Orice REST API trebuie sa fie capabil sa:

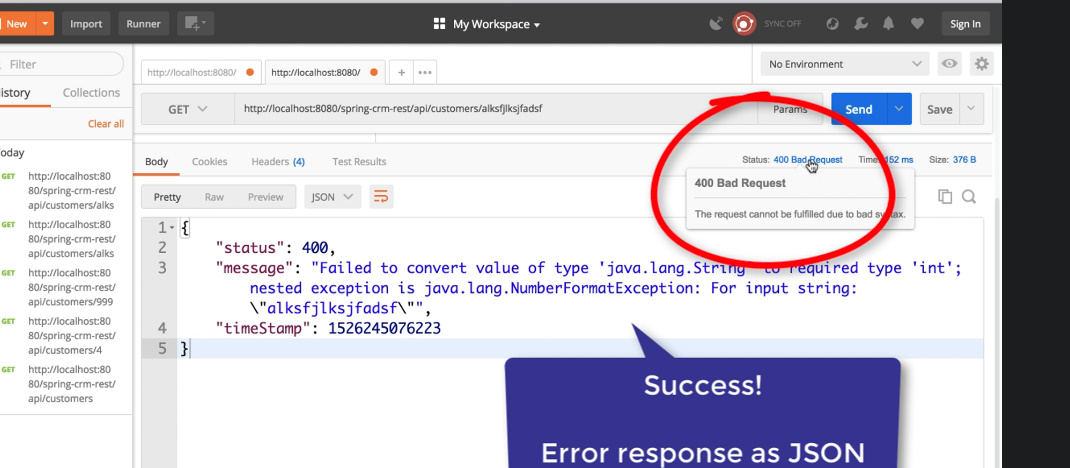
* returneze o lista de elemente
* sa returneze un element dupa id
* sa adauge un element
* sa actualizeze un element
* sa stearga un element

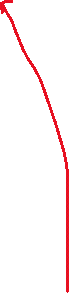
**Dezvoltarea app**

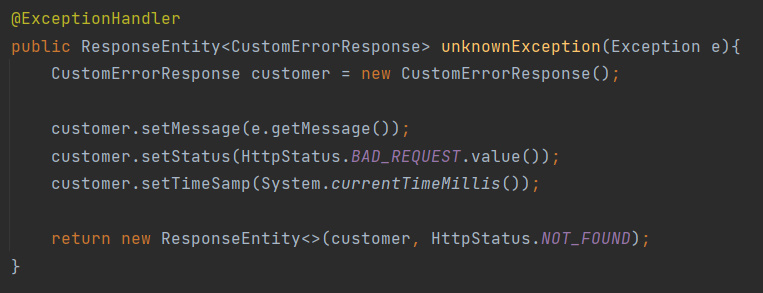
* Pentru a face asta, logica va fi aceeasi ca si in Web Controller, adica vom crea request mappinguri si evident vom crea un obiect de tip Service creat si il vom da @Autowired



* Daca vom returna null, nu va aparea vreo eroare, ci o pagina goala.



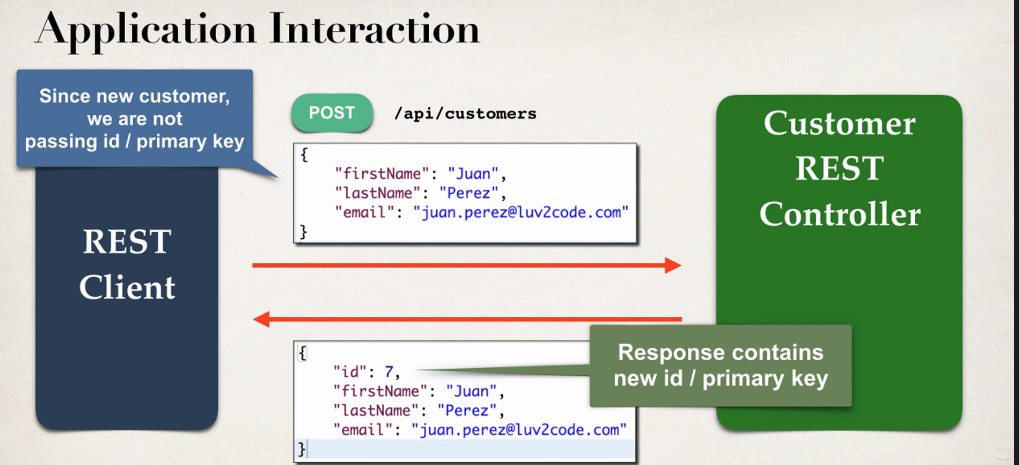






Deci, ce vom seta in ResponseEntity ca al 2 parametru drept erroare, va aparea si la user.

**AddCustomer(POST)**



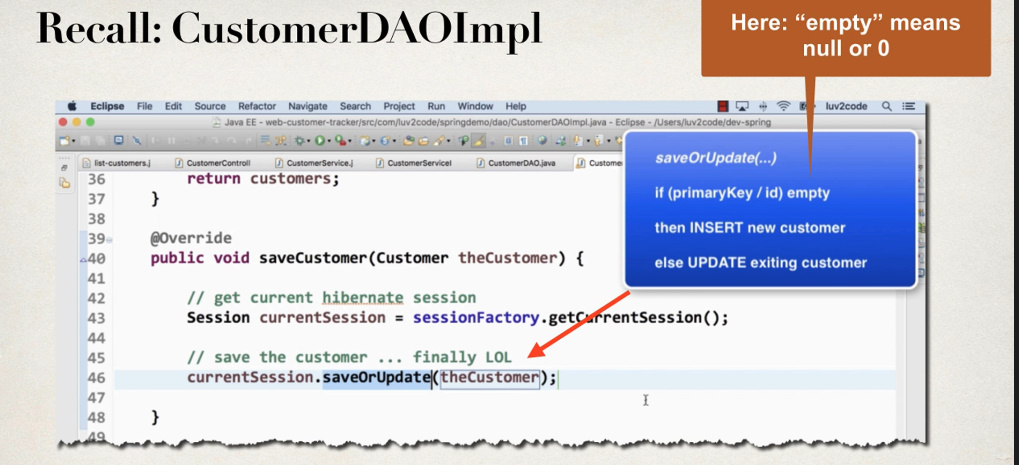
* Deci, clientul va adauga un user prin rest API, insa nu va putea face asta prin browser, ci prin Postman, ca sa poata folosi POST si datele vor fi trimsie ca JSON
* Logic ca clientul nu va oferi si un id pentru user
* Apoi, serverul va returna in JSON userul creat,

**@RequestBody**



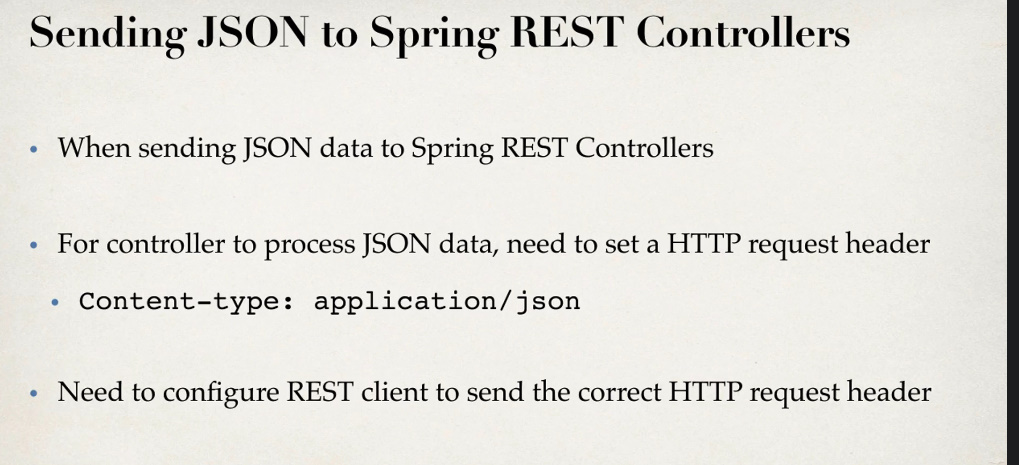
* Deci, nu modificam requestul, doar adaugam o metoda si pentru POST
* Jackson va converti JSON in POJO, si pentru a accesa acel POJO creat de el, folosim @RequestBody





@PostMapping("/customers")  
public Customer addCustomer(@RequestBody Customer customer){  
 customer.setId(0);  
 customerService.saveCustomer(customer);  
  
 return customer;  
}

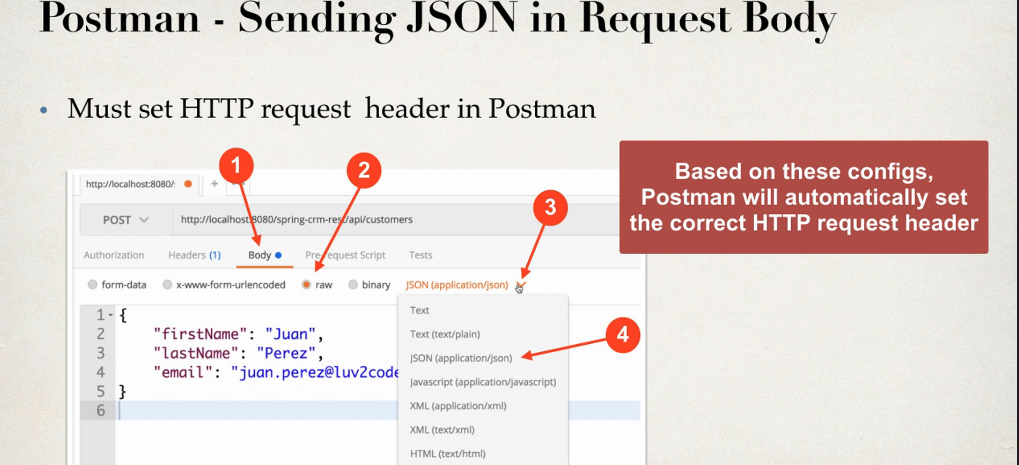
Acest customer object returnat de JACKSON va fi trimis in Service, apoi in DAO si deci Hibernate se va asigura sa-i atribute un ID, asa cum el e null, adica 0, si acest ID il seteaza chiar in obiect si apoi il pune in baza de date si noua ni-l va returna exact asa cum l-a pus il in DB



**De ce setam id ca 0**

Chiar daca metoda saveOrUpdate nu adauga obiectul daca id este null, clientul s-ar putea sa ofere un user cu id specificat de el, si asa sa actualizeze un user deja existent. Deci, pentru a ne asigura ca asa ceva nu se va intampla, setam deoadata id = 0, caci 0 = null, si deci mereu se va adauga un user nou, ca specifica clientul sau nu id

**PostMan**



application/json – datele in format json sunt trimise aplicatiei

Deci atentie sa selectam POST si acolo dam datele noului user si apoi vom primi drept raspuns userul adaugat cu tot cu ID ales de Hibernate pentru el.

**Update(PUT) a Customer**

Aici deja va trebui sa oferim ID pentru user si noile date si sa returnam userul modificat.

@PutMapping("/customers")  
public Customer updateCustomer(@RequestBody Customer customer){  
 customerService.saveCustomer(customer);  
  
 return customer;  
}

In PostMan vom folosi deja PUT

**Delete a Customer**

Folosim /api/customers/id

@DeleteMapping("/customers/{customerId}")  
public String deleteCustomer(@PathVariable int customerId){  
 Customer customer = customerService.getCustomer(customerId);  
  
 if(customer == null)  
 throw new CustomNotFoundException("Customer with id "+customerId+" wasn't found")  
   
 customerService.deleteCustomer(customerId);  
  
 return "Deleted customer with id "+customerId;  
}

Aici putem sa verificam intai daca un user cu asa id exista. Daca da, totul e bine, daca nu, returnam o exceptie si spunem ca asa user nu exista.

In PostMan vom alege delete, si vom folosi asa request:

<http://localhost:8080/api/customers/10>

[Spring & Hibernate for Beginners (includes Spring Boot) | Udemy](https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/14799250#questions)

[Spring & Hibernate for Beginners (includes Spring Boot) | Udemy](https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/10637786#questions)