Anàlisi, disseny i implementació d'una aplicació fent servir l'arquitectura J2EE

- Accés i consulta de la informació relacionada amb un centre mèdic -

ESTUDIANT: DAVID FARRÉ VIDAL.
E.T.I.G.

CONSULTOR: JOSEP JOAN RODRÍGUEZ.

20 de Juny de 2011

Agraïments,

A tots els que m'envolten, especialment a la meva dona, Núria. En els moments difícils sempre ha estat al meu costat donant-me ànims.

També volia agrair el suport rebut per part del consultor de l'assignatura, en Josep Joan Rodríguez, doncs sempre que ha calgut m'ha ajudat a resoldre els dubtes que m'han sorgit.

Finalment, agrair a la pròpia UOC ja que gràcies a les facilitats que ofereix per cursar estudis universitaris he pogut finalitzar l'Enginyeria Tècnica en Informàtica de Gestió.

0. RESUM.

El present document té com objectiu presentar amb detall les diferents fases (estudi inicial, anàlisi funcional, disseny i implementació) que he portat a terme per tal de realitzar aquest projecte, *l'accés i consulta de la informació relacionada amb un centre mèdic.*

Es tracta d'una aplicació basada en l'arquitectura J2EE en la qual segons l'accés que es faci a la mateixa *(login d'usuari)*, es tindrà accés a unes funcionalitats o bé a unes altres. Com veurem més endavant, s'han creat tres tipus d'usuaris: administrador, pacient i metge. També existeix una part accessible per tots els usuaris que vulguin accedir a l'aplicatiu per tal de consultar informació relacionada amb el centre mèdic (certa part d'aquesta informació estarà gestionada per l'administrador).

Per què he escollit aquest tipus de projecte? La tecnologia JAVA juntament amb l'arquitectura J2EE s'han convertit en un estàndard per al desenvolupament d'aplicacions distribuïdes a Internet.

El conjunt de tecnologies agrupades sobre J2EE permet realitzar una aplicació robusta, escalable i reutilitzable.

Amb la utilització del framework Struts 2, s'ha desenvolupat una aplicació web tenint com a base el patró Model - Vista - Controlador. Aquest model ens permet separar el model de dades, les interfícies d'usuari i la lògica de negoci en tres components diferenciats.

Àrea del TFC:

- J2EE.

Paraules Clau:

- JAVA, J2EE.
- Patró de disseny MVC.
- Framework Struts 2.
- Servidor Web.

INDEX DE CONTINGUTS.

1. INTRODUCCIÓ.

1.1. JUSTIFICACIÓ DEL TFC. PUNT DE PARTIDA I APORTACIÓ.

Punt de partida.

He escollit aquesta àrea del TFC per conèixer un tipus d'arquitectura que em permetés desenvolupar aplicacions en un entorn web. Durant els darrers anys, he treballat desenvolupant software utilitzant el llenguatge de programació Visual Basic 6.0 implementant programes "d'escriptori" de més o menys dificultat i professionalment m'havia estancat en una posició 'bastant còmoda' però sense previsió de millorar professionalment.
En l'actualitat, la majoria d'aplicacions estan pensades per funcionar en un entorn web i l'arquitectura J2EE s'ha convertit en un estàndard per al desenvolupament d'aquest tipus d'aplicacions distribuïdes a Internet.

Aportació.

Quan vaig decidir escollir aquesta àrea els meus coneixements del llenguatge JAVA, de l'arquitectura J2EE, del llenguatge HTML i fulles d'estil CSS així com del funcionament de les aplicacions distribuïdes era nul. Gràcies al desenvolupament d'aquest projecte he pogut *'renovar una mica'* els meus coneixements informàtics. En l'actualitat i des d'un punt de vista humil vull remarcar que no em considero un expert *'a nivell professional'* però si penso que he après els coneixements suficients i podria formar part d'un equip de treball que es dediqui a desenvolupar aplicacions web.

1.2. OBJECTIUS DEL TFC.

J2EE està pensat per escriure aplicacions distribuïdes basades en components. D'aquesta manera, la creació d'aplicacions distribuïdes més o menys complexes es simplifica perquè la funcionalitat que han de proporcionar s'encapsula en els propis components de J2EE.

Una de les característiques principals de la plataforma J2EE es que utilitza un model d'aplicació multicapa (Capa Client, Capa Web, Capa de Negoci, Capa de recursos d'informació).

Objectius:

- Aprofundir en l'ús de la tecnologia JAVA, conèixer l'arquitectura J2EE.
- Entendre el concepte d'aplicació distribuïda utilitzant el model multicapa.
- Conèixer i utilitzar el framework Struts 2.
- Implementar una aplicació amb una interfície senzilla, usable i accessible, basant-me en aquest tipus d'arquitectura.

Des del meu punt de vista, en cap moment considero que l'objectiu del TFC sigui realitzar una 'gran aplicació' degut a la meva desconeixença en el desenvolupament d'aquest tipus d'aplicacions. Considero que l'objectiu principal es realitzar una aplicació senzilla, que utilitzi el patró de disseny MVC i veure la traducció en Struts 2 d'aquest patró. A més a més, per garantir un correcte funcionament de l'aplicació i complir les expectatives creades pels nostres usuaris s'han tingut en compte els següents punts:

- Visibilitat de l'estat del sistema. Sempre haurem d'informar a l'usuari sobre el que està succeint.
- L'aplicació web haurà de ser consistent i estàndard, així resultarà familiar i fàcil d'utilitzar pels usuaris.
- Prevenció d'errors. Millor que missatges d'error, serà fer un disseny que previngui que surtin errors.
- Llenguatge comú entre sistema i usuari. El sistema ha de parlar la llengua de l'usuari, deixant enrere tecnicismes incomprensibles o missatges críptics.

Per assolir aquests objectius també he hagut d'investigar i treballar en els següents camps:

- Decidir i aprendre com funciona l'entorn integrat de desenvolupament.
  En aquest cas, he optat per utilitzar *NetBeans*. En una primera fase vaig investigar quin IDE escollir i vaig dubtar entre NetBeans o bé Eclipse. Investigant per Internet vaig trobar uns videotutorials del *professor Jesús Conde* (Cursos Iniciació a Java amb NetBeans), els quals van ajudar-me a començar des de 0 i aquest va estar el motiu de la meva elecció.
- Servidor Web.
  He optat per utilitzar *Apache TomCat.*
- Gestor de bases de dades.
  He optat per utilitzar *MySQL.*

1.3. ENFOCAMENT I MÈTODE SEGUIT.

Els projectes es divideixen en etapes per facilitar la seva gestió i el control. Aquest conjunt d'etapes s'anomena cicle de vida del projecte. En el meu cas, el projecte *Centre Mèdic FA_VI* ha estat dividit en les següents etapes:

- PLA DE TREBALL.
- ANÀLISI i DISSENY.
- IMPLEMENTACIÓ i PROVES.

- MEMÒRIA i PRESENTACIÓ.

Paral·lelament, s'ha fet un estudi tant del llenguatge JAVA com de l'arquitectura J2EE.

1. Introducció. Aproximació a JAVA i NetBeans.
2. Programació bàsica (variables, operadors, condicionals,...).
3. POO en JAVA (Jerarquia de classes i objectes, import, extend,...).
4. Aplicacions Web amb NetBeans (instal·lació del servidor d'aplicacions, instal·lació del SGBD, framework Struts 2, ...)

Com es pot veure la corba d'aprenentatge ha estat elevada i crec que he pogut realitzar el desenvolupament del projecte gràcies a una bona planificació del mateix així com al sistema que es fa servir d'avaluació continuada durant el desenvolupament del TFC.

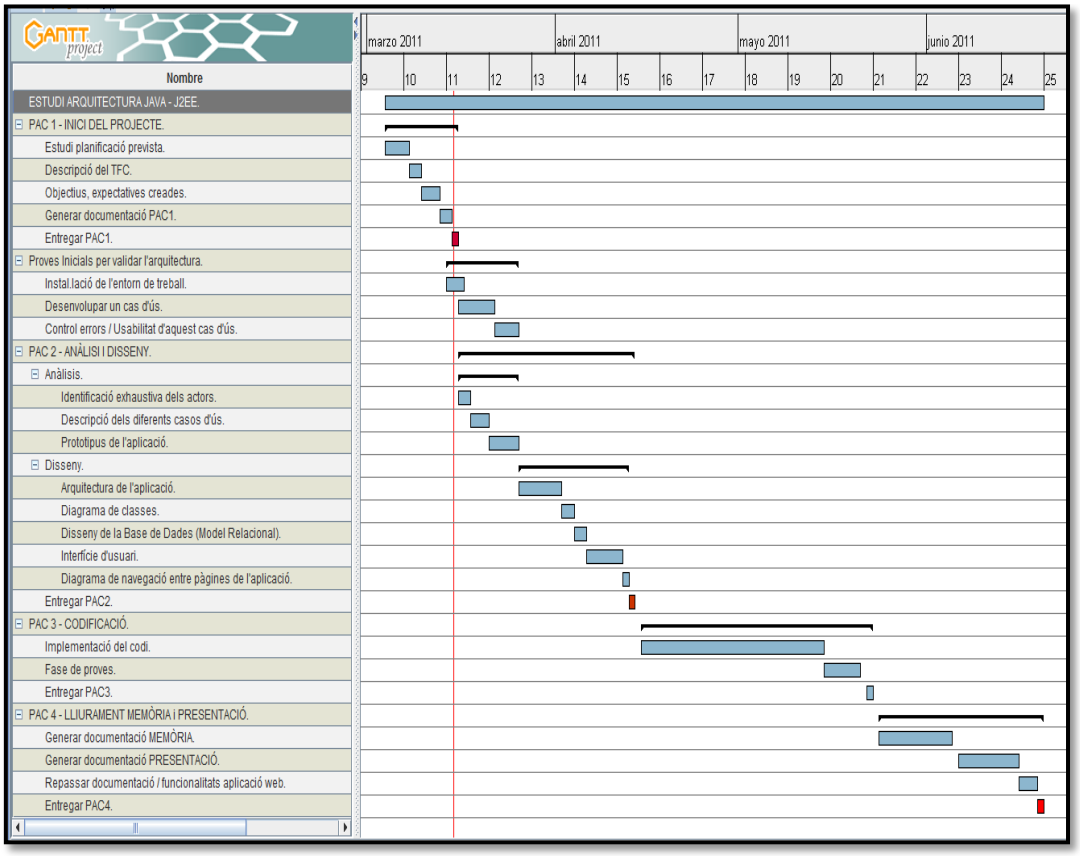
1.4. PLANIFICACIÓ DEL PROJECTE.

La planificació del projecte està basada en la data d'entrega de les diferents PACs que formen part de l'avaluació continuada. Per planificar el projecte d'una forma visual he utilitzat el programa ***GanttProject.***

Com es pot comprovar en el següent diagrama de Gantt, la tasca d'aprenentatge de la tecnologia JAVA no està planificada en un interval tancat de temps, ja que paral·lelament a l'entrega de les diferents PACs he estat estudiant JAVA i J2EE *(Fins i tot una vegada finalitzi l'entrega del TFC vull seguir adquirint nous coneixements basant-me en aquesta arquitectura).*

Respecte a la planificació prevista els principals desajustaments s'han produït en l'etapa d'implementació del codi. Gràcies a l'avaluació continuada de l'assignatura he pogut aprofitar al màxim la documentació entregada en les diferents PACs i d'aquesta manera he utilitzat temps del previst per generar la memòria final a finalitzar la implementació del codi així com realitzar la fase de proves a posteriori.

Nombre	Fecha de inicio	Fecha de fin	Duración
ESTUDI ARQUITECTURA JAVA - J2EE.	4/03/11	20/06/11	108
PAC 1 - INICI DEL PROJECTE.	4/03/11	16/03/11	12
Estudi planificació prevista.	4/03/11	8/03/11	4
Descripció del TFC.	8/03/11	10/03/11	2
Objectius, expectatives creades.	10/03/11	13/03/11	3
Generar documentació PAC1.	13/03/11	15/03/11	2
Entregar PAC1.	15/03/11	16/03/11	1
Proves Inicials per validar l'arquitectura.	14/03/11	26/03/11	12
Instal.lació de l'entorn de treball.	14/03/11	17/03/11	3
Desenvolupar un cas d'ús.	16/03/11	22/03/11	6
Control errors / Usabilitat d'aquest cas d'ús.	22/03/11	26/03/11	4
PAC 2 - ANÀLISI I DISSENY.	16/03/11	14/04/11	29
Anàlisis.	16/03/11	26/03/11	10
Identificació exhaustiva dels actors.	16/03/11	18/03/11	2
Descripció dels diferents casos d'ús.	18/03/11	21/03/11	3
Prototipus de l'aplicació.	21/03/11	26/03/11	5
Disseny.	26/03/11	13/04/11	18
Arquitectura de l'aplicació.	26/03/11	2/04/11	7
Diagrama de classes.	2/04/11	4/04/11	2
Disseny de la Base de Dades (Model Relacional).	4/04/11	6/04/11	2
Interfície d'usuari.	6/04/11	12/04/11	6
Diagrama de navegació entre pàgines de l'aplicació.	12/04/11	13/04/11	1
Entregar PAC2.	13/04/11	14/04/11	1
PAC 3 - CODIFICACIÓ.	15/04/11	23/05/11	38
Implementació del codi.	15/04/11	15/05/11	30
Fase de proves.	15/05/11	21/05/11	6
Entregar PAC3.	22/05/11	23/05/11	1
PAC 4 - LLIURAMENT MEMÒRIA i PRESENTACIÓ.	24/05/11	20/06/11	27
Generar documentació MEMÒRIA.	24/05/11	5/06/11	12
Generar documentació PRESENTACIÓ.	6/06/11	16/06/11	10
Repassar documentació / funcionalitats aplicació web.	16/06/11	19/06/11	3
Entregar PAC4.	19/06/11	20/06/11	1



1.5. PRODUCTES OBTINGUTS.

El producte obtingut en la realització d'aquest TFC consisteix en:

- Una aplicació web senzilla on les seves principals funcionalitats són:

  - Reservar cita prèvia a través d'Internet de les diferents especialitats que ofereix el centre mèdic.

  - Consultar les opcions que ofereix el centre mèdic (especialitats, notícies, instal·lacions,...)

  - Mantenir el contingut de l'aplicació actualitzat gràcies a la figura de l'administrador que podrà gestionar les especialitats, les notícies relacionades amb el centre mèdic, ...

Per l'entrega final es lliuren els següents productes:

- Memòria del projecte.
- Presentació virtual del projecte sintetitzant tota la informació.
- Aplicació empaquetada en un fitxer *.war* adjuntant els fitxers *.java.*
- Script SQL (creació de la Base Dades, taules necessàries, dades, ...)
- Instruccions per tal d'instal·lar l'aplicació i la Base de Dades.

1.6. BREU DESCRIPCIÓ DELS SEGÜENTS CAPÍTOLS.

En la resta de capítols es presenten els següents aspectes:

Anàlisi.

1. Es fa la distinció dels diferents actors que intervenen en el projecte així com els casos d'ús que s'han implementat.
2. Es mostra el prototipus proposat per fer la interfície gràfica.

Disseny i Implementació.

1. S'explica l'arquitectura J2EE, el patró de disseny MVC.
2. S'estudia el framework Struts 2 així com els seus fitxers de configuració més importants.
3. Es realitza el disseny de la base de dades i s'explica com es fa l'accés a les dades seguint el model DAO.
4. S'identifiquen les diferents classes d'entitats que formen el projecte.
5. Es mostra la interfície gràfica resultant.
6. Es presenten aspectes relacionats amb la implementació de l'aplicació com validacions, internacionalització,...
7. Es mostren les proves realitzades (a nivell de validació, seguretat, ...)
8. S'indiquen les eines utilitzades per desenvolupar el projecte.

Conclusions. S'expliquen les principals conclusions obtingudes.

Glossari.

Webgrafia / Bibliografia consultada.

Annexes.

1. Donat que he utilitzat el framework Struts 2, faig una comparativa amb el seu antecessor (Struts 1) tot i que Struts 2 no es una extensió de Struts 1 ja que té un disseny completament nou.
2. Es detallen els interceptors que Struts 2 ofereix per defecte.
3. Finalment dedico un apartat per destacar una sèrie de complements analitzats i que per manca de temps o bé per alguna mena de dificultat afegida NO he pogut implementar.

2. ANÀLISI.

En aquest apartat, estudiarem els diferents actors que intervenen en l'aplicació estudiant els seus casos d'ús. A més, mostrarem un prototipus del disseny (interfície gràfica de l'aplicació). Després en la fase de disseny es veurà el resultat obtingut.

2.1. ACTORS.

- *Usuari NO REGISTRAT.*

Aquest tipus d'usuari serà qualsevol persona que vulgui accedir a la pàgina web del centre mèdic per consultar les funcionalitats que ofereix el propi centre: notícies, especialitats mèdiques, instal·lacions, ...

- *Usuari REGISTRAT (L'anomenarem PACIENT).*

Els pacients tindran accés al seu propi espai dins del centre per tal de sol·licitar una cita prèvia, donada una especialitat mèdica.
També tindran l'opció de consultar les seves visites pendents així com veure el seu historial clínic des de que formen part del centre mèdic.
Finalment, podran consultar les cites reservades d'una especialitat en una data.

- *ADMINISTRADOR.*

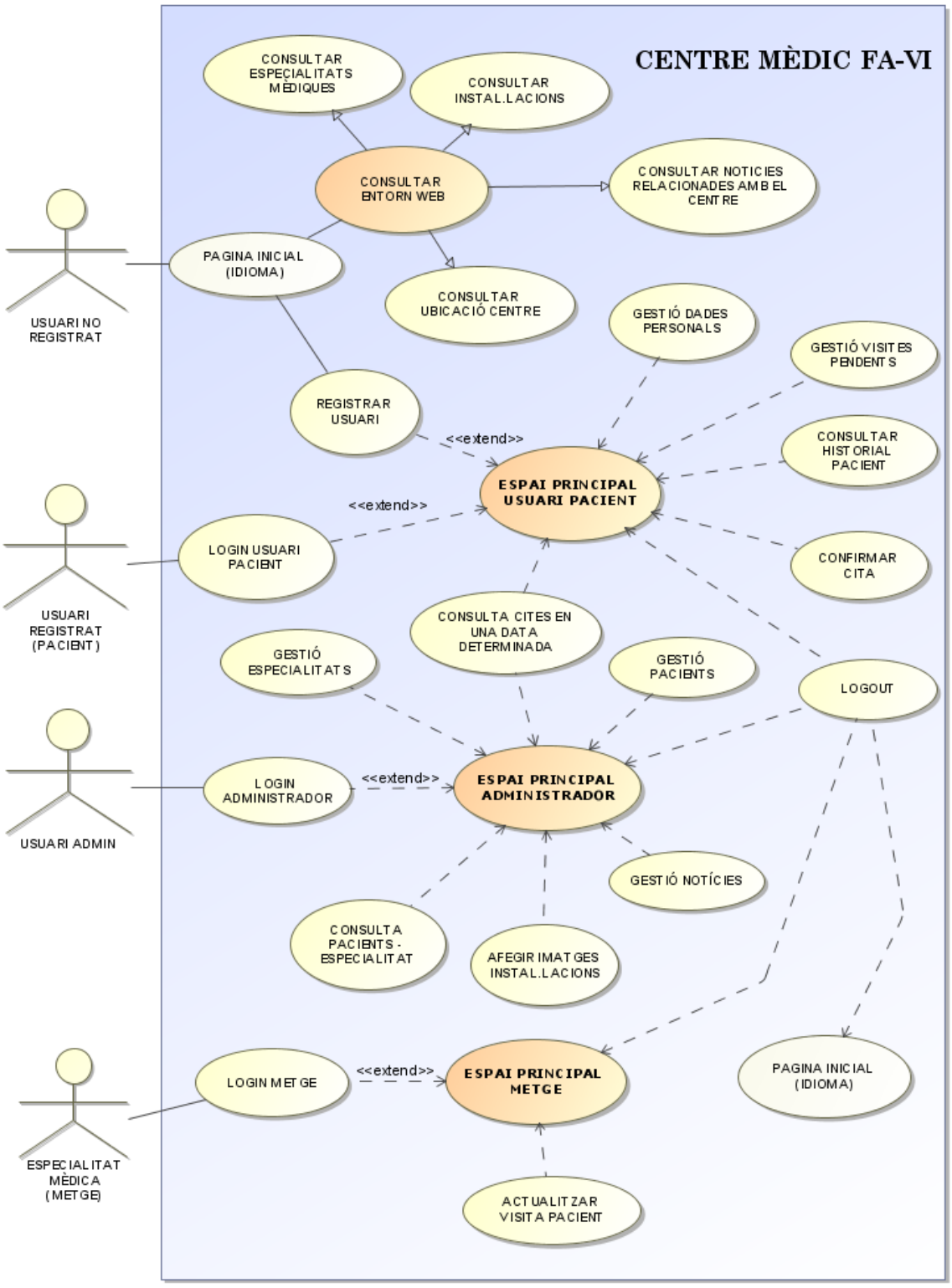
Donat un nom d'usuari i contrasenya, l'aplicació detectarà que un usuari té privilegis d'administrador. Aquest serà l'encarregat de :
- Mantindre la informació actualitzada en quant a les especialitats que ofereix el centre (He suposat que el concepte especialitat mèdica / metge va relacionat 1:1, l'administrador podrà donar d'alta especialitats, modificar el nom del metge assignat a una especialitat, eliminar una especialitat, ...)
- Actualitzar les notícies relacionades amb el centre.
- Gestió de pacients (Alta / Baixa / Modificació).
- Pujar imatges.
- Consulta una sèrie de llistats relacionats amb les cites realitzades per part dels pacients.

- *METGE (o bé especialitat mèdica).*

Actor creat per l'administrador. La figura del metge també farà que l'aplicació s'executi de forma totalment diferent.
Serà l'encarregat d'actualitzar la informació referent a la visita que tingui amb un pacient. D'aquesta manera totes les visites realitzades quedaran enregistrades i el pacient podrà consultar-les des del seu propi espai.

2.2. DIAGRAMA DE CASOS D'ÚS.

Una vegada hem definit els actors que formaran part de l'aplicació, mostrem el diagrama de casos d'ús resultant:



2.3. DESCRIPCIÓ TEXTUAL DELS DIFERENTS CASOS D'ÚS.

Cas d'ús: CONSULTAR ENTORN WEB
Funcionalitat general: Permet navegar per la part exterior de l'aplicació, consultant totes les opcions que no necessiten validació d'usuari (Especialitats mèdiques, instal·lacions, ...).
Actor: Usuari NO REGISTRAT.
Precondició: -
Postcondició: L'usuari coneix les opcions que ofereix el centre mèdic.
Flux d'events:
- L'usuari pot escollir entre les diferents opcions que es mostren en la pantalla principal per tal de conèixer les diferents funcionalitats que presenta el centre mèdic.

Cas d'ús: REGISTRAR USUARI
Funcionalitat general: Permet registrar a l'usuari al centre mèdic.
Actor: Usuari NO REGISTRAT.
Precondició: -
Postcondició: L'usuari queda registrat (Usuari PACIENT) i accedeix al seu propi espai dins de la web.
Flux d'events:
- Des de la pantalla de benvinguda, l'usuari selecciona l'opció *registra't.* - S'obrirà un formulari per tal que l'usuari pugui donar-se d'alta. - Un cop introduïdes totes les dades es validarà la informació. - Es mostrarà un missatge de confirmació per pantalla i l'usuari podrà accedir al seu espai personal.
Flux alternatiu:
- Si l'usuari introdueix dades NO vàlides es mostrarà un error i l'usuari podrà entrar les dades de nou. - En cas que l'usuari no vulgui continuar, l'usuari seleccionarà l'opció *cancel·lar* i es tornarà a la pantalla de benvinguda.

Cas d'ús: LOGIN USUARI *(PACIENT, ADMINISTRADOR, METGE)*
Funcionalitat general: Accedir a l'aplicació, validant si l'usuari està registrat.
Actor: PACIENT, ADMINISTRADOR o bé METGE.
Precondició: -
Postcondició: L'usuari accedeix al seu propi espai o bé es queda en la pantalla de benvinguda si hi ha error.
Flux d'events:
- Usuari introdueix el seu nom d'usuari i contrasenya. - Prem l'opció Entrar i espera resposta per part de l'aplicació (Verificació de les dades) - Segons el rol d'usuari s'accedeix al seu espai personal.
Flux alternatiu:
- L'usuari no està donat d'alta o bé ha introduït les dades incorrectament. - L'aplicació mostra un missatge d'error i retorna el control a l'usuari.

Cas d'ús: GESTIÓ DADES PERSONALS
Funcionalitat general: Permet modificar les dades personals així com l'opció de donar-se de baixa del centre mèdic.
Actor: PACIENT.
Precondició: El pacient ha accedit a l'aplicació i vol consultar les seves dades personals.
Postcondició: El pacient ha modificat les seves dades o bé s'ha donat de baixa.
Flux d'events:
- El pacient selecciona l'opció *Dades Personals* dins del seu espai personal. - Es mostra un formulari amb totes les dades personals del pacient juntament amb les opcions: Modificar Dades / Cancel·lar / Donar-se de Baixa. - Si el pacient es dóna de baixa del centre mèdic, s'eliminarà en cascada tota la informació associada al pacient. Es retornarà a la pantalla inicial *(Selecció Idioma)*. - Si el pacient selecciona l'opció Modificar Dades, s'actualitzarà la informació. Es mostrarà un missatge a l'usuari de confirmació i s'accedirà de nou a la pantalla principal del seu propi espai.
Flux alternatiu:
- Si el pacient introdueix alguna dada incoherent, es mostrarà un missatge d'error i es tornarà el control a l'usuari. - L'usuari podrà sortir del formulari sense actualitzar les dades (Opció Cancel·lar).

Cas d'ús: GESTIÓ VISITES PENDENTS
Funcionalitat general: Consultar totes les visites pendents que té el pacient. Un pacient pot tindre 'n' visites pendents però sempre que siguin d'especialitats diferents.
Actor: PACIENT.
Precondició: El pacient ha accedit a l'aplicació i vol consultar o bé eliminar visites pendents.
Postcondició: -
Flux d'events:
- El pacient selecciona l'opció *Consultar visites pendents* dins del seu espai personal. - Es mostra un formulari amb totes les visites pendents indicant la data, l'especialitat i l'opció d'eliminar cadascuna de les mateixes. - Si el pacient vol eliminar una cita, seleccionarà l'acció corresponent. Per seguretat, es preguntarà a l'usuari si realment vol fer aquesta operació i en cas afirmatiu s'eliminarà la cita en qüestió. Es mostrarà de nou el formulari anterior. Seleccionant l'opció <<*Tornar*, s'accedirà de nou a la pantalla principal.

Cas d'ús: CONSULTAR HISTORIAL PACIENT
Funcionalitat general: Consultar totes les visites ateses que ha tingut el pacient en el centre mèdic.
Actor: PACIENT.
Precondició: El pacient ha accedit a l'aplicació i vol consultar el seu historial clínic.
Postcondició: -
Flux d'events:
- El pacient selecciona l'opció *Historial Pacient* dins del seu espai personal. - Es mostra un formulari amb totes les visites que han estat ateses indicant l'especialitat, la data, el motiu i les recomanacions proposades pel metge. Seleccionant l'opció <<*Tornar,* s'accedirà de nou a la pantalla principal.

Cas d'ús: CONFIRMAR CITA
Funcionalitat general: El pacient pot reservar cita per una especialitat en concret en una data determinada **(es tracta de la funcionalitat principal de l'aplicació).**
Actor: PACIENT
Precondició: El pacient ha accedit a l'aplicació i vol reservar cita online.
Postcondició: El pacient ha reservat cita per visitar-se d'una especialitat en concret.
Flux d'events:
- Des del propi espai principal, el pacient seleccionarà una especialitat i una data (dia / hora) en concret. - Es validarà si es pot realitzar la cita prèvia. - Es mostrarà un missatge al pacient, indicant que la cita s'ha realitzat amb èxit.
Flux alternatiu:
Controls: - El pacient ja té una visita pendent per l'especialitat seleccionada o bé la data escollida està reservada. En aquests casos, es mostrarà un missatge per pantalla indicant *que la reserva no ha estat realitzada* i es tornarà el control a l'usuari.

Cas d'ús: CONSULTA CITES EN UNA DATA DETERMINADA
Funcionalitat general: Seleccionant una especialitat i una data (dia), es mostra un llistat amb les hores 'reservades'.
Actor: PACIENT, ADMINISTRADOR.
Precondició: L'usuari es troba dins del seu espai personal i, donada una especialitat i una data en concret, vol conèixer les hores que ja estan ocupades.
Postcondició: A nivell d'usuari PACIENT. L'usuari coneix la disponibilitat horària d'una especialitat en un dia, per tant podrà reservar cita sabent quines hores pot escollir. A nivell d'usuari ADMINISTRADOR. L'usuari disposa d'un llistat (per pantalla) de totes les cites diàries per una especialitat en concret.
Flux d'events:
1. En cas que l'usuari sigui l'administrador, selecciona l'opció *'Llistat cites d'una especialitat en concret'*. En cas que l'usuari sigui un PACIENT, selecciona l'opció *'CONSULTAR HORES RESERVADES'*. *2.* L'usuari escull una especialitat i una data i selecciona l'opció *'VEURE LLISTAT'.* *3.* Es mostra el llistat per pantalla. 4. L'usuari pot tornar enrere seleccionant l'opció <<*TORNAR.*

Cas d'ús: ACTUALITZAR VISITA PACIENT
Funcionalitat general: El metge actualitzarà les dades d'una visita que tingui amb un pacient. D'aquesta manera, el pacient podrà consultar l'historial de visites realitzades.
Actor: METGE.
Precondició: L'usuari ha entrat a l'aplicació amb perfil d'usuari METGE.
Postcondició: Les dades de la visita d'un pacient han quedat actualitzades.
Flux d'events:
- Donada una llista amb les visites pendents que té assignades, el metge seleccionarà el registre corresponent al pacient que s'està visitant (Suposarem que tots els pacients que es visiten al centre mèdic han reservat cita prèvia a través de la web). - El metge actualitzarà la informació del pacient amb les dades corresponents (Motiu visita i recomanacions).
Flux alternatiu:
- S'han introduït dades incorrectes. Es mostrarà el missatge corresponent i es tornarà el control a l'usuari. o bé - El metge no troba el registre corresponent al pacient que s'està visitant. Això equival a dir que el pacient no ha reservat cita prèvia i s'ha presentat al centre mèdic (OPCIÓ NO IMPLEMENTADA).

Cas d'ús: GESTIO NOTICIES
Funcionalitat general: L'administrador podrà afegir, modificar o bé eliminar notícies relacionades amb el centre mèdic.
Actor: ADMINISTRADOR.
Precondició: L'usuari ha entrat a l'aplicació amb perfil d'administrador i selecciona l'opció *Gestionar Notícies.*
Postcondició: L'administrador ha afegit, modificat o bé eliminat les notícies corresponents.
Flux d'events:
- L'administrador podrà afegir noves notícies amb un títol i una descripció associada. De la mateixa manera, podrà modificar notícies i si ho considera oportú eliminar-les.
Flux alternatiu:
- S'han introduït dades incorrectes. Es mostrarà el missatge corresponent i es tornarà el control a l'usuari.

Cas d'ús: GESTIO ESPECIALITATS
Funcionalitat general: L'administrador podrà afegir, modificar o bé eliminar especialitats dins del centre mèdic.
Actor: ADMINISTRADOR.
Precondició: L'usuari ha entrat a l'aplicació amb perfil d'administrador i selecciona l'opció *Gestionar Especialitats*.
Postcondició: L'administrador ha afegit, modificat o bé eliminat les especialitats corresponents.
Flux d'events:
- Si selecciona *Afegir especialitat*, l'administrador estarà creant un nou usuari tipus METGE. Entrarà nom d'usuari, contrasenya, nom especialitat, descripció, nom metge assignat, ...) - Si selecciona *Modificar especialitat*, l'administrador podrà modificar qualsevol dels camps anteriorment mencionats (excepte el camp nom d'usuari). - Si selecciona *Eliminar especialitat*, s'eliminarà l'especialitat del centre mèdic.
Flux alternatiu:
- S'han introduït dades incorrectes. Es mostrarà el missatge corresponent i es tornarà el control a l'usuari. o bé - L'administrador vol eliminar una especialitat que té assignades visites al centre (ja siguin pendents o bé finalitzades). Es mostrarà un missatge d'avís indicant que no es pot eliminar l'especialitat seleccionada i es tornarà el control a l'usuari. *Tal com s'ha dissenyat la base de dades, per consultar l'especialitat associada a una visita d'un pacient ho consultarà de la taula d'especialitats i si l'administrador elimina aquest registre, no hi haurà integritat de dades.*

Cas d'ús: GESTIÓ PACIENTS
Funcionalitat general: L'administrador podrà afegir / modificar / marcar[1] o bé eliminar pacients dins del centre mèdic.
Actor: ADMINISTRADOR.
Precondició: L'usuari ha entrat a l'aplicació amb perfil d'administrador.
Postcondició: L'administrador ha afegit / modificat / marcat / eliminat els pacients corresponents.
Flux d'events:
- Si selecciona *Afegir pacient*, l'administrador podrà crear un nou usuari de perfil PACIENT. Entrarà nom d'usuari, contrasenya, nom, cognom, ...) - Si selecciona *Editar pacient,* l'administrador podrà modificar la informació personal del propi pacient (excepte el camp nom d'usuari). - Si selecciona *Activar / Desactivar pacient,* l'administrador podrà habilitar / restringir l'accés al centre a un usuari tipus PACIENT (Un usuari 'desactivat' NO podrà accedir al sistema però tota la seva informació es mantindrà emmagatzemada en la Base de Dades). - Si selecciona *Eliminar pacient*, s'eliminarà el pacient del centre mèdic.

Cas d'ús: AFEGIR IMATGES INSTAL·LACIONS
Funcionalitat general: L'administrador podrà pujar imatges relacionades amb les instal·lacions que ofereix el centre mèdic.
Actor: ADMINISTRADOR.
Precondició: L'administrador selecciona l'opció *Afegir imatges.*
Postcondició: Imatge pujada correctament al servidor *(../webapps/Projecte_UOC/imatges_centre).*
Flux d'events:
1. L'usuari selecciona l'opció *'Examinar'* per tal de localitzar la imatge en concret. 2. Selecciona l'opció *>>Pujar Imatge.* 3. El sistema mostra un missatge per pantalla indicant que la imatge ha pujat correctament i es torna el control a l'usuari.
Flux alternatiu:
Possibles errors: 1. Només es permet pujar imatges (format .png, .gif, .jpg). 2. Tamany màxim imatge (Aprox: 2 MG). 3. No es permet pujar una imatge amb el mateix nom.
NOTA: En un primer moment, volia implementar totes les funcionalitats lligades amb la gestió d'imatges (Afegir / Modificar / Eliminar) i lligar-ho amb la Base de Dades. Per manca de temps, només he pogut realitzar la funcionalitat *Afegir Imatges.*

Cas d'ús: CONSULTAR PACIENTS RELACIONATS AMB UNA ESPECIALITAT
Funcionalitat general: Seleccionant una especialitat en concret, es mostra un llistat amb els pacients que tenen algun tipus de visita (pendent o bé realitzada) respecte aquesta especialitat.
Actor: ADMINISTRADOR
Precondició: L'usuari es troba dins del seu espai personal i vol conèixer les cites que s'han de realitzar (o bé que s'han realitzat) d'una especialitat en concret.
Postcondició: Es mostra un llistat per pantalla de totes les visites vinculades amb l'especialitat seleccionada.
Flux d'events:
1. L'administrador selecciona l'opció *'Llistar Pacients relacionats amb una especialitat'*. 2. L'administrador escull una especialitat en concret i selecciona l'opció *'VEURE LLISTAT'*. 3. Es mostra el llistat per pantalla. 4. L'administrador pot tornar enrere seleccionant l'opció *<<TORNAR.*

Cas d'ús: LOGOUT
Funcionalitat general: Tancar la sessió actual.
Actor: PACIENT, ADMINISTRADOR o bé METGE.
Precondició: L'usuari es troba dins de l'aplicació *(sessió d'usuari oberta)*. L'enllaç per fer Logout es accessible en tot moment.
Postcontdició: L'usuari ha tancat la sessió actual. Es retornarà a la pantalla inicial *(Selecció Idioma)*.
Flux d'events:
L'usuari ha realitzat les tasques pertinents i vol sortir de l'aplicació. En tot moment, l'usuari pot seleccionar l'opció *Tancar sessió (Logout).*

[1] marcar pacients: activar / desactivar pacients.

2.4. PROTOTIPUS.

L'objectiu d'aquest disseny es definir una estructura de pàgina amb capçalera i peu de pàgina amb un menú lateral de navegació i una zona de continguts. En aquest apartat, mostrarem el prototipus d'alguna de les pantalles que formen part de l'aplicació.

Suposant l'actor **USUARI NO REGISTRAT**, podem pensar en un prototipus com el següent:

- *Pantalla de Benvinguda (Presentació centre_mèdic):*

CENTRE MÈDIC FA - VI	
Zona de validació d'usuari (Nom d'usuari i contrasenya)	
> Presentació > Especialitats mèdiques > Instal·lacions > Notícies > On som? > Contacta amb nosaltres	Presentació del centre mèdic indicant les característiques principals, història, ... afegint alguna fotografia, així quedarà més presentable. *(Pantalla amb contingut estàtic).*
Peu de pàgina	

- *Si seleccionem Especialitats mèdiques:*

CENTRE MÈDIC FA - VI	
Zona de validació d'usuari (Nom d'usuari i contrasenya)	
> Presentació > Especialitats mediques > Instal·lacions > Notícies > On som? > Contacta amb nosaltres	**Especialitats mèdiques:** - Cardiologia. - Pediatria. - Dermatologia. - Medicina General. - Psicologia. - Ginecologia. - … *(Pantalla amb contingut dinàmic)*
Peu de pàgina	

- Tant les especialitats mèdiques que ofereix el centre mèdic com les notícies relacionades amb el centre es consultaran d'una Base de Dades.

Tot seguit, mostrem el prototipus de les pantalles principals dels diferents actors que formen part de l'aplicació.

- *Prototipus de la pantalla principal per l'actor **PACIENT:***

CENTRE MÈDIC FA - VI	Missatge de Benvinguda
Benvingut a la clínica. Selecciona l'especialitat desitjada i reserva visita ONLINE. *Llista desplegable amb les especialitats i un botó Reserva Cita on es desplegarà un calendari.* Especialitats Reserva Cita	> Dades personals. > Consultar visites pendents. > Historial pacient.

- *Prototipus de la pantalla principal per l'actor **ADMINISTRADOR:***

PÀGINA PRINCIPAL ADMINISTRADOR.

- Gestió notícies.
Seleccionant aquesta opció, l'administrador podrà afegir, modificar o bé eliminar notícies relacionades amb el centre mèdic.

- Gestió especialitats.
Seleccionant aquesta opció, l'administrador podrà afegir, modificar o bé eliminar especialitats que ofereix el centre mèdic.

- Gestió pacients.
Seleccionant aquesta opció, l'administrador podrà afegir, modificar o bé eliminar pacients.

- Afegir imatges. Opció de pujar imatges.

- Prototipus de la pantalla principal per l'actor **METGE:**

PÀGINA PRINCIPAL METGE.

- Es mostraran les visites pendents relacionades amb el metge que hagi entrat a l'aplicació entre les quals trobarem la del pacient que s'està visitant en aquests moments. Seleccionant el registre corresponent es mostrarà un petit formulari que ha d'omplir el metge.

Formulari a omplir:

- Motiu visita.

- Solució / Recomanació metge.

3. DISSENY i IMPLEMENTACIÓ.

3.1. ARQUITECTURA DE L'APLICACIÓ.

Per què arquitectura J2EE?

J2EE ha estat pensat per escriure aplicacions distribuïdes basades en components. D'aquesta manera, la creació d'aplicacions complexes distribuïdes es simplifica, perquè la funcionalitat que han de proporcionar s'encapsula en els components de J2EE.
S'utilitza una lògica de programació desenvolupada en capes. D'aquesta manera ens permetrà separar elements de la nostra aplicació en parts clarament definides.

CAPA CLIENT / PRESENTACIÓ.
Els components d'aquesta capa s'executen en la màquina client, esta composada pels programes que interactuen amb l'usuari. Es realitzaran peticions a les capes superiors *(la majoria de les peticions van dirigides a components de la capa WEB)* i aquestes a la seva vegada enviaran respostes a aquestes sol·licituds.

CAPA WEB.
Els components d'aquesta capa s'executen en el servidor J2EE i utilitzen el protocol HTTP per rebre les peticions i enviar les respostes a altres components. Aquesta capa proporcionarà a l'aplicació, la funcionalitat d'Internet ja que permet proporcionar serveis a la capa client.

CAPA DE NEGOCI.
Els components d'aquesta capa s'executen en el servidor J2EE. La lògica de negoci s'executa en aquesta capa.

CAPA DE RECURSOS D'INFORMACIÓ / CAPA DE DADES.
Els components d'aquesta capa s'executen en el servidor que administra aquests recursos. Connecta l'aplicació J2EE amb el software que no forma part de J2EE, incloent sistemes administradors de Bases de Dades i altres servidors ja existents.

Aquest tipus d'arquitectura fomenta l'ús del patró de disseny

MODEL – VISTA - CONTROLADOR (MVC),

el qual permet separar la part lògica de la presentació en una aplicació d'Internet.

MODEL.
El formen els components que controlen les dades que utilitza l'aplicació. Es la part responsable de la gestió de la informació. S'inclouen les classes, llibreries que permetran l'accés a les dades, així com el modelatge de les mateixes. Algunes de les tecnologies que s'utilitzen son JDBC, EJB, HIBERNATE, ...

CONTROLADOR.
El formen els components responsables de la gestió d'esdeveniments i de coordinar les activitats del model i de la vista.

Es el responsable de processar les peticions, determinant quina classe es la que ha de respondre.
Es responsable de modificar el model en funció dels paràmetres que rep executant la lògica de negoci que el desenvolupador hagi definit.
Es responsable de redirigir a la vista en funció de la lògica de negoci.
Permet l'ús de tecnologies estàndards tals com Java Filters, Java Beans, ...

VISTA.
La formen els components que presenten les dades al client. S'inclouen les pàgines HTML i en general tots els elements de visualització de l'aplicació. S'alimentarà de la informació que el controlador ha captat del model.
Exemples de tecnologies que s'utilitzen: HTML, JSP, ...

Com s'acaba de comentar, en l'actualitat s'utilitza el model de disseny de tipus MVC pel desenvolupament d'aplicacions Web. Una de les eines que permet optimitzar els temps de treballs es la utilització de frameworks *(WebWork, Spring, JSF, Struts 1, Struts 2, ...)*.
Per desenvolupar aquest projecte ens ajudarem del framework ***STRUTS 2*** [2], el qual ens facilitarà tasques com per exemple:

- *Validació de les dades.*

- *Buscar quina acció s'ha de realitzar en el model segons la petició.*

- *Buscar la pàgina que mostrarem a l'usuari segons la resposta del model.*

...

3.2. FRAMEWORK STRUTS 2.

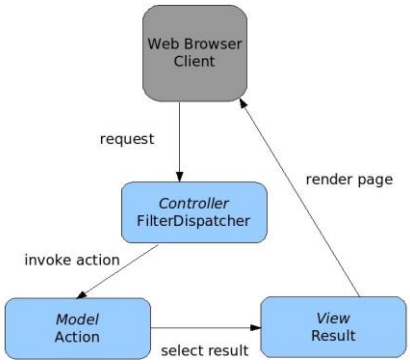
Pensat per facilitar les tasques pròpies del cicle de vida del software, incloent la construcció, desenvolupament i manteniment de la solució.

Per què utilitzar Struts 2?

- Es fàcilment extensible ja que tots els elements estan basats en interfaces i classes bases.
- Inclou un conjunt de llibreries desenvolupades per facilitar la creació d'aplicacions web dinàmiques, facilitant la creació d'elements de 'client' que permetin mostrar informació, validar-la, localitzar-la de forma senzilla i sense que impacti en els temps de desenvolupament, fent els desenvolupaments més senzills de mantindre.
- Qualsevol classe pot ser utilitzada com una 'Action Class' (POJO).
- Integració simple d'eines com JSTL, Spring o Hibernate.
- L'ús del llenguatge d'expressió OGNL.
- Suport d'AJAX. S'inclou un *theme Ajax* que permet fer aplicacions interactives de forma més senzilla.
- Facilitat per afegir plugins.
- Completa validació de suport. Admet una àmplia gamma de regles de validació.
- L'ús d'etiquetes permet aplicar temes o plantilles.
- Múltipes opcions de *vista* (JSP, FreeMarker, Velocity,...)

[2] Nota: Veure Annex 1 (Comparativa entre Struts 1 i Struts 2).

El framework Struts 2 està basat en el patró MVC. Com es tradueix en Struts 2 aquest patró?



Model: Les accions.

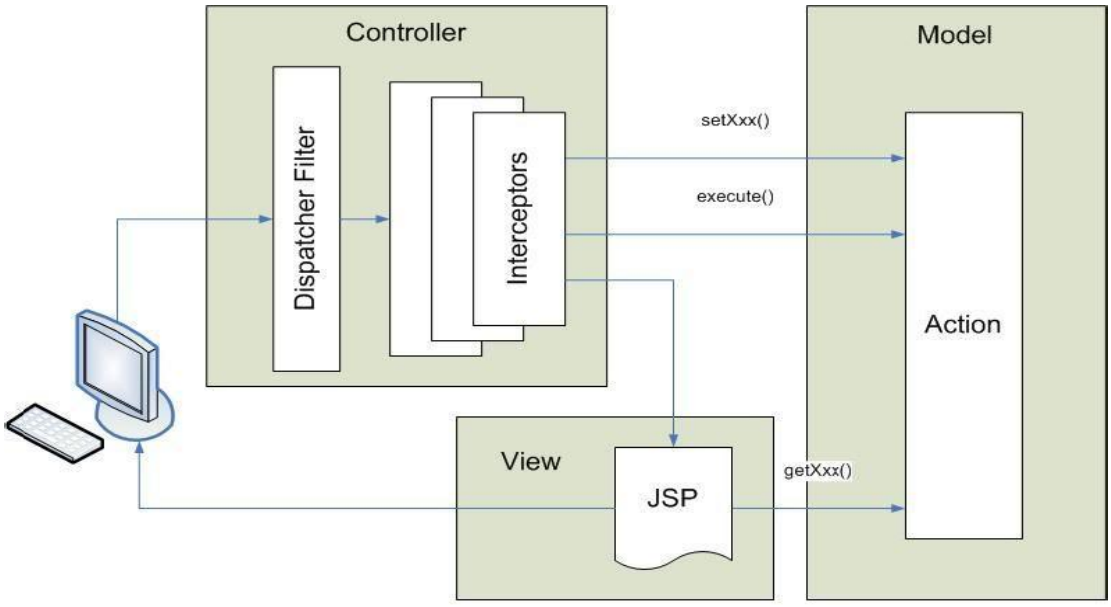
Els **ACTION** seran els encarregats d'executar la lògica necessària per manipular un request en concret.
Es basen en unes classes estàndards Java, anomenades **POJO** (Plain Old Java Object).

Vista: Els resultats.

Els **RESULT** són objectes que encapsulen el resultat d'un ACTION.

Controlador: S'implementa mitjançant el filtre FilterDispatcher.[3]

Serà el punt d'entrada a l'aplicació. A partir d'aquí es llança l'execució del processament per cada request que involucri al framework. S'ha de registrar en el fitxer web.xml.



ARQUITECTURA STRUTS 2
(Patró Model – Vista – Controlador)

[3] El filtre **FilterDispatcher** (org.apache.struts2.dispatcher.FilterDispacher) s'ha utilitzat en les versions anteriors a la versió 2.1.3 del framework Struts 2. Si s'utilitza una versió >= 2.1.3 es recomanable substituir-lo pel filtre **StrutsPrepareAndExecuteFilter.**
En la realització d'aquest projecte, he utilitzat la versió 2.1.8. Per tant, s'implementa el controlador mitjançant el filtre **StrusPrepareAndExecuteFilter.**

El framework Struts 2 es basa en una declaració de l'arquitectura en forma d'arxius XML o amb anotacions Java localitzades en els arxius de classes d'accions. Es un framework orientat a les accions. Les accions es descomponen en tres funcions:

- Encapsular el processament i el treball que haurà de realitzar el servei.
- Les accions permeten manipular automàticament les dades de les consultes durant les transferències.
- El framework determina quin resultat ha de ser retornat i la vista presentada en resposta a un processament.

Les accions de Struts 2 implementen objectes JavaBeans (classes Java simples) per a cada grup de dades enviat en la consulta. Cada paràmetre de la consulta es declara en la classe d'acció amb un nom idèntic per realitzar automàticament l'assignació de valors.

BIBLIOTECA D'ETIQUETES.

Struts 2 inclou una biblioteca d'etiquetes, per les pàgines JSP. Aquesta biblioteca es composa d'una primera categoria per la gestió de dades i d'una segona per les propietats, paràmetres, condicionals, ...

Per mostrar dades podem destacar les següents etiquetes:

Nom	Descripció
<s:form/>	Permet mostrar un formulari HTML.
<s:textfield/>	Permet crear un camp de text.
<s:password/>	Permet crear un camp tipus contrasenya.
<s:submit/>	Permet crear un botó de validació d'un formulari.
<s:textarea/>	Permet crear una etiqueta HTML de tipus camp de text de varies línies.
<s:select/>	Permet crear una llista desplegable HTML.

Per gestionar l'accés a les dades, condicionals, ... podem destacar les següents etiquetes:

Nom	Descripció
<s:property/>	Permet mostrar un formulari HTML.
<s:action/>	Permet crear un camp de text.
<s:param/>	Permet crear un camp tipus contrasenya.
<s:if>, <s:else/>	Permet crear un botó de validació d'un formulari.
<s:iterator/>	Permet crear una etiqueta HTML de tipus camp de text de varies línies.

Per utilitzar aquestes etiquetes en les pàgines JSP, s'ha d'incloure una declaració en la part superior de cada pàgina:

```
<%@ taglib prefix="s" uri="/struts-tags"%>
```

VALUE-STACK.

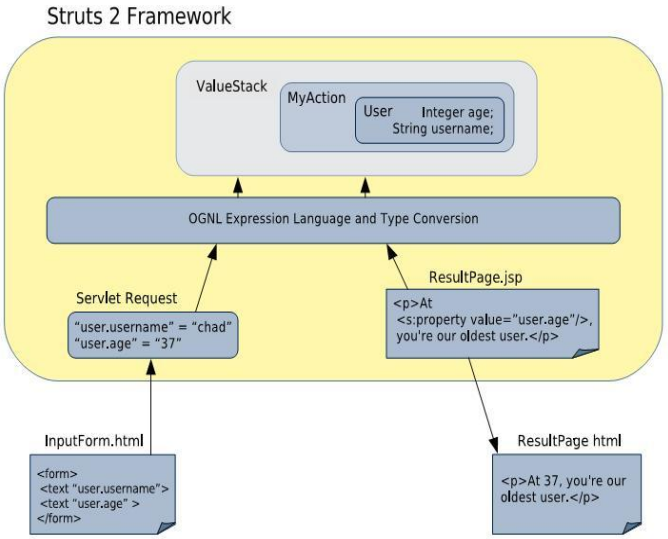
Struts 2 està proveït d'una pila d'objectes que s'anomena *VALUE-STACK* la qual permet recuperar els valors en la vista JSP amb independència del context (sessió, aplicació, request) en el qual està guardat el valor. El contingut d'aquesta pila està format per quatre nivells:

- *Objectes temporals (Objectes creats durant l'execució d'una JSP i ubicats dins de la Value-Stack).*
- *Objectes del model (Si es vol utilitzar els objectes del model, aquests seran ubicats en la Value-Stack abans que s'executi l'Action).*
- *Objectes de l'Action.*
- *Objectes nombrats (#application, #session, #request, #attr i #parameters).*

OGNL (Object-Graph Navigation Language)

És un llenguatge de descripció que permet accedir i modificar les propietats dels objectes Java. OGNL fa referència als objectes en funció del seu àmbit d'aplicació i de la col·lecció de dades (Context Map): *application, session, request, parameters, attr.*

El framework Struts 2 utilitza aquest llenguatge d'expressions el qual ens proporciona un mecanisme per poder navegar sobre els diferents objectes de la pila, utilitzant una notació de punt, expressions i crides a mètodes dels objectes.



Les aplicacions de Struts 2 tenen un fitxer de configuració anomenat *struts.xml.* Aquest fitxer es el més important i substitueix a la definició dels *servlets* en el descriptor de la implementació *web.xml.* Per tal d'aprofundir una mica en l'arquitectura d'Struts 2, he considerat oportú explicar més en detall el contingut d'aquest fitxer.

3.3. FITXER STRUTS.XML

En aquest fitxer, es defineix la configuració general de l'aplicació:

- Paràmetres de configuració d'Struts.
- Les accions.
- Els resultats de les accions.
- Els interceptors.

ETIQUETA <package/>

Les accions s'agrupen per paquets. Els atributs principals d'aquesta etiqueta són:
- name: Atribut obligatori que determina el nom del paquet.
- namespace: Atribut opcional que determina l'espai de nom per a totes les accions del paquet.
- extends: Atribut obligatori que determina el paquet 'pare'.

Exemple:

```
<package name="com" namespace="/" extends="struts-default">
        ...
</package>
```

ETIQUETA <include/>

En una aplicació podem tenir diferents paquets i una gran quantitat de línies en aquest fitxer de configuració. Per millorar l'administració i el manteniment d'aquest fitxer es possible dividir-lo en diferents subfitxers. L'etiqueta <include/> permet fer aquesta divisió, l'habitual serà que cada subfitxer defineixi el seu propi paquet.

Exemple:

Fitxer struts.xml

```
        ...
        <struts>
                <include file="nom_subfitxer.xml"/>
                ...
        </struts>
```

Fitxer nom_subfitxer.xml

```
        ...
        <struts>
                <package name....       />
                ...
        </struts>
```

ETIQUETA <action/>

Una acció està, en general, associada a una classe, però no és obligatori. Una acció que no requereix de classe utilitza una instància de la classe per defecte, *ActionSupport.* Suposant que s'ha definit una classe, la sintaxi serà:

```
<action name= "nom_accio" class = "nompaquet.nomclasse" method = "nom_metode" />
```

Exemple:

```
<action name="DoLogin" class="com.UsuariAccion" method="Validar_Usuari">
```

name="DoLogin": identificador mitjançant el qual es crida a l'acció des de la 'Vista' (fitxers .JSP).
class="com.UsuariAccion": controlador encarregat de realitzar la lògica de l'acció.

method="Validar_Usuari": mètode que es cridarà de la classe UsuariAccion. Si no s'especifica cap mètode, per defecte es cridarà al mètode execute().

ETIQUETA <result/>

S'utilitza en l'interior d'una etiqueta <action/> com sub-element. Aquesta etiqueta indicarà al framework Struts on retornar el resultat de l'acció. Els atributs d'aquesta etiqueta són *name* i *type*.
L'atribut *name* permet donar un nom al resultat que correspondrà al valor *return* de l'acció (el valor per defecte d'aquest atribut es *success).*
L'atribut *type* permet definir el tipus de resultat. El valor per defecte és *dispatcher* (redirecció simple).
Una acció pot retornar diferents resultats, per tant podrà haver-hi diferents etiquetes <result/>, cada una correspondrà a un resultat possible. Per exemple, una acció pot retornar una pàgina d'error en cas d'existir un problema en les dades introduïdes (input) o una pàgina de confirmació per mostrar el resultat (success).

Exemple:

```
<action name="Visita_Pacient" class="com.CitaAccion" method="Actualitzar_Visita_Pacient">
        <result name="input" type = "dispatcher">
                jsp/Principal_Metge.jsp
        </result>
        <result name="success" type = "dispatcher">
                jsp/Pagina_Missatges.jsp?iden=VISITAPACIENT
        </result>
</action>
```

Tipus de resultats possibles que podem indicar en l'atribut *type:*

- *dispatcher:* correspon a una redirecció a una pàgina JSP (en la majoria de casos) i es el més utilitzat. Es tracta d'un redireccionament sense pèrdua de paràmetres però no permet realitzar redirecció a un recurs extern o una URL absoluta.
  Es tracta del valor per defecte i per tant, es pot obviar.

```
...
<result name="input">
        jsp/Principal_Metge.jsp
</result>
...
```

- *redirect:* permet realitzar una redirecció a un recurs intern o bé extern, però amb pèrdua de paràmetres.

```
...
<result type = "redirect">
        http://www.google.es
</result>
...
```

  També podem transmetre paràmetres a una redirecció utilitzant les propietats. La sintaxis ${propietat} permet transmetre les dades presents a la CLASS ACTION amb els seus descriptors d'accés.

```
...
<result name = “success” type = “redirect”>
        jsp/MostrarDadesPacient.jsp?identificador=${identificador}
</result>
...
```

- *redirectAction:* S’utilitza per realitzar una redirecció cap a una altra acció de manera similar al tipus *redirect (amb pèrdua de paràmetres)*

```
...
<result name = "SUCCESS_ADMIN" type = "redirectAction" >
        Inicial_Admin
</result>
...
```

  També podem transmetre paràmetres:

```
...
<result name = "success" type = "redirectAction" >
        <param name = “actionName”> AfegirPacient </param>
        <param name =  “identificador”> ${identificador} </param>
</result>
...
```

- *chain:* Redirecció cap una altra acció conservant l’estat de l’acció original en l’acció de destí. Aquest tipus de redirecció permet realitzar un encadenament sense pèrdua de paràmetres *(Es recomanable utilitzar el tipus redirectAction).*[4]

```
...
<result name = "success" type = "chain" >
        Mostrar_Llistat
</result>
...
```

- Altres tipus de resultats:
  - *freemarker:* Permet utilitzar el motor de plantilles FreeMarker.
  - *velocity:* Permet utilitzar el motor de plantilles Velocity.
  - *httpheader:* Permet retornar una capçalera HTTP configurada.
  - *stream:* Permet retornar un resultat Stream al navegador (vídeo, imatge,...)
  - *xslt:* Permet utilitzar resultats XML i fulles d’estil XSLT.
  - *plaintext:* Permet retornar el resultat en format de text per visualitzar el codi font

ETIQUETA <param/>

S’utilitza amb els elements <action/>, <result-type/> e <interceptors/> per assignar un valor a un objecte afectat. L’etiqueta <param/> conté un atribut *name* per nombrar el valor. Si s’utilitza amb una acció, l’etiqueta <param/> permet assignar un valor a una propietat.

Exemples:

```
<action name = "userImage" class = "com.ImatgesAccion">
        <interceptor-ref name="fileUpload">
                <param name="maximumSize">2097152</param>
```

[4] http://struts.apache.org/2.0.14/docs/action-chaining.html

```
        </interceptor-ref>
        ...
<action name="Afegir_Usuari" class ....>
        <param name = "identificador">David</param>
</action>
```

ETIQUETA <constant/>

Per administrar la configuració d'Struts, podem utilitzar el fitxer struts.properties (els valors per defecte es presenten en el fitxer default.properties) o bé utilitzar l'etiqueta <constant/> present en el fitxer struts.xml.

Exemple:

Si volem utilitzar Struts 2 en mode de desenvolupament per l'administració dels registres i depuracions, utilitzarem la següent configuració:

```
        <constant value="true" name="struts.devMode"/>
```

Utilitzant el fitxer struts.properties (s'hauria de crear en la carpeta WEB-INF/src),

```
        struts.devMode = true
```

ETIQUETA <global-results/>

Si una acció no troba una definició de resultat en la seva pròpia definició, Struts 2 buscarà el resultat en la definició d'aquesta etiqueta <global-results/>. Si no troba cap correspondència ja sigui de manera local o bé dins d'aquesta etiqueta 'global', es produirà una excepció.

Exemple:

```
<global-results>
        <result name="login" type="redirectAction">TancarSessio</result>
</global-results>
```

ETIQUETA <interceptors/>

S'utilitza per definir els interceptors. Degut a la seva importància, anem a explicar-ho una mica més en detall. Mitjançant els interceptors podrem executar codi abans i després de l'execució d'un ACTION [5], això ens permet afegir funcionalitat als ACTIONS de la nostra aplicació.
Struts 2 es basa en una llista d'interceptors encarregats de prestar serveis específics per la gestió de paràmetres, validació de formularis, ...
Un interceptor no es més que una classe Java que implementa la interface *com.opensymphony.xwork2.Interceptor.*

Mètodes que ofereix aquesta interface i que l'interceptor haurà d'implementar:

void **init ( ) :** Es crida al mètode init ( ) abans que s'hagi creat l'interceptor per inicialització dels recursos i únicament quan l'aplicació estigui carregada.

[5] http://struts.apache.org/2.0.11/docs/interceptors.html

void **destroy ( ) :** Es crida després de que s'hagi executat l'interceptor per alliberar recursos i únicament quan l'aplicació estigui descarregada.

String **intercept (ActionInvocation actionInvocation)** throws Exception: És el mètode que conté la lògica a executar abans i després. El framework crida al mètode intercept() de cada interceptor declarat per una acció. L'objecte de la classe ActionInvocation representa l'estat de l'acció i permetrà recuperar els objectes *Action i Result* associats.

Per utilitzar un interceptor, ha d'estar declarat en l'etiqueta <interceptors/>.

Exemple:

```
<interceptors>
<interceptor name="Autenticacion" class="com.interceptors.Autenticacion_Interceptor"/>
<interceptor name="roles" class="com.interceptors.RolesInterceptor"/>...
</interceptors>
```

Una vegada registrat, podrem utilitzar-lo amb l'etiqueta <interceptor-ref/> que es un sub-element de l'etiqueta <action/>

```
<action name="nom_accio" class ....>
        <interceptor-ref name = "Autenticacion"/>
        <interceptor-ref name = "roles"/>
        <result> ...
</action>
```

NOTA IMPORTANT: L'ordre de declaració dels interceptors té importància, determinarà l'ordre d'execució dels interceptors per a cada acció. En aquest exemple, l'interceptor *Autenticacion* s'executarà en primer lloc.

Per evitar repetir declaracions d'interceptors, Struts 2 ofereix la possibilitat de crear grups (piles) d'interceptors, no caldrà realitzar múltiples referències per cada etiqueta <action/> que utilitza els mateixos interceptors.

Seguint amb l'exemple anterior,

```
        <interceptor-stack name="secureStack">
                <interceptor-ref name="i18n"/>
                <interceptor-ref name="Autenticacion"/>
                <interceptor-ref name="defaultStack"/>
        </interceptor-stack>
        <interceptor-stack name="secureStack_METGE">
        <interceptor-ref name="secureStack"/>
                <interceptor-ref name="roles">
                        <param name="allowedRoles">METGE</param>
                </interceptor-ref>
        </interceptor-stack>
        ...
```

Per fer referència a un grup d'interceptors, només cal afegir l'etiqueta <interceptor-ref/> i l'atribut *name* dins d'una acció.

```
        <action name="Inicial_Metge" class="... >
                <interceptor-ref name="secureStack_METGE"/>
```

```
        <result name="success">jsp/Principal_Metge.jsp</result>
    </action>
```

Existeix el paquet **struts-default** el qual inclou una sèrie d'interceptors per defecte [6]. Un dels més importants es l'interceptor *params (responsable de realitzar el 'mapping' entre els camps dels formularis i els seus descriptors d'accés respectius. Les cadenes de caràcters rebuts pel protocol HTTP es converteixen automàticament al tipus definit en la CLASS ACTION).*
L'etiqueta <*default-interceptor-ref/*> especifica el grup d'interceptors per defecte:

```
<default-interceptor-ref name = "defaultStack"/>
```

EXEMPLE D'AUTENTICACIÓ D'USUARIS UTILITZANT INTERCEPTORS.

Cas estudiat:
Suposem que *un usuari tipus Pacient* veu l'historial del navegador i copia una URL d'accés a les opcions d'Administrador.

Per tal de garantir l'accés al sistema segons el perfil de cada usuari, he creat un interceptor anomenat ***roles*** (declarat en el fitxer struts.xml).

```
...
<interceptors>
        <interceptor name="roles" class="com.interceptors.RolesInterceptor"/
        ...

        <interceptor-stack name="secureStack_ADMIN">
                <interceptor-ref name="roles">
                        <param name="allowedRoles">ADMIN</param>
                </interceptor-ref>
        </interceptor-stack>
```

Qualsevol ACTION que inclogui aquest interceptor només continuarà el seu processament en cas que l'usuari carregat en la sessió estigui inclòs en la llista de valors que formen part del paràmetre *allowedRoles.*

```
<action name="List_Pacient" class="com.UsuariAccion" method="llistat">
        <interceptor-ref name="secureStack_ADMIN"/>
        ...
</action>
```

Si copiem *http://localhost:8080/com/List_Pacient* en el navegador, només processarà el resultat de l'ACTION *List_Pacient* si l'usuari carregat en la sessió té permisos d'ADMIN.

```
public class RolesInterceptor extends AbstractInterceptor
{

        private List<String> allowedRoles = new ArrayList<String>();

        public void setAllowedRoles(String roles)
        {
```

[6] Nota: Veure Annex 2 (Interceptors).

```
                this.allowedRoles = stringToList(roles);
        }

        public String intercept(ActionInvocation invocation) throws Exception
        {
                Map session = invocation.getInvocationContext().getSession();
                Usuari usuari = (Usuari) session.get("usuari");

                if (!isAllowed(invocation.getAction(),usuari))
                {
                        return Action.LOGIN;
                        /* L'interceptor ha detectat un accés inapropiat. Invocarà a l'ACTION
                        corresponent i retornarà a la pàgina inicial de l'aplicació. */
                }
                else
                {
                        return invocation.invoke();          /* Continuarà el procés normal */
                }
        }
        ...
}
```

3.4. FITXER WEB.XML

El projecte també inclou un arxiu descriptor encarregat de la configuració de l'aplicació Web. En aquest fitxer, haurem d'agregar el filtre que gestiona les peticions a Struts 2:

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-
class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

L'element <filter> permet declarar la classe associada al filtre.
L'element <filter-mapping> permet administrar el 'mapping' entre un nom i una URL.

D'aquesta manera, estarem indicant al servidor web Tomcat que totes les peticions han de ser interceptades i gestionades pel filtre StrutsPrepareAndExecuteFilter *(Aquest filtre substitueix al filtre FiterDispatcher).*

En aquest fitxer, també podrem configurar paràmetres d'inicialització del context de la nostra aplicació.

```
<!-- Paràmetres globals -->
<context-param>
        <param-name>dataSourceJNDI</param-name>
        <param-value>java:/comp/env/jdbc_centremedic_MySQL</param-value>
</context-param>
```

Ens permet definir una classe 'escoltadora'. En aquest cas, per aplicar la connexió al gestor de bases de dades utilitzarem un *listener* aplicat a l'iniciar l'aplicació.

```
<listener>
        <listener-class>com.Utils.ApplicationListener</listener-class>
</listener>
```

Definició del 'timeout' de la sessió en minuts.

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
```

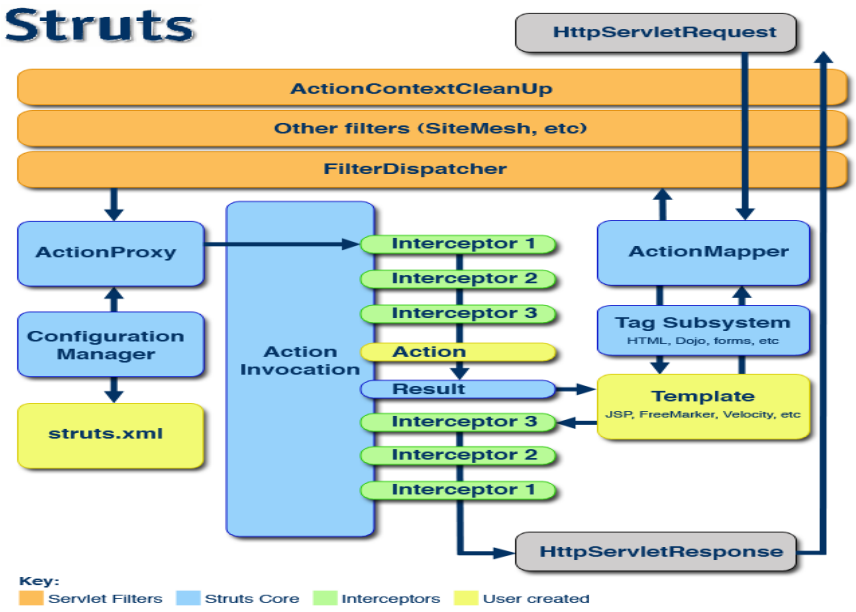
Ens permet declarar recursos externs que utilitzarem en l'aplicació, en aquest cas la connexió amb la base de dades.

```
<resource-ref>
        <description>Conexión a la base de datos MySQL</description>
        <res-ref-name>jdbc_centremedic_MySQL</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
</resource-ref>
```

Existeixen més elements, com <welcome-file-list, <error-page, ... però no entro en detall perquè no els he utilitzat en la realització d'aquest projecte.

CICLE DE VIDA D'UNA PETICIÓ.

En el següent diagrama[7] podem veure de forma gràfica quins elements són els que intervenen dins del processament d'una petició (HTTP), diferenciant quins són elements estàndards de J2EE, quins són elements propis de STRUTS 2 i quins són creats per l'usuari.



[7] Enllaç diagrama: http://struts.apache.org/2.x/docs/big-picture.html

1. Usuari envia la petició (request).

2. Aquesta petició arriba a un SERVLET (HttpServletRequest). Part del controlador que rebrà la petició.

3. El servlet delega en l'ActionMapper per determinar el ACTION que s'executarà.

4. El filtre FilterDispatcher (*StrutsPrepareAndExecuteFilter en el nostre cas*) determina el ACTION que haurà de respondre, és a dir, és l'encarregat de determinar si per resoldre la petició es requereix l'execució o no d'un ACTION. En el cas de que s'hagi d'utilitzar un ACTION, es delega en l'ActionProxy (El framework Struts 2 disposa dels elements per tals que el dispatcher sigui capaç de determinar quin ACTION es el responsable de rebre la petició i processar-lo).

5. Tal com hem dit, si la petició requereix l'execució d'un ACTION, s'executarà l'ActionProxy el qual utilitzarà el que s'anomena Gestor de Configuracions per a crear l'ActionInvocation. El gestor de configuració és l'element encarregat de llegir el fitxer de control (Struts.xml), analitzar-lo i en cas que sigui correcte, desplegar els elements que s'hi referencien.

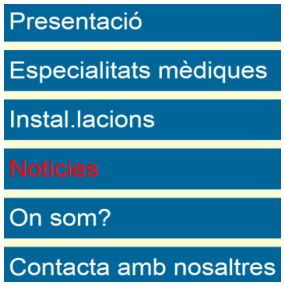
6. Acte seguit, s'invoca a l'objecte de tipus ActionInvocation, responsable d'executar cada un dels interceptors i després executarà el ACTION corresponent, el qual contindrà la lògica de negoci que s'ha d'executar *(Sabrem quina classe s'ha d'executar així com els interceptors que hi intervindran gràcies a que abans s'ha fet la lectura del fitxer de configuració).*
   Tal com ja hem vist abans, els interceptors són classes que s'utilitzen per especificar el cicle de vida d'una petició i estan pensats per afegir funcionalitat extra als ACTION, existeixen diferents interceptors que es poden configurar per l'execució de diferents funcionalitats com workflows, validacions dels paràmetres d'entrada, ... Aquests interceptors permetran realitzar operacions abans i després de la invocació d'un ACTION.

7. Una vegada executat l'ACTION, s'obtindrà un resultat (RESULT), el qual s'utilitzarà per a determinar quins dels elements de la vista s'han de cridar (pàgina a retornar).

8. S'executa l'enllaç cap a la pàgina en concret i es retorna la petició. Per a realitzar la devolució, s'executen els interceptors que corresponguin.

9. Es mostra el resultat a l'usuari final, retornant el control a l'usuari, el qual podrà visualitzar el resultat en el seu navegador.

Anem a veure-ho en un cas pràctic basant-nos en el nostre projecte:



Suposem que un usuari vol veure les noticies relacionades amb el centre mèdic:
http://localhost:8080/com/Llistar_Noticies

- En la pàgina .jsp corresponent tenim el següent codi:

```
<a href="Llistar_Noticies"> <s:text name="label.noticies"/> </a>
```

- En el fitxer de configuració struts.xml:

```
<action name="Llistar_Noticies" class = "com.NoticiaAccion" method = "llistat">
        <result>jsp/Presentacio_Noticies.jsp</result>
</action>
```

Quan l'usuari seleccioni l'opció *'Noticies'*, estarà invocant a l'acció *'Llistar_Noticies'* la qual executarà el mètode 'llistat' que es troba ubicat en la classe *'NoticiaAccion'*. En aquest exemple, no tenim interceptors. Per tant, directament s'executarà l'ACTION.

- En el fitxer NoticiaAccion.java tenim:

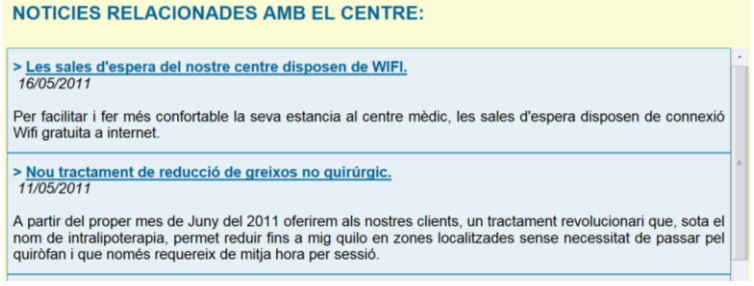
```
public String llistat( ) {
        ModelNoticiaDAO modelNoticiaDAO=new ModelNoticiaDAO( );
        listaNoticias=(ArrayList<Noticia>)modelNoticiaDAO.getLlistaNoticies( );
        return SUCCESS; }
```

Una vegada executat l'ACTION, s'obté un resultat (en aquest cas *'SUCCESS'*) el qual s'utilitza per determinar quin element de la vista s'ha de retornar (en el nostre exemple: Presentacio_Noticies.jsp).

- En el fitxer Presentacio_Noticies.jsp tenim:

```
...
<s:iterator value="listaNoticias">
<div id = Noticia>
                <font color ="#006699"/>
        <b>> <u><s:property value="Titol"/></u></b>
        <font color ="black"/>
        <br>
         <i><s:property value="Data_Noticia"/></i>
        <br>
        <br>
        <s:property value="Descripcio"/>
        </div>
</s:iterator>
...
```

Resultat final mostrat en el navegador.



3.5. CAPA MODEL i ACCÉS A LES DADES.

3.5.1. CREACIÓ DE LA BASE DE DADES.

La creació de la base de dades és una tasca específica del sistema administrador de bases de dades, en el nostre cas treballarem amb MySQL.

- SCRIPT SQL DE CREACIÓ DE LA BASE DE DADES I DE LES TAULES CORRESPONENTS.

```
CREATE DATABASE `centremedic`;

USE `centremedic`;


-- Table structure for table `usuari`

DROP TABLE IF EXISTS `usuari`;
CREATE TABLE `usuari` (
  `int_User` int(11) NOT NULL AUTO_INCREMENT,
  `Num_Identificatiu` varchar(45) NOT NULL,
  `Rol_Usuari` varchar(45) NOT NULL,
  `Activat` smallint(6) NOT NULL,
  `Password` varchar(45) NOT NULL,
  PRIMARY KEY (`int_User`),
  UNIQUE KEY `Num_Identificatiu_UNIQUE` (`Num_Identificatiu`)
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=latin1;


-- Table structure for table `especialitat`

DROP TABLE IF EXISTS `especialitat`;
CREATE TABLE `especialitat` (
  `idEspecialitat` int(11) NOT NULL AUTO_INCREMENT,
  `Nom` varchar(45) NOT NULL,
  `Descripcio` varchar(300) NOT NULL,
  `Nom_Metge` varchar(45) NOT NULL,
  `Cognom_Metge` varchar(45) NOT NULL,
  `Num_Identificatiu` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idEspecialitat`),
  UNIQUE KEY `idEspecialitat_UNIQUE` (`idEspecialitat`),
  KEY `fk_Especialitat_Usuari1` (`Num_Identificatiu`),
  CONSTRAINT `fk_Especialitat_Usuari1` FOREIGN KEY (`Num_Identificatiu`)
REFERENCES `usuari` (`Num_Identificatiu`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=38 DEFAULT CHARSET=latin1;


-- Table structure for table `noticies`

DROP TABLE IF EXISTS `noticies`;
CREATE TABLE `noticies` (
  `idNoticies` int(11) NOT NULL AUTO_INCREMENT,
  `Titol` varchar(100) NOT NULL,
  `Descripcio` varchar(300) NOT NULL,
  `Data_Noticia` datetime DEFAULT NULL,
  PRIMARY KEY (`idNoticies`)
) ENGINE=InnoDB AUTO_INCREMENT=107 DEFAULT CHARSET=latin1;
```

```
-- Table structure for table `pacient`

DROP TABLE IF EXISTS `pacient`;
CREATE TABLE `pacient` (
  `idPacient` int(11) NOT NULL AUTO_INCREMENT,
  `NomPacient` varchar(45) NOT NULL,
  `Cognom1` varchar(45) NOT NULL,
  `Cognom2` varchar(45) NOT NULL,
  `Sexe` varchar(1) NOT NULL,
  `Email` varchar(45) NOT NULL,
  `Telefon` varchar(45) DEFAULT NULL,
  `Direccio` varchar(45) DEFAULT NULL,
  `CP` varchar(45) DEFAULT NULL,
  `Poblacio` varchar(45) DEFAULT NULL,
  `Pacient_desde` datetime DEFAULT NULL,
  `Num_Identificatiu` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idPacient`),
  UNIQUE KEY `idPacient_UNIQUE` (`idPacient`),
  KEY `fk_Pacient_Usuari1` (`Num_Identificatiu`),
  CONSTRAINT `fk_Pacient_Usuari1` FOREIGN KEY (`Num_Identificatiu`) REFERENCES
`usuari` (`Num_Identificatiu`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=26 DEFAULT CHARSET=latin1;

-- Table structure for table `relacio_pacient_especialitat`

DROP TABLE IF EXISTS `relacio_pacient_especialitat`;
CREATE TABLE `relacio_pacient_especialitat` (
  `idRelacio` int(11) NOT NULL AUTO_INCREMENT,
  `estat_visita` bit(1) DEFAULT NULL,
  `data_visita` datetime DEFAULT NULL,
  `motiu_visita` varchar(45) DEFAULT NULL,
  `recomanacio` varchar(150) DEFAULT NULL,
  `hora_visita` varchar(10) DEFAULT NULL,
  `idEspecialitat` int(11) DEFAULT NULL,
  `idPacient` int(11) DEFAULT NULL,
  PRIMARY KEY (`idRelacio`),
  UNIQUE KEY `idRelacio_UNIQUE` (`idRelacio`),
  KEY `fk_Relacio_Pacient_Especialitat_Especialitat` (`idEspecialitat`),
  KEY `fk_Relacio_Pacient_Especialitat_Pacient1` (`idPacient`),
  CONSTRAINT `fk_Relacio_Pacient_Especialitat_Especialitat` FOREIGN KEY
(`idEspecialitat`) REFERENCES `especialitat` (`idEspecialitat`) ON DELETE
CASCADE ON UPDATE NO ACTION,
  CONSTRAINT `fk_Relacio_Pacient_Especialitat_Pacient1` FOREIGN KEY
(`idPacient`) REFERENCES `pacient` (`idPacient`) ON DELETE CASCADE ON UPDATE
NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=38 DEFAULT CHARSET=latin1;
```

Com podem veure es tracta d'una base de dades molt senzilla. Tot seguit s'explica la funcionalitat de cadascuna de les taules.

Taula Usuari.

Contindrà els diferents actors que intervenen en l'aplicació. El camp *Rol_Usuari* ens indicarà el tipus d'usuari: *ADMIN, METGE o bé PACIENT.*

L'administrador serà l'encarregat de gestionar els usuaris tipus *Metge (Especialitat)* i també podrà gestionar els usuaris tipus *Pacient.*[8]
S'ha creat un camp anomenat *Activat* per tal de permetre l'accés o no als usuaris tipus *Pacient (Suposant el cas que l'Admin vol restringir l'accés a l'aplicació a un Pacient però sense eliminar-lo de la BBDD).*

Taula Pacient.

S'actualitzarà amb la informació corresponent al pacient. Tant l'administrador com el propi pacient podran gestionar les dades des de l'aplicació.

Taula Especialitat.

He decidit que la relació Especialitat - Metge es 1:1. Per tant, no he creat cap taula *'Metge'* i he ficat les dades del metge com camps dins de la taula Especialitat.

Taula Relacio_Pacient_Especialitat.

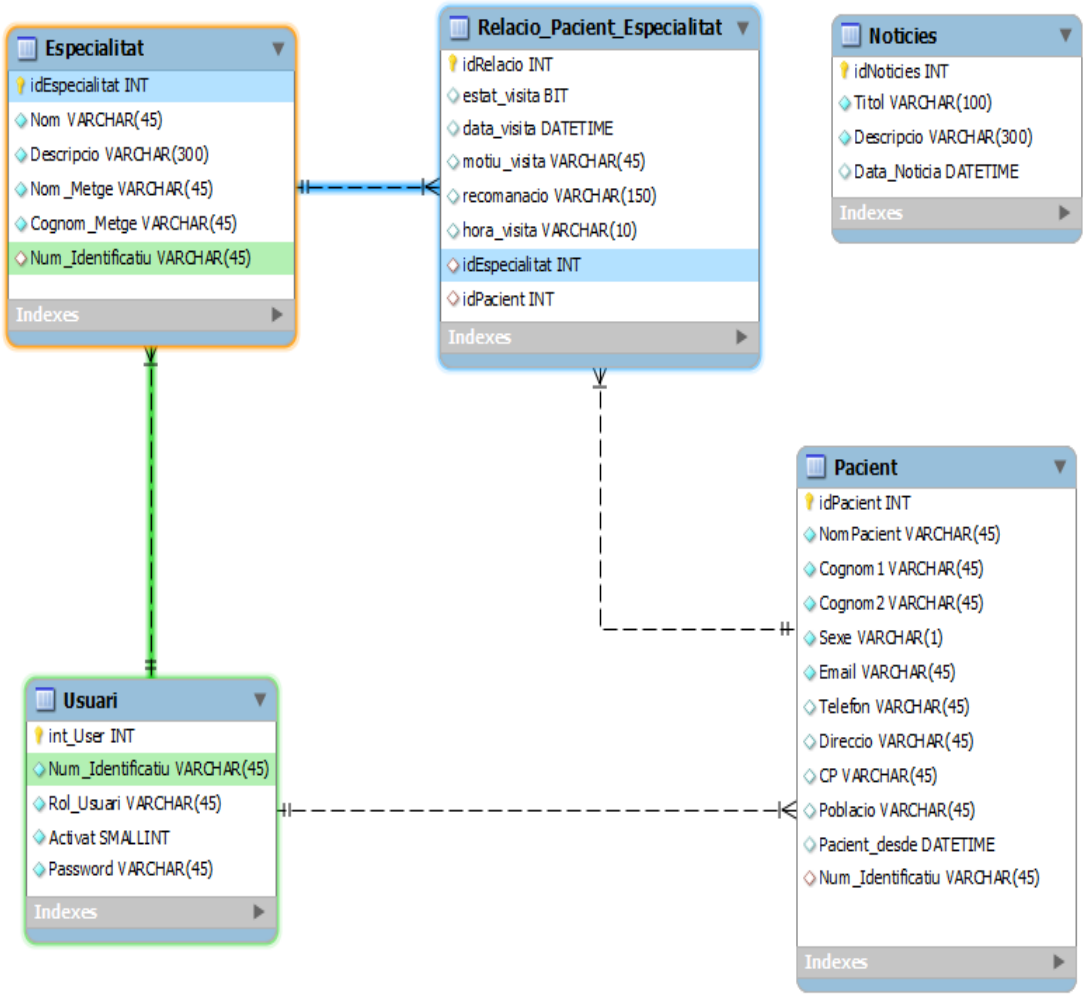
Tal com diu el seu nom, contindrà les relacions existents entre pacient i especialitat.

Taula Noticies.

Taula totalment independent on s'inclouran les noticies relacionades amb el centre. L'usuari tipus Admin gestionarà els continguts d'aquesta taula.

[8] Si s'elimina un usuari tipus Pacient de la taula *Usuari*, automàticament s'eliminarà 'en cascada' de la taula *Pacient*. De la mateixa manera, s'eliminaran automàticament totes les relacions que tingui aquest usuari en la Taula *Relacio_Pacient_Especialitat*.
Si s'elimina un usuari tipus Metge de la taula *Usuari*, primer es comprovarà que no hi hagi relacions creades amb pacients. En cas que no existeixi cap relació, s'eliminarà 'en cascada' de la taula *Especialitat.*

3.5.2. DISSENY DE LA BASE DE DADES (Model Relacional).



Una vegada tenim creada la base de dades, anem a veure com accedir a les dades.

3.5.3. ACCÉS I MANIPULACIÓ DE DADES.

Les Bases de Dades s'utilitzen per garantir la persistència de les dades. En JAVA, per a gestionar la persistència de les dades s'utilitza els models:

- Objecte – Relacional (Objectes JAVA cap a Base de Dades).
- Relacional – Objecte (Base de Dades cap a Objectes JAVA).

Aquestes transformacions s'anomenen *mapping*. Actualment existeixen moltes eines (més o menys funcionals) per portar a terme aquests serveis. Els més coneguts són la llibreria *Open-Source Hibernate* i *Java Persistence API (JPA)*. Sense necessitat d'utilitzar una eina de *mapping*, podem utilitzar el model *Data Access Object (DAO)*.
El model o capa de persistència ens ofereix un conjunt de mètodes per consultar, agregar, modificar, eliminar o realitzar un llistat dels objectes (imatges dels valors de la base de dades).

3.5.4. EL MODEL DATA ACCESS OBJECT (DAO).

Aquest model facilita un conjunt de regles que han de seguir-se per a configurar un sistema de persistència d'objectes. Es molt útil per abstraure i encapsular tots els accessos a la base de dades.

El model DAO de base s'utilitza per a cada classe d'acció de STRUTS que ha de portar a terme una operació en la base de dades. Amb aquest model, cada classe utilitzada en el sistema ha de tindre la seva pròpia classe model per gestionar la seva persistència (mètodes comuns i la declaració de les connexions al SGBD). La interface DAO s'implementa per la classe DAO principal que s'haurà de sobrecarregar utilitzant el mètode getConnection ( ).

Característiques principals:

- Separar l'accés a dades de la lògica de negoci.

- Encapsula la font de dades.

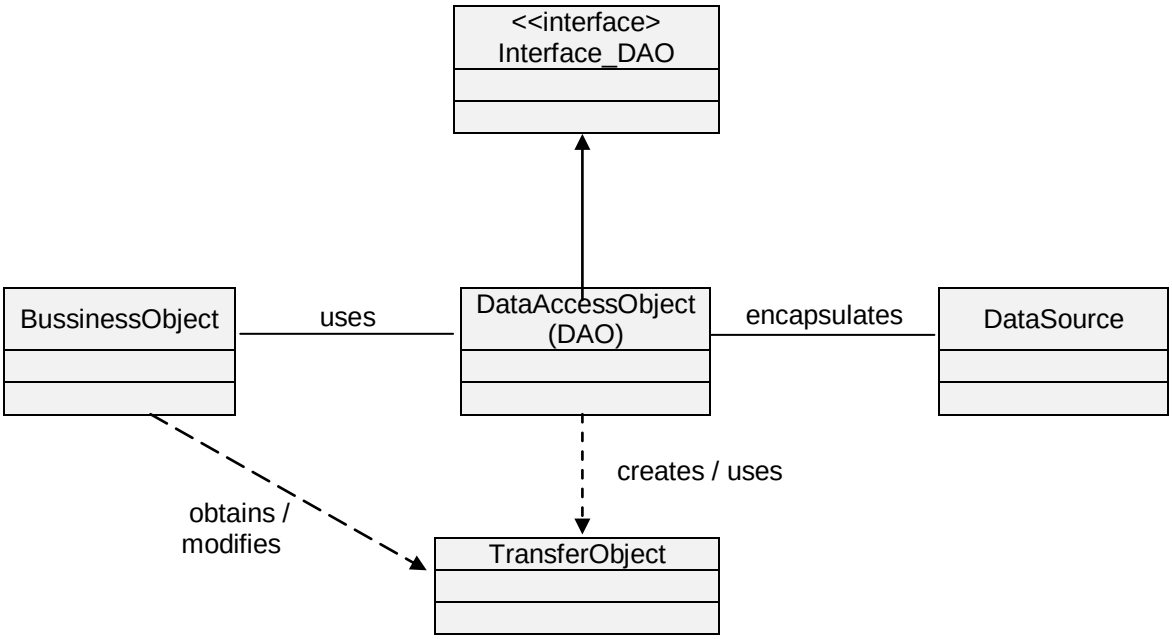
- Centralitza tots els accessos a dades en una capa independent.

Les aplicacions poden utilitzar l'API JDBC per accedir a les dades d'una base de dades relacional, aquest API permet una forma estàndard d'accedir i manipular dades en una base de dades relacional, permet utilitzar sentències SQL, que són el mètode estàndard per accedir a taules i vistes. La idea d'aquest patró DAO es ocultar la font de dades i la complexitat de l'ús de JDBC a la capa de presentació o bé de negoci.

L'avantatge d'utilitzar objectes d'accés a dades es que qualsevol objecte de negoci no requereix coneixement directe del destí final de la informació que manipula, d'aquesta manera si es vol canviar el motor de la base de dades (per exemple de MySQL a PostgreSQL), aplicant el patró de disseny DAO no tindrem massa problemes per realitzar-ho.

Quan la capa de lògica de negoci necessita interactuar amb la base de dades, ho fa a través de l'API que li ofereix DAO. Generalment, aquest API consisteix en mètodes CRUD (Create, Read, Update i Delete). El patró DAO defineix la relació entre la lògica de presentació i negoci per una part i per l'altra les dades. El patró DAO té una interface comú, sigui quin sigui el mode i font d'accés a les dades.

- Diagrama de classes representant les relacions existents del patró DAO:



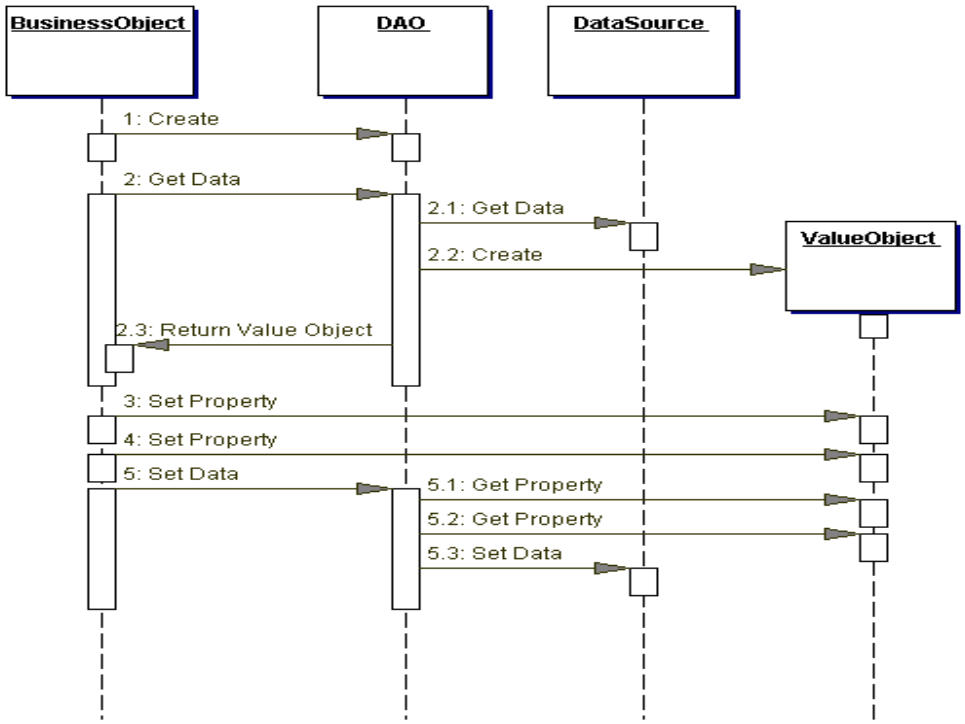
BussinessObject: Representa les dades del client. Es l'objecte que requereix l'accés a la font de dades per obtindre i emmagatzemar dades.

DataAccessObject: Objecte principal d'aquest patró. El DAO accedeix a la font de dades i encapsula per als objectes client tant la font com el mode (JDBC) d'accedir-hi, d'aquesta manera l'accés per part del BusinessObject a la font de dades és transparent.

DataSource: Representa la implementació de la font de dades.

TransfertObject: Objecte intermig entre el BusinessObject i el DataAccesObject. També anomenat ValueObject encapsula una unitat d'informació de la font de dades, es com una representació d'una taula de la base de dades. D'aquesta manera representem les columnes de la taula com atributs del TransferObject.

- Diagrama de seqüència del patró DAO:



En el diagrama anterior, es mostren les interaccions entre els elements que formen el patró, podem observar les crides que rep i genera el DAO per una consulta i actualització de dades.

1. El client (BusinessObject) crea el DAO (Punt 1).

2. El client sol·licita les dades al DAO (Punt 2).

3. El DAO respon a la crida demanant les dades a la font de dades (Punt 2.1).

4. Per a cada fila que rep, el DAO crea un TransferObject (ValueObject) (Punt 2.2).

5. El DAO retorna al client el/els TransferObject (Punt 2.3).

6. A continuació, el client defineix un TransferObject mitjançant crides a setProperty. Per exemple, suposem que volem trobar pacients amb edat 34 anys i nom David. El BusinessObject defineix en l'objecte de la classe Pacient, l'edat i el nom que busca i després invoca al DAO passant al pacient com argument (Punts 3, 4 i 5).

7. A DAO, setData( ) es sol·licita (Punt 5.1 i 5.2) al TransferObject les dades per a realitzar l'accés a dades: DataSource.setData( ) (Punt 5.3)

3.6. DIAGRAMA DE CLASSES.

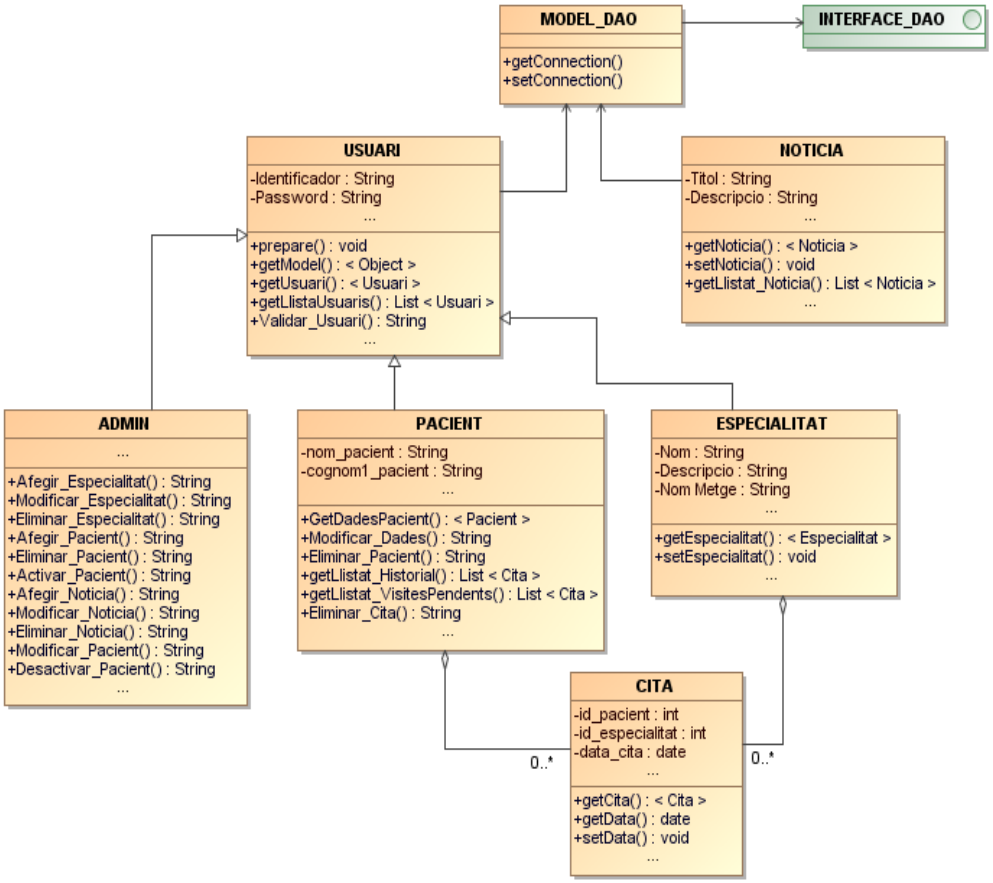


Diagrama de clases UML

La classe 'MODEL_DAO' es situa en la part superior de la jerarquia e implementa l'interface DAO, d'aquesta manera podrem implementar el mètode getConnection ( ) el qual serà utilitzat per les classes secundàries.

La **interface DAO** només requereix d'una sola escriptura del mètode getConnection ( ) per la connexió amb al SGBD, a nivell de codi serà:

```
import java.sql.Connection;

public interface DAO
{
        public Connection getConnection( );
}
```

La **classe MODEL_DAO** conté la implementació del mètode getConnection( ) a partir d'un pool de connexió (DataSource). Els pools de connexió (javax.sql.DataSource) permeten declarar connexions a partir del fitxer XML de l'aplicació (**web.xml**). La connexió amb la font de dades s'emmagatzema en el context de l'aplicació (ServletContext). Aquest mètode retornarà una connexió SQL que podrà ser utilitzada per qualsevol classe secundària del model.

```
public class MODEL_DAO implements DAO
{
        DataSource dataSource = null;

        public Connection getConnection( )
        {
                ServletContext servletContext =
                ServletActionContext.getServletContext();

                if (this.dataSource == null)
                {
                        dataSource = (DataSource)
                        servletContext.getAttribute("dataSource");
                }
                Connection connection = null;
                if (dataSource != null)
                {
                        try
                        {
                                connection = dataSource.getConnection( );
                        } catch (SQLException e)
                        {

                        }
                }
                return connection;
        }

        public void setConnection(DataSource dataSource)
        {
                this.dataSource = dataSource;
        }
}
```

3.7. INTERFÍCIE D'USUARI.

En aquest capítol mostrem la interfície gràfica resultant [9]. Com es pot comprovar, es tracta d'una traducció directa del prototipus proposat en la fase d'anàlisi del producte.
Es tracta d'una interfície gràfica senzilla, fàcil d'utilitzar i seguint els estàndards (capçalera, menú lateral, zona de continguts i peu de pàgina).

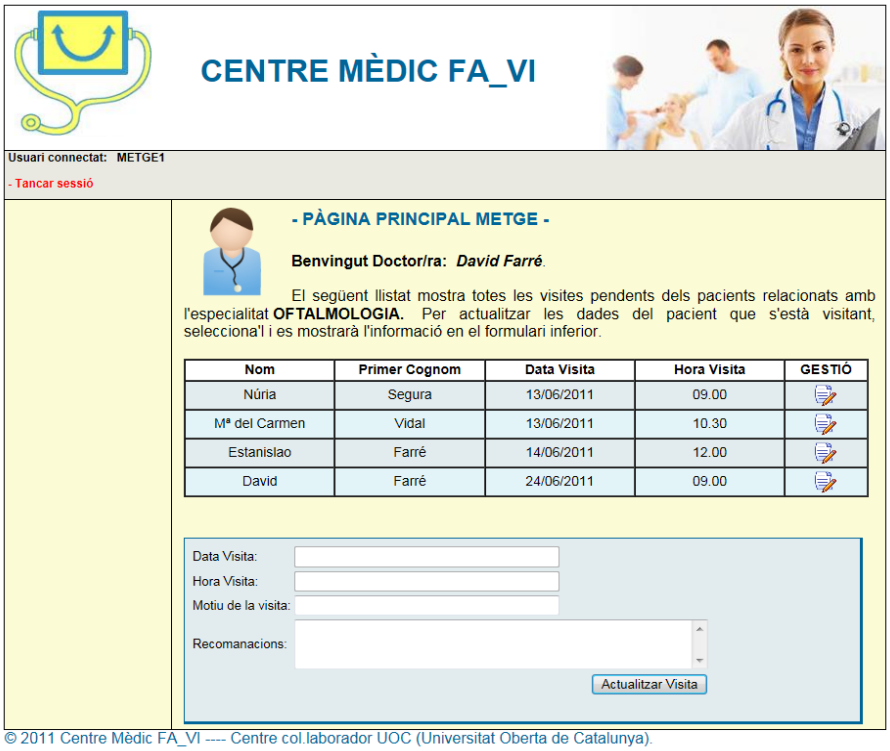
- *Pantalla de Benvinguda:*

- *Pantalla principal actor Pacient:*

[9] No mostrarem totes les pàgines resultants, només les que tradueixen de forma directa les proposades en el capítol anterior (Prototipus).

- Pantalla principal actor Administrador:

- Pantalla principal actor Metge:



3.8. VALIDACIO D'ENTRADES.

Struts 2 facilita un mètode de validació senzill basat en el framework de validació XWork. Els validadors de Struts 2 no requereixen programació, cada definició d'una regla es configura en relació amb una propietat o un objecte en un fitxer de text en format XML de fàcil manteniment.
Validarem les entrades d'un formulari mitjançant aquest *'fitxer de validacions'*, el nom del qual s'ha adaptar al model següent: ***NomClasseAccio-validation.xml*.**
L'etiqueta <field> conté un atribut *name*, la qual estableix el vincle amb el nom d'un camp del formulari HTML que ha de validar-se. L'etiqueta <field/> conté una o varies etiquetes <field-validator/> associades amb un atribut *type* que defineix el tipus de validació (camp obligatori, e-mail, número enter, ...)
Per últim, l'etiqueta <message/> permet descriure el missatge que es mostrarà en cas d'error.

Contingut del fitxer UsuariAccion-validation.xml:

```
<validators>
        <field name="nom_pacient">
                <field-validator type="requiredstring">
                        <param name="trim">true</param>
                        <message>Error, s'ha d'introduir nom de pacient.</message>
                </field-validator>
                <field-validator type="stringlength">
                        <param name="maxLength">20</param>
                        <param name="trim">true</param>
                        <message> Error, màxim 20 caràcters.</message>
                </field-validator>
        </field>
</validators>
```

En aquest cas, la validació *requiredstring* ens permet comprovar que el camp 'nom_pacient' no estigui buit.
La validació *stringlength* permet validar la longitud del camp 'nom_pacient'. Es pot especificar la longitud mínima i màxima.
Sense entrar en detall, podem comentar que existeixen altres validacions estàndards que inclou Struts 2 com són: required, int, date, expression, fieldexpression, e-mail,...

A més a més, s'haurà de definir la pàgina a la qual s'ha de cridar en cas d'error en la introducció de dades. Aquesta definició es realitza en el fitxer de configuració *struts.xml* mitjançant l'etiqueta <result name = "input"/>.

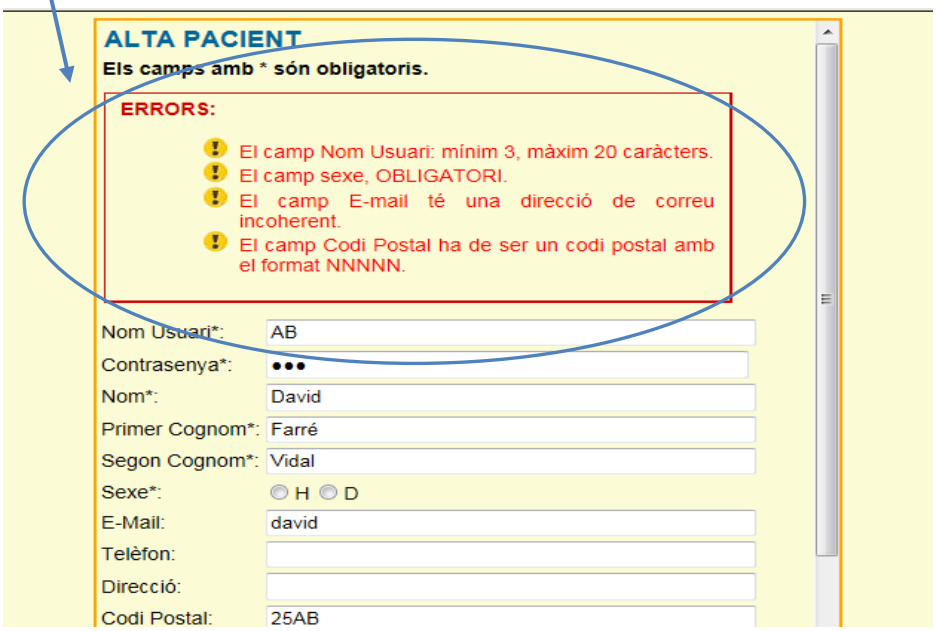
Definició de la pàgina a la que s'ha de cridar en cas d'error (al fitxer struts.xml):

```
<action name="AfegirPacient" class="com.UsuariAccion" method="Afegir_Pacient">
        <result name="input">jsp/AltaPacient.jsp</result>
        <result name="success" type="redirectAction">Llistat_Especialitats</result>
</action>
```

Finalment, per mostrar els missatges d'errors, ho farem amb l'etiqueta que ens proporciona Struts 2, ***<s:fielderror/>***

Exemple per mostrar els missatges d'errors dins de la vista (Alta_Pacient.jsp):

```
...
<s:fielderror/>
<s:form name="Add_Pacient" action = "AfegirPacient" method="POST">
<s:textfield name="usuari.identificador" label="%{getText('label.iduser')}" size = "50" required = "true"/>
<s:password name="usuari.contrasena" label="%{getText('label.password')}" size = "50" showPassword="true" required = "true"/>
<s:textfield name="usuari.nom_pacient" label="%{getText('label.nomuser')}" size="50" required = "true"/>
...
```



Struts 2 també permet validar les dades dins de l'**ACTION CLASS**, és a dir, podem crear una funció que ens validi si un nom d'usuari ja existeix en la Base de Dades.

Exemple:

```
...
usuari_auxiliar = model_usuari_dao.getUsuari(usuari.getIdentificador());

if (usuari_auxiliar != null)
{
        //L'identificador d'usuari ha de ser únic, per tant si un usuari vol donar-se d'alta
        //amb un identificador existent, ho indicarem per pantalla.
        addActionError("Nom d'usuari existent");
        Retorn = "ERROR";
}
```

Per afegir errors, podem utilitzar el mètode *addActionError(String error)* o bé el mètode *addFieldError(String fieldName, String errorMessage).*
Tant un mètode com l'altre formen part de la interface ValidationAware que ActionSupport implementa.

Com mostrar els missatges d'error si s'ha utilitzat el mètode addActionError ?
Ho farem amb l'etiqueta que ens proporciona Struts 2***, <s:actionerror/>*** (de la mateixa manera que hem vist en l'exemple anterior utilitzant l'etiqueta <s:fielderror/>).

Vist això, també podríem controlar les dades a validar d'un formulari utilitzant una funció de validació en l'ACTION CLASS:

```
public void validate ( )
{
        if (getIdentificador ( ).length( ) == 0)
        {
                addFieldError("identificador","Identificador obligatori");
        }
        ...
}
```

Struts 2 automàticament executarà el mètode validate( ). Si es detecta algun error, Struts 2 no cridarà al mètode execute( ) i retornarà "input" com a resultat.

Existeixen altres mètodes de validació,

- **Validacions utilitzant anotacions.**
  ```
  @RequiredStringValidator(message="Supply name")
  public String getUsername() {
          return username;
  }
  ```

- ***Validacions en el client***, escrivint codi que generi JavaScript per la validació en el client.

- ***Validació AJAX de formularis.***

Com podem veure, Struts 2 ens facilita diferents maneres de treballar per validar les dades. En aquest projecte s'ha optat per utilitzar la validació de camps automàtica de Struts 2 utilitzant el fitxer de configuració **–validation.xml**. També m'ajudo de les validacions dins del ACTION CLASS per fer comprovacions de validació respecte les dades que hi ha emmagatzemades en la Base de Dades.

3.9. INTERNACIONALITZACIÓ.

La gestió dels missatges d'error i confirmació és una tasca important durant el desenvolupament d'aplicacions informàtiques. La internacionalització consisteix en gestionar els idiomes i així facilitar pàgines multilingües. Per tal de modificar de manera dinàmica el idioma de l'aplicació, Struts 2 facilita el paràmetre **request_locale**.

Exemple:

```
<s:url id="url" action="Llistar_Presentacio">
        <s:param name="request_locale">cat</s:param>
 </s:url>
 <s:a href="%{url}">Català</s:a>

<s:url id="url" action="Llistar_Presentacio">
        <s:param name="request_locale">es</s:param>
</s:url>
<s:a href="%{url}">Castellano</s:a>
```

Com localitzarà Struts 2 els fitxers de missatges per cada idioma? Si volem configurar-ho en el fitxer Struts.xml, haurem de fer-ho utilitzant l'etiqueta <constant/> de la següent manera:

```
<constant value="Missatges" name="struts.custom.i18n.resources"/>
```

Només ens farà falta generar tants fitxers com idiomes vulguem contemplar en l'aplicació.

***Missatges_cat**.properties*
***Missatges_es**.properties*
...

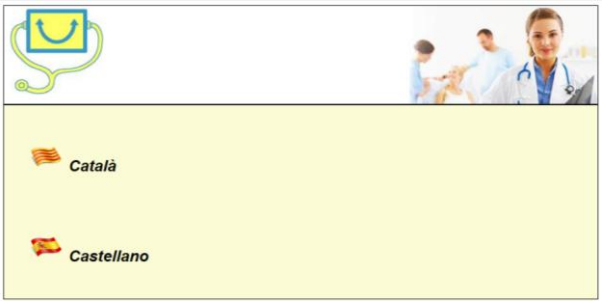
El contingut d'aquests fitxers estarà format per etiquetes amb el corresponent text associat.

Exemple:

label.especialitats = Especialitats mèdiques (En el fitxer Missatges_cat.properties)
label.especialitats = Especialidades médicas. (En el fitxer Missatges_es.properties)

Finalment, en les pàgines .JSP, .JSPF resultants, escriurem directament les etiquetes enlloc del text que s'hauria d'escriure per cada idioma.

```
<a href="Llistar_Especialitats"><s:text name="label.especialitats"/></a>
```



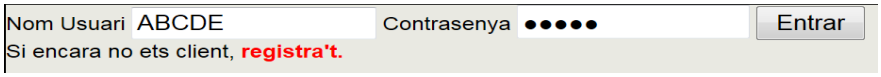
- Pantalla Inicial Aplicació –

3.10. PROVES REALITZADES.

Per finalitzar amb el procés d'implementació de l'aplicació es fan una sèrie de proves per comprovar el correcte funcionament de totes les funcionalitats.

- VALIDACIÓ LOGIN.

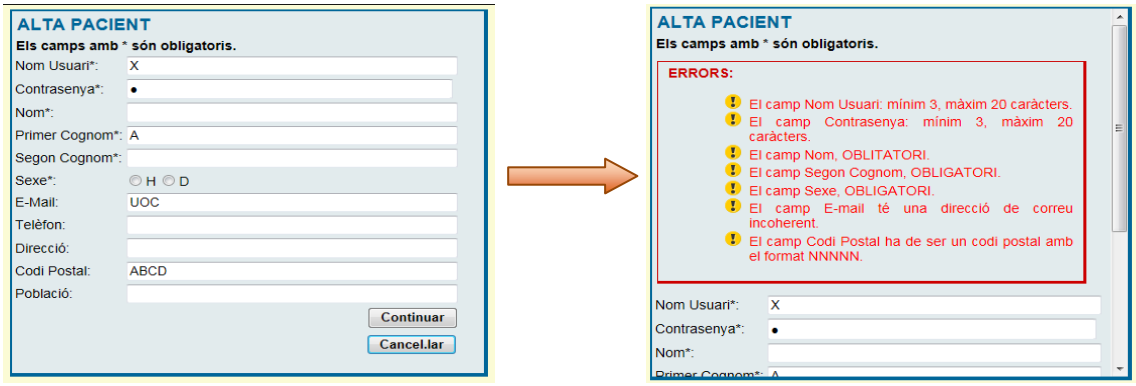
Usuari NO registrat *(o bé usuari registrat però entrant dades incoherents)* intenta accedir al sistema ficant una parella de valors *Nom Usuari, Contrasenya* inexistents a la Base de Dades.



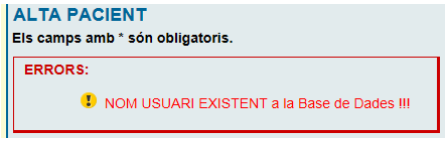
El sistema respon amb un missatge d'error i torna el control a l'usuari.

- REGISTRE D'USUARI.

Usuari NO registrat s'intenta registrar entrant dades incoherents. El sistema validarà les dades i retornarà el control a l'usuari indicant els errors existents.



El nom d'usuari és clau, per tant únic. Si un usuari es vol registrar fent servir un *NOM USUARI* existent a la Base de Dades, el sistema respondrà amb un missatge d'error i tornarà el control a l'usuari.

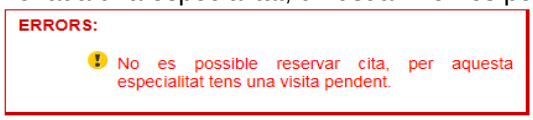


- RESERVAR CITA.

Donada una especialitat en concret en una data determinada, l'usuari tipus PACIENT intenta confirmar una cita.

Tot seguit es detallen els controls que s'han implementat.

1. Donada una especialitat, un usuari només pot tindre una visita pendent.



2. En una mateixa data (dia i hora) no pot reservar cita per dues especialitats diferents.

3. En una mateixa data (dia i hora) un altre usuari té reservada la mateixa especialitat.

Com sap l'usuari quines hores queden lliures en un dia en concret? Per facilitar la feina a l'usuari, he creat una consulta en la qual l'usuari podrà veure totes les hores 'reservades' en un dia en concret.

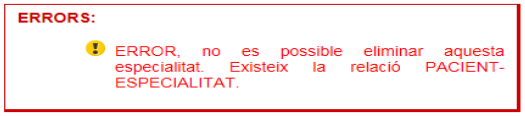
Altres proves realitzades,

Acció: Afegir Imatges (Actor ADMIN).

L'administrador intenta pujar una imatge de mida més gran que el màxim permès o bé en un format desconegut. Amb l'ajuda de l'interceptor *fileUpload* farem aquests controls.

Acció: Baixa Especialitat (Actor ADMIN).

Només es podrà donar de baixa una especialitat si no existeix cap mena de relació (històric o bé visites pendents) amb cap pacient.



Per no fer massa extens aquest apartat, menciono la resta de funcionalitats en les quals també he realitzat proves però no es detalla el resultat obtingut.

- Alta noticies (Actor ADMIN)
Introducció de dades coherents.

- Alta / Modificació especialitats (Actor ADMIN).
Introducció de dades coherents.
Controls respecte el nom de l'especialitat així com del nom d'usuari associat a l'especialitat (Recordem que ESPECIALITAT equival a Actor METGE per tant quan es dóna d'alta l'especialitat també s'ha de donar d'alta un nom d'usuari així com una contrasenya).

- Alta / Modificació dades pacients (Actor ADMIN).
Mateixos controls que quan es registra un usuari tipus PACIENT.

- Actualització dades PACIENT (Actor METGE).
Introducció de dades coherents.

3.11. EINES UTILITZADES EN EL DESENVOLUPAMENT DEL TREBALL.

Per la implementació del TFC m'he ajudat de les següents eines:

- Com a IDE (Entorn Integrat de Desenvolupament) he utilitzat **NetBeans 6.9.1.**
- Com a servidor web: **TOMCAT 6.0.**
- Navegador habitual: **INTERNET EXPLORER 9.**
- Per la capa de presentació, he utilitzat pàgines **JSP** *(utilitzant llenguatge HTML, fulles d'estil CSS i la biblioteca d'etiquetes que inclou el framework Struts 2).*
- La comunicació amb la capa de negoci ha estat mitjançant el framework **STRUTS 2.**
- Com a gestor de Bases de Dades, he utilitzat **MySQL** (Motor de bases de dades gratuït i extensament utilitzat).
- Per vincular l'aplicació amb la BBDD, he treballat utilitzant el model **Data Access Object DAO.**
- Per generar la documentació (memòria + presentació),
  - Editor de text: **Microsoft Word 2007.**
  - Planificació de les tasques: **GanttProject 2.0.10.**
  - Diagrames: **MagicDraw UML Personal Edition 16.0.**
  - Disseny Bases de Dades: **MySQL Workbench 5.2.33.**
  - Presentació: **Microsoft PowerPoint 2007.**

4. CONCLUSIONS.

L'elecció de realitzar el TFC relacionat amb l'arquitectura J2EE va ser motivada per l'interès personal en el desenvolupament d'aplicacions web. En termes generals, el resultat de la realització d'aquest projecte ha estat satisfactori i sobretot molt enriquidor.

- Satisfactori en l'aspecte que he pogut desenvolupar més o menys totes les funcionalitats previstes en la PAC 2.

- Enriquidor en l'aspecte dels coneixements adquirits. Personalment i tal com he dit anteriorment, tenia nocions més aviat nul·les de totes les tecnologies que es veuen involucrades en el desenvolupament d'un projecte d'aquest tipus (només tenia nocions mínimes del llenguatge JAVA). He dedicat moltes hores a aprendre l'arquitectura J2EE, el framework Struts 2, disseny gràfic de l'aplicació,... però crec que tot aquest temps ha estat ben invertit.

Que canviaria si hagués de tornar a començar?

Si hagués de tornar a començar hauria deixat per una última fase la part visual de l'aplicació. Adquirir coneixements d'HTML, fulles d'estil CSS (menús, taules, llistats,...) així com la incorporació de tot plegat en les pàgines .JSP m'ha ocupat un temps considerable. Això m'ha implicat restar temps en l'estudi d'altres aspectes més interessants, com per exemple, incorporació del framework *Hibernate* per la persistència, utilització de la tecnologia AJAX... [10]

Per finalitzar, volia comentar que l'estil d'avaluació continuada del TFC que ofereix la UOC m'ha ajudat molt per no caure en el desànim i abandonar el projecte. En un primer moment, dóna la sensació que és impossible realitzar tots els objectius previstos però marcant unes dates d'entrega per les diferents parts que formen el projecte *(pla de treball, anàlisi i disseny, implementació i realització de la memòria)* he anat evolucionant en la realització del mateix.

[10] He cregut oportú dedicar un apartat on explico les funcionalitats i complements estudiats però NO implementats en l'aplicació resultant (Veure apartat 7.3.)

5. GLOSSARI.

J2EE (Java 2 Enterprise Edition).

Defineix un estàndard per al desenvolupament d'aplicacions empresarials *multicapa* dissenyat per Sun Microsystems. Simplifica les aplicacions empresarials basant-les en components modulars i estandarditzats, proveint un complet conjunt de serveis a aquests components i manejant moltes de les funcions de l'aplicació de forma automàtica, sense necessitat de una programació complexa.

MVC (Model View Controller).

És un patró de disseny que ens ajuda a donar-li certa estructura lògica a les aplicacions. El seu objectiu es separar la lògica de negoci de la lògica de presentació. El *MODEL* està format pels components que controlen les dades que utilitza l'aplicació, la *VISTA* està formada pels components que presenten les dades i el *CONTROLADOR* està format pels components responsables de la gestió d'events i de coordinar les activitats del model i de la vista.

CSS (Cascade Style Sheets)

És un mecanisme simple que descriu com es mostrarà un document en la pantalla, o com s'imprimirà o inclús com s'haurà de pronunciar la informació present en un document a través d'un dispositiu de lectura. S'utilitza per donar estil als documents HTML i XML separant el contingut de la presentació. Permet als desenvolupadors Web controlar l'estil i el format de múltiples pàgines Web al mateix temps.

SERVIDOR D'APLICACIONS.

S'anomena servidor d'aplicacions a un servidor en una xarxa de computadors que executa certes aplicacions. Normalment es tracta d'un dispositiu de software que proporciona serveis d'aplicació a les computadores client. Un servidor d'aplicacions generalment, gestiona la majoria de les funcions de lògica de negoci i d'accés a les dades de l'aplicació. Degut a l'èxit del llenguatge de programació Java, el concepte *servidor d'aplicacions* normalment fa referència a un servidor d'aplicacions Java EE (Podem mencionar JBoss o bé GlassFish com a servidors 'gratuits').

APACHE TOMCAT.

Funciona com un contenidor de *servlets.* Implementa les especificacions dels *servlets* i de *JavaServer Pages*. Es un servidor web, NO és un servidor d'aplicacions. Inclou el compilador Jasper, que compila JSPs convertint-les en servlets. Tomcat pot funcionar com un servidor web per si mateix i funciona en qualsevol sistema operatiu que disposi de la màquina virtual Java.

SERVLET.

Programa escrit en JAVA que s'executa en el contenidor Web d'un servidor d'aplicacions. Un servlet accepta peticions d'un client, processa la informació relativa a la petició realitzada i li retorna els resultats que podran ser mostrats mitjançant applets, pàgines HTML, ...
També pot realitzar altres tasques com comunicar-se amb un altre *servlet* per ajudar-lo en el seu treball o bé facilitar l'accés a Bases de Dades.

JSP (Java Server Pages).

Tecnologia orientada a crear pàgines Web amb programació JAVA. Les pàgines JSP estan composades de codi HTML combinat amb etiquetes especials (ordres) i amb codi escrit en Java (scriptlets – seqüències d'ordres). El motor de les pàgines JSP està basat en els *servlets*

de Java. El funcionament general de la tecnologia JSP es que el Servidor d'Aplicacions interpreta el codi obtingut en la pàgina JSP per construir el codi Java del servlet a generar. Aquest servlet serà el que generi el document (normalment HTML) que es presentarà en la pantalla del navegador de l'usuari.

FRAMEWORK.

Un framework és un conjunt de biblioteques, eines i normes a seguir que ajuden a desenvolupar aplicacions. Està composat per segments/components que interactuen els uns amb els altres. Els objectius principals són: accelerar el procés de desenvolupament, reutilització del codi ja existent i promoure 'bones pràctiques' de desenvolupament com l'ús de patrons.
Entre els més destacats utilitzats en Java: Apache Struts, WebWork, JSF, Spring,...

JAVASCRIPT.

Llenguatge de programació interpretat (no requereix compilació) orientat a les pàgines web amb una sintaxis semblant a la del llenguatge Java. S'utilitza en pàgines web HTML, per realitzar tasques i operacions en el marc de l'aplicació client (validació de formularis, canviar el punter del mouse, creació de barres de scroll, obrir popups,...)

AJAX (Asynchronous JavaScript And XML).

És una tècnica de desenvolupament web basada en el llenguatge JavaScript que permet fer consultes HTTP sense necessitat de tornar a carregar les pàgines, això augmenta la interactivitat, velocitat i usabilitat en les aplicacions.

6. WEBGRAFIA / BIBLIOGRAFIA CONSULTADA.

6.1. WEGRAFIA CONSULTADA:

- *Introducció a JAVA / J2EE.*

  http://www.illasaron.com/ (VideoTutorials realitzats pel professor Jesús Conde).

- *Informació en general del framework Struts 2:*

  http://struts.apache.org/2.x/

  http://es.scribd.com/doc/6212696/Struts-2-Basics (diapositives)

  http://es.scribd.com/doc/54049030/Manual-Struts2 (manual)

  http://www.epidataconsulting.com/tikiwiki/tiki-print.php?page=Struts2

  http://www.kamlov.site90.net/?p=613

  http://www.lifia.info.unlp.edu.ar/es/difusion/20080103.htm

- *Interceptors:*

  http://struts.apache.org/2.0.11/docs/interceptors.html

  http://www.vitarara.org/cms/struts_2_cookbook/creating_a_login_interceptor

  http://ungranoparajava.blogspot.com/2009/12/interceptors-en-struts2.html

  http://es.debugmodeon.com/articulo/autentificacion-web-usando-interceptor-de-struts-2

- *Validació:*

  http://struts.apache.org/2.x/docs/basic-validation.html

  https://docs.google.com/viewer?a=v&pid=explorer&chrome=true&srcid=0B3mCI2MxFIlxZmM0N2E3MmMtZDg4Ny00YTM5LTg2NjktYWQ2ZTlmOWYxMjUz&hl=es

  http://www.easywayserver.com/blog/struts2-validation-example-struts-2-validation/

  http://www.vaannila.com/struts-2/struts-2-example/struts-2-validation-example-1.html

  http://www.roseindia.net/struts/struts2/struts-2-client-side-validation-example.shtml

- *Internacionalització:*

  http://cemicursoj2ee.googlecode.com/svn/trunk/Struts2ExampleMultilanguage/

- *Complement DatePicker (jQuery):*

  http://code.google.com/p/struts2-jquery/

- *Complement Tiles:*

  https://docs.google.com/viewer?a=v&pid=explorer&chrome=true&srcid=0B3mCI2MxFIlxMDJjNjU4NTktMzJhMS00N2ZjLWEyMzctMzliNWU4YTczZjVi&hl=es

6.2. BIBLIOGRAFIA CONSULTADA:

- Struts 2 – El framework de desarrollo de aplicaciones Java EE. Autor: Jérome LAFOSSE.
- JAVA 2 – Interfaces gráficas y aplicaciones para Internet. Autor: Fco. Javier Ceballos.
- STRUTS 2 in Action (versió .pdf). Autors: Donald Brown, Chad Michael Davis, Scott Stanlick.

7. ANNEXES.

7.1. ANNEX 1. COMPARATIVA ENTRE STRUTS 1 & STRUTS 2.

En aquesta secció farem una comparativa entre les dues versions existents d'aquest framework. Podem afirmar que Struts 2 és molt més simple i eficient que Struts 1. No solament explota millor les potencialitats de J2EE si no que també fa més fàcil el desenvolupament de noves aplicacions i la integració amb altres tecnologies.

DEPENDÈNCIA DEL SERVLET.
Struts 1. Hi ha una gran dependència amb el API Servlet, cosa que es fica de manifest en què els paràmetres del mètode execute() necessiten els objectes *HttpServletRequest* i *HttpServletResponse.* **Struts 2.** Les Actions estan desacoblades del contenidor, sent el *Servlet Controller* el responsable de realitzar els *set* de paràmetres, això es tradueix en una facilitat de testeig de les Actions.
CLASSE ACTION.
Struts 1. Les classes Action han d'estendre d'una classe base abstracta i no permet interfaces. **Struts 2.** No es necessari fer aquest *extends* d'una classe base. S'utilitzen interfaces o la possibilitat d'estendre de la classe *ActionSupport* la qual implementa una sèrie de interfaces que ens proporcionaran totes les eines necessàries per la correcta execució de les Action.
VALIDACIONS.
Struts 1. Permet validació manual mitjançant el mètode validate el qual es troba en la classe ActionForm o bé fer validacions via l'extensió del *validator.* **Struts 2.** Permet validació manual mitjançant el mètode validate que es troba en el Action o bé pot realitzar validacions utilitzant el framework *XWORK*. Aquesta tecnologia permet fer validacions encadenades utilitzant regles definides per l'usuari.
RECUPERACIÓ DE DADES D'ENTRADA.
Struts 1. Utilitza classes que estenen de la classe genèrica ActionForm. Això fa que un JavaBean no pugui ser utilitzat en Struts 1, amb la qual cosa s'ha de replicar el codi del Bean en el ActionForm corresponent. No compleix el principi DRY (Don't Repeat Yourself) **Struts 2.** Utilitza les propietats set() i get() per recuperar la informació. Si tenim un JavaBean, només cal ubicar-lo en el Action amb els seus corresponents set() i get() per poder-lo utilitzar.
LLENGUTATGE D'EXPRESSIONS.
Struts 1. Struts està integrat amb JSTL amb la possibilitat d'utilitzar el llenguatge d'expressions JSTL-EL el qual no és molt potent quan treballem amb col·leccions i propietats indexades. **Struts 2.** No només inclou JSTL sinó també un conjunt de llibreries que permeten una integració efectiva amb les propietats dels actions. A més, les etiquetes permeten múltiples

configuracions utilitzant els themes. Suporta un llenguatge d'expressions més flexible anomenat OGNL (Object Graph Navigation Language).
CONVERSIÓ DE TIPUS.
Struts 1. Totes les propietats de Struts són Strings, utilitzant el *Commons-BeanUtils* per conversions. **Struts 2.** Utilitza OGNL per les conversions. A més, inclou una sèrie de *converters* per a tipus bàsics i comuns els quals es poden configurar mitjançant *Annotations* o bé mitjançant *converters* personalitzats.
VALORS EN LA VISTA.
Struts 1. Per treballar amb les vistes, Struts 1 utilitza un mecanisme estàndard de JSP per vincular objectes a la propietat en el context de pàgina. Si les JSP requereixen accedir a altres contexts s'haurà de programar. **Struts 2.** Utilitza la tecnologia *ValueStack* que permet recuperar els valors amb independència del context (sessió, aplicació, request,...) en el qual estiguin guardats els valors.
CONTROL DE L'EXECUCIÓ DE LES ACTION.
Struts 1. Totes les ACTIONS dins d'un mòdul tenen el mateix cicle de vida. **Struts 2.** Suporta diferents cicles de vida per ACTION. Aquests, estan basats en les piles d'interceptors. Aquestes piles d'interceptors es poden personalitzar, per tant, es poden crear cicles de vida personalitzats per ACTION.

7.2. ANNEX 2. INTERCEPTORS.

Tal com ja hem mencionat anteriorment, els interceptors són filtres que permetran simplificar i millorar el treball dels desenvolupadors. En la següent taula, podem veure els interceptors per defecte que ofereix Struts 2.

INTERCEPTOR	DESCRIPCIÓ
alias	Gestió dels noms de paràmetres en les consultes HTTP.
chain	Gestió de l'encadenament d'accions i de la transmissió de paràmetres.
checkbox	Gestió de les caselles de verificació en els formularis, afegint un valor false per defecte.
createSession	Gestió de la sessió de l'usuari actual, es crea automàticament una sessió HttpSession.
conversionError	Gestió dels errors de conversió en les accions.
debugging	Gestió de la depuració de les aplicacions.
execAndWait	Gestió de la càrrega de pàgines.
exception	Gestió del processament d'excepcions.
fileUpload	Gestió de la pujada de fitxers.

i18n	Proporciona internacionalització.
logger	Es mostra les accions que s'estan executant.
modelDriven	Gestió dels models de persistència. Ubica l'objecte del model dins de la Value Stack per les accions que implementen a la interface modelDriven.
params	Gestió del mapping entre els paràmetres de les consultes i les propietats de les accions.
prepare	Invoca al mètode prepare( ) de les accions que implementen a la interface Prepare.
profiling	Gestió de la creació de perfils de les accions.
roles	Permet que l'action s'executi si l'usuari forma part dels rols configurats.
servletConfig	Gestió de l'accés a les classes HttpServletRequest i HttpServletResponse.
scope	Gestió del mecanisme de les sessions.
scopedModelDriven	Gestió dels models de persistència amb gestió de sessió.
staticParams	Gestió dels paràmetres estàtics declarats en les definicions i classes d'acció.
timer	Gestió del temps d'execució de les accions.
token	Gestió del 'doble submit'. Ens assegura que no es produeix un doble click en un submit d'un formulari.
tokenSession	Igual que l'anterior amb un paràmetre de sessió.
validation	Gestió de les validacions dels formularis.
workflow	Gestió de la crida dels mètodes de validació en les classes d'acció.

Struts2 disposa de piles d'interceptors. Existeixen 'piles pre-configurades' utilitzables per les Action. L'ordre en què es crida a cada interceptor és important, una pila d'interceptors personalitzats amb un ordre inapropiat pot produir resultats incoherents.

PILA INTERCEPTORS	INTERCEPTORS INCLOSOS
basicStack	exception, servletConfig, prepare, checkbox, params, conversionError
chainStack	chain, basicStack
defaultStack	exception, alias, servletConfig, prepare, i18n, chain, debugging, profiling, scopedModelDriven, modelDriven, fileUpload, checkbox, staticParams, params, conversionError, validation, workflow
i18nStack	i18n, basicStack
executeAndWaitStack	execAndWait, defaultStack
fileUploadStack	fileUpload, basicStack
modelDrivenStack	modelDriven, basicStack
paramPrepareParamsStack	exception, alias, params, servletConfig, prepare, i18n, chain, modelDriven, fileUpload, checkbox, staticParams, params, conversionError, validation, workflow
validationWorkflowStack	basicStack, validation, workflow

7.3. FUNCIONALITATS / COMPLEMENTS.

En aquest apartat vull mencionar feina realitzada durant l'elaboració del projecte però que per manca de temps o bé per dificultats en la seva implementació no he pogut desenvolupar.

7.3.1. STRUTS i AJAX JQUERY.

Des de fa uns anys, s'estan desenvolupant nombrosos frameworks especialitzats en JavaScript que ens ofereixen un conjunt de serveis anomenats Web 2.0 (conjunt de tecnologies i eines basats en l'ergonomia de les interfícies gràfiques). Podem anomenar: *Prototype, ExtJs o bé JQuery.* La més utilitzada i recomanada pels dissenyadors de Struts 2 es *JQuery.*
Aquesta llibreria permet manipular documents HTML i XHTML. A més, ofereix un sistema senzill d'accés a les etiquetes dels documents mitjançant notació de punts. Aquest framework proposa cents de complements per gestionar la visualització, els formularis, les validacions, ...

La configuració es senzilla:

1. Descàrrega de la llibreria JavaScript i de manera opcional dels complements necessaris.
2. Instal·lació dels fitxers *.js* descarregats en un directori de l'aplicació.
3. Declaració de les inclusions de llibreries en les pàgines JSP que l'utilitzaran.

Un d'aquests complements JQuery s'anomena JQueryUI. Aquest complement permet gestionar calendaris, taules,...
Per tal de fer una aplicació amb un disseny modern i més funcional volia utilitzar aquesta llibreria per incorporar:

- Quadres de diàleg dinàmics col·locats a primer pla.
- Taules.
- Calendari.

Degut als problemes amb els quals m'he trobat, només m'ha donat temps d'utilitzar el complement *DatePicker* el qual permetrà als pacients, donada una especialitat en concret, seleccionar una data per així reservar una cita.

Exemple utilització del component DATEPICKER:

- *He tingut que descarregar la llibreria struts2-jquery-plugin-2.5.3.jar*
  Com en qualsevol llibreria de tags de Struts 2, s'ha de declarar el seu ús en la capçalera.
  <%@taglib prefix="sj" uri="/struts-jquery-tags"%>

- *S'ha d'incloure un tag en la capçalera de la pàgina que permet fer el include de les CSS i dels arxius de Javascript de jQuery i jQuery UI.*

```
<head>
        <sj:head jqueryui="true" jquerytheme="cupertino"/>
</head>

<body>
...
<sj:datepicker value="tomorrow" id="data_citaprevia" name="cita.data_citaprevia"
key="label.EscollirData" labelposition="top" displayFormat="dd/mm/yy"
```

```
changeMonth="true"
 changeYear="true"
 onChangeMonthYearTopics="onDpChangeMonthAndYear"
showAnim="fadeIn" showOptions="{direction: 'up'}" duration="slow"
minDate="0" buttonImage="images/calendar.png"/>
...
</body>
```

7.3.2. COMPLEMENT TILES.

Tiles permet gestionar l'aparença i les divisions de les aplicacions Web. Proporciona una biblioteca d'etiquetes que permet definir la presentació per a totes les pàgines d'usuari (un canvi en la definició de les pàgines es converteix en una modificació instantània de totes les pàgines d'usuari).

Aquest complement TILES es basa en dos components: la plantilla i la definició.

- La plantilla es defineix a partir d'una pàgina JSP.

Exemple:

/jsp/PlantillaPresentacio.jsp

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<html>
<head>
<title>
        <tiles:getAsString name = "Titol_Pagina"/>
</title>
<body>
        <tiles:insertAttribute name = "capçalera"/>
        <tiles:insertAttribute name = "menu_lateral"/>
        <tiles:insertAttribute name = "contingut"/>
        <tiles:insertAttribute name = "peu_pagina"/>
</body>
```

- La definició de plantilla s'utilitza entre la pàgina de presentació de la plantilla i les pàgines .JSP

S'estableix en un fitxer *tiles.xml* ubicat en el directori /WEB-INF

```
...
<tiles-definitions>
        <definition template="/jsp/PlantillaPresentacio.jsp" name="Plantilla_Principal">
                <put-attribute name="capçalera" value="/jspf/Capçalera.jspf"/>
                <put-attribute name="menu_lateral" value="/jspf/Menu_Lateral.jspf"/>
                <put-attribute name="contingut" value="/jsp/Presentacio.jsp"/>
                <put-attribute name="peu_pagina" value="/jspf/PeuPagina.jspf"/>
        </definition>
```

La resta de pàgines es reemplaçarien de la següent forma (només cal actualitzar les zones de la 'plantilla' que seran diferents, la resta són omplertes per la definició principal).

```
        <definition name="Llistat_Especialitats" extends="Plantilla_Principal">
                <put-attribute name="contingut" value="/jsp/Presentacio_Especialitats.jsp"/>
        </definition>

        <definition name="Llistat_Noticies" extends="Plantilla_Principal">
                <put-attribute name="contingut" value="/jsp/Presentacio_Noticies.jsp"/>
        </definition>
...
</tiles-definitions>
```

PASSOS NECESSARIS PER APLICAR EL COMPLEMENT TILES.

- Importar les llibreries necessàries:

- *commons-beanutils-1.X.jar*
- *commons-digester-1.X.jar*
- *commons-collections-3.X.jar*
- *tiles-core-2.X.jar*
- *tiles-api-2.X.jar*
- *tiles-jsp-2.X.jar*
- *struts2-tiles-plugin-2.1.X.jar*

- Definició del *listener* associat a Tiles en el deployment descriptor (fitxer web.xml):

```
<listener>
        <listener-class>
                org.apache.struts2.tiles.StrutsTilesListener
        </listener-class>
</listener>
```

- Definició de la ruta del fitxer de definicions de Tiles (fitxer web.xml):

```
<context-param>
        <param-name>
        org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG
        </param-name>
        <param-value>
                /WEB-INF/tiles.xml
        </param-value>
</context-param>
```

- Definició dels resultats de tipus *tiles* en el fitxer de configuració struts.xml i utilització dels resultats del tipus *tiles* en els resultats de les accions de Struts 2.

```
<result-types>
        <result-type name = "tiles" class = "org.apache.struts2.views.tiles.TilesResult"/>
</result-types>

<action name = "Llistar_Noticies" class = …
        <result name = "success" type = "tiles"> Llistat_Noticies </result>
</action>
```

Apunta al nom lògic de la vista, definit al fitxer tiles.xml

D'aquesta manera, si s'ha de modificar la plantilla per algun motiu, només s'ha de fer en un lloc (*PlantillaPresentacio.jsp)* i aquest canvi es veurà reflexat en totes les pàgines que utilitzin aquesta plantilla.

Per què no he incorporat aquest complement en l'aplicació que he desenvolupat?
Una vegada fet tot l'estudi, he provat d'incorporar-ho però per algun motiu 'X', algunes etiquetes com <s:submit>, <s:property>, ... no s'incorporaven correctament a la vista resultant i donat que el temps d'entrega del projecte s'aproximava, vaig decidir documentar-ho però no implementar-ho.

7.3.3. PROCÉS D'AUTENTICACIÓ DELS USUARIS.

Per tal d'assegurar l'accés als diferents espais de l'aplicació que requereixen de control d'usuari per accedir-hi, m'he ajudat dels interceptors per controlar-ho. [11]

Ara bé, fent proves he comprovat que l'aplicació té punts febles en aquest procés d'autenticació:

- Utilització de les 'fletxes' del navegador per anar enrere i endavant.

Cas estudiat:
Un usuari 'X' executa l'aplicació, fa les operacions corresponents i per finalitzar tanca sessió però NO tanca el navegador. Si un nou usuari selecciona la fletxa del navegador 'enrere' entrarà a l'espai de l'usuari 'X'.

- Execució de l'aplicació per 'n' usuaris en paral·lel.

Cas estudiat:
Un usuari 'X' executa l'aplicació i es valida. Sense tancar l'aplicatiu, un altre usuari 'Y' executa una nova instància i també es valida. Si torno al navegador on tinc carregat l'usuari 'X' i faig qualsevol acció es modifica l'usuari carregat.

[11] Detallat en l'apartat 3.3 (secció Interceptors).