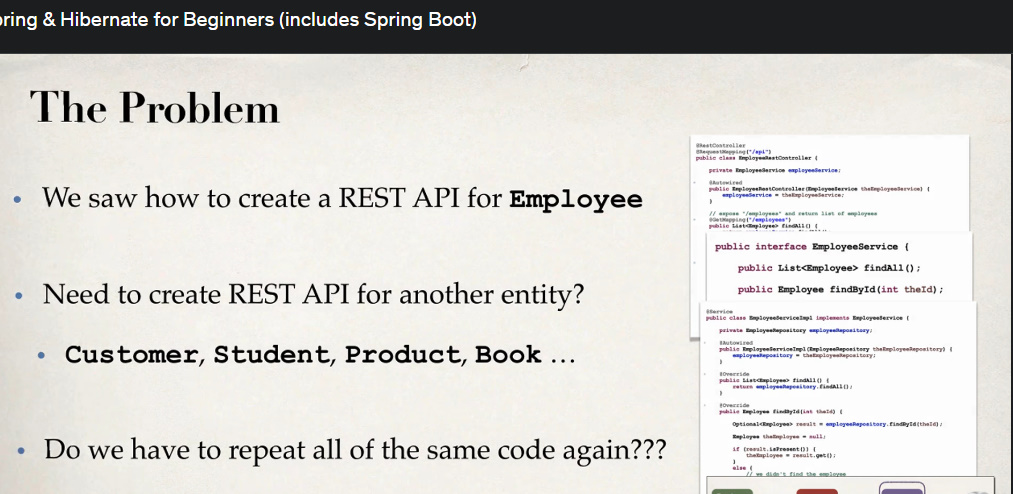
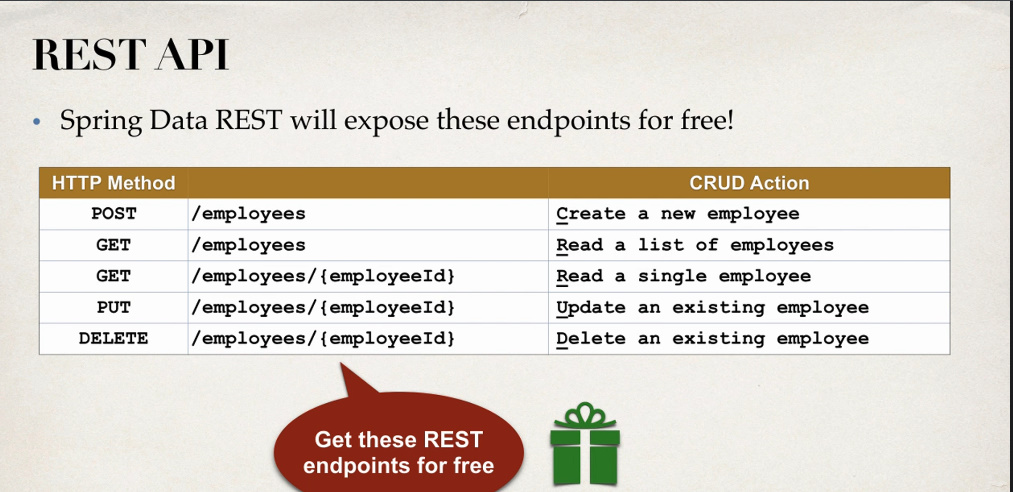
**Spring Data REST**





Deci, daca am vrea sa cream un REST API pentru alte entitati, ar trebui sa tot repetam cam acelasi cod in REST Controller, dar Spring Data REST ne ajut cu asta

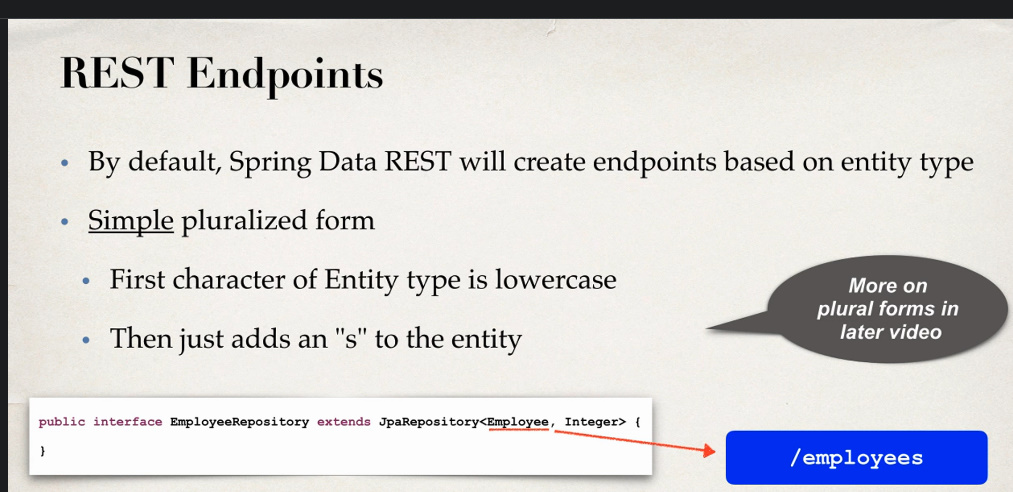
.



**Cum functioneaza**

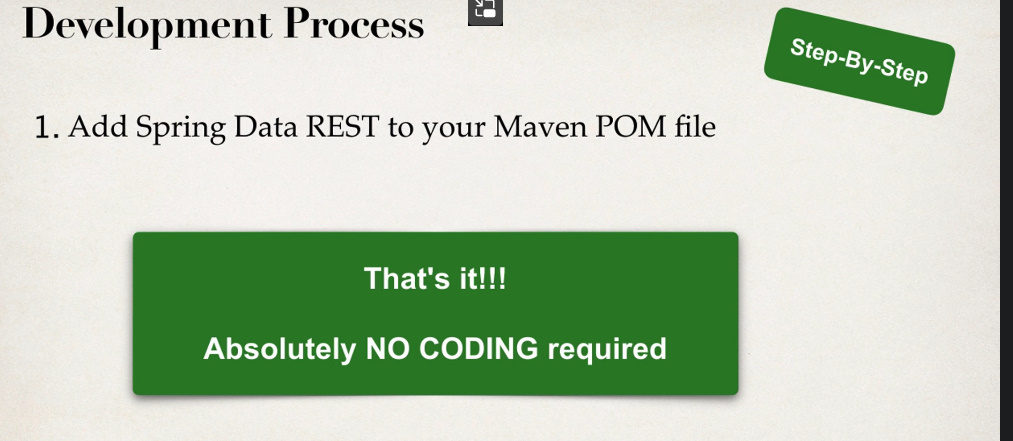


**Endpoints**

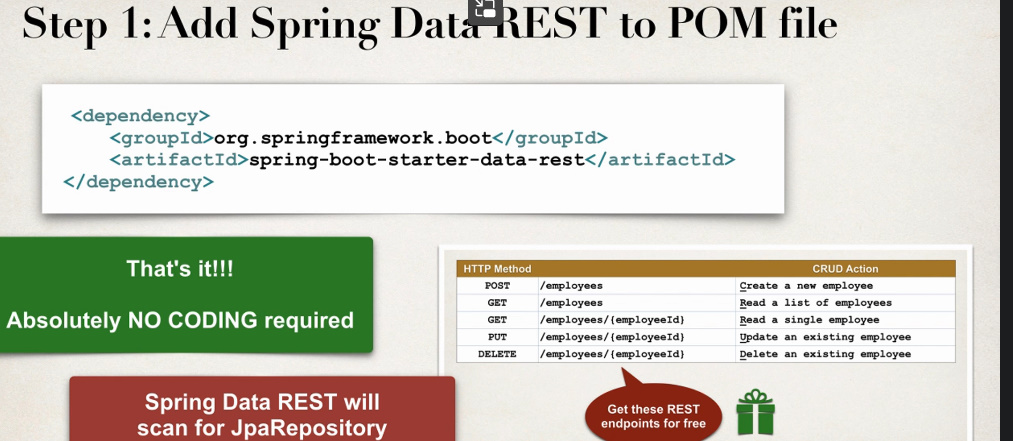


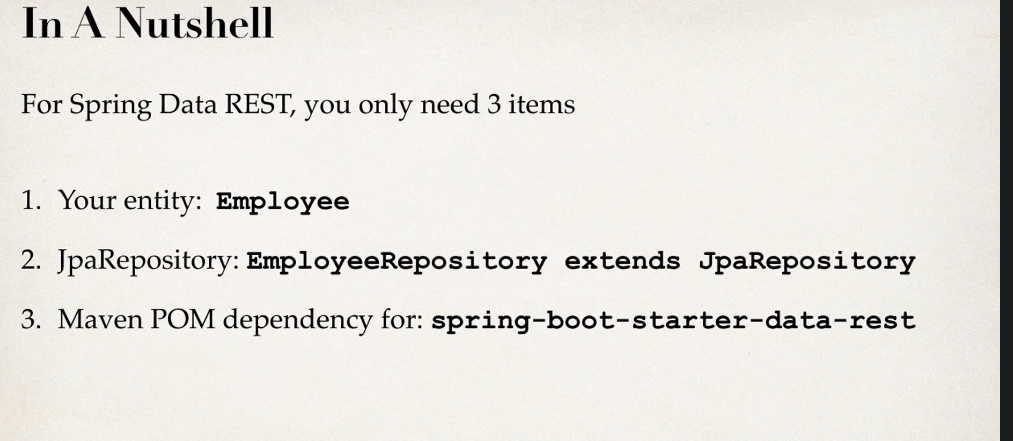
Spring Data REST creaza endpoints, luand numele clasei care am trimis-o ca Generic, dar cu litera mica la inceput si cu s la final.

**Dezvoltare**

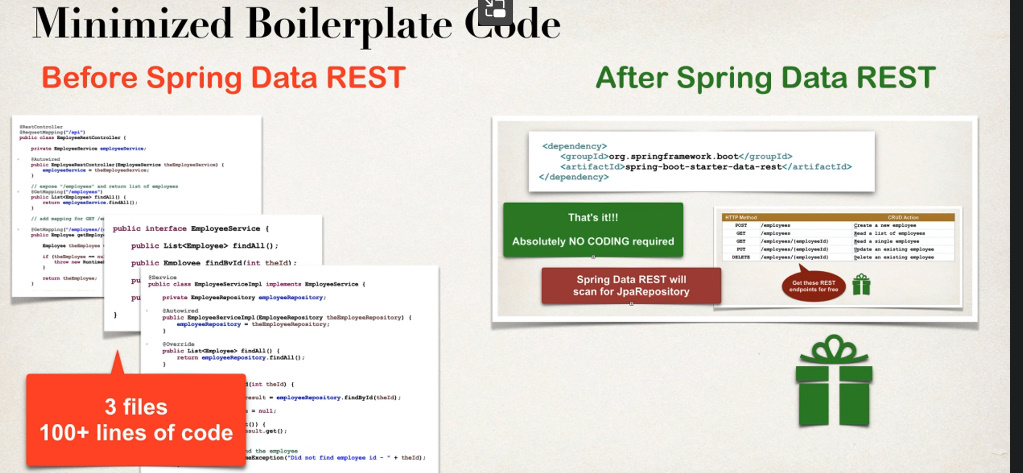
****

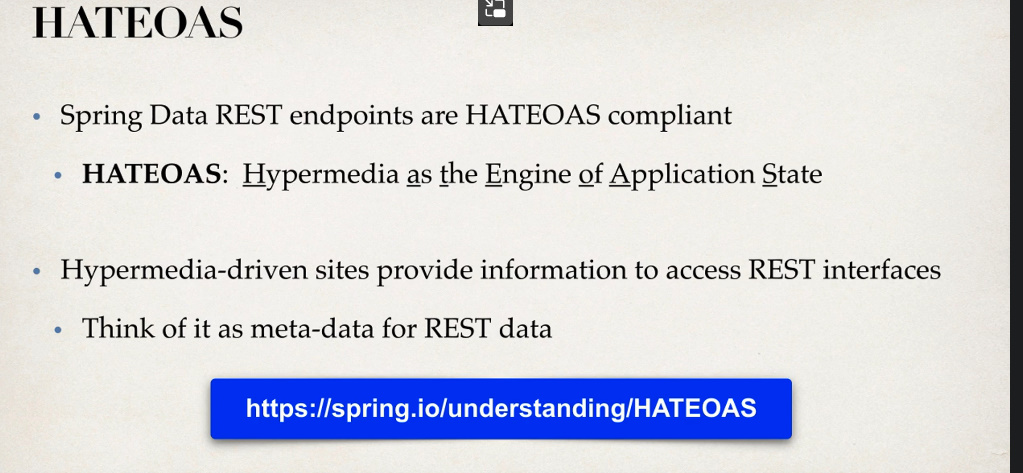
Asta e tot, mai mult nu trebuie sa facem nimic. Pentru fiecare interfata care extinde JpaRepository, Spring o va scana si se va crea un endpoints cu REST support, si vor fi oferite automat metode pentru get, put, post etc.

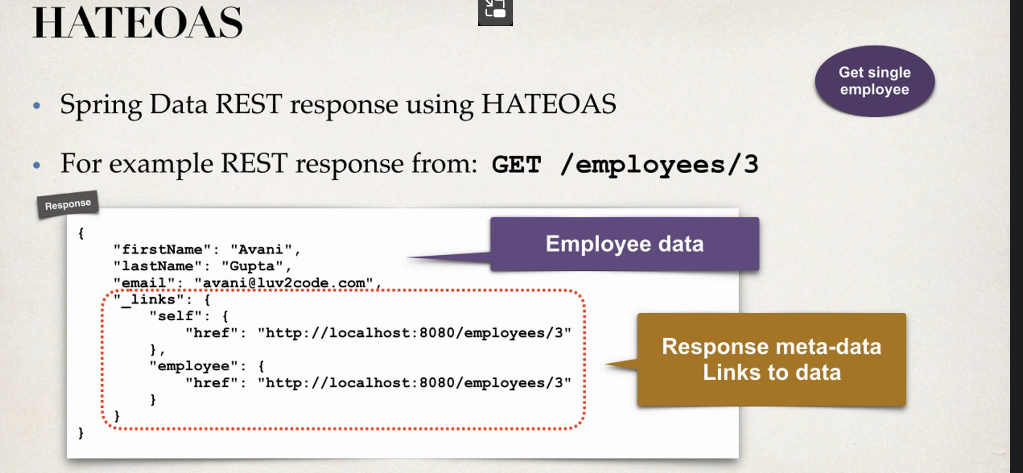


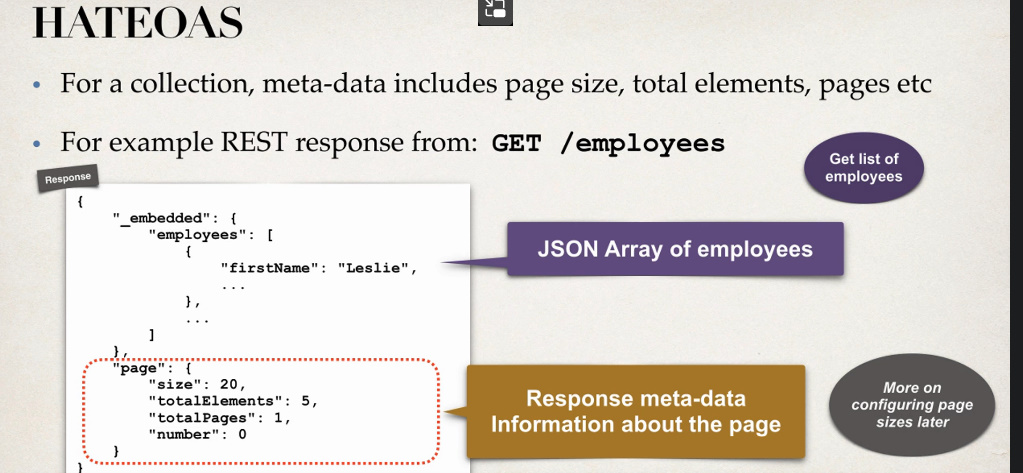


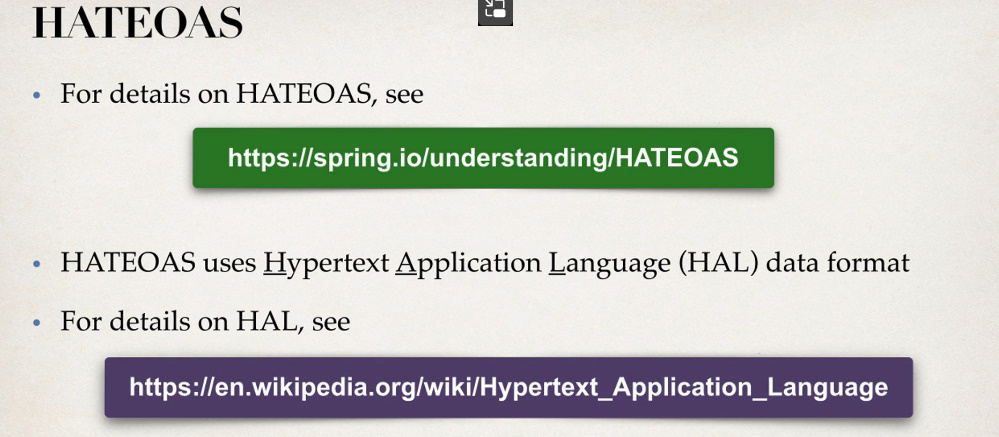
Nu avem nevoie de @Service sau @RestController, ci doar de dependenta, entity i JpaRepository

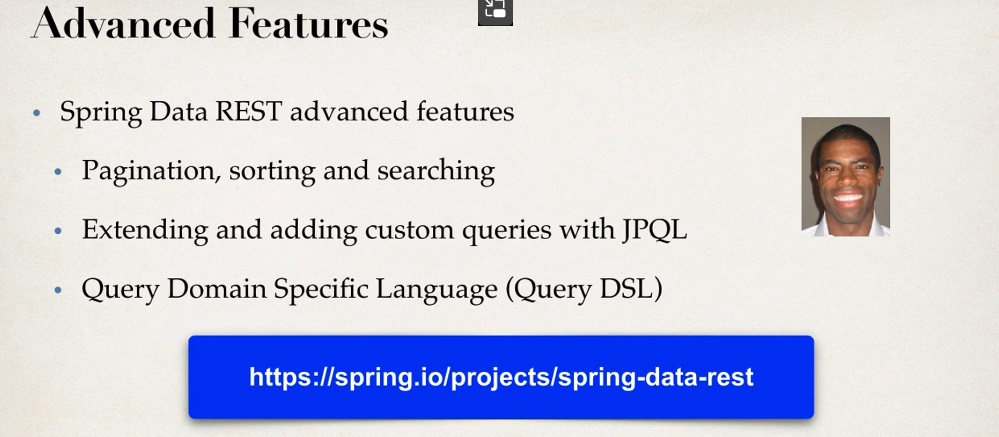


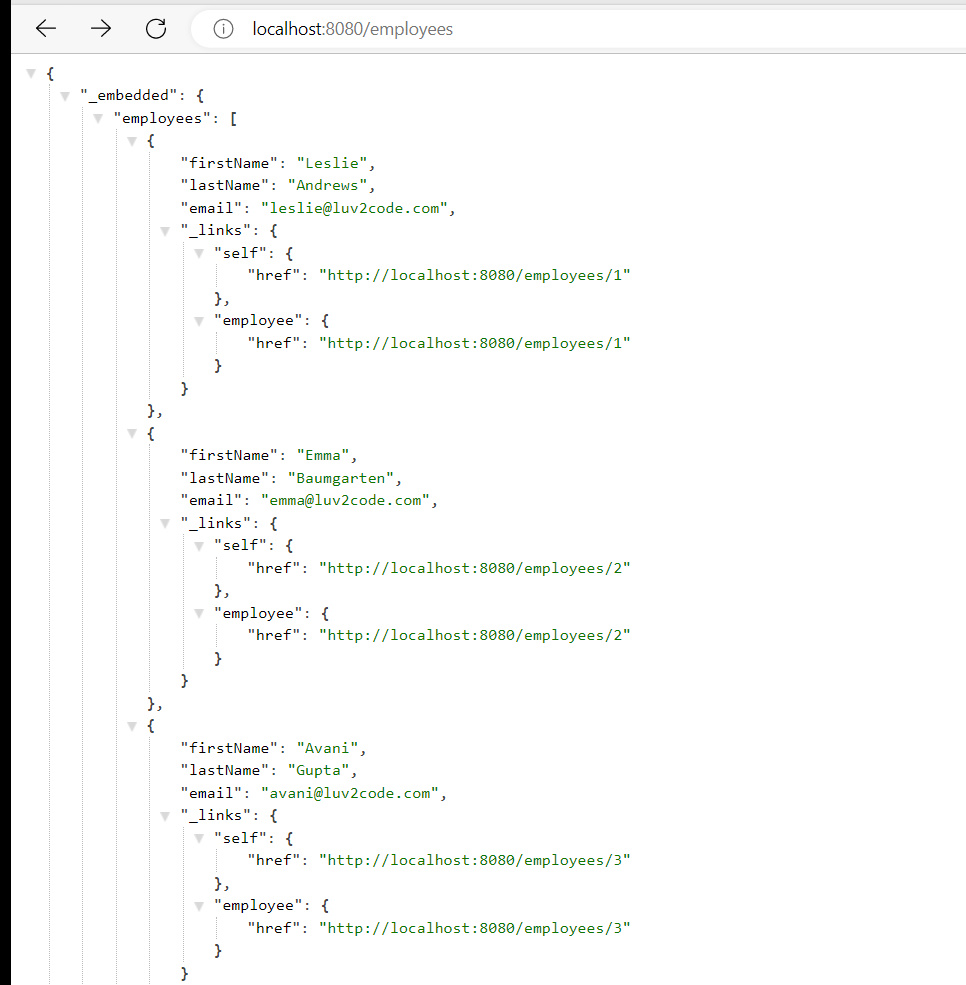








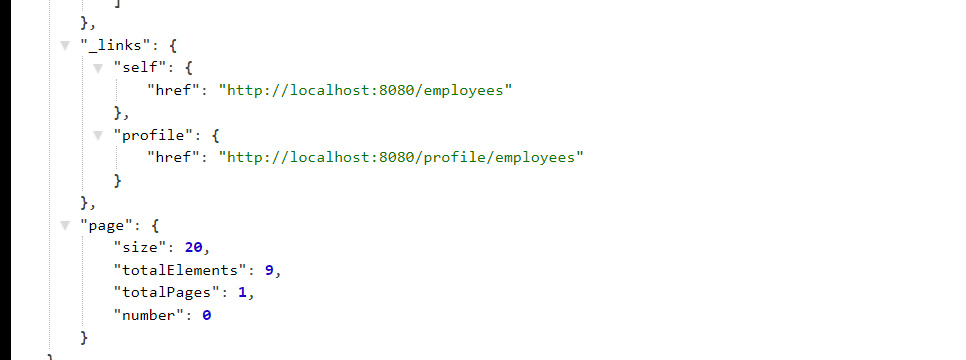


**Exemplu**



Fiecare obiect returnat contine si info despre linkul la care putem informatii particulare pentru el.

**La final mai vedem ceva**

****

**Aici vedem info: pagini, cate obiecte sunt returnate etc.**

**Atentie la PUT**

****

**La put, nu vom oferi si id in JSON, dar il vom pune in URL. Daca il punem in JSON, atunci Spring il va ignora pur si simplu.**

**Delete nu va returna nimic**

**Customize endpoint base path**

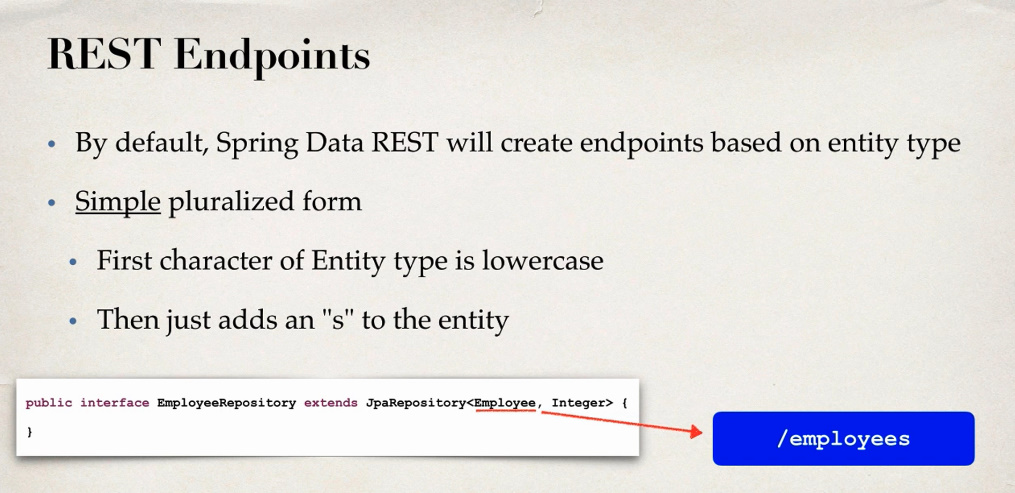
Putem aduauga configuratii in application.properties, de ex:

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

asa, vom folosi /api/employees/

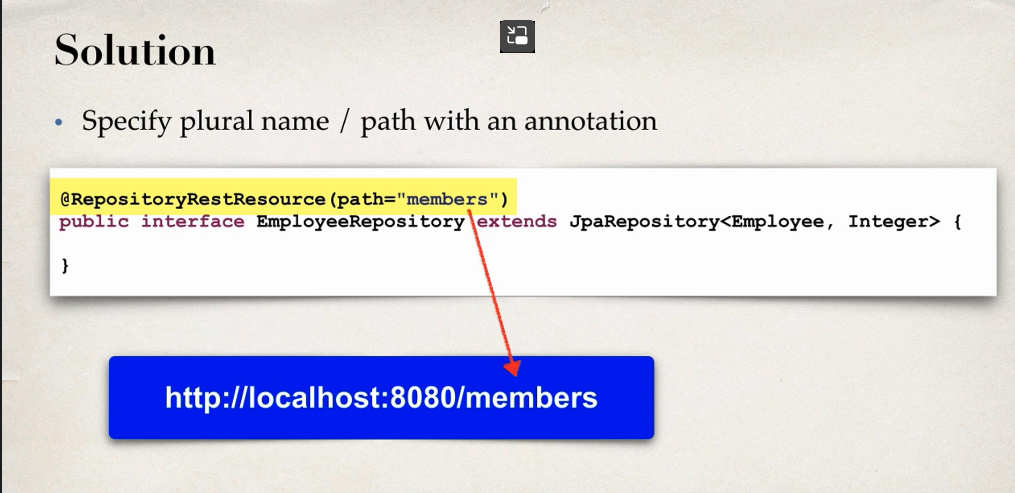
**REST Endpoints**

* Deci, endpoint e creat pe baza la numele la clasa, insa prima litera e mica si la final se pune s. Dar, asta poate fi modificat.

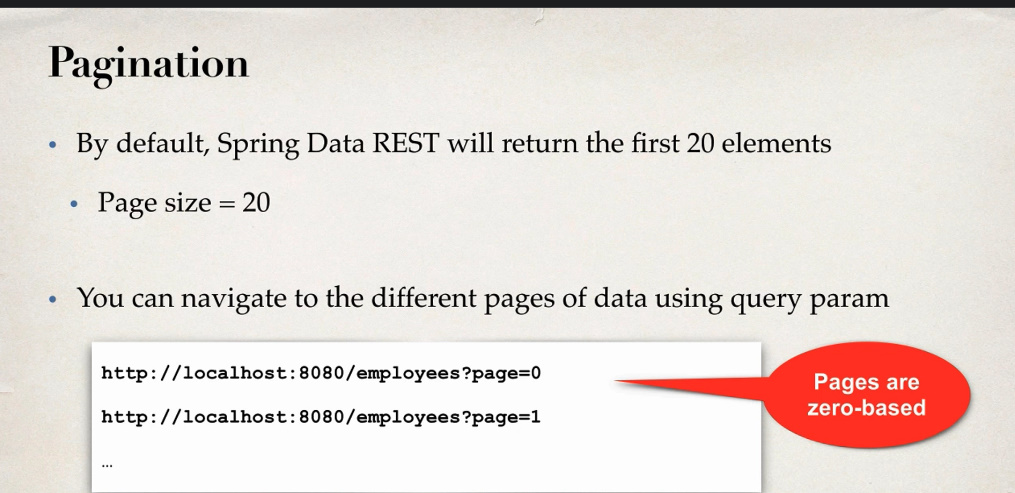


* Totusi, punerea unui s la final nu mereu face ca cuvantul sa fie corect scris, de ex sheep nu e sheeps sau person e people
* Si poate in loc de employees am vrea sa fie customers

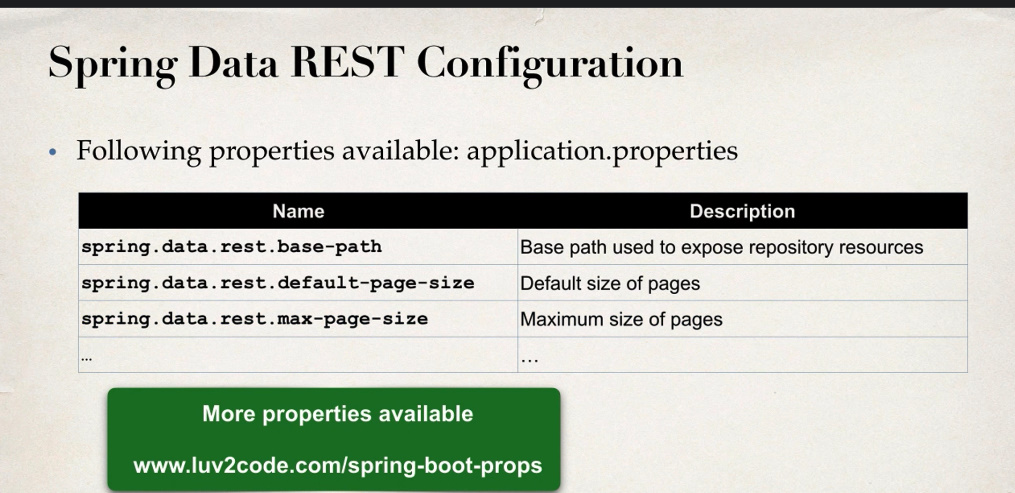
**@RepositoryRestResource(path=”nume”)**

* Anotatia data o punem deasupra la DAO ce extinde JpaRepository, si anume numele pus de noi va fi folosit in Rest api
* 

**Pagination**

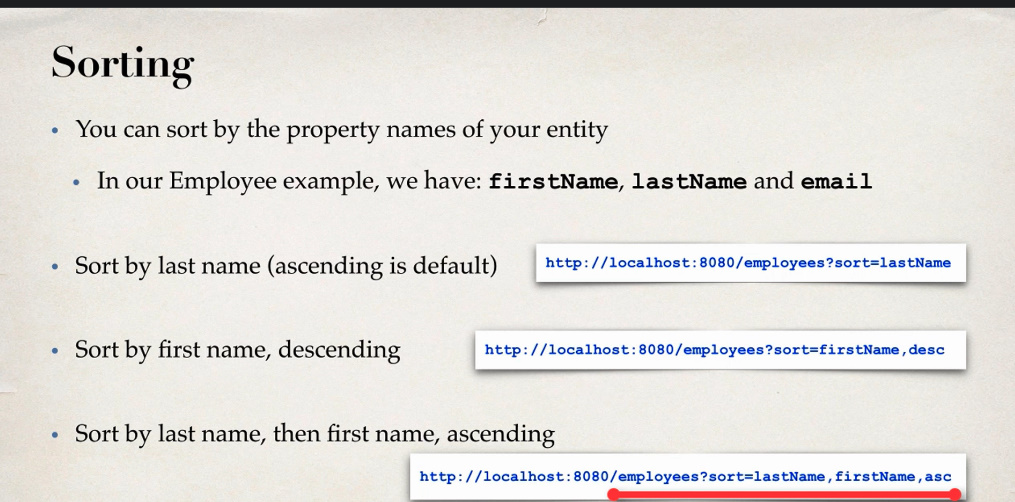


zero-based – incep de la 0



default-page-size – nr de elemente per pagina

**Sorting**



**Problema cu HibernateTransactionManager**

* Pentru a putea folosi RestJPA, avem nevoie sa folosim interfetele de la JPA. SessionFactory si Session nu sunt probleme,asa cum ele implementeaza interfete, dar sunt probleme cu HibernateTransactionManager.
* Rest JPA are nevoie de JpaTransactionManager, nu HibernateTransactionManager
* @SpringBootApplication  
  public class DemoApplication implements WebMvcConfigurer {  
    
   public static void main(String[] args) {  
   SpringApplication.*run*(DemoApplication.class, args);  
    
   }  
   @Bean()  
   public DataSource dataSource(){  
   DriverManagerDataSource dataSource = new DriverManagerDataSource();  
   dataSource.setUsername("testuser");  
   dataSource.setPassword("Frb2eshox!");  
   dataSource.setUrl("jdbc:mysql://localhost:3306/test?useSSL=false&serverTimezone=UTC");  
   dataSource.setDriverClassName(com.mysql.jdbc.Driver.class.getName());  
    
   return dataSource;  
   }  
   @Bean("entityManagerFactory")  
   public LocalSessionFactoryBean sessionFactoryBean(){  
   LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
   sessionFactoryBean.setPackagesToScan("com.example.demo");  
   sessionFactoryBean.setDataSource(dataSource());  
   Properties properties = new Properties();  
   properties.setProperty("hibernate.dialect",org.hibernate.dialect.MySQLDialect.class.getName());  
   sessionFactoryBean.setHibernateProperties(properties);  
    
   return sessionFactoryBean;  
   }  
    
   @Bean()  
   public JpaTransactionManager transactionManager(@Autowired SessionFactory sessionFactory){  
   JpaTransactionManager transactionManager = new JpaTransactionManager();  
   transactionManager.setEntityManagerFactory(sessionFactory);  
    
   return transactionManager;  
   }  
    
    
  }