Tekst do prezentacji Spring Data

Slajd 2. Czym jest Spring Data?

Spring Data jest jednym z wielu modułów udostępnianych przez framework Spring. Jego głównym zastosowaniem jest możliwość łatwej i przejrzystej implementacji warstwy dostępu do danych.

Spring Data umożliwia między innymi:

- ułatwiony dostęp relacyjnych i nierelacyjnych baz danych np. MySQL, MongoDB
- wydzielenie poziomu abstrakcji dla mapowania relacyjno - obiektowego (ORM)
- tworzenie dynamicznych, automatycznie implementowanych zapytań do przechowywanych danych
- integrację i ujednolicenie dostępu do danych z różnych typów źródeł (baz danych, systemów plików, chmury)
- wyeliminowanie tzw. boilerplate code w czasie tworzenia kolejnych repozytoriów

Slajd 3. Najważniejsze moduły Spring Data

Spring Data Commons - stanowi część projektu Spring Data, która zapewnia dzieloną infrastrukturę dla innych modułów. Zawiera między innymi neutralne repozytoria, interfejsy oraz metadane dla klas Javy.

Spring Data JPA - pozwala na łatwą implementację repozytoriów opartych na Java Persistence Api. Zapewnia wysokie wsparcie dla budowania warstwy dostępu do danych z wykorzystaniem technologii Spring oraz automatyczną implementację wielu metod do manipulacji danymi.

Spring Data MongoDB - pozwala na łatwą integrację aplikacji z nierelacyjną bazą danych MongoDB. W znacznym stopniu ułatwia mapowanie klasy Javy na kolekcje oraz dokumenty wykorzystywane przez MongoDB.

Spring Data Redis - daje możliwość prostej konfiguracji oraz dostępu do bazy Redis z poziomu aplikacji. Oferuje zarówno nisko jak i wysoko poziomowy poziom abstrakcji do interakcji z przechowywanymi danymi.

Spring Data REST - pozwala na swobodne budowanie aplikacji opartych o styl architektoniczny REST. Poprzez między innymi analizę modelu domeny aplikacji pozwala na konwersję do zasobów sterowanych hipermediami.

Slajd 4. Architektura Spring Data

Architektura w Spring Data opiera się na interfejsie Repository, który stanowi podstawową bazę dla innych interfejsów, używanych podczas implementacji.

T oznacza typ, który jest przechowywany w danym repozytorium natomiast ID jest typem klucza głównego, który zawiera się w klasie wskazanej przez T. Co ciekawe, interfejs Repository nie ma żadnej metody.

Często używanymi interfejsami, które implementują Repository są CrudRepository oraz PagingAndSortingRepository.

CrudRepository zapewnia głównie metody pokrywające się z operacjami CRUD (Create, Read, Update, Delete).

PagingAndSortingRepository zapewnia metody do bardziej zaawansowanej paginacji oraz sortowania rekordów.

Slajd 5. Architektura Spring Data

Implementujemy interfejsy aby stworzyć pewien model/opis sposobu manipulacji encją między systemem, a bazą danych.

Tworzenie repozytorium obejmuje 3 kroki:

- konfiguracja Spring
- stworzenie encji wraz z kluczem głównym (obiektu reprezentującego dane)
- stworzenie interfejsu, który implementuje Repository, najczęściej pośrednio poprzez CrudRepository lub PagingAndSortingRepository.

Możemy tworzyć różne odmiany interfejsów.

Każdy z nich może się różnić poziomem dostępu do metody.

Slajd 6. Metody udostępniane przez interfejs CrudRepository

Interfejs CrudRepository zapewnia domyślną implementację dla podstawowych operacji CRUD. Oznacza to, że wystarczy rozszerzyć nasz interfejs o CrudRepository, a metody odpowiadające za wspomniane operacje będą dostępne, bez potrzeby pisania dodatkowego kodu.

Metoda save - zapisuje encję przekazaną jako parametr oraz ponownie zwraca jej instancję w celu umożliwienia na niej dalszych operacji.

Metoda saveAll - zapisuje kolekcję encji przekazaną jako parametr oraz ponownie zwraca jej instancję w celu umożliwienia na niej dalszych operacji.

Uwaga: Należy zauważyć, że metody save oraz saveAll odpowiadają nie tylko za dodawanie, ale także za aktualizację encji. Jeżeli jako parametr metody przekazana zostanie istniejąca już encja, lecz o zmienionych atrybutach, to nie zostanie ona ponownie dodana, lecz wartości jej pól zostaną zaktualizowane.

Metoda findById - znajduje encję posiadającą dane id.

Metoda findAll - zwraca wszystkie instancje danej encji.

Metoda count - zwraca liczbę dostępnych encji.

Metoda delete - usuwa daną encję.

Metoda deleteAll - usuwa wszystkie istniejące instancje danej encji.

Metoda existById - zwraca wartość logiczną w zależności od tego czy encja o danym id istnieje w bazie.

Slajd 7. Autoimplementacja metod

Spring Data zapewnia znaczne udogodnienie w procesie implementacji repozytoriów poprzez zapewnienie automatycznej implementacji metod, jedynie na podstawie ich nazwy.

Spring Data wykrywa takie słowa kluczowe jak find, get, count oraz liczne modyfikatory - Top, Bottom, Distinct…

Uwaga: Bardzo przydatną funkcję stanowi wykrywanie i autouzupełnianie możliwych implementacji metody już przy wpisaniu początku nazwy metody.

Slajd 8. Definiowanie własnych zapytań do bazy danych

Definiowanie własnych zapytań: Oprócz automatycznej implementacji standardowych zapytań, Spring Data pozwala także na definiowanie własnych zapytań do bazy danych. Najpopularniejsze rozwiązanie stanowi wykorzystanie wykorzystanie adnotacji @Query.

Aby dana metoda zwracała rezultat zapytania do bazy danych należy:

- oznaczyć ją adnotacją @Query gdzie jako parametr value określić żądane zapytanie SQL
- aby mieć pewność właściwego działania wskazane jest ustawienie flagi nativeQuery na wartość true
- wykorzystać adnotację @Param do oznaczenia parametrów metody jako parametrów zapytania SQL
- pamiętać o możliwości wykorzystania klauzul języka SQL - order by, where, limit…

Uwaga: Należy upewnić się czy zapytanie zwraca pojedynczą encję czy kolekcję encji, by w zależności od tego właściwie dostosować typ zwracany przez metodę.

Slajd 9. JPA - Java Persistence API

JPA, czyli Java Persistence API to ORM przeznaczony do języka programowania Java.

ORM(Object-Relational Mapping) służy do odzwierciedlania architektury obiektowej systemu na relacyjną bazę danych.

Slajd 10. Przykładowe implementacje JPA

W Javie występuje wiele implementacji JPA - Hibernate, OpenJpa, EclipseLink oraz DataNucleus.

Poniższa prezentacja skupi się na najpopularniejszej z nich, czyli Hibernate.

Slajd 11. Jak rozpocząć pracę z JPA?

Prace z JPA rozpoczynamy od modyfikacji pliku projektu – czyli build.gradle lub pom.xml.

Skupimy się na środowisku budowania gradle, natomiast w przypadku mavena, robi się bardzo analogicznie.

Slajd 12. Konfiguracja gradle.build

Do poprawnego działania JPA wymagane będzie dodanie kilku zależności:

Spring Boot JPA – obsługa JPA
implementation 'org.springframework.boot:spring-boot-starter-data-jpa:3.0.0'

PostgreSQL – sterownik do obsługi bazy danych
implementation 'org.postgresql:postgresql:42.5.1'

Lombok – dodatkowa biblioteka dodająca adnotacje generujące automatycznie gettery, settery, konstruktory itp. Znacznie upraszczająca i upiększająca kod. Zapobiega powtarzalności kodu.
compileOnly 'org.projectlombok:lombok:1.18.20'
annotationProcessor 'org.projectlombok:lombok:1.18.20'

Slajd 13. Budowanie projektu - model

Model to klasa, która posiada pełne odzwierciedlenie po stronie bazy. To znaczy wszystkie jej pola znajdą się w bazie danych w tabeli o nazwie i kolumnach analogicznych jak obiekt tej klasy.

Występują w niej przykładowe adnotacje.

Slajd 14. Budowanie projektu - repozytorium

Następnie implementujemy repozytorium – czyli interfejs służący do późniejszej komunikacji z bazą danych.

Najważniejszą adnotacją jest @Repository, która oznacza interfejs jako repozytorium.
Domyślnie JPA umożliwia budowanie automatycznych zapytań za pomocą nazwy funkcji, natomiast jest możliwość zdefiniowania własnego zapytania za pomocą @Query.

Slajd 15. Budowanie projektu - serwis

Na końcu implementujemy Serwis, czyli klasę, która odpowiada za logikę biznesową aplikacji. W jej implementacji odwołujemy się do beanów repozytoriów. Analogicznie jak wcześniej zamieszczamy odpowiednią adnotację, w tym przypadku @Service. Dodajemy pola o typach utworzonych repozytoriów, następnie przekazujemy je wszystkie w konstruktorze.

Nad konstruktorem dodajemy adnotację @Autowired, czyli wstrzykujemy zależności do wcześniej utworzonych beanów. Proces taki zapobiega tworzeniu wielu takich samych obiektów, dzięki czemu zyskujemy na szybkości i pamięci.

Slajd 16. Połączenie z bazą danych

Podpięcie bazy danych następuje w pliku application.properties

spring.datasource.url=jdbc:{sterownik_bazy}://${address}:{port}/{baza} – nakierowanie na bazę danych
spring.datasource.username = user – login do operatora bazy danych
spring.datasource.password = password – hasło do operatora bazy danych
spring.jpa.hibernate.ddl-auto = update – tryb w jakim ma się uruchamiać JPA, update – aktualizacja bazy

Slajd 17. Testy

Do testów stworzono adnotację @DataJpaTest, którą należy umieszczać nad nazwą klasy. Dostarcza ona środowisko z bazą danych, dzięki której można przetestować w przyjazny sposób działanie zapytań.

Slajd 18. MongoDB

MongoDB - nierelacyjna baza przechowująca dane w postaci dokumentów z pomocą JSON-a. Należy do grupy języków NoSQL. Posiada ważne zalety takie jak bardzo wysoka skalowalność oraz elastyczność. Pomimo, że MongoDB nie jest bazą relacyjną, to JPA obsługuje ten typ bazy.

Slajd 19. Połączenie z MongoDB

Zależność testCompile flapdoodle.embed.mongo pozwala na automatyczne pobieranie oraz uruchamianie instancji MongoDB przez aplikację, dzięki temu podczas wykonywania testów nie musimy mieć lokalnie uruchomionej bazy danych.

Slajd 20. Przykład klasy modelu w MongoDB

Adnotacja Document, podobnie jak Entity dla baz relacyjnych, oznacza klasę jako obiekt domenowy który chcemy zachować w bazie danych.

W przypadku bardziej rozbudowanych klas modelu, pojawiają się także inne różnice w adnotacjach, lecz różnice te nie są duże.

Pozostałe klasy wykorzystywane do komunikacji z bazą danych wyglądają bardzo podobnie jak te zaprezentowane w przykładzie komunikacji z PostgreSQL.