Auditing in Spring Data JPA

Bu projenin gerçek QuestionIngestService'inin elle yazdığı LocalDateTime.now() deseninden başlayarak, @CreatedDate/@LastModifiedDate, @EntityListeners (AuditingEntityListener.class) + @EnableJpaAuditing bağlantısı, @CreatedBy /@LastModifiedBy, AuditorAware<T>, ve @MappedSuperclass ile audit alanlarını paylaşmak. Spring Data JPA kategorisinin 8.'si.

Orta 25 dk

Son güncelleme: 24 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/jpa-auditing

---

Bu projenin kendi `QuestionIngestService`'i, yeni bir `Question`'ı `.createdAt(LocalDateTime.now()).updatedAt(LocalDateTime.now())` ile, tam olarak servis metodunun içinde elle yazarak inşa eder. Çalışır -- ama bu tam olarak Spring Data JPA'nın başka yerlerde genelde ortadan kaldırdığı türden tekrarlı, unutulması kolay kod. Bu ders, tam olarak bunun için inşa edilmiş aracı işliyor: auditing.

Sorun: createdAt/updatedAt'i Elle Ayarlamak

Auditli bir entity'yi oluşturan ya da güncelleyen her yer, her seferinde zaman damgasını doğru ayarlamayı hatırlamak zorundadır.

```
import java.time.LocalDateTime;

// This is the SAME pattern as this project's own real
// QuestionIngestService -- it builds a Question with
// .createdAt(LocalDateTime.now()).updatedAt(LocalDateTime.now()), by hand,
// right there in the service method. Nothing is technically wrong with
// this, but it's a pattern that has to be repeated correctly in EVERY
// place that creates or updates ANY audited entity -- forget it once, in
// one service method, and that row's timestamp is silently wrong.
class ManualTimestampProblemExample {

    record Question(String text, LocalDateTime createdAt, LocalDateTime updatedAt) {
    }

    static Question createQuestion(String text) {
        LocalDateTime now = LocalDateTime.now();
        return new Question(text, now, now); // set by hand, exactly like this project's
QuestionIngestService
    }

    static Question updateQuestion(Question existing, String newText) {
        // Every single update site also needs to remember this line --
        // and remember it EVERY time, not just once.
        return new Question(newText, existing.createdAt(), LocalDateTime.now());
    }
}
```

`createQuestion(...)` ve `updateQuestion(...)`, ikisi de kendi `LocalDateTime.now()` satırına ihtiyaç duyar -- tam olarak bu projenin kendi `QuestionIngestService` deseni. Burada teknik olarak bozuk hiçbir şey yok, ama ikinci bir servis metodu (ya da üçüncü, ya da onuncu) aynı türden bir entity'yi oluşturması ya da güncellemesi gerektiği anda, aynı satırın her seferinde, her yerde doğru şekilde hatırlanması ve tekrarlanması gerekir -- bir kez unutmak, sessizce yanlış bir zaman damgasına sahip bir satır bırakır.

@CreatedDate ve @LastModifiedDate

Bağlandığında, iki annotation o elle yazılan satırı tamamen değiştirir.

```
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.Id;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import java.time.LocalDateTime;

// @EntityListeners(AuditingEntityListener.class) is what actually makes
// @CreatedDate/@LastModifiedDate below do anything -- it registers a
// listener that runs automatically on this entity's own lifecycle events
// (right before the first INSERT, and right before every UPDATE), instead
// of any application code needing to set these fields itself.
@Entity
@EntityListeners(AuditingEntityListener.class)
class AuditedQuestionExample {

    @Id
    private Long id;

    private String text;

    // Populated automatically, exactly once, the moment this entity is
    // first persisted -- never touched again on later updates.
    @CreatedDate
    private LocalDateTime createdAt;

    // Populated automatically on the INITIAL insert, and then re-populated
    // on every single update after that -- this is the field
    // ManualTimestampProblemExample had to remember to update by hand,
    // every time, in every place.
    @LastModifiedDate
    private LocalDateTime updatedAt;
}
```

`@CreatedDate`, bir entity ilk kez kalıcı hale getirildiği anda, tam olarak bir kez, otomatik olarak doldurulur -- sonrasında bir daha asla dokunulmaz. `@LastModifiedDate`, aynı ilk insert'te doldurulur, ve sonra sonraki her güncellemede otomatik olarak yeniden doldurulur -- bu, `ManualTimestampProblemExample`'ın ona dokunan her tek yerde elle güncellemeyi hatırlamak zorunda olduğu alan.

Bağlamak: @EntityListeners ve @EnableJpaAuditing

`@CreatedDate` / `@LastModifiedDate`'in gerçekten bir şey yapması için iki parçanın yerinde olması gerekir -- ikisinden yalnızca biri yeterli değildir.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

// @EntityListeners alone isn't quite enough -- @EnableJpaAuditing, on a
// @Configuration class, is what turns Spring Data JPA's auditing
// infrastructure on for the application as a whole. Without it,
// @CreatedDate/@LastModifiedDate fields are simply never populated --
// silently left null, with no error to point at the missing piece.
@Configuration
@EnableJpaAuditing
class JpaAuditingConfig {
}

// With both pieces in place -- @EntityListeners on the entity, and
// @EnableJpaAuditing on a configuration class -- saving an
// AuditedQuestionExample no longer needs anything like
// ManualTimestampProblemExample's "LocalDateTime.now()" line anywhere:
//
//   AuditedQuestionExample q = new AuditedQuestionExample();
//   repository.save(q); // createdAt AND updatedAt are populated automatically
```

`@EntityListeners(AuditingEntityListener.class)`, entity'nin kendisinde, o entity'nin yaşam döngüsü olaylarında (ilk insert'ten hemen önce, ve her update'ten hemen önce) otomatik olarak çalışan bir listener kaydeder. `@EnableJpaAuditing`, bir `@Configuration` sınıfında, Spring Data JPA'nın auditing altyapısını bütün uygulama için açar. İkisinden birini eksik bırakmak, `@CreatedDate` / `@LastModifiedDate`'in basitçe hiç doldurulmaması anlamına gelir -- sessizce `null` kalır, neyin eksik olduğunu işaret eden hiçbir hata olmadan.

Kimi Kaydetmek: @CreatedBy ve @LastModifiedBy

Aynı mekanizma, yalnızca NE ZAMAN'a değil, KİM'in değişikliği yaptığına da genişler.

```
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.Id;
import org.springframework.data.annotation.CreatedBy;
import org.springframework.data.annotation.LastModifiedBy;
import org.springframework.data.domain.AuditorAware;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Optional;

// @CreatedBy/@LastModifiedBy work exactly like @CreatedDate/@LastModifiedDate
// -- same listener, same lifecycle timing -- but capture WHO made the
// change instead of WHEN. This project's real Question entity already has
// a "reviewedBy" column (set manually, by an admin doing a DB UPDATE, per
// this project's own review workflow) -- @CreatedBy/@LastModifiedBy solve
// a related but different problem: recording who created/last touched the
// row itself, automatically, not a separate manual review action.
@Entity
@EntityListeners(AuditingEntityListener.class)
class AuditedByQuestionExample {

    @Id
    private Long id;

    @CreatedBy
    private String createdBy;

    @LastModifiedBy
    private String lastModifiedBy;
}

// AuditorAware<T> is where "who" actually comes from -- Spring Data JPA
// has no idea who the current user is on its own; this bean is what
// supplies that answer, called automatically every time an audited entity
// is saved.
@Configuration
class AuditorAwareConfig {

    @Bean
    AuditorAware<String> auditorProvider() {
        // A real application would read this from Spring Security's
        // SecurityContextHolder (the currently authenticated user's name);
        // returning a fixed value here keeps the example focused on
        // AuditorAware's role, not on Spring Security itself.
        return () -> Optional.of("system");
    }
}
```

`@CreatedBy` ve `@LastModifiedBy`, tarih karşılıklarıyla birebir aynı şekilde çalışır -- aynı listener, aynı yaşam döngüsü zamanlaması -- ama bir zaman damgası yerine değişikliği yapan kişinin kimliğini yakalar. Bu projenin gerçek `Question` entity'sinin zaten bir `reviewedBy`

sütunu var, ama bu elle, bilinçli bir admin inceleme eylemi (bu projenin kendi soru havuzu inceleme iş akışı gereği) olarak ayarlanır -- satırın kendisini kimin oluşturduğunu ya da son değiştirdiğini otomatik olarak kaydetmekten gerçekten farklı bir şey.

AuditorAware<T>: "Kim" Nereden Gelir

Spring Data JPA'nın "mevcut kullanıcı" diye yerleşik bir kavramı yoktur -- bir şeyin bu cevabı sağlaması gerekir.

`AuditorAware<T>`, o şeydir: değişikliği "şu anda" kimin yaptığını temsil eden bir `Optional<T>` döndüren, auditli bir entity her kaydedildiğinde otomatik olarak çağrılan tek-metotlu bir interface. Gerçek bir uygulama bunu, sabit bir değer döndürmek yerine, Spring Security'nin `SecurityContextHolder`'ından -- şu anda kimliği doğrulanmış kullanıcının adı ya da id'si -- okuyarak implement ederdi; örnekteki sabit `"system"` değeri, odağı Spring Security'de değil (bu kategorinin kapsamadığı), `AuditorAware`'in kendi rolünde tutar.

@MappedSuperclass ile Audit Alanlarını Paylaşmak

Birden fazla entity aynı audit alanlarına ihtiyaç duyduğu anda, `@CreatedDate`/`@LastModifiedDate`'i her birinde tekrarlamak, tam olarak auditing'in baştan ortadan kaldırmayı amaçladığı türden bir tekrar hâline gelir.

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import java.time.LocalDateTime;

// @MappedSuperclass isn't itself an @Entity -- it's a base class whose
// fields get copied into every entity that extends it, without a table of
// its own. This is where the audit fields belong once more than one
// entity needs them: this project's real Question already has createdAt/
// updatedAt columns (set by hand today, in QuestionIngestService) -- if
// QuestionOption or another entity needed the same two columns, repeating
// @CreatedDate/@LastModifiedDate on each one would itself become the same
// kind of repetition auditing was meant to remove in the first place.
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
abstract class AuditableBaseExample {

    @CreatedDate
    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @LastModifiedDate
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
}

// Any entity extending AuditableBaseExample gets createdAt/updatedAt for
// free -- no @EntityListeners of its own needed (it's inherited), and no
// repeated field declarations either.
@Entity
class AuditableQuestionExample extends AuditableBaseExample {
    @Id
    private Long id;
    private String text;
}
```

`@MappedSuperclass`, kendisi bir `@Entity` değildir ve kendi tablosu yoktur -- onu genişleten her entity'ye alanları kopyalanan bir base sınıftır. Bu projenin gerçek `Question`'ının zaten `createdAt` / `updatedAt` sütunları var; `QuestionOption` ya da başka bir entity tam olarak aynı iki alana ihtiyaç duysaydı, paylaşılan bir `@MappedSuperclass`'ı genişletmek, `@CreatedDate` / `@LastModifiedDate`'i (ve `@EntityListeners`'ı) her birinde ayrı ayrı bildirmekten kaçınırdı.

Yaygın Yanlış Anlamalar

"Yalnızca `@CreatedDate`, auditing'in çalışması için yeterlidir." Değildir -- entity'de `@EntityListeners(AuditingEntityListener.class)` VE uygulamada bir yerde `@EnableJpaAuditing` olmadan, alan basitçe hiç doldurulmaz, hiçbir hata olmadan. **"**

`@LastModifiedDate`**, yalnızca gerçek alan değişikliklerinde güncellenir."** Veritabanına ulaşan her kaydetmede güncellenir, tam olarak dirty checking'in ("Transaction Management" te işlenen) izlenen herhangi bir değişikliği yazması gibi -- hangi kaydetmelerin "gerçekten" anlamlı bir şeyi değiştirdiği konusunda seçici bir şekilde akıllı değildir. **"`AuditorAware`'in çalışması için Spring Security'ye ihtiyacı vardır."** Yapısal olarak ona bağlı değildir -- yalnızca bir `Optional<T>` döndüren bir interface'tir; gerçek bir uygulama bunu genelde Spring Security'den okuyarak implement eder, ama mekanizmanın kendisi bu belirli seçimden bağımsızdır.

Sırada Ne Var

Bu kategorideki her topic şimdiye kadar entity verisini doğru okumaya, yazmaya ya da izlemeye odaklandı. Bu kategorinin son dersi "Spring Data JPA Repository'lerini Test Etmek", bunların hepsinin -- repository'ler, sorgular, projection'lar, ilişkiler, hatta auditing -- `@DataJpaTest` ile bu derslerin tarif ettiği gibi gerçekten davrandığını doğrulamaya geçiyor.

Best Practices

- `@EntityListeners(AuditingEntityListener.class)`'ı ve `@EnableJpaAuditing`'i birlikte ekle -- biri olmadan diğeri sessizce hiçbir şey yapmaz.
- İkinci bir entity aynı audit alanlarına ihtiyaç duyduğu anda, annotation'ları her birinde tekrarlamak yerine `@MappedSuperclass`'a başvur.
- `AuditorAware<T>`'i gerçek bir uygulamada, sabit bir değer yerine mevcut kullanıcıyı Spring Security'nin `SecurityContextHolder`'ından okuyarak implement et.
- Gerçekten yalnızca "bu ne zaman oluşturuldu/değiştirildi" olan herhangi bir alan için elle yazılmış `LocalDateTime.now()` çağrıları yerine auditing'i tercih et -- elle zaman damgası alanlarını, bu projenin `reviewedAt`'ı gibi kendi ayrı anlamı olan durumlar için sakla.

Yaygın Hatalar

- `@EntityListeners` ya da `@EnableJpaAuditing` olmadan `@CreatedDate`/`@LastModifiedDate` eklemek, ve alanların neden `null` kaldığına kafa karıştırmak.
- `@LastModifiedDate`'in yalnızca "anlamlı" bir şey değiştiğinde güncellendiğini varsaymak, veritabanına ulaşan her kaydetmede değil.
- `reviewedBy`'ı (bu projenin kendi `Question` entity'sindeki gibi, bilinçli, elle bir inceleme eylemi) `@LastModifiedBy` (satırı kimin son kaydettiğinin otomatik kaydı) ile karıştırmak -- farklı sorulara cevap verirler.
- Birden fazla entity onlara ihtiyaç duyduğu anda paylaşılan bir `@MappedSuperclass`'a çıkarmak yerine, `@CreatedDate`/`@LastModifiedDate`'i her entity'de ayrı ayrı tekrarlamak.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `@CreatedDate` / `@LastModifiedDate`, elle yazılmış `LocalDateTime.now()` çağrılarını (bu projenin gerçek `QuestionIngestService` deseni gibi) otomatik zaman damgalarıyla değiştirir.
- Hem `@EntityListeners(AuditingEntityListener.class)` (entity'de) hem `@EnableJpaAuditing` (bir configuration sınıfında) birlikte gereklidir -- yalnızca biri hiçbir şey yapmaz.
- `@CreatedBy` / `@LastModifiedBy`, tarih annotation'larıyla aynı listener mekanizmasını kullanarak değişikliği kimin yaptığını kaydeder.
- `AuditorAware<T>`, "kim"i sağlar -- gerçek bir uygulamada genelde mevcut kullanıcıyı Spring Security'den okuyarak implement edilir.
- `@MappedSuperclass`, audit alanlarını, her birinde annotation'ları tekrarlamadan, birden fazla entity arasında paylaştırır.

Cheat Sheet

```
// Uygulama için auditing'i aç
@Configuration
@EnableJpaAuditing
class JpaAuditingConfig {}

// Auditli bir entity
@Entity
@EntityListeners(AuditingEntityListener.class)
class Question {
    @CreatedDate
    private LocalDateTime createdAt;

    @LastModifiedDate
    private LocalDateTime updatedAt;

    @CreatedBy
    private String createdBy;

    @LastModifiedBy
    private String lastModifiedBy;
}

// "Kim"i sağlamak
@Bean
AuditorAware<String> auditorProvider() {
    return () -> Optional.ofNullable(SecurityContextHolder.getContext().getAuthentication())
            .map(Authentication::getName);
}

// Alanları entity'ler arasında paylaşmak
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
abstract class Auditable {
    @CreatedDate private LocalDateTime createdAt;
    @LastModifiedDate private LocalDateTime updatedAt;
}
```

Terimler Sözlüğü

- **@CreatedDate / @LastModifiedDate**: bir zaman damgası alanını insert'te (ve ikincisi için, sonraki her update'te) otomatik olarak dolduran annotation'lar.
- **@EntityListeners(AuditingEntityListener.class)**: bir entity üzerinde, auditing annotation'larını gerçekten süren listener'ı kaydeder.
- **@EnableJpaAuditing**: Spring Data JPA'nın auditing altyapısını bütün uygulama için açar.
- **@CreatedBy / @LastModifiedBy**: tarih annotation'larıyla aynı listener mekanizmasını kullanarak değişikliği kimin yaptığını kaydeden annotation'lar.
- **AuditorAware<T>**: "mevcut kullanıcının kim olduğunu" `@CreatedBy` / `@LastModifiedBy` 'a sağlayan interface.
- **@MappedSuperclass**: alanları (audit alanları gibi) onu genişleten her entity tarafından, kendi tablosu olmadan miras alınan, entity olmayan bir base sınıf.