AVS BLOG SPRING BOOT PROJESİ

Hazırlayan : Ali Volkan Şahin

Bölüm : Java-12 Bilge Adam Boost Programı

Proje Açıklaması :

Kullanıcıların blog yazabileceği, yazıları okuyabileceği, yorum yapabileceği ve aynı zamanda birbirleri arasındaki takip isteklerini gerçekleştirebileceği, yorum veya yazıları beğenebileceği ve bunları favorilerine ekleyebileceği bir blog platformu, arama ve kategorilendirme işlemlerini destekleyecek şekilde Restful API tabanlı ve monolotik mimariye sahip bir spring boot projesi olarak hazırlanmıştır.

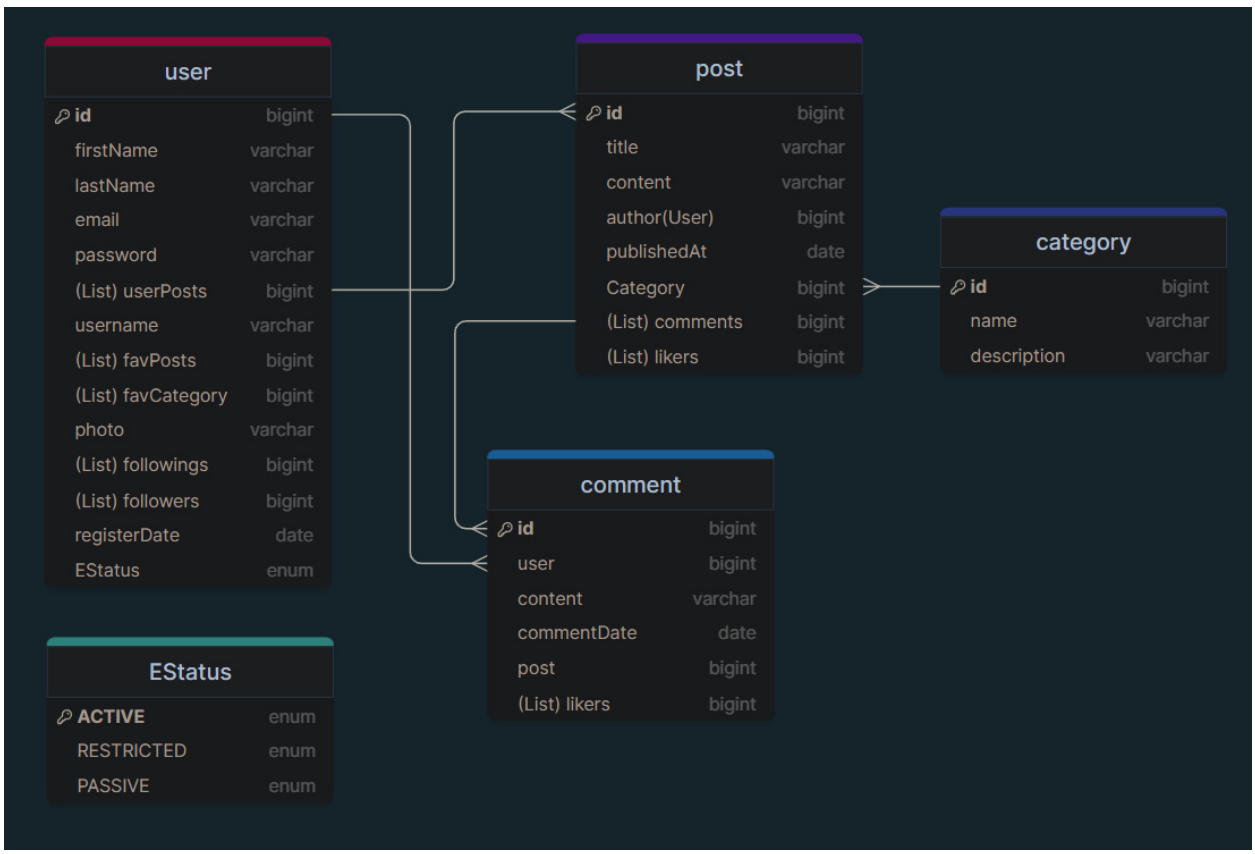
Bunun için Java 17 sürümü kullanılmış, build türü olarak Gradle-Groovy yapısı seçilmiştir.

Proje için kullanılan bağımlılıklar aşağıda belirtilmiştir :

- Spring Boot DevTools
- Lombok
- Spring Web
- Spring Data JPA (Hibernate)
- PostgreSQL Driver
- Mapstruct Core & Mapstruct Processor (Dto ve Entity sınıfları arasında eşleştirme için)
- SpringDoc OpenAPI UI (Swagger-UI kullanımı için)
- Spring Boot Starter Validation(Springte kullanılan constraintleri @Valid ile kontrol edebilmek için)

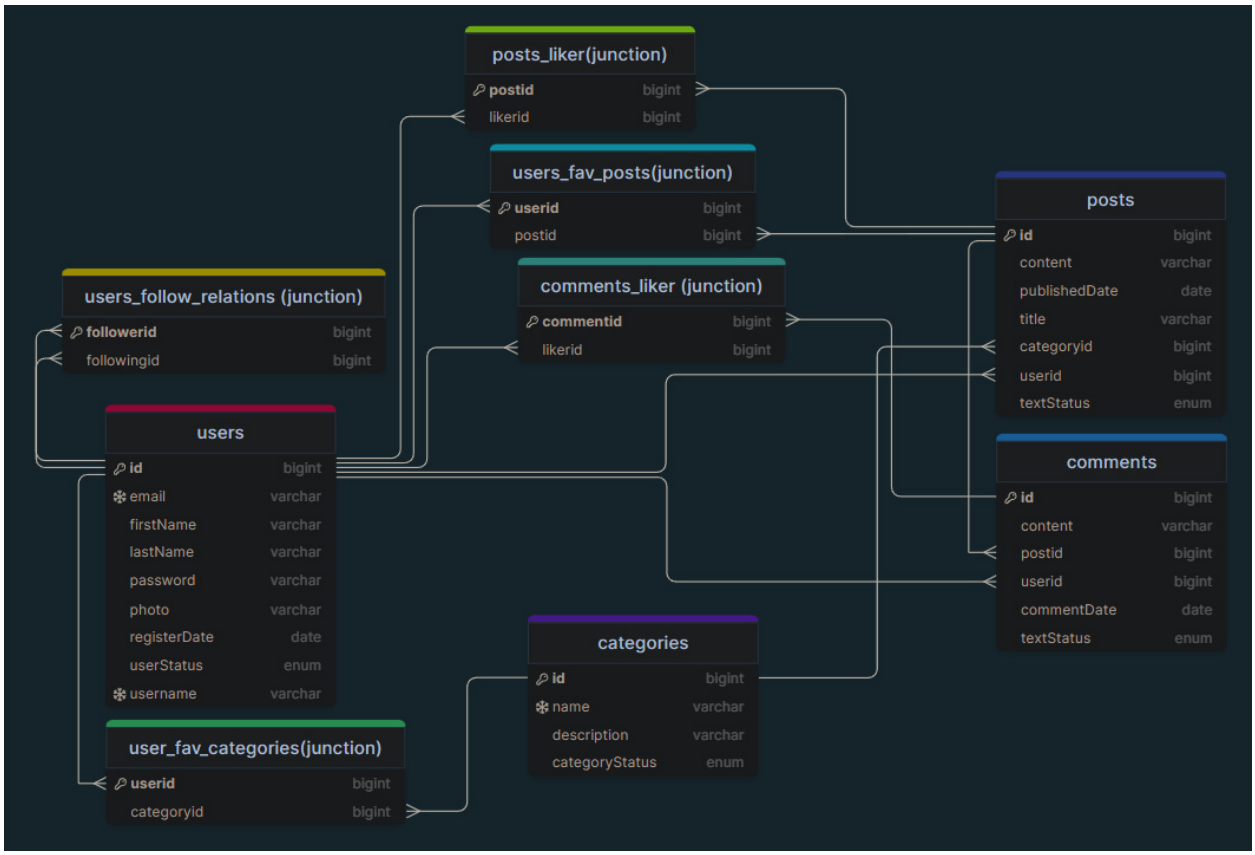
Giriş :

Projeye başlanırken Entityler için User, Post, Category ve Comment sınıfları düşünülmüş ve aralarındaki One-To-Many, Many-To-Many gibi ilişkileri kurarken https://drawsql.app/diagrams websitesinden yararlanılmıştır. Başlangıçta düşünülen kurgu java fieldları düşünülerek yapıldı, aşağıdaki gibidir.



Kurgu özgürlüğü bize bırakıldığı için, delete gibi işlemlerde veriyi komple silmektense entitylere atanacak olan statusleri değiştirmek hedeflendi. Aynı zamanda userde followings ve followers ilişkileri de many-to-many olarak düşünüldü. İki taraflı da sorgular atabilmek için ilişkiler genelde bidirectional olarak düşünüldü. İleride geliştirmeye açık fakat şimdilik işleri karıştırmasını istemediğimden ötürü comment entity içerisinde comment tutulması es geçildi (bir commentin alt commentleri es geçildi).

Burdan yola çıkılarak entity sınıfları oluşturulurken, kurgu üzerinde de ufak tefek değişiklikler yapıldı ve drawsql üzerindengösterilen kurgunun, doğru bir biçimde gösterilmiş son hali aşağıdaki gibidir.



Yapılanlar :

1)Katmanlı Mimari:

Öncelikle katmanlı mimari için packagelar açıldı. Controller, Service, Repository, Entity paketleri oluşturuldu. Entity paketi içerisinde enums paketinde entity statuslerini tutacak enumlar tutuldu. Mapper kullanımını için mapper interfaceleri mapper paketi altında bir arada tutuldu. HTML üzerinden Data Transfer Objectlerle işlemler yapılacağı için dto paketi ve onun altında request ve response paketleri ve içlerinde bunlara ait sınıflar tutuldu. Custom annotation kullanımı yapılacağı için annotation paketi oluşturuldu. Kendi hatalarımı fırlatmak ve bunların yönetimini yapmak için oluşturulan sınıfların hepsi exception paketi altında bir arada tutuldu. Servis katmanındaki sınıflarımın bazı methodlarını genericleştirdiğim için bu generic interface'i ve onun implementasyonunu utility paketi içinde tuttum, service katmanındaki sınıflar buradaki ServiceManager'dan yani implementasyondan miras alıyor.

2) Entityler Oluşturuldu:

Entityler oluşturulurken Lombok ve Hibernate anotasyonlarından yararlanıldı. Aynı zamanda çeşitli kolum özellikleri belirlendi – nullable false, length = 100, unique true, name = "..." gibi.

Bazı fieldlarda yazım işlemlerinde düzenlik sağlanması ve unique'lik özelliğinin kontrolünün sağlanabilmesi için @ColumnTransformer sayesinde yazdırırken herşeyi küçük harfe çevirme işlemi uygulandı.

@Size anotasyonu ile minimum karakter sayısı da belirlendi. Column ile database üzerinden direkt işlem yaparken karakter sınırı tanıyorken, Size ile entityi daha database'e kaydetme işlemine başlamadan önce karakter sınırı kontrolü yapabilmeyi sağladık + minimum karakteri de belirleyebildik.

Enumlar String olarak tutuldu @Enumerated(EnumType.STRING) ile. Entity ilk oluşturulurken bolca Lombok anotasyonu olan @Builder.Default kullanıldı. User sınıfında 17.11.2023 Cuma günü proje arası öncesi son uygulama dersinde gördüğümüz custom annotation oluşturma ve kullanmadan yola çıkıldı ve 2 tane custom annotation yaratıldı. Hem o gün derste hem de ekstradan incenilmiş olan kaynak linki:

https://www.baeldung.com/spring-mvc-custom-validator

@PasswordDigitValidation : Bu anotasyon ile internetten araştırılmış regex sayesinde kullanıcı kayıt yaparken ya da kendi bilgilerini güncellerken girilen şifrede 0-9 rakamlarından en az 1 tane içermesi gerekliliği sağlandı. Eğer şifre herhangi bir rakam içermiyorsa GlobalExceptionHandler exception sınıfında ConstraintViolationException hatası yakalandı ve sebebiyle birlikte kullanıcıya dönüldü.

@UsernameSpecialCharacterValidation : Bu anotasyon ile internetten araştırılmış regex sayesinde kullanıcı kayıt yaparken yada kendi bilgilerini güncellerken girdiği username'de A-Z (büyük/küçük) ve 0-9 haricinde herhangi bir karakter içermemesi gerekliliği sağlandı. Eğer belirtilen karakterler dışında karakter varsa GlobalExceptionHandler exception sınıfında ConstraintViolationException hatası yakalandı ve sebebiyle birlikte kullanıcıya dönüldü.

User sınıfında List<User> followers ve List<User> followings fieldları mevcuttur. Entity ile bu fieldlar arasındaki ilişki her birisi için @ManyToMany olarak belirlenmişti fakat bu durumda users_followers ve users_followings diye iki ayrı junction table ortaya çıkıyordu. Hem proje geliştirme aşamasında hatalar aldığım için hem de tek bir tablo üzerinden bir işlem gerçekleştirmek daha güzel olacağı düşünüldüğü için aşağıdaki yapıya geçildi.

```java
@Builder.Default
@ManyToMany
@JoinTable(name = "users_follow_relations",
        joinColumns = @JoinColumn(name="following_id", referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name="follower_id", referencedColumnName = "id"))
private List<User> followers = new ArrayList<>();

@Builder.Default
@ManyToMany
@JoinTable(name = "users_follow_relations",
        joinColumns = @JoinColumn(name="follower_id", referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name="following_id", referencedColumnName = "id"))
private List<User> followings = new ArrayList<>();
```

Hatadan yola çıkılan ve düzenlemede destek alınan kaynak:
https://stackoverflow.com/questions/77151752/self-join-many-to-many-relationship-leads-to-stack-overflow

Burada kurgulanan mantık şu oldu. Diyelim 2 tane user olsun birisinin idsi 2 diğerinin 7. 2 numaralı user 7yi takip ettiği zaman userId2.getFollowings().add(userId7) ile tabloya ekleme işlemi gerçekleşiyor.

Tabloda aşağıdaki gibi oluşuyor:

Follower id	Following id
2	7

Hayal edecek olursak Followers fieldi tabloya sağdan bakıyor ve 7 numaralı usera 2 numaralı useri ekliyor. Followings fieldı ise tabloya soldan bakıyor ve 2 numaralı usera 7 numaralı useri ekliyor. Böylelikle tek bir tabloyu 2 field kullanıyor. Aynı mantıkla eğer bu işlem sonrası 7 numaralı user da 2 numara useri takip etseydi tablo şu şekilde olacaktı.

Follower id	Following id
2	7
7	2

3)Dtolar yardımıyla Kontrol ve Servis Katmanları dolduruldu:

Burada standart controller ve servis işlemleri uygulandı. Projede istenilen şekilde @GetMapping, @PostMapping, @PutMapping, @DeleteMapping eşleşmeleri yapıldı ve methodlar mümkün olduğu şekilde servis kısmında tek bir işi yapacak şekilde tasarlanmaya ve kod tekrarını azaltmaya yönelik çalışıldı. @PathVariable kullanımı öğrenildi ve uygulanıldı.https://www.baeldung.com/spring-pathvariable

Projeyi yaparken Mapper sınıfı için expression belirlemek araştırıldı. Geri dönüş fieldlarının hepsi String olarak belirlendiği durumlarda fakat entity sınıfları içerisinde farklı türlerde fieldlar olduğu zaman aralarındaki dönüşümün @Mapping anotasyonunun içerisindeki expression ile yapılabileceği öğrenildi, projede uygulanmış olan örnek kullanım aşağıdaki gibidir. Beni bu kullanıma araştırmaya tetikleyen ilk kaynak linki: https://stackoverflow.com/questions/66204915/how-to-map-java-collection-to-its-size

```
1 usage  1 implementation  alivolkansahin
@Mapping(target = "id", expression = "java(String.valueOf(post.getId()))")
@Mapping(target = "sender", expression = "java(post.getUser().getUsername())")
@Mapping(target = "publishDate", expression = "java(post.getPublishDate().toString())")
@Mapping(target = "category", expression = "java(post.getCategory().getName())")
@Mapping(target = "commentCount", expression = "java(String.valueOf(post.getComments().size()))")
@Mapping(target = "likeCount", expression = "java(String.valueOf(post.getLiker().size()))")
@Mapping(target = "status", expression = "java(post.getStatus().name())")
PostGetAllResponseDto getAllPostToDto(Post post);
```

Aynı zamanda Endpoint sınıfı ile her controllerdaki eşleştirme adresleri için genel bir düzen sağlanmaya çalışıldı. Proje isterleri arasında aynı bağlantı üzerinden search ve category olarak farklı arama işlemi yapılabileceği isteri vardı aşağıdaki gibi:

Arama ve Kategorilendirme:

- GET /api/posts?search={arama_kelimesi}: Belirli bir kelimeye göre yazıları arar.
- GET /api/posts?category={kategori}: Belirli bir kategoriye ait yazıları getirir.

Bu iki ister ayrı 2 methodda yazılmıştı controller üzerinde fakat ikisinin de endpointinin aynı olması isteri dikkat çekti. Bu yüzden burdan yola çıkarak @RequestParam kullanımı öğrenildi ve projede istenilen aşağıdaki ister için yani aynı path üzerinden gelecek kelimeye göre şu şekilde bir çözüm uygulandı. İçerisinde bulundurulan exceptiondan ve genel projede oluşturulan exceptionlardan bir sonraki adımda bahsedilecek.

```
alivolkansahin
@GetMapping // proje pdfinde aynı path üzerinden verildiği için ikisi de bu şekile çevrildi.
public ResponseEntity<List<PostGetAllResponseDto>> getAllPosts(@RequestParam(required = false) String search, @RequestParam(required = false) String category) {
    if (search != null) return ResponseEntity.ok(postService.getAllPostsBySearchParameter(search)); // Belirli bir kelimeye göre yazıları arar.
    else if (category != null) return ResponseEntity.ok(postService.getAllPostsByCategoryParameter(category)); // Belirli bir kategoriye ait yazıları getirir.
    else throw new AvsBlogException(ErrorType.BLANK_PARAMETER_ENTRY);
}
```

Özetlemek gerekirse;e controllerda ve servis katmanında yapılanlar proje isterleriyle bağlantılı şekilde yapıldı. Her bir entity için hepsini listeleme, sadece bir tanesini listeleme, yeni bir tane entity oluşturma, mevcut entitynin bilgilerini güncelleme ve mevcut bir entityi silme methodları yazıldı. Kurgu gereği delete işleminde komple veriyi silip kaybetmektense statüs değişikliği yapılmasına karar verildi. İlave olarak ilgili controller ve servislerde userlar arası takipleşme, post beğenme, yorum yapma, yorum beğenme gibi kişisel çalışmalar ve uygun methodlar da yapıldı.

4) Exceptionların belirlenmesi, tutulması ve dökümente edilmesi:

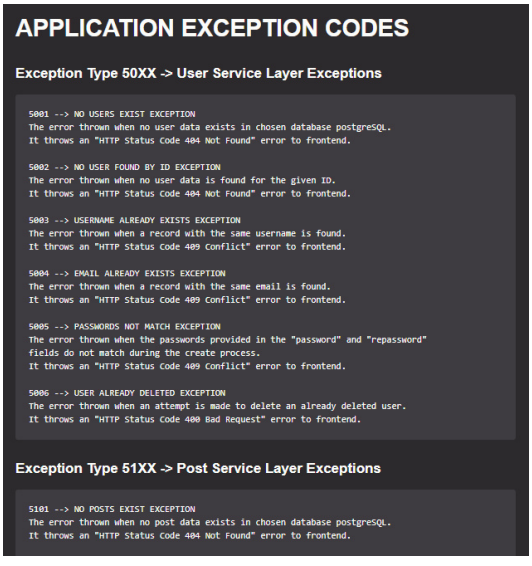
Son olarak projenin en önemli noktalarından birisi uygulamada oluşabilecek hatalar, bunların tutulması ve geriye dönüşlerinin ayarlanması idi. Controller ve servis katmanlarında çeşitli testler yapıldı ve oluşabilecek hatalar tespit edildi. Benim kendim oluşturduğum hatalar için exception paketi içerisinde ErrorType enumı oluşturuldu. Bunlar yine aynı paket içerisindeki GlobalExceptionHandler sınıfında yakalandı. Aynı zamanda kullanılan anotasyonlarda oluşabilecek 2 farklı hata türü – MethodArgumentNotValidException ve ConstraintViolationException da yine bu sınıfta yakalandı. Eğer birden fazla anotasyonda aynı hata türü ile karşılaşılırsa diye streamlerden yararlanıldı ve bütün hata mesajı, bu hataların tek tek nerede gerçekleştiği ve kendi mesajıyla birlikte aralarında virgül koyacak şekilde tek bir mesaj olarak kullanıcıya dönülmesi hedeflendi. Bu kullanım aşağıdaki görselde gösterildi.

Ne olur ne olmaz diye ve ileride projenin geliştirileceği düşünülerek hareket edilerek ArithmeticException ve henüz tam düzenlenmesi yapılmamış şekilde Exception sınıfı hataları da yakalanıp yine burada usera döndürüldü. Bu son 2 exception yakalaması ileride henüz bilgi dahilimde olmayan bir hata oluşması durumunda programın ayakta kalabilmesi için eklenildi.

```
alivolkansahin
@ExceptionHandler(MethodArgumentNotValidException.class) //5501   // @Valid ile sağlanmayan (bu kurguda @NotBlank) hataları yakalanıyor
@ResponseBody
public ResponseEntity<ErrorMessage> handleValidationException(MethodArgumentNotValidException ex){
    String customizedMessage = ex.getBindingResult().getFieldErrors().stream().map(error -> error.getDefaultMessage()).collect(Collectors.joining( delimiter: ","));
    HttpStatus httpStatus = HttpStatus.BAD_REQUEST; // 400
    return new ResponseEntity<>(createError(ex,  value: 5501, customizedMessage), httpStatus);
}

alivolkansahin *
@ExceptionHandler(ConstraintViolationException.class) //5601    // Entity sınıfındaki hazır kütüphane annotationları ve custom annotationların hataları yakalanıyor.
@ResponseBody
public ResponseEntity<ErrorMessage> handleValidationException(ConstraintViolationException ex){
    String customizedMessage = ex.getConstraintViolations().stream().map(error -> error.getMessage()).collect(Collectors.joining( delimiter: ","));
    HttpStatus httpStatus = HttpStatus.BAD_REQUEST; // 400 // 422 Unprocessable Entity dee kullanılabilirdi.
    return new ResponseEntity<>(createError(ex,  value: 5601, customizedMessage), httpStatus);
}
```

Bütün yapılan testler sonucu tespit edilen ve oluşturulan hatalar kendi kodu, açıklaması, HttpStatusCode olarak geri dönüşü ile birlikte proje içerisinde bir dökümantasyonda tutuldu. Bu dökümantasyonun adı da “ExceptionCodes.md” ‘dir, aşağıdaki görselde ufak bir görüntüsü paylaşılmıştır.



APPLICATION EXCEPTION CODES

Exception Type 50XX -> User Service Layer Exceptions

5001 --> NO USERS EXIST EXCEPTION
The error thrown when no user data exists in chosen database postgreSQL.
It throws an "HTTP Status Code 404 Not Found" error to frontend.

5002 --> NO USER FOUND BY ID EXCEPTION
The error thrown when no user data is found for the given ID.
It throws an "HTTP Status Code 404 Not Found" error to frontend.

5003 --> USERNAME ALREADY EXISTS EXCEPTION
The error thrown when a record with the same username is found.
It throws an "HTTP Status Code 409 Conflict" error to frontend.

5004 --> EMAIL ALREADY EXISTS EXCEPTION
The error thrown when a record with the same email is found.
It throws an "HTTP Status Code 409 Conflict" error to frontend.

5005 --> PASSWORDS NOT MATCH EXCEPTION
The error thrown when the passwords provided in the "password" and "repassword"
fields do not match during the create process.
It throws an "HTTP Status Code 409 Conflict" error to frontend.

5006 --> USER ALREADY DELETED EXCEPTION
The error thrown when an attempt is made to delete an already deleted user.
It throws an "HTTP Status Code 400 Bad Request" error to frontend.

Exception Type 51XX -> Post Service Layer Exceptions

5101 --> NO POSTS EXIST EXCEPTION
The error thrown when no post data exists in chosen database postgreSQL.
It throws an "HTTP Status Code 404 Not Found" error to frontend.

Sonuç :

Sonuç olarak elimizde proje açıklamasında yazılan isterlerin hepsini gerçekleştirebilen ve kendi kurgumla ilave ettiğim takipleşme, beğenme gibi şeylerin de yapılabileceği bir restful api tabanlı spring boot projesi yapılmıştır.

Proje sırasında aynı zamanda @PatchMapping kullanımı da uygulanmak istenmiş fakat kullanımının biraz daha araştırılması gerektiği farkedilmiştir.

Geliştirmeye yönelik commentin de commenti olabilir kurgusu da kurulabileceği düşünülmektedir. Listeleme işlemleri yapılırken sadece aktif postları getirmek, silinmiş (statüsü değiştirilmiş) postları getirmemek gibi methodların da yapılabileceği bir kenara not alındı.

İleride security kısmı öğrenildiğinde login işlemi yaptırmak ve bütün kurguda Unauthorized durumlarını kontrol etme gibi durumlar da bir kenara not alındı.

Projede Hata kodları oluşturulurken HTTP.StatusCode'ları her ne kadar araştırılıp uygun bir biçimde kullanılmaya çalışılmış olsa da , bazı durumlarda birbirine çok yakın durumlarla karşılaşıldı ve kararsızlıklar oluştu. Buna verebileceğim örnek olarak, mevcut entity'i güncellemek isterken 400 – Bad Request ile 422 – Unprocessable Entity kodlarından hangisi uygun olur diye ikilemde kalmışlığım oldu.