Validators

Possiamo effettuare la validazione di una serie di dati (che grazie al data binding diventano un oggetto Java o POJO) su 2 livelli:

1) nel form HTML in cui li inseriamo, quindi lato front-end attraverso Javascript, JQuery ed adesso anche HTML5, analizzando la correttezza dei dati inseriti e impedendo l'invio dei dati del form se questi non sono corretti;

2) nell'applicazione web, lato server, all'atto del data binding (se non volessimo utilizzare il data binding basta analizzare i singoli valori ottenuti e con essi, se validi, creare un nuovo oggetto Java POJO utilizzando i valori validi come attributi dello stesso e passandoli attraverso il costruttore dell'oggetto java oppure attraverso i setter method). Se usiamo il data binding, Java Spring riconoscerà che i dati inseriti rappresentano un POJO e lo istanzierà automaticamente. Ma prima dobbiamo analizzare la validità dei singoli parametri anche lato server. Questo avviene tramite due strategie:

a)JSR-303 ovvero le annotation sui singoli attributi del POJO Java (es @NotNull @NotBlank @NotEmpty) che definiscono quando un POJO è @Valid e quando non lo è.

b) SpringValidator che è un validatore custom che analizza i singoli attributi.

Vediamo l'esempio:

POJO:

```
package com.quicktutorials.learnmicroservices.AccountMicroservice.entities;


//1) JPA
//2) JSR-303
//3) LOMBOK



import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.hibernate.validator.constraints.NotBlank;
import org.hibernate.validator.constraints.NotEmpty;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;

@AllArgsConstructor @NoArgsConstructor   //Lombok annotations
@Entity                                  //JPA defines an Entity
@Table(name = "users")                   //JPA (if table name in the DB differs from Class Name)
public class User {
```

```
    //String ID, String USERNAME, String PASSWORD, String PERMISSION

    @Id                                 //JPA id of the table
    @Column(name="ID")                  //JPA (if column name is different from variable name)
    @NotEmpty @NotBlank @NotNull        //JSR-303 Validation
    @Getter @Setter                     //Lombok annotations
    private String id;

    @Column(name="USERNAME")            //JPA (if column name is different from variable name)
    @NotEmpty @NotBlank @NotNull        //JSR-303 Validation
    @Getter @Setter                     //Lombok annotations
    private String username;

    @Column(name="PASSWORD")            //JPA (if column name is different from variable name)
    @NotEmpty @NotBlank @NotNull        //JSR-303 Validation
    @Getter @Setter                     //Lombok annotations
    private String password;

    @Column(name="PERMISSION")          //JPA (if column name is different from variable name)
    @NotEmpty @NotBlank @NotNull        //JSR-303 Validation
    @Getter @Setter                     //Lombok annotations
    private String permission;



}
```

CONTROLLER

```
package com.quicktutorials.learnmicroservices.AccountMicroservice.controllers;


import com.quicktutorials.learnmicroservices.AccountMicroservice.entities.User;
import org.springframework.validation.BindingResult;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import org.springframework.web.bind.annotation.RequestMapping;


import javax.validation.Valid;

@org.springframework.web.bind.annotation.RestController
public class RestController {

    @RequestMapping("/hello")
    public String sayHello(){
        return "Hello everyone!";
    }

    /*  testare con PostMan con modalità x-www-form-urlencoded */



    //NO VALIDATION

    //if pwd is null it will still return a user
    @RequestMapping("/newuser1")
    public String addUser(User user){
```

```
        return "User added correctly:" + user.getId() + ", "+ user.getUsername();
    }



    //JAVA JSR-303 VALIDATION: Java gives the error message

    //if pwd is null it will return a JAVA JSR-303 error message thanks to @Valid

    @RequestMapping("/newuser2")
    public String addUserValid(@Valid User user){
        return "User added correctly:" + user.getId() + ", "+ user.getUsername();
    }



    //JAVA JSR-303 VALIDATION: Spring BindingResult gives us the JAVA JSR-303 error message

    //if pwd is null it will return a JAVA JSR-
303 error message thanks to Spring object BindingResult

    @RequestMapping("/newuser3")
    public String addUserValidPlusBinding(@Valid User user, BindingResult result){
        if(result.hasErrors()){
            return result.toString();
        }
        return "User added correctly:" + user.getId() + ", "+ user.getUsername();
    }



    //SpringValidator VALIDATION: Spring BindingResult gives us the SpringValidator error message


//if pwd is null it will return a SPRING VALIDATOR error message thanks to Spring object BindingResult

    @RequestMapping("/newuser4")
    public String addUserValidPlusBinding2(User user, BindingResult result){
        /* Spring validation */
        UserValidator userValidator = new UserValidator();
        userValidator.validate(user, result);

        if(result.hasErrors()){
            return result.toString();
        }
        return "User added correctly:" + user.getId() + ", "+ user.getUsername();
    }



    /*--------------------------INNER CLASS----------------------*/
    //Spring Validator Example
    private class UserValidator implements Validator {

        @Override
        public boolean supports(Class<?> clazz) {
            return User.class.equals(clazz);
        }

        @Override
        public void validate(Object obj, Errors errors) {
            User user = (User) obj;
            if (user.getPassword().length() < 8) {
                errors.rejectValue("password", "the password must be at least 8 chars long!");
            }
        }
```

```
    }
    /*----------------------------------------------------------------*/


}
```