

PRÁCTICA TRONCAL (versión 2)

Este documento describe la práctica troncal de la asignatura de Ingeniería del Software III que deberán realizar los alumnos, preferentemente en parejas. La práctica completa está valorada con dos puntos y deberá ser entregada a los profesores de la asignatura en el mes de Enero. Ésta será evaluada mediante un examen práctico, que deberá ser resuelto individualmente, valorado a su vez en otros cuatro puntos, y que consistirá en la realización de una modificación sobre la práctica de cada alumno.

Este documento se corresponde con la versión 2, ya que describe la segunda fase o hito de la práctica: el desarrollo de las capas de control y presentación de una aplicación web de gestión docente mediante la framework Struts 2.

1. CAPAS DE CONTROL Y PRESENTACIÓN

Toda aplicación web bien diseñada actualmente ofrece normalmente una arquitectura basada en el patrón de diseño Modelo-Vista-Controlador (MVC). En la versión 1 de esta práctica abordamos la capa del modelo a través de la framework Hibernate, apoyada en una BBDD MySQL. En esta segunda versión adoptamos la framework Struts 2 (<http://struts.apache.org/2.x/>) que nos ofrece soporte para los otros dos pilares de MVC, esto es, el Controlador y las Vistas. Esta segunda versión tiene una doble misión:

- Definir e implementar el conjunto de acciones de control requeridas para la gestión de una entidad docente.
- Diseñar e implementar las vistas, es decir, el conjunto de JSPs sobre los que se volcarán los resultados generados por las acciones de control al consultar el modelo de datos que se implementó en Hibernate.

1.1. Arquitectura

La Figura 1 muestra el funcionamiento típico de una aplicación web siguiendo el Model 2 o patrón MVC. Ésta es exactamente la arquitectura interna de nuestra aplicación de gestión docente, a la que vamos a denominar **GestorUD**, a partir de ahora. En la versión 1 ya implementamos la capa de Modelo. En esta versión debemos implementar las dos capas restantes:

- **Capa de control** – para ello vamos a tener que modificar el fichero provisto con el armazón de la práctica **conf/struts.xml**. Este fichero como se vio en la teoría tiene por objeto configurar los recursos que componen una aplicación web en Struts 2: Acciones, Resultados e Interceptores. Nosotros nos vamos a limitar a definir puentes entre los dos primeros introduciendo nuevas entradas en este fichero.

Además, en esta capa de control implementaremos el código de las acciones que den respuesta a las peticiones entrantes y que serán declaradas en el fichero `struts.xml` como nuevos elementos `<action>`.

- **Capa de vista** – consistirá en el conjunto de JSPs a los que las acciones definidas en la capa de control, con la ayuda del fichero de configuración struts.xml, deleguen tras haber consultado el modelo y obtenido la información lista para ser volcada en los Resultados, es decir, JSPs.

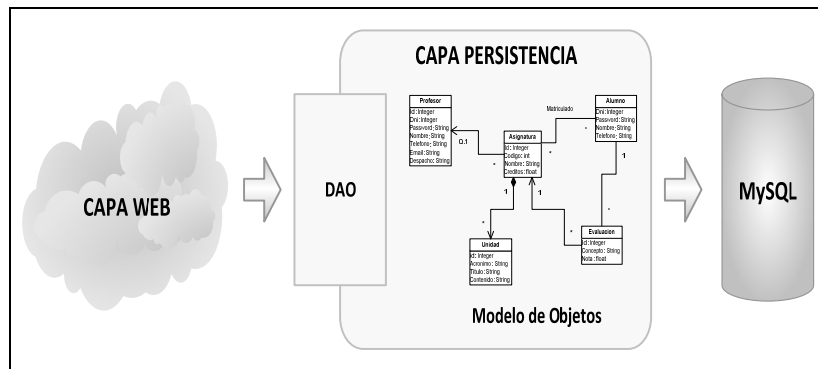


Figura 1: Estructura de capas de la aplicación web GestorUD.

1.2. Estructura de la práctica

Para facilitar la labor de esta segunda fase del desarrollo de la práctica se facilita a los alumnos un archivo denominado **gestorUD.zip** con la siguiente estructura de directorios (ver Figura 2), sobre la que basar sus desarrollos:

- **build.xml** – fichero que inicialmente se proporciona prácticamente vacío y en el que los alumnos deberán incluir las tareas Ant necesarias para compilar el código de la práctica, generar un archivo .war a partir de él y desplegar tal fichero en el directorio webapps de Tomcat. Se proporciona una tarea de ayuda (help) que servirá de pista al alumno para saber qué otras tareas se espera que sean incluidas. La ejecución de este fichero requerirá la definición por parte del usuario de la variable de entorno TOMCAT_HOME, apuntando al directorio de instalación de Tomcat.
- **bin** – directorio creado automáticamente por Ant donde se despliegan los .class de las clases acción y del modelo de la aplicación.
- **conf** – directorio que contiene todos los ficheros de configuración de la práctica mostrados en la Figura 2. Realmente, el alumno solamente tiene que modificar los ficheros gestorUDXXX.properties conteniendo recursos multilingües y el fichero de configuración de Struts 2, struts.xml. El fichero de configuración de hibernate debería copiar/pegarse de la versión 1 de la práctica.
- **css** – contiene el fichero main.css con la hoja estilo que se puede utilizar para el desarrollo de la práctica (opcional).

- **dist** – directorio creado automáticamente por Ant donde se crea el fichero .war que contiene toda la aplicación web y que es desplegado a Tomcat.
- **html** – directorio donde se pueden colocar los ficheros html de la aplicación si los hubiera. Se ha colocado un fichero index.html que básicamente delega a la acción showLogin.action. Por tanto, se recomienda crear una acción con tal nombre en vuestro fichero struts.xml (ver Figura 3).
- **jsp** – directorio donde colocar el conjunto de JSPs que constituyen los resultados de las acciones ejecutadas.
- **lib** – directorio que contiene todas las librerías a utilizar en el desarrollo de la aplicación gestorUD, necesarias tanto para su compilación como ejecución.
- **sql** – directorio inicialmente vacío y que está destinado a contener los script SQL que permitan crear la estructura de vuestra base de datos (**ptDB**), poblar las tablas con datos y asignarla usuarios (**eside/eside**). Se recomienda al alumno preparar estos script (es muy simple hacerlo una vez creada la base de datos a través de la herramienta de backup de MySQL) de cara a facilitar en el examen la puesta en marcha rápida de la práctica.
- **src** – tal como se muestra en la Figura 2, contiene la jerarquía de paquetes que compondrá el código de gestorUD. Los subdirectorios **dao** y **model** contendrán el código generado en la versión 1 de la práctica. El subdirectorio **action** contendrá todas las clases que han de generarse para implementar las acciones Struts 2 de la práctica. Por su parte, el subdirectorio **service** contiene la interfaz PtService y la clase PtDAOService que la implementa. Ambas entidades constituyen la interfaz con el modelo de datos utilizada por las clases acción. Tal interfaz abstrae a las clases acción del mecanismo de serialización utilizado, en este caso, Hibernate.

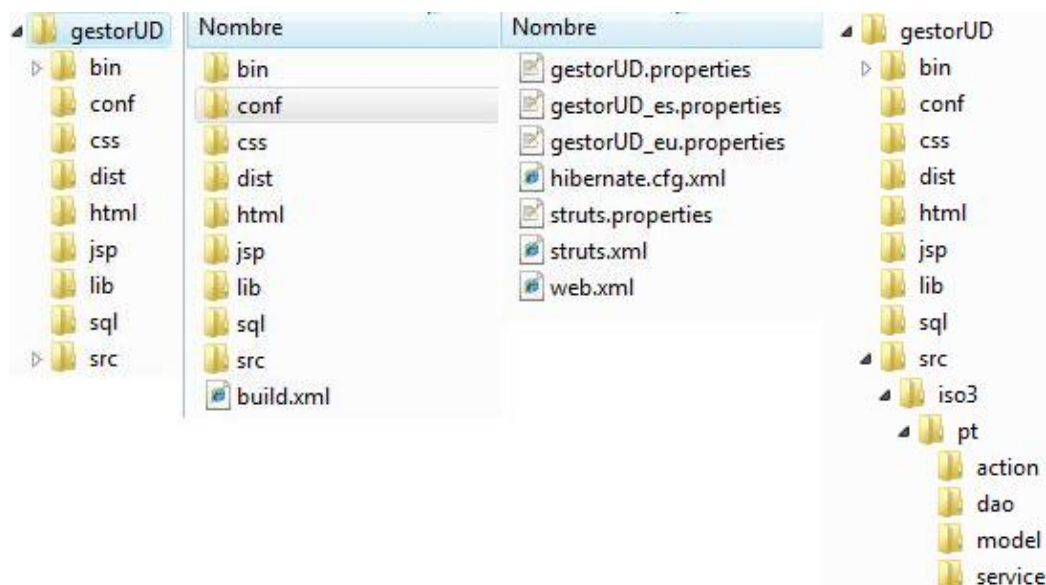


Figura 2: Estructura de directorios de la práctica gestorUD.

1.3. Modelo de desarrollo en Struts 2

Incorporar nueva funcionalidad a una aplicación web desarrollada mediante Struts 2 implica la realización de la siguiente secuencia de pasos:

1. **Creación de una clase de Acción** – que implementa opcionalmente la interfaz `com.opensymphony.xwork2.Preparable` y, por conveniencia más que obligación, normalmente hereda de `com.opensymphony.xwork2.ActionSupport`. La interfaz **Preparable** supone que la clase Acción ha de implementar el método `prepare()` que es invocado antes de llamar al método de ejecución de la acción, para que prepare los datos con los que trabajará la lógica de negocio implementada en el método `execute()` por defecto, o cualquier otro método con la sintaxis `doXXX()`. La clase `ActionSupport` ofrece una implementación por defecto de una acción y contiene métodos de utilidad tales como: `addActionError()`, `getText()` o `validate()`. Consultar la API Struts 2 para más detalles: <http://struts.apache.org/2.0.6/struts2-core/apidocs/>.
2. **Creación de un Resultado de acción en forma de un JSP** – creado mediante etiquetas personalizadas Struts2, cuyo cometido es volcar los objetos creados por la ejecución de un método de una acción en una página de marcado HTML que es devuelta al usuario que inició mediante una petición HTTP cursada por su navegador la ejecución de tal acción. Es muy común que una acción contenga varios métodos de ejecución, cada uno de los cuales puede derivar en un Resultado, es decir JSP, diferente.
3. **Edición de una nueva entrada <action> en el fichero struts.xml** – que contiene los metadatos de la acción definida, es decir, la clase que lo implementa, el conjunto de resultados a los que puede dar lugar o la pila de interceptores que se aplicarán antes y después de la ejecución de la acción. La Figura 3 muestra en negrita la declaración de la acción denominada `LoginAction` que permita controlar el logeo tanto de estudiantes como profesores y que podría utilizarse en la implementación de la práctica. Se ha incluido también una segunda acción denominada `showLogin` cuyo propósito es evitar que se ejecute la lógica de validación asociada `LoginAction` la primera vez que el usuario accede a la aplicación.
4. **Insertión de un conjunto de nuevas entradas en los ficheros de recursos** – por cada idioma soportado. El mismo conjunto de entradas usando las mismas claves pero diferentes valores deben figurar en tales ficheros de recursos. Tales claves serán referenciadas en los JSPs de resultados a través de la etiqueta personalizada `s:text` principalmente. Los ficheros de recursos para la práctica reciben los nombres `gestorUD.properties`, `gestorUD_es.properties` y `gestorUD_eu.properties` y se encuentran localizados en la carpeta `conf` del proyecto. Es obligatorio que el código de la práctica soporte tanto inglés como al menos castellano o euskera a elección de los alumnos.

5. **[OPCIONAL] Creación de un fichero <ClaseAction>-validation.xml** – por cada formulario de un resultado sobre el que se quiera efectuar validación declarativa de datos, soportada por Struts 2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- Configuration for the default package. -->
    <package name="default" extends="struts-default">
        <!-- Default interceptor stack. -->
        <default-interceptor-ref name="paramsPrepareParamsStack"/>
        <action name="showLogin">
            <result>/jsp/Login.jsp</result>
        </action>
        <action name="login" class="iso3.pt.action.LoginAction" method="login">
            <result name="success">/jsp/Login.jsp</result>
            <result name="input">/jsp/Login.jsp</result>
            <result name="error">/jsp/error.jsp</result>
            <result name="listStudentSubjects">/jsp/StudentSubjects.jsp</result>
            <result name="listLecturerSubjects">/jsp/LecturerSubjects.jsp</result>
        </action>
        ...
    </package>
</struts>
```

Figura 3: Declaración de una acción en el fichero struts.xml.

1.4. Modelo de acciones

La aplicación va a estar dividida en distintos bloques funcionales que tendrán una traducción más o menos directa en acciones. Se sugiere, aunque no obliga, a los alumnos a utilizar el siguiente conjunto de acciones en la implementación de la práctica. Este conjunto de acciones es sólo orientativo y ha sido utilizado por los profesores en la implementación que ellos han realizado. Los nombres de las acciones de nuevo son orientativos:

1. **Acciones de logeo** (iso3.pt.action.LoginAction) – se va a distinguir entre logeo realizado por profesores y logeo realizado por alumnos. El motivo es que el flujo de la aplicación será diferente en cada caso:

- **Logeo como profesor** – una vez logeado el profesor (ver Figura 4) verá un listado de las asignaturas que imparte (ver Figura 5). A continuación, sobre una asignatura impartida podrá ver los estudiantes matriculados a la misma (ver Figura 6). Finalmente, eligiendo un alumno del listado de alumnos matriculados podrá añadir una nota (ver Figura 7) y obtendrá como resultado las notas de ese alumno en tal asignatura (ver Figura 8).
 - **Logeo como alumno** – una vez logeado el alumno (ver Figura 9) verá el listado de las asignaturas en las que está matriculado (ver Figura 10). En esta pantalla podrá: a) matricularse a nuevas asignaturas (ver Figura 11), b) mostrar todas las notas de todas las asignaturas en las que está matriculado (ver Figura 12), c) hacer logout volviendo a la pantalla de logeo (ver Figura 9), d) des-matricularse de una asignatura dada, volviendo a mostrar la pantalla de la Figura 10 y e) ver las notas de una asignatura dada (ver Figura 8).
2. **Acciones de profesor** (iso3.pt.action.LecturerAction) – su cometido es generar el listado de asignaturas impartidas por un profesor y el listado de alumnos matriculados en una asignatura.
 3. **Acciones de estudiantes** (iso3.pt.action.StudentAction) – su cometido es generar el listado de las asignaturas en las que está matriculado un estudiante y permitir su matriculación en nuevas asignaturas.
 4. **Acciones de asignatura** (iso3.pt.action.SubjectAction) – su cometido es coordinar las acciones de control asociadas a asignaturas: a) generar el listado de unidades de una asignatura, b) generar el listado de las notas de un alumno en una asignatura, c) generar el listado de todas las notas de un alumno en todas sus asignaturas.
 5. **Acciones de evaluación** (iso3.pt.action.SubjectMarkingAction) – su cometido es coordinar las acciones de control asociadas a la evaluación de asignaturas. Concretamente ha de procesar la solicitud de un profesor que a través de un formulario introduce una nota para una asignatura de un estudiante.

ATENCIÓN: Se quiere volver a hacer hincapié en que el alumno podrá elegir si lo desea otra estrategia de reparto/correspondencia de funcionalidades en acciones, y que no necesariamente tiene que ser peor que la propuesta.

Por ejemplo, pudiera optarse por un enfoque con un mayor grado de descomposición en acciones, donde en lugar de tener unas pocas acciones con varios métodos cada una para implementar un conjunto de funcionalidades, se empleen más acciones, con menos métodos cada una, y donde cada acción soporte una o a lo sumo unas pocas funcionalidades. Con este enfoque tendríamos una acción responsable sólo de sacar el listado de asignaturas de un estudiante, otra distinta para permitir la matriculación en nuevas asignaturas, etc., y no ambas funcionalidades incluidas en una única acción.

En la implementación de todas estas acciones se seguirá un proceso muy similar. A continuación, a modo indicativo, se detalla qué métodos habría que implementar para desarrollar la acción LoginAction:

- **doLogin** – para comprobar si los datos de logeo suministrados son correctos, apoyándonos en la clase PtService se consultará la capa de modelo a través del método loginAlumno (para un estudiante) o loginProfesor (para un profesor). En tal caso simplemente se delegará a la vista que muestra bien las asignaturas del estudiante o del profesor, dependiendo del tipo de logeo realizado. Delegar en el contexto de un método de una acción significa devolver un string que coincida con el nombre de un resultado declarado para una acción en struts.xml. Por ejemplo, SUCCESS, INPUT o “listLecturerSubjects”. Una vez logeado el estudiante o el profesor, sus información deberá almacenarse en un objeto de tipo HttpSession (**ActionContext.getContext().getSession()**) ya que posteriormente habrá ocasiones en las que nos será útil conocer quién se encuentra logeado. Como ya sabemos, tanto los datos de entrada a la acción como los de salida a utilizar por los JSP que generarán el resultado serán inicializados y obtenidos a través de sendos métodos setter y getter, que se mencionan después.
- **doLogout** – eliminará al usuario logeado actualmente del objeto HttpSession y presentará al usuario con el formulario de logeo de nuevo. Para presentar una página como resultado solamente es necesario devolver el string que identifica a un resultado de una acción en struts.xml.
- **prepare** – Básicamente, a través de este método se “prepara” o alisa el camino para la acción. Es decir, a partir de un atributo o un conjunto de atributos de un objeto, mediante este método Struts 2 permite recuperar las instancias de objetos asociados a tales atributos. Esos objetos recuperados normalmente nutren la lógica del método de acción. Es muy probable que dependiendo de la implementación concreta de LoginAction que hagáis quizás no necesitéis escribir nada en este método. Sin embargo, es seguro que en la implementación de otras acciones deberéis hacer uso del mismo.
- **getUsername, setUsername, getPassword, setPassword, getUserType, setUserType, getStudent, setStudent, getLecturer, setLecturer** – son algunos de los métodos setter o getter que pueden definirse para permitir el intercambio de información entre los JSPs y las acciones. Recuerda que al recibir una petición HTTP Struts 2 realiza invocaciones a método setXXX() con el mismo nombre que los parámetros recibidos. Además, los JSPs de resultados ejecutan métodos getXXX() cada vez que se requieren recuperar datos generados por la acción. De hecho, en los JSPs no veremos directamente las invocaciones a métodos getXXX(), sino atributos XXX declarados dentro de etiquetas personalizadas de Struts 2 que los invocarán por nosotros.

1.5. Modelo de vistas

Los resultados generados por las acciones descritas en el punto anterior darán lugar a los siguientes JSPs, a colocar en la carpeta de la práctica **jsp**, que orientativamente podrían tener los siguientes nombres:

1. **Login.jsp** – contendrá un formulario similar al de la Figura 4 compuesto por una caja de texto para el nombre de usuario (s:textfield), otra caja de texto para la contraseña (s:password), una caja de selección de tipo de logeo, es decir, como profesor o alumno (s:select) y un botón de entrega de formulario (s:submit).

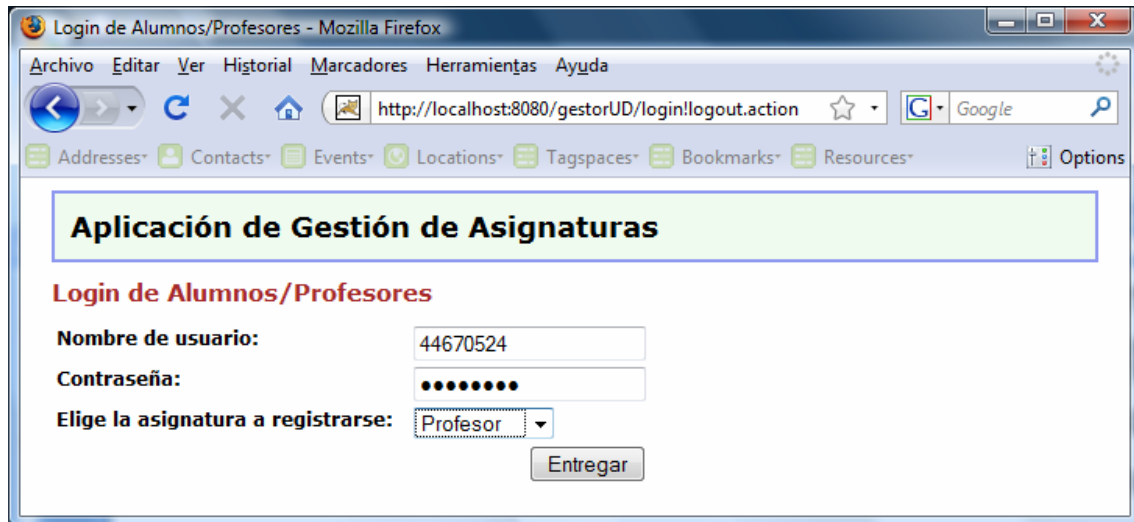


Figura 4: Pantalla de logeo de un profesor en GestorUD.

2. **LecturerSubjects.jsp** – generará una tabla en la que se muestren los detalles de las asignaturas impartidas por un profesor tal como la mostrada en Figura 5. Las siguientes etiquetas personalizadas de Struts 2 deberían ayudarlos en la generación de estas pantallas.

- **s:text**: vuelca el contenido de una variable definida en un fichero de recursos multilingüe al stream de salida de un JSP. Por ejemplo: `<s:text name="application.title"/>`
- **s:url**: ofrece un modo de sencillo y elegante de generar una URL a partir de nombres de acción. Por ejemplo:

```
<s:url action="subject!!listUnits" id="urlSubject" escapeAmp="false">
  <s:param name="subject.id" value="id"/>
</s:url>
```

- **s:property**: se utiliza para volcar el contenido de una variable dentro de un atributo HTML. Por ejemplo:

```
<s:url id="urlLogout" action="login!logout" escapeAmp="false"/>
<a href="<s:property value="#urlLogout"/>">
```


- **s:iterator**: permite la iteración sobre una colección, donde la colección es obtenida a través de un método en la forma `get<NombreColección>()` definido en la acción asociada. Por ejemplo: `<s:iterator value="lecturerSubjects" status="status">`. Para obtener el contenido de los atributos asociados a los objetos que conforman una colección se usa otra vez el tag `s:property`. Por ejemplo: `<s:property value="codigo"/>`.

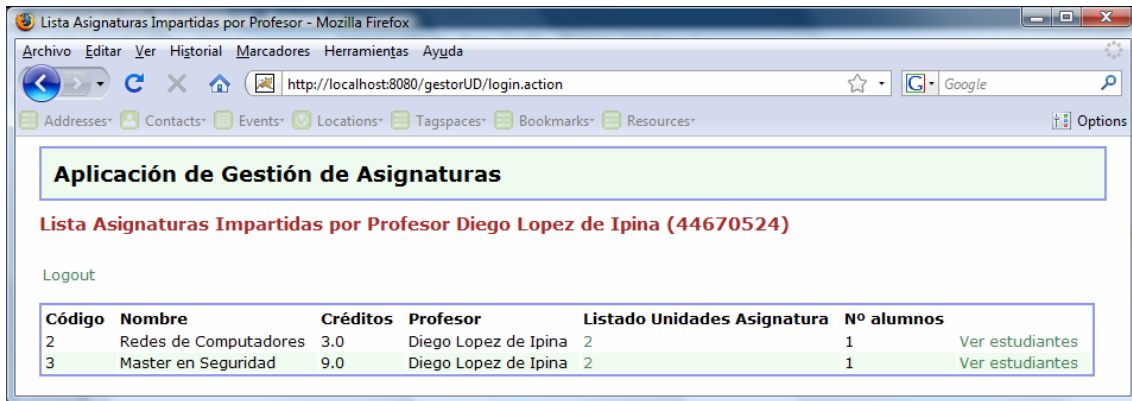


Figura 5: Listado de asignaturas impartidas por un profesor.

3. **SubjectStudents.jsp** – generará una tabla en la que se muestren los detalles de los alumnos matriculados en una asignatura. Obsérvese que el listado mostrado en Figura 6 sería generado como resultado de haber hecho click sobre el enlace “Ver estudiantes” de Figura 5.

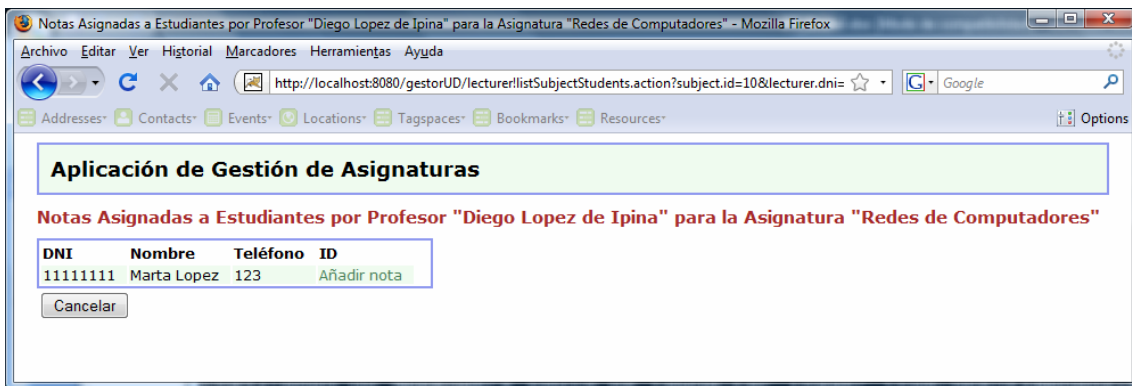


Figura 6: Listado de alumnos matriculados en una asignatura.

4. **StudentSubjectMarkingForm.jsp** – generará un formulario, similar al mostrado en Figura 7. Para la generación de tal formulario será necesario hacer uso de las siguientes etiquetas personalizadas de Struts 2: `s:form`, `s:textfield`, `s:hidden` y `s:submit`, que son equivalentes a las etiquetas homónimas en HTML. **Atención** en este caso a la utilidad de la etiqueta `s:hidden`.



Aplicación de Gestión de Asignaturas

Añadir nota

DNI	Nombre	Teléfono
11111111	Marta Lopez	123

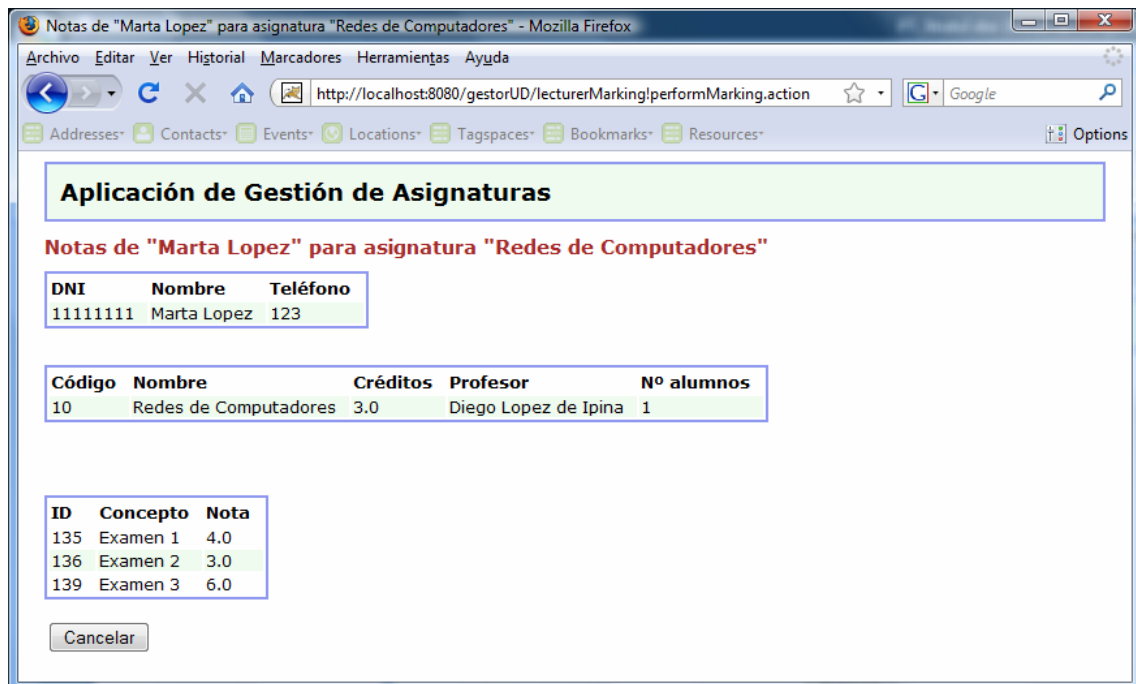
Código	Nombre	Créditos	Profesor	Nº alumnos
10	Redes de Computadores	3.0	Diego Lopez de Ipina	1

Concepto:

Nota:

Figura 7: Formulario de introducción de nota para alumno.

5. **StudentSubjectMarks** – al igual que otros JSPs ya descritos su misión principal es iterar por la colección de notas de un alumno en una asignatura con la ayuda de `:iterator` y mostrar tales notas como filas de una tabla.



Aplicación de Gestión de Asignaturas

Notas de "Marta Lopez" para asignatura "Redes de Computadores"

DNI	Nombre	Teléfono
11111111	Marta Lopez	123

Código	Nombre	Créditos	Profesor	Nº alumnos
10	Redes de Computadores	3.0	Diego Lopez de Ipina	1

ID	Concepto	Nota
135	Examen 1	4.0
136	Examen 2	3.0
139	Examen 3	6.0

Figura 8: Listado de notas para alumno "Marta López" en la asignatura "Redes de Computadores".

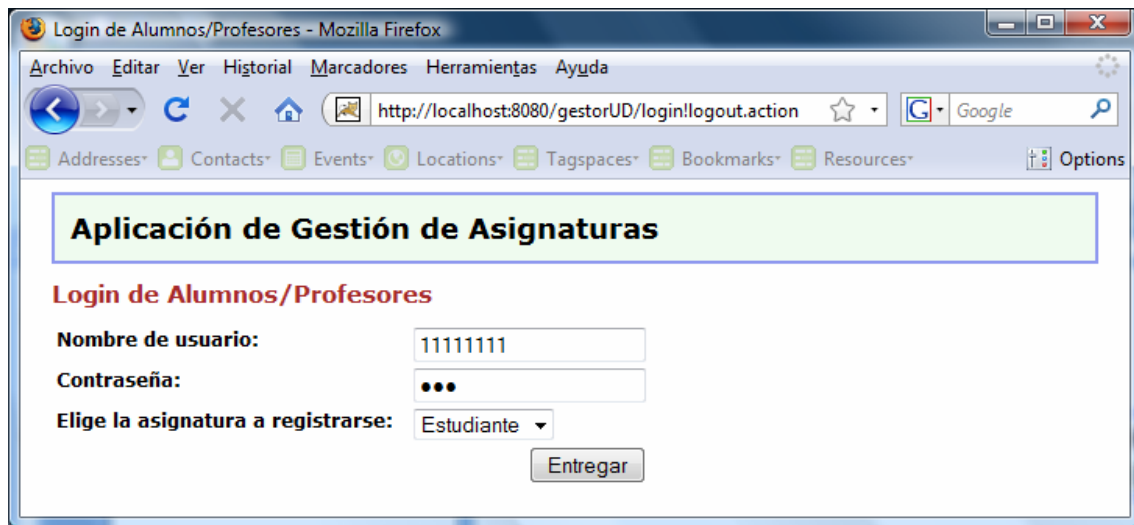


Figura 9: Pantalla de logeo de un alumno en GestorUD.

6. **StudentSubjects.jsp** – el logeo de un alumno a través de la pantalla Login.jsp antes comentada, ver Figura 9, dará lugar a una página web similar a la mostrada en Figura 10, donde se mostrarán en una tabla las asignaturas a las que un alumno se ha matriculado. Encima de la tabla que muestra las asignaturas a las que un estudiante se ha registrado aparecen acciones asociadas al estudiante: a) matricularse de una nueva asignatura, b) mostrar las notas de todas las asignaturas y c) hacer logout del sistema. Por cada asignatura las siguientes acciones serán posibles: a) des-matricularse de una asignatura que derivará en la misma vista y b) ver las notas concretas de esa asignatura. Al seleccionar el enlace “Ver notas” sobre una asignatura a la que se ha matriculado un alumno (ver Figura 10), se mostrará un listado similar al que aparece en Figura 8, que contendrá todas las notas obtenidas en una asignatura a la que se ha matriculado un estudiante.

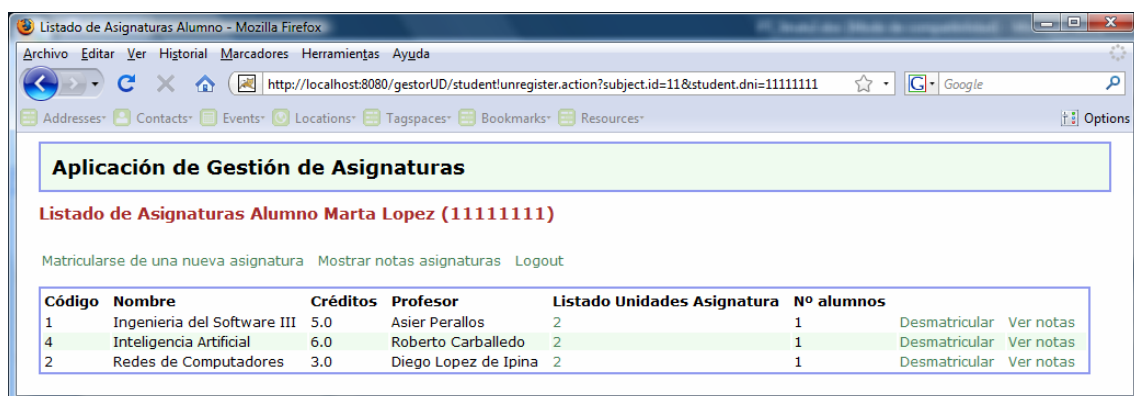


Figura 10: Pantalla mostrando asignaturas a las que un alumno se ha matriculado.

7. **StudentSubjectRegistrationForm.jsp** – al seleccionar el enlace “Matricularse de una nueva asignatura” en StudentSubjects.jsp, se mostrará un formulario similar al que aparece en Figura 11. Lo más destacado de este formulario es el uso de la

etiqueta personalizada de Struts 2 `s:select`, así como las etiquetas `s:hidden` que nuevamente haya que usar.

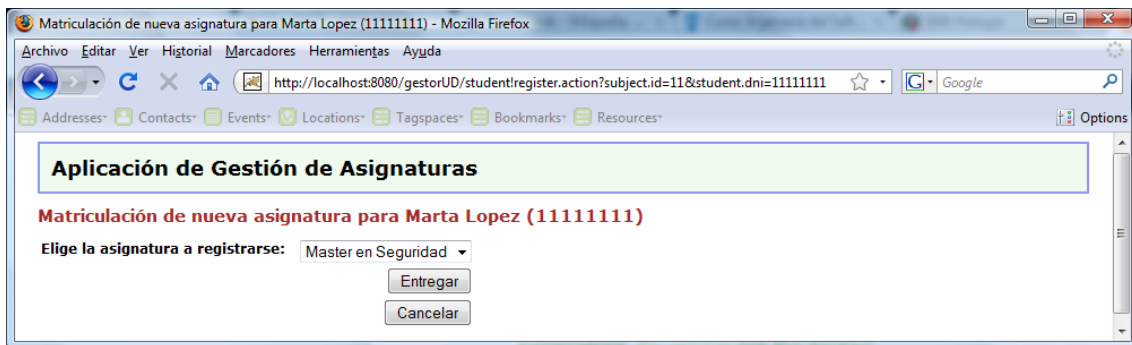


Figura 11: Pantalla con formulario para la selección de nueva asignatura a la que matricularse.

8. **StudentAllSubjectsMarks.jsp** – al seleccionar el enlace “Mostrar notas asignaturas” en StudentSubjects.jsp, se mostrará un listado similar al que aparece en Figura 12, que contendrá todas las notas obtenidas en asignaturas a las que se ha matriculado un estudiante.

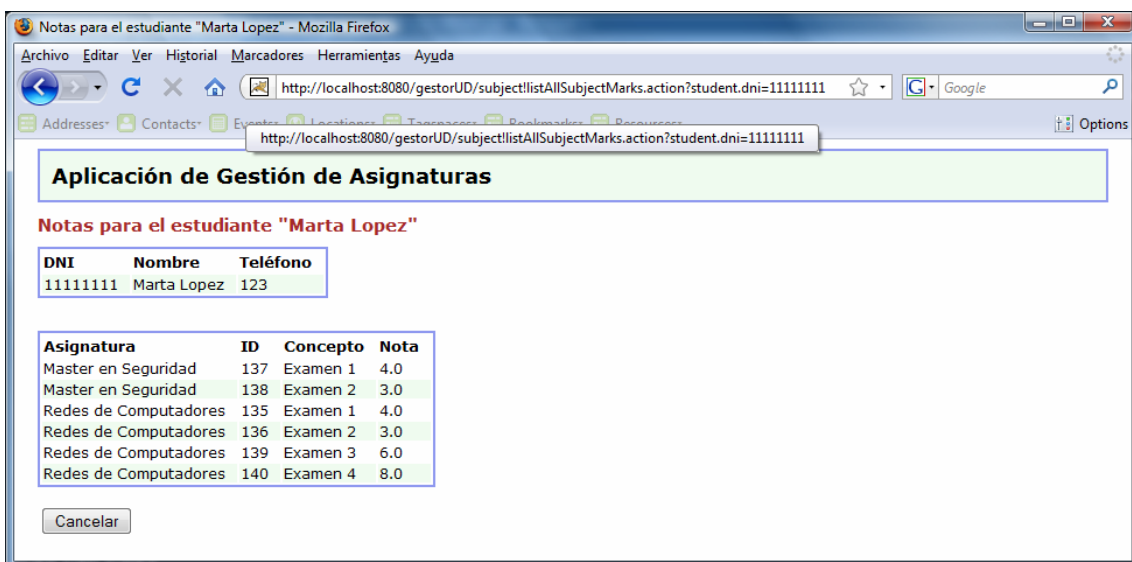


Figura 12: Pantalla mostrando todas las notas de un estudiante.

1.6. Consideraciones sobre la implementación

El enlace entre la capa de control (clases acción) y la capa de modelo debe efectuarse a través de la interfaz `PtService` y su implementación `PtDaoService`, provistas en la carpeta `iso3/pt/service` del armazón de la práctica. Para acceder desde una clase acción a `PtService` habrá de declararse el siguiente atributo:

```
private PtService service = new PtDaoService();
```

Y luego hacer uso de tal atributo dentro de los métodos de acción con sentencias como: `service.getAsignatura(subject.getId());`

Se ofrece a los alumnos una implementación de PtDaoService que quizás tengáis que adaptar para acomodarse a vuestra implementación concreta de la capa de modelo. No obstante, si se respetaron las interfaces provistas en la versión 1 ningún trabajo de adaptación debería ser necesario.

Resumiendo, el alumno deberá implementar la funcionalidad descrita en las secciones de modelo de acciones y de vistas. Tal implementación implicará los siguientes tipos de modificaciones en el armazón de práctica entregado:

- **Modificar los ficheros que contienen los recursos de propiedades multilingües** – en el directorio conf cada vez que se quiera definir un nuevo string dentro de un JSP correspondiente a un resultado.
- **Añadir una nueva declaración de acción** – cada vez que se defina una nueva acción o modificar su entrada XML <action> en conf/struts.xml cada vez que una acción deba generar un nuevo resultado.
- **Crear JSPs en el directorio jsp** – por cada nuevo resultado generado por una acción. Tales JSPs deberán utilizar las etiquetas personalizadas de Struts 2, evitando en todos los casos el uso de código Java dentro de un JSP.
- **Añadir nuevas clases Acción** – en el directorio src/iso3/pt/action que hereden de ActionSupport e implementen Preparable, si es necesario, tal como se ha explicado anteriormente.
- **Opcionalmente añadir ficheros de validación .xml** – para los formularios definidos en las vistas de la aplicación.

ATENCIÓN: Para entender mediante un ejemplo qué es lo que hay que hacer en esta segunda fase de la práctica es muy recomendable revisar en detalle la aplicación de ejemplo **struts2tutorial** suministrada. En particular, revisar el código correspondiente a la acción languageDesigners, dado que los alumnos tienen que repetir un proceso similar para cada acción a crear en la práctica.