UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

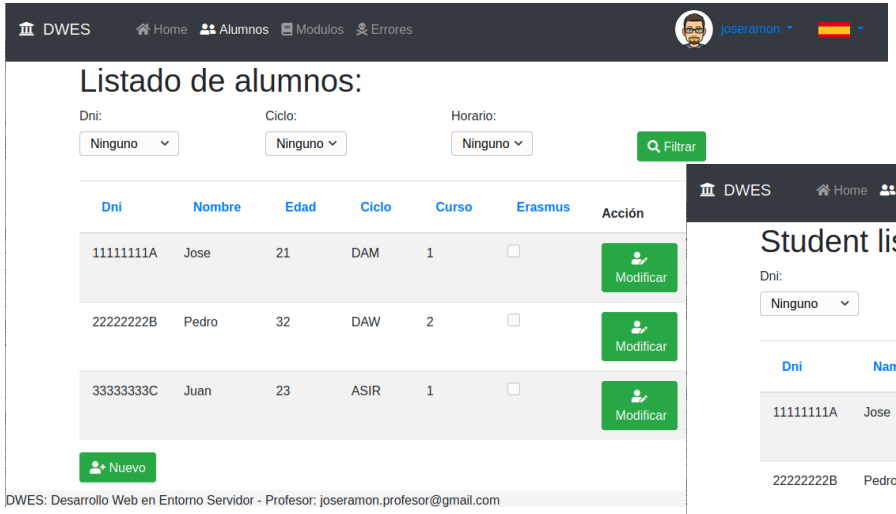
- *Comprender el **estándard i18n**.*
- *Aprender a **configurar varios idiomas en nuestra aplicación.***
- *Detectar el **idioma del navegador y configurar la respuesta del servidor** en dicho idioma.*
- *Configurar el **enconding** en nuestra webapp.*
- *Aprender a **internacionalizar los mensajes de los validadores**, tanto por defecto como los creados por nosotros.*

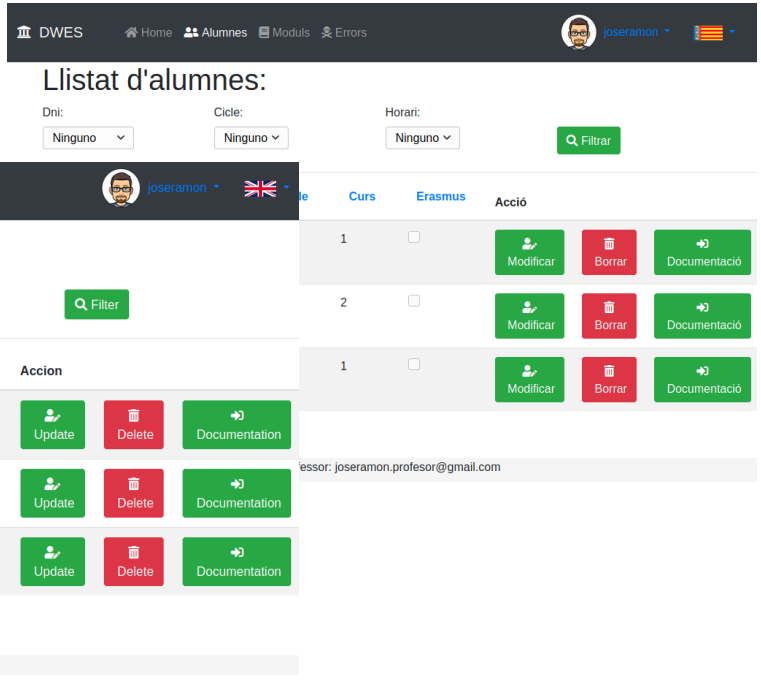
UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

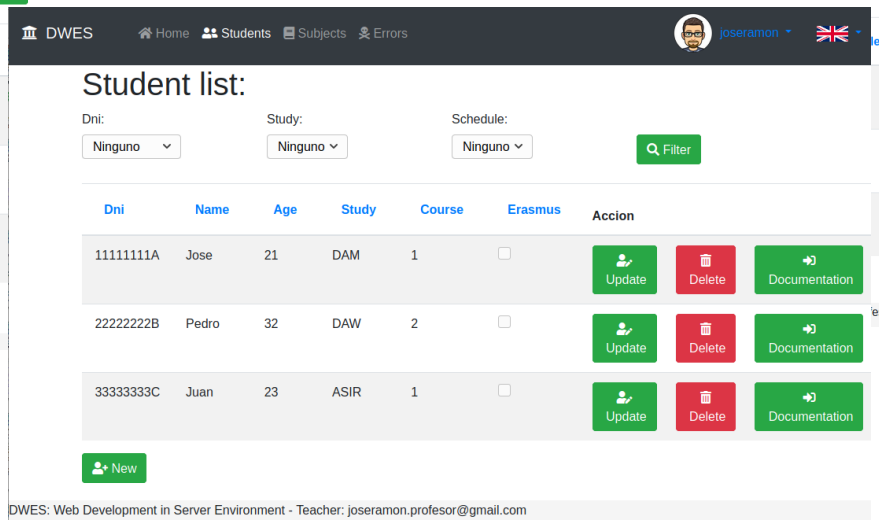
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Es cada vez más importante crear aplicaciones que muestren la información en más de un idioma. Cuando hablamos de **internacionalizar una aplicación** es muy útil contar con un lenguaje de programación que nos permita utilizar un **estandar i18n fácil de usar**.

i18n es la abreviatura de “internationalization”, palabra inglesa que tiene 18 letras, y entre la “i” y la “n” hay 18 caracteres.







¿Que sistema nos proporciona Spring para mostrar la información en varios idiomas de una manera cómoda?

UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

La idea básica es **tener un fichero de texto por idioma soportado**, en donde guardaremos las cadenas de texto con las traducciones. **La aplicación consultará el fichero correspondiente dependiendo del idioma a utilizar**.

messages_en.properties

```
1 alumnos.menuSuperior=Students
2 modulos.menuSuperior=Subjects
3 errores.menuSuperior=Errors
4 desconocido.usuario=Unknown
5 usuario.editar=Edit
6 usuario.foto=Photo
7 usuario.salir=Logout
8 login.introduzca.usuario=Enter
9 login.introduzca.pw=Enter your
10 home.bienvenida= Welcome
11 alumnos.titulo=Student list
12 modulos.titulo=Subject list
13 modulos.explicacion=Fill in da
14 log.errores.titulo=Error logs
15 usuarioImagen.titulo=User Image
```

messages_es.properties

```
1 alumnos.menuSuperior=Alumnos
2 modulos.menuSuperior=Modulos
3 errores.menuSuperior=Errores
4 desconocido.usuario=Desconocido
5 usuario.editar=Editar
6 usuario.foto=foto
7 usuario.salir=Salir
8 login.introduzca.usuario=Introduzca su
9 login.introduzca.pw=Introduzca su cont
10 home.bienvenida=Bienvenid@
11 alumnos.titulo=Listado de alumnos
12 modulos.titulo=Listado de módulos
13 modulos.explicacion=Introduzca los datos del nu
14 log.errores.titulo=Log de errores
15 usuarioImagen.titulo=Imagen del usuario
```

messages_ca.properties

```
1 alumnos.menuSuperior=Alumnes
2 modulos.menuSuperior=Moduls
3 errores.menuSuperior=Errors
4 desconocido.usuario=Desconegut
5 usuario.editar=Editar
6 usuario.foto=foto
7 usuario.salir=Eixir
8 login.introduzca.usuario=Introduïx l'usuari
9 login.introduzca.pw=Introduïx la contrasenya
10 home.bienvenida=Benvingut
11 alumnos.titulo=Llistat d'alumnes
12 modulos.titulo=Llistat de mòduls
13 modulos.explicacion=Introduïx les dades del n
14 log.errores.titulo=Log d'errors
15 usuarioImagen.titulo=Imatge de l'usuari
```

Podemos ver que para no volvernos locos en una aplicación con miles de traducciones, tendremos que crear una reglas de nombrado de mensajes.

¿Que pasos necesitamos realizar para configurar el i18n en nuestra aplicación?

Pasos:

Configurar los mensajes:

1º Configurar los origenes de los mensajes

Vistas:

2º Configurar e internacionalizar las vistas (JSPs)

Controladores y Servicios:

3º Detectar y configurar el idioma del navegador

4º Internacionalizar los mensajes de texto que envían los controladores

Encoding UTF-8:

5º Configurar el encoding de nuestra app como UTF-8

Validadores:

6º Configurar mensajes de las anotaciones de los validadores por defecto

7º Configurar mensajes de los validadores personalizados

Vista:

8º Configurar iconos para cambiar idioma

1º Configurar los origenes de los mensajes:

Para conseguirlo debemos de modificar el fichero alumno-servlet.xml para **configurar los orígenes de los mensajes ("messageSource") donde almacenar las cadenas de texto en castellano, ingles y valenciano**. Para ello configuraremos los beans **"messageSource", "localeResolver" y "localeChangeInterceptor"**. Este último nos permitirá cambiar el lenguaje utilizando el parámetro "language":

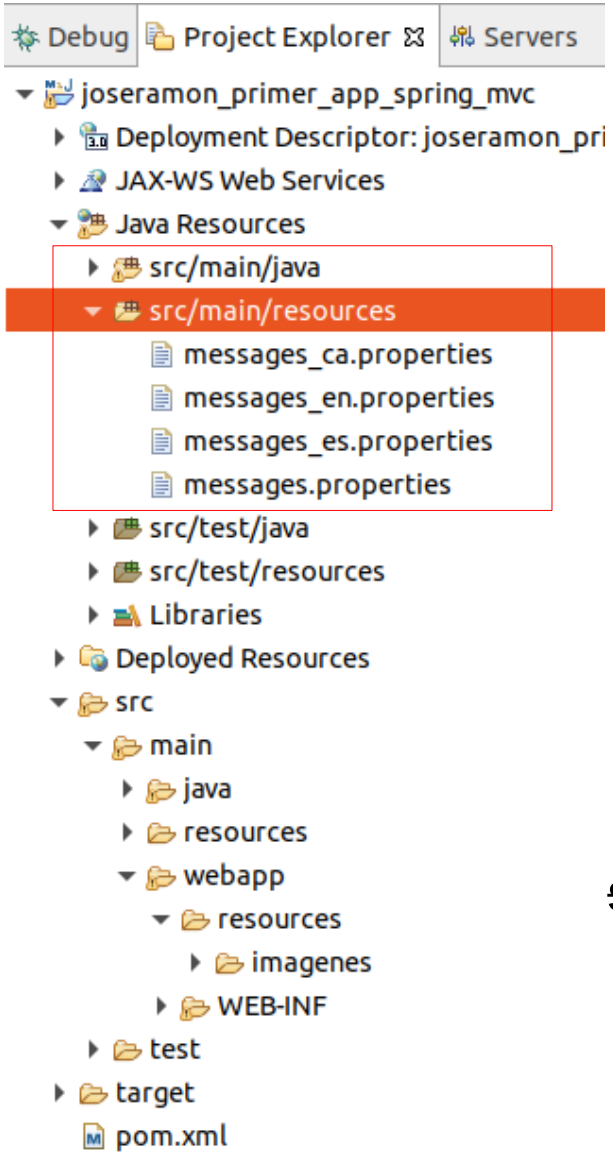
alumno-servlet.xml

```
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
</bean>
<bean id="multipartResolver"
        class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>

<!-- Internacionalización i18n -->
<bean id="messageSource"
    class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basename" value="classpath:messages" />
    <property name="defaultEncoding" value="UTF-8" />
</bean>
<bean id="localeResolver"
    class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
    <property name="defaultLocale" value="es" />
</bean>
<mvc:interceptors>
    <bean id="localeChangeInterceptor"
        class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
        <property name="paramName" value="language" />
    </bean>
</mvc:interceptors>
<!-- Fin internacionalización i18n -->

<mvc:resources mapping="/webjars/**" location="/webjars/"/>
<mvc:resources mapping="/imagenes/**" location="/resources/imagenes/"/>
</beans>
```

1º Configurar los origenes de los mensajes:



Ahora debemos de copiar los ficheros donde almacenaremos las traducciones.

Copiar los 4 ficheros con las traducciones en castellano (es) , ingles (en) y valenciano (ca) del fichero comprimido del Drive para que esten en la carpeta resources. Se añade un 4º fichero que será el que se utilizará cuando se indique un idioma que no exista.

Si nemos problemas visualizando los ficheros con las traducciones ver pag. 16 para cambiar el enconding a UTF-8.

Ojo: Los ficheros deben copiarse dentro de la carpeta **src\main\resources**, y NO en la carpeta ~~src\main\webapp\resources~~.

2º Configurar e internacionalizar las vistas (JSPs):

Si queremos poder **utilizar i18n en nuestras vistas (ficheros .jsp)** deberemos de incluir una librería más con el prefijo “spring” en el fichero “header.jspf”:

header.jspf

```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
3  <%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <link href="webjars/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
8  <link href="webjars/font-awesome/5.15.1/css/all.min.css" rel="stylesheet">
9  <title>${pagina.getTitulo()}</title>
10 <style>
11     .footer {
12         position: absolute;
13         bottom: 0;
14         width: 100%;
15         height: 60px;
16         background-color: #f5f5f5;
17     }
18 </style>
19 </head>
20 <body>
```

Hará falta volver a hacer “Maven Install” para que se descargue la librería correspondiente.

2º Configurar e internacionalizar las vistas (JSPs):

Vamos a empezar internacionalizando el menú superior. Para poder leer las traducciones del fichero correspondiente debemos utilizar el tag “spring:message” con el nombre que tengamos configurado en el fichero de traducciones:

menuSuperior.jspf

```
12    <ul class="navbar-nav mr-auto">
13      <li class="nav-item ${pagina.getStrBootstrapActiva("login")}">
14        <a class="nav-link" href="login"><i class="fas fa-home"></i> Home
15        </a>
16      </li>
17        <li class="nav-item ${pagina.getStrBootstrapActiva("list-alumno")}">
18        <a class="nav-link" href="list-alumno">
19            <i class="fas fa-user-friends"></i> <spring:message code="alumnos.menuSuperior" />
20        </a>
21      </li>
22        <li class="nav-item ${pagina.getStrBootstrapActiva("list-modulo")}">
23        <a class="nav-link" href="list-modulo">
24            <i class="fas fa-book"></i> <spring:message code="modulos.menuSuperior" />
25        </a>
26      </li>
27        <li class="nav-item ${pagina.getStrBootstrapActiva("list-logerror")}">
28        <a class="nav-link" href="list-logerror">
29            <i class="fas fa-skull-crossbones"></i> <spring:message code="errores.menuSuperior"
30        </a>
31      </li>
32  </div>
33    <div class="nav-item dropdown">
34        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="d
35            <img alt="${loginName}" src="imagenUsuario/${loginNickName}" class="rounded-circle" style
36           ${loginNickName}
37        </a>
```

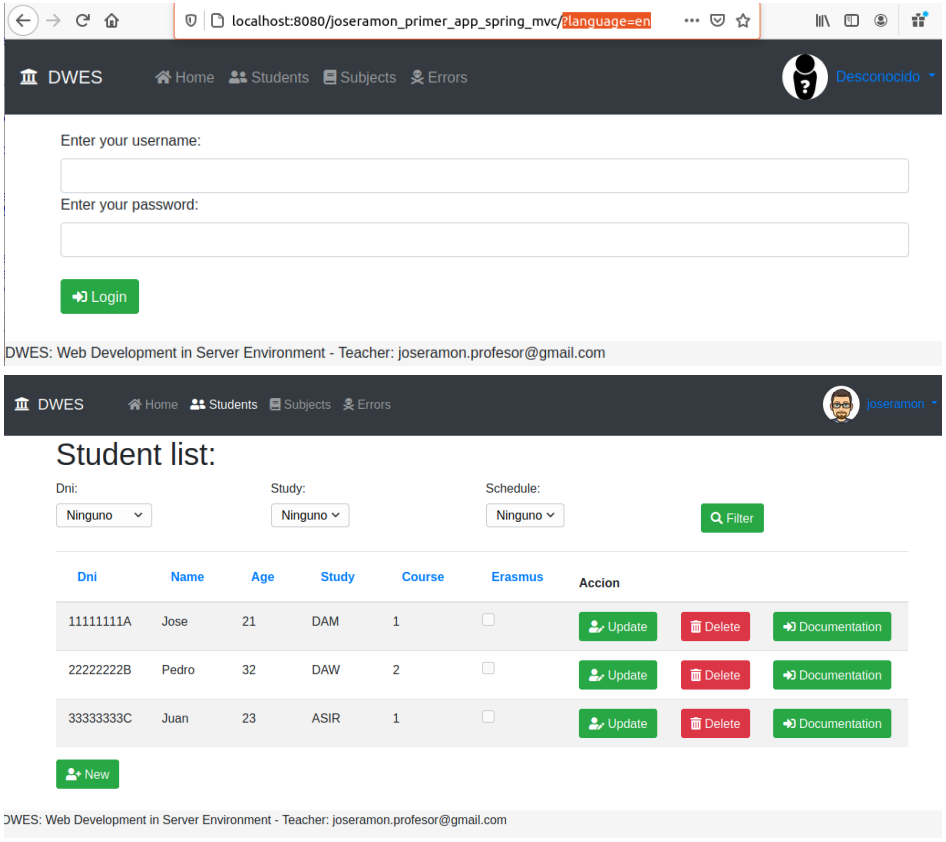
¿Como lo probamos?

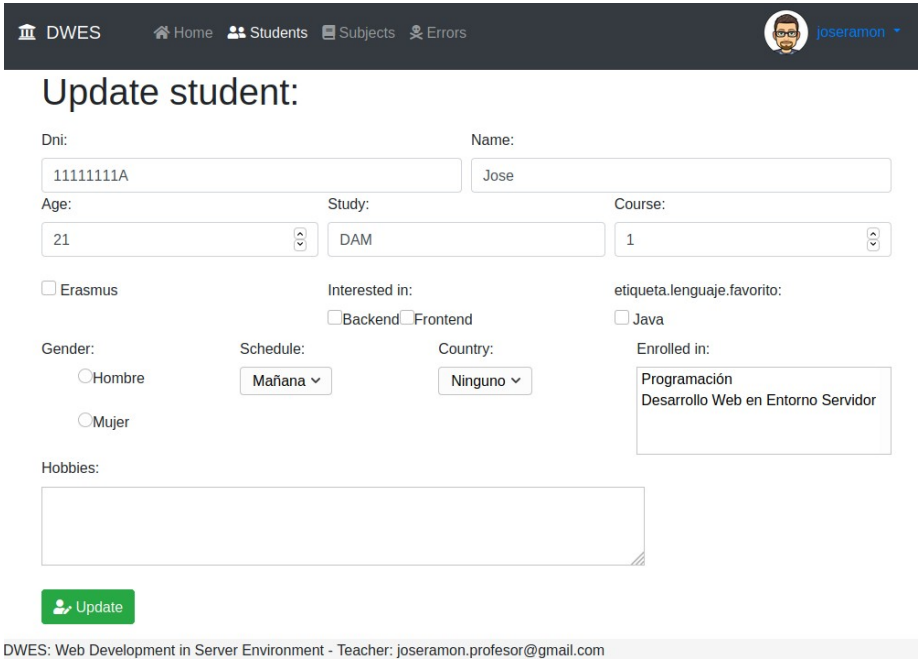
UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

2º Configurar e internacionalizar las vistas (JSPs):

Para probarlo añadimos a la url de cualquier página de nuestra web el parámetro con el idioma. Debemos añadir a la url **“?language=en”** (el parámetro language lo hemos definido en alumno-servlet.xml). Modifica TODAS las vistas de la web (jsp’s) para incorporar las traducciones que estan ya hechas en los ficheros de properties: Por ahora no nos preocuparemos de como traducir lo que llega del servidor, solo los textos dentro de los jsp...





3º Detectar y configurar el idioma del navegador:

No vamos a traducir todavía los mensajes que vienen del servidor, sino que vamos a **detectar el lenguaje de nuestro navegador para ponerlo por defecto:**

Configura en tu navegador el valenciano como 1º idioma por delante del castellano (idioma por defecto en la app) para esta prueba.

Si añadimos el siguiente código a 'mostrarLogin' de LoginController podemos ver por consola que si tenemos el navegador en valenciano nos muestra dicha información y nos dice que el idioma por defecto es "es" =castellano (al haberlo configurado en alumno-servlet.xml).

¿ Como configuramos la app con el idioma del navegador?

LoginController.java

```
21
22 @Controller
23 @SessionAttributes({"usuario","loginName","loginNickName"})
24 public class LoginController {
25     @Autowired
26     LoginService loginService;
27     @Autowired
28     LogErrorService logErrorService;
29
30     @RequestMapping(value={"/","/login"},method = RequestMethod.GET)
31     public String mostrarLogin(HttpServletRequest request,
32             Locale locale,ModelMap model) {
33         //Traza i18n
34         //Información idioma de la peticion del navegador
35         System.out.println("Accept-Language: "
36             +request.getHeader("Accept-Language"));
37         //Información del localeResolver
38         System.out.println(String
39                 .format("Petición recibida. Languaje: %s, País: %s %n",
40                 locale.getLanguage(), locale.getDisplayCountry()));
41         // Datos para la cabecera de la página
42         model.put("loginName", "Usuario desconocido");
43         model.put("loginNickName", "Desconocido");
44         //Datos para el formulario de login
45         model.addAttribute(new Usuario());
46         return "login";
47     }
```

Console | Problems | Debug Shell

```
Tomcat v9.0 Server at localhost [Apache Tomcat] /usr/lib/jvm/java-13-openjdk-amd64/bin/java

Accept-Language: ca-valencia,es-ES;q=0.8,es;q=0.6,en-US;q=0.4,en;q=0.2
Petición recibida. Languaje: es, País:
```

UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

3º Detectar y configurar el idioma del navegador:

La idea es detectar el lenguaje de nuestro navegador , en vez de tener que pasar “language” . Para ello vamos a copiar un servicio nuevo I18nService.java que se proporciona parcialmente en el fichero comprimido del Drive:

Lee atentamente el código suministrado para poder entender su funcionamiento.Desde LoginController, en el método ‘mostrarLogin’ que atiende el GET deberemos de llamar al método ‘configuraIdiomaPeticion’ y deberemos de tener los parámetros que necesitamos en el método mostrarLogin para poder configura el idioma:

```
@RequestMapping(value={"/","/login"},method = RequestMethod.GET)
public String mostrarLogin(HttpServletRequest request,
        HttpServletResponse response,
        Locale locale,ModelMap model) {
    //i18n:Detectamos idioma de la petición (lo rellena el navegador) y lo
    //cambiamos si es distinto al valor por defecto (locale).
    i18nService.configuraIdiomaPeticion(request, response, locale);
```

I18nService.java

```
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.context.NoSuchMessageException;
16 import org.springframework.context.i18n.LocaleContextHolder;
17 import org.springframework.context.support.ReloadableResourceBundleMessageSource;
18 import org.springframework.stereotype.Service;
19 import org.springframework.web.servlet.i18n.SessionLocaleResolver;
20
21 @Service
22 public class I18nService {
23     @Autowired
24     private ReloadableResourceBundleMessageSource i18n_mensaje;
25     @Autowired
26     private SessionLocaleResolver idiomaPeticiones; //bean definido en alumno-service.xml
27
28     //Consulta el idioma configurado en la peticion y devuelve su traducción
29     private String getI18nMessage(String msg) {
32
33     // Devuelve la lista traducida al idioma configurado en la petición
34     public List<String> getTraduccion(List<String> listaMsgOriginal) {
47
48     // Devuelve el mapa con el contenido (2º String)traducido al idioma configurado en la petición
49     public Map<String, String> getTraduccion(Map<String, String> mapMsgOriginal) {
62
63     public String getTraduccion(String msgOriginal) {
72
73     public String getTraduccion(String msgOriginal,String parametro1) {
82
83     //Detectamos idioma de la petición (navegador) y lo cambiamos si es distinto al
84     //valor por defecto.
85     public void configuraIdiomaPeticion(HttpServletRequest request,
86             HttpServletResponse response, Locale locale) {
87         //Consultamos el idioma de la petición. El navegador suele incluir en
88         //su petición el idioma configurado por defecto
89         //System.out.println("Idioma por defecto: "+locale.getLanguage());
90         Optional<Locale> idiomaPeticion= getIdiomaPeticion(request);
91         if (idiomaPeticion.isPresent()) {//La petición contiene idioma
92             //porque lo rellena el navegador
93             //System.out.println("Idioma Petición: "+idiomaPeticion.get().getLanguage());
94             if (!locale.equals(idiomaPeticion.get())){
95                 //Cambiamos el idioma de la sesión al de la petición
96                 //System.out.println("Actualizamos idioma al de la petición");
97                 idiomaPeticiones.setLocale(request, response, idiomaPeticion.get());
98             }
99         }
100    }
101    public Optional<Locale> getIdiomaPeticion(HttpServletRequest request) {
102        Locale locale = request.getLocale();
103        if (locale == null)// la petición no ha enviado 'Accept-Language'
104            return Optional.empty();
105        else // Devolvemos el idioma de la petición (idioma del navegador)
106            return Optional.of(locale);
107    }
108 }
```

UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

4º Internacionalizar los mensajes de texto que envían los controladores:

Para internacionalizar los mensajes, el método más limpio es crear métodos nuevos en I18nService y luego llamarlos desde el controlador:

Si tenemos una lista de String's habrá un 'getTraducción' cuyo parámetro sea una lista de String's y devuelva la lista de String's traducida.

En el caso del HasMap, el primer valor será el indice y el segundo el que se traducirá.Implementar los métodos que están minimizados en la captura de pantalla:

I18nService.java

```java
@Service
public class I18nService {
    @Autowired
    private ReloadableResourceBundleMessageSource i18n_mensaje;
    @Autowired
    private SessionLocaleResolver idiomaPeticiones; //bean definido en alumno-service.xml

    //Consulta el idioma configurado en la peticion y devuelve su traducción
    private String getI18nMessage(String msg) {
        return i18n_mensaje.getMessage(msg, null,LocaleContextHolder.getLocale());
    }

    // Devuelve la lista traducida al idioma configurado en la petición
    public List<String> getTraduccion(List<String> listaMsgOriginal) {

    // Devuelve el mapa con el contenido (2º String)traducido al idioma configurado en la petición
    public Map<String, String> getTraduccion(Map<String, String> mapMsgOriginal) {

    public String getTraduccion(String msgOriginal) {
        //Idioma actual de la aplicación = LocaleContextHolder.getLocale()
        try {
        return getI18nMessage(msgOriginal);
        } catch(NoSuchMessageException e) {
            System.out.println("ERROR: String I18nService.getTraduccion(String):"+ e.getMessage());
            return msgOriginal;
        }
    }

    //Detectamos idioma de la petición (navegador) y lo cambiamos si es distinto al
    //valor por defecto.
    public void configuraIdiomaPeticion(HttpServletRequest request,
            HttpServletResponse response, Locale locale) {
        //Consultamos el idioma de la petición. El navegador suele incluir en
        //su petición el idioma configurado por defecto
        //System.out.println("Idioma por defecto: "+locale.getLanguage());
        Optional<Locale> idiomaPeticion= getIdiomaPeticion(request);
```

AlumnoController.java

```java
    @ModelAttribute("opcionesGenero")
    public List<String> getListaGenero() {
        List<String> i18nLista=i18nService.getTraduccion(alumnoService.listaGenero());
        return i18nLista;
    }

    @ModelAttribute("opcionesTipoDoc")
    public List<String> getListaTipoDoc() {
        List<String> i18nLista=i18nService.getTraduccion(alumnoService.listaTipoDoc());
        return i18nLista;
    }


    @ModelAttribute("horarioLista")
    public List<String> getListaHorario() {
        List<String> i18nLista=i18nService.getTraduccion(alumnoService.listaHorarios());
        return i18nLista;
    }

    @ModelAttribute("paisLista")
    public Map<String, String> getListaPais() {
        Map<String, String> i18nLista=i18nService.getTraduccion(alumnoService.listaPais());
        return i18nLista;
    }

    @ModelAttribute("moduloLista")
    public List<Modulo> getListaModulos() {
        return moduloService.listaModulos();
    }
```

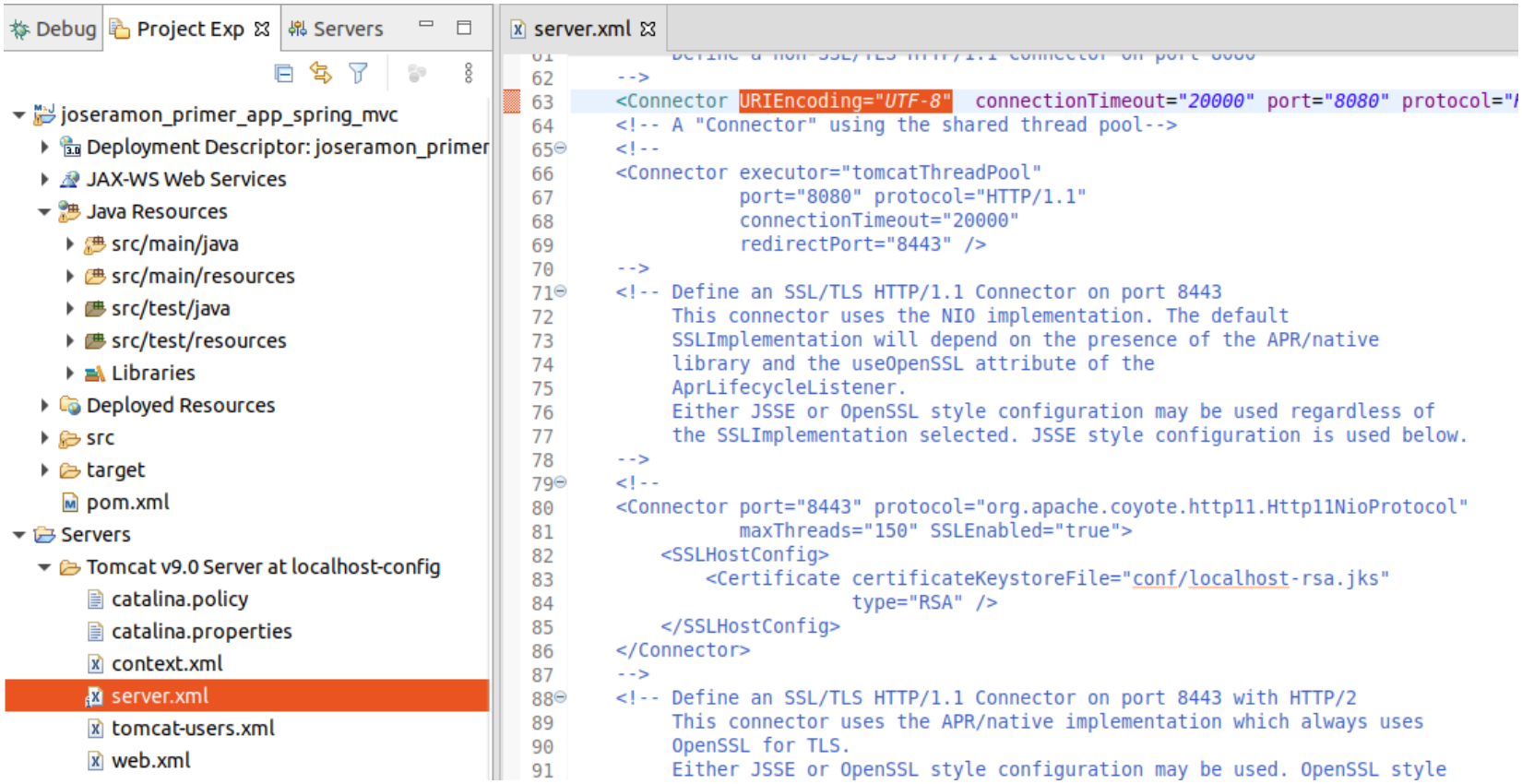
No nos preocuparemos por el momento de como almacenar los datos solo en un idioma...

Ayuda: Los distintos getTraduccion() a implementar pueden llamar al getTraduccion() original que traduce tal cual un solo String.

5º Configurar el encoding de nuestra app como UTF-8:

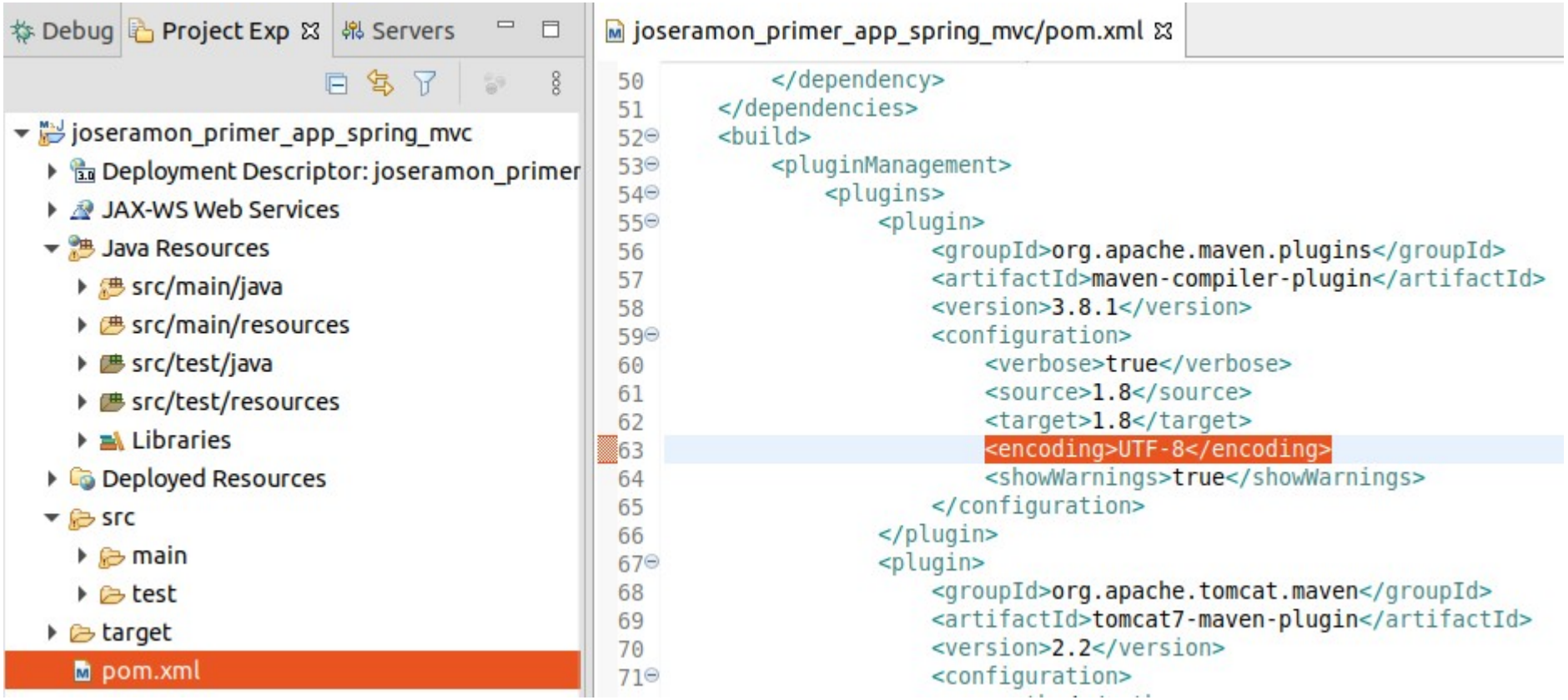
Si no configuramos el encoding podemos tener problemas con acentos, dieresis,etc…

Aunque es **opcional** ,es recomendable empezar por configurar el conector del contenedor de aplicaciones Tomcat con el encoding UTF-8, para ello añadiremos el encoding al conector en el **fichero server.xml** :



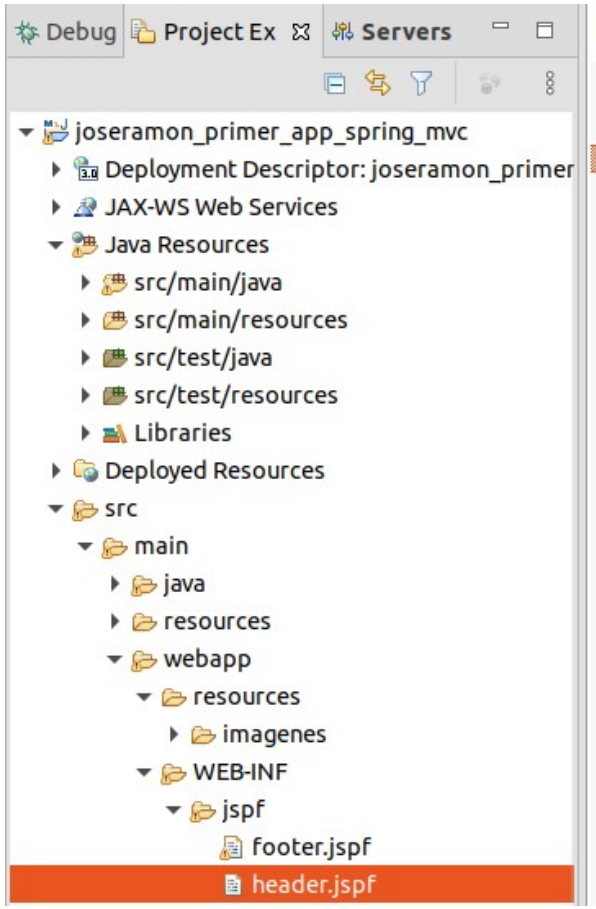
5º Configurar el encoding de nuestra app como UTF-8:

Para continuar configuramos el encoding UTF-8 en el plugin del compilador de maven, para ello añadiremos el encoding en el **fichero pom.xml** :



5º Configurar el encoding de nuestra app como UTF-8:

Por último configuramos el encoding UTF-8 en el fichero header.jspf para que lo tengan todas las páginas:

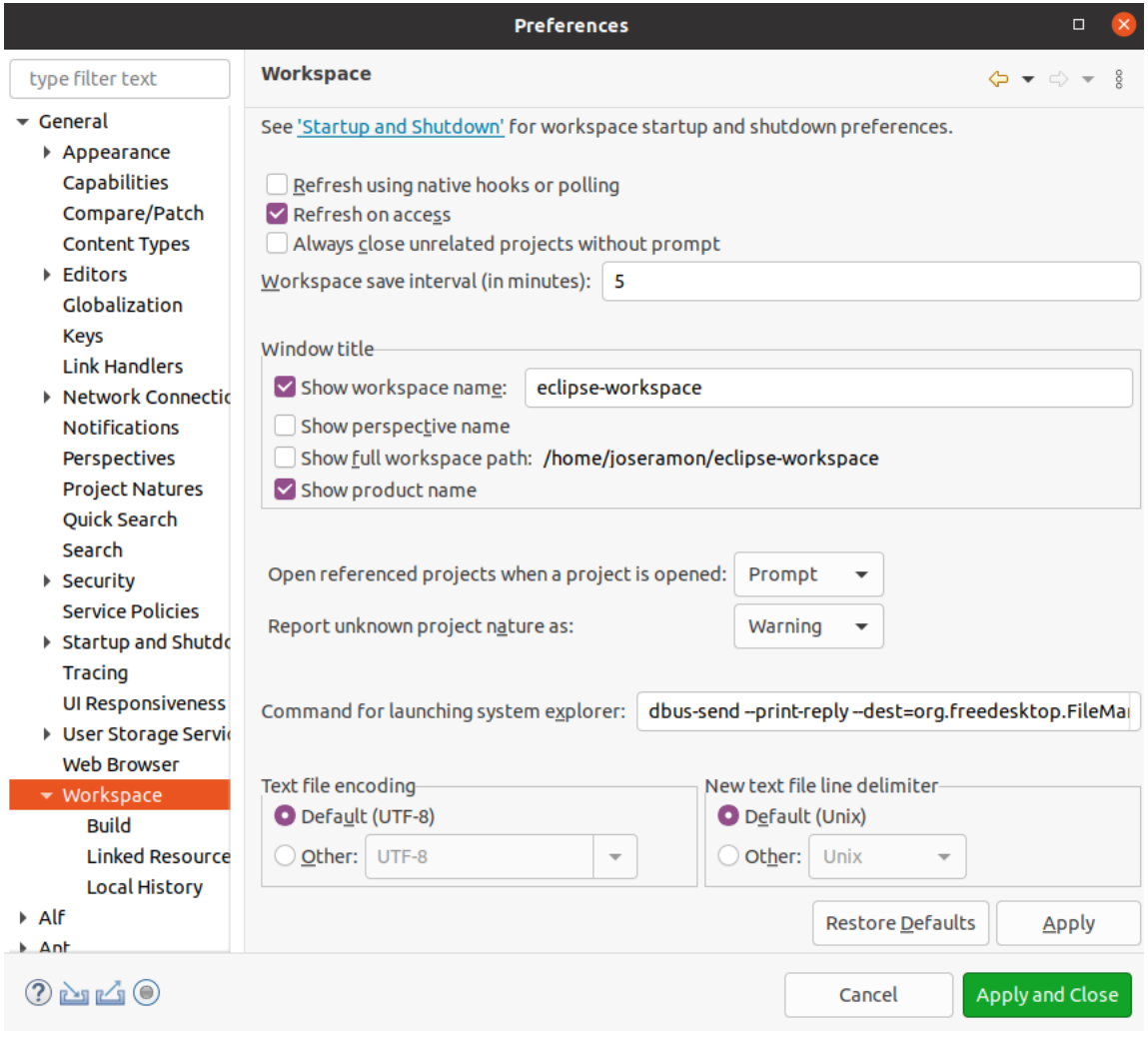


header.jspf

```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
3  <%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
4  <%@page pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>
5  <!DOCTYPE html>
6  <html>
7  <head>
8  <link href="webjars/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
9  <link href="webjars/font-awesome/5.15.1/css/all.min.css" rel="stylesheet">
10 <title>${pagina.getTitulo()}</title>
11 <style>
12     .footer {
13         position: absolute;
14         bottom: 0;
15         width: 100%;
16         height: 60px;
17         background-color: #f5f5f5;
18     }
19 </style>
20 </head>
21 <body>
22
```

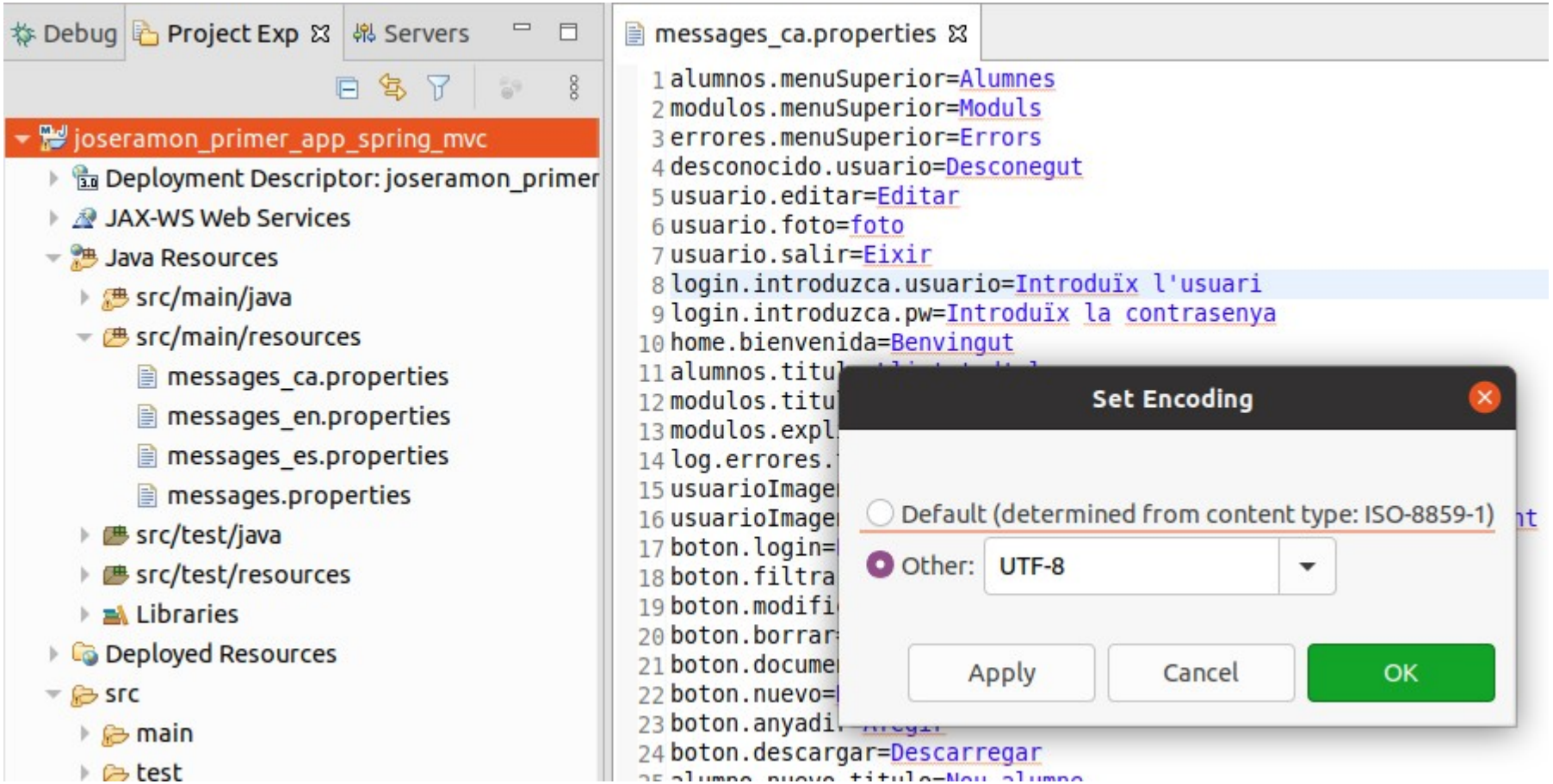
5º Configurar el encoding de nuestra app como UTF-8:

No esta de más que comprobemos que nuestro Workspace tiene por defecto el encoding UTF-8 mediante al menú \Window\Preferences\General\Workspace :



5º Configurar el encoding de nuestra app como UTF-8:

Y si tenemos problemas con los ficheros de configuración les pongamos el encoding UTF-8 mediante al menú \Edit\Set Encoding :



Ojo: Si modificamos un fichero de properties y tenemos un encoding incorrecto podemos destrozar el fichero entero introduciendo caracteres extranos sin querer.

UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

6º Configurar mensajes de las anotaciones de los validadores por defecto:

Para configurar la internacionalización en los mensajes de los validadores tan solo hay que añadir la notación en los ficheros messages.properties de todos los idiomas con el formato **notación.nombreclase.propiedad**. Veamos como se hace en la validación de alumno y luego cambiemos todas las validaciones menos las personalizadas (@ImagenValida, @DocumentoAlumnoValido):

Ejemplo: El nombre del alumno tiene una limitación de tamaño (@Size) expresada en el fichero de properties como 'Size.alumno.nombre'.

```
public class Alumno implements Modificable<Alumno>,Serializable, Comparable<Alumno>{
    private static final long serialVersionUID = 1L;
    @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y una letra")
    private String dni;
    @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
    private String nombre;
    @NotNull(message = "La edad no puede estar vacia")
    @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor o igual a 9
    @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni más de 2 díg
    private Integer edad;
    @Size(min = 3, message = "El ciclo debe tener almenos 3 carácteres")
    private String ciclo;
    @NotNull(message = "El curso no puede estar vacio")
    @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto")
    @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
    private Integer curso;
```

messages.properties

```
65 Espanya=Espana
66 Pattern.alumno.dni=El dni debe tener 8 números y una letra
67 Size.alumno.nombre=El nombre debe de tener un tamaño mínimo de 5 carácteres
68 NotNull.alumno.edad=La edad no puede estar vacia
69 Range.alumno.edad=La edad debe ser igual o mayor a 18 y menor o igual a 99
70 Digits.alumno.edad=La edad no puede tener decimales ni más de 2 dígitos
71 Size.alumno.ciclo=El ciclo debe tener almenos 3 carácteres
72 NotNull.alumno.ciclo=El curso no puede estar vacio
73 Digits.alumno.curso=El curso tiene un formato incorrecto
74 Range.alumno.curso=El curso solo admite los valores 1 o 2
```

messages_en.properties

```
64 Francia=France
65 Espanya=Spain
66 Pattern.alumno.dni=DNI must have 8 numbers and 1 letter
67 Size.alumno.nombre=NAME must have a minimum size of 5 characters
68 NotNull.alumno.edad=AGE cannot be empty
69 Range.alumno.edad=AGE must be equal to or greater than 18 and less
70 Digits.alumno.edad=AGE cannot have decimals or more than 2 digits
71 Size.alumno.ciclo=STUDY must have at least 3 characters
72 NotNull.alumno.ciclo=COURSE cannot be empty
73 Digits.alumno.curso=COURSE has an incorrect format
74 Range.alumno.curso=COURSE only supports values 1 or 2
```

messages_ca.properties

```
64 Francia=França
65 Espanya=Espanya
66 Pattern.alumno.dni=El dni cha de tenir 8 valor numèrics
67 Size.alumno.nombre=El nom ha de tenir un tamany mínim de
68 NotNull.alumno.edad=L'edat no pot estar buida
69 Range.alumno.edad=L'edat cal que siga igual o major a 18
70 Digits.alumno.edad=L'edat no pot tenir decimals ni més d
71 Size.alumno.ciclo=El cicle ha de tenir almenys 3 caràcte
72 NotNull.alumno.ciclo=El curs no pot estar buit
73 Digits.alumno.curso=El curs té un format incorrecte
74 Range.alumno.curso=El curs només admet els valors 1 o 2
```

7º Configurar mensajes de los validadores personalizados:

Veamos en esta y en la siguiente página como hacerlo con “ImagenValida” para castellano y luego el alumno deberá realizar el mismo cambio en todos los idiomas para la interfaz “DocumentoAlumnoValido”:

ImagenValida.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.validaciones;
2  import javax.validation.Constraint;
8
9  // Información sobre como creear una anotación en Java:
10 // https://www.baeldung.com/java-custom-annotation
11
12 @Target(ElementType.FIELD)
13 @Retention(RetentionPolicy.RUNTIME)
14 @Constraint(validatedBy = {ValidadorImagenes.class})
15 public @interface ImagenValida {
16     String message() default "{imagenValida.mensajePorDefecto}";
17     Class<?>[] groups() default {};
18     Class<? extends Payload>[] payload() default {};
19 }
```

messages_es.properties

```
69 Range.alumno.edad=La edad debe ser igual o mayor a 18 y menor o igual a 99
70 Digits.alumno.edad=La edad no puede tener decimales ni más de 2 dígitos
71 Size.alumno.ciclo=El ciclo debe tener almenos 3 carácteres
72 NotNull.alumno.ciclo=El curso no puede estar vacio
73 Digits.alumno.curso=El curso tiene un formato incorrecto
74 Range.alumno.curso=El curso solo admite los valores 1 o 2
75 imagenValida.mensajePorDefecto=Imagen incorrecta
76 imagenValida.vacia=La imagen no puede estar vacia
77 imagenValida.tipoIncorrecto=Solo se permiten imagenes PNG, JPG o GIF.
78 imagenValida.tamanyoIncorrecto=Tamaño máximo de la imagen excedido (524288 bytes)
```

UD 2: Modelo Vista Controlador
11.- Internacionalizacion (i18n)

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

7º Configurar mensajes de los validadores personalizados:

ValidadorImagenes.java

```
24
25      @Override
26      public boolean isValid(MultipartFile multipartFile, ConstraintValidatorContext context) {
27          //Por defecto resultado de la comprobación válido hasta que encontremos un error
28          boolean result = true;
29          //comprobar lista de errores
30          ArrayList<String> listaErrores=
31                  (ArrayList<String>) i18nService.getTraduccion(mensajesErrorImagen(multipartFile));
32          //Si hay errores añadirlos al contexto
33          if (!listaErrores.isEmpty()){
34              context.disableDefaultConstraintViolation();
35              //iteramos por la lista de errores para añadirlos al contexto
36              for(String textoError:listaErrores) {
37                  context.buildConstraintViolationWithTemplate(
38                          textoError)
39                         .addConstraintViolation();
40              }
41              //Comprobación incorrecta (resultado no valido)
42              result = false;
43          }
44          //Devolvemos resultado de la comprobación
45          return result;
46      }
47
48      public static ArrayList<String> mensajesErrorImagen(MultipartFile fichero) {
49
50       ArrayList<String> errores=new ArrayList<String>();
51       //Fichero no vacio
52       if (fichero.isEmpty()) {
53          errores.add("imagenValida.vacia");
54       }
55       //Validar tipo de fichero
56       String contentType = fichero.getContentType();
57       if (!tipoDeImagenValido(contentType)) {
58          errores.add("imagenValida.tipoIncorrecto");
59       }
60       //Comprobar tamaño máximo
61       if (fichero.getSize()>MAX_BYTES) {
62          errores.add("imagenValida.tamanyoIncorrecto");
63       }
64
65       return errores;
66   }
```

UsuarioController.java

```
74      public String guardarImagenUsuario(ModelMap model,
75              @Valid ImagenUsuario imagenUsuario, BindingResult validacion) {
76          //paginaServicio.setPagina(pagina);
77          //model.addAttribute("pagina", paginaServicio.getPagina());
78          if (validacion.hasErrors()) {
79              // Hay errores y debemos volver al formulario de modificación
80              return "update-imagenUsuario";
81          }
82          // Si llega aquí no hay errores de validación
83
84          String nickName=imagenUsuario.getNickname();
85          MultipartFile fichero =imagenUsuario.getImagen();
86          try {
87              if (model.getAttribute("usuario") == null) {
88                  throw new Exception("Para realizar modificaciones debe estar logeado");
89              }
90              //Guardar la imagen y actualizar usuario
91              //Si no se ha podido listaErroresAlGuardar no estará vacio
92              ArrayList<String> listaErroresAlGuardar=fileService.guardaImagenUsuario(ficher
93              if(!listaErroresAlGuardar.isEmpty()) {
94                  //Rellenar los errores al intentar guardar para pasarselos a la excepcion
95                  String mensajeCompleto="";
96                  for(String mensaje:listaErroresAlGuardar) {
97                      mensajeCompleto+=i18nService.getTraduccion(mensaje)+"<br>";
98                  }
99                  //lanzar excepción
```

7º Configurar mensajes de los validadores personalizados:

Ahora vamos a intentar parametrizar valores en las traducciones.

Por ejemplo, si nos hemos fijado en el código de la validación hemos tenido que poner el valor fijo del número de bytes máximo de una imagen en la traducción sin tener opción de mostrar la constante que realmente tiene el valor.

¿ Como podemos traducirlo a todos los idiomas añadiendo un parámetro?

7º Configurar mensajes de los validadores personalizados:

La solución pasa por mostrar un mensaje parametrizado:

messages_ca.properties

```
76 imagenValida.vacia=L'imatge no pot estar buida
77 imagenValida.tipoIncorrecto=Sols estàn permitides imatges PNG, JPG o GIF.
78 imagenValida.tamanyoIncorrecto=Tamany màxim de l'imatge excedit ({0} bytes)
79 documentoAlumnoValido.mensajePorDefecto=Document incorrecte
```

Y llamar al traductor con los parámetros:

ValidadorImagenes.java

```
47
48      public static ArrayList<String> mensajesErrorImagen(MultipartFile fichero) {
49
50       ArrayList<String> errores=new ArrayList<String>();
51       //Fichero no vacio
52       if (fichero.isEmpty()) {
53           errores.add("imagenValida.vacia");
54       }
55       //Validar tipo de fichero
56       String contentType = fichero.getContentType();
57       if (!tipoDeImagenValido(contentType)) {
58           errores.add("imagenValida.tipoIncorrecto");
59       }
60       //Comprobar tamaño máximo
61       if (fichero.getSize()>MAX_BYTES) {
62           errores.add("imagenValida.tamanyoIncorrecto#"+MAX_BYTES);
63       }
64
65       return errores;
66   }
```

7º Configurar mensajes de los validadores personalizados:

Ahora solo nos queda hacer las modificaciones para comprobar si hay parámetros para sustituir los parámetros con sus valores o realizar la traducción de manera normal:

```
//Consulta el idioma configurado en la peticion y devuelve su traducción
private String getI18nMessage(String msg) {
    if (msg.contains("#")) {//hay parámetros y debemos convertir el mensaje en un vector
        ArrayList<String> parametros= new ArrayList(List.of(msg.split("#")));
        //quitamos el 1º que es el msg a buscar su traducción en el fichero de traducciones
        parametros.remove(0);
        //calcular el valor a traducir sin los parámetros
        String valorATraducir=msg.substring(0,msg.indexOf("#"));
        //Podemos traducir el valor
        String traduccion=i18n_mensaje.getMessage(valorATraducir,null,LocaleContextHolder.getLocale());
        //cambiar parametros por lo valores reales
        for(int i=0;i<parametros.size();i++) {
            traduccion=traduccion.replace("{"+i+"}", parametros.get(i));
        }
         return traduccion;
    }else {//no hay parámetros
         return i18n_mensaje.getMessage(msg, null,LocaleContextHolder.getLocale());
    }
}
```

De esta manera si llamamos a `errores.add("imagenValida.tamanyoIncorrecto#"+MAX_BYTES);` lo que hará será traducir la 1º parte (por ejemplo al ingles) con `imagenValida.tamanyoIncorrecto=Maximum image size exceeded ({0} bytes)` y luego sustituirá {0} por ***MAX_BYTES***

8º Configurar iconos para cambiar idioma:

Si queremos mostrar los iconos en la barra superior derecha deberemos de copiar las banderas proporcionadas en el Drive y modificar menusuperior.jspf para añadir siguiente texto como última opción antes de cerrar la etiqueta “nav” final:

```
<div class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-
haspopup="true" aria-expanded="false">
            <img alt="castellano" src="imagenes/${pagina.getIdioma()}.png" >
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" href="${pagina.getPaginaActiva()}?language=es">
              <img alt="castellano" src="imagenes/es.png" > es
           </a>
          <a class="dropdown-item" href="${pagina.getPaginaActiva()}?language=ca">
            <img alt="english" src="imagenes/ca.png" > va
          </a>
           <a class="dropdown-item" href="${pagina.getPaginaActiva()}?language=en">
            <img alt="english" src="imagenes/en.png" > en
          </a>
        </div>
</div>
```

De esta manera mostramos las banderas, pero debemos de configurar la bandera por defecto “pagina.getIdioma()”, como veremos a continuación.

8º Configurar iconos para cambiar idioma:

En el modelo de 'Pagina' debemos añadir un campo "idioma" con sus getter y su setter. Por defecto en el constructor tendrá el valor "es" y deberemos de crear otro constructor para añadir el idioma directamente si nos interesa:

Pagina.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model;
2
3  public class Pagina {
4      private String titulo;
5      private String paginaActiva;
6      private String filtroLogErrores;
7      private String idioma;
8      public Pagina(String titulo, String paginaActiva) {
9          super();
10         this.titulo = titulo;
11         this.paginaActiva = paginaActiva;
12         this.filtroLogErrores="";
13         this.idioma="es";
14     }
15     public Pagina(String titulo, String paginaActiva,String idioma) {
16         super();
17         this.titulo = titulo;
18         this.paginaActiva = paginaActiva;
19         this.idioma=idioma;
20     }
```

8º Configurar iconos para cambiar idioma:

En el servicio I18nService deberemos de tener un método para averiguar el idioma actual del usuario:

I18nService.java

```
114     public String getIdioma() {
115         return LocaleContextHolder.getLocale().getLanguage();
116     }
117 }
```

Y así poder incluirlo en los controladores (solo en los metodos list-alumno,list-modulo y list-logerror que son los que pueden recibir la petición con el language y deben actualizar la bandera del idioma):

LoginController.java

```
32      I18nService i18nService;
33
34      @RequestMapping(value={"/","/login"},method = RequestMethod.GET)
35      public String mostrarLogin(HttpServletRequest request,
36              HttpServletResponse response,
37              Locale locale,ModelMap model) {
38          //i18n:Detectamos idioma de la petición (lo rellena el navegador) y lo
39          //cambiamos si es distinto al valor por defecto (locale).
40          i18nService.configuraIdiomaPeticion(request, response, locale);
41          // Datos para la cabecera de la página
42          model.put("pagina",new Pagina("Home","login",i18nService.getIdioma()));
43          model.put("loginName", "Usuario desconocido");
44          model.put("loginNickName", "Desconocido");
45          //Datos para el formulario de login
46          model.addAttribute(new Usuario());
47          return "login";
48      }
```

AlumnoController.java

```
71      @RequestMapping(value = "/list-alumno", method = RequestMethod.GET)
72      public String listarAlumnos(ModelMap model, @RequestParam(required =
73          pagina.setIdioma(i18nService.getIdioma());
74          paginaServicio.setPagina(pagina);
75          if (orden!=null)
76              model.addAttribute("alumnos", alumnoService.listaAlumnos(orde
77          else
78              model.addAttribute("alumnos", alumnoService.listaAlumnos());
79          //model.put("filtroAlumno", new FiltroAlumno());
80          model.put("filtroAvanzadoAlumno", new FiltroAvanzadoAlumno());
81          model.addAttribute("pagina", paginaServicio.getPagina());
82          return "list-alumno";
83      }
```

REFLEXIÓN FINAL:

Al internacionalizar una web nos damos cuenta de un **problema de diseño** que no vamos a solucionar EN ESTA PRÁCTICA por su elevado costo de modificación.

Cuando almacenamos valores que se quieren tener traducidos deberíamos tenerlos en tablas y tener claves ajenas numéricas que apunten a dicho valor.

Por ejemplo, Al almacenar un 'Alumno' en vez de guardar como String el horario (mañana/tarde) o el género (hombre/mujer) deberiamos de haber guardado valores numéricos.

De tal manera que **podemos utilizar un HashMap<Integer><String> con la versión original a traducir en el campo String**.

Mejor solución aún seria utilizar DTOs ...

EJERCICIO:

Sube la aplicación final al moodle.

Para ello:

1º Haz un "Run As \Maven Clean" para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).

2º Comprime la carpeta de tu aplicación y ponle como nombre al fichero comprimido UD2_practica9_nombreAlumno.tar.gz donde nombreAlumno es el nombre del alumno que entrega la práctica.

3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviesemos Windows, podemos comprimir en ZIP.