Enoca Backend challenge cevapları

Cevap 1 -> Bu sorunun cevabını 2 farklı açıdan verebiliriz. İlk olarak basit çözümden başlayacağım. Eğer bilet oluştururken kullanılan parametrelerin değerleri static ise backend tarafından bu sayfa ilk oluşturulduğunda bu fiyat bilgilerini frontend 'de tutarız ve sayfa üzerindeki her onChange methodu durumunda yeni fiyatı güncelleriz. Örneğin benim kullandığım js frameworkü olan React' te useState olarak tutar bilgisini tutarsak ve form üzerindeki bir bilgi onChange ile değiştirildiğinde tutar bilgisi yeni değere güncellenir.

İkinci çözüm ise biraz daha karmaşık bir yöntem içeriyor fakat gerçek hayat uygulamalarında daha doğru bir çözüm yöntemi olacağını düşünüyorum. Bu yöntemde asıl görev backend tarafında gerçekleşiyor. Fiyat bilgisi dinamik olarak değişkenlik gösterdiği durumda ki bu uçak bileti alma durumun bunun için ideal bir örnek. Firmalar genellikle uçak bileti fiyatı güncellemelerini havayolu firmalarından anlık olarak alırlar. Biz bu soruda şunu yapmamız lazım. Frontend üzerinde onChange methodu her tetiklendiğinde backend tarafına bir istek atılır. Backend gerekli havayolu firmasının api'sine bir istek atar ve anlılk fiyat bilgisi alınır. Backend tarafı havayolu firmasından aldığı fiyatın üzerine kendi ücretini ekledikten sonra bunu frontend tarafına gönderir. Yine önceki örnekte olduğu gibi useState değişkeni olarak tutulan tutar bilgisi güncellenmiş fiyat bilgisine göre hesaplanır ve ilgili bölüme kaydedilir ve kullanıcı anlık olarak değişkenlik gösteren fiyat bilgisine bu şekilde sayfayı yenilemeden erişmiş olur.

Cevap 2 -> Bu sorunun cevabını yazmadan önce öncelikli olarak sorunun çözümünü sözel olarak ifade edelim. İlk koşulda “brand name” ve “model year” verileri birbirine eşit olan arabaları tespit etmemiz gerekiyor. Daha sonra bu tespit edilen arabaların galerilerde kaçar tane olduğuna bakıp 3 taneden fazla olan arabaları geri döndürmemiz isteniyor. Sql sorgusu olarak bu açıkladıklarımın ifade şekli şu şekildedir:

```
SELECT c.id, c.brand_name, c.model_year, c.price, ag.name AS auto_gallery_name
FROM Car c
INNER JOIN Auto_Gallery ag ON c.auto_gallery = ag.id
WHERE (c.brand_name, c.model_year) IN (
   SELECT brand_name, model_year
   FROM Car
   GROUP BY brand_name, model_year
   HAVING COUNT(*) > 3
);
```

Cevap 3 ->

@Component: Bu anotasyon, bir sınıfın bir Spring bileşeni olduğunu belirtir. Spring, bu sınıfları tarama yaparak bulur ve yönetir.

@Controller: Bir sınıfın bir Spring MVC denetleyicisi olduğunu belirtir. Bu sınıf, HTTP isteklerini işlemek ve cevap vermek için kullanılır.

@Service: İş mantığını içeren sınıfları işaretler. Spring, bu sınıfları bileşen olarak yönetir ve gerektiğinde enjekte eder.

@Repository: Veritabanı işlemlerini gerçekleştiren sınıflar için kullanılır. Bu anotasyon, veritabanı erişimi sağlayan sınıfları işaretler.

@Autowired: Bağımlılıkları enjekte etmek için kullanılır. Bu anotasyon, sınıfın içindeki alanları, setter veya constructor üzerinden otomatik olarak enjekte eder.

@Value: Özellik değerlerini dışarıdan enjekte etmek için kullanılır. Özellik dosyasından veya çevre değişkenlerinden değerleri alır.

@RequestMapping: Bir Spring MVC denetleyicisinde, bir HTTP isteği ile eşleşen bir işlemi belirtmek için kullanılır. İsteğin URL yolunu ve HTTP yöntemini tanımlar.

@PathVariable: URL yolu içindeki değişkenleri almak için kullanılır. Örneğin, **/users/{id}** gibi bir URL'den **{id}** değerini alır.

@RequestParam: HTTP isteği parametrelerini almak için kullanılır. URL'den veya form verilerinden parametre değerlerini alır.

@Configuration: Bu anotasyon, Java tabanlı bir konfigürasyon sınıfını işaretler. Bu sınıflar, Spring uygulamanızın yapılandırmasını tanımlamanıza olanak sağlar.

@Bean: Bu anotasyon, Spring konteynerine bir bileşen eklemek için kullanılır. **@Configuration** sınıflarında **@Bean** yöntemleri tanımlayarak uygulanır.

@Transactional: Veritabanı işlemleri sırasında işlem sınırlarını tanımlamak için kullanılır. Bu anotasyon, işlemlerin başarılı bir şekilde tamamlanmasını veya geri alınmasını sağlar.

Cevap 4 -> **isGiftCart** metodunun bir exception oluşturması beklenmez. Ancak, **cart.getType()** çağrısı NullPointerException fırlatabilir. Bu durumda, **NullPointerException'ı** yakalamak için bir try-catch bloğu kullanılabilir.

NullPointerException'ın nedeni, cart parametresinin null olmasıdır. Bu durumda, **isGiftCart** metodunun null değerli bir **CartModel** parametresiyle çağrılması sonucu bir exception oluşur. Bu durumu önlemek için, **isGiftCart** metodunun başında null kontrolü yapılabilir. Örneğin:

```
public boolean isGiftCart(CartModel cart) {
    try {
        return cart.getType().equals(CartType.GIFT);
    } catch (NullPointerException e) {
        return false;
    }
}
```