Spring Data REST and Microservices

Spring Data JPA

- Spring Data JPA, частина сімейства Spring Data, дозволяє легко впровадити сховища на основі JPA. Цей модуль стосується розширеної підтримки рівнів доступу до даних на основі JPA. Це спрощує створення Spring-програм, що використовують технології доступу до даних.
- Для виконання простих запитів, а також виконання пагінації та аудиту потрібно написати занадто багато типового коду. Spring Data JPA має на меті значно покращити реалізацію рівнів доступу до даних, зменшивши зусилля до необхідної кількості. Розробник пише власні інтерфейси сховища, включаючи користувацькі методи пошуку, а Spring забезпечує реалізацію автоматично.

Пакети jpa

- *Repository<T, ID extends Serializable>* interface is a marker interface that has two purposes:
  - It captures the type of the managed entity and the type of the entity's id.
  - It helps the Spring container to discover the "concrete" repository interfaces during classpath scanning.
- *CrudRepository<T, ID extends Serializable>* interface provides CRUD operations for the managed entity.
- *PagingAndSortingRepository<T, ID extends Serializable>* interface declares the methods that are used to sort and paginate entities that are retrieved from the database.
- *QueryDslPredicateExecutor<T>* interface is not a "repository interface". It declares the methods that are used to retrieve entities from the database by using QueryDsl *Predicate* objects.

Spring JPA. One-to-many

```
@Entity
public class Library {...
    @OneToMany(mappedBy = "library", cascade = CascadeType.ALL)
    private Set<Book> books = new HashSet<>();
...}
@Entity
public class Book {…
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "library_id")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Library library;
…}
```

One-to-many. Додавання даних

Контролер первинної таблиці

```
@PostMapping
    public ResponseEntity<Library> create(@RequestBody Library library) {
        Library savedLibrary = libraryRepository.save(library);
        URI location = ServletUriComponentsBuilder.
                fromCurrentRequest().path("/{id}")
            .buildAndExpand(savedLibrary.getId()).toUri();
        return ResponseEntity.created(location).body(savedLibrary);
    }
```

Оновлення даних

```
@PutMapping("/{id}")
    public ResponseEntity<Author> update(@PathVariable Integer id,
                                         @RequestBody Author library) {
        Optional<Author> optionalLibrary = authorRepository.findById(id);
        if (!optionalLibrary.isPresent()) {
            return ResponseEntity.unprocessableEntity().build();
        }
        library.setId(optionalLibrary.get().getId());
        authorRepository.save(library);
        return ResponseEntity.noContent().build();
    }
```

Spring Data REST

Spring Data REST є частиною загального проекту Spring Data і спрощує створення веб-служб REST, керованих гіпермедіа, над репозиторіями Spring Data.
Spring Data REST будується поверх репозиторіїв Spring Data, аналізує модель домену вашої програми та виставляє керовані гіпермедіа ресурси HTTP для сукупностей, що містяться в моделі.

REST API

Концепція REST

REST(Representational State Transfer) – стиль архітектури розподілених систем.

Описаний і популяризований 2000

року Роєм Філдінгом, одним із

творців протоколу HTTP.

В основі REST закладено принципи функціонування Web і, зокрема, можливості HTTP.

REST API

REST-стиль, зручний для створення Single Page Application, що часто використовують спеціальні javascript-фреймворки типу Angular, React або Vue.js. За допомогою веб-служб на основі веб-API можна отримувати доступ до даних з інших додатків різних технологій та платформ.

REST

URI(Uniform Resource Identifier) – це ідентифікатор ресурсу, адреса, за якою знаходиться інформація.

Приклад URI - http://www.microsoft.com/uk-ua/default.aspx

Ім'я методу HTTP	Тип взаємодії
GET	Отримання джерела, idempotent
POST	Створення нового джерела
PUT	Оновлення існуючого джерела, idempotent
DELETE	Видалення джерела

Idempotent – не залежить від кількості викликів. Результат завжди однаковий.

REST

Основні принципи REST:

Client-Server–поділ відповідальності

Stateless–спілкування між клієнтом і сервером не має на увазі зберігання стану

Cacheable–клієнти можуть кешувати відповіді. Відповідь має явно чи не явно вказувати клієнту, чи може той його кешувати.

Layered System –клієнт не може точно знати, чи підключений він на пряму до сервера або користується проміжним проксі сервером.

Uniform Interface –уніфікований інтерфейс між клієнтом і сервером заснований на URI і використання HTTP методу для визначення операції, яку необхідно виконати на сервері.

Spring Data REST

- Розкриває доступний REST API для вашої моделі домену, використовуючи HAL як тип носія.
- Виставляє ресурси колекції, предметів та асоціацій, що представляють вашу модель.
- Підтримує пагінацію за допомогою навігаційних посилань .
- Дозволяє динамічно фільтрувати ресурси колекції.
- Виставляє виділені ресурси пошуку для методів запитів, визначених у ваших сховищах.

Spring Data REST

•Дозволяє підключитись до обробки запитів REST , обробляючи Spring ApplicationEvents.
•Виставляє метадані про модель, виявлену як схема ALPS та JSON.
•Дозволяє визначати конкретні уявлення клієнта за допомогою проекцій .
•Поставляється спеціальний варіант браузера HAL, щоб використовувати відкриті метадані.
•В даний час підтримує JPA, MongoDB, Neo4j, Solr, Cassandra, Gemfire.
•Дозволяє розширене налаштування доступних ресурсів за замовчуванням.

Maven

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId
    <artifactId>spring-boot-starter-data-rest</artifactId></dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

Оголошення репозиторію

```
@RepositoryRestResource(
        collectionResourceRel = "users",
        path = "users")
public interface UserRepository extends
        PagingAndSortingRepository<WebsiteUser, Long> {
    List<WebsiteUser> findByName(
        @Param("name") String name);
}
```

Приклад зі зв’язками

https://www.baeldung.com/spring-data-rest-relationships

Анотація RepositoryRestResource

Spring Data REST використовує, RepositoryDetectionStrategy щоб визначити, чи експортується сховище як ресурс REST. Перелік RepositoryDiscoveryStrategiesвключає такі значення

`DEFAULT`	Виставляє всі інтерфейси загальнодоступного сховища, але враховує `exported` прапор `@(Repository)RestResource`.
`ALL`	Виставляє всі сховища незалежно від видимості типу та анотацій.
`ANNOTATED`	Виставляються лише сховища, котрі анотовано `@(Repository)RestResource`, якщо для їх `exported` прапора не встановлено значення `false`.
`VISIBILITY`	Виставляються лише анотовані загальнодоступні сховища.

Зміна базового url

```
@Configuration
class CustomRestMvcConfiguration {
 @Bean
public RepositoryRestConfigurer repositoryRestConfigurer() {
return new RepositoryRestConfigurer() {
@Override
public void configureRepositoryRestConfiguration(
RepositoryRestConfiguration config, CorsRegistry cors) {
config.setBasePath("/api");
} };
} }
```

Валідація даних

JSR 380 - це специфікація Java API для перевірки компонентів Jakarta EE та JavaSE. Це гарантує, що властивості компонента відповідають певним критеріям, використовуючи такі анотації, як *@NotNull* , *@Min* та *@Max* .

Ця версія вимагає Java 8 або новішої версії та використовує нові функції, додані в Java 8, такі як анотації типів та підтримка нових типів, таких як *Optional* та *LocalDate* .

Валідація даних

Відповідно до специфікації JSR 380, залежність *validation-api* містить стандартні API перевірки:

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
```

Hibernate Validator - це еталонна реалізація API перевірки.

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

Стандартні анотації з JSR

- ***@NotNull*** перевіряє, чи анотоване значення властивості не має значення *null* .
- ***@AssertTrue*** перевіряє, чи значення властивості є *істинним*.
- ***@Size*** перевіряє, чи анотоване значення має розмір між атрибутами *min* та *max* ; можна застосувати до властивостей *рядка* , *колекції* , *карти* та масиву.
- ***@Min*** перевіряє, що анотоване властивість має значення не менше атрибута *value* .
- ***@Max*** перевіряє, що анотоване властивість має значення не більше атрибута *value* .
- ***@Email*** перевіряє, чи вдастивість є адресою електронної пошти.

Стандартні анотації з JSR

- ***@NotEmpty*** перевіряє, чи властивість не є нульовим або порожнім значенням.
- ***@NotBlank*** можна застосувати лише до текстових значень і перевіряє, що властивість не має значення null або пробілів.
- ***@Positive*** та ***@PositiveOrZero*** застосовуються до числових значень і підтверджують, що вони суворо позитивні або додатні, включно з 0.
- ***@Negative*** та ***@NegativeOrZero*** застосовуються до числових значень і підтверджують, що вони строго або від'ємні, включно з 0.
- ***@Past*** та ***@PastOrPresent*** підтверджують, що значення дати є минулим або минулим, включаючи сьогодення.
- ***@Future*** *та* ***@FutureOrPresent*** підтверджують, що значення дати є в майбутньому або в майбутньому, включаючи сьогодення.