

Spring Data REST

<https://github.com/jirkapinkas/javadays-2020>

Spring Data REST

- K čemu je to dobré?
 - Generování API pro číselníky anebo další entity, pro které potřebujete jednoduché CRUD operace přes REST API.
 - https://www.reddit.com/r/java/comments/68hzgq/are_projects_like_spring_data_rest_viable_for/
 - Moje situace cca. v roce 2015:
 - Měl jsem za úkol vytvořit jednoduché REST API pro prakticky všechny tabulky v databázi (v té době cca. 50 tabulek), které by implementovalo všechny standardní operace (GET, POST, PUT, DELETE) a zpřístupňovalo databázi dalšímu systému. Mohl jsem pro každou tabulku vytvořit controller, service, repository, entity + DAO + mapper třídu, nebo použít Spring Data REST a vytvořit pouze entitu a repozitář a výsledek bude stejný. Hádejte co jsem si vybral? :-)

Hello World I.

- Spring Data REST je nadstavbou nad Spring Data projektem pro tvorbu jednoduchých REST WS na základě Spring Data repozitářů.
- Spring Data REST může být nadstavbou nad JPA, MongoDB, Neo4J, GemFire & Cassandra. V přednášce se zaměřím na JPA, ale základní principy budou fungovat na všech uvedených databázích.
- Základní 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>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Hello World II.

- Existuje několik proměnných, pomocí kterých je možné upravit výchozí chování Spring Data REST. Zdaleka nejdůležitější je:

```
spring.data.rest.base-path=/api
```

- Pomocí tohoto nastavení budou Spring Data REST repozitáře dostupné na URL: <http://localhost:8080/api> (ve výchozím nastavení by byly v ROOTu aplikace a mohly by kolidovat s ostatními controllery v aplikaci)
- Toto nastavení budu dále používat v celé přednášce.

Hello World III.

- Dále potřebujeme entitu a Spring Data interface:

```
@Entity
public class Dog {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;

    private double weight;

    @ManyToOne
    @JoinColumn(name = "person_id")
    private Person owner;

    // getters & setters
}

public interface DogRepository extends JpaRepository<Dog, Long> {
}
```

Hello World IV.

- Hotovo! Spring Data REST endpoint je nyní k dispozici na URL:
 - <http://localhost:8080/api/dogs>
- Co umí dál?
 - Řazení:
 - <http://localhost:8080/api/dogs?sort=name> (vzestupné řazení)
 - <http://localhost:8080/api/dogs?sort=name,asc>
 - <http://localhost:8080/api/dogs?sort=name,desc>
 - Řazení podle více atributů:
 - <http://localhost:8080/api/dogs?sort=weight,desc&sort=name,desc>
 - Stránkování (standardně číslované od nuly):
 - <http://localhost:8080/api/dogs?sort=name&page=1>
 - Definování maximálního počtu záznamů na stránce:
 - <http://localhost:8080/api/dogs?sort=name&page=0&size=100>

Poznámka:
Plus samozřejmě
detail záznamu,
insert, update &
delete klasickým
REST způsobem

HAL Explorer

- Základní online explorer pro procházení Spring Data REST API:

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

The screenshot shows the HAL Explorer web application. At the top, there's a dark header with the logo and navigation links: Theme, Layout, and About. Below the header, there's a search bar with the text "/api/" and a "Go!" button. The main content area is divided into two columns. The left column, titled "Links", contains a table with columns: Relation, Name, Title, HTTP, and Doc. The table lists three links: "persons", "dogs", and "profile", each with a set of navigation buttons (back, forward, search, etc.). The right column, titled "Response Status", shows "200 (OK)". Below that, "Response Headers" are listed: "content-type" (application/), "date" (Tue, 06 Oct), and "vary" (Origin, Acce). At the bottom, "Response Body" is shown as a JSON object with links to "persons", "dogs", and "profile".

Relation	Name	Title	HTTP	Doc
persons			< > > > >	
dogs			< > > > > >	
profile			< > > > > >	

Po přidání této
dependency bude
HAL Explorer na URL:
<http://localhost:8080/api>

Pokročilejší konfigurace

@Configuration

```
public class RestApiConfiguration implements RepositoryRestConfigurer {
```

@Autowired

```
private EntityManager entityManager;
```

@Bean

```
public HateoasPageableHandlerMethodArgumentResolver customResolver(  
    HateoasPageableHandlerMethodArgumentResolver pageableResolver) {  
    pageableResolver.setOneIndexedParameters(true);  
    pageableResolver.setFallbackPageable(PageRequest.of(0, 20));  
    pageableResolver.setMaxPageSize(Integer.MAX_VALUE);  
    return pageableResolver;  
}
```

@Override

```
public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {  
    // https://stackoverflow.com/questions/30912826/expose-all-ids-when-using-spring-data-rest  
    config.exposeIdsFor(entityManager.getMetamodel().getEntities().stream()  
        .map(Type::getJavaType)  
        .toArray(Class[]::new)  
    );  
}
```

```
}
```

První stránka nebude mít index "0", ale "1"

Když nedefinujeme atribut "page", pak se bude brát toto výchozí nastavení

Maximální počet stránek, ve výchozím nastavení 2000

Díky tomuto nastavení se bude také vracet IDčko entity (ve výchozím nastavení se nezobrazuje)

Poznámka: Zde je možné nastavit výrazně víc věcí

Nastavení repozitáře

- Když nechceme nějaký repozitář exportovat, použijeme toto nastavení:

```
@RepositoryRestResource(exported = false)
public interface InternalTableRepository extends JpaRepository<InternalTable, Long> {
}
```

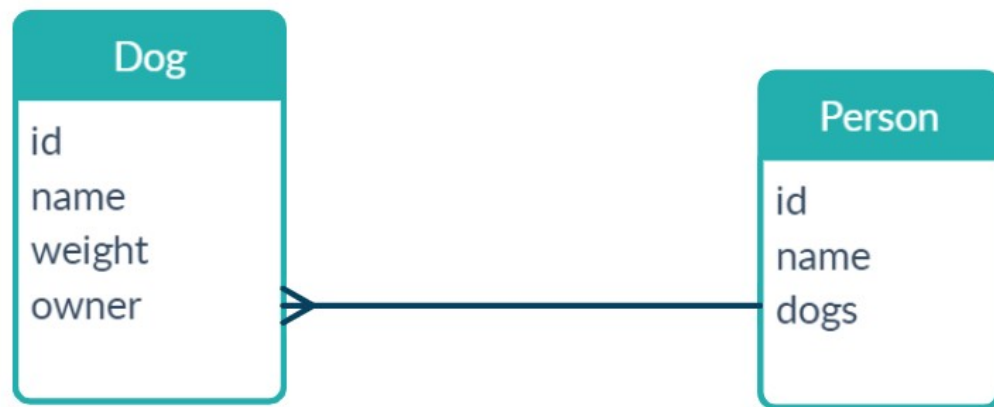
- Pokud již Spring Data používáme a chceme jenom přidat Spring Data REST k pár vybraným repozitářům, tak všude co chceme excludovat nemusíme psát `@RepositoryRestResource(exported = false)`, můžeme použít opačný přístup a exportovat pouze ty repozitáře, které budou mít `@RepositoryRestResource(exported = true)`:

```
spring.data.rest.detection-strategy=annotated
```

- Ve výchozím nastavení URL repozitáře končí na název entity v množném čísle. Například pro entitu Dog to je `"/dogs"`. Jak to změnit když to nevyhovuje?

```
@RepositoryRestResource(collectionResourceRel = "pets", path = "pets")
public interface DogRepository extends JpaRepository<Dog, Long> {
}
```

Příklad 1: Dogs & Persons (owners)



Poznámka:

Vytvořil jsem entity: Dog, Person

a repozitáře: DogRepository, PersonRepository

http://localhost:8080/api/dogs/1

```
{
  "id" : 1,
  "name" : "Hafík",
  "weight" : 0.0,
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/dogs/1"
    },
    "dog" : {
      "href" : "http://localhost:8080/api/dogs/1"
    },
    "owner" : {
      "href" : "http://localhost:8080/api/dogs/1/owner"
    }
  }
}
```

Samotný obsah zprávy

Odkazy na další relevantní záznamy

Pro atribut "owner" (Person) existuje PersonRepository repozitář, tudíž je zde link, pomocí kterého je možné získat detail tohoto záznamu

```
@Entity
public class Dog {

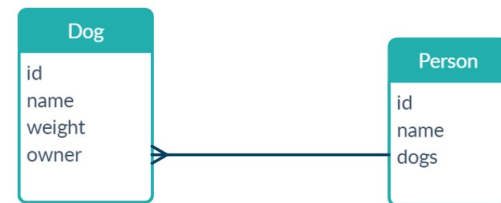
    @Id
    @GeneratedValue
    (strategy = GenerationType.IDENTITY)
    private long id;

    private String name;

    private double weight;

    @ManyToOne
    @JoinColumn(name = "person_id")
    private Person owner;

    // getters & setters
}
```

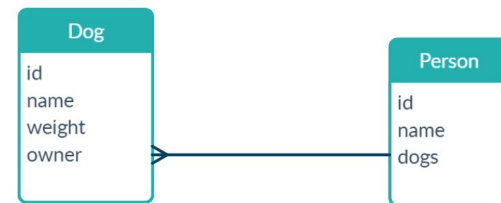


http://localhost:8080/api/dogs

```
{
  "_embedded" : {
    "dogs" : [ { ... }, { ... } ]
  },
  "_links" : {
    "first" : {
      "href" : "http://localhost:8080/api/dogs?page=1&size=20"
    },
    "self" : {
      "href" : "http://localhost:8080/api/dogs"
    },
    "next" : {
      "href" : "http://localhost:8080/api/dogs?page=2&size=20"
    },
    "last" : {
      "href" : "http://localhost:8080/api/dogs?page=19&size=20"
    },
    "profile" : {
      "href" : "http://localhost:8080/api/profile/dogs"
    }
  },
  "page" : {
    "size" : 100,
    "totalElements" : 366,
    "totalPages" : 4,
    "number" : 1
  }
}
```

Zde máme list záznamů. Z meta-informací jsou k dispozici odkazy na:

- první stránku
- aktuální stránku
- další stránku
- poslední stránku
- informace o kontraktu služby (něco jako OpenAPI / Swagger)



http://localhost:8080/api/persons/1

```
{
  "id" : 1,
  "name" : "Jirka",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api/persons/1"
    },
    "person" : {
      "href" : "http://localhost:8080/api/persons/1"
    },
    "dogs" : {
      "href" : "http://localhost:8080/api/persons/1/dogs"
    }
  }
}
```

```
@Entity
public class Person {

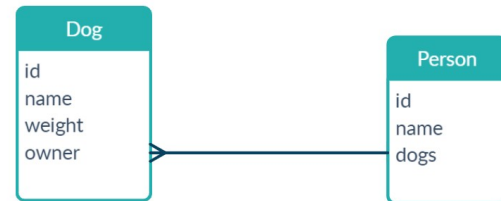
    @Id
    @GeneratedValue
    (strategy = GenerationType.IDENTITY)
    private long id;

    private String name;

    @OneToMany(mappedBy = "owner")
    private List<Dog> dogs;

    // getters & setters
}
```

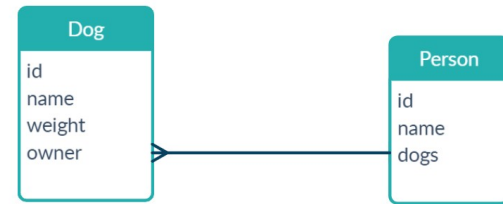
Odkaz pro získání listu objektů typu Dog
(protože existuje repozitář DogRepository)



http://localhost:8080/api/persons/1/dogs

```
{
  "_embedded" : {
    "dogs" : [ {
      "id" : 1,
      "name" : "Hafík",
      "weight" : 0.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api/dogs/1"
        },
        "dog" : {
          "href" : "http://localhost:8080/api/dogs/1"
        }
      },
      "owner" : {
        "href" : "http://localhost:8080/api/dogs/1/owner"
      }
    }
  ], {
```

```
    "id" : 2,
    "name" : "Ňafík",
    "weight" : 0.0,
    "_links" : {
      "self" : {
        "href" : "http://localhost:8080/api/dogs/2"
      },
      "dog" : {
        "href" : "http://localhost:8080/api/dogs/2"
      },
      "owner" : {
        "href" : "http://localhost:8080/api/dogs/2/owner"
      }
    }
  ]
},
  "_links" : {
    "self" : {
      "href" : "http://localhost:"
```

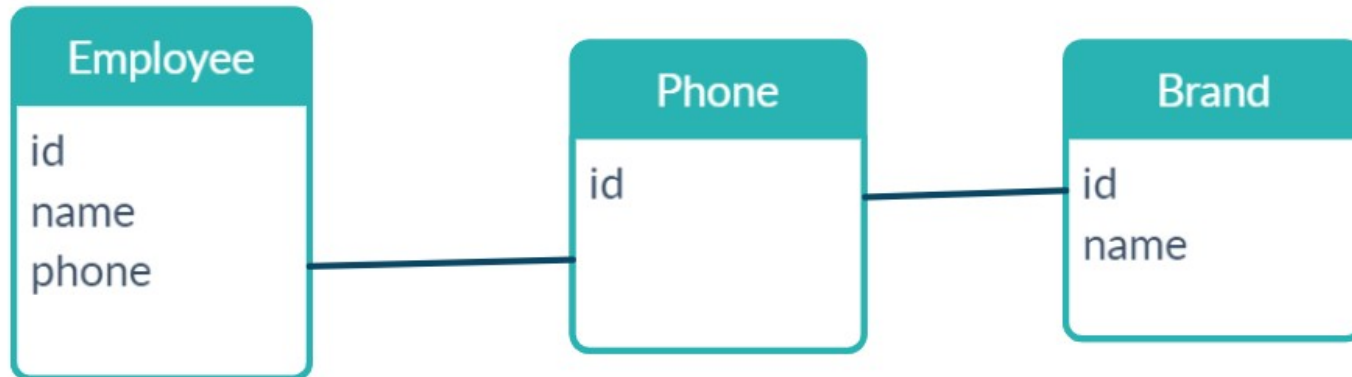


Příklad 2: Employee & Phone (& Brand)

A)



B)



Příklad 2 A)

- Mějme tyto dvě entity:
 - Employee
 - Phone
- Employee obsahuje unidirectional @OneToOne vazbu na Phone a rozdíl oproti předcházejícímu příkladu je, že neexistuje PhoneRepository (nebo existuje, ale má nastavené “exported = false”).

```
@Entity
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;

    @OneToOne
    @JoinColumn(name = "phone_id")
    private Phone phone;

    // getters + setters
}

@Entity
public class Phone {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String brandName;

    // getters + setters
}
```


Příklad 2 A)

- Zaměstnanec s ID = 1 bude na URL:

- <http://localhost:8080/api/employees/1>

- Co bude výsledkem?

```
{
  "id" : 1,
  "name" : "Jirka",
  "phone" : {
    "id" : 1,
    "brandName" : "iPhone"
  },
  "links" : {
    "self" : {
      "href" : "http://localhost:8080/api/employees/1"
    },
    "employee" : {
      "href" : "http://localhost:8080/api/employees/1"
    }
  }
}
```

Když entita Phone nemá PhoneRepository (nebo má, ale je u něj nastavené “exported = false”), tak bude její obsah součástí entity Employee.

Poznámka: V tomto příkladě záměrně není bidirectional vazba mezi Employee a Phone kvůli tomu, aby nedošlo k cyklu!!! Takovou situaci v tomto případě musíme řešit sami!!!



Příklad 2 A)

- Jak to bude s efektivitou SELECTů?
 - SELECT, kterým se získá detail záznamu bude v tomto případě efektivní:
 - `select * from employee left join phone
on employee.phone_id = phone.id where employee.id = ?`
 - Poznámka: SELECT je kvůli čitelnosti zjednodušený, ale obsahuje to důležité
 - Získání listu záznamů ale moc efektivní nebude:
 - `select * from employee limit ?`
 - `select * from phone where id = ?`
 - `select * from phone where id = ?`
 - Poznámka: V databázi jsou v tabulce Employee dva záznamy. Pro získání informací o telefonech by se vykonalo tolik SELECTů, kolik je záznamů na aktuální stránce.

Příklad 2 A)

- Jak to změnit?

```
@NamedEntityGraph(name = Employee.GRAPH_PHONE, attributeNodes = { @NamedAttributeNode(value = "phone") })
@Entity
public class Employee {

    public static final String GRAPH_PHONE = "graph.Employee.phone";

    // zbytek je stejný
}

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

    @EntityGraph(Employee.GRAPH_PHONE)
    @Override
    List<Employee> findAll();

    @EntityGraph(Employee.GRAPH_PHONE)
    @Override
    List<Employee> findAll(Sort sort);

    @EntityGraph(Employee.GRAPH_PHONE)
    @Override
    Page<Employee> findAll(Pageable pageable);
}
```

Opět zjednodušený SELECT, který se provede:

```
select * from employee left join phone
on employee.phone_id=phone.id limit ?
```

Příklad 2 B)

- Co když je graf závislostí složitější? Přidal jsem další entitu (Brand) a unidirectional vazbu z Phone na Brand:

```
@Entity
public class Phone {

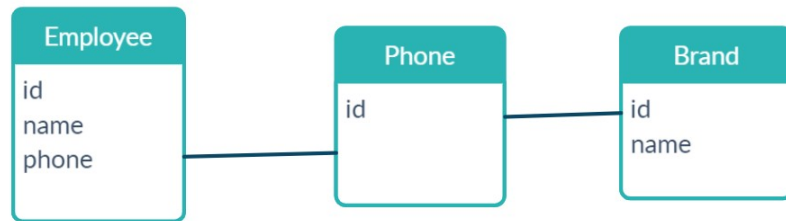
    @OneToOne
    @JoinColumn(name = "brand_id")
    private Brand brand;

    // zbytek je stejný
}

@Entity
public class Brand {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;
}
```



Příklad 2 B)

- Opět se dostaneme do stejného problému jako předtím, jenom tentokrát s entitou Brand. Řešením je rozšíření @NamedEntityGraph u entity Employee do následující podoby:

```
@NamedEntityGraph(  
    name = Employee.GRAPH_PHONE,  
    attributeNodes = {  
        @NamedAttributeNode(value = "phone", subgraph = "brand")  
    }, subgraphs = {  
        @NamedSubgraph(name = "brand", attributeNodes = @NamedAttributeNode("brand"))  
    })  
})
```

Hotovo :-)

Příklad 3: Základní filtrování

- Spring Data REST out-of-the-box nepodporuje filtrování záznamů, ale takovou funkcionalitu je možné ve spojení s Query DSL lehce doimplementovat.
- Implementace “EXACT MATCH” (==) funkcionality je jednoduchá, stačí přidat níže uvedenou dependency do pom.xml a rozšířit Spring Data repository interface způsobem uvedeným na další stránce:

```
<dependency>
  <groupId>com.querydsl</groupId>
  <artifactId>querydsl-jpa</artifactId>
</dependency>
<dependency>
  <groupId>com.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
  <version>${querydsl.version}</version>
  <classifier>jpa</classifier>
</dependency>
```

Příklad 3: Základní filtrování

```
@NoRepositoryBean
public interface SpringDataRestRepository<T, ID> extends Repository<T, ID>, QuerydslPredicateExecutor<T>, QuerydslBinderCustomizer {
    default void customize(QuerydslBindings bindings, EntityPath root) {
    }
}
```

Díky této anotaci se Spring Data JPA nebude snažit vytvářet implementaci tohoto interface.

```
@NoRepositoryBean
public interface SpringDataRestJpaRepository<T, ID> extends SpringDataRestRepository<T, ID>, JpaRepository<T, ID> {
}
```

```
public interface CarRepository extends SpringDataRestJpaRepository<Car, Long> {
}
```

```
@Entity
public class Car {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;

    private double price;
    // getter & setter
}
```

Základní filtrování

- Příklady:
 - Vypíše auta, která mají name = “Tesla Model S”
 - <http://localhost:8080/api/cars?name=Tesla%20Model%20S>
 - Vypíše auta, jejichž cena je 2.5 miliónu:
 - <http://localhost:8080/api/cars?price=2.5>

Pokročilejší filtrování

- U základního filtrování ale nemusíme zůstat, v metodě “customize” můžeme implementovat další způsoby filtrování:
 - <https://github.com/jirkapinkas/javadays-2020/blob/master/spring-data-rest/spring-data-rest-hello-world/src/main/java/com/example/helloworld/repository/SpringDataRestRepository.java>
 - Vypíše auta, která mají v názvu “Model X” (hledá se text “%Model X%”)
 - <http://localhost:8080/api/cars?name=%25Model%20X%25>
 - Poznámka: Místo “%” je zapotřebí posílat na server “%25”!!!
 - Vypíše auta, jejichž cena je v rozsahu 1 – 2 milióny:
 - <http://localhost:8080/api/cars?price=1&price=2>

Integrace s OpenAPI (Swagger)

- Knihovna Springdoc OpenAPI má přímou podporu pro Spring Data REST:

pom.xml:

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.4.8</version>
</dependency>
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-data-rest</artifactId>
  <version>1.4.8</version>
</dependency>
```

application.properties:

```
springdoc.swagger-ui.path=/swagger-ui.html
```

<http://localhost:8080/swagger-ui.html>

Security

- Pokud chceme zabezpečit Spring Data REST repozitář, pak máme několik možností:
 - Zabezpečení celého repozitáře:

```
@PreAuthorize("hasRole('ADMIN')")  
public interface DogRepository extends JpaRepository<Dog, Long> {  
}
```

- Zabezpečení jenom některých metod:

```
public interface DogRepository extends JpaRepository<Dog, Long> {  
  
    @Override  
    @PreAuthorize("hasRole('ADMIN')")  
    <S extends Dog> S save(S entity);  
  
    @Override  
    @PreAuthorize("hasRole('ADMIN')")  
    void delete(Dog entity);  
}
```


Ale pozor na to, že JpaRepository má hromadu metod, které mění stav entity! Tento přístup je náchylný na chyby!

Security

- Další možností je extendovat Repository interface, které neobsahuje žádné metody a nastavit mu pouze ty, které chceme, aby byly implementovány:

```
public interface DogRepository extends Repository<Dog, Long> {  
    Page<Dog> findAll(Pageable pageable);  
    Optional<Dog> findById(Long id);  
}
```

Syntaxe metod je stejná jako kdybychom extendovali JpaRepository a prováděli override method, které tam jsou definované.



- Nebo:

```
public interface DogRepository extends Repository<Dog, Long> {  
    Page<Dog> findAll(Pageable pageable);  
    Optional<Dog> findById(Long id);  
  
    @PreAuthorize("hasRole('ADMIN')")  
    <S extends Dog> S save(S entity);  
}
```

Gotcha's, Custom findBy metody

- Ve Spring Data REST fungují základní findBy metody:

```
public interface DogRepository extends JpaRepository<Dog, Long> {  
    List<Dog> findByName(String name);  
}
```

- Jak tuto operaci zavolat:
 - <http://localhost:8080/api/dogs/search/findByName?name=Haf%C3%ADk>
- Je možné s tím implementovat jednodušší operace, pokročilejší operace s tím neuděláte:
 - Není možné vracet něco jiného než entitu nebo projekci
 - @Modifying (DML) operace nejsou podporované!!!

Gotcha's

- Spring Data REST nemá rád situaci, když máte více repozitářů pro stejnou entitu. V tu chvíli není úplně definované pro jaký repozitář se vlastně endpoint vygeneruje, nebo jestli se vygeneruje vůbec nějaký.
- Kdy byste se do takové situace dostali?
 - Kdybyste měli custom Spring Data repozitáře, kde máte `@Query` a `@Modifying` `@Query` metody, které nepodporuje Spring Data REST a zapomenete tyto custom repozitáře oannotovat s `@RepositoryRestResource(exported = false)`
 - Kdybyste měli dva Spring Data REST repozitáře, jeden by implementoval `Repository` (v základu zde nejsou žádné operace, ale můžete je přidat) a druhý `JpaRepository` (jsou zde všechny operace) a k prvnímu repozitáři by mohli přistupovat všichni a ke druhému například pouze administrátoři. Pak je zapotřebí použít dvě rozdílné entity, viz. následující snímek.

Příklad 4: Note

- Jak mít dva repozitáře, které budou pracovat s jednou entitou:

```
@MappedSuperclass
public abstract class BaseNote {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String note;

    // gettery a settery
}

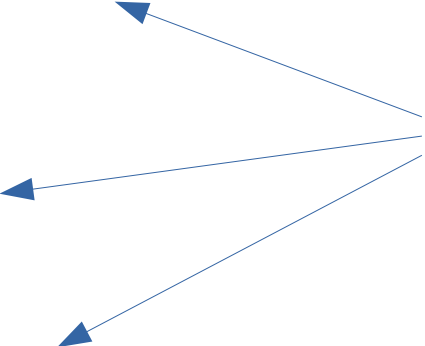
@Entity
@Table(name = "note")
public class ExportNote extends BaseNote {
}

@Entity
public class Note extends BaseNote {

    private String internalMemo;

    // gettery a settery
}
```

Máme dvě rozdílné entity,
které dědí svůj stav z
@MappedSuperclass třídy



Poznámka: V databázi je jedna
tabulka NOTE, která má sloupce:
ID, NOTE, INTERNAL_MEMO

Příklad 4: Note

K tomuto repozitáři bude mít přístup pouze uživatel s rolí ADMIN a bude mít k dispozici všechny operace včetně POST, PUT & DELETE

```
@PreAuthorize("hasRole('ADMIN')")  
public interface NoteRepository extends JpaRepository<Note, Long> {  
}
```

```
public interface ExportNoteRepository extends Repository<ExportNote, Long> {  
  
    // Když bude tato operace zapoznámkováná,  
    // tak půjde získat list záznamů, ale bez stránkování  
    // Page<ExportNote> findAll(Pageable pageable);  
  
    List<ExportNote> findAll();  
  
    Optional<ExportNote> findById(Long id);  
  
}
```

Tento repozitář bude veřejný a bude pomocí něj možné získat informace o poznámkách, ale nebude možné je měnit.

Events

- Předtím než se nějaká entita uloží, tak je možné zavolat nějakou událost:
 - <https://docs.spring.io/spring-data/rest/docs/current/reference/html/#events>
 - Toto jsem v životě využil jenom jednou. Pokud bychom chtěli něco flexibilnějšího, tak IMHO je nejlepší vytvořit custom @RestController a nesnažit se znásilnit Spring Data REST na něco, k čemu nebyl nikdy určen.

Dokumentace

- V této přednášce jsem se snažil shrnout několik let zkušeností s používáním tohoto projektu. V dokumentaci řadu z těchto věcí nenajdete, ale zase tam pro změnu můžete najít něco dalšího zajímavého, co jsem v této přednášce mohl opomenout:
 - <https://docs.spring.io/spring-data/rest/docs/current/reference/html/>

Děkuji za pozornost