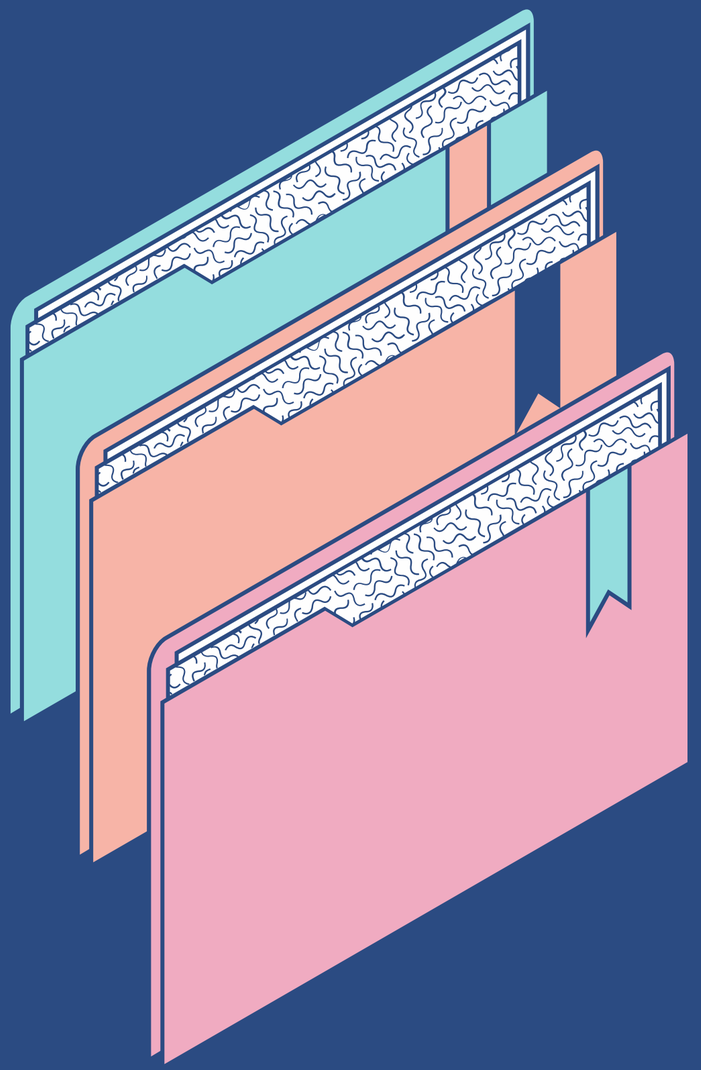
STAJ SONU GENEL SUNUM

Jpa ve Reactive Programing Karşılaştırma

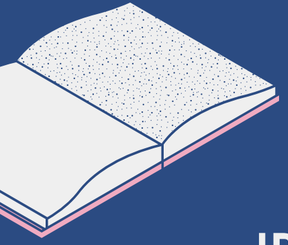
Hazırlayanlar:
Berkay DURMUŞ , Fikret Efe YARDIM



NELER ÖĞRENDİK?

BU SUNUMDA TARTIŞILAN ANA KONULAR

- Jpa Nedir?
- Reactive Programing Nedir?
- Jpa Kullanım Metodları.
- Reactive Kullanım Metodları.



JPA NEDİR?

İŞLEM MODELİ

- JPA, bloklama (blocking) işlem modelini kullanır. Bu, bir veritabanı sorgusu yapıldığında, işlem tamamlanana kadar o thread'in bloke olması anlamına gelir. Bu model, senkron ve genellikle tekil işlemler için uygundur.

PERFORMANS

- JPA, genellikle bir thread başına bir işlem yaparak çalışır. Yüksek sayıda eşzamanlı istek olduğunda, bu model ölçeklenebilirlik açısından sınırlamalar getirebilir.

VERİ YÖNETİMİ

- JPA, EntityManager gibi nesneler üzerinden çalışır ve veritabanı ile etkileşim genellikle geleneksel SQL sorguları veya JPQL kullanılarak yapılır. Nesne-ilişkisel haritalama (ORM) modeli, veritabanı tablolarını Java nesnelerine eşler.

KOD YAPISI

- Kod, genellikle senkron işlemler ve metodlar üzerinden yürütülür. CRUD işlemleri için standart repository yapısı kullanılır.

KULLANIM ALANI

- Genellikle standart CRUD işlemleri, geleneksel web uygulamaları ve veritabanı işlemleri için kullanılır. Bloklama modelinin sorun yaratmadığı durumlar için uygundur.

REAKTİF PROGRAMLAMA NEDİR?

İŞLEM MODELİ

- Reaktif programlama, bloklamayan (non-blocking) bir işlem modeli kullanır. İşlemler, tamamlandığında bir geri çağrı (callback) veya sinyal ile bildirilir. Bu model, yüksek verimli, eşzamanlı ve asenkron işlemler için idealdir.

PERFORMANS

- Reaktif programlama, çok sayıda isteği aynı anda yönetebilir ve işlemciyi daha verimli kullanır. İstekler tamamlanana kadar threadler serbest bırakılır, bu da daha yüksek ölçeklenebilirlik sağlar.

VERİ YÖNETİMİ

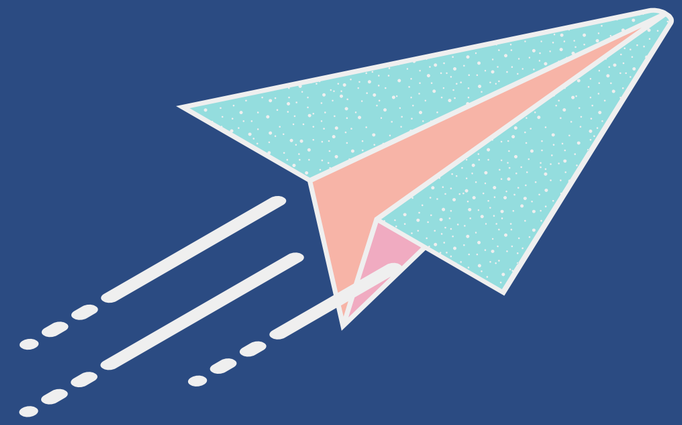
- JReaktif programlama, genellikle R2DBC (Reactive Relational Database Connectivity) gibi reaktif veri erişim teknolojilerini kullanır. Bu, aynı veritabanı işlemlerini reaktif bir şekilde yönetir, ancak geleneksel JPA gibi ORM desteklemeyebilir.

KOD YAPISI

- Reaktif programlama, Mono, Flux gibi reaktif veri türleriyle çalışır ve işlemler reaktif operatörler kullanılarak tanımlanır. Kod yapısı asenkron olduğundan, hata yönetimi ve veri işleme reaktif yaklaşımlarla yapılır.

KULLANIM ALANI

- Yüksek eşzamanlılık, düşük gecikme süreleri ve yüksek performans gerektiren uygulamalar (örneğin, gerçek zamanlı veri işleme, mikroservisler) için idealdir.

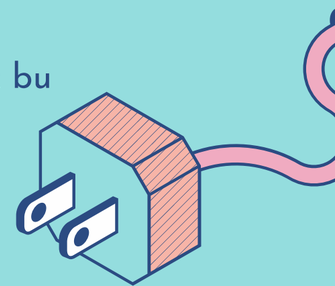


JPA'DA KULLANDIĞIMIZ METODLAR

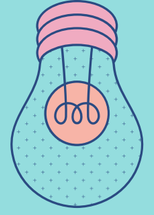
- **@Entity:** Bu, UserEntity sınıfının bir JPA varlığı olduğunu belirtir. JPA, bu sınıfın bir veritabanı tablosuna karşılık geldiğini anlar.
- **@Table(name = "intern_table"):** Bu, veritabanındaki tablonun adının intern_table olduğunu belirtir.
- **@Id ve @GeneratedValue:** id alanının bu varlık için birincil anahtar olduğunu belirtir ve değerin otomatik olarak oluşturulmasını sağlar. GenerationType.IDENTITY, veritabanının otomatik olarak bir değer atamasını sağlar (genellikle birincil anahtarın artan bir sayı olması durumunda).
- **@Repository:** Bu, bu arayüzün bir veri erişim bileşeni olduğunu belirtir.
- **JpaRepository<UserEntity, Long>:** JpaRepository JPA'nın sağladığı bir arayüzdür ve temel CRUD operasyonları ile bazı ek yöntemleri içerir. UserEntity varlığı üzerinde işlem yapacak ve birincil anahtarın türü Long olacaktır.
- **findAll():** Veritabanındaki tüm kayıtları getirir.
- **findById(ID id):** Belirli bir ID'ye sahip olan kaydı getirir.
- **save(S entity):** Yeni bir kayıt ekler veya mevcut bir kaydı günceller.
- **deleteById(ID id):** Belirli bir ID'ye sahip olan kaydı siler.
- **@Service:** Bu sınıfın bir servis bileşeni olduğunu belirtir. İş mantığını içerir ve genellikle veri erişim katmanından veri alır ve işlemler yapar.
- **@Autowired:** Bu, Spring'in UserRepository ve UserConverter nesnelerini otomatik olarak bu sınıfa enjekte etmesini sağlar.
- **getAllUser():** UserRepository kullanarak tüm UserEntity nesnelerini getirir. Eğer tablo boşsa bir RuntimeException fırlatır.
- **@RestController:** Bu sınıfın bir RESTful web servisi olduğunu belirtir. JSON formatında veri döner.
- **@RequestMapping("/users"):** /users URL yolunu bu denetleyiciye yönlendirir.
- **@GetMapping:** HTTP GET isteğiyle çağrılır. getAllUsers() yöntemi tüm kullanıcıları getirir ve bir HTTP 200 yanıtı döner.

Özet

Bu kodlar, temel JPA özelliklerini kullanarak bir kullanıcı veritabanı ile etkileşimde bulunan bir Spring Boot uygulaması oluşturur. JPA, veritabanı işlemlerini basitleştirir ve CRUD (Create, Read, Update, Delete) işlemleri için kolay bir arayüz sağlar. JpaRepository kullanarak bu işlemleri kolayca yapabilir ve veritabanı ile nesne tabanlı etkileşimi yönetebilirsiniz.



REAKTİF PROGRAMALAMADA KULLANDIĞIMIZ METODLAR



- **@Configuration:** Bu sınıfın bir konfigürasyon sınıfı olduğunu belirtir. Spring'in uygulama bağlamında bu sınıftaki @Bean yöntemlerini kullanarak bean'leri oluşturmasını sağlar.
- **@EnableR2dbcRepositories:** R2DBC (Reactive Relational Database Connectivity) ile reaktif veritabanı erişimini etkinleştirir ve belirtilen paket içerisindeki repository'leri tarar.
- **R2dbcEntityTemplate:** Reaktif veri işlemleri için kullanılan bir şablondur. Veritabanı işlemlerini reaktif bir şekilde yapmanıza olanak tanır.
- DatabaseClient: Reaktif veritabanı işlemleri yapmak için kullanılan bir bileşendir. SQL sorguları yapmanıza ve veritabanı ile reaktif olarak etkileşimde bulunmanıza imkan tanır.
- **@RestController:** Bu sınıfın bir RESTful web servisi olduğunu belirtir ve HTTP isteklerine yanıt verir.
- **Flux<UserEntity>:** Flux veri akışlarını temsil eden bir reaktif tiptir. Bu, birden fazla UserEntity nesnesinin akışını temsil eder. Reaktif akışlar veriyi aşamalı olarak işler ve yüksek performans sağlar.
- **ResponseEntity<Flux<UserEntity>>:** Bu, HTTP yanıtı olarak reaktif bir veri akışını döndürür. Reaktif veri akışları, istemcinin yanıtı beklemeden işlemlere devam etmesine olanak tanır.
- **Flux<UserDataTransfer>:** Flux kullanarak UserEntity nesnelerini UserDataTransfer nesnelerine dönüştürür. map() operatörü, her UserEntity nesnesini bir UserDataTransfer nesnesine dönüştürür.
- **@Table:** Bu, UserEntity sınıfının intern_table adlı veritabanı tablosunu temsil ettiğini belirtir.
- **@Id:** Bu alanın birincil anahtar olduğunu belirtir.
- **ReactiveCrudRepository:** Reaktif CRUD işlemleri için kullanılan bir arayüzdür. ReactiveCrudRepository, veritabanı işlemlerini reaktif olarak gerçekleştirir ve findAll(), findById(), save(), deleteById() gibi temel CRUD işlemleri sağlar.
- **Flux<UserEntity>:** findAll() metodu, veritabanındaki tüm UserEntity nesnelerini reaktif bir akış olarak döndürür.
- **Flux<UserEntity>:** Servis katmanında, tüm kullanıcıları reaktif bir akış olarak döndürür. Bu, veriyi aşamalı olarak işleyebilmenize ve büyük veri kümeleriyle verimli bir şekilde çalışmanıza olanak tanır.

Özet

Reaktif programlama, uygulamanızın veritabanı ve diğer veri kaynakları ile asenkron olarak etkileşimde bulunmasını sağlar. Flux ve Mono gibi reaktif türler, veri akışlarını temsil eder ve büyük veri kümeleriyle etkili bir şekilde çalışmanıza olanak tanır. Reaktif programlama ile, verileri tüketiciye iletmeden önce işleyebilir, hataları yönetebilir ve yüksek performanslı uygulamalar oluşturabilirsiniz. Reaktif CRUD işlemleri de veritabanı işlemlerini reaktif bir şekilde gerçekleştirir, böylece uygulamanız daha hızlı ve verimli hale gelir.

NETTY VE TOMCAT

NETYY

Kullanım Alanı: Yüksek performanslı, düşük gecikmeli ve asenkron uygulamalar, özellikle mikrosunucu (microservice) mimarilerinde ve reaktif programlama ile kullanılan uygulamalarda tercih edilir.
Mimarisi: Non-blocking (bloklamayan) ve asenkron bir model kullanır. Bu, aynı anda çok sayıda isteği daha az kaynakla işleyebilmesine olanak tanır.
Reaktif Programlama ile Kullanımı: Reaktif programlama paradigmasıyla uyumlu çalışır. Bu, isteklerin ve yanıtların işlenmesinde akış (stream) tabanlı, geri çağırma (callback) veya reaktif aboneliklerin (reactive subscriptions) kullanıldığı anlamına gelir.
Öne Çıkan Özellik: Yüksek performans, düşük kaynak kullanımı, reaktif programlama desteği.

Yüksek eşzamanlılık, düşük gecikme süresi, asenkron işleme ve reaktif programlama desteği gerektiren uygulamalar için idealdir. Bu, özellikle mikro hizmetler ve modern web uygulamaları gibi yüksek ölçeklenebilirlik gerektiren ortamlarda tercih edilmesini sağlar.

TOMCAT

Kullanım Alanı: Geleneksel Java EE uygulamaları, özellikle Servlet ve JSP tabanlı uygulamalar için yaygın olarak kullanılır.
Mimarisi: Bloklama (blocking) tabanlı bir model kullanır. Her istek için bir iş parçacığı (thread) ayrılır, bu da yüksek trafikli uygulamalarda kaynak tüketimini artırabilir.
JPA ile Kullanımı: JPA (Java Persistence API) ile birlikte kullanıldığında, tipik olarak senkron işlemler gerçekleştirilir. Her veri tabanı işlemi, genellikle işlem tamamlanana kadar mevcut iş parçacığını bloke eder.
Öne Çıkan Özellik: Kolay kullanım, olgunluk ve geniş topluluk desteği, geleneksel web uygulamaları için uygunluk.

Geleneksel, senkron ve iş parçacığı (thread) tabanlı işleme ihtiyaç duyulan, genişletilebilir ve olgun bir ortamda çalışan uygulamalar için uygundur.

NELER GÖZLEMLEDİK?

JPA vs REAKTİF PROGRAMLAMA
YANIT SÜRESİ ve KULLANICI DENEYİMİ

```
Started InternApplication in 4.664 seconds
```

JPA

JPA tabanlı uygulamalar genellikle yanıt sürelerinde yavaşlık gösterebilir, çünkü veri erişimi sırasında iş parçacıkları bekleyebilir. Bu, yüksek gecikmelere ve kötü kullanıcı deneyimine yol açabilir.

```
Started InternApplication in 2.671 seconds
```

REAKTİF PROGRAMLAMA

Reaktif programlama, daha düşük yanıt süreleri sağlar çünkü veri akışları asenkron olarak işlenir. Uygulama, verileri işlemeyi ve iletmeyi daha verimli bir şekilde yapar, bu da kullanıcı deneyimini iyileştirir.