UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

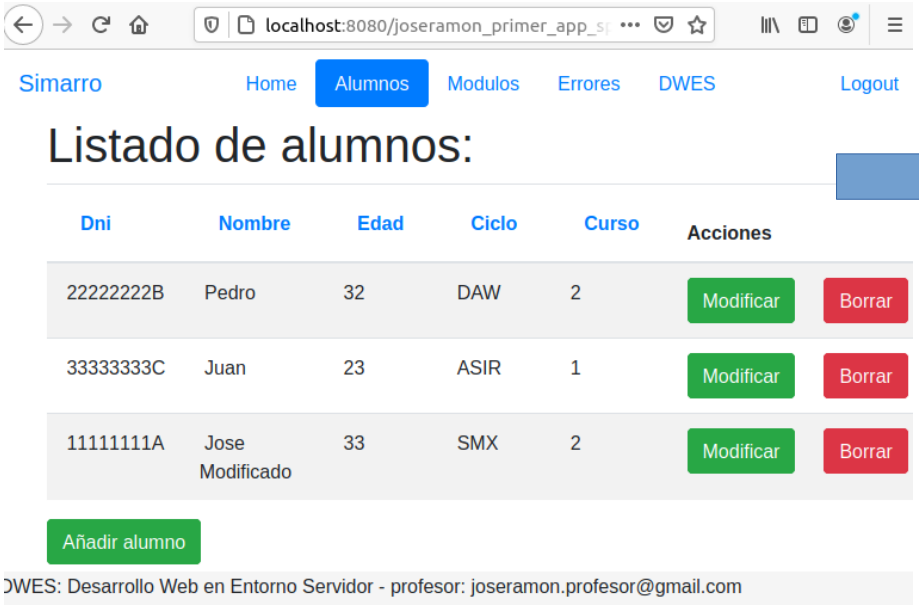
Objetivos de la sesión:

- *Entender porque se borran y se almacenan los datos incorrectamente cuando más de un **usuario modifica de manera concurrente el mismo alumno**.*
- *Aprender a **crear interfaces aplicables a cualquier clase**.*
- *Utilizar una interfaz para **resolver de una manera estandarizada los problemas de concurrencia** ocasionados por las actualizaciones concurrentes.*

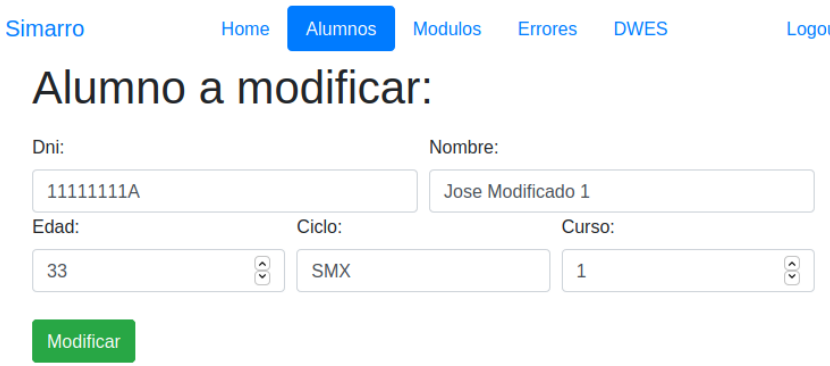
UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

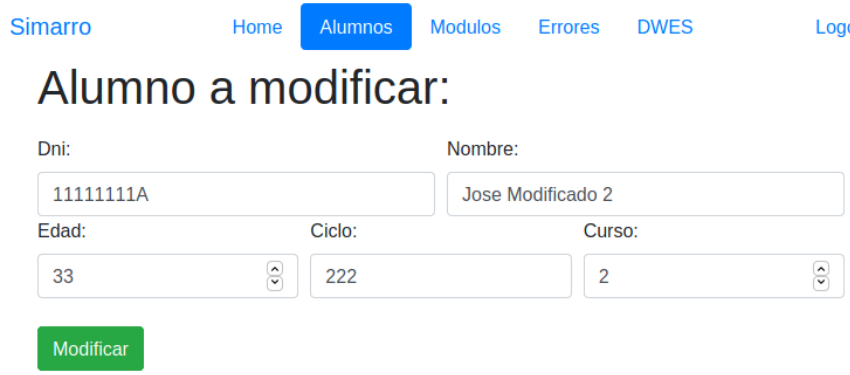
Vamos a simular 2 usuarios que pretenden modificar el mismo alumno a la vez abriendo 2 pestañas de modificación sobre el mismo alumno: Para ello copiar la url de la primera pestaña y pégala en una nueva pestaña



1º Pestaña:Modificamos el nombre y el curso



2º Pestaña: Modificamos el nombre y el ciclo

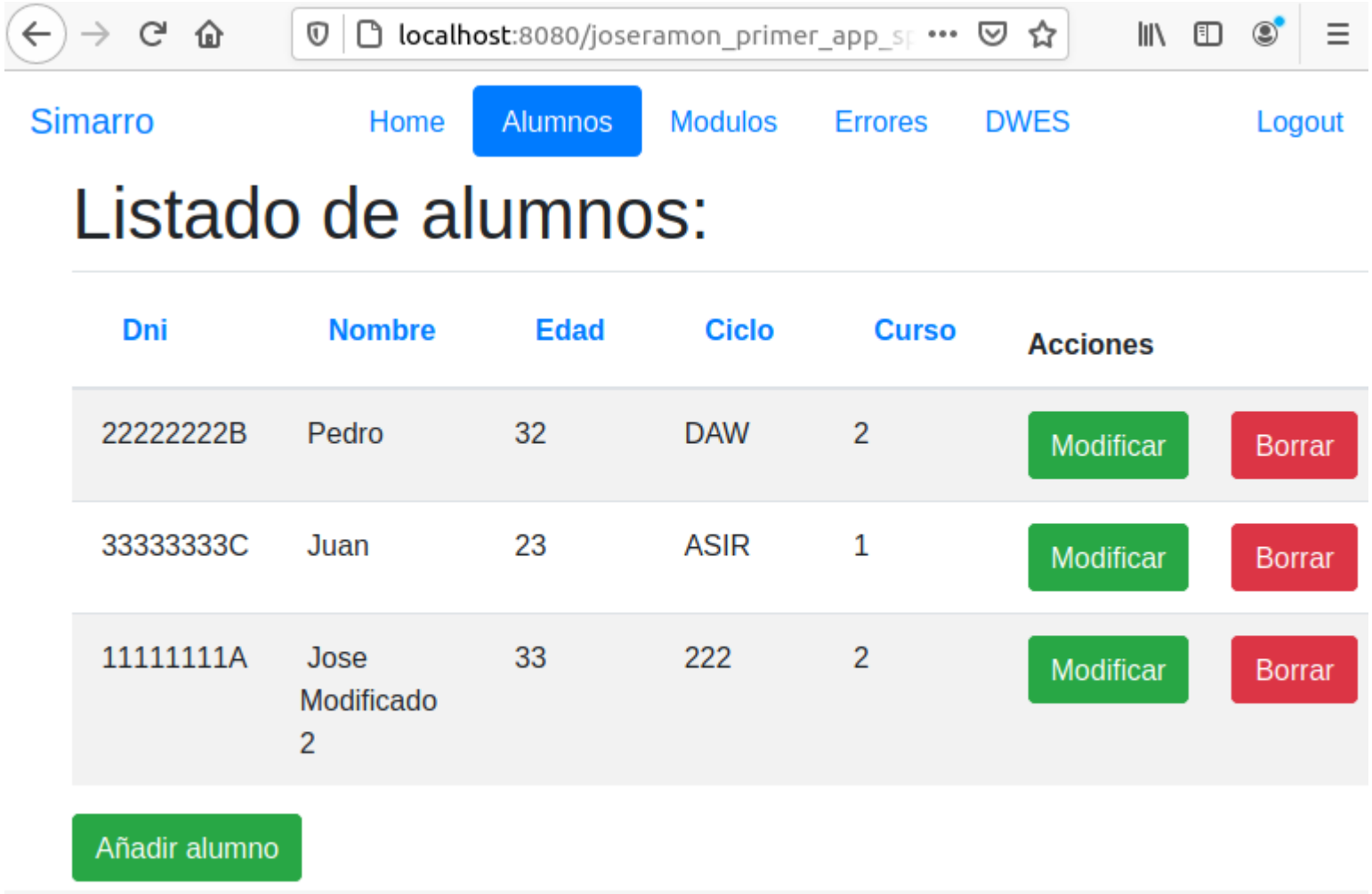


¿Que nombre, ciclo y curso se guardará si pulsamos en “Modificar” en la primera pestaña y luego modificamos en la segunda pestaña?

UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

¿Que ha pasado? ¿Por qué no queda ni rastro de las modificaciones de la primera pestaña? ¿Como podemos solucionarlo?



UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

El control de modificaciones concurrentes es un problema que debemos ser capaces de solucionar en aplicaciones a las que puede acceder más de un usuario a la vez (la mayoría).

¿Por qué se ha producido este problema?

La pestaña 1 se cargó con los datos iniciales del alumno, al igual que la pestaña 2. Sin embargo, cuando se modificaron los datos en la pestaña 1 no se avisó a la pestaña 2 , que tenía los datos antiguos del alumno. Por ello , cuando la pestaña 2 guardó sus cambios se machacó el alumno entero con los datos iniciales del alumno más los cambios de la pestaña 2.

¿Como podemos solucionar este problema de una manera sencilla?

Una manera sencilla de solucionar el problema es añadir ***2 campos nuevos al bean "Alumno" : "ts" y "user"***.

· **"ts":** es una abreviatura de 'TimeStamp', y se utilizará para almacenar la fecha y hora de la última modificación.

· **"user":** almacenará el 'nickname' del usuario que ha hecho la modificación.

¿Como conseguimos solucionar la concurrencia con estos campos?

Si intentamos modificar un alumno y dicho alumno tiene una fecha de modificación o un usuario distinto al actual no permitiremos modificar el alumno.

Si tienen la misma fecha y usuario de modificación modificaremos el alumno y procederemos a actualizar la fecha de modificación y el usuario a la nueva fecha y usuario.

¿Podemos generalizar este comportamiento para reutilizarlo?

Una de las formas de generalizar este comportamiento es **crear de una Interface nueva en el paquete "model.interfaces"** que llamaremos **"Modificable"**.

La mayor complejidad de esta interfaz es que debe de **ser capaz de aplicar el método booleano sobre cualquier clase** y para ello utilizamos **<AnyType>**:

Modificable.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.interfaces;
2
3  import java.util.Date;
4
5  public interface Modificable <AnyType>{
6      //Getters y Setters necesarios en el 'itemActual'
7      public Date getTs();
8
9      public String getUser();
10
11     public void setTs(Date ts);
12
13     public void setUser(String user);
14
15     //Sobre el 'itemActual'
16     //se aplicará el método "sePuedeModificarUtilizando(itemModificado)
17     //para saber si se puede modificar. Si no tenemos los datos
18     // actualizados (fecha y usuario iguales ) nos dirá que no.
19     public boolean sePuedeModificarUtilizando(AnyType itemModificado);
20
21     //Mensaje en caso de que no podamos actualizar el Item
22     public String mensajeNoSePuedeModificar();
23
24 }
```

UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

La clase alumno deberá implementar la interface modificable, para lo cual hay que **crear en Alumno los atributos necesarios (ts y user) , sus getters y setters e implementar los 2 métodos principales de la interface:**

- **sePuedeModificarUtilizando()**
- **mensajeNoSePuedeModificar()**

NOTA: Se ha dejado en el Drive un pequeño “regalito”, la clase “Ts” que sirve para formatear fechas.

Alumno.java

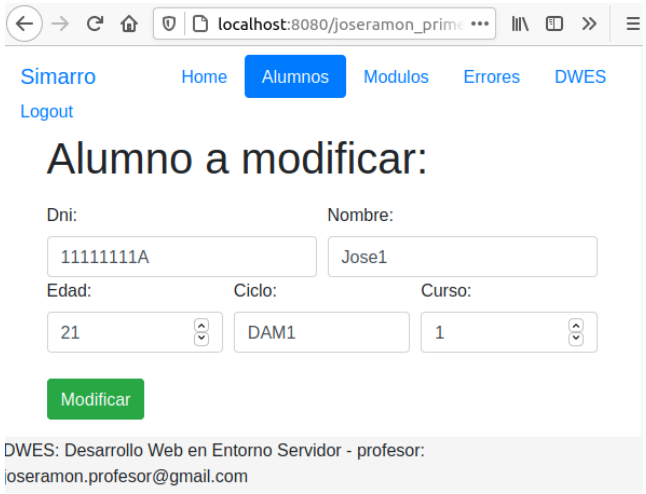
```
144    public boolean sePuedeModificarUtilizando(Alumno itemModificado) {
145        if (this.getUser() != null && this.getTs()!= null) {
146            //Existe un usuario y una fecha Inicial y tenemos que comprobar
147            String usuarioActual = this.getUser();
148            String usuarioModificado = itemModificado.getUser();
149            //formateamos fechas gracias a la clase Ts que formatea fechas
150            Date fechaActual = Ts.parseIso(Ts.formatIso(this.getTs()));
151            Date fechaModificada = Ts.parseIso(Ts.formatIso(itemModificado.getTs()));
152            if (!usuarioActual.equals(usuarioModificado) || !fechaActual.equals(fechaModificada))
153                //El usuario no es el mismo o la fecha cambia
154                return false;
155        }
156        //No tenemos fecha o usuario-> 1º modificación, por lo que se puede modificar
157        return true;
158    }
159
160
161    public String mensajeNoSePuedeModificar() {
162        //Mensaje genérico para poder reutilizarlo
163        String msg="\r\n\t[ERROR]\r\n<br/>" +
164                "\t'$item' ha sido modificado por otro usuario.\r\n<br/>" +
165                "\tPara evitar la pérdida de información se impide guardar '$item'.\r\n<br/>" +
166                "\tÚltima modificación realizada por [" + this.getUser() + "] el [" +
167                Ts.ts(this.getTs()) + "]\r\n<br/>";
168        //Para concretar el tipo de registro modificado sustituimos $item por Alumno
169        return msg.replace("$item", "Alumno");
170    }
171 }
```

UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

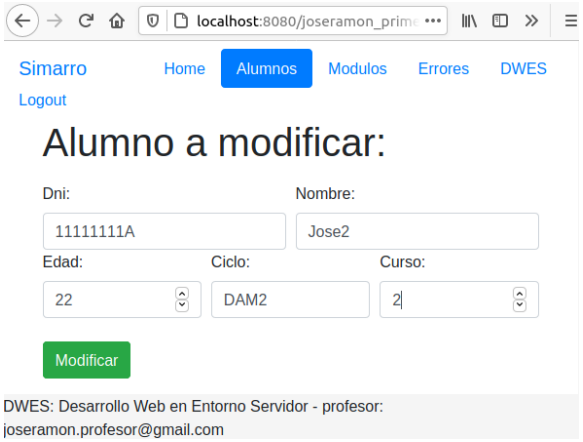
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

En la siguiente diapositiva veremos como implementar este control cuando lo tengamos todo hecho. De manera resumida el problema de concurrencia se produce cuando :

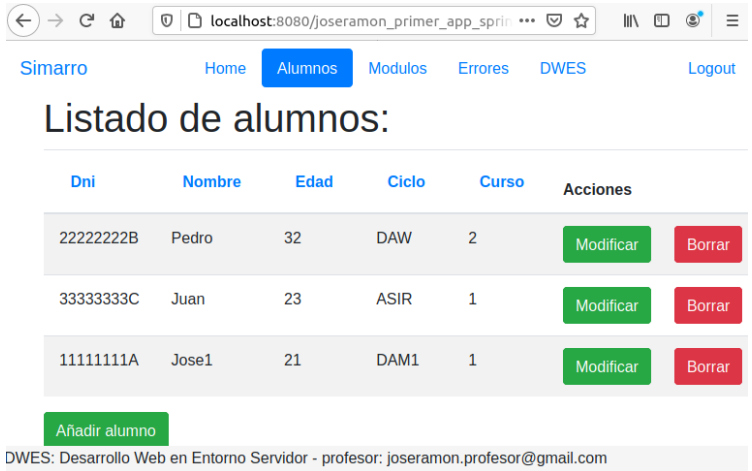
1º Modificamos un alumno



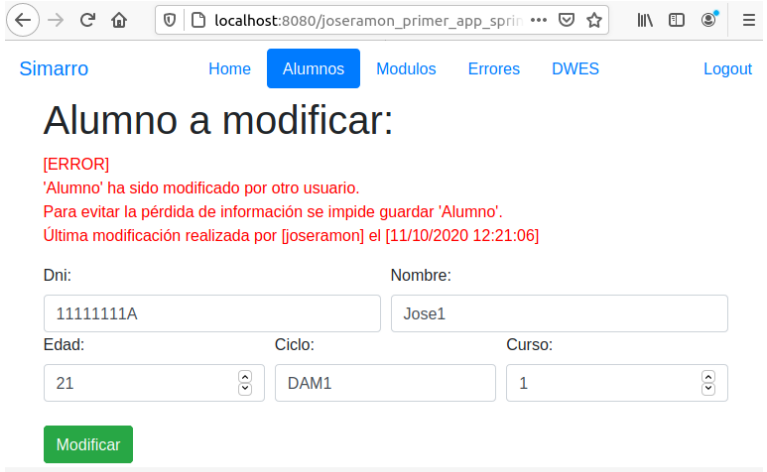
2º Sin haber guardado copiamos el enlace en otra pestaña y lo volvemos a modificar



3º Pulsamos modificar en la 1º pestaña



4º Pulsamos modificar en la 2 pestaña



Para implementar el comportamiento anterior hay que **realizar los siguientes cambios:**

1º AlumnoService.java:

Modificar el método modificarAlumno para que tenga los siguientes parámetros:

```
public void modificaAlumno(Alumno alumnoModificado,String usuarioModificacion) throws Exception
```

De manera resumida el algoritmo a implementar será:

```
if ( // no tenemos los datos del alumno o el usuario de modificación)
   //lanzar excepción con el error
else
   // 1º obtener alumnoActual (el alumno tal cual esta almacenado sin cambios)
   if alumnoActual.sePuedeModificarUtilizando(alumnoModificado)
        //borrar alumnoActual del ArrayList , actualizar el alumnoModificado con
        "usuarioModificación" y la fecha actual y volverlo a añadir al ArrayList
   else
      //lanzar excepcion con el error que viene en
      "alumnoActual.mensajeNoSePuedeModificar()"
```

Es importante controlar que si no estamos logeados no podamos realizar ninguna modificación. Esto se consigue comprobando que usuarioModificación no sea nulo.

2º AlumnoController.java:

Modificaremos el método que atiende el POST de la modificación para que una vez comprobado que no hay errores de validación llamemos al servicio de modificación (modificarAlumno) con los 2 parámetros: el alumno recogido del formulario y el nickname del usuario logeado (que leemos de la sesión).

Si todo va bien redirigiremos al listado, pero si se produce un error deberemos de volver a la pantalla de modificación, informar del error y mostrar el 'Alumno' actualizado, no el alumno con nuestros datos modificados. Esto implica que se perderán los datos de la 2º pestaña al machacarse con los de la 1º pestaña (los actualizados) para que al volver a darle a botón de "Modificar" se pueda realizar la operación de modificación sin errores (porque la fecha y usuario ya son las últimas).

Realiza los cambios anteriores!!

3º update-alumno.jsp:

Modificamos el formulario para que contenga la fecha y el usuario de modificación. Sin estos atributos no podemos hacer la comparación cuando realizamos una petición POST en el controlador. Como la fecha y usuario es una información interna debemos ponerla como campo oculto.

update-alumno.jsp

```
30                      <div class="col">
31                          <mvc:label path="ciclo">Ciclo:</mvc:label>
32                          <mvc:input type="text" id="ciclo" path="ciclo" class="form-control"/>
33                          <mvc:errors path="ciclo" cssClass="text-warning"/>
34                      </div>
35                      <div class="col">
36                          <mvc:label path="curso">Curso:</mvc:label>
37                          <mvc:input type="number" id="curso" path="curso" class="form-control"/>
38                          <mvc:errors path="curso" cssClass="text-warning"/>
39                      </div>
40              </div>
41              <mvc:hidden path="user"/>
42              <mvc:hidden path="ts"/>
43              <br><input type="submit" value="Modificar"  class="btn btn-success">
44          </mvc:form>
45      </div>
46
47  <%@ include file="../jspf/footer.jspf"%>
```

UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

... continuación 3º update-alumno.jsp:

Si vemos el código fuente de la página mostrada en el navegador vemos que el ResolverViewer de Spring traduce los tags del JSP a tags HTML entendibles por el navegador:

¿Podría un usuario con malas intenciones modificar el usuario y/o la fecha de modificación ?

view-source:http://localhost:8080/joseramon_primer

```
<h1>Alumno a modificar:</h1>
<p>
    <font color="red"></font>
</p>

<form id="alumno" action="update-alumno" method="post">
     <div class="form-row">
            <div class="col">
                <label for="dni">Dni:</label>
                <input id="dni" name="dni" type="text" class="form-control" value="11111111A"/>

            </div>
            <div class="col">
                <label for="nombre">Nombre:</label>
                <input id="nombre" name="nombre" type="text" class="form-control" value="Jose1"/>

            </div>
    </div>
    <div class="form-row">
            <div class="col">
                <label for="edad">Edad:</label>
                <input id="edad" name="edad" type="number" class="form-control" value="21"/>

            </div>
            <div class="col">
                <label for="ciclo">Ciclo:</label>
                <input id="ciclo" name="ciclo" type="text" class="form-control" value="DAM"/>

            </div>
            <div class="col">
                <label for="curso">Curso:</label>
                <input id="curso" name="curso" type="number" class="form-control" value="1"/>

            </div>
    </div>
    <input id="user" name="user" type="hidden" value=""/>
    <input id="ts" name="ts" type="hidden" value=""/>
    <br><input type="submit" value="Modificar"  class="btn btn-success">
</form>
</div>

<footer class="footer">
    <p>DWES: Desarrollo Web en Entorno Servidor - profesor: joseramon.profesor@gmail.com</p>
</footer>
```

UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

… continuación 3º update-alumno.jsp:

Pensar las cosas bien desde el Backend es lo que tiene: **Un usuario con malas intenciones no puede hacer nada**. Si se cambia manualmente el usuario o la fecha, aunque tengamos los datos del último 'Alumno' actualizados nos dará error y nos avisará alguien ya ha modificado el 'Alumno'.

Resaltar que por seguridad **SOLO debemos avisar quien modifico por última vez el 'Alumno' si es una aplicación web interna**.

Dependiendo de la aplicación el mensaje puede variar y solo decir que no se puede actualizar el 'Alumno' porque no se cuentan con los datos actualizados.

Aparentemente hemos hecho todos los cambios y debería de funcionar!!

view-source:http://localhost:8080/joseramon_primer

```
<h1>Alumno a modificar:</h1>
<p>
    <font color="red"></font>
</p>

<form id="alumno" action="update-alumno" method="post">
     <div class="form-row">
            <div class="col">
                <label for="dni">Dni:</label>
                <input id="dni" name="dni" type="text" class="form-control" value="11111111A"/>

            </div>
            <div class="col">
                <label for="nombre">Nombre:</label>
                <input id="nombre" name="nombre" type="text" class="form-control" value="Jose1"/>

            </div>
    </div>
    <div class="form-row">
            <div class="col">
                <label for="edad">Edad:</label>
                <input id="edad" name="edad" type="number" class="form-control" value="21"/>

            </div>
            <div class="col">
                <label for="ciclo">Ciclo:</label>
                <input id="ciclo" name="ciclo" type="text" class="form-control" value="DAM"/>

            </div>
            <div class="col">
                <label for="curso">Curso:</label>
                <input id="curso" name="curso" type="number" class="form-control" value="1"/>

            </div>
    </div>
    <input id="user" name="user" type="hidden" value=""/>
    <input id="ts" name="ts" type="hidden" value=""/>
    <br><input type="submit" value="Modificar"  class="btn btn-success">
</form>
</div>

<footer class="footer">
    <p>DWES: Desarrollo Web en Entorno Servidor - profesor: joseramon.profesor@gmail.com</p>
</footer>
```

4º AlumnoController.java:

Si ejecutamos la aplicación e intentamos modificar un alumno pulsando sobre "Modificar" vemos que todavía no funciona.

Si **ponemos un punto de interrupción** para ver que está pasando vemos que hay un error al verificar el bean porque no sabe convertir un tipo 'String' a 'Date':

Failed to convert property value of type 'java.lang.String' to required type 'java.util.Date' for property 'ts'; nested exception is org.springframework.core.convert.ConversionFailedException: Failed to convert from type [java.lang.String] to type [java.util.Date] for value ''; nested exception is java.lang.IllegalArgumentException

AlumnoController.java | update-alumno.jsp

```
85          }
86
87          @RequestMapping(value="/update-alumno",method = RequestMethod.POST)
88          public String procesaUpdateAlumno(ModelMap model, @Valid Alumno alumno, BindingR
89              paginaServicio.setPagina(pagina);
90              model.addAttribute("pagina",paginaServicio.getPagina());
91              if (validacion.hasErrors()) {
92                  //Hay errores y debemos volver al formulario de modificación
93                  return "update-alumno";
94              }
95              //Si llega aquí no hay errores de validación
96              try {
97                  alumnoService.modificaAlumno(alumno,model.getAttribute("nombre").toStrin
98                  //Para evitar pasar parámetros inncesarios
99                  model.clear();
100                 /* Para evitar volver a modificar redirigimos a listar*/
101                 return "redirect:list-alumno";
102             } catch (Exception e) {
103                 //Le pasamos el alumno actualizado
104                 model.addAttribute(alumnoService.encontrarAlumnoPorDni(alumno.getDni()))
105                 //Pasamos los errores
106                 model.addAttribute("errores",e.getMessage());
107                 //Hay errores y debemos volver al formulario de modificación
108                 return "update-alumno";
109             }
110
111         }
112
113         @RequestMapping(value="/del-alumno",method = RequestMethod.GET)
```

Variables | Breakpoints | Expressions

Name	Value
no method return v	
this	AlumnoController (id=75)
model	BindingAwareModelMap (id=76)
alumno	Alumno (id=87)
validacion	BeanPropertyBindingResult (id=91)
autoGrowCollect	256
autoGrowNestec	true
beanWrapper	BeanWrapperImpl (id=124)
conversionServic	DefaultFormattingConversionService (id=135)
errors	ArrayList<E> (id=145)
[0]	FieldError (id=174)
arguments	Object[1] (id=179)
bindingFail	true
codes	String[4] (id=180)
defaultMes	"Failed to convert property value of type 'java.lang.String' to require
field	"ts" (id=183)
objectNam	"alumno" (id=167)

<Choose a previously entered expression>

Failed to convert property value of type 'java.lang.String' to required type 'ja

… continuación 4º AlumnoController.java:

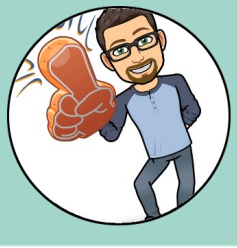
Para resolver el problema aparece en acción una nueva anotación @InitBinder. Debemos crear un método protegido que contenga un parámetro de tipo WebDataBinder que nos permita indicar como tratar los datos de tipo fecha:

AlumnoController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.mvc;
2
3  import java.text.SimpleDateFormat;
25
26 @Controller
27 @SessionAttributes("nombre")
28 public class AlumnoController {
29         Pagina pagina= new Pagina("Alumnos","list-alumno");
30         @Autowired
31         AlumnoService alumnoService;
32         @Autowired
33         PaginaService paginaServicio;
34
35         @InitBinder
36         protected void initBinder(WebDataBinder binder) {
37             SimpleDateFormat dateFormat= new SimpleDateFormat("dd/MM/yy HH:mm:ss");
38             binder.registerCustomEditor(Date.class,new CustomDateEditor(dateFormat,true));
39         }
40
41
42         @RequestMapping(value="/list-alumno",method = RequestMethod.GET)
43         public String listarAlumnos(ModelMap model) {
```

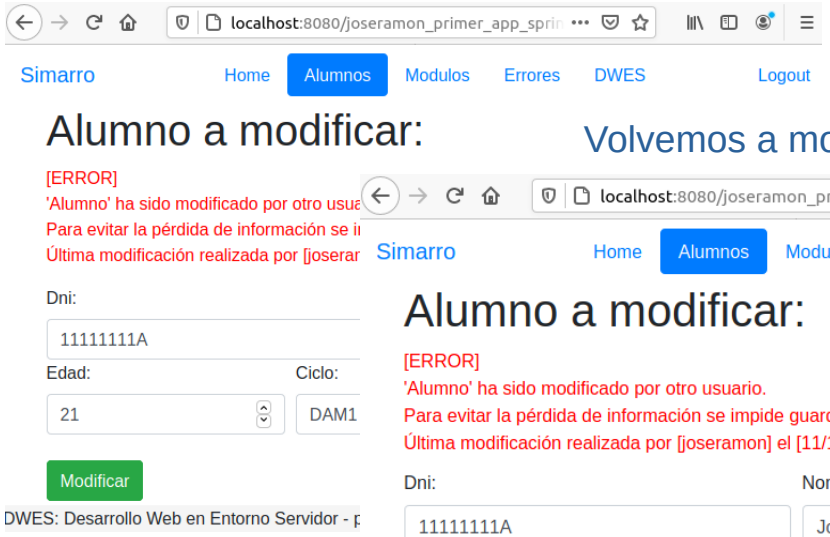
UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

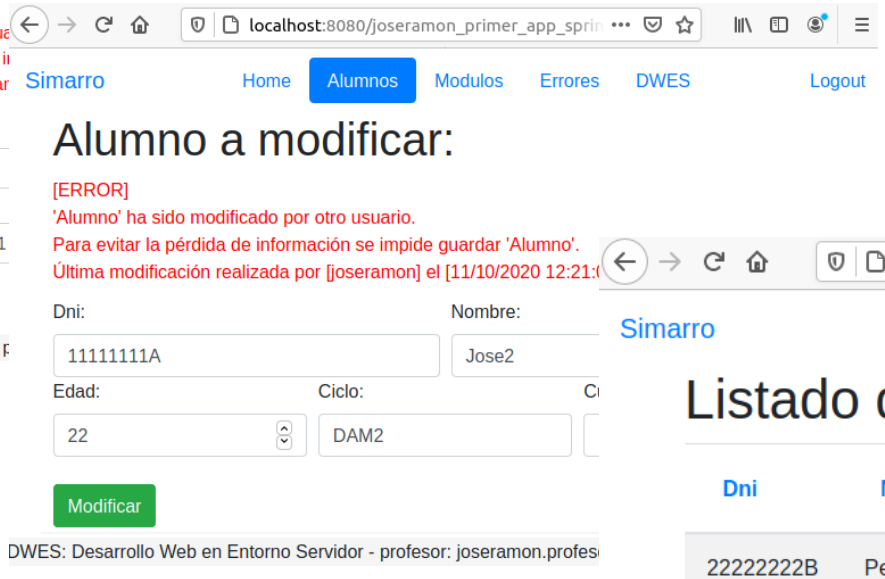


Si todo ha ido bien podremos comprobar que al modificar de nuevo el formulario ya nos guarda los datos:

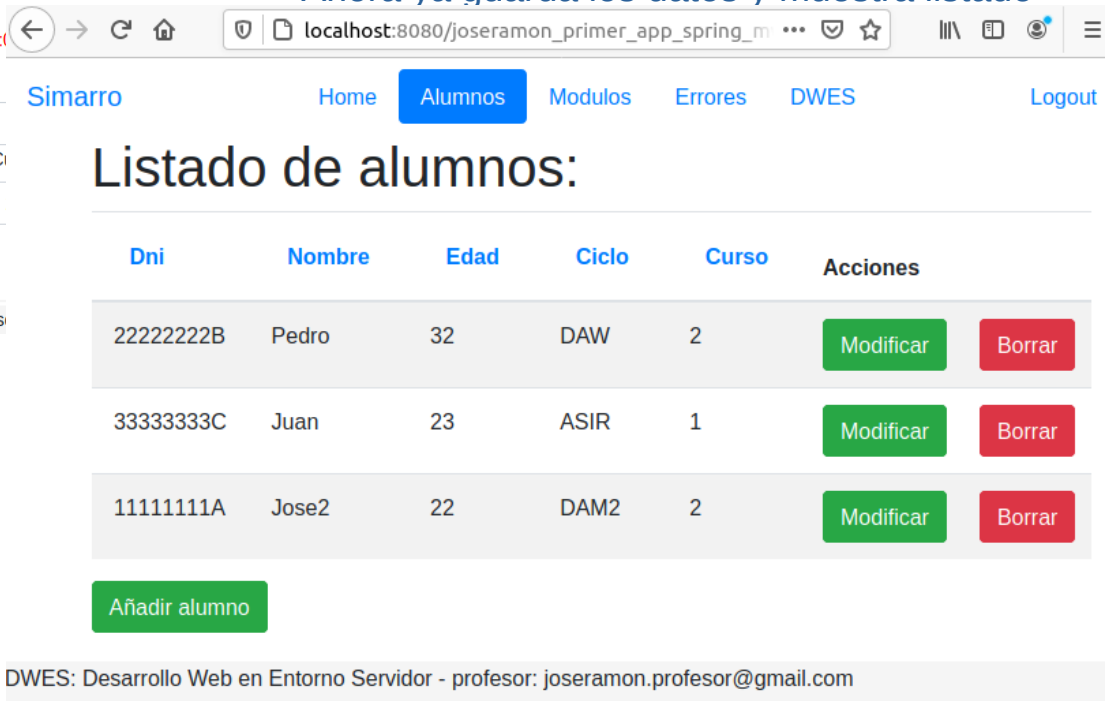
Pantalla inicial que nos muestra el error



Volvemos a modificar los datos y pulsamos en “Modificar”



Ahora ya guarda los datos y muestra listado



UD 2: Modelo Vista Controlador
6.- Formularios: Control de modificaciones

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

EJERCICIO:

Sigue todos los pasos de los PDF**.** 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_practica4_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.