Contenido

[**Sección 81: Spring Framework 5: Formularios Thymeleaf y Data Binding** 3](#_Toc147426642)

[631. Creando nuevo proyecto form 3](#_Toc147426643)

[632. Actualización: para Spring Boot 2.3.0 o superior agregar dependencia validation 3 min 3](#_Toc147426644)

[633. Añadiendo la vista del formulario 3](#_Toc147426645)

[634. Método handler para recibir y procesar los datos enviados 3](#_Toc147426646)

[635. Creando la clase model del formulario 6](#_Toc147426647)

[636. Mapeando los campos del formulario al objeto model 8](#_Toc147426648)

[637. Validación del formulario usando anotaciones 8](#_Toc147426649)

[**REVISAR BIEN LUEGO EL PORQUE NO FUNCIONA COMO ESPERO** 9](#_Toc147426650)

[PROYECTO FORM 2 9](#_Toc147426651)

[638. Añadiendo los mensajes de errores en la vista del formulario 13](#_Toc147426652)

[639. El atributo object de thymeleaf en el elements form 14](#_Toc147426653)

[640. Mostrar valores de atributos del objeto model en el formulario 14](#_Toc147426654)

[641. La anotación @SessionAttributes para manter los datos durante el ciclo del form 14](#_Toc147426655)

[642. La anotación @Size para rangos y @Email para validar correo electrónico 14](#_Toc147426656)

[643. Mensajes de errores personalizados usando properties 14](#_Toc147426657)

[644. Archivos properties y acentos con UTF-8 14](#_Toc147426658)

[645. Validación personalizada usando anotación @Pattern para expresiones regulares 14](#_Toc147426659)

[646. Validación personalizada usando una clase Validation 14](#_Toc147426660)

[647. Registrando una clase validador con la anotación @InitBinder 14](#_Toc147426661)

[648. Validación personalizada usando anotaciones 14](#_Toc147426662)

[649. Validación personalizada usando 14](#_Toc147426663)

[anotaciones ejemplo requerido 14](#_Toc147426664)

[650. Validación de números enteros con @Max @Min y NotNull 14](#_Toc147426665)

[651. Validación de fechas con @NotNull 14](#_Toc147426666)

[dando formato con @DateTimeFormat 14](#_Toc147426667)

[652. Validación de fechas con input type date de HTML5 y @Past y @Future 14](#_Toc147426668)

[653. Formateando fechas con @lnitBinder y registrando un CustomDateEditor 14](#_Toc147426669)

[654. Formateando fechas con @lnitBinder y registrando un CustomDateEditor Parte 2 15](#_Toc147426670)

[655. Implementando propio filtro custom property editor para convertir a mayúscula 15](#_Toc147426671)

[656. Lista select desplegable 15](#_Toc147426672)

[657. Formateando fechas en vistas thymeleaf 15](#_Toc147426673)

[658. Llenando lista select con Map 15](#_Toc147426674)

[659. Llenando lista select con objetos del la clase Pais 15](#_Toc147426675)

[660. Añadiendo componente service de Pais 15](#_Toc147426676)

[661. Añadiendo componente pais property editor 15](#_Toc147426677)

[662. Implementando checkboxes 15](#_Toc147426678)

[663. Llenando checkboxes con Map 15](#_Toc147426679)

[664. Llenando checkboxes con objetos del tipo Role 15](#_Toc147426680)

[665. Añadiendo componente role property editor 15](#_Toc147426681)

[666. Checkbox booleano true o false 15](#_Toc147426682)

[667. Radio button 15](#_Toc147426683)

[668. Input type hidden 15](#_Toc147426684)

[669. Poblando pais y roles 15](#_Toc147426685)

[670. Redirect después del POST procesar 15](#_Toc147426686)

[671. Añadiendo estilos CSS al formulario con Bootstrap 15](#_Toc147426687)

[672. Añadiendo estilos CSS Bootstrap a la plantilla resultado 15](#_Toc147426688)

[673. Cargando archivo css Bootstrap localmente 15](#_Toc147426689)

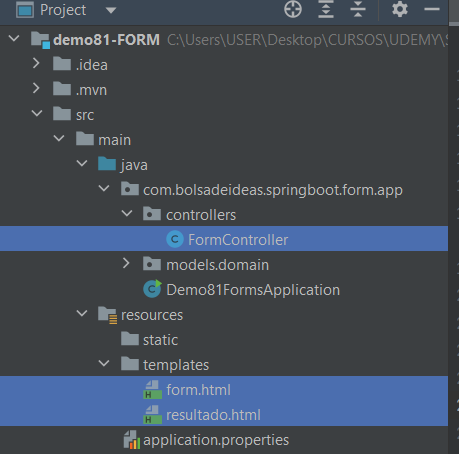
# **Sección 81: Spring Framework 5: Formularios Thymeleaf y Data Binding**

# 631. Creando nuevo proyecto form

# 632. Actualización: para Spring Boot 2.3.0 o superior agregar dependencia validation 3 min

# 633. Añadiendo la vista del formulario

# 634. Método handler para recibir y procesar los datos enviados



package com.bolsadeideas.springboot.form.app.controllers;  
  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
@Controller  
public class FormController {  
  
 @GetMapping("/form")  
 public String form(Model model){  
 model.addAttribute("titulo","Creando usuario");  
 return "form";  
 }  
  
 @PostMapping("/form")  
 public String procesar(Model model,  
 @RequestParam String username,  
 @RequestParam String password,  
 @RequestParam String email){  
  
  
 model.addAttribute("titulo","Procesando usuario");  
  
 model.addAttribute("username",username);  
 model.addAttribute("password",password);  
 model.addAttribute("email",email);  
  
 return "resultado";  
 }  
  
}

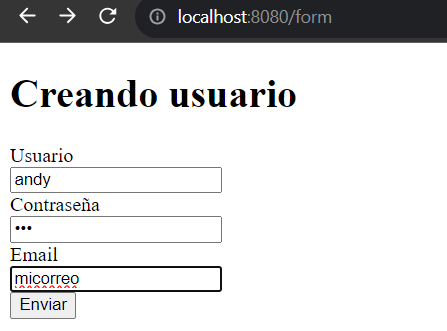
form.html

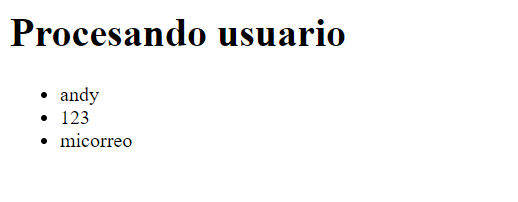
<!doctype html>  
<html lang="en" xmlns:th="http://thymeleaf.org">  
  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport"  
 content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <title>Form</title>  
</head>  
  
<body>  
  
<h1 th:text="${titulo}"></h1>  
  
 <form th:action="@{/form}">  
  
 <div>  
 <label for="username">Usuario </label>  
 <div>  
 <input type="text" name="username" id="username">  
 </div>  
 </div>  
  
 <div>  
 <label for="password">Contraseña </label>  
 <div>  
 <input type="password" name="password" id="password">  
 </div>  
 </div>  
  
 <div>  
 <label for="email">Email </label>  
 <div>  
 <input type="text" name="email" id="email">  
 </div>  
 </div>  
  
 <div>  
 <div>  
 <input type="submit" value="Enviar">  
 </div>  
 </div>  
  
 </form>  
  
</body>  
  
</html>

Resulado.html

<!doctype html>  
<html lang="en" xmlns:th="http://thymeleaf.org">  
  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport"  
 content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <title>Form</title>  
</head>  
  
<body>  
  
<h1 th:text="${titulo}"></h1>  
  
<ul>  
  
 <li th:text="${username}"></li>  
 <li th:text="${password}"></li>  
 <li th:text="${email}"></li>  
</ul>

</body>  
  
</html>





# 635. Creando la clase model del formulario

A diferencia del anterior creamos la clase models/domain/Usuario

package com.bolsadeideas.springboot.form.app.models.domain;  
  
import lombok.Data;  
  
@Data  
public class Usuario {  
  
 //deben ser igual al name del form  
  
 private String username;  
 private String password;  
 private String email;  
  
}

Modificamos el metodo con el Post del controller (PARA ALMACENAR EL ANTERIOR COMENTAMOS Y LO VOLVIMOS A CREAR)

package com.bolsadeideas.springboot.form.app.models.domain;  
  
import lombok.Data;  
  
@Data  
public class Usuario {  
  
 //deben ser igual al name del form  
  
 private String username;  
 private String password;  
 private String email;  
  
}

FormController

instanciamos la clase Usuario y la add al model

@PostMapping("/form")  
public String procesar(Model model,  
 @RequestParam String username,  
 @RequestParam String password,  
 @RequestParam String email){  
  
 model.addAttribute("titulo","Procesando usuario");  
  
 Usuario usuario = new Usuario();  
 usuario.setUsername(username);  
 usuario.setPassword(password);  
 usuario.setEmail(email);  
  
 model.addAttribute("usuario",usuario);  
  
 return "resultado";  
}

form.html

no se modifico nada

resultado.html

solo agregamso el usuario.

<ul>  
 <!-- usuario es del cotroller -->  
 <li th:text="${usuario.username}"></li>  
 <li th:text="${usuario.password}"></li>  
 <li th:text="${usuario.email}"></li>  
</ul>

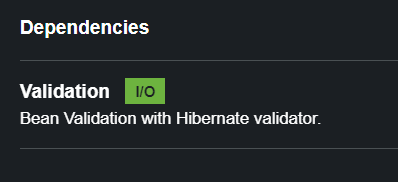
# 636. Mapeando los campos del formulario al objeto model

N el controller movemos la instancia a los parámetros del metodo

//MAPEAMOS LOS CAMPOS DEL FORMULARIO AL OBJETO MODEL  
@PostMapping("/form")  
public String procesar( Usuario usuario , Model model){  
  
 model.addAttribute("titulo","Procesando usuario");  
 model.addAttribute("usuario",usuario);  
  
 return "resultado";  
}

El resto esta igual

# 637. Validación del formulario usando anotaciones



<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>2.0.1.Final</version>

</dependency>

NO NECESITO ESA DEPENDENCIA, necesito usar jakarta desde el model primero exportato del

En el controller necesito

import jakarta.validation.Valid;

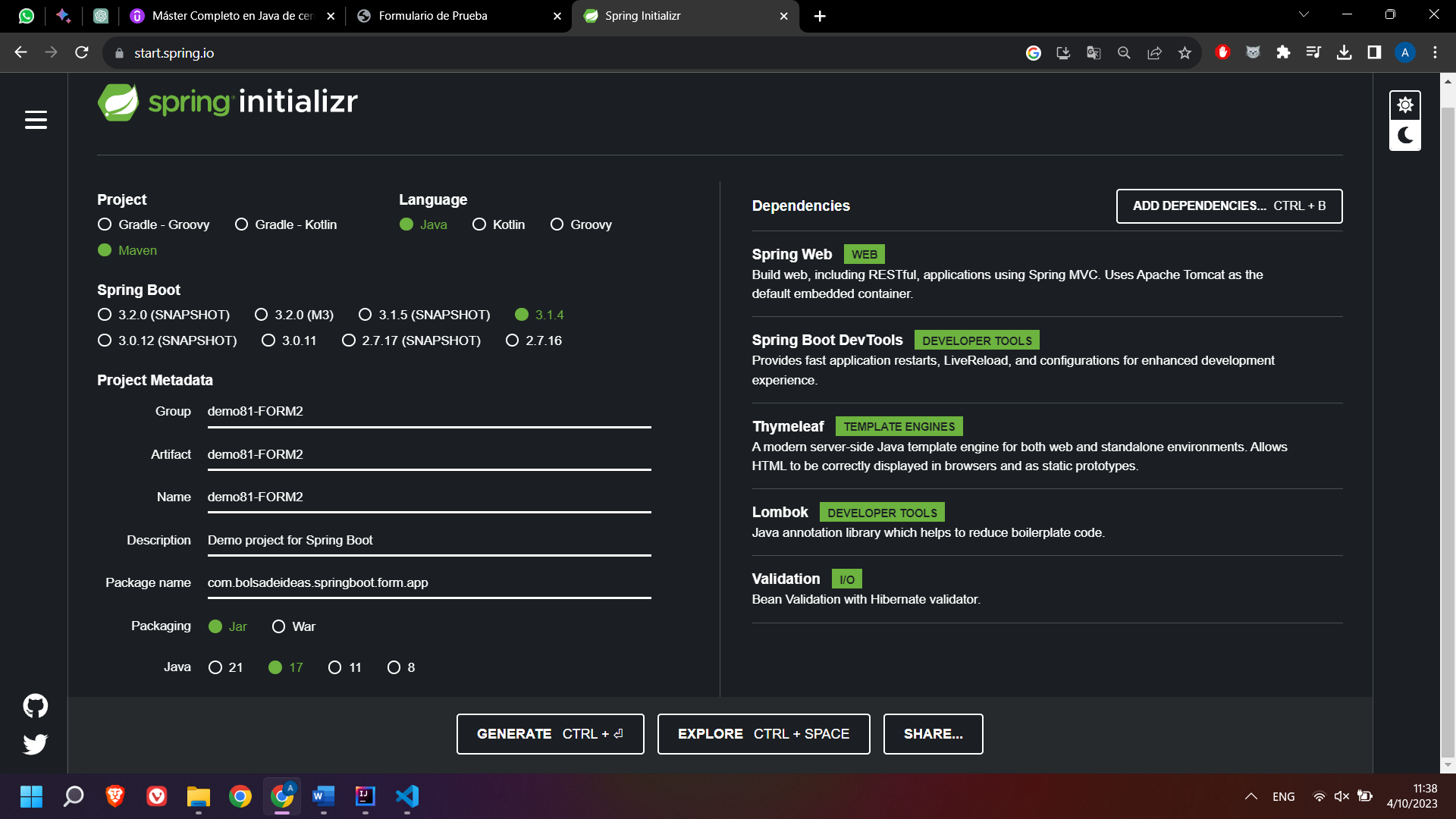
y en el Model

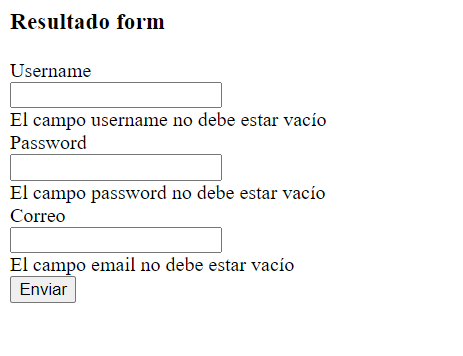
import jakarta.validation.constraints.NotEmpty;

# **REVISAR BIEN LUEGO EL PORQUE NO FUNCIONA COMO ESPERO**

**ESRAMOS TRABAJANDO CON EL FORM2 PERO QUITE EL POMP EL VALIDATE**

# PROYECTO FORM 2



****

package com.bolsadeideas.springboot.form.app.controllers;  
  
import com.bolsadeideas.springboot.form.app.models.domain.Usuario;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import jakarta.validation.Valid;//@Valid  
  
import java.util.HashMap;  
import java.util.Map;  
  
@Controller  
public class FormController {  
  
 @GetMapping("/form")  
 public String form(Model model) {  
 Usuario usuario = new Usuario();  
 model.addAttribute("titulo", "Formulario usuarios");  
 model.addAttribute("usuario", usuario);  
 return "form";  
 }  
  
 //Validación del formulario usando anotaciones  
 @PostMapping("/form")  
 public String procesar(@Valid Usuario usuario, BindingResult result, Model model) {  
  
 model.addAttribute("titulo", "Resultado form");  
  
 if(result.hasErrors()) {  
 Map<String, String> errores = new HashMap<>();  
 result.getFieldErrors().forEach(err ->{  
 errores.put(err.getField(), "El campo ".concat(err.getField()).concat(" ").concat(err.getDefaultMessage()));  
 });  
 model.addAttribute("error", errores);  
 return "form";  
 }  
  
 model.addAttribute("usuario", usuario);  
  
 return "resultado";  
 }  
  
}

package com.bolsadeideas.springboot.form.app.models.domain;  
  
import jakarta.validation.constraints.NotEmpty;  
import lombok.Data;  
  
@Data  
public class Usuario {  
  
 @NotEmpty  
 private String username;  
  
 @NotEmpty  
 private String password;  
  
 @NotEmpty  
 private String email;  
  
}

form.html

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
<meta charset="UTF-8">  
<title th:text="${titulo}"></title>  
</head>  
<body>  
 <h3 th:text="${titulo}"></h3>  
  
 <form th:action="@{/form}" method="post">  
  
 <div>  
 <label for="username">Username</label>  
 <div>  
 <input type="text" name="username" id="username" th:value="${usuario.username}">  
 </div>  
 <div th:if="${error != null && error.containsKey('username')}" th:text="${error.username}"></div>  
 </div>  
 <div>  
 <label for="password">Password</label>  
 <div>  
 <input type="password" name="password" id="password">  
 </div>  
 <div th:if="${error != null && error.containsKey('password')}" th:text="${error.password}"></div>  
 </div>  
 <div>  
 <label for="email">Correo</label>  
 <div>  
 <input type="text" name="email" id="email" th:value="${usuario.email}">  
 </div>  
 <div th:if="${error != null && error.containsKey('email')}" th:text="${error.email}"></div>  
 </div>  
   
 <div>  
 <div>  
 <input type="submit" value="Enviar">  
 </div>  
 </div>   
 </form>  
</body>  
</html>

Resultado.html

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
<meta charset="UTF-8">  
<title th:text="${titulo}"></title>  
</head>  
<body>  
 <h3 th:text="${titulo}"></h3>  
  
 <ul>  
 <li th:text="${usuario.username}"></li>  
 <li th:text="${usuario.password}"></li>  
 <li th:text="${usuario.email}"></li>  
 </ul>  
</body>  
</html>

# 638. Añadiendo los mensajes de errores en la vista del formulario

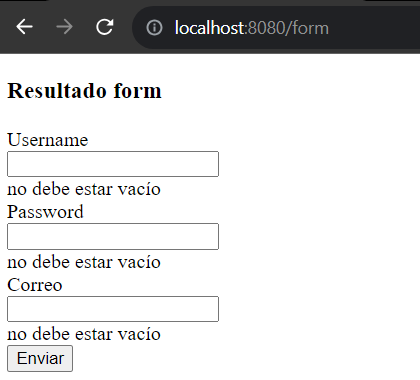
Es el mismo 637 pero la parte del front

# 639. El atributo object de thymeleaf en el elements form

Con el \* se mapea campos al formulario

El $ es para una expresión código java ya sea validar th if etc

Con el @ son rutas URL



Ahora muestra mensajes propiosde Spring

//El atributo object de thymeleaf en el elemento form  
@PostMapping("/form")  
public String procesar(@Valid Usuario usuario, BindingResult result, Model model) {  
  
 model.addAttribute("titulo", "Resultado form");  
  
 if (result.hasErrors()) {  
 return "form";  
 }  
  
 model.addAttribute("usuario", usuario);  
  
 return "resultado";  
}

Form.html

<form th:action="@{/form}" method="post" th:object="${usuario}">  
  
 <div>  
 <label for="username">Username</label>  
 <div>  
 <input type="text" id="username" th:field="\*{username}">  
 </div>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{username}"></div>  
 </div>  
  
 <div>  
 <label for="password">Password</label>  
 <div>  
 <input type="text" id="password" th:field="\*{password}">  
 </div>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{password}"></div>  
 </div>  
  
 <div>  
 <label for="email">Correo</label>  
 <div>  
 <input type="text" id="email" th:field="\*{email}">  
 </div>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{email}"></div>  
 </div>  
   
 <div>  
 <div>  
 <input type="submit" value="Enviar">  
 </div>  
 </div>   
</form>

# 640. Mostrar valores de atributos del objeto model en el formulario

en el controller campos quemados en el form usando los sets

# 641. La anotación @SessionAttributes para manter los datos durante el ciclo del form

Conjunto al 640

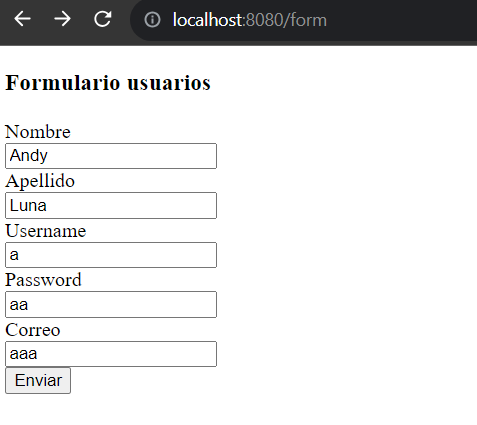
Con el atributo identificador del modelo validaremos que un campo al actualizar se podría cambiar a null al no ser consistente, usamos sesiones para que no se pierda .

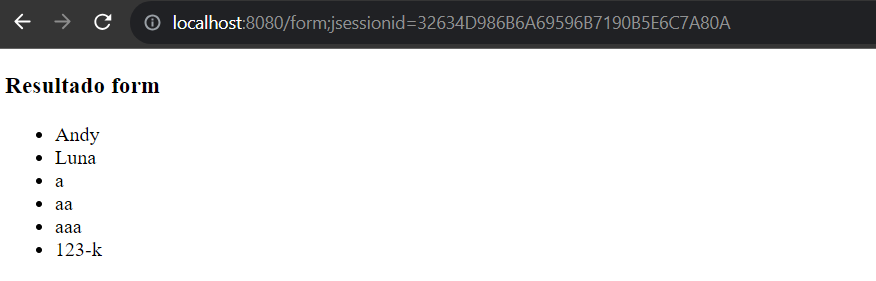
package com.bolsadeideas.springboot.form.app.models.domain;  
  
import jakarta.validation.constraints.NotEmpty;  
import lombok.Data;  
  
@Data  
public class Usuario {  
  
  
 private String identificador;  
  
 @NotEmpty  
 private String nombre;  
  
 @NotEmpty  
 private String apellido;  
  
 @NotEmpty  
 private String username;  
  
 @NotEmpty  
 private String password;  
  
 @NotEmpty  
 private String email;  
  
}

package com.bolsadeideas.springboot.form.app.controllers;  
  
import com.bolsadeideas.springboot.form.app.models.domain.Usuario;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import jakarta.validation.Valid;//@Valid  
import org.springframework.web.bind.annotation.SessionAttributes;  
import org.springframework.web.bind.support.SessionStatus;  
  
  
@Controller  
@SessionAttributes("usuario")// se guarda en una sesion http  
public class FormController {  
  
 @GetMapping("/form")  
 public String form(Model model) {  
 Usuario usuario = new Usuario();  
  
 //mostrando valores de atributos de objeto model en el formulario (only nombre y apellido)  
 usuario.setNombre("Andy");  
 usuario.setApellido("Luna");  
 usuario.setIdentificador("123-k");  
  
 model.addAttribute("titulo", "Formulario usuarios");  
 model.addAttribute("usuario", usuario);  
 return "form";  
 }  
  
 //El atributo object de thymeleaf en el elemento form  
 @PostMapping("/form")  
 public String procesar(@Valid Usuario usuario, BindingResult result, Model model, SessionStatus sessionStatus) {  
  
 model.addAttribute("titulo", "Resultado form");  
  
 if (result.hasErrors()) {  
 return "form";  
 }  
  
 model.addAttribute("usuario", usuario);  
 sessionStatus.setComplete();//elimina el objeto usuario de la sesion  
 return "resultado";  
 }  
  
}

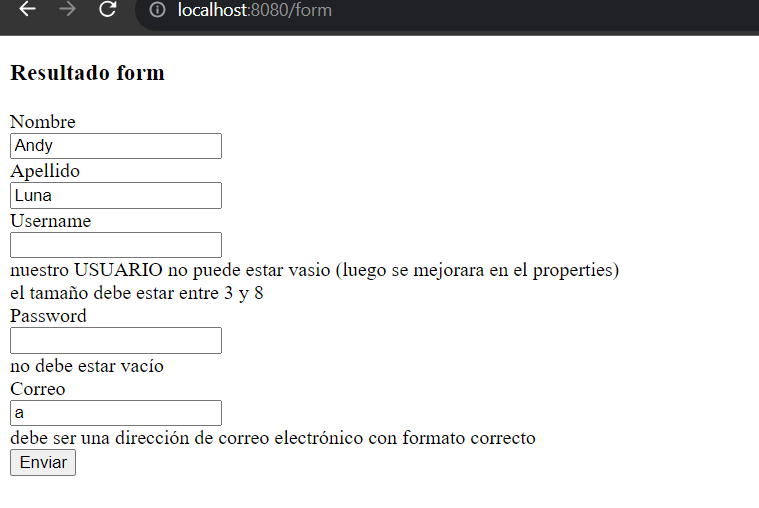
<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
<meta charset="UTF-8">  
<title th:text="${titulo}"></title>  
</head>  
<body>  
 <h3 th:text="${titulo}"></h3>  
  
 <ul>  
 <li th:text="${usuario.nombre}"></li>  
 <li th:text="${usuario.apellido}"></li>  
 <li th:text="${usuario.username}"></li>  
 <li th:text="${usuario.password}"></li>  
 <li th:text="${usuario.email}"></li>  
 <li th:text="${usuario.identificador}"></li>  
 </ul>  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
<meta charset="UTF-8">  
<title th:text="${titulo}"></title>  
</head>  
<body>  
 <h3 th:text="${titulo}"></h3>  
  
 <form th:action="@{/form}" method="post" th:object="${usuario}">  
  
<!-- mostrando valores de atributos de objeto model en el formulario (only nombre y apellido)-->  
 <div>  
 <label for="nombre">Nombre</label>  
 <div>  
 <input type="text" id="nombre" th:field="\*{nombre}">  
 </div>  
 <div th:if="${#fields.hasErrors('nombre')}" th:errors="\*{nombre}"></div>  
 </div>  
  
 <div>  
 <label for="apellido">Apellido</label>  
 <div>  
 <input type="text" id="apellido" th:field="\*{apellido}">  
 </div>  
 <div th:if="${#fields.hasErrors('apellido')}" th:errors="\*{apellido}"></div>  
 </div>  
  
 <div>  
 <label for="username">Username</label>  
 <div>  
 <input type="text" id="username" th:field="\*{username}">  
 </div>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{username}"></div>  
 </div>  
  
 <div>  
 <label for="password">Password</label>  
 <div>  
 <input type="text" id="password" th:field="\*{password}">  
 </div>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{password}"></div>  
 </div>  
  
 <div>  
 <label for="email">Correo</label>  
 <div>  
 <input type="text" id="email" th:field="\*{email}">  
 </div>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{email}"></div>  
 </div>  
   
 <div>  
 <div>  
 <input type="submit" value="Enviar">  
 </div>  
 </div>   
 </form>  
  
</body>  
</html>





# 642. La anotación @Size para rangos y @Email para validar correo electrónico



package com.bolsadeideas.springboot.form.app.models.domain;  
  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.Data;  
  
@Data  
public class Usuario {  
  
  
 private String identificador;  
  
 @NotEmpty  
 private String nombre;  
  
 @NotEmpty  
 private String apellido;  
  
 @NotEmpty (message = "nuestro USUARIO no puede estar vasio (luego se mejorara en el properties)")  
 @Size(min = 3, max = 8)  
 private String username;  
  
 @NotEmpty  
 private String password;  
  
 @NotEmpty  
 @Email  
 private String email;  
  
}

# 643. Mensajes de errores personalizados usando properties

Creamos un archivo messages.properties

NotEmpty=el campo es requerido  
NotEmpty.usuario.nombre=el campo NOMBRE es requerido

Y podemos validar todos, aquí validamos todos los campos con la anotación @NotEmpty y el objeto con el atributo usuario, MEJORAREMOS ESTO EN EL SGTE

# 644. Archivos properties y acentos con UTF-8

En propiedades cambiar a utg 8 para los acentos

# 645. Validación personalizada usando anotación @Pattern para expresiones regulares

# 

EN LA CLASE USUARIO

// 3 dígitos seguidos de un punto, seguidos de 3 dígitos, seguidos de un punto, seguidos de 3 dígitos,  
// seguidos de un guión y, finalmente, seguidos de una letra mayúscula.  
 @Pattern(regexp = "\\d{3}\\.\\d{3}\\.\\d{3}-[A-Z]")  
  
 private String identificador;

# 646. Validación personalizada usando una clase Validation

Usuario

package com.bolsadeideas.springboot.form.app.models.domain;  
  
//import com.bolsadeideas.springboot.form.app.validations.IdentificadorRegex;  
import jakarta.validation.constraints.\*;  
import lombok.Data;  
@Data  
public class Usuario {  
  
// 3 dígitos seguidos de un punto, seguidos de 3 dígitos, seguidos de un punto, seguidos de 3 dígitos,  
// seguidos de un guión y, finalmente, seguidos de una letra mayúscula.  
 // @Pattern(regexp ="[0-9]{2}[.][\\d]{3}[.][\\d]{3}[-][A-Z]{1}")  
 private String identificador;  
  
 //@NotEmpty  
 private String nombre;  
  
 @NotEmpty  
 private String apellido;  
  
 @NotEmpty(message = "nuestro USUARIO no puede estar vasio (luego se mejorara en el properties)")  
 @Size(min = 3, max = 8)  
 @NotBlank //VALIDA NO ESTE VADIO  
 private String username;  
  
 @NotEmpty  
 private String password;  
  
 @NotEmpty  
 @Email  
 private String email;  
  
}

FormController

package com.bolsadeideas.springboot.form.app.controllers;  
  
import com.bolsadeideas.springboot.form.app.models.domain.Usuario;  
import com.bolsadeideas.springboot.form.app.validations.UsuarioValidador;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.WebDataBinder;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.InitBinder;  
import org.springframework.web.bind.annotation.PostMapping;  
import jakarta.validation.Valid;//@Valid  
import org.springframework.web.bind.annotation.SessionAttributes;  
import org.springframework.web.bind.support.SessionStatus;  
  
  
@Controller  
@SessionAttributes("usuario")// se guarda en una sesion http  
  
public class FormController {  
  
 @Autowired  
 private UsuarioValidador validador;  
  
 @GetMapping("/form")  
 public String form(Model model) {  
 Usuario usuario = new Usuario();  
  
 //mostrando valores de atributos de objeto model en el formulario (only nombre y apellido)  
 usuario.setNombre("Andy");  
 usuario.setApellido("Luna");  
 usuario.setIdentificador("12.456.789-K");  
 usuario.setEmail("andy@gmail.com");  
 usuario.setUsername("1234");  
 usuario.setPassword("1234");  
 model.addAttribute("titulo", "Formulario usuarios");  
 model.addAttribute("usuario", usuario);  
 return "form";  
 }  
  
 //El atributo object de thymeleaf en el elemento form  
 @PostMapping("/form")  
 public String procesar(@Valid Usuario usuario, BindingResult result, Model model, SessionStatus sessionStatus) {  
  
 //pasamos el validador que se inyecta  
 validador.validate(usuario, result);  
  
 model.addAttribute("titulo", "Resultado form");  
  
 if (result.hasErrors()) {  
 return "form";  
 }  
  
 model.addAttribute("usuario", usuario);  
 sessionStatus.setComplete();//elimina el objeto usuario de la sesion  
 return "resultado";  
 }  
  
}

UsuarioValidador

package com.bolsadeideas.springboot.form.app.validations;  
  
import com.bolsadeideas.springboot.form.app.models.domain.Usuario;  
import org.springframework.stereotype.Component;  
import org.springframework.validation.Errors;  
import org.springframework.validation.ValidationUtils;  
import org.springframework.validation.Validator;  
  
  
//LOS METODOS EL ID NOS AYUDA A GENERARLOS  
@Component  
public class UsuarioValidador implements Validator {  
  
 @Override  
 public boolean supports(Class<?> clazz) {  
 return Usuario.class.isAssignableFrom(clazz);  
 }//  
  
 @Override  
 public void validate(Object target, Errors errors) {  
 Usuario usuario = (Usuario) target;  
  
 //nombre no vacio o con espacios en blanco  
 ValidationUtils.*rejectIfEmptyOrWhitespace*(errors,"usuario","requerido.usuario.nombre");  
  
 //identificador  
 if(!usuario.getIdentificador().matches("[0-9]{2}[.][\\d]{3}[.][\\d]{3}[-][A-Z]{1}")){  
 errors.rejectValue("identificador","pattern.usuario.identificador");  
 }  
  
 }//  
  
}

message.properties

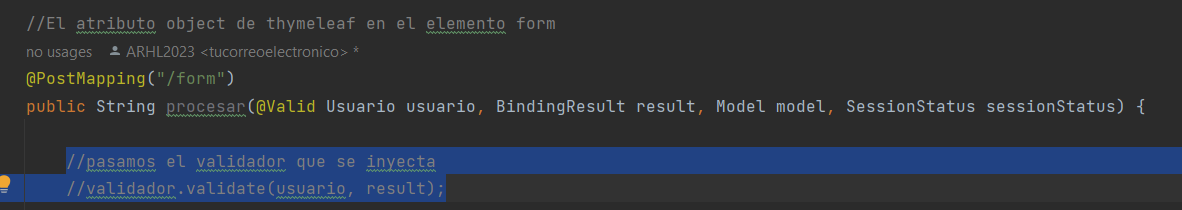
NotEmpty= el campo es requerido (NotEmpty)  
NotEmpty.usuario.nombre= el campo usuario es requerido (NotEmpty.usuario.nombre)  
requerido.usuario.nombre = campo nombre es requerido (requerido.usuario.nombre)  
pattern.usuario.identificador = formato de la expresion regular incorrecto (pattern.usuario.identificador) ("[0-9]{2}[.][\\d]{3}[.][\\d]{3}[-][A-Z]{1}")

# 647. Registrando una clase validador con la anotación @InitBinder

En form controller, arriba de los metodos

@InitBinder  
public void initBinder(WebDataBinder binder){  
 binder.addValidators(validador);  
}

Y ya no usamos el método del validsdor en el post, aquí simplemente lo comentaremos



# 648. Validación personalizada usando anotaciones

Es mejor validar como antes esto es complejo xd

Recordar marcar el atributo en el Usuario

@IdentificadorRegex  
private String identificador;

Y el método del usuario validador lo podemos comentar porque no lo usaremos

//identificador  
// if(!usuario.getIdentificador().matches("[0-9]{2}[.][\\d]{3}[.][\\d]{3}[-][A-Z]{1}")){  
// errors.rejectValue("identificador","pattern.usuario.identificador");  
// }

Pero primeri creamos la anotación (maracar como anotación no como clase)

IdentificadorRegex

package com.bolsadeideas.springboot.form.app.validations;  
  
  
import jakarta.validation.Constraint;  
import jakarta.validation.Payload;  
  
import java.lang.annotation.Retention;  
import java.lang.annotation.Target;  
  
import static java.lang.annotation.ElementType.*FIELD*;  
import static java.lang.annotation.ElementType.*METHOD*;  
import static java.lang.annotation.RetentionPolicy.*RUNTIME*;  
  
@Constraint(validatedBy = IdentificadorRegexValidador.class)  
@Retention(*RUNTIME*)  
@Target({*FIELD*,*METHOD*})  
public @interface IdentificadorRegex {  
  
 String message() default "Identificador invalido";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
  
}

y la clase

IdentificadorRegexValidador

package com.bolsadeideas.springboot.form.app.validations;  
  
import jakarta.validation.ConstraintValidator;  
import jakarta.validation.ConstraintValidatorContext;  
  
public class IdentificadorRegexValidador implements ConstraintValidator<IdentificadorRegex,String> {  
  
  
 @Override  
 public boolean isValid(String value, ConstraintValidatorContext constraintValidatorContext) {  
 if(value.matches("\\d{3}\\.\\d{3}\\.\\d{3}-[A-Z]")==false) {  
 return true;  
 }  
 return false;  
 }  
}

la diferencia es que como isismos antes con una clase Validador podemos validar varios campos, y con anotaciones se valida un valor en especifico pero puede replicar y reutilizar en varios campos

usando anotaciones es la forma nativa de la api anotación de java, y usando clases con la interfaz validator es la forma de spring

# 649. Validación personalizada usando anotaciones ejemplo requerido

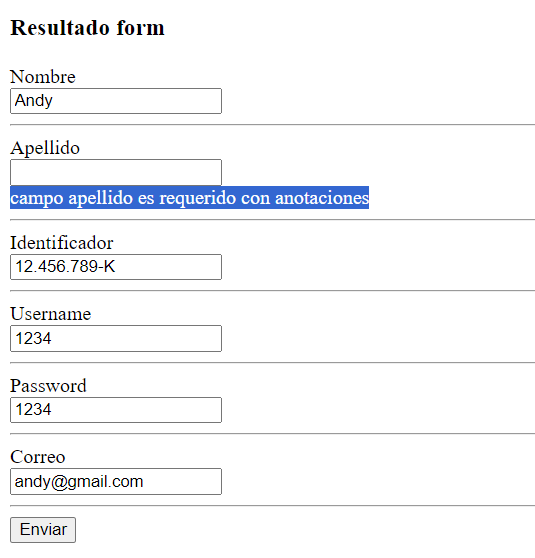
Practicamos usando anotaciones para el apellido en ves de not empty

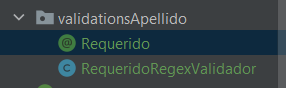
Primeor comentamos su anotación not empty y la anotamos, en este caso requerido

// @NotEmpty  
 @Requerido  
 private String apellido;

Tambien podemos usar el @Requerido en otro campo como en el email comentando su notempty, el email tendrá el mensaje propio de requerido mientras apellido tendrá el del properties

// @NotEmpty  
 @Requerido  
 @Email  
 private String email;





Requerido

package com.bolsadeideas.springboot.form.app.validations.validationsApellido;  
  
  
import jakarta.validation.Constraint;  
import jakarta.validation.Payload;  
  
import java.lang.annotation.Retention;  
import java.lang.annotation.Target;  
  
import static java.lang.annotation.ElementType.*FIELD*;  
import static java.lang.annotation.ElementType.*METHOD*;  
import static java.lang.annotation.RetentionPolicy.*RUNTIME*;  
  
@Constraint(validatedBy = RequeridoRegexValidador.class)  
@Retention(*RUNTIME*)  
@Target({*FIELD*,*METHOD*})  
public @interface Requerido {  
  
 String message() default "campo es requerido con anotaciones";  
 Class<?>[] groups() default {};  
 Class<? extends Payload>[] payload() default {};  
  
}

RequeridoRegexValidator

package com.bolsadeideas.springboot.form.app.validations.validationsApellido;  
  
import jakarta.validation.ConstraintValidator;  
import jakarta.validation.ConstraintValidatorContext;  
import org.springframework.util.StringUtils;  
  
  
public class RequeridoRegexValidador implements ConstraintValidator<Requerido,String> {  
  
  
 @Override  
 public boolean isValid(String value, ConstraintValidatorContext constraintValidatorContext) {  
  
 if( value==null || !StringUtils.*hasText*(value)){ //value.isEmpty() || value.isBlank() eslo mismo que !StringUtils.hasText(value)  
 return false;  
 }  
 return true;  
 }//  
  
}

# 650. Validación de números enteros con @Max @Min y NotNull

En form html creamos su input

<div>

            <label for="cuenta">Cuenta</label>

            <div>

                <input type="text" id="cuenta" th:field="\*{cuenta}">

            </div>

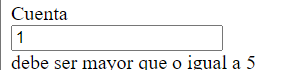
            <div th:if="${#fields.hasErrors('cuenta')}" th:errors="\*{cuenta}"></div>

        </div>

        <hr>

Y en resultado

        <li th:text="${usuario.cuenta}"></li>



En el properties colocamos por si se ingresa una letra (toman en cuenta que el input es text)

typeMismatch.java.lang.Integer = debe ser un entero  
typeMismatch.usuario.cuenta = el campo cuenta ser un entero



Si el typy es number en el input solo deja escribir números asi que es meor dejarlo en number

Para continuar lo dejaremos como number

# 651. Validación de fechas con @NotNull dando formato con @DateTimeFormat

En el form.html

<div>

            <label for="fechaNacimeinto">Fecha Nacimiento</label>

            <div>

                <input type="text" id="fechaNacimeinto" th:field="\*{fechaNacimeinto}" placeholder="yyyy/MM/dd">

            </div>

            <div th:if="${#fields.hasErrors('fechaNacimeinto')}" th:errors="\*{fechaNacimeinto}"></div>

        </div>

        <hr>

Resultado.html

        <li th:text="${usuario.fechaNacimeinto}"></li>

En Usuario en el back

@NotNull  
@DateTimeFormat(pattern = "yyyy/MM/dd")  
private Date fechaNacimeinto;

En el properties

typeMismatch.java.util.Date = fecha debe ser formato yyyy/MM/dd (typeMismatch.java.util.Date)

typeMismatch.usuario.FechaNacimeinto = fecha debe ser formato yyyy/MM/dd (typeMismatch.usuario.FechaNacimeinto)

cualquiera de las 2

# 652. Validación de fechas con input type date de HTML5 y @Past y @Future

En el form solo cambio el inout de text a date

<div>

            <label for="fechaNacimeinto">Fecha Nacimiento</label>

            <div>

                <input type="date" id="fechaNacimeinto" th:field="\*{fechaNacimeinto}" placeholder="yyyy/MM/dd">

            </div>

            <div th:if="${#fields.hasErrors('fechaNacimeinto')}" th:errors="\*{fechaNacimeinto}"></div>

        </div>

        <hr>

En Usuario

// @Future  
// @Past  
 @NotNull  
 @DateTimeFormat(pattern = "yyyy/MM/dd")  
 private Date fechaNacimeinto;

Aquí no usaremos estas anotaciones para validar una fecha después o anterior a la actual

# 653. Formateando fechas con @lnitBinder y registrando un CustomDateEditor

Comentamos en el Usuario el @DateTimeFormat

En Form Controlle teníamos

@InitBinder  
public void initBinder(WebDataBinder binder){  
 binder.addValidators(validador);  
}

Agregamos para las fechas, seria lo mismo que el @DateTimeFormat en el model pero esto es que init binder

@InitBinder  
public void initBinder(WebDataBinder binder){  
 binder.addValidators(validador);  
  
 //fechas seria lo mismo del @DateTimeFormat  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
 binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat,false));  
 dateFormat.setLenient(false);  
}

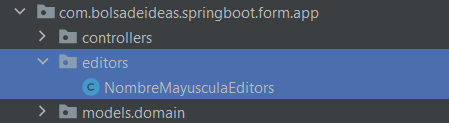
# 654. Formateando fechas con @lnitBinder y registrando un CustomDateEditor Parte 2

//binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat,false));  
binder.registerCustomEditor(Date.class,"fechaNacimiento", new CustomDateEditor(dateFormat,false));

La diferencia es que ahora el custom editor es para un atributo particular y no para todos, MYORMENTE NO HABRA DIFERENCIA AQUI

# 655. Implementando propio filtro custom property editor para convertir a mayúscula

AQUÍ SE HARA QUE EL TEXTO DE OMBRE YA APELLIDO PASE COMO MAYUSCULA



Creamos la clase NombreMayusculaEditors

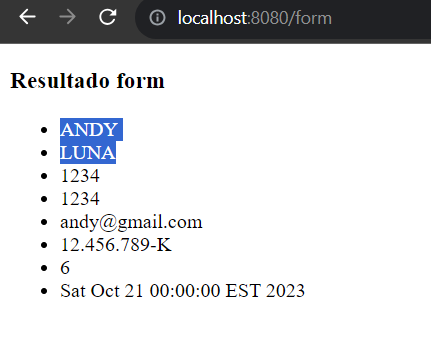
package com.bolsadeideas.springboot.form.app.editors;  
  
import java.beans.PropertyEditorSupport;  
  
public class NombreMayusculaEditors extends PropertyEditorSupport {  
  
 //podemos crear este metodo con el ide- generate-override y buscamos setAsText  
 @Override  
 public void setAsText(String text) throws IllegalArgumentException {  
 setValue(text.toUpperCase().trim());  
 }  
  
}

en el form controller

@InitBinder  
public void initBinder(WebDataBinder binder) {  
 binder.addValidators(validador);  
  
 //fechas seria lo mismo del @DateTimeFormat  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
 dateFormat.setLenient(false);  
  
 //SOLO COMENTE PARA TENER LAS 2 VERSIONES, la seguna aplica el formateo al campo fechaNacimeinto, la priema a todos los date  
 //binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat,false));  
 binder.registerCustomEditor(Date.class, "fechaNacimeinto", new CustomDateEditor(dateFormat, false));  
  
 //format para mayusculas, si quieor hacer con mas solo debo copiar e indicar los otros campos  
 binder.registerCustomEditor(String.class, "nombre", new NombreMayusculaEditors());  
 binder.registerCustomEditor(String.class, "apellido", new NombreMayusculaEditors());  
  
}//

Lo que se agreogo son las ultimas 2 lineas

En el form estaban con minusculas



# 656. Lista select desplegable

En el Usuario

@NotEmpty  
private String pais;

En form controller

//recuerda que ModelAttribute hace que un metodo lo que retorna se pasa y se guarda en la vista  
@ModelAttribute("paises")  
public List<String> paises(){  
 return Arrays.*asList*("Ecuador","Argentina","Marruecos","Japón");  
}

En form

        <div>

            <label for="pais">Pais</label>

            <div>

                <select id="pais" th:field="\*{pais}">

                    <option value="">Seleccionar</option>

                    <option th:each="pais:${paises}" th:text="${pais}" th:value="${pais}"></option>

                </select>

            </div>

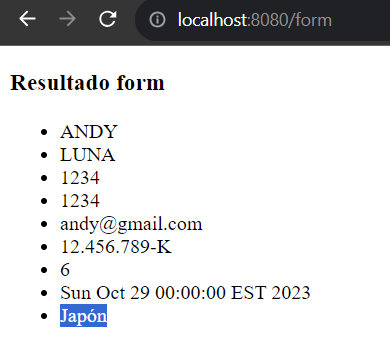
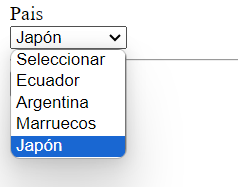
            <div th:if="${#fields.hasErrors('pais')}" th:errors="\*{pais}"></div>

        </div>

        <hr>

En Resultado

        <li th:text="${usuario.pais}"></li>

# 657. Formateando fechas en vistas thymeleaf

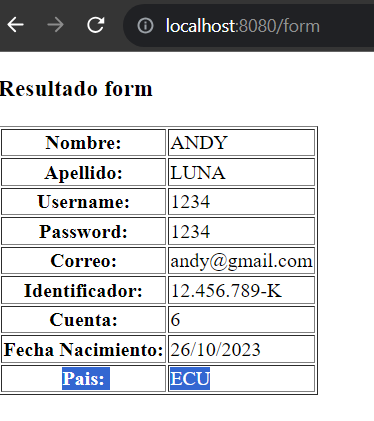
En resultado.html

           <td th:text="${#dates.format(usuario.fechaNacimeinto,'dd/MM/yyyy')}"></td>



# 658. Llenando lista select con Map

Usando Map llave y valor mostraremso solo la llave



En cotroller no usaremos el list de paises, usaremos paísesMap

@ModelAttribute("paisesMap")  
public Map<String,String> paisesMap(){  
 Map<String,String> paises = new HashMap<>();  
 paises.put("ECU","Ecuador");  
 paises.put("ARG","Argentina");  
 paises.put("MAR","Marruecos");  
 paises.put("JAP","Japón");  
 return paises;  
}

Y modificamos el form.html solo el div de países

<div>

                <label for="pais">Pais</label>

                <div>

                    <select id="pais" th:field="\*{pais}">

                        <option value="">Seleccionar</option>

                        <option th:each="pais:${paisesMap.entrySet()}" th:text="${pais.value}" th:value="${pais.key}"></option>

                    </select>

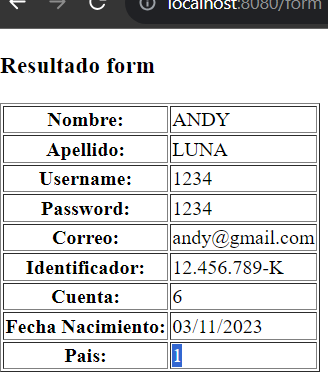
                </div>

                <div th:if="${#fields.hasErrors('pais')}" th:errors="\*{pais}"></div>

            </div>

            <hr>

# 659. Llenando lista select con objetos del la clase Pais



En form.html

<!-- 659. Llenando lista select con objetos del la clase Pais -->

            <div>

                <label for="pais.id">Pais</label>

                <div>

                    <select id="pais.id" th:field="\*{pais.id}">

                        <option value="">Seleccionar</option>

                        <option th:each="pais:${listaPaises}" th:text="${pais.nombre}" th:value="${pais.id}"></option>

                    </select>

                </div>

                <div th:if="${#fields.hasErrors('pais.id')}" th:errors="\*{pais.id}"></div>

            </div>

            <hr>

En resultado.html

<td th:text="${usuario.pais.id}"></td>

En controller

//659. Llenando lista select con objetos del la clase Pais  
@ModelAttribute("listaPaises")  
public List<Pais> listaPaises() {  
 // Crea una lista de objetos Pais  
 return Arrays.*asList*(  
 new Pais(1, "EC", "Ecuador"),  
 new Pais(2, "AR", "Argentina"),  
 new Pais(3, "MA", "Marruecos"),  
 new Pais(4, "JP", "Japón")  
 );  
}

En modelo Usuario

private Pais pais;

Creamos modelo Pais

package com.bolsadeideas.springboot.form.app.models.domain;  
  
import jakarta.validation.constraints.NotNull;  
import lombok.Data;  
  
@Data  
public class Pais {  
  
 @NotNull  
 private Integer id;  
 private String codigo;  
 private String nombre;  
  
 public Pais() {  
 }  
  
 public Pais(Integer id, String codigo, String nombre) {  
 this.id = id;  
 this.codigo = codigo;  
 this.nombre = nombre;  
 }  
  
}

# 660. Añadiendo componente service de Pais

# 661. Añadiendo componente pais property editor

# 662. Implementando checkboxes

# 663. Llenando checkboxes con Map

# 664. Llenando checkboxes con objetos del tipo Role

# 665. Añadiendo componente role property editor

# 666. Checkbox booleano true o false

# 667. Radio button

# 668. Input type hidden

# 669. Poblando pais y roles

# 670. Redirect después del POST procesar

# 671. Añadiendo estilos CSS al formulario con Bootstrap

# 672. Añadiendo estilos CSS Bootstrap a la plantilla resultado

# 673. Cargando archivo css Bootstrap localmente