalışan veri tabanına bağlanmak için kullanılır.

5432: PostgreSQL veri tabanının varsayılan port numarası.

project: Bağlanılacak veri tabanının adı. Burada "project" adlı veri tabanı kullanılacak.



ONAYLAYAN YETKİLİ

ADI SOYADI	ÜNVANI	İMZA

server.port=3030:

Spring Boot uygulamasının çalışacağı port numarasını belirtir. Varsayılan port 8080'dir, ancak burada 3030 olarak değiştirilmiş. Bu, uygulamanın <http://localhost:3030> adresinde çalışacağı anlamına gelir.

spring.datasource.username=postgres:

Veri tabanına bağlanmak için kullanılacak kullanıcı adını belirtir. Burada kullanıcı adı postgres olarak tanımlanmış.

spring.datasource.password=123456:

Veri tabanına bağlanmak için kullanılan kullanıcı adı postgres'in şifresini belirtir. Burada 123456 olarak tanımlanmış (örnek olarak verilmiş).

spring.jpa.hibernate.ddl-auto=update:

Bu ayar, Hibernate'in veri tabanı şeması üzerindeki işlem davranışını belirler. "update" değeri, uygulama her başlatıldığında veri tabanındaki mevcut şemayı günceller.

- Not:

create: Her başlatmada tabloyu sıfırdan oluşturur (tüm veriler silinir).

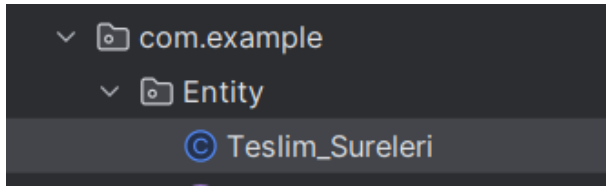
create-drop: Uygulama kapanırken tabloyu siler.

none veya validate: Veri tabanı şemasını değiştirmez, yalnızca mevcut yapıyı doğrular.

spring.datasource.driver-class-name=org.postgresql.Driver:

PostgreSQL veri tabanına bağlanmak için kullanılan sürücünün (driver) sınıf adını belirtir. Bu sürücü, Spring Boot'un PostgreSQL veri tabanı ile iletişim kurmasını sağlar.

Adım 5: Entity Sınıflarını Oluşturma:



-Not: Entity, veritabanında bir tabloyu temsil eden bir sınıftır. ORM ile tablo-satır yapısına nesne yönelimli erişim sağlar. @Entity tabloyu, @Id birincil anahtarı belirtir.

ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

1. TeslimSureleri tablosu için:

```
@Getter 8 usages
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name="teslim_sureleri")

public class Teslim_Sureleri {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    @Column(name = "gun")
    private int gun;
    @Column(name = "cut_off_time")
    private LocalDateTime cut_off_time;
    @Column(name = "kabul_ili")
    private int kabul_ili;
    @Column(name = "varis_ili")
    private int varis_ili;
    @Column(name = "gonderi_turu")
    private String gonderi_turu;
    @Column(name = "varis_gunu")
    private String varis_gunu;
```

Lombok Anotasyonları:

@Getter: Tüm alanlar için otomatik olarak getter (alma) metodları oluşturur.

@Setter: Tüm alanlar için otomatik olarak setter (atma) metodları oluşturur.

@AllArgsConstructor: Tüm alanları içeren bir yapıcı metod (constructor) oluşturur.

@NoArgsConstructor: Parametresiz bir yapıcı metod oluşturur.

JPA ve Entity Anotasyonları:

@Entity: Bu sınıfın bir veritabanı tablosunu temsil ettiğini belirtir.

@Table(name="teslim_sureleri"): Veritabanında bu sınıfın "teslim_sureleri" adlı tabloya karşılık geldiğini belirtir.

Alanlar ve Anotasyonları:

@Id: id alanını tablonun birincil anahtarı olarak tanımlar.

@GeneratedValue(strategy = GenerationType.AUTO): id alanının otomatik olarak üretilmesini sağlar.

@Column(name = "gun"): gun adlı alanı "gun" adlı sütunla eşleştirir.

cut_off_time: Son teslim zamanı olarak LocalDateTime türünde bir alan. "cut_off_time" adlı sütunla eşleştirilir.

kabul_ili ve varis_ili: Kabul ve varış illerini temsil eden alanlar.

gonderi_turu: Gönderi türünü temsil eder.

varis_gunu: Varış günü bilgisi, metin olarak saklanır.

- Not:

AUTO: Varsayılan stratejiyi kullanır (genellikle Hibernate'e göre belirlenir).

IDENTITY: Veri tabanının otomatik artan değer özelliğini kullanır.

SEQUENCE: Özellikle sıralar (sequence) kullanarak kimlik oluşturur, PostgreSQL gibi veri tabanları için yaygındır.

TABLE: Kimlik değerlerini tutmak için bir tablo oluşturur.

```
@Id
@GeneratedValue(strategy = GenerationType.)
private int id;
@Column(name = "gun")
private int gun;
@Column(name = "cut_off_time")
private LocalDateTime cut_off_time;
@Column(name = "kabul_ili")
```

AUTO
IDENTITY
SEQUENCE
UUID
TABLE

ONAYLAYAN YETKİLİ

ADI SOYADI

UNVANI

İMZA

Adım 6: Repository'leri Tanımlama

▼ Repository

Teslim_Sureleri_Repository

Spring Data JPA kullanarak veri tabanından veri çekmek için Repository'leri tanımlıyoruz.

1.Teslim_Sureleri_Repository:

```
public interface Teslim_Sureleri_Repository extends JpaRepository<Teslim_Sureleri, Integer> {  
    @Query("SELECT t FROM Teslim_Sureleri t WHERE t.varis_ili = :varis_ili AND t.kabul_ili = :kabul_ili")  
    List<Teslim_Sureleri> findByVaris_iliAndKabul_ili(int varis_ili, int kabul_ili);  
}
```

-public interface Teslim_Sureleri_Repository extends JpaRepository
<Teslim_Sureleri, Integer>:

Bu satır, Teslim_Sureleri_Repository adlı bir repository arayüzünü tanımlar ve JpaRepository arayüzünü genişletir.

JpaRepository<Teslim_Sureleri, Integer> ifadesi, Teslim_Sureleri entity'si için temel CRUD (Create, Read, Update, Delete) işlemlerini ve diğer veritabanı işlemlerini sağlar.

Integer ise Teslim_Sureleri entity'sinde kullanılan birincil anahtarın veri türünü belirtir (bu örnekte id alanı int olduğu için Integer kullanılmıştır).

-@Query("SELECT t FROM Teslim_Sureleri t WHERE t.varis_ili = :varis_ili AND t.kabul_ili = :kabul_ili"):

Bu anotasyon, özel bir JPQL (Java Persistence Query Language) sorgusu tanımlar.

SELECT t FROM Teslim_Sureleri t ifadesi, Teslim_Sureleri tablosundan tüm sütunları t takma adıyla seçer.

WHERE t.varis_ili = :varis_ili AND t.kabul_ili = :kabul_ili: varis_ili ve kabul_ili alanlarının, metod parametreleri olan :varis_ili ve :kabul_ili değerlerine eşit olduğu kayıtları filtreler.

findByVaris_iliAndKabul_ili(int varis_ili, int kabul_ili):

Bu metod, varis_ili ve kabul_ili parametrelerine göre Teslim_Sureleri kayıtlarını döndürür.

List<Teslim_Sureleri>, yani varis_ili ve kabul_ili değerlerine uygun tüm Teslim_Sureleri kayıtlarını bir liste olarak döndürür.

-Not:

Optional<Teslim_Sureleri>findByTeslim_SureleriByVaris_iliAndKabul_ili(int varis_ili, int kabul_ili) ; Optional<Teslim_Sureleri>: Sonucun var olup olmadığını kontrol etmek için kullanılır; eğer kayıt bulunamazsa Optional boş döner.

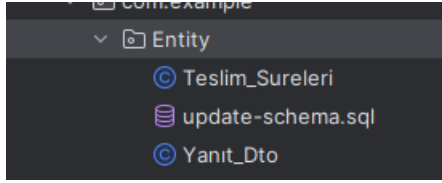
ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

Adım 7: Yanıt_Dto classını oluşturma:



Yanıt_Dto sınıfı, Teslim_Sureleri entity'sindeki verilerin yalnızca belirli bir kısmını istemciye aktarmak için kullanılıyor. Yanıt_Dto, yalnızca gun, varis_gunu, gonderi_turu ve cut_off_time alanlarını içerir; veritabanı tablosunun diğer alanları bu DTO'da yer almaz. Böylece veri transferi daha az karmaşık hale gelir.

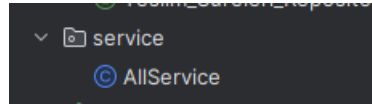
```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDateTime;

@Getter 5 usages
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Yanıt_Dto {
    // private int id;
    private int gun;
    private String varis_gunu;

    private String gonderi_turu;
    private String cut_off_time;
}
```

Adım 8: Service Katmanını Oluşturma:



- İş kurallarını ve sorguları yazacağın Service katmanı.
- Bu sınıf, @Service anotasyonu ile işaretlenmiş bir Spring servisi olarak tanımlanmıştır. Servis, iş mantığını uygulamak için kullanılan bir katmandır ve burada Teslim_Sureleri verileriyle etkileşimde bulunmak için bir repository (veri erişim arayüzü) kullanmaktadır.

```
import java.util.stream.Collectors;

@Service 3 usages
public class AllService {
    private final Teslim_Sureleri_Repository teslim_Sureleri_Repository; 2 usages

    public AllService(Teslim_Sureleri_Repository teslim_Sureleri_Repository) {
        this.teslim_Sureleri_Repository = teslim_Sureleri_Repository;
    }
}
```

ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

teslim_Sureleri_Repository: Bu alan, Teslim_Sureleri tablosuna erişmek için kullanılan bir repository nesnesidir. Teslim_Sureleri_Repository nesnesini alır ve sınıfın alanına atar. Bu sayede, sınıf içinde bu repository aracılığıyla veritabanına erişim sağlanır.

- Not:

Constructor:

```
public AllService(Teslim_Sureleri_Repository teslim_Sureleri_Repository) { ... }
```

Bu yapıcı metod, AllService sınıfının bir örneği oluşturulurken

Teslim_Sureleri_Repository nesnesinin dışarıdan geçirilmesini sağlar. Bu, bağımlılığı AllService sınıfının dışında oluşturmak anlamına gelir.

Bu kullanım, Dependency Injection (DI) prensibini yansıtır. DI, bir nesnenin bağımlılıklarını (bu örnekte Teslim_Sureleri_Repository) kendisinin oluşturmak yerine dışarıdan alması anlamına gelir. Bu yaklaşım, sınıfların daha esnek, test edilebilir ve bakımı daha kolay hale gelmesini sağlar.

```
public List<Yanıt_Dto> getTeslim_Sureleri(int varis_ili,int kabul_ili) { 1 usage
    List<Teslim_Sureleri> sureleri=teslim_Sureleri_Repository.findByVaris_iliAndKabul_ili(varis_ili,kabul_ili);

    return sureleri.stream().map(t -> new Yanıt_Dto(
        t.getGun(),
        hesaplaVarisGunu(t.getGun(), t.getCut_off_time()),
        t.getGonderi_turu(),
        t.getCut_off_time().toString()))
        .collect(Collectors.toList());
}
```

sureleri.stream():

-sureleri listesi, Teslim_Sureleri nesnelerinin bir listesidir. stream() metodu, bu liste üzerinden bir akış (stream) oluşturur. Bu akış, elemanlar üzerinde işlem yapmayı mümkün kılar.

map(...):

-map metodu, akıştaki her bir elemanı (her bir Teslim_Sureleri nesnesi) Yanıt_Dto nesnesine dönüştürmek için kullanılır. Her Teslim_Sureleri nesnesi için, yeni bir Yanıt_Dto nesnesi oluşturulur. Bu nesne, gun, varis_günü, gonderi_turu, ve cut_off_time değerlerini alır.

ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

collect(Collectors.toList());

-collect metodu, akıştaki elemanları bir koleksiyona (bu durumda bir listeye) toplamak için kullanılır. Collectors.toList() ifadesi, akıştaki elemanların bir List (liste) olarak toplanmasını sağlar. Yani, map metoduyla oluşturulan Yanıt_Dto nesneleri, bir List<Yanıt_Dto> nesnesine dönüştürülür.

```
private String hesaplaVarisGunu(int gun, LocalTime cutOffTime) { 1usage
    List<String> days = Arrays.asList(
        "Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma", "Cumartesi"
    );

    LocalTime seventeenOClock = LocalTime.of( hour: 17, minute: 0); // 17:00
    LocalDate today = LocalDate.now(); // Bugünün tarihi

    // Eğer şu anki zaman 17:00'dan önceyse, bugünün adını döndürüyoruz.
    if (cutOffTime.isBefore(seventeenOClock)) {
        int currentDayIndex = today.getDayOfWeek().getValue() - 1; // DayOfWeek enum'ı Pazartesi için 1 döner
        int calculatedIndex = (currentDayIndex + gun) % 6; // 0-5 arasında bir değer alır (Pazar hariç)
        return days.get(calculatedIndex);
    } else {
        // 17:00'den sonra ise ertesi gün olacak şekilde günü hesaplıyoruz.
        int nextDayIndex = (today.getDayOfWeek().getValue()) % 6; // Ertesi günün index'i (Pazar hariç)
        int calculatedIndex = (nextDayIndex + gun) % 6; // 0-5 arasında bir değer alır (Pazar hariç)
        return days.get(calculatedIndex);
    }
}
```

Eğer kesim saati (cut-off time) 17:00'dan önceyse, gün hesaplaması bugünkü günden yapılır; 17:00'dan sonra ise ertesi gün üzerinden hesaplama yapılır. Günlerin indeksi (0-5) arasında hesap yapılarak Pazar gününe karşılık gelen değer atlanır.

- Not;

```
public Optional<Teslim_Sureleri> GetTeslim(int varis_ili, int kabul_ili) {
    return
    teslim_Sureleri_Repository.queryTeslim_SureleriByVaris_iliAndKabul_ili(varis_ili,
    kabul_ili);
}
```

Bu metot, belirli bir varis_ili ve kabul_ili için bir Teslim_Sureleri nesnesini bulmayı amaçlar.

queryTeslim_SureleriByVaris_iliAndKabul_ili(...):Bu, muhtemelen bir sorgu metodu olup, belirtilen varis_ili ve kabul_ili değerlerine göre veritabanından bir Teslim_Sureleri nesnesi döndürür.

```
public List<Teslim_Sureleri> GetTeslim(int varis_ili, int kabul_ili) { return
    teslim_Sureleri_Repository.findByVaris_iliAndKabul_ili(varis_ili, kabul_ili); }
```

Bu metot da belirli bir varis_ili ve kabul_ili için Teslim_Sureleri nesnelerini bulmayı amaçlar.

ONAYLAYAN YETKİLİ

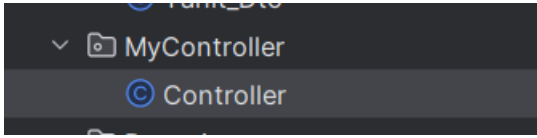
ADI SOYADI

UNVANI

İMZA

Adım 9:Controller Sınıfını Yazma:

- API'den gelen istekleri yönetecek Controller katmanı.



```
@RestController
@RequestMapping(path = @"/api")
public class Controller {
    private final AllService allService; 2 usages

    public Controller(AllService allService) {
        this.allService = allService;
    }

    @GetMapping(path = @"/")
    public List<Yanıt_Dto> getTeslimSureleri(@RequestBody Teslim_Sureleri teslimSureleri) {
        return allService.getTeslim_Sureleri(teslimSureleri.getVaris_ili() ,
        teslimSureleri.getKabul_ili());
    }
}
```

private final AllService allService;

- Amaç: Bu satır, AllService türünde bir nesne tanımlar. final anahtar kelimesi, bu değişkenin yalnızca bir kez atanabileceğini belirtir; böylece değişkenin değeri daha sonra değiştirilemez.
- Bağımlılık: allService, kontrolörde kullanılacak olan servis sınıfını temsil eder. Bu, Dependency Injection (bağımlılık enjeksiyonu) kullanarak oluşturulmuştur, yani kontrolör nesnesi oluşturulurken AllService nesnesi dışarıdan sağlanır.

public Controller(AllService allService) { ... }

- Amaç: Bu, kontrolörün yapıcı (constructor) metodudur. Servis nesnesini (allService) alır ve sınıfın allService değişkenine atar.
- Dependency Injection: Spring framework, bu yapıcı metodun çağırılması sırasında AllService nesnesini otomatik olarak sağlar. Bu sayede kontrolör, AllService fonksiyonlarına erişebilir.

ONAYLAYAN YETKİLİ

ADI SOYADI

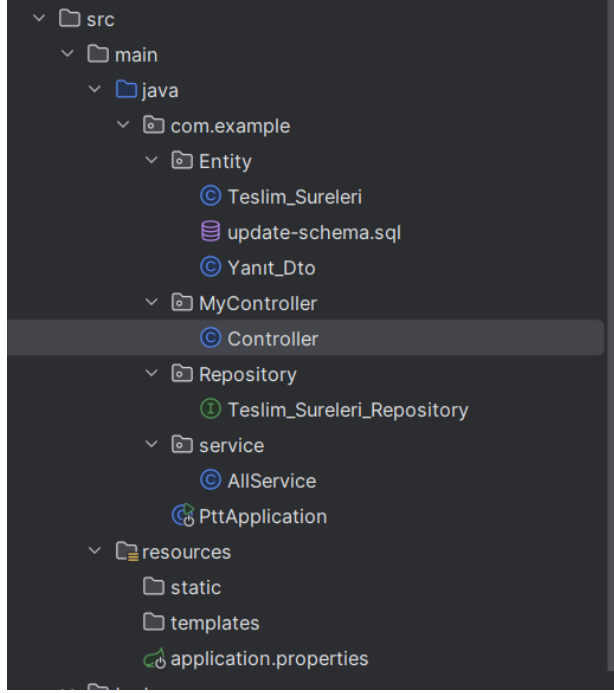
ÜNVANI

İMZA

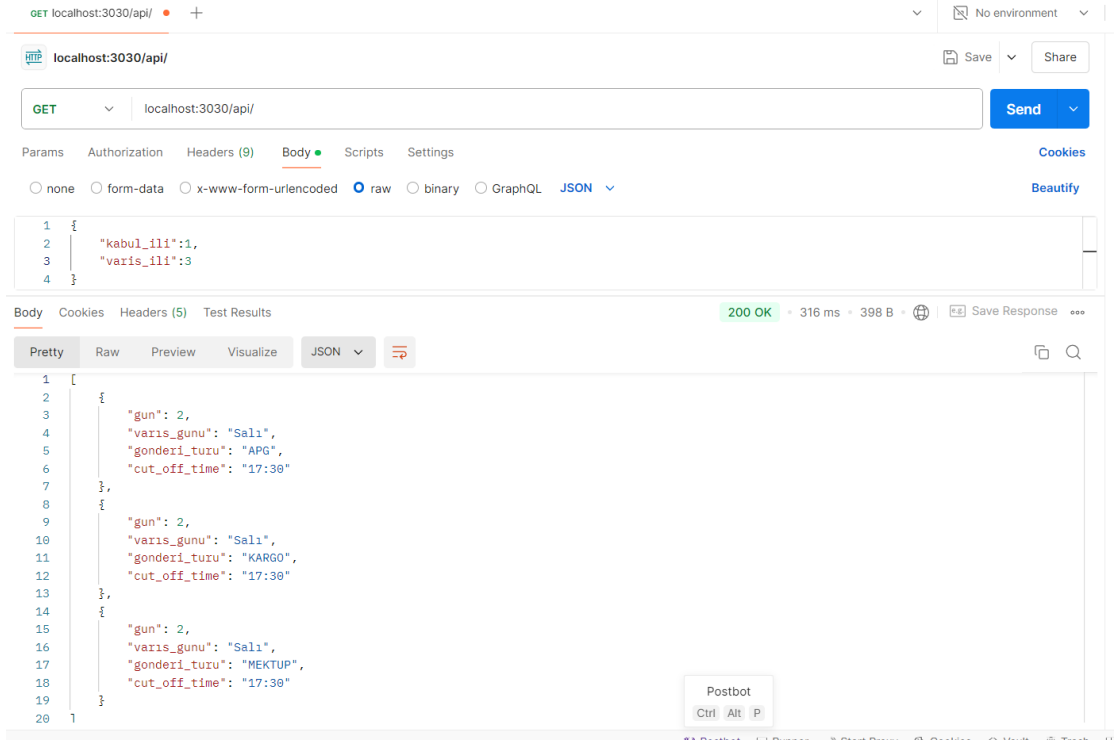
TARİH:...../...../20.....		SAYFA NO:
<p>@GetMapping(path = "/")</p> <ul style="list-style-type: none"> Amaç: Bu, HTTP GET isteği alındığında çalışacak olan metodun üzerindeki bir notasyondur. path = "/" belirterek, ana URL (örneğin, http://localhost:3030/) için GET isteği yapıldığında bu metodun çağrılacağını belirtir. İstemci İsteği: Kullanıcı ana URL'ye eriştiğinde, bu metod tetiklenecek ve ilgili işlemler gerçekleştirilir. <pre>public List<Yanıt_Dto> getTeslimSureleri(@RequestBody Teslim_Sureleri teslimSureleri) { ... }</pre> <ul style="list-style-type: none"> Amaç: Bu metod, HTTP GET isteğiyle çağrıldığında çalışır ve Teslim_Sureleri nesnesini alır. Parametre: <ul style="list-style-type: none"> @RequestBody: Bu notasyon, istemciden gelen verinin HTTP isteği gövdesinde yer aldığını belirtir. Yani, istemciden gönderilen JSON formatındaki Teslim_Sureleri nesnesi, otomatik olarak bu metoda parametre olarak geçirilir. Dönüş Tipi: List<Yanıt_Dto> döndürür. Bu, metodun sonucunun bir Yanıt_Dto nesneleri listesi olacağını belirtir. <pre>return allService.getTeslim_Sureleri(teslimSureleri.getVaris_ili(), teslimSureleri.getKabul_ili());</pre> <ul style="list-style-type: none"> İşlem: allService nesnesinin getTeslim_Sureleri metodunu çağırarak, istemciden alınan teslimSureleri nesnesinin varis_ili ve kabul_ili değerlerini geçirir. Sonuç: Servis metodundan dönen Yanıt_Dto listesi, HTTP yanıtı olarak istemciye geri döner. 		
ONAYLAYAN YETKİLİ		
ADI SOYADI	ÜNVANI	İMZA

TARİH:...../...../20.....

SAYFA NO:



→Output:Postmana **request** atıyoruz **reponse** alıyoruz.



ONAYLAYAN YETKİLİ

ADI SOYADI

ÜNVANI

İMZA

